Ops School Curriculum Documentation

Release 0.1

Avleen Vig

January 19, 2018

Welcome to the Ops School curriculum documentation site.

Ops School is a comprehensive program that will help you learn to be an operations engineer. Operations engineers are highly skilled people who manage the computer systems of businesses big and small. In addition to corporate systems, operations engineers also maintain the systems that allow websites, networks, payments systems and other Internet services to function. The field of operations engineering covers a wide variety of topics, from systems administration, to security, networking and beyond. Ops School will guide you through all of these skill sets from beginner to expert.

Since the early 90's, operations engineers have been in high demand. As a result, these positions often offer high salaries and long term job security. The SAGE/LISA Salary Survey has charted the average salaries for systems administrators and operations engineers since 1999 and has consistently shown the field to be prosperous for those people who enjoy diving into the inner workings of computer systems.

If you already know about the profession and want to know how to start, read *How to become an operations engineer*.

If you are reading about this career for the first time and want to know if it is for you, check out the *Careers in Operations* chapter.

CHAPTER 1

Table of Contents

Introduction

Todo

Explain "What is Ops School?"

Goals and Focus

To have a complete syllabus for training smart persons with high potential who already have a basic understanding of Linux or other UNIX variant, to become strong junior systems administrators. This course will focus on the Linux platform as a basis for competency in any UNIX ecosystem.

Usenix defines the sysadmin role as:

> Familiarity with an operating system and its commands/utilities at a user level; ability to edit files, use basic utilities and commands, find users' home directories, navigate through the file system, install software on workstations, and use I/O redirection; some understanding of how user authentication happens in a directory service context. Ability to identify/locate shared resources and perform simple tasks (e.g., manipulate jobs in a print queue, figure out why a network file system isn't available).
>
> —Usenix, *Job Descriptions for System Administrators, 3d Edition Revised*

Our goal is to teach these topics, but also to provide a good understanding of how the components work, which enables the students to continue to grow their skills and contributions.

Our focus is any person who has an interest in operations. There are often personality traits which allow people to understand the role of a systems administrator more easily, but the material should be easily accessible to anybody who wishes to learn about it.

The required minimum skill level before participating will be as follows:

- Has installed Linux or other UNIX variant
- Is comfortable with, or even prefers, working on the command line
- Has a basic understanding on navigating the filesystem, and the most basic tools (ps, ls, top, etc).

Syllabus layout

As we create this syllabus, there will inevitably be items which are found to be too advanced for a first pass but should be kept and taught as a progression into an intermediate systems administrator.

- Junior topics are labelled “101”.
- Intermediate topics are labelled “201”.
- Senior topics are labelled “301”.

Contributions

Ops School is a community driven effort, and we always need people to contribute. Currently we need people who are able to fill in areas in our documentation - whether it’s a little or a lot, everything helps.

How we’ll organize work

This is a living document. We write headings and their breakdowns as bullet points. We turn each bullet point into the material we want people to read. That’s right. The syllabus IS the course. For now, at least, until we find a better way.

You should start writing the actual material to be taught right into this syllabus. We’ll worry about formatting and things later. As you write, remember that “101” material is aimed at people who are working up to being junior sysadmins. They’re mostly blank slates.

How to contribute

You can find the documentation source on GitHub and send pull requests:

https://github.com/opsschool/curriculum

Please fork the repo to your own account, create a branch, make your changes there, and issue a pull request from your fork/branch to `opsschool:master`. Be descriptive in your commit message for anything non-obvious.

If all of this git and GitHub stuff is a little overwhelming, take a look at GitHub’s documentation. If you have still have questions after reading that, please feel free to join #opsschool on irc.freenode.net and ask your questions there. We’ll be glad to help you get your patches merged–but please remember that many of us are busy ops professionals, so we may not respond immediately.

If you’d like to join the mailing list, email avleen@gmail.com.

Rewards for contributions

We have a special reward for reaching either of the following goals with your contributions:

- 10 pull requests totaling 50 or more lines changed
- 1 recorded Ops School video

For completing either of these, you will receive a dozen freshly baked cookies, made fresh for you by a professional baker, and a 3-inch die-cut sticker of the Ops School logo.

Once you reach either of these, please fill in the OpsSchool rewards form and we will get your reward over to you!

Ops School Videos

In collaboration with O’Reilly Media, we are filming the Ops School curriculum to provide another method of learning for students. Filming happens approximately every 3 months in New York or Silicon Valley.

O'Reilly is publishing the videos online, in individual form and in package form. Any profits which would be made by Ops School are donated to non-profit organisations which aid in learning for low-income persons.

Video can be short (10-20 mins), or longer. Depending on the content in the video, some presenters choose to use slides, while others prefer screen-casting live demonstrations. Some videos may not need any supporting material at all!

The next scheduled filming is at the Santa Clara Hyatt, from Monday June 17th 2013 to Friday June 21st 2013. Filming slots are available all day.

The topics which have already been filmed are:

- Boot Process 101
- DNS 101
- Filesystems 101
- MySQL 101
- Nagios 101
- Networking 101
- Outage Investigation 101
- Package Management 101
- Shells 101
- SSH 101
- Application Level Monitoring
- Careers In Operations
- Operable Service - What People Expect From an Operable Service
- Productivity Tools and Techniques
- The `/proc` Filesystem
- Web Flow - The Life of a Web Request

If you are interested in filming, please contact Avleen Vig (avleen#gmail.com) or open an issue on the Github project.

How to write sections

In order to help students learn as much as possible, we are taking the following approach to the curriculum (this isn't a hard-and-fast rule, but an encouraged guideline wherever possible):

- Approach your writing in three steps:
    1. Tell students what they're going to learn
    2. Teach them what they need to know
    3. Tell them what they have learnt
- As much as possible, treat this as an interactive exercise. For example if you are writing about virtual machines, don't just write about virtual machines. Have the student create a virtual machine, and then explain what just happened. Don't tell students that package managers install packages, have them install a few packages and then explain how the package manager did its thing.

Please read the *Conventions* topic for some style guideline and preferences.

Overwriting existing content

There are times when you will want to update or replace sections of text written by others. When doing so, follow these guidelines to ensure your changes are integrated smoothly:

- Submit your pull request
- Reference the original author in your commit

We'll wait a few days (up to one week) to let the original author comment, if we feel there may be anything contentious in the commit. For most simple and straightforward commits, we'll simply accept the commit.

Credits

If you contribute to this document and would like your name in lights (or, at least, written down somewhere) please add it here along with an email address and any affiliation:

Name	Company
Avleen Vig <avleen@etsy.com>	Etsy, Inc
Patrick McDonnell	
Michael Rembetsy <mcr@etsy.com>	Etsy, Inc
Magnus Hedemark <magnus@yonderway.com>	Wireless Generation
Ariel Jolo <ajolo@sysarmy.com.ar>	sysARmy
Ryan Frantz <ryanleefrantz@gmail.com>	Crabby Admins (www.crabbyadmins.org)
Mike Fiedler <miketheman@gmail.com>	Datadog
Nathan Milford <nathan@milford.io>	Outbrain, Inc.
Patrick Cable <pc@pcable.net>	
Benjamin Krueger <benjamin@seattlefenix.net>	Sourcefire, Inc
Mårten Gustafson <marten.gustafson@gmail.com>	
Phil Hollenback <philiph@pobox.com>	
Adam Jacob <adam@opscode.com>	Opscode, Inc.
Mark Imbriaco <mark@github.com>	GitHub
James Turnbull <james@lovedthanlost.net>	Puppet Labs
Scott Nyce <snyce@codetwit.com>	TiVo, Inc.
Christian Paredes <cp@redbluemagenta.com>	Amazon
Jan Schaumann <jschauma@netmeister.org>	
Stephen Balukoff <sbalukoff@bluebox.net>	Blue Box Group
Evan Pettrey <jepettrey@gmail.com>	LOPSA
Khalid Maqsudi <khalid7621@hotmail.com>	Ashford.com
Paul Graydon <paul@paulgraydon.co.uk>	
Harold "Waldo" Grunenwald	
Martin Gehrke <martin@teamgehrke.com>	LOPSA
John Boris <jborissr@gmail.com>	LOPSA
John Dewey <john@dewey.ws>	AT&T
Carolyn Rowland <unpixie@gmail.com>	
Jordan Dea-Mattson <jdm@dea-mattson.com>	Numenta, Inc.
Sean Escriva <sean.escriva@gmail.com>	Heavy Water Ops
Adam Compton <comptona@gmail.com>	
Franck Cuny <franck@lumberjaph.net>	SAY Media
Chris Nehren <cnehren@omniti.com>	OmniTI
Brian Rak <dn@devicenull.org>	
Divij Rajkumar <drajkuma1@gmail.com>	
Aslan Brooke <aslandbrooke@yahoo.com>	ZynxHealth.com, InfoQ.com
Glen Kaukola <gkaukola@cs.ucr.edu>	

Continued on next page

Table 1.1 – continued from previous page

Name	Company
Spencer Krum <krum.spencer@gmail.com>	UTi Worldwide Inc.
Jeremy Grosser <jeremy@synack.me>	Uber Technologies, Inc.
Hugo Landau <hlandau@devever.net>	
Konark Modi <modi.konark@gmail.com>	MakeMyTrip.com
Josh Reichardt <josh.reichardt@gmail.com>	thepracticalsysadmin.com
Ben Reichert <ben@benreichert.com>	
Simon Aronsson <simon.aronsson@gmail.com>	itshale.com, simonaronsson.se
Andrew Langhorn <andrew@ajlanghorn.com>	
Abubakr-Sadik Nii Nai Davis <dwa2pac@gmail.com>	
Mike Julian	
Bram Verschueren	

Guidelines

Other places may use the phrase "Code of Conduct", or something similar - this is essentially that.

Like the technical community as a whole, the Ops School team and community is comprised of a mixture of professionals and volunteers from all over the world, who collaborate on bettering Ops School, including mentorship, teaching and connecting people.

Diversity is one of our biggest strengths, but it can also bring increased communication challenges at times.

To that end, as you are working with others on Ops School, please keep in mind the following guidelines, which apply equally to founders, mentors, those who submit new content or ideas, and to anyone who is seeking help and guidance.

The following list isn't exhaustive; it is intended to help all of us communicate well so that the Ops School community can work better together:

- Be welcoming, friendly, and patient.
- Be considerate.
- Be respectful.
- Be professional.
- Be careful in the words that you choose.
- When we disagree, let's all work together to understand why.

This list applies to all forms of communication: IRC, mailing lists, and any other forum that is used involving of Ops School. Please keep in mind that:

- Your work will be used by other people, and you in turn will depend on the work of others.
- Decisions that you make will often affect others in the community.
- Disagreements happen, but should not be an excuse for poor behavior and bad manners. When disagreements do happen, let's work together to solve them effectively and in a way that ensures that everyone understands what the disagreement was.
- People may not understand jokes, sarcasm, and oblique references in the same way that you do. Remember that and be kind to the other members of the community.
- Sexist, racist, or exclusionary comments are not welcome in Ops School.

- The strength of the community comes from all of the people in it, their wide range of backgrounds, and the contributions everyone makes. Let's keep working together, learning from our mistakes, and making the community as awesome as it can be.

(Inspired by the Chef Community, Speak Up! project, and predecessors)

Careers in Operations

Taking a step into the world of Operations can be daunting. At present there are few professional degrees, and the ones that exist focus on specialized topics such as systems administration, network engineering or security.

However, the Operations landscape is significantly larger than that. This page aims to provide a detailed view on the careers you can move to as you enter the field.

Deciding a career path

Around the end of the 101 level of Ops School, you will want to start thinking about which career path you want to take. The paths tend to overlap in places, and both paths can lead to similar places down the road.

Imagine two roads that run straight, and occasionally get close or overlap, but generally head in the same direction.

Your two options are usually:

- Operations generalist
- Operations specialist

The one you choose, should be the one you feel most comfortable with. Over time, as your skills grow, both paths should lead to similar opportunities: team leadership, senior technical leadership, and management.

The way you travel each path is considerably different, and it isn't uncommon to switch from one path to another once or twice.

Generalized career paths

Persons in generalized careers are often in high demand by employers. While the adage states "Jack of all trades, master of none", an operations generalist is very much expected to be *"Master of almost all trades."*

It's quite common to see a good Operations Engineer who is well-versed in systems administration, database administration, storage engineering, network engineering, security engineering, and other topics.

Similarly, Operations managers are expected to know a wide variety of subjects at a reasonable enough level to make good technical decisions. Often, people who choose to become Operations managers begin their careers as either generalists, or specialists in one or more fields.

Generalist roles are often most highly prized in technical companies and startups, where it is beneficial to be able to work with multiple technologies and groups well.

Operations Engineer

Operations engineers are expected to be able to wear any of the following hats (and sometimes all of them, at the same time):

- Database administrator
- Systems administrator
- Network engineer
- Security engineer
- Architect
- Performance engineer
- Part-time software engineer
- Storage engineer
- High Performance Computing engineer

The role can be summed up appropriately as this: When somebody wants to know about any production system, no matter what it is, they will ask the Operations engineer. Your job as the Operations engineer, is to know the system well enough to be able to answer any question, or know how to find the answer quickly.

In the medical field, you would be a Doctor of Internal Medicine. In culinary, you would be an Iron Chef.

Operations Manager

Operations managers are similar in many ways to operations engineers. If you have read the 21 Irrefutable Laws of Leadership, then this is a leadership role, not a management role. An Operations manager works to bring their team and other teams closer together. A good reference on managing an operations team, is Michael Rembetsty's PICC 2012 talk on DevOps Management. It covers moving from traditional Operations to DevOps, and then developing and growing an Operations team.

Specialized career paths

Unlike generalists, specialists are often hired to take care of certain components of larger systems. While generalists tend to focus on increasing both the breadth and depth of their knowledge over time, specialists work to become deep subject matter experts. Common areas of focus are databases, networking, security, storage, capacity planning, project management, training, and more. In almost all cases these require at least the 101 level of understanding, and fully understanding through the 201 level is better.

Systems Administrator

The Systems Administrator is the classic and probably most recognized Operations role. Key responsibilities usually include managing desktops, servers, operating systems, databases, middleware and applications.

Systems Administrators can range from "jack of all trades" with knowledge of multiple systems and platforms to specialists who focus on one system or platform, for example Microsoft Windows or Linux.

Whilst perhaps more "general" than some of the other specialist roles, Systems Administrators tend to focus on managing individual hosts, usually desktops or servers, rather than looking more broadly at infrastructure like Operations generalists.

Database Administrator

Database Administrators, or DBAs, are specialists in managing the performance, security and stability of database systems. Once a common role, they are less frequently seen today and much of the work involved in this role has been replaced by more advanced database systems, automation and the growth of these skills in related roles.

DBAs usually have specialized skills in managing the performance of database systems, are often experts in understanding database features like stored procedures and are called upon to improve the performance of database systems using techniques like query analysis.

Network Engineer

Network Engineers are Operations people who focus on network devices and network management. Network Engineers manage the provisioning, configuration, security and availability of networking infrastructure.

Network Engineers are able to architect and design networks both internal to organizations and between organizations and their customers, for example Internet-facing infrastructure. As a result their skills often overlap with Security Engineers in technologies such as firewalls, proxies and gateway services like Virtual Private Networks (VPN).

They are expected to have a deep understanding of the *OSI* model and its components especially physical networking technologies like Ethernet and transport and session components like TCP/IP, UDP, and SSL. They are often called to identify and fix problems with applications and their connectivity and hence have strong skills in diagnosis, log and data analysis, and troubleshooting.

Security Engineer

Whilst seen by many as a separate discipline, Security Engineers are Operations people with a focus on security and security technology. Security Engineering roles can include:

- Traditional Systems Administrators who maintain security equipment like firewalls and proxies
- Specialists who design and manage complex cryptographic systems
- Penetration testers who attempt to identify security vulnerabilities in infrastructure and applications
- Engineers with a focus on Identity Management who manage complex authorization, access control and authentication systems
- Analysts and incident response personnel who respond to security events and incidents

Security Engineers usually have many of the same skills as their more mainstream Operations colleagues but often include deeper skills in fields such as Compliance Management (ensuring companies maintain compliance to industry and government regulations), Risk Management (identifying, documenting and managing Risk), education (teaching people about how to stay secure), Cryptography, and related areas.

Storage Engineer

Seen largely in enterprise-scale organizations, Storage Engineers focus on managing storage technologies such as disk arrays, Network Attached Storage (NAS) devices, Storage Area Networks (SANs), Tape and Media management systems and related backup technologies.

Storage Engineers provision, configure and manage this infrastructure which then provides storage for web and file servers, database systems, applications and backups.

They usually have strong skill overlaps with Network Engineers (with so much modern storage being network-attached in some manner) and usually have strong skills in capacity planning and performance management of infrastructure.

High Performance Computing Engineer

High Performance Computing (HPC) involves large scale computing infrastructure, which is at the cutting edge of presently available technology, oftentimes making it at TOP500.ORG

Typically, HPC platforms are used for scientific computing, big data, complex models and may have applications in fields as diverse as physics, finance, medicine, defence, economics etc. for instance: Meteorology and Climate prediction may appear first as a tiny slice of the bigger picture yet they tend to be applicable to several aspects of human life and scientific efforts to improve it.

HPC engineers are expected to master an array of high-end technologies in fields such as networking (InfiniBand, multi-Gigabit Ethernet), computing (several computer architectures), parallel storage/filesystems/filers (Lustre, GPFS, Isilon, NetApp, PanaSAS), as well as be able to give advice on a number of software components (gnu/intel/pgi compilers, debuggers, mpi stacks, FFTW, linear algebra and other optimized math libraries etc etc).

Most importantly, HPC engineers should be able to interface with other operations engineers, each a specialist in their own field, in order to let all systems run both at top performance and within the maximum range of their reliability envelope, since HPC downtimes come at a high cost.

How to become an operations engineer

Employers look for a number of things when hiring junior engineers and admins:

- An understanding of the basics of Unix-style and/or Microsoft Windows operating systems, including installing the operating system, installing and configuring packages and editing files. You can find these in the *Unix fundamentals 101 MS Windows fundamentals 101* sections.
- Knowledge of common internet protocols and systems, and how to implement and manage them, including *DNS 101*, *SMTP 101* and *Networking 101*.
- A solid grasp of how to *troubleshoot problems*.
- Repeated success in completing the *Labs exercises*.

These are only the beginning, and the bare minimum you should expect to know as a junior level engineer. While demand for operations engineers continues to grow at a fast pace, you will still find there is competition for positions. The more you know, the stronger your chances of finding a job.

Simply reading the 101 sections of Ops School is not sufficient, you must *understand* it. As an example: The *DNS* section explains there are 13 root name servers. In addition to knowing this fact, you have to understand why there are 13 root name servers and be able to explain it confidently to others.

Sysadmin 101

System Administration is the blanket title for a myriad of job responsibilities involving the management of information systems and/or services. We choose to refer to it as a "blanket title" because the responsibilities are broad and can differ from organization to organization. In some organizations, it even differs from group to group!

System Administration is often referred to as "operations" or "ops" too - you might have guessed that though, having decided to enroll in Ops School.

Like almost everything in this world, system administration does not exist in a vacuum. There are two other groups of people that exist to make an information system or service come together: developers and customers. Customers are often referred to as end-users or people or humans - basically, they're the folks you're implementing a system or service for.

A system administrator experiences life in all three of these roles: sometimes you will be tasked with maintaining a system or service, and sometimes you may be developing parts of a system. Many times, you'll be someone else's customer.

This section will get into what system administration (ops) professionals do, what developers do, where they compare and contrast, what role that ops plays in the larger business context, a mindset and approach to systems administration, demonstrate the importance of problem solving and ethics in the every day life of a system administrator.

What is Systems Administration?

Like we mentioned above, the role of the System Administrator is diverse.

To illustrate some of the latitude present in the role, consider how the following descriptions compare and contrast:

- **Matt works for an elementary school.** He's referred to by other employees at the school as the system administrator. To the other employees, Matt ensures that the computers "just work." Realistically, Matt spends time making sure that the systems that everyone uses are patched - to avoid security issues. Other employees talk to him about software they may need installed – on their computer, or maybe all of them! Matt also has the responsibility of making sure that the school's website is available for parents to get information on, and that the classroom management software is up and running and accepting requests. Matt also needs to be able to look into the future and figure out how much space the employees may need on the server, and be able to justify that to his boss. When report card time runs around, Matt has to make sure that the teachers are able to give their report cards out on time.
- **Sarah works for a large website that many people use on a day to day basis.** Downtime is a big deal for this website: after all, when the site is down, people aren't spending money. When these problems occur, Sarah gets paged to determine the root cause of the issue and respond. Fortunately for Sarah she's not always being paged. During the normal day-to-day operations of the site, Sarah writes various scripts and programs that help her other administrators manage the site on a day to day basis, and strives to reduce the amount of "manual" tasks that the team she's on has to deal with.
- **Will works for a small town.** The town has 35 employees who use computers every day, and they need access to email and other collaboration tools. With so many services and options for e-mail, which one makes the most amount of sense for the town? Will collects input from the employees about what capabilities are important, input from legal council about what laws a town must follow regarding electronic communication, and evaluates a few services or products with his users' needs in mind along with the town's. If he ends up using a third-party service, he'll have to figure out how to roll it out to the users. If Will chooses to host himself, he'll have to coordinate getting the appropriate infrastructure set up to make sure that it can meet the capacity of the users. He'll also need to set it all up.
- **Karen works for a large business.** The business has many groups, but she works with one set of users - hardware engineers. She maintains a network that they use to run their hardware simulations before the hardware goes out to be built. She regularly writes scripts to reduce the number of manual steps something takes - like

adding new users to the system. She also has to coordinate how the network will be patched regularly, what investments will be required for the network to continue operating at peak performance, and work with the other administrators to create something that works for the engineers at the business.

- **Darren works for a large financial company.** The company has many teams that do different things, but there's collaboration between the different system administrators - the operations team (staffed with administrators) implements things that the research team (also administrators) recommends based on the needs of the business. They work with developers too - since the financial company has a lot of custom software that is written by the developers. The developers have different goals when they write their software - so they meet with Darren and his team to make sure they understand the context surrounding where and how their application will run, and to make sure that Darren's team knows what they need as well. Darren's team may have ideas about what statistics are important to monitor.
- **Chris works for a consultancy.** The company has many clients for which it provides software development and operations support. Every client has their own infrastructure, their own requirements - some of them very similar to the previous examples, some very different. Therefore, Chris needs to be familiar with a wide range of technologies, in addition to being able to learn new things quickly. There's a lot of variety in the day-to-day job. While it's not his primary responsibility to write code, sometimes the job calls for it - where context has already determined the implementation language. Chris works closely not only with the clients but also with his developer coworkers and the rest of the operations team to meet client goals. In addition, Chris also works with the rest of the operations team to support the consultancy's own infrastructure.

Comparing Stories

Did you notice any themes throughout the stories? How do they contrast?

Highlighted themes include:

- **System Administrators rarely work on their own.** Some of the stories feature System Administrators who are the only "Information Technology" staff in their organization - but it's a requirement for them to work with everyone: the folks who "sign the paycheck at the end of the day," *and* the people who use their technology.
- **System Administration can be difficult if you don't automate!** Imagine if Matt had to install a software package on 200 machines by walking to each machine and installing it? How could he get any of his other requests done?
- **At times, the system administrator may need to write code.** Maybe that means a script that can take a list of users and add them to a system, maybe that means a script that can check to make sure a service is responding appropriately for a monitoring system.
- **System Administration has a wide variety of meanings to different people.** The other non-IT employees are looking for something that "just works" - the ops person is concerned with proper implementation of a service so it continues to be scalable and reliable for the future.

What is Development?

As mentioned in the section introduction, system administration doesn't exist in a vacuum. Computers don't just compute for the sake of it; not yet, anyway.

Developers write applications in a variety of programming languages to make a computer do *something*. Developers created the backend that allows you to order a book on a site like Amazon.com, post a status on Facebook, or research a topic in a literary journal database.

These backends can be big, and have many considerations behind their design.

- **Rachel works for an ecommerce company.** The company has a large team of developers, all working in different applications, usually in Ruby or JavaScript, and the applications talk to each other via APIs to support

the main site that customers interact with. Her obligations to the business are to create new features for the website and maintain or fix the old ones. Her obligations to her other developers are to keep the code clean and readable, with tests so others can confidently refactor her code. She works closely with the Operations team to make the app-level changes that help the ops team maintain a robust infrastructure, like making pages fully cachable or eliminating unnecessary HTML, CSS, JavaScript or images being sent to the browser.

- **Tyler is a systems developer at a small technology company.** He takes complex processes, like developing a search engine or collecting statistics, and creates several pieces of software to accomplish those tasks. He works primarily in C, Perl and Python and has to have a deep understanding of the operating system his code will run on. He works closely with his Operations engineers to make sure his code is performant and on capacity planning.
- **Francesca is one of five students working on the website for her school's internationally famous hackathon.** She and her friends run a Tornado web server with a Postgres database. For the front end, they use GitHub pages and React. Their website is hosted on Amazon Web Services. The team uses New Relic and Pingdom to profile the code to see what time, CPU, and memory it consumes, as well as to alert them if any aspect of the site is acting abnormally.
- **Jean works at a large technology firm.** The company has many teams of developers, all working on different aspects of their product. Jean is a data engineer who manages the collection and display of certain statistics that are useful for other employees of the company. If Jean has any issues with their computer or development environment, they ask the people running the information technology desk. Core parts of the technology at this company are maintained by site reliability engineers. Jean also has an embedded site reliability engineer on their team who makes sure that the features they create are scalable and maintainable.

Todo

"What is Development" Section needs more developer perspective.

Contrasting Development and Operations

At the end of the day, both groups have an important shared goal: to ensure that a system or service remains as *available* (think: accessible, usable, works how people expect it to) as a customer expects it to be. You'll see references to the idea of how "available" a system or service is later when Service Level Agreements (SLAs) are discussed.

That being said, Development and Operations have different day-to-day thoughts.

Operations thoughts include:

- How are we going to install (or deploy) the servers that run this application?
- How will we monitor the system/service to make sure it's working as we expect?
- Can we deploy this system or service in a way that is easy to maintain?
- What are the pros/cons of implementing this application in this way?

Development thoughts include:

- How will I implement message passing between two parts of this application?
- What's the best algorithm to use for searching through this amount of data?
- Should I be thinking of a key-value store for this data vs. using a relational database?
- What language will I implement this in?

Again, this is by no means an exhaustive list - entire books could be written about the subject. It's just to give a feel for the different considerations.

History of Development and Operations

Historically, Developers made a product or application and Operations implemented it. Some companies still use this model today. Unfortunately, the effects of this style of work can be dangerous:

- **Developers and Operations personnel may have different goals.** Something important to the operations folk may not be important to the Development folk.
- **Siloing these two organizations means that the most important goal of service availability is compromised.** The person who is using your program or service doesn't care who made a mistake, the service is down.
- **Speaking of mistakes:** when companies don't encourage their development and operations teams to work together, it's possible that groups get too invested in assigning blame, instead of working together to fix an issue

Fortunately, in recent years, many organizations have made more effort to ensure that both teams are familiar with the concepts of the other; this is often referred to as *DevOps*, the combination of Development and Operations. Recall *Darren's story* - he's an operations person who worked with the developers to ensure that the developers understand the environment that their application will run on. The street went both ways, though: the developers need to share how they plan to implement various product features so that the operations team can figure out how to best support the developers' needs.

If you're working in an environment without developers, that's OK. There are other people who you share a common, larger goal with. Businesses may have analysts that interpret needs and look to you for assistance. In *Karen's story*, she supports hardware engineers who have requirements to deliver a particular sensor. Their ability to work hinges on Karen's ability to deliver a service for simulation that is available for them to work, which requires an understanding of their requirements and needs as well.

What System Administration Isn't

System Administration, like many things in life, can suffer from a cultural perception issue. Primarily, some outside of the operations field that believe that the role of IT is not to innovate; rather that IT exists to enforce rules decided on by others.

We challenge that mindset. System administration is not about saying "no" to requests but about finding ways to intelligently fulfill the needs of a business in a way that increases maintainability, usability, and security for a group of people and enhances their ability to do useful work.

Another perception issue is that of the BOFH. While there's something of a grain of truth to the meme, most system administrators who adopt such an attitude quickly find themselves out of work. Beyond the social implications, this sort of obstructionist, adversarial behavior is the least effective way of addressing problems. The best path to success and satisfaction is collaboration, not obstinacy. This requires cooperation from all involved, whether their role is technical or not. An administrator's job is to create order from chaos, to control the technology – not the people. As with all social interactions, everyone has their own needs and goals, and it can be complex to navigate safely, but it is critical to the success of the organization.

Problem Solving

Learning Styles - Ways to improve skill set

Methodologies for finding solutions

Ethics

- LOPSA ethics statement/SAGE(now LISA) ethics statement?

Where to draw the line

Keeping yourself safe from yourself

Unix fundamentals 101

File systems

Background

A filesystem is the street grid of your hard drive. It's a map of addresses to where data is located on your drive. Your operating system uses the filesystem to store data on the drive.

There are a number of different types of filesystems. Some are better at handling many small files (ReiserFS), some are much better at large files and deleting files quickly (XFS, EXT4).

The version of Unix you use will have picked a filesystem which is used by default, on Linux this is often EXT3.

Understanding the way filesystems work is important when you have to fix issues related to disk space, performance issues with reading and writing to disk, and a host of other issues.

In this section we will discuss creating partitions, file systems on those partitions, and then mounting those file systems so your operating system can use them.

Navigating the filesystem

When you log into a Unix system, you will be given a command line by the *shell* which may look something like this:

```
user@opsschool ~$
```

By default you will be in the "current working directory" of the process that spawned the shell. Normally this is the home directory of your user. It can be different in some edge cases, such as if you manually change the current working directory, but these cases are rare until you start doing more advanced things.

You can find the name of the current directory with the `pwd` command:

```
user@opsschool ~$ pwd
/home/opsschool
```

You can see the list of files and directories in this directory with the `ls` command:

```
user@opsschool ~$ ls
file1.txt file2.txt tmpdir
```

The `ls` command also accepts the `-l` argument to provide a long-listing, which will show you permissions, dates, ownership and other information:

```
user@opsschool ~$ ls -l
-rw-r--r--  1 opsschool opsgroup   2444 Mar 29  2012 file1.txt
-rw-r--r--  1 opsschool opsgroup  32423 Jun 03  2011 file2.txt
drwxr-xr-x 15 opsschool opsgroup   4096 Apr 22  2012 tmpdir
```

You can see the contents of other directories, by giving the name of the directory:

```
user@opsschool ~$ ls -l /
dr-xr-xr-x    2 root root  4096 Apr 26  2012 bin
dr-xr-xr-x    6 root root  1024 Sep 18 14:09 boot
```

```
drwxr-xr-x   19 root root  8660 Jan  8 16:57 dev
drwxr-xr-x  112 root root 12288 Feb  8 06:56 etc
drwxr-xr-x   67 root root  4096 Feb  7 19:43 home
dr-xr-xr-x   13 root root  4096 Mar  6  2012 lib
drwx------    2 root root 16384 Sep 18  2011 lost+found
drwxr-xr-x    5 root root  4096 Nov 19 18:53 mnt
drwxr-xr-x    4 root root  4096 Sep  4 15:15 opt
dr-xr-xr-x 1011 root root     0 Sep 23  2011 proc
dr-xr-x---   10 root root  4096 Jan 23 23:14 root
dr-xr-xr-x    2 root root 12288 Oct 16 22:23 sbin
drwxr-xr-x   13 root root     0 Sep 23  2011 sys
drwxrwxrwt   65 root root 16384 Feb 11 04:37 tmp
drwxr-xr-x   16 root root  4096 Feb  8  2012 usr
drwxr-xr-x   27 root root  4096 Nov  4 03:47 var
```

You may have noticed that the names of directories follow a pattern. `/` is also called the root directory. All directories and files are contained under it. From the first example, the `/` directory contains the `/home` directory, which in turn contains the `/home/opsschool` directory.

To change directories, use the `cd` command:

```
user@opsschool ~$ cd /tmp
user@opsschool ~$ pwd
/tmp
```

There may be times you need to find a file on your filesystem, based on its name, date, size, or other particulars. For this you can use the `find` command:

```
user@opsschool ~$ find /home/opsschool -type f -name file3.txt
/home/opsschool/tmpdir/file3.txt
```

Working with disks in Linux

Disks in Linux are normally named `/dev/sda`, `/dev/sdb`, etc. If you are in a VM, they may be named `/dev/xvda`, `/dev/xvdb`, etc. The last letter ("a", "b", "c"..) relates to the physical hard drive in your computer. "a" is the first drive, "b" is the second.

If you have an already configured system, you will likely see entries like this:

```
-bash-4.1$ ls -la /dev/sd*
brw-rw---- 1 root disk 8, 0 Jul  6 16:51 /dev/sda
brw-rw---- 1 root disk 8, 1 Sep 18  2011 /dev/sda1
brw-rw---- 1 root disk 8, 2 Sep 18  2011 /dev/sda2
brw-rw---- 1 root disk 8, 3 Sep 18  2011 /dev/sda3
```

The number at the end of each drive maps to the partition on the drive. A partition refers to a fixed amount of space on the physical drive. Drives must have at least one partition. Depending on your specific needs, you might want more than one partition, but to start with, we'll assume you just need one big partition.

Configuring your drive with partitions

The `parted` tool is for modifying and creating disk partitions and disk labels. Disk labels describe the partitioning scheme. Legacy Linux systems will have the msdos partitioning table, although newer systems with EFI use the gpt partitioning table. Drives with `msdos` disk labels will have a Master Boot Record, or MBR, at the beginning of the drive. This is where the bootloader is installed. GPT-labeled drives will usually have a FAT-formatted partition at the beginning of the disk for EFI programs and the bootloader.

parted has a subshell interface. It takes as an argument a device name.

```
root@opsschool ~# parted /dev/sda
GNU Parted 2.3
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 42.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      8225kB  42.9GB  42.9GB  primary  ext4         boot

 (parted)
```

In this example parted ran against /dev/sda. The user then used the print command to print out information about the disk and the current partitioning scheme. The user found a 43 GB disk using the msdos partition table format. The disk had one partition which was flagged as bootable. Looking at a second example:

```
root@opsschool ~# parted /dev/sdb
GNU Parted 2.3
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Error: /dev/sdb: unrecognised disk label
(parted) mklabel msdos
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 8590MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start  End  Size  Type  File system  Flags

(parted) mkpart primary 1 1G
(parted) set 1 boot on
(parted) mkpart primary 1G 5G
(parted) mkpart primary 5G 7G
(parted) mkpart primary 7G 8G
(parted) p
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 8590MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  1000MB  999MB   primary               boot
 2      1000MB  5000MB  3999MB  primary
 3      5000MB  7000MB  2001MB  primary
 4      7000MB  8590MB  1590MB  primary

(parted)
```

parted failed to read the label, so the user created a new msdos disk label on the disk. After that parted was able to see that the disk was 8GB. We created a primary 1GB partition at the beginning of the disk for /boot and set the bootable flag on that partition. We created 4GB, 2GB, and 1GB partitions for root, var, and swap, respectively.

Formatting partitions with new file systems

New filesystems are created with the `mkfs` family of commands. There are a variety of file systems to choose from, `man fs` has a list of filesystems with short descriptions of each. Choosing a filesystem involves characterizing the workload of the filesystem and weighing engineering tradeoffs. On Linux systems, ext4 is a good general purpose choice. Following from the example above, we will create filesystems on each of the four partitions we created.

`fdisk` is another, older, tool to view and modify partitions on disk. It is limited to the msdos disk label.

```
root@opsschool ~# fdisk -l  /dev/sdb

Disk /dev/sdb: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0004815e

   Device Boot      Start         End      Blocks   Id  System
   /dev/sdb1   *        2048     1953791      975872   83  Linux
   /dev/sdb2         1953792     9764863     3905536   83  Linux
   /dev/sdb3         9764864    13672447     1953792   83  Linux
   /dev/sdb4        13672448    16777215     1552384   83  Linux
```

The first partition, to contain `/boot`, will be ext2. Create this by running:

```
root@opsschool ~# mkfs.ext2 /dev/sdb1
mke2fs 1.42 (29-Nov-2011)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
61056 inodes, 243968 blocks
12198 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=251658240
8 block groups
32768 blocks per group, 32768 fragments per group
7632 inodes per group
Superblock backups stored on blocks:
  32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

The second and third partitions, to contain `/` and `/var`, will be ext4. Create these by running:

```
root@opsschool:~# mkfs.ext4 /dev/sdb2
root@opsschool:~# mkfs.ext4 /dev/sdb3
```

The output of `mkfs.ext4` is very close to the output of `mkfs.ext2` and so it is omitted.

Finally, `/dev/sdb4` is set aside for swap space with the command:

```
root@opsschool:~# mkswap /dev/sdb4
Setting up swapspace version 1, size = 1552380 KiB
no label, UUID=cc9ba6e5-372f-48f6-a4bf-83296e5c7ebe
```

Mounting a filesystem

Mounting a filesystem is the act of placing the root of one filesystem on a directory, or mount point, of a currently mounted filesystem. The mount command allows the user to do this manually. Typically, only the superuser can perform mounts. The root filesystem, mounted on /, is unique and it is mounted at boot. See *The Boot Process*.

```
root@opsschool # mount -t ext4 -o noatime /dev/sdb1 /mnt
```

It is common to specify which filesystem type is present on /dev/sdb1 and which mounting options you would like to use, but if that information is not specified then the Linux mount command is pretty good at picking sane defaults. Most administrators would have typed the following instead of the above:

```
root@opsschool # mount /dev/sdb1 /mnt
```

The filesystem type refers to the format of the data structure that is used as the filesystem on disk. Files (generally) do not care what kind of filesystem they are on, it is only in initial filesystem creation, automatic mounting, and performance tuning that you have to concern yourself with the filesystem type. Example filesystem types are ext2, ext3, ext4, FAT, NTFS HFS, JFS, XFS, ZFS, BtrFS. On Linux hosts, ext4 is a good default. For maximum compatibility with Windows and Macintosh, use FAT.

http://en.wikipedia.org/wiki/Comparison_of_file_systems

The fstab, or file system table, is the file that configures automatic mounting at boot. It tabulates block devices, mount points, type and options for each mount. The dump and pass fields control booting behavior. Dumping is the act of creating a backup of the filesystem (often to tape), and is not in common use. Pass is much more important. When the pass value is nonzero, the filesystem is analyzed early in the boot process by fsck, the file system checker, for errors. The number, fs_passno, indicates priority. The root filesystem should always be 1, other filesystems should be 2 or more. A zero value causes checks to be skipped, an option often used to accelerate the boot process. In /etc/fstab, there are a number of ways to specify the block device containing the filesystem . UUID, or universally unique identifier, is one common way in modern Linux based systems to specify a filesystem.

```
root@opsschool # cat /etc/fstab
```

```
# <file system> <mount point>  <type>  <options>         <dump>  <pass>
/dev/sda5         /            ext4    errors=remount-ro 0       1
/dev/sda6         none         swap    sw                0       0
/dev/sda1         /boot/efi    auto    auto              0       0
```

This /etc/fstab file mounts /dev/sda5 on / using the ext4 filesystem . If it encounters a filesystem corruption it will use the fsck utility early in the boot process to try to clear the problem. If the physical disk reports errors in writing while the filesystem is mounted, the os will remount / readonly. The /dev/sda6 partition will be used as swap. The /dev/sda1 partition will be mounted on /boot/efi using autodetection, the partition will not be scanned for filesystem errors.

```
root@opsschool # cat /etc/auto.master
```

```
/home -rw,hard,intr,nosuid,nobrowse bigserver:/exports/home/&
/stash ldap:ou=auto_stash,ou=Autofs,dc=example,dc=com -rw,hard,intr,nobrowse
```

The auto.master file controls the autofs service. It is another way to tabulate filesystems for mounting. It is different from the /etc/fstab because the filesystems listed in auto.master are not mounted at boot. The automounter allows the system to mount filesystems on demand, then clean up those filesystems when they are no longer being used. In this case, the system mounts home directories for each user from a remote NFS server. The filesystem remains unmounted until the user logs in, and is unmounted a short time after the user logs out. The automounter is triggered by an attempt to cd into /home/<key>, it will then attempt to find an nfs share on /exports/home/<key> and mount it on /home/key, then allow the cd command to return successfully. The /home example above is using the & expansion syntax, the second line is using the LDAP syntax to look up a key under /stash/<key> in LDAP.

LDAP will be covered later in the curriculum. The `auto.master` file is known as `auto_master` on FreeBSD, Solaris, and Mac OS X.

Filesystem options

Passing options to the `mount` command, or inserting them into the `/etc/fstab` file, control how the filesystem behaves. Different filesystems at different versions support different options, but some options are ubiquitous.

async

The `async` option sets writing operations to the filesystem to be asynchronous. This means that the `cp` command will exit normally before the entire copy is done, and that the system will persist the files to the disk at some point later on. You don't have a lot of guarantees here about when that will happen, though for a generally unloaded system you won't have to wait long. It is also hard to tell when the filesystem is actually done with the copy. The `sync` utility can be used to force a filesystem sync to immediately persist to disk. The opposite of this option, the `sync` option is the default on most filesystems.

noatime

The `noatime` option tells the filesystem not to keep track of `atime` or access time. If you recall your `inode` lessons, you'll remember that the `inode` keeps track of three dates: `ctime` (change time), `mtime` (modification time), and `atime` (access time). Under normal circumstances, whenever a user reads from a file, the operating system will write a new `atime` to the `inode`. For large groups of small files, read by a number of people, or by automated processes, this final write operation can hurt disk performance. As a result, many admins will turn off `atime` on filesystems to increase performance.

Note that `atime` is not really a security/auditing feature. Any regular user can use the `touch` utility on a file to set the `atime` to some point in the past, if they have the appropriate permissions for that file.

Also note that contrary to popular belief, `ctime` does *not* store a file's creation time. The change time stored in `ctime` is when the file's attributes or contents were changed, whereas the modification time stored in `mtime` only changes if the file's contents are modified.

ro

The `ro` option, called 'read-only', tells the filesystem to not allow writes to any of the files on the filesystem. This is useful for a legitimately read-only backing store such as a CD-ROM. It is also useful for protecting the filesystem from yourself or others, such as when you are mounting a drive you pulled from a different machine. If the filesystem is mounted read-only, you can't accidentally delete any data off of it. It is common for network shares to be mounted read-only, this way client machines can copy files from the network share, but can't corrupt the share with write locks or deletes. Sometimes, when the operating system detects that the block device (such as a hard drive) beneath the filesystem is beginning to fail, the operating system will remount the filesystem read-only. This will cause lots of problems for your running system, but it is intended to give you the maximum amount of time to copy your important data off of the filesystem before the disks beneath it fail completely. If this happens you can usually see it by checking `mount` for filesystems that should be read-write mounted as read-only. You can also check `dmesg` for messages like:

```
root@opsschool # dmesg

[26729.124569] Write(10): 2a 00 03 96 5a b0 00 00 08 00
[26729.124576] end_request: I/O error, dev sda, sector 60185264
[26729.125298] Buffer I/O error on device sda2, logical block 4593494
[26729.125986] lost page write due to I/O error on sda2
```

These messages strongly indicate that the disk `/dev/sda` is dying. In some cases you can recover the filesystem with the file system checker `fsck`. You may also be able to force remount the filesystem read-write, as shown in the `remount` section below.

rw

The `rw` option, called 'read-write', tells the filesystem to allow reads and writes to all files on the filesystem.

Mounting a filesystem read-write is usually the default. There are times where you will not be able to make a read-write mount, such as when the backing physical media is fundamentally read-only, like a CD-ROM.

remount

Sometimes a filesystem will be mounted read-only, either by default, or because the operating system has remounted it `ro` because of disk failures. It is possible to remount the filesystem read-write with the following command:

```
root@opsschool ~# mount | grep boot
/dev/sda1 on /boot type ext3 (ro)
root@opsschool ~# mount -o remount,rw /boot
root@opsschool ~# mount | grep boot
/dev/sda1 on /boot type ext3 (rw)
```

The syntax of the remount option is `-o remount,<option>`.

noexec

The `noexec` option tells the filesystem to ignore the execute bit on a filesystem. This would never work for the root filesystem, but it is often used as a security measure on world-writeable filesystems such as `/tmp` . Note that there are many ways to get around this restriction.

nosuid

Like the `noexec` option, the `nosuid` option ignores any setuserid bits set on files in the filesystem. This is also used for security purposes. It is generally recommended for removable devices such as CD-ROMs, USB sticks, and network filesystems.

nobarriers rbind

How filesystems work

Files, directories, inodes

Inodes

What they contain, how they work

The POSIX standard dictates files must have the following attributes:

- File size in bytes.
- A device ID.
- User ID of file's owner.

- Group ID of file.
- The file's mode (permissions).
- Additional system and user flags (e.g. append only or ACLs).
- Timestamps when the inode was last modified (ctime), file content last modified/accessed (mtime/atime).
- Link count of how many hard links point to the inode.
- Pointers to the file's contents.

http://en.wikipedia.org/wiki/Inode

File system layout

File system hierarchy standard is a reference on managing a Unix filesystem or directory structure.

http://www.pathname.com/fhs/

Fragmentation in unix filesystems

Filesystem objects

Filesystem contain more than just files and directories. Talk about devices (mknod), pipes (mkfifo), sockets, etc.

Shells

What is a shell?

A [LilU]nix shell is the command-line interface between the user and the system. It is used to perform some action, specifically, typing commands and displaying output, requested by the user.

Introduction to Bash

Bash is known as the Bourne-again shell, written by Brian Fox, and is a play on the name of the previously named Bourne shell (/bin/sh) written by Steve Bourne. [1]

Shell Fundamentals

Command-line Editing Modes

Over time, as one progresses with using the shell, it will become necessary to execute commands that may contain many arguments or have output piped to other commands. Beyond simple commands such as `cat` and `ls`, it can be tedious should one need to re-type, or, in most cases, edit, a given command. Luckily, `bash` provides a facility to make this scenario easier: the command-line editing mode.

The command-line editing mode emulates the movement functions of two common text editors, `emacs` and `vi`. In the case of the shell, the cursor's movement is being controlled.

Todo

Tighten up the above sentence. It's wordy and doesn't seem to make the point I want it to make.

---

[1] C Programming by Al Stevens, Dr. Dobb's Journal, July 1, 2001

By default, `bash` operates in `emacs` mode.

Example Edit Commands The following commands are very common while using the `emacs` mode.

- `Ctrl-b`: Move backward by one character
- `Ctrl-f`: Move forward by one character
- `Ctrl-a`: Move to the beginning of the line
- `Ctrl-e`: Move to the end of the line
- `Ctrl-k`: Delete from the cursor forward
- `Ctrl-u`: Delete from the cursor backward
- `Ctrl-r`: Search the command history

The following commands are very common while using the `vi` mode.

- `h`: Move backward by one character
- `l`: Move forward by one character
- `0`: Move to the beginning of the line
- `$`: Move to the end of the line
- `d$`: Delete from the cursor to the end of the line
- `d0`: Delete from the cursor the beginning of the line
- `:history s`: Search the command history

Setting the Mode One can manually set the desired mode using the below commands.

- `emacs`

  ```
  set -o emacs
  ```

- `vi`

  ```
  set -o vi
  ```

See *Text Editing 101* for details on appropriate edit commands to use on the command line.

Environment variables

Environment variables are used to define values for often-used attributes of a user's shell. In total, these variables define the user's environment. Some environment variables provide a simple value describing some basic attribute, such the user's current directory (`$PWD`). Others define the behavior of a command, such as whether or not the `history` command should log repeated commands individually or log the repeated command once (`$HISTCONTROL`).

$PATH The most common, or most recognized, environment variable is the `$PATH` variable. It defines the set of directories that the shell can search to find a command. Without an explicit path provided when calling a command (i.e. `/bin/ps`), the shell will search the directories listed in the `$PATH` variable until it finds the command. If the command is not found in any of the defined directories in `$PATH`, the shell will produce an error explaining as much.

```
$ foobar -V
-bash: foobar: command not found
```

To view the contents of the `$PATH` variable, use `echo` to print the variable's value:

```
$ echo $PATH
/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin
```

The order of the directories in the `$PATH` variable, from left to right, is important; when searching directories for a command, the shell will stop looking after it finds its first match. In other words, using our example `$PATH` variable above, if there is a version of `ps` that exists in `/usr/local/bin` that is preferred (by the sysadmin) over the version that exists in `/bin`, the shell will still execute `/bin/ps` due to the precedence of the directories defined in the `$PATH` variable.

To list all of the shell's environment variables, use the `env` command:

```
$ env
HOSTNAME=foobar
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
USER=root
PATH=/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/usr/local/bin
MAIL=/var/spool/mail/root
PWD=/root/curriculum
PS1=[\[\e[33;1m\]\t \[\e[31;1m\]\u\[\e[0m\]@\[\e[31;1m\]\h\[\e[0m\] \W\[\e[0m\]]#
HISTCONTROL=ignoredups
SHLVL=1
SUDO_COMMAND=/bin/bash
HOME=/root
HISTTIMEFORMAT=[%Y-%m-%d %H:%M:%S]
OLDPWD=/tmp
```

Shell Profiles

Shell profiles are used to define a user's environment. In cases where an environment variable needs to be set or a script needs to be run at login, a profile can be used to ensure this happens consistently and automatically. Anything that can be run in a standalone shell script can be placed in a profile.

There are two types of profiles:

- Global profile (`/etc/profile`)
- User profile (`~/.bash_profile`)

`/etc/profile` This is the global profile. Any environment variable set in this file applies to all users. Any script called from this file is executed for all users.

`~/.bash_profile` This is the user profile. Any environment variable set in this file applies to the user only. Any script called from this file is executed for the user only.

Profile Precedence **NOTE**: It is possible to override settings from `/etc/profile` via `~/.bash_profile` as `~/.bash_profile` is executed after `/etc/profile`.

Special Environment Variables

Certain variables exist that provide useful information about what is happening in the environment. For example, it may be necessary to know the ID of a running process or the exit status of an executed command.

Process ID: `$$` To determine the process ID (PID) of the current shell, first run `ps` to find the PID, then run `echo $$` to confirm the PID.

```
$ ps -efl | grep bash
 502 20440 20439   0 10:25PM ttys001    0:00.01 -bash                 4006  31  0  2433436   1228 -
 502 20447 20440   0 10:29PM ttys001    0:00.00 grep bash             4006  31  0  2432768    620 -
$ echo $$
20440
```

Background Process ID: `$!` Occasionally, commands will need to be executed in the background (via the `&` operator). The PID of that process can be found using the `$!` variable. For example, call `sleep` for 30 seconds and place it in the background. Then `echo $!` to see the PID. Alternatively, call `ps` to confirm the PID.

```
$ sleep 30 &
[1] 20450
$ echo $!
20450
$ ps -efl | grep sleep
502 20450 20440   0 10:33PM ttys001    0:00.00 sleep 30              4006  31  0  2432748    496 -
```

Exit Status: `$?` When a command is executed, it always has an exit status. That status defines whether or not the command was successful. For success, the exit status is **0**. Non-zero values denote failure. Many commands provide multiple exit codes to help define what the reason for failure was. This helps the user troubleshoot any problems.

As an example, try to list a known file within a user's home directory, then list a file that is known **not** to exist. After each command, execute `echo $?` to see the exit status.

```
$ ls .bash_profile
.bash_profile
$ echo $?
0
$ ls foobar
ls: foobar: No such file or directory
$ echo $?
1
```

History

The history is a handy facility within `bash`; it stores all of the commands that the user has executed.

To see the history, simply type `history` and all of the stored commands will be displayed to the terminal. Similarily, run `cat ~/.bash_history` to see all stored commands.

Re-executing Commands A benefit of storing command history is that the commands can be easily recalled. To execute the last command, type `!!`:

```
$ ls
file1 file2 dir1
$ !!
ls
file1 file2 dir1
```

Note that the last command is echoed just above the output of that command.

To execute the nth command in history, run `!n` where `n` is the line number of the command as found in the output of `history`:

```
$ history | less
  1  ls -la
  2  ls -F
  3  pwd
$ !2
ls -F
file1 file2 dir1/
```

Searching History Finally, one can search the history by typing `Ctrl-r` followed by a string to match a command:

```
$ (reverse-i-search)'ls': ls -F
```

To execute the command (if a match is found), simply type Enter.

Job control

From time to time, it may be necessary to manage commands running in the background. These are typically referred to as jobs. A command can be placed in the background via the `&` operator. Use the `jobs` command to see the job in the background. To bring it back to the foreground, run `fg`:

```
[23:24:22 ~]$ sleep 30 &
[1] 20969
[23:24:26 ~]$ jobs
[1]+  Running                 sleep 30 &
[23:24:29 ~]$ fg
sleep 30
[23:24:56 ~]$
```

While in the foreground, the job can be suspended via `Ctrl-z` and sent to the background once more using `bg`:

```
[23:24:56  ~]$ sleep 30 &
[1] 21078
[23:28:25  ~]$ jobs
[1]+  Running                 sleep 30 &
[23:28:27  ~]$ fg
sleep 30
^Z
[1]+  Stopped                 sleep 30
[23:28:33  ~]$ bg
[1]+ sleep 30 &
[23:28:37  ~]$ jobs
[1]+  Running                 sleep 30 &
[23:29:39 ~ ]$ jobs
[1]+  Done                    sleep 30
```

Multiple Jobs It is possible to have multiple jobs running in the background at the same time. Use the `jobs` command to track them via their job ID, noted between the square brackets after sending a job to the background. Knowing the job ID is helpful in the event that the job needs to be pulled into the foreground (via `fg`) or if the job needs to be killed:

```
[23:36:01  ~]$ sleep 120 &
[1] 21086
[23:36:16  ~]$ sleep 240 &
[2] 21087
[23:36:20  ~]$ jobs
```

```
[1]-  Running                 sleep 120 &
[2]+  Running                 sleep 240 &
[23:36:21  ~]$ fg %1
sleep 120
^C
[23:36:33  ~]$ jobs
[2]+  Running                 sleep 240 &
[23:36:35  ~]$ kill %2
[23:36:39  ~]$ jobs
[2]+  Terminated: 15          sleep 240
```

NOTE: When manipulating jobs with any command, the jobs are described by their ID using the `%n` notation where `n` is the job ID.

For information on ensuring running jobs continue, even when terminal connectivity is lost, see the sections on *GNU Screen* and *Tmux*.

Package management

Workflow

What is a package manager?

Modern operating systems use package managers to take care of the installation, maintenance and removal of software. On Windows this is Windows Installer (formerly Microsoft Installer). On Linux there are two popular package managers:

- APT (used by Debian, Ubuntu)
- RPM (RedHat, CentOS, Fedora, SuSe)

Specific commands for each system vary, but at their core they all provide the same functionality:

- Install and uninstall packages
- Upgrade packages
- Install packages from a central repository
- Search for information on installed packages and files

RPM and yum (RedHat, CentOS, Fedora, Scientific Linux)

In the following examples, we will be using the package `dstat`. The process however applies to any software you may want to install.

Yum provides a wrapper around RPM, which can be used to search for, and install packages from multiple package repositories. It also resolves dependencies, so that if a package has prerequisites, they will be installed at the same time.

If your Linux distribution uses RPM and yum, you can search for packages by running:

```
bash-4.0$ yum search dstat
======================== N/S Matched: dstat ========================
dstat.noarch : Versatile resource statistics tool
```

Installing packages

You can install a package using yum, by running:

```
bash-4.0$ yum install dstat

================================================================================
 Package        Arch            Version              Repository          Size
================================================================================
Installing:
 dstat          noarch          0.7.0-1.el6          CentOS-6           144 k

Transaction Summary
================================================================================
Install       1 Package(s)

Total download size: 144 k
Installed size: 660 k
Is this ok [y/N]:
```

If you have a downloaded RPM file, you can also install the package directly with the `rpm` command:

```
bash-4.0$ rpm -i dstat-0.7.0-1.el6.noarch.rpm
```

Upgrading packages

RPM and yum both make it easy to upgrade existing packages, too. Over time, new packages may be added to the yum repositories that are configured on your system, or you may have a newer RPM for an already installed package.

To upgrade a package using yum, when a newer package is available, simply ask yum to install it again:

```
bash-4.0$ yum install dstat
```

To upgrade all packages that have newer versions available, run:

```
bash-4.0$ yum upgrade
```

To upgrade a package with an RPM file, run:

```
bash-4.0$ rpm -Uvh dstat-0.7.1-1.el6.noarch.rpm
```

Uninstalling packages

To uninstall a package using yum, run:

```
bash-4.0$ yum remove dstat
```

Similarly, you can uninstall a package with rpm:

```
bash-4.0$ rpm -e dstat
```

Cleaning the RPM database

You can clean the RPM database, forcing it to refresh package metadata from its sources on next install or upgrade operation.

```
bash-4.0$ yum clean all
```

Querying the RPM database

Occasionally you will want to find out specific information regarding installed packages. The `-q` option to the `rpm` command comes in handy here. Let's take a look at a few examples:

One common task is to see if you have a package installed. The `-qa` option by itself will list ALL installed packages. You can also ask it to list specific packages if they are installed:

```
bash-4.0$ rpm -qa dstat
dstat-0.7.0-1.el6.noarch
```

Now let's say we want to list all of the files installed by a package. The `-ql` option is the one to use:

```
bash-4.0$ rpm -ql dstat
/usr/bin/dstat
/usr/share/doc/dstat-0.7.0
/usr/share/doc/dstat-0.7.0/AUTHORS
/usr/share/doc/dstat-0.7.0/COPYING
/usr/share/doc/dstat-0.7.0/ChangeLog
...
```

We can also do the reverse of the previous operation. If we have a file, and want to known which package it belongs to:

```
bash-4.0$ rpm -qf /usr/bin/dstat
dstat-0.7.0-1.el6.noarch
```

Creating packages

Todo

Mention spec files and roughly how RPMs are put together.

Todo

Then introduce FPM and tell them not to bother with spec files yet.

There are two todos here.

dpkg and APT (Debian, Ubuntu)

In the following examples, we will be using the package `dstat` in our examples. The process however applies to any software you may want to install.

apt provides a wrapper around dpkg, which can be used to search for, and install packages from multiple package repositories.

If your Linux distribution uses dpkg and apt, you can search for packages by running:

```
bash-4.0$ apt-cache search dstat
dstat - versatile resource statistics tool
```

Installing packages

You can install a package through apt, by running:

```
bash-4.0# apt-get install dstat

The following NEW packages will be installed:
  dstat
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/79.3 kB of archives.
After this operation, 351 kB of additional disk space will be used.
Selecting previously unselected package dstat.
(Reading database ... 124189 files and directories currently installed.)
Unpacking dstat (from .../archives/dstat_0.7.2-3_all.deb) ...
Processing triggers for man-db ...
Setting up dstat (0.7.2-3) ...
```

If you have a downloaded DEB file, you can also install the package directly with the `dpkg` command:

```
bash-4.0# dpkg -i dstat_0.7.2-3_all.deb
```

Upgrading packages

dpkg and APT both make it easy to upgrade existing packages, too. Over time, new packages may be added to the apt repositories that are configured on your system, or you may have a newer deb for an already installed package.

To upgrade using apt, when a newer package is available, simply ask apt to install it again:

```
bash-4.0# apt-get install dstat
```

To upgrade a package with an deb file, run:

```
bash-4.0# dpkg -i dstat_0.7.2-3_all.deb
```

Uninstalling packages

To uninstall a package using apt, run:

```
bash-4.0# apt-get remove dstat
```

Similarly, you can uninstall a package with dpkg:

```
bash-4.0# dpkg -r dstat
```

With APT and dpkg, removing a package still leaves behind any configuration files, in case you wish to reinstall the package again later. To fully delete packages and their configuration files, you need to `purge`:

```
bash-4.0# apt-get purge dstat
```

or:

```
bash-4.0# apt-get --purge remove dstat
```

or:

```
bash-4.0# dpkg -P dstat
```

Querying the dpkg database

Occasionally you will want to find out specific information regarding installed packages. The `dpkg-query` command has many options to help. Let's take a look at a few examples:

One common task is to see if you have a package installed. The `-l` option by itself will list ALL installed packages. You can also ask it to list specific packages if they are installed:

```
bash-4.0$ dpkg-query -l dstat
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version      Architecture Description
+++-==============-============-============-=================================
ii  dstat          0.7.2-3      all          versatile resource statistics tool
```

Now let's say we want to list all of the files installed by a package. The `-L` option is the one to use:

```
bash-4.0$ dpkg-query -L dstat
/.
/usr
/usr/bin
/usr/bin/dstat
/usr/share
...
```

We can also do the reverse of the previous operation. If we have a file, and want to know to which package it belongs:

```
bash-4.0$ dpkg-query -S /usr/bin/dstat
dstat: /usr/bin/dstat
```

The Boot Process

The boot process of a modern system involves multiple phases. Here we will discuss each step, how it contributes to the boot process, what can go wrong and things you can do to diagnose problems during booting.

Learning the details of the boot process will give you a strong understanding of how to troubleshoot issues that occur during boot - either at the hardware level or at the operating system level.

You should read this document first, and then power on a computer. Note each of the phases described in this section. Some of them will last many seconds, others will fly by very quickly!

The following components are involved in the boot process. They are each executed in this order:

Power Supply Unit

When the power button is pressed, an electric circuit is closed which causes the power supply unit to perform a self test. In order for the boot process to continue, this self test has to complete successfully. If the power supply cannot confirm the self test, there will usually be no output at all.

Most modern x86 computers, especially those using the ATX standard, will have two main connectors to the motherboard: a 4-pin connector to power the CPU, and a 24-pin connector to power other motherboard components. If the self test passes successfully, the PSU will send a signal to the CPU on the 4-pin connector to indicate that it should power on.

Possible failures:

- If the PSU is unable to perform a good self test, the PSU may be damaged. This could be caused by a blown fuse, or other damage caused by over-/under-current on the power line. Using a UPS or a good surge protector is always recommended.

BIOS and CMOS

At its core, the Basic Input/Output System (*BIOS*) is an integrated circuit located on the computer's motherboard that can be programmed with firmware. This firmware facilitates the boot process so that an operating system can load.

Let's examine each of these in more detail:

- *Firmware* is the software that is programmed into Electrically Erasable Programmable Read-Only Memory (EEPROM). In this case, the firmware facilitates booting an operating system and configuring basic hardware settings.
- An *integrated circuit* (IC) is what you would likely think of as a stereotypical "computer chip" - a thin wafer that is packaged and has metal traces sticking out from it that can be mounted onto a printed circuit board.

Your BIOS is the lowest level interface you'll get to the hardware in your computer. The BIOS also performs the Power-On Self Test, or POST.

Once the CPU has powered up, the first call made is to the BIOS. The first step then taken by the BIOS is to ensure that the minimum required hardware exists:

- CPU
- Memory
- Video card

Once the existence of the hardware has been confirmed, it must be configured.

The BIOS has its own memory storage known as the CMOS (Complimentary Metal Oxide Semiconductor). The CMOS contains all of the settings the BIOS needs to save, such as the memory speed, CPU frequency multiplier, and the location and configuration of the hard drives and other devices.

The BIOS first takes the memory frequency and attempts to set that on the memory controller.

Next the BIOS multiplies the memory frequency by the CPU frequency multiplier. This is the speed at which the CPU is set to run. Sometimes it is possible to "overclock" a CPU, by telling it to run at a higher multiplier than it was designed to, effectively making it run faster. There can be benefits and risks to doing this, including the potential for damaging your CPU.

POST tests

Once the memory and CPU frequencies have been set, the BIOS begins the Power-On Self Test (POST). The POST will perform basic checks on many system components, including:

- Check that the memory is working
- Check that hard drives and other devices are all responding
- Check that the keyboard and mouse are connected (this check can usually be disabled)
- Initialise any additional BIOSes which may be installed (e.g. RAID cards)

Possible failures:

In the event that a POST test fails, the BIOS will normally indicate failure through a series of beeps on the internal computer speaker. The pattern of the beeps indicates which specific test failed. A few beep codes are common across systems:

- One beep: All tests passed successfully (Have you noticed that your computer beeps once when you press the power button? This is why!)
- Three beeps: Often a memory error
- One long, two short beeps: Video card or display problem

Your BIOS manual should document what its specific beep codes mean.

Reading the Partition Table

The next major function of the BIOS is to determine which device to use to start an operating system.

A typical BIOS can read boot information from the devices below, and will boot from the first device that provides a successful response. The order of devices to scan can be set in the BIOS:

- Floppy disks
- CD-ROMs
- USB flash drives
- Hard drives
- A network

We'll cover the first four options here. There's another section that deals with booting over the network.

There are two separate partition table formats: Master Boot Record (MBR) and the GUID Partition Table (GPT). We'll illustrate how both store data about what's on the drive, and how they're used to boot the operating system.

Master Boot Record (the old way)

Once the BIOS has identified which drive it should attempt to boot from, it looks at the first sector on that drive. These sectors should contain the Master Boot Record.

The MBR has two component parts:

- The boot loader information block (448 bytes)
- The partition table (64 bytes)

The boot loader information block is where the first program the computer can run is stored. The partition table stores information about how the drive is logically laid out.

The MBR has been heavily limited in its design, as it can only occupy the first 512 bytes of space on the drive (which is the size of one physical sector). This limits the tasks the boot loader program is able to do. The execution of the boot loader literally starts from the first byte. As the complexity of systems grew, it became necessary to add "chain

boot loading". This allows the MBR to load an another program from elsewhere on the drive into memory. The new program is then executed and continues the boot process.

If you're familiar with Windows, you may have seen drives labelled as "C:" and "D:" - these represent different logical "partitions" on the drive. These represent partitions defined in that 64-byte partition table.

GPT - The GUID Partition Table (the new way)

The design of the IBM-Compatible BIOS is an old design and has limitations in today's world of hardware. To address this, the United Extensible Firmware Interface (UEFI) was created, along with GPT, a new partition format.

There are a few advantages to the GPT format, specifically:

- A Globally-Unique ID that references a partition, rather than a partition number. The MBR only has 64 bytes to store partition information - and each partition definition is 16 bytes. This design allows for unlimited partitions.
- The ability to boot from storage devices that are greater than 2 TBs, due to a larger address space to identify sectors on the disk. The MBR simply had no way to address disks greater than 2 TB.
- A backup copy of the table that can be used in the event that the primary copy is corrupted. This copy is stored at the 'end' of the disk.

There is some compatibility maintained to allow standard PCs that are using an old BIOS to boot from a drive that has a GPT on it.

The Bootloader

The purpose of a bootloader is to load the initial kernel and supporting modules into memory.

There are a few common bootloaders. We'll discuss the GRand Unified Bootloader (GRUB), a bootloader used by many Linux distributions today.

GRUB is a "chain bootloader," meaning it initializes itself in stages. These stages are:

- *Stage 1* - This is the very tiny application that can exist in that first part of the drive. It exists to load the next, larger part of GRUB.
- *Stage 1.5* - Contains the drivers necessary to access the filesystem with stage 2.
- *Stage 2* - This stage loads the menu and configuration options for GRUB.

On an MBR-formatted drive and standard BIOS

These stages must fit in that first 448 bytes of the boot loader information block table. Generally, Stage 1 and Stage 1.5 are small enough to exist in that first 448 bytes. They contain the appropriate logic that allow the loader to read the filesystem that Stage 2 is located on.

On a GPT-formatted drive and UEFI

UEFI motherboards are able to read FAT32 filesystems and execute code. The system firmware looks for an image file that contains the boot code for Stages 1 and 1.5, so that Stage 2 can be managed by the operating system.

The Kernel and the Ramdisk

The kernel is the main component of any operating system. The kernel acts as the lowest-level intermediary between the hardware on your computer and the applications running on your computer. The kernel abstracts away such resource management tasks as memory and processor allocation.

The kernel and other software can access peripherals such as disk drives by way of device drivers.

So what, then, is this Initial RAM Filesystem, or Ramdisk?

You can imagine there are tens of thousands of different devices in the world. Hard drives made with different connectors, video cards made by different manufacturers, network cards with special interfaces. Each of these needs its own device driver to bridge the hardware and software.

For our small and efficient little boot process, trying to keep every possible device driver in the kernel wouldn't work very well.

This lead to the creation of the Initial RAM disk as a way to provide module support to the kernel for the boot process. It allows the kernel to load just enough drivers to read from the filesystem, where it can find other specific device drivers as needed.

With the kernel and ramdisk loaded into memory, we can attempt to access the disk drive and continue booting our Linux system.

OS Kernel and Init

The organizational scheme for determining the load order for system services during the boot process is referred to as an init system. The traditional and still most common init system in Linux is called "System V init".

After the initial ramdisk sets the stage for the kernel to access the hard drive, we now need to execute the first process that will essentially "rule them all" - `/bin/init`.

The init process reads `/etc/inittab` to figure out what script should be run to initialize the system. This is a collection of scripts that vary based on the desired "runlevel" of the system.

Runlevels

Various runlevels have been defined to bring the system up in different states. In general, the following runlevels are consistent in most Linux distributions:

- 0: Halt the system
- 1: Single User Mode
- 6: Reboot the machine

Across distributions there can be various meanings for runlevels 2-5. RedHat-based distributions use runlevel 3 for a multiuser console environment and 5 for a graphical-based environment.

Multiuser vs. Single user runlevels

As the name implies, in some runlevels multiple users can use the machine, versus one user in single user mode. So why does single user mode exist, anyways?

In multiuser runlevels, the system boots as normal. All standard services such as SSH and HTTP daemons load in the order defined in the init system. The network interfaces, if configured, are enabled. It's business as usual if you're booting to a multiuser runlevel.

Conversely, single user mode has the bare minimum of services enabled (notably there is no networking enabled), making it useful for troubleshooting (and not much else).

You will need (or involuntarily find yourself in) single user mode when something breaks: something you configured interferes with the boot process and you need to turn it off, or perhaps a key filesystem is corrupt and you need to run a disk check.

In single user mode, the only available access is via the console, although that need not be limited to physical presence. Remote console access by way of serial consoles and similar devices is a common management tool for data centers.

Getty

Todo

Check this section. I think i've got it down, but I'm not super familiar with this part.

After all the system initialization scripts have run, we're ready to present the user with a prompt to login. The method of doing this is to provide a login prompt on a "TTY" which is short for teletype. This is a holdover from the days that a user running a Unix-based operating system sat at a serially-connected teletype machine. A TTY can be a physical or virtual serial console, such as the various terminals you'd be presented with if you used ALT+F# on the console of a Linux machine.

Getty is often used to continuously spawn `/bin/login`, which reads the username and password of the user and, if authentication succeeds, spawn the user's preferred shell. At this point, the boot and login process has completed.

Useful shell tools

When you work in a unix environment, you will need to make frequent use of the command line tools available to you in order to complete the tasks you have.

The Unix Philosophy

Unix is characterized by its modular design philosophy. Unix programs are small, single purpose tools that can easily be chained together with other Unix programs. This modular approach allows for a much simpler, more flexible system, since you can combine multiple command line programs to solve a unique problem, without having to write a whole new program to get the job done.

Working with your system

ps

`ps` shows the processes currently running on the system. `ps` takes many arguments but some good recipes are `ps aux` to see all processes from a user standpoint, whether they have a tty or not. Also good is `ps -lfU <username>` to see all processes owned by `<username>` in long format. `ps` has output formatting with the `-o` switch and works well with `grep`. `pgrep` combines `ps` and `grep` into one command.

```
$ ps aux | grep vim
opsschool      5424  1.1  0.2 247360 10448 pts/0    Sl+  17:01   0:00 vim shell_tools_101.rst
opsschool      5499  0.0  0.0  13584   936 pts/5    S+   17:01   0:00 grep --color=auto vim
```

Note that the `grep` command also shows up in the process table.

top

`top` shows the top most cpu intensive processes in a table form with system stats summarized at the top. It refreshes every 1 second. By pressing `m` while running, `top` will sort not by processor use but by memory use.

df

`df` looks at all your mounted filesystems and reports on their status. This includes the filesystem type, total size, available space, percent used, and mountpoint. `df -h` will show the sizes in human readable form. `df -h .` will show only the filesystem the current working directory is on, very useful when debugging `nfs` and `autofs` mounts.

```
$ df -h .
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda5        43G   31G  9.6G  77% /
```

du

`du` estimates the size on disk of a file or files. `du -h` returns the usage information in human readable format. If the argument to `du` is a directory, `du` will run and return a new value for each file in that directory, recursing into subdirectories as well. `du -sh` can be run on a directory to prevent this behavior and return the sum of the space on disk of all files under that directory.

```
$ du -sh drawing.svg
24K drawing.svg
4.0K    style.css
20K sitemeter.js
```

find

`find` is for finding files on the system. `find .` recursively walks the entire filesystem tree below the current working directory and prints out each file by its relative path. `find . -name 'opsschool'` again recursively walks the entire filesystem tree, but only prints out files named `opsschool`. `find . -name 'opsschool' -type d` prints out only directories and `-type f` prints out only regular files. The `-name` filter also accepts wildcards. `find . -name '*.html'` will print out all files ending in `.html`. As you become more skilled at Unix/Linux, use the find command often, as it becomes more and more useful with experience.

kill

`kill` is used to send a signal to a process, which is a way of interrupting it. The most common use of this is to stop, or kill, processes. For example, to stop a process with the PID '123':

```
$ kill 123
```

Once the signal is sent the process can decide what to do with it. For instance the above `kill 123` sends a `TERM` signal to the process with PID 123. `TERM` (also known as `SIGTERM`) means "Software Termination Signal". Some processes will notice that signal, finish what they are doing, and shut down. Others will just shut down immediately or ignore it entirely.

There are many more types of signals. (Check out `man signal` for a good list.) For instance, if the above kill command did not succeed in terminating your process, a more heavy-handed option is to send a `KILL` signal to the process:

```
$ kill -KILL 123
```

Every signal has a name and a number. You can reference them by either one. Another way of running `kill -KILL` is:

```
$ kill -9 123
```

Be careful when using the `KILL` signal as it is the one signal that cannot be caught by the process. It will not have a chance to gracefully shut down. This can lead to temporary files not being removed, open files not being closed, or even corruption of database files.

Signals can be used in a wide variety of ways, not just for terminating processes. One interesting use: if your system is running Unicorn processes, you can send a `TTIN` signal to the master process and it will spawn an additional worker. Likewise, you can send a `TTOU` signal and it will remove one of the workers. Another example is Apache HTTPD Web Server which accepts `USR1`, which causes it to close and re-open log files, which is useful when you need to rotate your log files.

For more on signals see *Signals*.

ls

`ls` is used to list the contents of a directory. It's most basic usage would be to list the contents of your shell's current working directory:

```
$ ls
bar  foo
```

You can also pass a directory name to the `ls` command and it will list the contents of that directory:

```
$ ls /usr/local
bin  etc  games  include  lib  libexec  sbin  share  src
```

There are a number of options that can be passed to the ls command to control both what is output and how it's formatted. Files and directories that begin with a '.' are referred to as hidden files. Two of the more useful options are: `-a` and `-l`:

- `ls -a` will list these hidden files and directories.
- `ls -l` outputs what's called a long listing, where various attributes are given in addition to the filenames.

Example of using both:

```
$ ls -la
total 26636
drwx------x. 39 luser luser   4096 Jun 28 01:56 .
drwxr-xr-x.  4 root root     4096 Dec 11  2012 ..
drwxrwxr-x   7 luser luser   4096 May 23 00:40 Applications
-rw-------.  1 luser luser  16902 Jun 28 01:33 .bash_history
-rw-r--r--.  1 luser luser     18 May 10  2012 .bash_logout
-rw-r--r--.  1 luser luser    176 May 10  2012 .bash_profile
```

In a long listing the first field lists the file type, its permissions, and also any special attributes it might have. The very first letter in the first field indicates the file type. Notice directories are indicated by a "d" and regular files are indicated by a "-". After the first field, from left to right the fields are filetypeattributespermissions, links, owner, group, file size, modification date, and file name.

Notice also the files named "." and "..". These are the current directory and the directory up one level, respectively.

lsof

`lsof` lists open files. This command can be very useful in examining what a particular process or user happens to be doing on a system. For each open file information is listed such as the process id that holds the file open, the command that started the process, and the name of the user running the process.

`lsof` doesn't just list regular files. Of particular use is examing what network activity is currently going on. This can be viewed with by issuing `lsof -i`.

man

The `man` command is used to access the `man` pages. A `man` page, short for manual page, is documentation on a particular aspect of your operating system, be it a command, a configuration file, or even functions from a library for a programming language. To access a `man` page, simply type the `man` command followed by the name of the command, file, etc. that you want to view documentation on. In the old days these manuals were hardcopy and on some systems (e.g. Solaris) you will still see evidence of their page layout. There are `man` pages for most, if not all, programs on your system. If you install new software from a package manager, usually new `man` pages will be installed. When man is invoked, it will display the `man` page to you, when you press 'q', the page will disappear.

The man pages are split up into different sections based on their types. For example if you access the `bash man` page, at the very top you will see "BASH(1)", indicating that the `bash` manual is in section 1: general commands. Depending on what you're trying to access, you may have to include a section number when you run man. For example `man printf` will show you the `printf` commands man page. But if instead you were wanting to view documentation on the C printf function you would type `man 3 printf` as section 3 contains documentation on library functions.

The `man` page sections are as follows:

- Section 1: General commands
- Section 2: System calls (functions provided by the kernel)
- Section 3: Library functions (functions within program libraries)
- Section 4: Special files (usually found in /dev)
- Section 5: File formats and conventions (eg /etc/passwd)
- Section 6: Games and screensavers
- Section 7: Miscellaneous
- Section 8: System administration commands

To search through the `man` pages run either `man -k` or `apropos` followed by your search term. This will return a list of man pages who's descriptions match your search term.

The `info` command is another way to find out information about the system and its utilities. Most system administrators are comfortable with the 'basics' of their core command set, but are frequently checking the `man` pages to look up odd flags and functionality.

mount

The `mount` command is used to mount filesystems. For example, mounting an ext4 filesystem that resides on the `/dev/sda2` partition could be done as follows: `mount -t ext4 /dev/sda2 /mnt` In this example the "-t" switch tells the `mount` command that the filesystem type is ext4.

When passed no arguments the `mount` command lists the filesystems that are currently mounted:

```
$ mount
/dev/sda2 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/sda1 on /boot type ext4 (rw)
```

The `mount` command will also consult `/etc/fstab` and if it's able to and use the entries and options it finds there. If an entry for /home exists in /etc/fstab one would be able to simply issue the command `mount /home`. This command would mount whatever partition is found that is associated with the `/home` entry in /etc/fstab, and use any options that happen to be present.

Items to you wish to mount don't necessarily have to be a partition on a disk to be mounted either. Mounting an ISO file, an image of a optical disk, is especially handy: `mount -o loop -t iso9660 /home/luser/installer.iso /mnt/cdrom`

`mount` can also operate on currently mounted filesystems. Of particular use is switching a currently mounted filesystem from read-write to read-only so that a filesystem check can be performed: `mount -o remount,ro /` This particular command tells mount to remount the currently mounted `/` filesystem as read-only.

There are quite a number of options that can be passed to the `mount` command's "-o" switch. Some are filesystem independent, while others depend entirely on the type of filesystem that's being mounted. Further documentation on either can be found in the `man` pages.

stat

Todo

stat command

vmstat

Todo

vmstat command

strace

Todo

strace command

ulimit

Todo

ulimit command

Extracting and manipulating data

A very common pattern in unix is to take some data (a text file, a directory listing, the output from a command) and either extract specific data from it, change some of the data, or both. These tools help you when you do this.

cat

`cat` outputs the contents of a file either to the shell, another file that already exists, or a file that does not yet exist.

Perhaps most frequently, `cat` is used to print the contents of a file to the shell. For example, if file `foo.txt` contains the word 'foo':

```
$ cat /tmp/foo.txt
foo
```

When `cat` is called on multiple files, the output is in the same order as the files. If we have another file `bar.txt` that contains 'bar' and run:

```
$ cat /tmp/foo.txt /home/jdoe/bar.txt
foo bar
```

If you want to combine the contents of the two files:

```
$ cat /tmp/foo.txt /home/jdoe/bar.txt > /home/jdoe/foobar.txt
$ cat /home/jdoe/foobar.txt
foo
bar
```

It is important to note that `foobar.txt` did not exist before running this command. For this particular usage, `cat` can create a file "on the fly".

`cat` can also be used to output the contents of one file to another file.

> **Warning:** You should be careful when using `cat` this way since it will overwrite the contents of the receiving file.
>
> ```
> $ cat /tmp/foo.txt > /home/jdoe/bar.txt
> $ cat /home/jdoe/bar.txt
> foo
> ```

There are many tools that you may use to parse the output of files, and since `cat` can provide a comfortable input method for other tools, it is not always necessary. Read more on Useless Use of cat.

cut

The `cut` utility cuts out selected portions of each line and writes them to the standard output.

As an example, let's take a file `students.txt` that stores a list of student names, ages and email addresses in columns separated by a tab:

```
$ cat students.txt
John  Doe     25      john@example.com
Jack  Smith   26      jack@example.com
Jane  Doe     24      jane@example.com
```

Here, you can see that the first two columns contain the student's name, the third has an age and the fourth, an email address. You can use `cut` to extract just the student's first name and email like this:

```
$ cut -f1,4 students.txt
John john@example.com
Jack jack@example.com
Jane jane@example.com
```

The flag, `-f` is used to select which fields we want to output.

`cut`, by default, uses tab as a delimiter, but we can change that by using the `-d` flag.

Suppose the `students.txt` instead stores data like this:

```
$ cat students.txt
John Doe|25|john@example.com
Jack Smith|26|jack@example.com
Jane Doe|24|jane@example.com
```

Now, if the `|` character is used as a delimiter, the first column would be the student's full name:

```
$ cut -f1 -d| students.txt
John Doe
Jack Smith
Jane Doe
```

If you want to use the space to delimit strings, you would do it like so:

```
$ cut -f1 -d' ' students.txt
John
Jack
Jane
```

`cut` also has some other options. If you have some input with fixed width columns, you can use `-c` to break them apart. For example, to show the login names and times of the currently logged in users:

```
$ who | cut -c 1-9,19-30
mike     Aug  1 23:42
mike     Aug  5 20:58
mike     Aug 22 10:34
mike     Aug  6 19:18
```

You might have to change some of the character positions to make it work on your system.

grep

`grep` matches patterns in files. Its name comes from the `ed` command g/re/p (globally search a regular expression and print the results). `grep` looks for the given pattern in the specified file or files and prints all lines which match.

Grep is an excellent tool for when you know the approximate location of the information you want and can describe its structure using a regular expression.

As you can learn from grep's man page, it takes some options, a pattern, and a file or list of files. The files can be specified by ordinary shell globbing, such as using `*.log` to refer to all files in the current directory whose names end in .log.

Grep's options allow you to customize what type of regular expressions you're using, modify which matches are printed (such as ignoring capitalization or printing only the lines which don't match), and control what's printed as output. This post explains some of the optimizations which allow GNU grep to search through large files so quickly, if you're interested in implementation details.

Intro to Regular Expressions

If you're looking for a specific string, either a word or part of a word, the pattern is just that string. For example, let's say that I vaguely recall someone telling me about OpsSchool on IRC, and I'd like to find the article that they linked me to. I think that it was mentioned in a channel called #cschat:

```
user@opsschool ~$ grep opsschool \#cschat.log
 23:02 < nibalizer> http://www.opsschool.org/en/latest/
```

That's the only place that 'opsschool' was mentioned. Since grep is case-sensitive by default, 'OpsSchool' would not have shown up in that search. To ignore case, use the -i flag:

```
user@opsschool ~$ grep -i opsschool \#cschat.log
 23:02 < nibalizer> http://www.opsschool.org/en/latest/
 15:33 < edunham> hmm, I wonder what I should use as an example in the OpsSchool writeup on grep...
```

That's better. But what if I can't remember whether there was a space in 'ops school'? I could grep twice, once with the space and once without, but that starts to get ugly very fast. The correct solution is to describe the pattern I'm trying to match using a regular expression.

There are a variety of regex tutorials available. The most important thing to remember about regular expressions is that some characters are special, and not taken literally. The special characters are:

Characters	Meaning
$	End of a line
^	Start of a line
[]	Character class
?	The preceding item is optional and matched at most once
*	Zero or more of the preceding item
+	One or more of the preceding item
{}	Match some number of the preceding item
|	Alternation (true if either of the regexes it separates match)
.	Any one character
\	Escape (take the following character literally)

Note that there's almost always more than one way to express a particular pattern. When you're developing a regular expression, it can be helpful to test it on simplified input to see what it catches. To test a regex for various spellings of opsschool, you might put a variety of spellings in a file and then grep it to see which regex catches which spellings.

```
user@opsschool ~$ cat ops.txt
 OpsSchool
 Ops School
 opsschool
 ops school
 ops School
 ops  School
 Ops   school
user@opsschool ~$ grep -i "ops *school" ops.txt
user@opsschool ~$ grep "[oO]ps[ ?][sS]chool" ops.txt
```

Try it yourself. Part of developing a regular expression is clarifying your ideas about exactly what pattern you're looking for. Will you get any false positives? For example, the first example above will return true if there are 2 or more spaces between the words of ops school. It's up to you to decide whether that behavior is correct. If your regular expression catches strings that it shouldn't, be sure to include some of those possible false positives when testing.

Think about which of the regular expressions above you'd prefer. Which is easier to read? Which is better at matching your idea of the correct output?

For more information about regular expressions, try `man 7 regex`, regularexpressions.info, and the Advanced Bash Scripting Guide chapter on the subject.

Single vs. Double quotes in the shell

When grepping for bash variables in scripts, you'll probably want the name of the variable. However, there might be times when you want its value. Below is a quick exercise to explore the difference:

```
user@opsschool ~$ echo "$HOME has my username in it" >> home.txt
user@opsschool ~$ echo '$HOME has a dollar sign in it' >> home.txt
user@opsschool ~$ cat home.txt
 /home/username has my username in it
 $HOME has a dollar sign in it
user@opsschool ~$ grep $HOME home.txt
 /home/username has my username in it
user@opsschool ~$ grep "$HOME" home.txt
 /home/username has my username in it
user@opsschool ~$ grep '$HOME' home.txt
 $HOME has a dollar sign in it
user@opsschool ~$ grep \$HOME home.txt
 $HOME has a dollar sign in it
```

awk

`awk` is a very powerful utility that lets you extract and manipulate data from files.

For example, if you had a file `students.txt` similar to the one above, but with the fields separated by a space:

```
$ cat students.txt
John Doe 25 john@example.com
Jack Smith 26 jack@example.com
Jane Doe 24 jane@example.com
```

You can use awk to extract just the student's first name and email like this:

```
$ awk '{print $1, $4}' students.txt
John john@example.com
Jack jack@example.com
Jane jane@example.com
```

By default, `awk` uses the space character to differentiate between columns. Using this, `$1` and `$4` told `awk` to only show the 1st and 4th columns of the file.

The order in which the columns is specified is important, because `awk` will print them out to the screen in exactly that order. So if you wanted to print the email column before the first name, here's how you would do it:

```
$ awk '{print $4, $1}' students.txt
john@example.com John
jack@example.com Jack
jane@example.com Jane
```

You can also specify a custom delimiter for `awk` and override the default one (the space character) by using the `-F` option. Suppose the `students.txt` instead stored data like this:

```
$ cat students.txt
John Doe - 25 - john@example.com
Jack Smith - 26 - jack@example.com
Jane Doe - 24 - jane@example.com
```

Now, if the - character is used as a delimiter, the first column would be the student's full name:

```
$ awk -F '-' '{print $1}' students.txt
John Doe
Jack Smith
Jane Doe
```

Using this same logic, the second column would be the student's age, and the third their email address.

```
$ awk -F '-' '{print $1, $3}' students.txt
John Doe john@example.com
Jack Smith jack@example.com
Jane Doe jane@example.com
```

`awk` functionality doesn't stop at printing specific columns out to the screen; you can use it to:

- extract a specific row from the file using the `NR` command

```
$ awk 'NR==2' students.txt
Jack Smith - 26 - jack@example.com
```

Note: The `-F` option was not used here since rows are being manipulated, and the `-F` option specifies a delimiter for column manipulation

---

- extract lines longer than a specific length by using the `length($0)` command

```
$ awk 'length($0) > 30' students.txt
John Doe - 25 - john@example.com
Jack Smith - 26 - jack@example.com
Jane Doe - 24 - jane@example.com

$ awk 'length($0) > 32' students.txt
Jack Smith - 26 - jack@example.com
```

- find the average of numbers in a column

```
$ awk -F '-' '{sum+=$2} END {print "Average age = ",sum/NR}' students.txt
Average age =  25
```

In the last example, with the average age, `{sum+=$2}` tells awk to take each value in the second column and add it to the existing value of the variable `sum`. It's important to note here that the variable `sum` didn't have to be declared or initialised anywhere, `awk` creates it on-the-fly. The `END` pattern tells `awk` what to do after all lines in the file have been processed. In our case, that involves printing out the average age of all students. To get the average age, the sum of all ages (stored in variable `sum`) was divided by the total number of lines in the file, represented by `NR`.

In addition to the `END` pattern, `awk` also provides a `BEGIN` pattern, which describes an action that needs to be taken before the first line of the file is processed.

For example:

```
$ awk 'BEGIN {print "This is the second line of the file"} NR==2' students.txt
This is the second line of the file
Jack Smith - 26 - jack@example.com
```

sed

Todo

Only talk about replacing text for now? It's the most common / needed piece of sed at this level.

---

sort

`sort` can be used to sort lines of text.

For example, if you had a file `coffee.txt` that listed different types of coffee drinks:

```
$ cat coffee.txt
Mocha
Cappuccino
Espresso
Americano
```

Running `sort` would sort these in alphabetical order:

```
$ sort coffee.txt
Americano
Cappuccino
Espresso
Mocha
```

You can also reverse the order by passing in `-r` to the command:

```
$ sort -r coffee.txt
Mocha
Espresso
Cappuccino
Americano
```

All very easy so far. But, say we have another file `orders.txt` that is a list of how many of each drink has been bought in a day:

```
$ cat orders.txt
100 Mocha
25 Cappuccino
63 Espresso
1003 Americano
```

What happens when we run `sort` on this file?

```
$ sort orders.txt
1003 Americano
100 Mocha
25 Cappuccino
63 Espresso
```

This isn't what we want at all. Luckily, `sort` has some more flags, `-n` is what we want here:

```
$ sort -n orders.txt
25 Cappuccino
63 Espresso
100 Mocha
1003 Americano
```

What if we want to sort the new list by name? We will have to sort by the second column, not the first one. `sort` assumes that columns are space separated by default. `sort` has the flag `-k` that let us specify what key we want to use.

```
$ sort -k2 orders.txt
1003 Americano
25 Cappuccino
```

```
63 Espresso
100 Mocha
```

There are many more flags available, `man sort` will show you them all. There is probably already something there for whatever you can throw at it.

Crontab

Laziness is considered a virtue in system administration, meaning you should figure out ways to avoid having to perform the same task regularly by hand. **Cron** to the rescue!

> **cron** is the time-based job scheduler in Unix-like computer operating systems. **cron** enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email.
>
> —cron - From Wikipedia, the free encyclopedia

cron allows you to run routine jobs on a Unix-based system automatically at a specific future time or at regular intervals rather than running such jobs manually. Even knowing just the basics of cron is a huge win, as this tool increases productivity, saves time and is not prone to human error, i.e. forgetting to run a job.

What is a "crontab"?

"crontabs" (or *cron tables*) are the way that a user schedules a job to run under **cron**. They're composed of the following parts:

- *cron expression*: A string that describes the time or times when the command should be run
- *user* (optional): If the crontab is specified in a centralized location instead of the user's personal crontab file, this field contains the username that the command should be run as. This allows **cron** to run a command as you, even if you're not logged in at the time. This field isn't used in a user's personal crontab file.
- *command*: The rest of the line is interpreted as the command that **cron** should run at the specified time.

"crontab" may also refer to a file containing any number of these cron tables, one per line.

crontab files are commonly found in a couple of different places:

- `/etc/cron.d`, `/etc/cron.daily`, `/etc/cron.hourly`, and so on: System crontabs, or crontabs installed by a software package
- `/var/spool/cron/<username>`: Users' crontabs; these are created and managed with the `crontab` command.

The `crontab` command

How do I list jobs scheduled to run with cron?

The easiest way to see jobs running regularly on the system (aka "cron jobs") is to type:

```
user@opsschool ~$ crontab -l
```

in the shell. This will show all the cron jobs scheduled to run as the current user. For example, if you are logged in as `jdoe` and there are no cron jobs running the output will be something like:

```
jdoe@opsschool ~$ crontab: no crontab for jdoe
```

If there are jobs scheduled to run, there will be a list of lines that looks something like this:

```
05 12 * * 0 /home/jdoe /home/jdoe/jobs/copy-to-partition
```

How do I schedule a job with cron?

The way to schedule a new job is to type:

```
user@opsschool ~$ crontab -e
```

in the shell. This will open an editor with your personal "crontab" file (or an empty file if you don't have any cron jobs already scheduled).

Then add a line to the file that looks like this:

```
0 * * * * date
```

and save and close the file.

The cron job you just created will run the command `date` at the top of every hour (e.g. at 10:00, 11:00, 12:00, etc.). Depending on how your system is configured, it may begin to email you the output of that command each time it's run, or it may be saved to a log file somewhere.

A more in-depth example

Let's look again at the example from before:

```
05 12 * * 0 /home/jdoe /home/jdoe/jobs/copy-to-partition
```

Let's dissect this a bit, as it will help when you're creating your own cron jobs. What is this output telling you? It is helpful to know what the fields of a "crontab" are. Here's a table with the order of fields, and their values:

MINUTE	HOUR	DAYOFMONTH	MONTH	DAYOFWEEK	COMMAND
0-59	0-23	1-31	1-12	0-6	filepath/command

Note: Order matters, and note that the first element is 0 for the minute, hour, and day of the week fields, while the day of the month and month fields begin at 1.

Knowing this, we can see that this "crontab" means:

> At 12:05 every Sunday, every month, regardless of the day of the month, run the command `copy-to-partition` in the `/home/jdoe/jobs` directory.

Caveat: entries in your crontab must be one long line; if you try to split it up (with the linebreak `\` character for example) you will get an error!

Let's take another example and create a cron job that checks disk space available every minute, every hour, every day of the month, every month, for every day of the week, and outputs it to a file named :file:`disk_space.txt`.

```
* * * * * df -h > disk_space.txt
```

This would get us what we wanted (`df -h` is the unix command for checking free disk space).

Field values can also be ranges. Let's say you want to edit this job to run the same command (`df -h`), but instead of running every minute, you only want the job to run it in the first 5 minutes of every hour, every day of the month, every month, for every day of the week.

Running `crontab -e` again and changing the line to:

```
0-5 * * * * df -h > disk_space.txt
```

will get you what you want.

How do I remove a crontab?

Lastly, if you want to remove the command, again type `crontab -e`, and then delete the line with that job in it from the file in your editor.

To remove all cron jobs for the current user, type:

```
crontab -r
```

What are some common "cron expressions"?

The "cron expression" syntax can be confusing to understand. Here are some common expressions to get you started.

- `* * * * *`: every minute
- `0 * * * *`: every hour, on the hour
- `0 0 * * *`: every day at midnight server time
- `0 0 * * 0`: every Sunday at midnight server time
- `*/10 * * * *`: every ten minutes
- `0 */4 * * *`: every four hours, on the hour

Advanced "crontab"

How do "cron expressions" work?

Here is the standard "cron expression" cheat sheet [2]:

```
# .---------------- minute (0 - 59)
# |   .------------- hour (0 - 23)
# |   |   .---------- day of month (1 - 31)
# |   |   |   .------- month (1 - 12) OR jan,feb,mar,apr ...
# |   |   |   |  .----- day of week (0 - 7) (Sunday=0 or 7)  OR sun,mon,tue,wed,thu,fri,sat
# |   |   |   |  |
# *   *   *   *  *  command to be executed
```

Put this template at the top of your crontab file so it'll be easy to remember what the fields do.

Notes on composing good "cron expressions"

- If you want to run something every N hours, be sure to specify a minute expression (the first number) also; otherwise, the command will be run once a minute for the entire hour.
- Commands run by **cron** won't have all of the configuration and environment variables that come from your shell initialization files (like `.bashrc` or `.zshrc` or such). In particular, make sure to specify the full path to your program if it's not in a commonly-used location like `/usr/bin`.

[2] "Examples" in cron - Wikipedia, a free encyclopedia

- If you have problems with the syntax, or something isn't working properly, there are websites [3] that will translate crontab to English, and vice versa. [4]

Modify a specific user's crontab

The `crontab` command can be used to view or modify a specific user's crontab file, instead of the current user's crontab file. For instance, if you are logged in as `jdoe` and you want to edit `jsmith`'s crontab (and you have the permissions to do so), type the following in a shell:

```
jdoe@opsschool ~$ crontab -e -u jsmith
```

This option also combines with the other options we looked at before (`-l` for listing and `-r` for removing a user's crontab file).

Modifying crontab parameters

With some cron implementations [5], you can add shell environment variables to the top of a crontab file that affect all of the commands run by those crontabs. For example, you could modify the `PATH`, `MAILTO`, `HOME`, or any other variable.

Cron Output

Output from cron commands can be emailed to the owner of the process using a local SMTP server, like ssmtp or postfix. If you'd like the output to be sent to an email different than your `root@yourhost` address use the `MAILTO="your@email"` expression before the cron command. An email will be sent containing the output from STDOUT/STDERR every time the command executes. Unfortunately, the email gives no indication of the return status of the process (success or failure) and often leads to a cluttered inbox.

Note: If your crontab is misconfigured you won't receive an email at all. Finding such failures (the lack of an email in your inbox) is hard to detect. To save time and avoid these *silent failures*, it is better to use a service that will notify you only on cron failures. [6]

Footnotes

Unix fundamentals 201

Kernel tuning

An introduction to `/proc`, `sysctl`, `mdb`, `ndd`

[3] Crontab to plain English
[4] English to crontab
[5] Where can I set environment variables that crontab will use?
[6] Receive notifications when cron commands fail

Signals

What Are Signals?

Signals are a form of inter-process communication (IPC) in Posix compliant systems like Linux, UNIX, and UNIX-like operating systems. They are an asynchronous mechanism the kernel uses to communicate with a process or a thread. Signals are always delivered by the kernel but can be initiated by the kernel, the process itself, or even another process. Signals are often referred to by their name, or their numeric integer value. For example, the kill signal is known as SIGKILL, or 9. There are many different signals that can be sent, although the signals in which users are generally most interested are SIGTERM and SIGKILL. The default signal sent is SIGTERM.

How Do Signals Work?

When a process receives a signal, what happens depends on the program. In the simplest case the process will just apply the default signal handlers defined in the C library. In most cases this will cause the program to terminate, suspend, and occasionally it will ignore the signal.

In some cases, programs have custom signal handlers. Programs that have custom signal handlers will behave differently when they receive a signal. For example many service daemons will reload their configurations when they receive the SIGHUP signal; the default action for SIGHUP is for the program to terminate.

In either case, when the kernel sends a signal to a process, the process will execute its signal handler immediately. Processes can enter states in which they are executing atomic instructions(like file locks), and will not process the signal until the instruction is completed.

How To Send a Signal to a Program

There are three ways to send a signal to another process. The simplest way is the execute the "kill" command with a signal specified. For example you can use the kill command from the shell to send the interrupt signal like so:

```
kill -SIGINT <PID>
```

You can write a simple program executing the kill system call. A basic example is below:

```
int signal_pid(int pid) {
  int rvalue;
  rvalue = kill(pid, SIGINT);
  return rvalue;
}
```

Lastly you can send signals from the keyboard in an interactive terminal. Ctrl-C will send SIGINT, and Ctrl-Z send SIGTSTP.

List Of Posix 1990 Signals

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal (signal sent by default by the kill command when not specified)
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

Syscalls

What are syscalls?

Applications require actions from the kernel to perform several tasks it cannot handle by itself. Accessing files, connecting to sockets, creating processes are examples of tasks handled by the kernel. System calls are the interface for an application to interact with the kernel.

Using `strace`

The `strace` tool can be used to monitor interaction between a process and the kernel. It displays system calls initiated by the process, and signals received by the process.

Consider the simple script:

```
#!/bin/bash
echo "one line of text" > my_input.txt
while read line
do
        echo $line > my_output.txt
done < my_input.txt
rm my_input.txt
```

Exec

When invoking this script using `strace -omy_strace.log -ff ./strace_sample.sh`, the output of every (forked) process is written to my_strace.log.<pid>. Let's examine the logfile of our parent process (parts of the output are omitted, let's focus on the relevant stuff) :

```
execve("./strace_sample.sh", ["./strace_sample.sh"], [/* 17 vars */]) = 0
```

`execve` executes the file "strace_sample.sh" with arguments "./strace_sample.sh" (by convention the first should be the filename to be executed) and 17 environmental variables. At this point there are already 3 file descriptors open: stdin (fd 0), stdout (fd 1), stderr (fd2).

I/O

```
open("./strace_sample.sh", O_RDONLY|O_LARGEFILE) = 3
<..>
read(3, "#!/bin/bash\necho \"one line of te"..., 80) = 80
<..>
dup2(3, 255)                            = 255
close(3)                                = 0
<..>
read(255, "#!/bin/bash\necho \"one line of te"..., 126) = 126
<..>
```

`open` prepares the file for reading in file descriptor 3 (result of the syscall `open`). Next, `read` attempts to read the next 80 bytes from fd (file descriptor) 3.

At this point the interpreter line of the script is read, indicating that the /bin/bash binary needs to be invoked to process the script. `dup2` copies the fd 3 to the (new) fd 255. This is a bash-specific operation, don't mind too much; bash keeps the original file in fd 255 (last of the process' private fd) to free up low-numbered fd's.

Now that our original file is open in fd 255, fd 3 is not needed anymore and can be closed by `close`. At last the next 126 bytes (rest of the file) are read and stored in the buffer, now we can start to process the commands in the script file.

```
open("my_input.txt", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE, 0666) = 3
<..>
fcntl64(1, F_DUPFD, 10)                 = 10
<..>
dup2(3, 1)                              = 1
close(3)                                = 0
write(1, "one line of text\n", 17)      = 17
dup2(10, 1)                             = 1
<..>
close(10)                               = 0
```

Now we're going to write to a file using redirection. First, `open` opens/creates the "my_input.txt" file for writing in fd 3. Then, `fcntl64` uses F_DUPFD (10) command to get the next available fd numbered >=10 and copy fd 1 (initially opened for stdout) to the new fd 10. This saves the original content of fd 1 (stdout), so it can be restored later.

Next `dup2` copies fd 3 to fd 1 so that writing to fd 1 ends up in the file opened by fd 3. The `write` call then writes the next 17 bytes from the buffer to fd 1. Afterwards the redirection is reverted, by copying fd 10 (original value for fd 1) back to fd 1. Now fd 1 points to stdout as in our initial situation. The fd's which are not needed any longer are closed.

```
open("my_input.txt", O_RDONLY|O_LARGEFILE) = 3
<..>
fcntl64(0, F_DUPFD, 10)                 = 10
<..>
dup2(3, 0)                              = 0
close(3)                                = 0
<..>
read(0, "one line of text\n", 128)      = 17
open("my_output.txt", O_WRONLY|O_CREAT|O_APPEND|O_LARGEFILE, 0666) = 3
```

```
<..>
fcntl64(1, F_DUPFD, 10)                 = 11
<..>
dup2(3, 1)                              = 1
close(3)                                = 0
write(1, "one line of text\n", 17)      = 17
dup2(11, 1)                             = 1
<..>
close(11)                               = 0
read(0, "", 128)                        = 0
dup2(10, 0)                             = 0
<..>
close(10)
```

This time we'll both read and write using redirection.

Again, `open` prepares "my_input.txt" for reading in fd 3. This time, save fd 0 (by default stdin) to fd 10. `dup2` copies fd 3 to fd 0 (redirecting "my_input.txt" to stdin) and close fd 3. Next, read the next 128 bytes from fd 0 ("my_input.txt") and save to the buffer.

Then, "my_output.txt" is opened (created) for writing in fd 3. Now `fcntl64` uses F_DUPFD (10) to get the next available fd >= 10 (which at this point is 11 as fd 10 is already open) and copy fd 1 to it. The fd 3 is copied to fd 1 using `dup2`.

Finally, write 17 bytes from the buffer to fd 1, which at this moment points to "my_output.txt". The redirection is reverted by copying fd 11 to fd 1 with `dup2`, and fd 11 can be closed.

A next attempt to `read` from fd 0 results in 0 bytes read, indicating the end of file is reached. The redirection is reverted by copying fd 10 to fd 0 and closing fd 10.

`exec`, `open`, `close`, `read` and `write` are handled. Let's look at creating child processes and removing files.

Child processes

```
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xb6f50068)
```

The parent process uses `clone` to create a child process to execute the `rm` command. The logging of this child process is logged in the second my_strace.log.<pid> file, where in this example pid=3482, but this varies on each run.

```
execve("/bin/rm", ["rm", "my_input.txt"], [/* 17 vars */]) = 0
<..>
```

Unlink

```
newfstatat(AT_FDCWD, "my_input.txt", {st_mode=S_IFREG|0644, st_size=17, ...}, AT_SYMLINK_NOFOLLOW) =
faccessat(AT_FDCWD, "my_input.txt", W_OK) = 0
unlinkat(AT_FDCWD, "my_input.txt", 0)   = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

The `rm` command is executed using `execve`, with arguments "rm" (as per convention this is the filename to be executed) and "my_input.txt". `newfstatat` gets the file status and `faccessat` check the file permissions of the file.

Finally, `unlinkat` removes the file's name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse. As a last step for this process the 3 standard fd's are closed, and `exit_group` exits all possible threads in the process.

Again in the parent's logfile, the interaction with the child process is logged.

```
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 3481
<..>
--- SIGCHLD (Child exited) @ 0 (0) ---
wait4(-1, 0xbec6bf39, WNOHANG, NULL)    = -1 ECHILD (No child processes)
<..>
read(255, "", 142)                      = 0
exit_group(0)                           = ?
```

`wait4` keeps the parent process waiting for the child process to terminate. Once terminated, wait releases and the parent process continues. A final `read` is attempted on fd 255, but as the end of the file is reached, this returns 0. The last exit_group exits all open threads in the process.

Output tuning

By default the `strace` produces all system calls performed by the executable. As this can be overwhelming, the -e switch can be used to look for specific system calls. When examining this with `-eopen` the following is output is given:

```
strace -eopen ls
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib64/librt.so.1", O_RDONLY)     = 3
open("/lib64/libacl.so.1", O_RDONLY)    = 3
open("/lib64/libselinux.so.1", O_RDONLY) = 3
open("/lib64/libc.so.6", O_RDONLY)      = 3
open("/lib64/libpthread.so.0", O_RDONLY) = 3
open("/lib64/libattr.so.1", O_RDONLY)   = 3
open("/lib64/libdl.so.2", O_RDONLY)     = 3
open("/lib64/libsepol.so.1", O_RDONLY)  = 3
open("/etc/selinux/config", O_RDONLY)   = 3
open("/proc/mounts", O_RDONLY)          = 3
open(".", O_RDONLY|O_NONBLOCK|O_DIRECTORY) = 3
```

This can come in handy to troubleshoot specific system calls. A list of available syscalls can be seen in `man syscalls`. For more details on a syscall, look it up in the man page. Some syscalls have several variant and might be referenced in strace output with different names; try to look them up without certain prefixes to find the relevant man pages. For performance measuring the `-T` and `-c` flags are usefull:

- `-T` Show the time spent in system calls. This records the time difference between the beginning and the end of each system call.
- `-c` Count time, calls, and errors for each system call and report a summary on program exit.

Booting over the network

Typically, a machine will boot off of some type of local device. However, there are some cases where it makes more sense to boot a machine off a remote host. For example:

- You have a large number of machines to install, and carrying a CD/USB drive to each of them is time consuming.
- You have a machine where the hard drive is failing, and you need to run some diagnostic tools
- You have a machine in a remote location, but you need to reinstall the operating system

In situations like this, it's possible to boot a machine entirely over the network. This relies on a little piece of firmware built into pretty much every modern NIC. This software is called Preboot eXecution Environment (PXE).

The network boot process works like this:

- The machine boots off the NIC, which has PXE firmware built in
- The PXE firmware initializes the NIC and sends out a DHCP request
- The DHCP response contains instructions to contact another server and download a file
- The PXE firmware downloads the file via TFTP, and begins executing it
- From here, many different things can happen depending on the type of software downloaded

TFTP

Trivial File Transfer Protocol (TFTP) is a very basic file transfer protocol that's used for bootstrapping the PXE firmware into something more useful. TFTP is implemented as a UDP service, typically running on port 69. TFTP does not support any type of authentication.

Given the low complexity of the TFTP protocol, servers are available for virtually every operating system. Configuration is typically just pointing the server at a directory full of files to serve up. Unless you have very specific reasons to do so (and fully understand the implications), you should make sure that your TFTP server is setup in read only mode.

PXE

The PXE stack by itself is very limited. It's designed to just be able to retrieve and execute a file, and by itself is not terribly useful. The two most popular software packages used with PXE are iPXE [7] and PXELINUX [8]. Of these, PXELINUX is the older one, and is a variant of SysLinux. SysLinux is most commonly used as the initial menu you see when you boot a Linux CD. iPXE is newer, and supports booting over many different protocols (HTTP, iSCSI, various SAN types, among others).

Basic Setup Process

For both iPXE and SysLinux you will need both a DHCP and TFTP server installed. Your operating system will likely have a tftp server package available. Don't worry too much about the exact one you use, there are only minor differences between them. After this is installed, you will need to locate the directory that it is serving files from. For the rest of this document we will refer to this directory as the 'tftpboot' directory.

For the DHCP server, we'll be using the ISC DHCPD server. This is available for many platforms, and has a large amount of documentation available. Aside from setting up a DHCP range, you'd need to add the following line to your dhcpd.conf file:

```
# Any clients that are PXE booting should attempt to retrieve files from this server
next-server <tftp server IP>;
```

That is the bare minimum needed for both types of PXE software we're going to set up.

iPXE Setup

Start by downloading iPXE [9]. Make sure you save this to your tftpboot directory. Next, add the following to your dhcpd.conf file:

[7] http://ipxe.org
[8] http://www.syslinux.org/wiki/index.php/PXELINUX
[9] http://boot.ipxe.org/undionly.kpxe

```
if exists user-class and option user-class = "iPXE" {
    filename "chainconfig.ipxe";
} else {
    filename "undionly.kpxe";
}
```

This will cause the DHCP server to first tell clients to download iPXE. Once iPXE starts up and does another DHCP request, it will be told the actual location of the configuration file to download. Without this, we would end up with a continuous loop of iPXE downloading itself. Create a chainconfig.ipxe file in your tftpboot directory with the following:

```
#!ipxe

:start
# Define a generic title for the menu
menu iPXE boot menu
# And now we define some options for our menu
item localboot      Boot from local disk
item centos6_x64    Install CentOS 6 x64
item centos6_i386   Install CentOS 6 i386

# Now we prompt the user to choose what they want to do.  If they don't select an option
# within 60s we default to booting from the local disk.  If they somehow choose an invalid
# option, we drop them to the iPXE shell for further debugging
choose --default localboot 60000 bootoption && goto ${bootoption} ||
echo Invalid option selected!
shell

# Here we define our two network installation options.  iPXE supports basic variable operations,
# so we can reuse much the same code for booting the two different architectures.  We'll define
# some options specifying the version and architecture we want, then jump to some common installer
# code
:centos6_x64
set centos-version 6
set arch x86_64
goto centos_installer

:centos6_i386
set centos-version 6
set arch i386
goto centos_installer

# This demonstrates some of the power of iPXE.  We make use of variables to prevent config duplicatio
# and we load the installer files directly off the CentOS mirror.  There's no need to copy everythin
# to a local TFTP server.  We also fallback to a shell if the boot fails so any issues can be debugge
:centos_installer
kernel http://mirror.centos.org/centos-${centos-version}/${centos-version}/os/${arch}/images/pxeboot/
initrd http://mirror.centos.org/centos-${centos-version}/${centos-version}/os/${arch}/images/pxeboot/
boot ||
goto shell

# This just exits iPXE entirely, and allows the rest of the boot process to proceed
:localboot
exit
```

PXELINUX setup

Start by downloading SysLinux [10]. Copy a few files from the archive into your tftpboot directory:

- com32/menu/vesamenu.c32
- core/pxelinux.0

Next, we'll need to create the menu config file. Create the file tftpboot/pxelinux.cfg/default:

```
# We want to load the vesamenu module, which generates GUI menus
UI vesamenu.c32
# Don't display a prompt for the user to type in a boot option (they'll be selecting one instead)
PROMPT 0
# Our default option is to boot from the local drive
DEFAULT localboot
# Wait 60s before booting to the default option
TIMEOUT 600

# Define a title for our menu
MENU TITLE SysLinux Boot Menu

# This is the internal name for this option.
LABEL centos6_x64
    # And a human readable description
    MENU LABEL Install CentOS 6 x64
    # This is the kernel file to download (via TFTP) and boot
    KERNEL centos/6/x64/vmlinuz
    # And any command line options to pass to the kernel
    APPEND initrd=centos/6/x64/initrd.img ramdisk_size=65535 noipv6 network

# Now for the i386 version.  As SysLinux doesn't support variables, we end up duplicating
# the majority of the config from the x64 version
LABEL centos6_i386
    MENU LABEL Install CentOS 6 i386
    KERNEL centos/6/i386/vmlinuz
    APPEND initrd=centos/6/i386/initrd.img ramdisk_size=65535 noipv6 network

LABEL localboot
    # Proceed through the rest of the normal boot process
    LOCALBOOT 0
```

Since PXELINUX doesn't support HTTP, we'll need to download the CentOS installer images to the tftpboot directory. Create two directories and download the initrd.img and vmlinuz files to them:

- Directory: tftpboot/centos/6/x64/ Files: http://mirror.centos.org/centos-/6/os/x86_64/images/pxeboot/
- Directory: tftpboot/centos/6/i386/ Files: http://mirror.centos.org/centos-6/6/os/i386/images/pxeboot/

References

`/bin/init` and its descendants

`init`

systemd

[10] https://www.kernel.org/pub/linux/utils/boot/syslinux/

upstart

Upstart is a project that aims to replace to old init system by providing one standard way of starting and stopping daemons with the correct environment. A second goal is to speed up a computer's boot time. It achieves this by removing the slow init shell scripts, and also by parallelizing as much of the startup as possible. Where old init daemons start daemons in successive order, upstart issues "events" on which "jobs" can listen.

Such an event can be e.g.: `filesystem` - indicates that the system has mounted all its filesystems and we can proceed to start any jobs that would depend on a filesystem. Each job then becomes an event of its own, upon which others can depend. These events can be broken up into stages: `starting(7)`, and `started(7)`; and `stopping(7)`, and `stopped(7)` respectively.

A good starting point for learning how different jobs on a system are interconnected is `initctl(8)`'s `show-config` command:

```
igalic@tynix ~ % initctl show-config
avahi-daemon
  start on (filesystem and started dbus)
  stop on stopping dbus
cgroup-lite
  start on mounted MOUNTPOINT=/sys
elasticsearch
  start on (filesystem or runlevel [2345])
  stop on runlevel [!2345]
mountall-net
  start on net-device-up
...
```

This snippet reveals that upstart will `stop` the `avahi-daemon` at the same time as dbus. Unlike many other daemons that depend on the whole filesystem, upstart will `start cgroup-lite` as soon as the `/sys` filesystem is mounted.

Upstart is also able to "supervise" programs: that is, to restart a program after it crashed, or was killed. To achieve this, upstart needs to "follow" a programs progression. It uses the `ptrace(2)` system call to do so. However, following a daemons forks is *complex*, because not all daemons are written alike. The upstart documentation recommends to avoid this whenever possible and force a to remain in the foreground. That makes upstart's job a lot easier.

Finally, upstart can also switch to a predefined user (and group) before starting the program. Unlike systemd and SMF, however, it cannot drop to a limited set of `capabilities(7)` before doing so.

Putting it all together in an example:

```
# httpd-examplecom - Apache HTTP Server for example.com
#

# description     "Apache HTTP Server for example.com"

start on filesystems
stop on runlevel [06]

respawn
respawn limit 5 30

setuid examplecom
setgid examplecom
console log

script
    exec /opt/httpd/bin/httpd -f /etc/httpds/example.com/httpd.conf -DNO_DETACH -k start
end script
```

In this example we define an upstart job for serving `example.com` from the Apache HTTP Server. We switch to the user/group `examplecom` and start `httpd` in the foreground, by passing the option `-DNO_DETACH`.

To activate this job, we simply place it in a file in `/etc/init/`, e.g. `/etc/init/httpd-examplecom.conf`. We can then start/stop the job by issuing:

```
% sudo start httpd-examplecom
```

Note that this job definition alone already will guarantee that the system will start the job on reboot. If this is not what we want, we can add the stanza `manual` to the job definition.

SMF

daemontools

Looking at system metrics

vmstat

iostat

systat

dstat

sar

MS Windows fundamentals 101

Text Editing 101

As a sysadmin, and *especially* as a Linux/Unix sysadmin, you're going to have to get used to not only making your way around the command line, but also editing text files. Both of these tasks require you to get familiarized and comfortable with the two most common text editors in this space: `vi` and `emacs`

Both of these editors, `vi` and `emacs`, are command-line utilities that you run from a terminal window. This means that you can also run them on remote machines over an `ssh` connection. While there are graphical versions of these tools available (`gvim` and `xemacs`), many administrators prefer the text-only versions, as it creates a uniform experience whether they are working on their own local machine, or a remote one being accessed over the network. Additionally, avoiding use of the mouse means your hands stay on the home row of the keyboard, meaning you work faster and with less fatigue.

A little history

Back when UNIX was just starting to appear on computers that had actual input/output systems (e.g. teletype), interaction with the computer was strictly a text-based affair. The computer would send some data in the form of ASCII* characters over a slow telephone link. The user would then hit some keys on the keyboard, which would send back more ASCII characters to the computer, again over the slow telephone link.

Because communication was so slow, UNIX programmers and users wrote their internal tools to be as efficient as possible. Early editors, such as `ed`, would only let you see one line of text in the file at a time, since the commonly used interface at the time, a teletype, printed on paper and thus there was no concept of "redrawing" or even of

a "screen". These early tools later evolved into the more advanced editors like `vi` and `emacs` in use today. But these tools retain their UNIX heritage of lean efficiency. Commands are sent usually in single characters, there's no "toolbar", and the editor provides powerful tools to quickly shuttle the cursor around the file quickly, without having to display unrelated text/context in the file.

* Note: you can also run `man ascii` to quickly get an ASCII table

`vi` basics

`vi` was created as an extension to `ex` (a successor to `ed`). When "modern" terminals (e.g. electronic displays, not teletypes) became common, it made sense to use a "visual" editing mode, where you could see the context of the line you were editing. From `ed`, you would enter "visual" mode by running `:vi` – thus VI was born.

Since then, `vi` has been written and re-written many, many times. But inside, you will still find good ol' `ed`. In `Text Editing 201`, you will see how you can still access the very powerful `ed` features by using the colon (`:`) command.

There are many versions of `vi` out there, but the most popular is `vim` (VI iMproved). Virtually every Linux distribution out there provides `vim` either by default, or as an easily installable package. `vim` extends the feature set provided by the POSIX-compliant `vi` utility in many ways. In the examples below, features that are only present in `vim` will be noted.

A Note About Vimtutor

Vim, the modern, extended version of `vi`, includes a tutorial of its own. On a system with Vim installed, you should be able to run `vimtutor` to get an interactive session that walks you through most of the same content you'll get on this page. You don't have to use `vimtutor` if you don't want to, though – just know that it is there if you wish to try it out.

`vi` Modes

The first thing to understand about how `vi` works is that it operates in different modes.

Normal mode is `vi`'s default mode and should be the mode you are in most of the time. When in normal mode, the keys you type are interpreted by `vi` as commands, not as text you wish to put into the file. For example, moving the cursor around the screen is a series of commands, as is deleting text.

Insert mode is when you are actually typing in text. The keys you type while in insert mode are simply dumped into the buffer (`vi`'s name for the file you are editing). You exit insert mode by pressing the `Esc` key.

Command-line mode is when you are typing a command on the cmd line at the bottom of the window. This is for the Ex commands, ":", the pattern search commands, "?" and "/", and the filter command, "!".

Hello, world!

Before learning anything else, you'll need to know how to do two basic operations, which are best described by working through these steps.

Start `vi` and edit a new file:

```
vi helloworld.txt
```

The editor will start up in normal mode. The first thing we want to do is add some text to the file. Press `i` to enter "insert" mode. You will notice that the editor begins to let you start typing text into the file. Enter some text into the file:

```
Hello, World!
```

Now you want to save the contents, so follow these steps:

1. Press `Esc` to exit insert mode and enter command mode
2. Press `:` and then `w`, then `Enter`. This will write out to the file.
3. The editor remains in command mode, waiting for your next command.

Now you want to exit the editor:

1. Press `Esc` to exit insert mode (hitting `Esc` while in command mode does nothing, so it's safe to just hit it for good measure)
2. Press `:` and then `q` (quit), then `Enter`. You will exit `vi`.

Cursor motion

Now that you know how to open a file, put some text into it, and then exit the editor. The next most important thing to learn is how to move around in the file. While using the arrow keys might work, it's far from optimal, and may not be implemented in strictly POSIX-compliant versions of `vi`.

When in **normal mode**, you use the `h`, `j`, `k`, and `l` (ell) keys to move the cursor around:

- `h` - left
- `j` - down
- `k` - up
- `l` - right

Seems confusing and awkward at first, right? It's actually not. You will quickly learn the motions without having to think about it, and you will be able to move around very fast in the file, without ever having to move your hands from the home row of the keyboard.

In addition to the simple cursor motion commands, there are a few other cursor motion commands that are extremely helpful to know:

- `w` - move forward one word
- `b` - move back one word
- `0` - move to the beginning of the line
- `^` - move to the first non-blank character of the line
- `$` - move to the end of the line
- `/some text` - search forward to the next match of `some text`, and place the cursor there
- `?some text` - search backward to the previous instance of `some text`, and place the cursor there
- `n` - repeat the most recent search
- `N` - repeat the most recent search, but in the opposite direction
- `gg` - Go directly to the top of the file
- `G` - Go directly to the bottom of the file
- `numberG` - Go directly to line `number` (example: `20G` goes to line 20)

Text insertion commands

`vi` gives you several options for how you actually want to insert text when you enter insert mode.

- `i` - insert text at the position of the cursor. The first character you type will appear to the left of where the cursor is
- `a` - append text at the position of the cursor. The first character you type will appear to the right of where the cursor is
- `I` - Same as insert, but first moves the cursor to the beginning of the line (equivalent to `^i`.
- `A` - Same as append, but first moves the cursor to the end of the line (equivalent to `$a`.
- `o` - Open a new line under the cursor and begin inserting text there
- `O` - Open a new line above the cursor and begin inserting text there

Text removal commands

You will notice that while in **insert mode**, you can use the backspace and delete keys as expected. This makes insert mode easiy to use, but it's not particularly efficient if you're trying to eliminate a whole paragraph or something from your document. When in **normal mode**, you can issue some commands that remove whole chunks of text:

- `x` - Delete the character under the cursor
- `dw` - Delete from the character under the cursor to the beginning of the next word
- `dd` - Delete the line under the cursor
- `NUMdd` - Delete NUM lines of text, ex: `10dd` deletes 10 lines
- `dgg` - Delete the current line, and everything else to the top of the file
- `dG` - Delete the current line, and everything else to the bottom of the file

Undo and Redo

Now that you know how to add and remove text, you'll inevitably end up making a mistake. Luckily, `vi` lets you undo the last command or insertion, by going back to **normal mode** and hitting the `u` key.

In `vim` (but not strict POSIX `vi`), you can also press `Ctrl-R` to redo the last thing you un-did.

Professional Fonts (vim?)

Syntax Highlighting (vim)

Directory navigation (NERDtree)

Edit/Open/Close Files

Edit/Open/Close Buffers (vim)

Text Editing 201

Vim

[What is Vim, compared to Vi?]

Window Splitting

Plugins

Emacs

[Intro to emacs, what it is, how it's different to Vi]

Vi was first developed on computers where the keyboards looked like this:

https://en.wikipedia.org/wiki/Vi#/media/File:KB_Terminal_ADM3A.svg

Emacs was first developed on computers where the keyboards looked like this:

https://en.wikipedia.org/wiki/Symbolics#/media/File:Symbolics-keyboard.jpg

The important thing to note in the Symbolics keyboard above is the keys on the bottom row:

`[HYPER] [SUPER] [META] [CTRL] [ SPACE ] [CTRL] [META] [SUPER] [HYPER]`

Where Vi has modes for different editing tasks with composable commands, Emacs relies on key combinations or "chords", particularly the `CTRL` and `META` keys, abbreviated `C` and `M`. The convention for canonically specifying a keyboard combination in Emacs is based on those abbreviations using a connecting dash for keys pressed simultanously and a space for follow-up key "arguments". For example, the most important key combination in Emacs is this one: `C-h ?` meaning, press and hold `Ctrl` then press `h`, release and press `?`. This takes you to Emacs' "help" options.

Modern keyboards lack a `META` key, but Emacs has long recognized `Alt` as `META`. For example, the second most important key combination in Emacs is `M-x`, meaning press and hold `Alt` and then press `x`. This brings up what amounts to Emacs' "command line" where you can enter any Emacs command to run.

It bears mentioning the "zeroth" most important key command: `C-g` meaning press and hold `Ctrl` and press `g`. This tells Emacs to cancel whatever it's doing. It almost always works, although sometimes after a short wait. `C-g` combines with every partial keyboard combination to cancel it.

What Emacs lacks in command composability, it makes up for in programmability. Nearly every aspect of the editing experience is modifiable using Emacs' native language, Emacs Lisp. You can use `M-:` to open another Emacs' command-line, this one takes any Emacs Lisp expression and evaluates it. You can also go to the "scratch" buffer, enter an Emacs lisp expression and press `C-x C-e` to evaluate it.

You can read Emacs' online manuals at `C-h i`. Going through the Emacs tutorial—`C-h t`—is highly recommended.

To close Emacs `C-x C-c`.

Edit/Open/Close Files

- `C-x C-f` `find-file` open a file in a buffer to edit
- `C-x C-r` `find-file-read-only`

- `C-x k` `kill-buffer` close a buffer
- `C-x C-f /sudo:localhost:/etc/hosts` use Tramp-mode to edit a privileged file (of course, you must be privileged)

Edit/Open/Close Buffers

- `C-x b` `switch-to-buffer` switch to another open buffer
- `M-x ibuffer` see a list of open buffers, to navigate or run other commands on

Directory Navigation

- `C-x d` see directory contents as a simple list
- `C-x D` use Dired to operate on a directory's contents

Syntax Highlighting

Emacs autodetects the kind of file one is operating on and applies syntax highlighting as part of the "mode" used for that buffer. Emacs comes with built-in modes for almost everything. You can explicitly run any mode with `M-x`.

Line numbers

Emacs typically shows the cursor's position in a buffer on the "mode-line" near the bottom of the screen. You can configure it to show the position in a "row and column" format running `M-x line-number-mode` (usually on by default) and `M-x column-number-mode`. To see line numbers in buffer use `linum-mode`. To highlight the current line use `hl-line-mode`.

Window Splitting

- `C-x 2` `split-window-below` splits the window horizonally (especially useful for looking at two widely separated parts of the same buffer).
- `C-x 3` `split-window-right` splits the window vertically
- `C-x o` `other-window` switches focus to other window
- `C-x 0` `delete-window` closes this window
- `C-x 1` `delete-other-windows` closes the other window
- `C-x 4 f` `find-file-other-window` opens file in other window
- `C-x 4 r` `find-file-read-only-other-window`
- `C-x 4 b` `switch-to-buffer-other-window`

Buffers

```
M-x ibuffer
```

References

- https://www.gnu.org/software/emacs/manual/ (the web version of `C-h i`)
- https://www.emacswiki.org
- http://planet.emacsen.org
- http://hyperpolyglot.org/text-mode-editors

Tools for productivity

The work of operations is often helped by using a combination of tools. The tools you choose to use will depend heavily on your environment and work style. Over time you may even change the tools you use.

This is a list of commonly used tools, we recommend trying as many of them as you have time for. Finding the right tool is often like finding a good pair of gloves - they have to fit just right!

Terminal emulators

OSX: iTerm2

Rather than stick with the default terminal, many sysadmins on OS X install iTerm2. iTerm2's greater flexibility and larger feature set have made it a de facto standard among technical Mac users. Like most modern terminals, it supports vertical and horizontal splits, tabs, profiles, etc. In addition, it stores the rendered buffer for a period of time which serves as a way to view previous things that might have been lost due to editor crashes or the like.

Linux: Gnome Terminator

Many advanced Linux sysadmins prefer tiling window managers such as i3, Awesome, xmonad, herbstluftwm, and others. However, these window managers can be daunting to a new user when compared to a more Windows-like desktop environment like Unity, KDE, or XFCE. A popular compromise is to use a tiling terminal emulator, such as Terminator, within a regular window manager.

Terminator has some features which are convenient for performing similar commands in several places at once, such as grouping terminals together and then broadcasting your keystrokes to all terminals in the group.

SSH

Common: `ssh`

If your operating system has a good terminal emulator available (e.g., Linux, BSD, OSX), then the `ssh` command line utility is by far the best tool to use for SSH interactions.

Windows: PuTTY

If you use Windows as your primary work space, PuTTY is possibly one of the best SSH clients available.

Android: Connectbot

Connectbot is a free app available in the Google Play store, and offers most of the functionality you would need from an SSH client.

iOS: iSSH

A universal app available on the App Store, iSSH supports most–if not all–of the features a demanding power user needs.

SSH Use Cases

Far more than just a secure way to access a shell on a remote machine, ssh has evolved to support a large collection of ways to encapsulate traffic over the encrypted channel. Each of these is useful in different situations. This section will present some common problems, and will present how to solve the problems with ssh. It is important to note that all port forwarding and proxying occurs over the secure ssh channel. Besides working around firewall rules, the tunnels also provided a layer of security where there may not otherwise be one.

Single port on foreign machine firewalled

For this usecase, consider a loadbalancer–`lb-foo-1`–with a Web management interface listening on port 9090. This interface is only routable to a LAN private to the datacenter–10.10.10.0/24. Its IP address is 10.10.10.20. There is a machine on the 10.10.10.0/24 network with ssh access–`jumphost-foo-1`, accessible via DNS with the LAN IP 10.10.10.19. Therefore, one way to access `lb-foo-1` is via bouncing through `jumphost-foo-1`. OpenSSH supplies the `-L` option to bind a local port to a port opened on the other side of the tunnel. On the remote end, ssh opens a connection to the host and port specified to the `-L` option. An example for this usecase:

```
ssh -L 9090:lb-foo-1:9090 jumphost-foo-1
```

This will open a connection from `jumphost-foo-1` to `lb-foo-1` on port 9090, and will bind the local port 9090 to a tunnel through the connection to that port. All traffic sent to the local port 9090 will reach `lb-foo-1:9090`. It is important to note that the host given to `-L` uses the perspective of the machine ssh connects to directly. This is important to remember for environments using hosts files or private / split-horizon DNS. In this usecase, an invocation like the following would work as well:

```
ssh -L 9090:10.10.10.20:9090 jumphost-foo-1
```

In this case, `jumphost-foo-1` would try to connect to `10.10.10.20` directly, skipping the DNS lookup.

Tunnel all traffic through remote machine

For this usecase, consider the rest of the machines in the 10.10.10.0/24 network. There are many ways to access them, including VPNs of various sorts. In addition to creating tunnels to specific ports, OpenSSH can also create a SOCKS proxy. OpenSSH provides the `-D` option for this. As an example, to bounce from `jumphost-1` to the rest of the network:

```
ssh -D 9999 jumphost-foo-1
```

Once the SOCKS proxy is in place, the operating system or applications needing access to the proxy need to be configured to use it. This will vary per-application. Some operating systems (OS X) provide system-wide proxy configuration.

Reverse Tunneling

In some situations it may be necessary to forward ports from the remote machine to the local one. While not as common as other port forwarding techniques, it is often the only option available in the situations where it sees use. For this feature, OpenSSH listens to a port on the remote machine and sends the traffic to the local host and port. The syntax for the `-R` option is the same as for the `-L` option described above. As an example, to have OpenSSH tunnel the remote port 9090 to the local port 8000 on host `workstation`:

```
ssh -R 9090:workstation:8000 jumphost-foo-1
```

Then, programs on `jumphost-foo-1`–or that can access its port 9090–will be able to access the local port 8000 on `workstation`.

Tunneling stdin and stdout

This method makes using a jumphost transparent. The idea is to tunnel not a single port, but the entire connection over the ssh channel. This usage is typically a matter of configuration in the `ssh_config` file, taking advantage of its wildcard-capable `Host` directive:

```
Host *.lan
ProxyCommand ssh -W %h:22 jumphost-foo-1
```

This configuration uses the ProxyCommand feature to cooperate with the `-W` flag. Hostnames are resolved from the perspective of the jumphost. In this example, the machines use an internal pseudo-top level domain of `.lan`. To reach, e.g., `www-1.lan`:

```
ssh www-1.lan
```

Before doing DNS resolution, OpenSSH will look in its `ssh_config` file for Host entries. Therefore, internal DNS on the foreign end is sufficient.

Multiplexers

Operations work regularly involves connecting to remote servers (there will be times when you do nothing but work on remote servers - parts of this curriculum were even typed on remote servers rather than contributors desktops and laptops!).

There are however two limitations to working this way:

1. You'll often need to be connected to more than one remote system at a time. Opening a whole new terminal each time can result in a lot of windows cluttering up precious screen space.
2. What happens if your internet connection stops working? All of your connections are reset. Any work you might have been doing on the remote servers can be lost.

Multiplexers are a good solution to this. They allow you to run multiple "virtual" windows inside a single windows. For example:

> Bob works on 10 remote servers, all of which run Linux. Bob's internet connection at work is questionable. To work around this, Bob connects to `server1` which is at his data centre. It is a reliable server which is close to the other servers Bob works on. On `server1`, Bob starts a multiplexer. The multiplexer gives Bob a regular looking command prompt, and Bob continues his work.
>
> If Bob's internet connection drops, he can reconnect to `server1`, and then re-attach to the multiplexer he started previously. His session is in the same state he left it before being disconnected, and he can continue his work.
>
> The multiplexer also lets Bob open more than one command prompt and switch between them as he needs to. Bob can now connect to many servers and see them all in one window.

GNU Screen

`screen` is one of the longest lived multiplexers. Almost everyone who has used a multiplexer has used screen, and you can't go far wrong with it.

`screen` is a full-screen window manager that multiplexes a physical terminal between several processes (typically interactive shells). It is useful for creating sessions that can be disconnected from and reconnected to later. This is useful for running tasks that can take a long time that you do not want to have an ssh session timeout on, such as a large database import. In these cases cron is also a very good way to run one off long running tasks.

`screen` is also **very useful** for creating sessions that users can share.

Installation

Debian and descendants (Ubuntu, Mint, Suse, etc):

```
apt-get install screen
```

On RedHat-style distributions install with the command:

```
yum install screen
```

Basic usage

Create a session:

```
screen -S session1
```

To detach from a session - in the session type Ctrl+a+d

List available screen sessions:

```
screen -ls
```

```
[gary@mc9 ~]# screen -ls
There are screens on:
        21707.session2  (Detached)
        21692.session1  (Detached)
        21936.session3  (Attached)
3 Sockets in /var/run/screen/S-gary.
[gary@mc9 ~]#
```

Here we can see 3 screen sessions are running, 2 detached and 1 attached.

Reattach to a session:

```
screen -r session1
```

Share a session:

User Alice starts session:

```
screen -S session1
```

User Bob can then attach to the same session (both Alice and Bob can send commands to the session):

```
sudo screen -x alice/session1
```

Non root users, must use sudo to attach to another user's session.

Create a session with a log:

```
screen -L -S session1
```

`screen` will output the session log to the user's home directory with the file `~/screenlog.0` (0 being the session id). PuTTY is also as a very useful and featureful ssh client that can be used for logging ssh sessions locally (Windows and Linux). `screen` can be used within a PuTTY session.

Create a session with a log and 20000 lines of scrollback in the terminal:

```
screen -h 20000 -L -S session1
```

Configuration

`screen` has a fairly extensive set of configuration options, when screen is invoked, it executes initialization commands from the files `/etc/screenrc` and `.screenrc` in the user's home directory.

Further info

```
man screen
```

There is a nifty cheat sheet for the most important `screen` and `tmux` keybindings (see below in tmux references [11]).

Tmux

`tmux` [12] is relatively new compared to `screen`. It covers the same basic feature set and has added a few more advanced features. It is recommended you get comfortable with `screen` first before attempting to use `tmux`.

In this chapter you will learn to start a tmux session, get to know a few first keyboard shortcuts and detach from and re-attach to the session.

Installation

tmux is available on Debian and its descendants like Ubuntu or Mint with the command:

```
apt-get install tmux
```

On RedHat-style distributions you will have to use the *EPEL* repository to get a pre-built package, and install with the command:

```
yum install tmux
```

On MacOS you can use Homebrew to install via:

```
brew install tmux
```

[11] http://www.dayid.org/os/notes/tm.html
[12] http://tmux.sourceforge.net/

tmux basics

`tmux` is usually started with the command `tmux` in a terminal window. Depending of your version of tmux you will see either a line at the bottom of the screen or nothing at all. `tmux` is controlled with keyboard shortcuts, the default shortcut usually is `ctrl-b`. If you press `ctrl-b` and then a `t` in the newly started tmux window you should see the local time displayed as a large digital clock. If you hit `ctrl-b` and `c` you should see a new empty window with an empty input prompt.

If you want to detach from the session you have to hit `ctrl-b` and `d`. The `tmux` window will disappear and you will see a message `[detached]` in your terminal window. All the shells and processes you started onside the `tmux` session continue to run, you can see this with a simple

```
ps -ef | grep tmux
```

You should see something like the following:

```
cdrexler 13751     1  0 Nov30 ?        00:00:41 tmux
```

You will notice that the `tmux` process has a parent process id of 1 which means that it is not a child process of the shell you started it in anymore. Accordingly you can leave your working shell, start a new one and attach to the running tmux process again which is very handy if your connectivity is flaky or you have to work from different locations. If you check the process table for the process id of the tmux process

```
ps -ef | grep 13751
```

you will find that is the parent process of the two shells you created in the beginning of the chapter:

```
cdrexler  4525 13751  0 17:54 pts/2    00:00:00 -zsh
cdrexler  4533 13751  0 17:54 pts/5    00:00:00 -zsh
```

If you want to get an overview of the running tmux processes on your system you can use the command

```
tmux ls
```

It will list all available `tmux` sessions on your system [13]. If there is only one you can attach to it with the command:

```
tmux att
```

If there is more than one session the output of `tmux ls` will look like this:

```
0: 3 windows (created Fri Nov 30 18:32:37 2012) [80x38]
4: 1 windows (created Sun Dec  2 17:44:15 2012) [150x39] (attached)
```

You will then have to select the right session with the `-t` command line switch:

```
tmux att -t 4
```

`tmux` runs as a server process that can handle several sessions so you should only see one tmux process per user per system.

You should see the original session with the two shells again after running this command.

tmux configuration

`tmux` is configured via a config file which is usually called `.tmux.conf` that should live in your `$HOME` directory.

A typical `.tmux.conf` looks like this:

[13] Please note that `tmux ls` will *only* list tmux sessions that belong to your userid!

```
#set keyboard shortcut to ctrl-g
unbind C-b
set -g prefix C-g
bind C-g send-prefix
bind g send-prefix
#end of keybord shortcut setting
# Highlight active window
set-window-option -g window-status-current-bg red
# Set window notifications
setw -g monitor-activity on
set -g visual-activity on
#automatically rename windows according to the running program
setw -g automatic-rename
#set scroll back buffer
set -g history-limit 10000
set -g default-terminal "xterm-256color"
set -g base-index 1
set -g status-left '#[fg=green]#H
```

This illustrates a method to change the default keybinding and some useful settings.

Please note that you can force `tmux` to use another configfile with the `-f` command line switch like so:

```
tmux -f mytmuxconf.conf
```

There is a nifty cheat sheet [3] for the most important `screen` and `tmux` keybindings or even a whole book about tmux [14].

byobu

Byobu is a wrapper around one of screen or tmux. It provides profile support, F-keybindings, configuration utilities and a system status notification bar for most Linux, BSD or Mac operating systems.

Byobu is available in major distros as a packaged binary. Launching byobu will run whichever the package maintainer included as a dependency, if you have both installed, you can select explicitly with byobu-{screen,tmux}. Basic configuration is launched with F9 or byobu-config.

Scrollback starts with F7, mark the start of your buffer by hitting the spacebar, then use the spacebar again to mark the end of your selection (which will be copied into byobu's clipboard automatically). Press Byobu-Escape (typically Ctrl-A) + [to paste.

References

Shell customisations

As you read in *Shells*, your shell is your primary tool during the work day. It's also incredibly customisable to suit your needs. Let's look at some changes you can make.

How to customise your shell

Your shell's configuration is stored in its `rc` file. For bash, this file is `~/.bashrc`. Each time you edit this, you can reload the configuration by typing:

```
source ~/.bashrc
```

[14] http://pragprog.com/book/bhtmux/tmux

Changing your prompt

Your default prompt probably looks something like this:

```
bash-3.2$
```

That's pretty plain and doesn't tell you much. In fact, all it does tell you is that you're using Bash version 3.2, and that you are not the root user (the `$` at the end signifies a regular user, whereas if you were root, you would see a `#` instead).

Let's change this up a little. Edit your `~/.bashrc` file, and add this line to the end:

```
PS1="\u@\h \w> "
```

Save, quit, and then reload your `.bashrc` file. Your prompt should change to something like this:

```
avleen@laptop ~>
```

Much better! Now your know your username, the name of the machine you're on (in this case "`laptop`"), and the directory you're in ("~" is your home directory).

The `PS1` variable has a lot of different options you can use to customise it further.

Mosh

Mosh (MObile SHell) is an alternative to remote shell commands, such as `ssh` or `rsh`. The beauty of the Mosh protocol is that it supports intermittent connectivity without losing the remote session.

You can start a `mosh` session just like you would with `ssh` on one side of town with one IP address, shut your laptop and go home, then open your laptop and connect back to your Mosh session like it was never interrupted. Also, if your wifi is spotty or internet connection is intermittent, `mosh` doesn't break the session when the connection drops out, unlike `ssh`. Mosh does not wait for the remote server to confirm each keystroke before displaying it to a terminal. Instead, it displays the typed characters locally and confirms entry on the remote end. There are packages available for GNU/Linux, FreeBSD, Solaris, Mac OS X, Chrome and even Android apps.

Mosh must be installed on both the client and remote server. When a session is started it spins up a mosh-server and a local mosh-client process. It can be installed in a home directory without privileged access. Mosh respects your current `~/.ssh/config` so migrating from `ssh` to `mosh` is relatively seamless.

SSH connections work by sending a stream of data back and forth between the client and server. This stream can be broken in various ways, such as the connection timing out due to inactivity, or the client machine suspending state and shutting down network devices. The Mosh protocol is based on UDP packets compared to the SSH protocol that uses TCP packets. Mosh is the first application to use the Stateless Syncronization Protocol. Instead of a stream of data between the server and client that makes up a session over SSH, Mosh works by keeping a mirrored copy of the session on the client and server and syncronizing the changes between them.

While `ssh` has been around long enough to have time tested security, `mosh` is relatively new and has not been through the same extensive testing. It is the first application to use the SSP protocol. Mosh does tunnel traffic encrypted with AES-128 in OES mode, however Mosh hasn't been under the security spotlight as long as SSH has.

Examples

Mosh works just like ssh:

```
mosh username@remoteserver.org
```

You can also have mosh utilize ssh style commands such as: This specifies the private key to use to authenticate with the remove server.

```
mosh username@remoteserver.org --ssh="ssh -i ~/.ssh/identity_file"
```

This tells mosh to connect via the ssh port 1234 on the remote server, where ssh normally runs on port 22.

```
mosh username@remoteserver.org --ssh="ssh -p 1234"``
```

References

Ticketing systems

Note-taking

Wiki

EverNote

OneNote

Security 101

Authentication in unix

Todo

Discuss how authentication works. Touch on `/etc/(passwd|group|shadow)`, hashing. What are groups? Lead in to the users/groups permissions model and how permissions are based on the user/group/other bits.

Adding and deleting users and groups

Standard unix filesystem permissions

The simplest way of displaying filesystem permissions is by typing:

```
$ ls -l
drwxr-xr-x   2 john  company  68  3 Oct 10:34 files
-rwxrwxrwx   1 john  company   0  3 Oct 10:29 hello_world.txt
```

The left column is a 10-character string that indicates the permissions for a file. It consists of the symbols d, r, w, x, -.

- **Directory(d)** - This is the first character in the permissions string. This indicates a *directory*. Otherwise, the first character is a - to indicate that it is not a directory.
- **Read (r)** - The *read* permission allows the user to read the contents of the file or list the files in the directory.
- **Write (w)**- The *write* permission allows the user to write or modify a file. In the case of directories, the use may delete files from the directory or move files into the directory.
- **Execute (x)** -The *execute* permission allows the user to execute a file or access the contents of a directory. In the case of directories, this indicated that the use may read files in the directory, provided that the user has read permission on an individual file.

The 9 remaining characters are split into 3 sets to represent the access rights based on 3 groups of users. Take the "files" directory above as an example, we can split the characters like this: `[d][rwx][r-x][r-x]`

- The first character, as explained above, indicates a directory or a file
- The first group gives the file permissions for the *owner* of the file or directory. This means that the user "john" has read/write/execute permissions to the directory.
- The second group gives the file permissions for the *group* of users to whom the file or directory belongs to. This means that anyone who is under the group "company" has read/execute permissions to the directory.
- The third group gives the file permissions for *other* users. Basically anyone who are not the owner or a part of the user group. This means that everyone else has read/execute permissions to the directory.

Some more examples of permissions:

- `-rwxrwxrwx` is a file everyone can read, modify (including delete), and execute.
- `-rw-------` is a file only the user can read and modify.

PAM

Chroot, jails and containers

Sudo (or, "Why you should not log in as root")

History and Lore

The Morris Worm

http://www.snowplow.org/tom/worm/worm.html

/bin/false is not security

https://web.archive.org/web/20150907095805/http://www.semicomplete.com/articles/ssh-security

Security 201

Centralised accounts

LDAP and Kerberos

Active Directory

NIS, NIS+, YP, Hesiod

Firewalls and packet filters

host vs network

"Crunchy outer shell, soft center is bad"

In general terms, implementing defense-in-depth strategies is always a sensible practice. The concept in which multiple layers of security controls (defense) are placed throughout an information technology (IT) system. Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited.

Implementing a firewall on the network **and** host-based packet filters provides defense-in-depth layers to your infrastructure. In today's landscape, the firewall aspect could be a service like ec2 security groups with host-based packet filters such as iptables, Windows Firewall (or WFAS). Although this can add additional complexity to deployment, that is not a reason to not implement it where appropriate.

The defense-in-depth concept is mostly regarded in terms of attack and compromise, however in ops it also safeguards **us** as everyone makes mistakes. Sometimes, we ourselves or our colleagues are the point of failure.

Strange as it may seem, people often make the mistake of disabling "rules" when something is not working and they cannot figure out why. The *just checking* test. This is always the first mistake. In real world operations these things do happen, whether it is a *just checking* mistake, an incorrect configuration or action, sometimes we make serious mistakes, to err is human.

In all these situations using a firewall/other and host-based packet filters comes to the fore:

- It protects us, the people working on the systems, from ourselves.
- It protects the organisation's assets in the event of a failure or compromise at either layer, whether it be user error or a systematic failure.
- It keeps us proficient and *in-hand* on both the network specific security implementation, the host-based security practices and the applications related to their management.
- It is good practice.

An old skool, real world example:

- You have a MSSQL server running on Windows Server 2000 (no "firewall" back then & ip filters are not enabled) - it has both private and public network interfaces.
- The MSSQL server has a public NIC because it has run replication with your customer's MSSQL server sometimes for dev purposes and catalog updates.
- You have rules on the diversely routed, mirrored NetScreen NS1000 firewalls that allows port 1433 between the MSSQL servers only on the public interface.
- Your colleague has an issue that cannot be resolved and quickly just sets the firewall/s to "Allows from all", the *just checking* test.
- **Immediate** result - network unreachable.
- SQL Slammer had just arrived and proceeded to gobble up 2Gbps of public T1 bandwidth.
- All points into the network and the network itself are saturated until you debug the issue via the serial port on the firewall and figure out what happened.

The synopsis is that a practice of disabling rules was implemented, as it had always been the last line in debugging. It was a practice that had been done many times by the engineers in the organisation at the time with no "apparent" consequences in the past. This time differed in that the MSSQL server had a public interface added to allow for replication with the customer **and** SQL Slammer was in the wild.

If the MSSQL server had ip filtering enabled as well the situation would have been mitigated. Needless to say, it was the last time "Allow from all" debug strategy was ever implemented. However it is useful to note, that because this was done all the time, the engineer in question did not even tie the action of "Allow from all" and the network becoming unreachable together at the time, because the action previously had never resulted in the outcome that was experienced in this instance.

Stateful vs stateless

IPTables: Adding and deleting rules

pf: Adding and deleting rules

Public Key Cryptography

Todo

What is PKI? What uses it? Why is it important?

Using public and private keys for SSH authentication

Two factor authentication

Building systems to be auditable

Data retention

Log aggregation

Log and event reviews

Role accounts vs individual accounts

Network Intrusion Detection

Host Intrusion Detection

Defense practices

Risk and risk management

Compliance: The bare minimum

What is compliance and why do you need it?

What kinds of data can't you store without it?

Legal obligations

Dealing with security incidents

ACLs and extended attributes (xattrs)

SELinux

AppArmor

Data placement

Eg, local vs cloud, the implications, etc

Additional reading

Ken Thompson, Reflections on Trusting Trust: http://dl.acm.org/citation.cfm?id=358210

Troubleshooting

A key skill for anyone doing operations, is the ability to successfully troubleshoot problems.

Methodologies

Here we will got over a few steps you can take to help quickly narrow down problems to their causes.

Triaging with differential diagnosis

What is broken? First think about how it works in most basic terms. Then build on that the things which can break. Then from those, pick the ones that could cause the symptoms you see.

Example: You cannot ping a server.

Preparing your toolbelt

You have a variety of tools at your fingertips to help work out the cause of a problem. Over time you will expand what is in your toolbelt, but to start with you must know how to use each of these:

- `top`, `vmstat`, `iostat`, `systat`, `sar`, `mpstat` help you see the current state of the system - what is running, what is using cpu, memory? Is the disk being heavily used? There is a great deal of information, and knowing how these tools work will help you pick out the bits you should focus on.
- `tcpdump`, `ngrep` If you suspect you have a network-related problem, `tcpdump` and `ngrep` can help you confirm it.

Walk through of a diagnosis

- Eliminating variables
- What changed recently?
- Could any of the symptoms be red herrings?
- Common culprits (is it plugged in? is it network accessible?)
- Look through your logs
- Communicating during an outage
- 'Talking Out-Loud' (IRC/GroupChat)
- Communicating after an outage (postmortems)

Recent changes

Often problems can be traced back to recent changes. Problems that start around the time of a change aren't usually coincidence.

Learning common errors

Over time you may find that a small set of errors cause a large portion of the problems you have to fix. Let's cause some of these problems and see how we identify and fix them.

Cannot bind to socket

There are two common reasons that you can't bind to a socket: the port is already in use, or you don't have permission. As an example, you can see what happens when I try to start a Python SimpleHTTPServer on a port that is already in use:

```
user@opsschool ~$ python -m SimpleHTTPServer 8080
...
socket.error: [Errno 98] Address already in use
```

Here's an example of what happens when I try to bind to a privileged port without proper permissions (in Linux, ports < 1024 are privileged):

```
user@opsschool ~$ python -m SimpleHTTPServer 80
...
socket.error: [Errno 13] Permission denied
```

Permission denied reading from / writing to disk

Out of disk space

(finding large files, and also finding deleted-but-open files)

Mystery problems and SELinux

Out of inodes

Manifests as "disk full" when `df` claims you have disk space free.

Working effectively during a crisis

Being able to work successfully through a crisis is crucial to being a good operations person. For some it is a personality trait, but it can certainly be learned and is almost a requirement for many employers.

A very important skill to learn is the ability to remain calm in the face of disaster. It's not always easy, especially with a client on the phone, but panicking will only make a situation worse. Yes, the most critical server in the infrastructure may have just completely failed without a backup. Instead of focusing on what will happen as a result of the crisis, focus on what needs to be done to bring the system back up. Deal with the results later, after fixing the immediate failure. The fallout of the crisis might be terrible, but it will almost certainly be worse if the immediate problem isn't fixed. A calm mind can carefully analyze a situation to determine the best solution. Panic responses do not benefit from the same calculating rationality.

Different people will adapt to handling crisis situations in different ways. Some will adopt the detached, analytical calm of a surgeon. Others will take a few deep breaths to calm themselves before digging in to analyze the problem. The ability to stay calm in the face of disaster is more important than the method by which calm is achieved. It will take practice to reach the point of reacting to a disaster calmly.

Avoid placing blame. It doesn't accomplish anything beyond creating animosity and tension when a team most needs cohesion and efficiency. While a good practice in general, it is even more important to resist the urge to point fingers during a crisis. It doesn't assist in solving the problem, which is the top priority. Everything else is secondary.

The Importance of Procedure

Creating procedures for responding to disasters provides both a checklist of things to do in the given situation as well as a structured way to practice responding to the situation. The practice serves to solidify understanding of how to react, while the procedure itself provides a target of mental focus during an actual disaster. Adhering to the procedure ensures the steps taken to resolve a crisis are well-known and tested. Focus on the procedure to the exclusion of everything else.

That said, not every situation will have an associated procedure. These situations call for their own procedures. Try to create a procedure for every situation that doesn't already have one. This diligence pays off over time, as history tends to repeat itself. In addition to this, a procedure for situations lacking a procedure provides a safety net when everything else fails. This will differ from one organization to the next, but the value is constant.

Like backups, no disaster recovery procedure is useful unless and until it is tested. Thorough testing and practicing–in a real environment if possible–quickly finds problems that will happen in the real world. Beyond having procedures for known possible failures, a procedure for situations other procedures do not cover provides a fallback for what to do in the inevitable unpredictable crisis.

In addition to the technical sector, other industries deal regularly with crisis response–fire fighters, law enforcement, paramedics. These organizations have their own procedures. These industries all predate technology, offering much to learn.

Non-technical skills

Situational Awareness (Mica Endsley) Decision Making (NDM and RPD) - Klein Communication (Common ground, Basic Compact, Assertiveness) Team Working (Joint Activity, fundamentals of coordination and collaboration) Leadership (before, during, after incidents) (Weick, Sutcliffe work on HROs) Managing Stress Coping with Fatigue Training and Assessment Methods Cognitive Psychology concerns (escalating scenarios, team-based troubleshooting)

Networking 101

This chapter should provide enough knowledge on networking to enable a systems administrator to connect a Linux server to a network and troubleshoot basic network-related problems. First, we will go over the basics of the 7-layer Open Systems Interconnection (*OSI*) model, which is a standard framework with which to describe communication system logical layers. Next, we will delve into each layer of the OSI model in more detail as it applies to the role of systems administration.

Before any discussion of networking, however, it's important to have a working knowledge of the numbered Request for Comments (*RFC*) documents and how they apply to computer networking. These documents describe the technical specifics for every protocol you will run into (e.g., TCP, IP, HTTP, SMTP), and as such are the authoritative source for how computers communicate with one another.

The RFC Documents

Starting in 1969, the RFC document series describes standards for how computers communicate. The series gets its name from the RFC process, wherein industry experts publish documents for the community at large and solicit comments on them. If the Internet community finds errors in a document, a new, revised version is published. This new version obsoletes the prior versions. Some documents, such as the document specifying email messages, have had several revisions.

The RFC Editor manages the RFC archive, as well as associated standards. New documents go to the RFC Editor for publication [15], whether revision or new standard. These documents go through a standardization process, eventually becoming Internet-wide standards. Many of the networking protocols discussed in later chapters have RFCs governing their behavior, and each section should provide information on the relevant RFCs.

Important RFCs

There are a number of RFCs which don't pertain to any specific technology but which are nevertheless seminal. These documents establish procedure or policy which have shaped everything after, and as such have a timeless quality. In some cases, later documents make references to them. This list is given in increasing numerical order, though is not exhaustive.

- **RFC 1796**: Not All RFCs are Standards

  This document describes the different kinds of documents in the RFC series.

[15] This is a simplification, as there are actually many standards bodies involved in the process. The RFC Editor Publication Process document explains in full detail.

- **RFC 2026**: The Internet Standards Process

  This document (and those that update it) describes in detail how RFCs are published and how they become Internet standards.

- **RFC 2119**: Key words for use in RFCs to Indicate Requirement Levels

  This document, referenced in many following RFCs, presents a common vocabulary for specifying the relationship between a standard and implementations of that standard. It provides keywords that specify how closely an implementation needs to follow the standard for it to be compliant.

- **RFC 5000**: Internet Official Protocol Standards

  This document provides an overview of the current standards documented by the RFCs and which RFC is the most recent for each standard. This document is regularly updated with the current standards document status.

OSI 7-layer model (OSI Reference Model)

The early history of computer networking was marked by various proprietary implementations of networking hardware and the protocols that ran on them (such as for example, IBM's "Systems Network Architecture" (SNA) and Digital Equipment Corp's "DECnet".) In the mid-1970's, two standards organizations (the International Organization for Standardization, known as the "ISO", and the International Telecommunications Union, known as the "ITU") led a push to define a set of open interconnection standards, which became known as the "Open Systems Interconnection" (OSI) standard. While the actual protocols they developed did not become popular and are not in wide use today, the model they came up with has achieved wide popularity as a way of thinking about the network protocol stack. It's important to understand that the OSI model is a conceptual model, and therefore more of a mental framework than a technical framework. It is useful primarily for troubleshooting and explanation purposes. While the OSI model may seem esoteric as you're learning it, it will eventually prove invaluable to your daily work once you have a grasp of the concepts.

The OSI model describes seven layers of abstraction that enable software programs to communicate with each other on separate systems. The seven layers are designed to allow communication to occur between systems at a given level, without concern for how the lower levels are implemented. In this way, innovation on the protocols in the various layers can be achieved without modifying any other layer's design and code. The job of each layer is to provide some service to the layer above by using the services provided by the layer below.

- Layer 1 - Physical layer

  The physical layer describes the physical connections between devices. Most enterprise networks today implement Ethernet at the physical layer, described in IEEE 802.3 for wired connections and IEEE 802.11 for wireless networks. Other Layer 1 protocols that are no longer in wide use today are Token-Ring, and FDDI.

- Layer 2 - Data link layer

  The data link layer defines the basic protocol for communicating between two points on a network, that may consist of many intermediate devices and cables, possibly spanning a large geographic area. The IEEE 802.3 specification defines the data link layer, which includes the Media Access Control (*MAC*) addresses that allow hosts to address their data to one or more systems on the same Ethernet segment. The MAC address is a flat (non-hierarchical) 48-bit address, and therefore when a node sends out a MAC broadcast frame (usually written as FF:FF:FF:FF:FF:FF), all stations on the Layer 2 segment receive it. Therefore, a layer 2 segment is also known as a "broadcast domain".

- Layer 3 - Network layer

  The network layer is what allows many flat Layer 2 segments to be interconnected, and separates broadcast domains. It is this layer of the OSI model that enables the Internet to exist, using Internet Protocol (IP) addressing. Version 4 of the Internet Protocol (IPv4), most commonly found in today's production networks, is described in **RFC 791**. Its successor, IP version 6 (IPv6) is described in **RFC 2460**. Both protocols utilize addresses that are

hierarchical in nature, providing for a *network* portion of the address space, and also a *host* portion within the network.

There are also two control protocols included in this layer; namely, the "Internet Control Message Protocol" ("ICMP") as described in **RFC 792**, and "Internet Group Management Protocol" ("IGMP"), as described in **RFC 1112**. The ICMP protocol provides for Layer 3 device control messaging, and among other uses, allows small test packets to be sent to a destination for troubleshooting purposes, such as those used by the ubiquitous `ping` utility. The IGMP protocol is used to manage multicast groups, which implements "one-to-many" packet sending between interested systems.

- Layer 4 - Transport layer

  The transport layer is where things really start to get interesting for the systems administrator. It is at the transport layer that the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP) are defined. The TCP and UDP protocols allow data to be sent from one system to another using simple "socket" APIs that make it just as easy to send text across the globe as it is to write to a file on a local disk - a technological miracle that is often taken for granted.

- Layer 5 - Session layer

  The purpose of the session layer is to provide a mechanism for ongoing conversations between devices using application-layer protocols. Notable Layer 5 protocols include Transport Layer Security / Secure Sockets Layer (TLS/SSL) and, more recently, Google's SPDY protocol.

- Layer 6 - Presentation layer

  The job of the presentation layer is to handle data encoding and decoding as required by the application. An example of this function is the Multipurpose Internet Mail Extensions (MIME) protocol, used to encode things other than unformatted ASCII text into email messages. Both the session layer and the presentation layer are often neglected when discussing TCP/IP because many application-layer protocols implement the functionality of these layers internally.

- Layer 7 - Application layer

  The application layer is where most of the interesting work gets done, standing on the shoulders of the layers below. It is at the application layer that we see protocols such as Domain Name System (DNS), HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), and Secure SHell (SSH). The various application-layer protocols are at the core of a good systems administrator's knowledge base.

TCP/IP (ARPA) 4-layer model

When the ARPAnet project spawned TCP/IP, which became the dominant network protocol stack used on the Internet, it was envisioned with a much simpler 4-layer stack (the ISO layers roughly map to the ARPA model as indicated below.) The elements of this stack from the lowest to highest are as follows:

- Link Layer

  Basically a combination of the ISO Layer 1 (physical) and Layer 2 (data link) layers. This layer covers the physical network card in the computer (and the transmission medium between the computers) as well as the device driver in the operating system of the computer. As an example, both Ethernet and MAC are covered in this layer.

- Network Layer

  This layer maps to the ISO's Layer 3, and covers the movement of packets between networks. For this reason, it is also called the "Internet Layer", and the main protocol used in this layer is named the "Internet Protocol", or "IP" as it is commonly referred to. As discussed above, the ICMP and IGMP protocols are also included in this layer.

- Transport Layer

  This layer maps to the ISO's Layer 4, and covers the creation, management and teardown of "virtual circuits" or "flows" between end hosts. There are two different protocols in use at this layer as discussed above in the ISO Layer 4 section, namely, TCP and UDP.

- Application Layer

  This layer maps to the ISO's Layer 5 through Layer 7, and covers the application processes that use the network to communicate.

IP Addressing

IPv4

Internet Protocol Version 4 (IPv4) is the fourth version of the Internet protocol, the first version to be widely deployed. This is the version of the protocol you're most likely to encounter, and the default version of the IP protocol in Linux.

IPv4 uses a 32-bit address space most typically represented in 4 dotted decimal notation, each octet contains a value between 0-255, and is separated by a dot. An example address is below:

    10.199.0.5

There are several other representations, like dotted hexadecimal, dotted octal, hexadecimal, decimal, and octal. These are infrequently used, and will be covered in later sections.

IPv6

The Internet has experienced tremendous growth from its beginnings in research at universities to its use in e-commerce and other applications today. As a result, more and more IPv4 addresses were given out to enable users to connect to the Internet. However, despite the number of IPv4 addresses being a large number (4294967296), they are slowly running out.

Internet Protocol Version 6 (IPv6) is the latest version of the Internet protocol which aims to address the IPv4 address exhaustion problem.

Probably the most obvious difference of IPv6 to IPv4 is the representation of addresses. Unlike IPV4, IPv6 uses a hexadecimal format to represent a 128-bit address. The format is grouped into 8 groups of four hexadecimal digits separated by a colon. For example:

    2001:0DB8:0000:CBAD:4321:0000:0000:1234

Leading zeroes may be left off of a group in the address. Whole groups of zeros may be left out entirely, but only once. Generally, the longest run of zeros is dropped, but any run may be. Using these rules, the above address can also be represented in either of these ways:

    2001:DB8::CBAD:4321:0000:0000:1234 2001:DB8:0000:CBAD:4321::1234

TCP vs UDP

Both TCP **RFC 793** and UDP **RFC 768** provide data transfer between processes through ports. These process ports can be on the same computer or separate computers connected by a network. TCP provides the following: reliability, flow control, and connections (see Example Difference 1 below). UDP is less feature-rich, it does its work with a header that only contains a source port, destination port, a length, and a checksum. TCP provides its capabilities by sending more header data, more packets between ports and performing more processing. UDP requires less header data in the individual packets and requires fewer packets on the network to do its work. UDP does no bookkeeping about the fate of the packets sent from a source. They could be dropped because of a full buffer at a random router

between the source and destination and UDP wouldn't account for it in itself (other monitoring systems can be put in place to do the accounting, however that is beyond the UDP protocol).

The choice of protocols to use is often based on whether the risk of losing packets in real-time without immediate alerting is acceptable. In some cases UDP may be acceptable, such as video or audio streaming where programs can interpolate over missing packets. However, TCP will be required due to its reliable delivery guarantee in systems that support banking or healthcare.

- Example 1

  The TCP protocol requires upfront communication and the UDP protocol does not. TCP requires an initial connection, known as the "three way handshake", in order to begin sending data. That amounts to one initial packet sent between ports from initiator of the communication to the receiver, then another packet sent back, and then a final packet sent from the initiator to the receiver again. All that happens before sending the first byte of data. In UDP the first packet sent contains the first byte of data.

- Example 2

  TCP and UDP differ in the size of their packet headers. The TCP header is 20 bytes and the UDP header is 8 bytes. For programs that send a lot of packets with very little data, the header length can be a large percentage of overhead data (e.g. games that send small packets about player position and state).

Subnetting, netmasks and CIDR

A subnet is a logical division of an IP network, and allows the host system to identify which other hosts can be reached on the local network. The host system determines this by the application of a routing prefix. There are two typical representations of this prefix: a netmask and Classless Inter-Domain Routing (CIDR).

Netmasks typically appear in the dotted decimal notation, with values between 0-255 in each octet. These are applied as bitmasks, and numbers at 255 mean that this host is not reachable. Netmask can also be referred to as a Subnet Mask and these terms are often used interchangeably. An example IPv4 address with a typical netmask is below:

IP Address	Netmask
192.168.1.1	255.255.255.0

CIDR notation is a two-digit representation of this routing prefix. Its value can range between 0 and 32. While having long been a staple in network equipment and used mainly by network engineers, CIDR notation can now be found in Linux (iproute's *ip* command uses CIDR notation). In addition, CIDR notation is often encountered in discussions, as it's quicker and simpler to reference than a netmask. Below is the same example as above with CIDR notation:

IP Address	CIDR
192.168.1.1	/24

As a server administrator, it is helpful to understand subnetting so as to communicate more effectively with network engineers, and to design networks yourself. There are many guides online for learning how to subnet, each with different approaches and tricks. A full subnetting guide is out of the scope of this document.

Classful addressing

You may hear people refer to networks as "Class A", "Class B", or "Class C". This is known as "Classful Addressing" and has been deprecated for decades, thanks to the introduction of CIDR in 1993 (**RFC 1519**). Classful addressing has the drawback of assuming that a network is drawn on very large boundaries. For example, in the table in the next section, each block would be a single network. One can see how using an entire /16 (65534 hosts) when only five hosts are needed would be wasteful. As such, CIDR was created, allowing people to create subnets only as large as required. For these reasons, one should not refer to any particular network block as a "Class X" (eg, 10.0.0.0/28 should not be called a Class A network) as it is misleading at best, and incorrect at worst. You should always use CIDR notation to ensure accuracy.

Private address space (RFC 1918)

Certain ranges of addresses were reserved for private networks. Using this address space you cannot communicate with public machines without a NAT gateway or proxy. There are three reserved blocks:

First Address	Last Address	Netmask	CIDR
10.0.0.0	10.255.255.255	255.0.0.0	/8
172.16.0.0	172.31.255.255	255.240.0.0	/12
192.168.0.0	192.168.255.255	255.255.0.0	/16

Static routing

NAT

Network Address Translation, or NAT, is a technology that allows multiple internet-connected devices to share one common **public** IP address, while still retaining unique, individual **private** IP addresses. The distinction between public and private is integral to the understanding of the service that NAT provides and how it works.

In our *Sample Network* we can see that two firewall machines sit between the Internet and the other hosts in the network; traffic going in and out of the network will pass through these firewall machines. The addresses assigned to the firewalls (10.10.10.1 and 10.10.10.2) are private IPs and are visible to just the hosts within the network. A device on the external Internet will, instead, see the public IP address for each firewall. It's important to note that none of the hosts within the network will have a public IP address, except for the firewalls and the DNS servers, since they are the only parts of the network that directly communicate with the external Internet.

When a device behind a NAT-enabled router sends a packet, the source IP address on the packet is the device's local, private IP address. If the packet is going outside the local network, it will pass through a router, which will modify the source IP address to its own public IP address. When a response for the packet is received by the router, it needs to ensure that the response can be forwarded to the host that sent the packet in the first place. To do this, the router maintains a **Translation Table**. This table maps a device's IP address and port to a port on the router itself. The router's public IP address and the correct port number from the table are used as the source IP and port on the packet and then sent to the destination.

These maps are temporary and exist on a per-connection basis. This means that each connection opened by a device will have a unique port number on the device and a unique port number on the router as well. This port on the router is used as the public port for that connection. Once the connection terminates, the router is free to assign that port to another connection. However, the total number of available ports is limited to 65,536, so it is entirely possible that a router has no more free ports and cannot assign a new NAT address. This is commonly referred to as port exhaustion.

Similar to port exhaustion, timeouts can also affect the router's ability to assign new NAT addresses. Each entry in the translation table has a timeout value which refers to the amount of time for which that entry can remain inactive and still keep its place in the table. An entry that has remained inactive for a period of time longer than the timeout will automatically be removed, freeing up space for a new one.

Networking cable

There are two main types of network cable in use today, namely copper and fiber-optic.

Copper

The most common type of network cables are what is known as "unshielded twisted pair" cables. They use 4 sets of twisted pairs of copper, relying on the twist with differential signaling to prevent noise and signal propagation between the pairs. The four pairs of twisted copper wires are encased in a plastic sheath.

There are different standards for copper network cables set by the Telecommunications Industry Association (TIA) and the International Organization for Standardization (ISO). Both organizations use the same naming convention ("Category __") for the components, but unfortunately differ on the naming for the cable standards. The most common reference is the TIA's, and the category designation is usually shortened to "Cat", so you'll hear references to "Cat5" or "Cat6" cable.

Copper Cable Standards

- Category 5e ("Cat5", ISO class D)
- Category 6 ("Cat6", ISO class E)
- Category 6A ("Cat6A", ISO class Ea)

Fiber

Fiber is a generic term that refers to optical transport mediums. It comes in several types, all of which look identical but are generally incompatible.

Multimode vs Single Mode

Single-mode fiber has a small core diameter, which only allows one (a single) mode of light to be transmitted through the fiber. Using a single mode of light completely eliminates the possibility of light dispersion and associated signal loss, and so is used mainly for long-haul runs, such as the cables that run between buildings and cities. However, since single-mode fiber can only transmit one wavelength of light at a time, it typically involves much more expensive light generation sources (i.e., laser diode transmitters) and is very expensive to produce.

Multimode fiber has a larger core diameter (either 50u or 62.5u) and can therefore carry multiple modes ("multimode") of light, which can be used to transmit much more information during a given timeslice. The drawback is that carrying multimode lightwaves causes light dispersion and associated signal loss, which limits its effective distance. Multimode is a less expensive fiber optic cable, that is typically useable with lower cost optical components. It is very common to see it used for building intra-building backbones, and system/switch to switch applications.

Multimode Fiber Standards

Multimode cables have classifications much like the copper cables discussed above; these are known as "Optical Multimode" (OM) classes. The four designations are:

- OM1 - a "legacy" fiber class, the core being 62.5u, and cladding being 125u. The bandwidth that can be carried ranges from 160 to 500 MHz.
- OM2 - a "legacy" fiber class, the core being 50u, and cladding being 125u. The bandwidth that can be carried is 500 MHz.
- OM3 - a "modern" fiber class, the core being 50u, and cladding being 125u. The bandwidth that can be carried ranges from 1500 to 2000 MHz.
- OM4 - a "modern" fiber class, the core being 50u, and cladding being 125u. The bandwidth that can be carried ranges from 3500 to 4700 MHz.

Optical Connector Types

LC and SC connectors are the two most common type of fiber connectors you will use.

LC stands for "Lucent Connector", but is also referred to as "Little Connector". They are typically used for high-density applications, and are the type of connector used on SFPs or XFPs. Typically the connector is packaged in a duplex configuration with each cable side by side, and have a latch mechanism for locking.

SC stands for "Subscriber Connector", but are also known as "Square Connector", or "Standard Connector". This is the type of connector typically used in the telecom industry. They have a larger form factor than the LC connectors, and can be found in single and duplex configurations. SC connectors have a push/pull locking mechanism, and because of this, are also colloquially known as "Stab-and-Click" connectors.

Transceivers

The variety in optical fiber makes for a correspondingly large variety in optical fiber interface standards. Different interface types will impose different requirements on the fiber used and the length of the connection.

If optical fiber interfaces were incorporated directly into network equipment, the number of models made by the manufacturer would have to be multiplied by the number of interface standards in existence. For this reason, modern network hardware rarely incorporates such interfaces directly. Instead, pluggable transceiver modules are used as a layer of indirection between medium-dependent and medium-independent interfaces. This allows a transceiver slot to be provided supporting any desired interface standard, whether copper or fiber. There are some limitations to this, detailed below.

Various module types have been introduced over the years:

Name	Introduced	Speed	Size
GBIC	1995	1 Gb/s	Large
SFP	2001	1 Gb/s	Small
XENPAK	2001	10 Gb/s	Large
XFP	2002	10 Gb/s	Small
SFP+	2006	10 Gb/s	Small
QSFP	2006	40 Gb/s	Small
CFP	2009	100 Gb/s	Large

There are a large number of compatibility issues with such modules. Some compatibility issues cause problems between two ends of a link; others cause problems between a module and its host device.

- Transceivers are not generally compatible with lower speed versions of the same standard. A 1000BASE-T Ethernet card can interface with a 10BASE-T card, but a 1 Gb/s fiber transceiver cannot interface with a 10 or 100 Mb/s transceiver. In the case of fiber, this is generally due to the different wavelengths used; but even many copper transceivers do not support lower speeds, although some do. You should assume that any transceiver will only support the exact interface for which it is designed unless specified otherwise.
- Modules are only made for the speed targeted by a format. For example, SFP+ modules are only made for 10 Gb/s standards, and not for lower speeds.
- Some equipment may accept SFP modules in SFP+ slots, but this is not universal.
- Vendor lock-in is widely practiced. Equipment may refuse to operate with the modules made by a different manufacturer. Workarounds are generally available, but this may complicate support or warranty arrangements.

These issues can create pathological cases. Suppose you have two switches which should be connected to one another. One is connected via a 1 Gb/s transceiver to fiber. The other only has SFP+ slots. If these slots also support SFP modules, then a 1 Gb/s SFP transceiver can be used, but if they do not, interconnection is impossible: all SFP+ modules target 10 Gb/s, and fiber transceivers do not support lower speeds.

Twinax

Networking 201

VLANs, 802.1q tagging

Sometimes it is advantageous to break up a single Layer 2 broadcast domain into multiple Layer 2 broadcast domains. As we had previously discussed, Layer 2 is addressed by MAC addresses, which are flat address space (non-routable) constructs; this creates a Layer 2 broadcast domain, also called a "segment" or "LAN" (local area network.) You can then map one or more IP networks (Layer 3 addressing) onto a Layer 2 segment. If you map more than one Layer 3 network to a segment, you can get into some headaches, such as having to address multiple IP addr's on various host and router interfaces (one for each network) with the attendant confusion, and also that broadcasts produced by each IP network (i.e. ARP requests) multiply, and bandwidth is consumed. Therefore, most networking professionals agree that it's best to only map one IP network to one Layer 2 segment.

So, if multiple Layer 3 networks need to be used, instead of using multiple switches for each Layer 2 segment, it is possible to divide a single VLAN-capable switch into two or more "virtual" switches. This is possible due to the concept of *VLAN*, which stands for "Virtual LAN".

To communicate between the Layer 3 networks mapped to their individual VLANs, packets need to pass through a router or other Layer 3 device. If one has a switch with many VLANs implemented on it, either of these two methods could work:

1. Devote a switch (and respective router) interface per VLAN, with a single link for each VLAN. This is the simplest method, but the most wasteful of (expensive) switch and router interfaces.
2. "Trunk" multiple VLANs over one physical link, which only requires a single interface per side (switch to router, or server to switch.) This method multiplexes multiple VLANs over the one link by "tagging" the frame with a 4-byte VLAN identifier field which is inserted after the destination MAC field in the Ethernet header. This tagging format is officially defined by the IEEE's 802.1q protocol. The tag field insertion happens at the sending trunk interface (which is a part of the sending station's VLAN), and the tag is stripped of by the receiving trunk interface, and the frame placed on the proper VLAN on the receiving device.

The VLAN identifier must be between 1 and 4094 inclusive. The 802.1q tag also provides for a 3-bit priority field and 1-bit discard eligibility field which can be used for QoS applications. The trunking and priority functions of 802.1q can be enabled and disabled separately.

Spanning Tree

As networks and broadcast domains grow and become more complex, it becomes much easier to intentionally or unintentionally create loops- multiple network paths between one or more switches. These loops may lead to a infinite forwarding of Ethernet packets, causing switches to be overwhelmed by an ever-increasing amount of packets being forwarded to all other switches in the broadcast domain. To address this concern, Spanning Tree Protocol (STP) was invented as a standard protocol used by Ethernet switches, designed to prevent such loops in broadcast domains.

To disable loops, each switch that implements Spanning Tree Protocol (STP / IEEE 802.1D) will participate in electing a "root" device where the Spanning Tree is calculated from. Once the root is chosen, all other participating switches will calculate their least-cost path for each of its ports to the root device. This is usually a port that traverses the least number of segments between the source and the root. Once each device has chosen its least-cost port, or Root Port (RP), it calculates the least-cost path for each network segment. By doing this, each switch obtains a Designated Port (DP), or a port on which it should expect to accept traffic to forward through its RP. Upon calculating these values, all other paths to the root device are disabled- marking them as Blocked Ports (BP).

A major benefit to using STP to prevent network loops is that network administrators can intentionally connect redundant paths for fault tolerance without worry of causing an infinite loop during normal operation. Should a path

in a broadcast domain fail, each participating switch will recalculate the DP and RP, modifying the status of BPs if necessary. This has the effect of repairing the broken path by routing traffic through segments around the failure.

Routing

When a network is subdivided into multiple Layer 2 broadcast domains, Layer 3 addressing enables hosts to communicate with another host in another broadcast domain.

The process of forwarding packets from one Layer 2 subdomain to another Layer 2 subdomain using Layer 3 addresses is called routing.

A Layer 3 device or router is the device responsible for this function. Routers may use many different ways to forward packets but these methods can be categorized into two types.

Static Routing

The method of using manually configured routes is called static routing.

Typically, there are three pieces of information that are needed to specify a static route:

1. the destination subnet
2. the latter's subnet mask and
3. the next hop host or outgoing interface.

Take for example the following static route configuration in Linux:

```
# ip route add 10.10.20.0/24 via 192.168.2.1
```

The above command means that packets intended for hosts in the 10.10.20.0/24 network must be forwarded to the host with ip address 192.168.2.1.

Below is another example in Cisco IOS:

```
Router(config)# ip route 10.10.20.0 255.255.255.0 Serial0/0/1
```

Instead of passing a packet to a specific ip address however, this configuration states that packets intended for the 10.10.20.0/24 network should be forwarded to the host directly connected to Serial 0/0/1 whose ip address can be anything.

Static routes do not change when the network changes but can be useful in cases where the network is small enough that it outweighs the cost of dynamic routing. It is often used in tandem with dynamic routing to specify a default route in case a dynamic route is unavailable.

Dynamic routing protocols (RIP, OSPF, BGP)

For a small network, manually configuring routes will work just fine. As a network grows larger, however, doing this can be very arduous if not infeasible. Dynamic routing solves this problem by programmatically building a routing table.

Here are some examples of dynamic routing protocols:

1. Routing Information Protocol (RIP) - RIP uses the number of routers between a router and the destination subnet as its path selection metric. When a RIP router first comes online, it sends a broadcast message and then all its neighbor routers respond back with their own routing table. This new RIP router then compares the routes in its own routing table with the routes from the other routing tables it has received. If there are routes to other subnets that are currently not in its own routing table, that route is added. More importantly, it updates routes in its own routing table if there is

a route with less hops to a destination subnet. RIP is part of a class of routing protocols called distance vector routing protocols. They are characterized by their sharing of routing tables with their neighbors. Similar protocols to RIP include Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). EIGRP is similar to RIP but uses a mix of path selection metrics such as bandwidth and delay to determine the route to a destination subnet. Unlike RIP, it only sends updates incrementally to reduce overhead.

2. Open Shortest Path First (OSPF) - OSPF is a routing protocol that calculates the shortest path to each destination subnet in the network. Routing protocols like OSPF that work this way are said to be called Link State routing protocols. To be able to know the shortest path to each destination subnet, each router should know everything about the network from links, subnets and even other routers. To do this in OSPF, each router packages routing information they know in a data structure called Link State Advertisements (LSA) and floods the whole network with this information. Fellow OSPF routers listen for LSAs, collect this data and store them in a Link State Database (LSDB). The point of all this sharing is for each router to have identical LSDBs. Once this is achieved, each OSPF router calculates routes to all subnets from its own point of view using a shortest path algorithm. These routes are added then added to the router's routing table.

3. Border Gateway Protocol (BGP) -

ACLs

Network Bonding (802.3ad / LACP link aggregation)

IOS switch configuration

Cisco devices use a command line-based configuration interface which can be accessed via RS-232, Telnet or SSH. The command line syntax is distinctive and has been loosely copied by many other vendors, so some familiarity with the Cisco-style configuration can go a long way.

When connecting via RS-232, use 9600 baud, 8 data bits, 1 stop bit, no parity, no flow control.

You can abbreviate keywords on the command line without having to press Tab, so long as they are unambiguous. You can also use Tab as you would in a UNIX shell.

While using the command line, you can type '?' at any time and receive context-sensitive help.

The command line is mode-based. The prompt tells you of your current mode. Awareness of your current mode is key to efficient operation of the command line. Here are some examples:

Prompt	Meaning
`hostname>`	'EXEC' mode, non-privileged access (like `$` on Linux)
`hostname#`	'EXEC' mode, privileged access (like `#` on Linux)
`hostname(config)#`	Global configuration mode
`hostname(config-if)#`	Interface configuration mode

EXEC mode allows you to execute imperative commands such as `ping`. The configuration mode allows you to add and remove configuration statements.

The following commands move you between modes:

Command	Mode	Effect
`enable`	EXEC	Become privileged (like `su` on Linux)
`conf t`	EXEC	Enter `(config)` mode.
`int <interface>`	config	Enter `(config-if)` mode for an interface.
`exit`	Any	Leave the current mode or logout.

Other useful commands include:

Command	Mode	Effect
`show ...`	EXEC	The subcommands of this command provide access to all available information.
`show run`	EXEC	Show the current configuration.
`show ip int brief`	EXEC	Show all interface names, IPv4 addresses, and their status.
`wr`	EXEC	Save the current configuration.
`ping <host>`	EXEC	Ping. '!' means a response, '.' means a timeout.
`no <command>`	config	Delete a configuration statement in config mode.
`do <command>`	config	Execute a command in EXEC mode from config mode.

Note that configuration changes become active as soon as they are made. The `show run` command shows the configuration which is currently in effect. This is not saved until you execute `wr`.

GRE and other tunnels

Multi-homed hosts

Similarities and differences between IPv4 and IPv6 networking

Implications of dual-stack firewalls (especially under Linux)

Multicast uses and limitations

Latency vs. Bandwidth

http://www.stuartcheshire.org/rants/Latency.html

VPNs

IPSec

SSL

Network Troubleshooting

ping

ping should always be the first step in any network-related troubleshooting session, due to the simple manner in which it works, and the information it returns. Many times, the resulting information from a ping will point you in the next direction. For example, if you notice jitter (ping responses varying wildly), you will know to start looking at layer 1 problems somewhere on the path, and that available bandwidth probably isn't an issue.

You're probably already familiar with the basic usage of ping, but there are some really handy options that can used (these are on Linux, but also exist in most versions of ping, but under different flags). Here are a couple of my most-often-used options:

-I - Change the source IP address you're pinging from. For example, there might be multiple IP addresses on your box, and you want to verify that a particular IP address can ping another. This comes in useful when there's more than just default routes on a box.

-s - Set the packet size. This is useful when debugging MTU mismatches, by increasing the packet size. Use in conjuction with the -M flag to set MTU Path Discovery hint.

An example of using ping to test MTU:

```
user@opsschool ~$ ping -M do -s 1473 upstream-host
PING local-host (local-host) 1473(1501) bytes of data.
From upstream-host icmp_seq=1 Frag needed and DF set (mtu = 1500)
From upstream-host icmp_seq=1 Frag needed and DF set (mtu = 1500)
From upstream-host icmp_seq=1 Frag needed and DF set (mtu = 1500)
```

I've used the *-M do* option to set the 'Don't Fragment' (DF) flag on the packet, then set the packet size to 1473. With the 28 bytes of overhead for Ethernet, you can see the total packet size becomes 1501–just one byte over the MTU of the remote end. As you can see from the example, since the DF flag is set and the packet needs to fragment, it spits back an error, and helpfully tells us what the MTU size is on the other end. ping can be used to determine Path MTU (the smallest MTU size along a path), but other tools are better for that (see below).

telnet

While telnet daemons are a big no-no in the wild (unencrypted traffic), the telnet client utility can be used to test whether TCP connections can be made on the specified port. For example, you can verify a TCP connection to port 80 by connecting via telnet:

```
user@opsschool ~$ telnet yahoo.com 80
Trying 98.138.253.109...
Connected to yahoo.com.
Escape character is '^]'.
```

A connection failure would look like this (using port 8000, since nothing is listening on that port):

```
user@opsschool ~$ telnet yahoo.com 8000
Trying 98.138.253.109...
telnet: connect to address 98.138.253.109: Connection timed out
```

You can also send raw data via telnet, allowing you to verify operation of the daemon at the other end. For example, we can send HTTP headers by hand:

```
user@opsschool ~$ telnet opsschool.org 80
Trying 208.88.16.54...
Connected to opsschool.org.
Escape character is '^]'.
GET / HTTP/1.1
host: www.opsschool.org
```

The last two lines are the commands sent to the remote host.

Here's the response from the web server:

```
HTTP/1.1 302 Found
Date: Fri, 26 Dec 2014 14:55:45 GMT
Server: Apache
Location: http://www.opsschool.org/
Vary: Accept-Encoding
Content-Length: 276
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.opsschool.org/">here</a>.</p>
<hr>
```

```
<address>Apache Server at www.opsschool.org Port 80</address>
</body></html>
Connection closed by foreign host.
```

Here we passed the bare minimum required to initiate an HTTP session to a remote web server, and it responded with HTTP data, in this case, telling us that the page we requested is located elsewhere.

Note that the port you're connecting to might the port for HTTP, but it could be something other than an HTTP daemon running. Nothing prevents a service from running on a port other than it's usual one. In effect, you could run a SMTP daemon on port 80, and an HTTP daemon on port 25. Testing TCP connections with telnet would verify TCP operation, but you still would not have a working web server on port 80. Since the scope of this section is focused only on the networking aspect, see the other sections of OpsSchool for troubleshooting daemon operation and Linux troubleshooting.

iproute / ifconfig

ifconfig is ubiquitous and a mainstay of any network-related work on Linux, but it's actually deprecated in RHEL7 (the net-tools package which contains *ifconfig* isn't included in RHEL 7/CentOS 7 by default) and many major distributions include iproute by default. The ifconfig man page also recommends using the iproute package. All examples used below will use iproute, and will cover only the basics of troubleshooting. It's highly recommended to play around and see what you can find. The ip man page contains a wealth of knowledge on the tool.

ip addr show

Show all IP addresses on all interfaces. Many options can be passed to filter out information. This will show several important pieces of information, such as MAC address, IP address, MTU, and link state.

```
user@opsschool ~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:8a:6d:07 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
    inet6 fe80::a00:27ff:fe8a:6d07/64 scope link
       valid_lft forever preferred_lft forever
```

ip route

Show all routes on the box.

```
user@opsschool ~$ ip route
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15
169.254.0.0/16 dev eth0  scope link  metric 1002
default via 10.0.2.2 dev eth0
```

ss

ss is the replacement for *netstat*, which is obsolete according to the netstat man page. While most distributions will probably have netstat available for some time, it is worthwhile to get used to using ss instead, which is already included in the iproute package.

ss is very useful for checking connections on a box. ss will show SOCKET, TCP, and UDP connections, in various connection states. For example, here's ss showing all TCP and UDP connections in the LISTEN state, with numeric representation. In other words, this shows all daemons listening on UDP or TCP with DNS and port lookup disabled.

```
user@opsschool ~$ ss -tuln
Netid  State      Recv-Q Send-Q         Local Address:Port         Peer Address:Port
tcp    LISTEN     0      128                    *:80                      *:*
tcp    LISTEN     0      50                     *:4242                    *:*
tcp    LISTEN     0      50                    :::4242                   :::*
tcp    LISTEN     0      50                     *:2003                    *:*
tcp    LISTEN     0      50                     *:2004                    *:*
tcp    LISTEN     0      128                   :::22                     :::*
tcp    LISTEN     0      128                    *:22                      *:*
tcp    LISTEN     0      100                    *:3000                    *:*
tcp    LISTEN     0      100                  ::1:25                    :::*
tcp    LISTEN     0      100              127.0.0.1:25                    *:*
tcp    LISTEN     0      50                     *:7002                    *:*
```

There are a few things to note in the output. Local address of * means the daemon is listening on all IP addresses the server might have. Local address of 127.0.0.1 means the daemon is listening only to the loopback interface, and therefore won't accept connections from outside of the server itself. Local address of ::: is the same thing as *, but for IPv6. Likewise, ::1 is the same as 127.0.0.1, but for IPv6. In this example, we used the flags *-tuln*, which just happens to be one of my more-often used sets of flags.

By default, ss shows only non-listening TCP connections:

```
user@opsschool ~$ ss

State      Recv-Q Send-Q                                  Local Address:Port
ESTAB      0      0                                           10.0.2.15:ssh
```

ss has many more useful flags than just these, which you can find in the ss man page.

traceroute

If you've familiarized yourself with the basics of networking, you'll know that networks are comprised of many different routers. The internet is mostly a messy jumble of routers, with multiple paths to end points. traceroute is useful for finding connection problems along the path.

traceroute works by a very clever mechanism, using UDP packets on Linux or ICMP packets on Windows. traceroute can also use TCP, if so configured. traceroute sends packets with an increasing TTL value, starting the TTL value at 1. The first router (hop) receives the packet, then decrements the TTL value, resulting in the packet getting dropped since the TTL has reached zero. The router then sends an ICMP Time Exceeded back to the source. This response indicates to the source the identity of the hop. The source sends another packet, this time with TTL value 2. The first router decrements it as usual, then sends it to the second router, which decrements to zero and sends a Time Exceeded back. This continues until the final destination is reached.

An example:

```
user@opsschool ~$ traceroute google.com
traceroute to google.com (173.194.123.39), 30 hops max, 60 byte packets
 1  (redacted) (redacted)  1.153 ms  1.114 ms  1.096 ms
 2  192.241.164.253 (192.241.164.253)  0.226 ms 192.241.164.241 (192.241.164.241)  3.267 ms 192.241.1
 3  core1-0-2-0.lga.net.google.com (198.32.160.130)  0.291 ms  0.322 ms 192.241.164.250 (192.241.164
 4  core1-0-2-0.lga.net.google.com (198.32.160.130)  0.290 ms 216.239.50.108 (216.239.50.108)  0.980
 5  216.239.50.108 (216.239.50.108)  1.166 ms 209.85.240.113 (209.85.240.113)  1.143 ms  1.358 ms
 6  209.85.240.113 (209.85.240.113)  1.631 ms lga15s47-in-f7.1e100.net (173.194.123.39)  0.593 ms  0
```

mtr

mtr is a program that combines the functionality of *ping* and *traceroute* into one utility.

```
user@opsschool ~$ mtr -r google.com
HOST: opsschool                  Loss%   Snt   Last   Avg  Best  Wrst StDev
1. (redacted)                     0.0%    10    0.3   0.4   0.3   0.5   0.1
2. 192.241.164.237                0.0%    10    0.3   0.4   0.3   0.4   0.0
3. core1-0-2-0.lga.net.google.c   0.0%    10    0.4   0.7   0.4   2.8   0.8
4. 209.85.248.178                 0.0%    10    0.5   1.1   0.4   6.2   1.8
5. 72.14.239.245                  0.0%    10    0.7   0.8   0.7   1.4   0.2
6. lga15s46-in-f5.1e100.net       0.0%    10    0.5   0.4   0.4   0.5   0.0
```

mtr can be run continuously or in report mode (-r). The columns are self-explanatory, as they are the same columns seen when running traceroute or ping independently.

Reading mtr reports can be a skill in itself, since there's so much information packed into them. There are many excellent in-depth guides to mtr that can be found online.

iftop

iftop displays bandwidth usage on a specific interface, broken down by remote host. You can use filters to filter out data you don't care about, such as DNS traffic. iftop is not available in the base reposities for RHEL/CentOS or Ubuntu, but is available in EPEL, and the Universe repository, respectively.

In this example, iftop is listening only to the eth0 interface, and for purposes of this document, is also using the -t option, which disables the ncurses interface (for your use, you won't need -t). This box is a very low-traffic VM, so there's not much here, but it does give a sense of what information is available via the tool.

```
user@opsschool ~$ sudo iftop -i eth0 -t
interface: eth0
IP address is: 10.0.2.15
MAC address is: 08:00:27:ffffff8a:6d:07
Listening on eth0
   # Host name (port/service if enabled)            last 2s   last 10s   last 40s cumulative
--------------------------------------------------------------------------------------------
   1 10.0.2.15                                 =>     804b       804b       804b       201B
     google-public-dns-a.google.com            <=     980b       980b       980b       245B
   2 10.0.2.15                                 =>     352b       352b       352b        88B
     10.0.2.2                                  <=     320b       320b       320b        80B
--------------------------------------------------------------------------------------------
Total send rate:                                    1.13Kb     1.13Kb     1.13Kb
Total receive rate:                                 1.27Kb     1.27Kb     1.27Kb
Total send and receive rate:                        2.40Kb     2.40Kb     2.40Kb
--------------------------------------------------------------------------------------------
Peak rate (sent/received/total):                    1.12Kb     1.27Kb     2.40Kb
Cumulative (sent/received/total):                     289B       325B       614B
============================================================================================
```

iperf

iperf is a bandwidth testing utility. It consists of a daemon and client, running on separate machines.

This output shows from the client's side:

```
user@opsschool ~$ sudo iperf3 -c remote-host
Connecting to host remote-host, port 5201
```

```
[  4] local 10.0.2.15 port 45687 connected to x.x.x.x port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   548 KBytes  4.48 Mbits/sec    0   21.4 KBytes
[  4]   1.00-2.00   sec   503 KBytes  4.12 Mbits/sec    0   24.2 KBytes
[  4]   2.00-3.00   sec   157 KBytes  1.28 Mbits/sec    0   14.3 KBytes
[  4]   3.00-4.00   sec  0.00 Bytes  0.00 bits/sec    0   14.3 KBytes
[  4]   4.00-5.00   sec   472 KBytes  3.88 Mbits/sec    0   20.0 KBytes
[  4]   5.00-6.00   sec   701 KBytes  5.74 Mbits/sec    0   45.6 KBytes
[  4]   6.00-7.00   sec   177 KBytes  1.45 Mbits/sec    0   14.3 KBytes
(snip)
```

Some of the really handy options:

-m - Use the maximum segment size (the largest amount of data, in bytes, that a system can support in an unfragmented TCP segment). This option will use the default size for the particular network media in use (eg, Ethernet is 1500 bytes).

-M - Set MSS, used in conjuction with the previous -m option to set the MSS to a different value than default. Useful for testing performance at various MTU settings.

-u - Use UDP instead of TCP. Since UDP is connectionless, this will give great information about jitter and packet loss.

tcpdump

For occasions where you need to get into the nitty-gritty and look at actual network behavior, tcpdump is the go-to tool. tcpdump will show raw connection details and packet contents.

Since one could devote entire pages to the usage of tcpdump, it is recommended to search online for any one of the many great guides on tcpdump usage.

Troubleshooting layer 1 problems

Layer one problems (physical) deserve their own section, because of how cryptic they can seem without a solid understanding of the causes and symptoms. There are several data points that fall under layer one.

Most layer problems are usually traced back to one of the following causes:

- Bad cabling
- Mismatched duplex/speed
- Electromagnetic interference (EMI)
- Network congestion

To find the metrics below, use either *ip -s link*, or *ethtool -S*.

Example output of *ip -s link show eth0*:

```
user@opsschool ~$ ip -s link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 04:01:1c:dc:82:01 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    150285334  221431013 0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    999121259  42410636 0       0       0       0
```

Here we can see that the link is up (*state UP*), the MAC address (*04:01:1c:dc:82:01*), bytes, packets, errors, drops, overruns, multicast, carrier, and collisions, for both RX (receive) and TX (transmit).

One thing you may notice is that overrun and mcast are only on RX, and carrier/collisions are only on TX. This is because overruns and multicast only affect the receiving side, while carrier and collisions only affect the transmitting side. The reason for this will be made clear when described below.

Example output of *ethtool eth0*:

```
user@opsschool ~$ sudo ethtool eth0
Settings for eth0:
  Supported ports: [ TP ]
  Supported link modes:   10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Advertised link modes:  10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Full
  Advertised pause frame use: No
  Advertised auto-negotiation: Yes
  Speed: 1000Mb/s
  Duplex: Full
  Port: Twisted Pair
  PHYAD: 0
  Transceiver: internal
  Auto-negotiation: on
  MDI-X: Unknown
  Supports Wake-on: umbg
  Wake-on: d
  Current message level: 0x00000007 (7)
             drv probe link
  Link detected: yes
```

As we can see, ethtool gives a lot of information, and with additional flags, there's even more. ethtool is especially useful for looking into layer 1 configuration, due to how much information it can provide. A particularly useful option is *-S*, which shows all the same metrics as the *ip* command mentioned above, plus many, many more:

```
user@opsschool ~$ sudo ethtool -S eth0
NIC statistics:
     rx_packets: 20831
     tx_packets: 11160
     rx_bytes: 14654723
     tx_bytes: 3637509
     rx_broadcast: 0
     tx_broadcast: 9
     rx_multicast: 0
     tx_multicast: 11
     rx_errors: 0
     tx_errors: 0
     tx_dropped: 0
     multicast: 0
     collisions: 0
     rx_length_errors: 0
     rx_over_errors: 0
     rx_crc_errors: 0
     rx_frame_errors: 0
     rx_no_buffer_count: 0
     rx_missed_errors: 0
     tx_aborted_errors: 0
     tx_carrier_errors: 0
```

```
tx_fifo_errors: 0
tx_heartbeat_errors: 0
tx_window_errors: 0
tx_abort_late_coll: 0
tx_deferred_ok: 0
tx_single_coll_ok: 0
tx_multi_coll_ok: 0
tx_timeout_count: 0
tx_restart_queue: 0
rx_long_length_errors: 0
rx_short_length_errors: 0
rx_align_errors: 0
tx_tcp_seg_good: 304
tx_tcp_seg_failed: 0
rx_flow_control_xon: 0
rx_flow_control_xoff: 0
tx_flow_control_xon: 0
tx_flow_control_xoff: 0
rx_long_byte_count: 14654723
rx_csum_offload_good: 0
rx_csum_offload_errors: 0
alloc_rx_buff_failed: 0
tx_smbus: 0
rx_smbus: 0
dropped_smbus: 0
```

Common Network metrics

RX/TX errors

These counters appear to be aggregates of various other counters, and information on exactly what's included in them is sparse. An increase of these counters is an indication that *something* is wrong, but they're too general to determine the exact cause. If you see these increasing, it will be more beneficial to check *ethtool -S* and see exactly which counters are increasing.

Cyclic Redundancy Check (CRC)

CRC is a sort of checksum. CRC errors can be a symptom for many different issues, such as a bad cable, noisy lines, and more.

Drops

Drops occur when the network link is simply too saturated. The solution is to increase the bandwidth available, perhaps by upgrading the network link, or implementing some form of traffic limitation such as Quality-of-Service (QoS). Traffic limitation/shaping is out of the scope of this section, and should be considered an advanced topic. You can verify the saturation by checking your graphs for RX/TX bytes and see how much data is going through.

Overruns

Overruns occur when the network device is receiving data faster than the kernel driver can process it. Overruns sound similar to drops, but overruns have to do with the kernel's ability to process data, rather than the network link itself.

The solution is either to replace the network device with a more capable one, or lower the amount of data coming in to the network device.

Carrier

Carrier errors occur when the link signal is having issues. In the absence of collisions, this can usually be attributed to a bad cable or network device. If collisions are also increasing, check the duplex settings.

Collisions

Collisions occur when there is a duplex or speed mismatch. Verify duplex and speed at both ends of the cable (at the switch and at the server). Auto-negotiation being enabled on one end but not on the other is often a cause of duplex/speed mismatch. Some devices don't play nicely with each other when auto-negotiation is enabled on both sides, so be sure to verify what auto-negotiation resulted in.

Electromagnetic interference (EMI)

The symptoms of electromagnetic interference can be seen by an increase of CRC errors. Electromagnetic interference can be caused by any number of things: power cables nearby, mechanical systems (such as HVAC units), etc. Since a visual inspection will often reveal electromagnetic interference culprits easier than checking metrics, it's simpler to do a visual inspection first if you suspect this. The problem can sometimes be solved/alleviated by replacing the cable with STP (Shielded Twisted Pair), but it's usually better to just relocate the network cable. Only copper cabling is suspectible to EMI. Fiber is immune to EMI, as signals are transmitted as light instead of current.

Testing copper cabling

A link tester is an inexpensive device made by many different vendors, for the purpose of verifying that a cable's functionality. Link testers only verify a small amount of criteria, mainly that there are no opens or shorts. A link tester, however, is different from a cable certifier, which will verify full compliance with TIA/EIA and ISO cable standards, including metrics for crosstalk allowance, cable length, current loss, and many others.

If you need to test cable used in low-impact environments (e.g., wall jack to desktop), a link tester is usually sufficient. If you need to test cable used in high-impact environments (e.g., datacenter), opt for a cable certifier. Reputable cable vendors will have already run certification tests on cable, so this is only useful if you are routinely making your own cable (which is not advised under most circumstances).

Many vendors sell great testers, with Fluke Networks being one of the primary vendors.

Fiber errors

Issues with fiber cable generally fall into two categories:

- Dirty/Scratched fiber
- Bad optic/transceiver

The symptoms for both are very similar: intermittent RX/TX and CRC errors indicate a dirty or scratched fiber, while persistent RX/TX and CRC errors indicate a bad optic/transceiver.

Differences in perspective: network engineering and systems administration

Network engineering and systems administration have a tendency to speak different languages, due to the divide in skillsets and lack of overlap.

A good way to view how network engineering sees the technology differently is consider the OSI model: network engineering is focused primarily on layers 1, 2, and 3, with the occasional venture into the higher layers when certain routing protocols are involved (eg, BGP peering sessions operate over TCP). System administrators, on the other hand, are typically more concerned with layers 4 through 7, with the occasional venture into layer 3 for IP addressing. If one considers the perspective of the other, empathy is understanding comes easier, and anticipating what the other side expects becomes straightforward.

As such, here are a few tips on how the two specializations see the same technology:

1. Network engineers output in bits-per-second (bps), while many server-specific utilities output in bytes-per-second (Bps). As such, be sure when you're sending throughput data to network engineering that it's in bits-per-second. Your monitoring tools will usually do this for you (look for a config option). In the occasion you need to do it by hand, and you're working in bytes, simply multiply by eight to get bits. For more information on conversions, wikipedia has a good article on unit measurements. Alternatively, use an online calculator.
2. Systems administrators often don't worry about network topology since it so often "just works". However, in some cases, especially situations where you're troubleshooting hosts across the open internet, you may run into something called an 'asymmetrical path', that is, the routing is using a different path out than it does coming back in. In such situations, one path might have issues, while the other path is perfectly fine. For this reason, when sending issue reports to a network engineer, be sure to send a *traceroute/mtr* report from *both* directions. This situation is not common on internal networks, but can be on the open Internet. You may also run into this situation when you have more complex routing set up on the local server.
3. A simple *ping* sometimes isn't enough for an issue report, for the simple reason that it contains so little information. A *traceroute/mtr* report is better. If there's suspected throughput issues, try to get an *iperf* report as well. Also include an interface configuration report, showing MTU, IP address, and MAC address of the relevant interface(s).

Common services

System daemons 101

NTP

Kernel daemons

`klogd`, etc

Filesystem daemons

`xfslogd`, etc

udevd

crond

atd

sshd

DNS 101

The Domain Name System is a very central part of how the Internet we use today works. Before the introduction of DNS, networked computers were referenced solely by IP address. After a while, this became confusing to remember and use on a daily name, thus DNS was born.

A Brief History of DNS

As the popularity of the Internet grew, and more networked computers came online, there was an increasing need to be able to reference remote machines in a less-confusing way than solely by IP address. With a small enough network, referencing machines by IP address alone can work absolutely fine. The addition of descriptive names, however, makes referencing machines much easier.

The first example of DNS was a `HOSTS.TXT` file created by staff running ARPANET. ARPANET staff would amend this global `HOSTS.TXT` file on a regular basis, and it was distributed to anyone on the Internet who wanted to use it to reference machines by name rather than by number. Eventually, as the Internet grew, it was realized that a more automated system for mapping descriptive names to IP addresses was needed. To that end, `HOSTS.TXT` can be seen as the immediate forerunner to the Domain Name System we use today.

In 1983, Paul Mockapetris authored **RFC 882**, which describes how a system mapping memorable names to unmemorable IP addresses could work. A team of students at UC Berkeley created the first implementation of Mockapetris' ideas in 1984, naming their creation Berkeley Internet Name Domain (BIND) server. Today, BIND is still the most widely-used nameserver software on the Internet with over 70% of domains using it, according to the ISC and Don Moore's survey. Since then, a number of RFC documents have been published which have continued to improve how DNS works and runs.

Terminology

Domain name

A domain name is likely the way you interface with DNS most often when browsing the Internet. Examples are, quite literally, everywhere - a very limited example set includes `google.com` and `wikipedia.org`.

Top-Level Domain

A top-level domain is an important, but rather generic, part of a domain name. Examples include `com`, `net`, `gov` and `org` - they were originally defined in **RFC 920**. [ICANN](https://www.icann.org) controls the TLDs, and delegate responsibility for the registration and maintenance of specific domains to registrars.

Fully Qualified Domain Name (FQDN)

A fully-qualified domain name is equivalent to the absolute name. An absolute is one in which the full location of an item is specified. For instance, in HTML, we might use `<a href=http://www.google.com/>Google</a>` to link to Google. But we might use `<a href=page.html>Page</a>` to link to a specific page relatively. The difference here is that relative names are relative to the current location. Domain names can also be relative to one another and therefore have a tendency to become ambiguous at times. Specifying an FQDN relative to the root ensures that you have specified exactly which domain you are interested in. Examples of FQDNs include `www.google.com.` and `www.gov.uk.`.

IP address

An IP address is used to uniquely address a machine on a network in numerical (IPv4) or alphanumerical (IPv6) form. It is important that we understand the concept of "network" used here to be relative to what we are trying to achieve. For instance, trying to contact another computer inside your home or office network means that the IP address of the machine you are trying to reach must be unique within your home or office. In terms of websites and publicly-accessible information available via the Internet, the "network" is - in fact - the Internet.

There are two types of IP addresses: one is becoming increasingly popular as we get close to running out of available IPv4 addresses.

An IPv4 address referenced everything in four sets of three period-separated digits. For instance, `8.8.8.8` and `102.92.190.91` are examples of IPv4 addresses. As more devices and people across the world come online, the demand for IPv4 addresses hit a peak, and ICANN are now very close to running out of available addresses. This is where IPv6 comes in.

IPv6 follows similar principles to IPv4 - it allows for machines to be uniquely referenced on the network on which they reside, but the addressing syntax incorporates alphanumeric characters to increase the number of available addresses by a significant base. They are written as `2001:0db8:85a3:0042:1000:8a2e:0370:7334`, although short-hand notations do exist (for instance, `::1` to refer to the local machine at any time).

Zonefile

A zonefile is simply a text file made of a variety of different records for an individual domain. Each line of a zonefile contains the name of a particular domain, and then the value and type associated with it. For instance, in `google.com`'s zonefile, there may exist a line which denotes `www` translates, via an `A record`, to `173.194.34.68`.

Records

A DNS record is a single mapping between a domain and relevant data - for instance, an IP address in the case of an `A record`, or a mail server's domain name, in the case of an `MX record`. Many records make up a zonefile.

How DNS works

Root Servers

At the very top of the DNS tree are root servers. These are controlled by the Internet Corporation for Assigned Names and Numbers. As of writing, there are thirteen unique root servers - however, an interesting caveat applies in that each of these root servers is actually a pool of root servers, acting in a load-balanced fashion in order to deal with the huge number of requests they get from the billions of Internet users daily. Their purpose is to handle requests for information about Top-Level Domains, such as `.com` and `.net`, where lower level nameservers cannot handle the request sufficiently. The root servers don't hold any records of real use, insofar as they cannot respond with an answer to a query on their own. Instead, they respond to the request with details of which nameserver is best advised to proceed further.

For example, let's assume that a request for `www.google.com` came straight in to a root server. The root server would look at its records for `www.google.com`, but won't be able to find it. The best it will be able to produce is a partial match for `.com`. It sends this information back in a response to the original request.

TLD Servers

Once the request for `www.google.com` has been replied to, the requesting machine will instead ask the nameserver it received in reply to the original request to the root server where `www.google.com` is. At this stage, it knows that this server handles `.com`, so at least it is able to get some way further in mapping the address to an IP address. The TLD server will try to find `www.google.com` in its records, but it will only be able to reply with details about `google.com`.

Domain-level nameservers

By this stage, the original request for `www.google.com` has been responded to twice: once by the root server to tell it that it doesn't handle any records, but knows where `.com` is handled, and once by the TLD server which says that it handles `.com`, and knows where `google` is. We've still got one more stage to get to, though - that's the `www` stage. For this, the request is played against the server responsible for `google.com`, which duly looks up `www.google.com` in its records and responds with an IP address (or more, depending on the configuration).

We've finally got to the end of a full request! In reality, DNS queries take place in seconds, and there are measures in place which we'll come on to in these DNS chapters about how DNS can be made faster.

Resource types

Whilst at it's most basic, DNS is responsible for mapping easily-remembered domain names to IP addresses, it is also used as a form of key/value database for the Internet. DNS can hold details on which mail servers are responsible for a domain's mail and arbitrary human-readable text which is best placed in DNS for whatever reason.

The most common types you'll see are:

Record Type	Description
A	Responsible for mapping individual hosts to an IP address. For instance, `www` in the `google.com` syntax.
AAAA	The IPv6 equivalent of an `A` record (see above)
CNAME	Canonical name. Used to alias one record to another. For example, `foo.example.com` could be aliased to `bar. example.com`.
MX	Specifies mail servers responsible for handling mail for the domain. A priority is also assigned to denote an order of responsibility.
PTR	Resolves an IP address to an FQDN. In practice, this is the reverse of an A record.
SOA	Specifies authoritative details about a zonefile, including the zonemaster's email address, the serial number (for revision purposes) and primary nameserver.
SRV	A semi-generic record used to specify a location. Used by newer services instead of creating protocol-specific records such as *MX*.
TXT	Arbitrary human-readable information that needs to be stored in DNS. Examples include verification codes and SPF records.

There's a good in-depth list of every record type, the description of its use and the related RFC in which it is defined in this Wikipedia article.

An example zonefile

```
$TTL     86400;      // specified in seconds, but could be 24h or 1d
$ORIGIN  example.com

@ 1D IN SOA ns1.example.com. hostmaster.example.com. (
```

```
            123456 ; // serial
                  3H     ;   // refresh
            15     ; // retry
            1w     ; // example
            3h     ; // minimum
            )

      IN NS ns1.example.com
      IN NS ns2.example.com // Good practice to specify multiple nameservers for fault-tolerance
      IN NS ns1.foo.com     // Using external nameservers for fault-tolerance is even better
      IN NS ns1.bar.com     // And multiple external nameservers is better still!

      IN MX 10 mail.example.com // Here, 10 is the highest priority mail server, so is the first to b
      IN MX 20 mail.foo.com     // If the highest priority mail server is unavailable, fall back to t

ns1   IN A     1.2.3.4
ns1   IN AAAA  1234:5678:a1234::12 // A and AAAA records can co-exist happily. Useful for supporting
ns2   IN A     5.6.7.8
ns2   IN A     1234:5678:a1234::89
mail  IN A     1.3.5.7
www   IN A     2.4.6.8
sip   IN CNAME www.example.com.
ftp   IN CNAME www.example.com.
mail  IN TXT   "v=spf1 a -all"

_sip._tcp.example.com. IN SRV 0 5 5060 sip.example.com.
```

Host-specific DNS configuration

If you are administering systems, specifically Unix systems, you should be aware of two pieces of host-side configuration which allow your machines to interface with DNS:

- `/etc/hosts`
- `/etc/resolv.conf`

/etc/hosts

The `/etc/hosts` file has the purpose of acting as a local alternative to DNS. You might use this when you want to override the record in place in DNS on a particular machine only, without impacting that record and its use for others - therefore, DNS can be overridden using `/etc/hosts`. Alternatively, it can be used as a back-up to DNS: if you specify the hosts that are mission-critical in your infrastructure inside `/etc/hosts`, then they can still be addressed by name even if the nameserver(s) holding your zonefile are down.

However, `/etc/hosts` is not a replacement for DNS - in fact, it is far from it: DNS has a much richer set of records that it can hold, whereas `/etc/hosts` can only hold the equivalent of `A` records. An `/etc/hosts` file might, therefore, look like:

```
127.0.0.1         localhost
255.255.255.255   broadcasthost
::1               localhost
fe80::1%lo0       localhost

192.168.2.2       sql01
192.168.2.3       sql02
192.168.1.10      puppetmaster puppet pm01
```

The first four lines of `/etc/hosts` are created automatically on a Unix machine and are used at boot: they shouldn't be changed unless you really know what you're doing! In fact, the last two lines of this section are the IPv6 equivalents of the first line. After these first four lines, though, we can specify a name and map it an IP address. In the above example, we've mapped `sql01` to `192.168.2.2`, which means that on a host with the above `/etc/hosts` configuration, we could refer to `sql01` alone and get to the machine responding as `192.168.2.2`. You'll see a similar example for `sql02`, too. However, there is a slightly odd example for the box named `puppetmaster` in that multiple friendly names exist for the one box living at `10.0.0.2`. When referenced in this way - with multiple space-separated names against each IP address - the box at `10.0.0.2` can be reached at any of the specified names. In effect, `puppetmaster`, `puppet`, and `pm01` are all valid ways to address `10.0.0.2`.

/etc/resolv.conf

`/etc/resolv.conf` exists on Unix machines to allow system administrators to set the nameservers which the machine should use. A DNS domain can also be referenced in this file, too. An example `/etc/resolv.conf` might look like:

```
domain     opsschool
nameserver 192.168.1.1
nameserver 192.168.1.2
nameserver 192.168.1.3
```

In this example, we would be specifying that any of `192.168.1.1`, `192.168.1.2` and `192.168.1.3` can be used by the host with the above configuration to query DNS. We are actually telling the host that it is allowed to use any of the nameservers in this file when it resolves (i.e.: makes a request for an entry and waits for a response) a host in DNS.

Setting the `domain` directive - as in the above example, where we specified it as `opsschool` - allows users to specify hosts by address relative the domain. For instance, a user could reference `sql01`, and a query would be sent to nameservers specified asking for records for both `sql01` and `sql01.home`. In most cases, the responses should match - just be careful if they don't, as you'll end up with some very confused machines when DNS has split-brained like this!

Caching

By itself, DNS doesn't scale very well. Imagine having a machine that needed to make many millions of DNS queries per day in order to perform its function - it would need to perform well and be constantly available. In order to cut the cost of hardware somewhat, to reduce pressure on networks, and to speed up receiving responses to common queries, many client machines will cache DNS records. The SOA record at the start of each zonefile on the nameservers specifies an `expiry` value, which tells clients for how long they can keep the zonefile in its current state before they must re-request it. This rather crude but effective updating method works well in the case of DNS.

Generally speaking, caching of DNS records (at least on Unix-based machines) is managed by individual applications. In a Windows environment, however, it is more centralised. To that end, whilst you cannot easily view the cache as it exists on an individual machine all in one place in Unix, you can using Windows - the `ipconfig /displaydns` command will print the cache as it stands. In Windows, you'll be presented with the record name as a number - this is a binary representation of the record type itself. Conversion charts can be found online, for example at Wikipedia.

Caching links directly to a phenomenon called propagation. Propagation is the process by which records that have previously existed and have been updated begin to get updated in other machines' caches. If the SOA record for a zonefile tells hosts to check back with the DNS server every 24 hours, then it should take - at most - 24 hours for machines to update their caches with the new record.

TTLs

TTLs, or 'time to live' values, are a useful feature in DNS which allows you to force the expiry of individual records, thus bypassing the `expiry` time referenced in the SOA record on a per-record basis. For instance, let's say that `opsschool.org` has moved to a new web host but it needs to ensure that the service is available as much as possible. By reducing the TTL for the `www` and `*` records in the `opsschool.org` zonefile, the switch between previous and new vendor should be relatively pain-free. TTLs and caching (see above) work well together - with a suitably high TTL and suitable caching in place, the time for a request to be responded to and the time for updated records to exist on caches are both dramatically reduced.

Forward and reverse DNS

By this point, we've covered many of the basic concepts of DNS - we've looked at what exactly DNS is, how the DNS tree works (in the forms of nameserver hierarchies and record types), and we've looked at host-side configuration using `/etc/resolv.conf` and `/etc/hosts`. There is, however, one further concept we need to cover: forward and reverse DNS.

Forward DNS is, in essence, simply DNS as described above. When `ftp.example.com` is requested, the root nameserver will reply with details of the nameserver responsible for `.com`, which will reply with the address of the nameserer responsible for `example.com`, which will then look in the `example.com` zonefile for the `ftp` record and reply appropriately. In fact, the terms 'forward DNS' and 'DNS' are pretty interchangeable: when talking about DNS, if you don't otherwise specify, most ops engineers will assume you're talking about forward DNS as it's the most often used direction.

However, whilst forward DNS is the type you're likely to run in to most often, it's also very important to know how reverse DNS works. If forward DNS maps hostnames to IP addresses, then reverse DNS does exactly the opposite: it maps IP addresses to hostnames. To do this, the zonefile in question must have a PTR record set for the record you're interested in. Getting used to PTR records and reverse DNS can be tricky, so it might take a few attempts until it catches on.

Domain names follow a specific syntax - `foo.tld`, where `.tld` is set by ICANN and chosen by the registrant when they register their domain. For instance, people can choose to register `.aero`, `.com` and `.tv` domains wherever they live in the world, subject to a fee. With reverse DNS, a similar syntax exists. Let's assume that we want to know which hostname responds at `22.33.44.55`. We do this as follows:

1. Reverse the octets of the IP address - `22.33.44.55` becomes `55.44.33.22`, for instance
2. Add `in-addr.arpa` to the end of the reversed address - we now have `55.44.33.22.in-addr.arpa`
3. The root nameserver tells queries to find the `arpa` nameserver
4. The `arpa` nameserver directs the query to `in-addr.arpa`'s nameserver
5. The `in-addr.arpa` nameserver then responds with details of `22.in-addr.arpa`, and so on...
6. In the zonefile, the IP address matching the query is then found and the relevant hostname is returned

Useful DNS tools

There are a number of very useful tools for querying DNS. A list of the most common and some example commands can be found below - for further instructions, see each tool's man page (found, in Unix, by typing `man $toolname` at a prompt, and in Windows by appending `-h` to the command.

Windows

`ipconfig` is a useful tool for diagnosing and configuring TCP/IP networks. Among its many switches, it allows the use of `/displaydns` which will dump the output of the DNS cache to the console for you. You can then use the `/flushdns` entry to clear your DNS cache on a Windows machine.

`nslookup`, however, might be more useful in your day-to-day use of DNS. It allows you to look up an entry on any nameserver that you know the public IP address or hostname for. In many respects, therefore, it acts much like `dig` on Unix systems does.

Unix

`dig` can be used to query a nameserver to see what values it holds for a specific record. For instance, you could run `dig opsschool.org` will produce the entire query and entire response, which whilst useful, is often not the information you are looking for. Running the same command, but specifying the `+short` switch just provides you with relevant detail - in the case of looking up an IP address for a hostname by way of A record, then the output from `dig` will just be the relevant IP address. Dig can also be used to query external nameservers, such as `8.8.8.8`, to see what values they hold.

For instance, `dig` can be invoked as follows:

- `dig opsschool.org a` for a verbose output listing only the `A record` for `opsschool.org`
- `dig opsschool.org a +short` for a much shorter, more concise version of the last command
- `dig @8.8.8.8 opsschool.org a +short` to repeat the same command as above, but against Google's 8.8.8.8 nameserver

In place of `dig`, you may also see `host` used in its place. Essentially, both tools perform approximately the same action - given a DNS server (or not, as not doing queries the one you have specified in `/etc/resolv.conf`), `host` also allows you to query a record and its value.

For further details on the usage of each tool, have a look at the relevant manual pages - type `man dig` and `man host` to find the man pages on any Unix system. You might choose to stick with one tool, or get used to both.

DNS 201

Architecture/design choices

Master/slave

Shadow master/slave

Shadow master/multi=slave

Split horizon

DNSSEC

DHCP

DHCP, or, Dynamic Host Control Protocol is a standard that allows one central source, i.e. a server or router, to automatically assign requesting hosts an IP address.

The most common DHCP servers on Linux platforms are the ISC's (Internet System Consortium) `dhcpd`, and `dnsmasq`. On Windows platforms there are a variety of DHCP servers, though the Microsoft DHCP server is arguably the best in terms of support and integration.

Tools: ISC dhcpd

`dhcpd` is the most common DHCP server, and it offers excellent integration with BIND (DNS server).

Protocol

dhcp helper

DHCP helpers are sometimes referred to as DHCP relayers. The basic idea is that a relay agent will forward DHCP requests to the appropriate server. This is necessary because a host that comes online on a subnet with no DHCP server has no way of finding the correct server; it needs a DHCP assigned address to find a route to the correct DHCP server, but can't get there because it has no IP address! DHCP relaying solves this chicken and egg problem by acting as an intermediary. See this wikipedia [16] article for more details.

Defining classes, leases

Options

Default gateway

DNS server(s)

(Tie in previous chapters re: TFTP, PXE with related options?)

HTTP 101 (Core protocol)

HTTP is a response-request protocol in the application layer of the *OSI* model. An HTTP server listens for an incoming client request on a socket bound to an address and a port. The address can either be a specific address configured on the host, or a wildcard to accept connection on all IP addresses configured on the host. By default the port is 80, but this is not required.

Request

Once the TCP connection is made (*OSI* layer 4), the client initiates HTTP communication by sending an HTTP request on the established connection. The HTTP request includes a request line, which is composed of a method, URI and protocol version. A request method defines the action to be executed on the target resource, which on its turn is identified by the URI. The protocol field indicates the version of HTTP to be used.

```
GET /index.html HTTP/1.1
Host: example.com
```

The request line is followed by headers containing client information and other metadata. As per protocol specification, the client must include the `Host` header in the HTTP request. A single server may serve several (virtual) hosts, so the `Host` header signals the server to which of the possible hosts to route the request. Some implementations might

[16] DHCP relaying explained

interpret a request without `Host` header as invalid, and respond with the `400 Bad Request` message. An empty line indicates the end of the request line and header section.

The available methods are :

Method	Description
GET	Tranfers a current representation of the target resource.
HEAD	Same as GET, but only transfer the status line and header section.
POST	Perform resource-specific processing on the request payload.
PUT	Replace all current representations of the target resource with the requested payload.
DELETE	Remove all current representations of the target resource.
CONNECT	Establish a tunnel to the server identified by the target resource.
OPTIONS	Describe the communication options for the target resource.
TRACE	Perform a message loop-back test along the path to the target resource.

(source: **RFC 7231**)

After the empty line, an optional message body in the request contains the payload (data) associated with the request.

Response

The server parses the request, and, depending on the validity of the request, sends one or more responses. An HTTP response starts with the status line, which contains the protocol version, success or error code, and a reason message.

The status line is mostly followed by a header section containing server information and other metadata. An empty line indicates the end of the status line and header. The final part of the response, the message body, contains the payload sent back to the client.

```
HTTP/1.1 200 OK
Date: Wed, 20 May 2015 19:45:37 GMT
Server: Apache/2.2.22 (Debian)
Last-Modified: Tue, 19 May 2015 14:25:26 GMT
ETag: "9dac8-2d-5178e2f1c2e8b"
Accept-Ranges: bytes
Content-Length: 45
Vary: Accept-Encoding
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>
```

Common response codes

Code	Reason	Description
200	OK	The request has succeeded.
301	Moved Permenantly	The target has moved to a new location and future references should use a new location. The server should mention the new location in the `Location` header in the response.
302	Found	The target resources been temporarily moved. As the move is temporarily, future references should still use the same location.
400	Bad Request	The server cannot process the request as it is perceived as invalid.
401	Unauthorized	The request to the target resource is not allowed due to missing or incorrect authentication credentials.
403	Forbidden	The request to the target resource is not allowed for reasons unrelated to authentication.
404	Not Found	The target resource was not found.
500	Internal Server Error	The server encountered an unexpected error.
502	Bad Gateway	The server is acting as a gateway/proxy and received an invalid response from a server it contacted to fulfill the request.
503	Service Unavailable	The server is currently unable to fulfill the request.

Tools: Speaking http with telnet/netcat/curl

Telnet

The telnet utility is used for interactive communication to a host on a given port. Once the connection to the remote host is established, an HTTP request can be send to the host by typing it in the prompt:

```
$ telnet www.opsschool.org 80
GET /en/latest/http_101.html HTTP/1.1
Host: www.opsschool.org

Trying 162.209.114.75...
Connected to ops-school.readthedocs.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Tue, 23 Jun 2015 20:41:10 GMT
Content-Type: text/html
Content-Length: 140673
Last-Modified: Wed, 27 May 2015 12:16:25 GMT
Connection: keep-alive
Vary: Accept-Encoding
ETag: "5565b599-22581"
X-Served: Nginx
X-Subdomain-TryFiles: True
X-Deity: chimera-lts
Accept-Ranges: bytes
```

```
<!DOCTYPE html>
<!--[if IE 8]><html class="no-js lt-ie9" lang="en" > <![endif]-->
<!--[if gt IE 8]><!--> <html class="no-js" lang="en" > <!--<![endif]-->
<head>
  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>HTTP 101 (Core protocol) &mdash; Ops School Curriculum 0.1 documentation</title>


  (...)

  </body>
</html>
```

If the requested resource is found on the specified host, it is returned in the body of the response.

cURL

cURL is a tool and library for transferring data with URL syntax, capable of handling various protocols. In contrast to telnet, cURL is able to create the HTTP request (or other known protocols) based on the given input. It is a command line tool, and can be used in scripts to automate HTTP communication.

In its simplest form, cURL sends a GET request to the given target

```
$ curl http://www.opsschool.org/en/latest/http_101.html

<!DOCTYPE html>
<!--[if IE 8]><html class="no-js lt-ie9" lang="en" > <![endif]-->
<!--[if gt IE 8]><!--> <html class="no-js" lang="en" > <!--<![endif]-->
<head>
  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>HTTP 101 (Core protocol) &mdash; Ops School Curriculum 0.1 documentation</title>

  (...)

</body>
</html>
```

With the `--request` or `-X` parameter, the method can be specified. To include the headers in the output, the `-i` can be used, and to only output the headers, use the `-I` switch.

```
$ curl -I --request GET  http://www.opsschool.org/en/latest/http_101.html
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Tue, 23 Jun 2015 20:42:25 GMT
Content-Type: text/html
Content-Length: 140673
Last-Modified: Wed, 27 May 2015 12:16:25 GMT
Connection: keep-alive
Vary: Accept-Encoding
ETag: "5565b599-22581"
X-Served: Nginx
X-Subdomain-TryFiles: True
```

```
X-Deity: chimera-lts
Accept-Ranges: bytes
```

cURL outputs a progress meter to the terminal, showing statistics about the operation. However, once cURL is about to write data to the terminal, the display of the progress meter is disabled so it may seem as if it was never present. The progress meter can still be seen when the output is written `-o <file>` or piped `>` to a file.

In case the download takes some time (for example downloading a big file) the progress meter can still be seen in the terminal. The `-O` switch makes cURL write output to a local file named like the remote file.

```
$ curl -O http://localhost/bigfile
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  100M  100  100M    0     0   400M      0 --:--:-- --:--:-- --:--:--  414M
```

When starting to download a large file with `-C` option, the download can be resumed in case of an interrupt.

cURL can use a proxy to retrieve a resource when specified to do so using the `-x` or `--proxy` option with the proxy specified as `[protocol://][user:password@]proxyhost[:port]`.

By default cURL does not follow HTTP redirects, and instead a 3xx redirection message is given. When browsing `www.opsschool.org` with cURL, the 302 response code indicates that the requested page is temporarily residing on a different location. The `Location` header contains a reference to the new location:

```
$ curl -I www.opsschool.org
HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
X-Deity: chimera-lts
X-Served: Flask
Content-Type: text/html; charset=utf-8
Date: Tue, 23 Jun 2015 21:04:01 GMT
Location: http://www.opsschool.org/en/latest/
X-Redirct-From: Flask
Connection: keep-alive
Content-Length: 229
```

The `-L` switch makes cURL automatically follow redirects and issue another request to the given location till it finds the targeted resource.

```
$ curl -IL www.opsschool.org
HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
X-Deity: chimera-lts
X-Served: Flask
Content-Type: text/html; charset=utf-8
Date: Tue, 23 Jun 2015 21:06:22 GMT
Location: http://www.opsschool.org/en/latest/
X-Redirct-From: Flask
Connection: keep-alive
Content-Length: 229

HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Tue, 23 Jun 2015 21:06:22 GMT
Content-Type: text/html
Content-Length: 195536
Last-Modified: Wed, 27 May 2015 12:16:27 GMT
Connection: keep-alive
Vary: Accept-Encoding
ETag: "5565b59b-2fbd0"
```

```
X-Served: Nginx
X-Subdomain-TryFiles: True
X-Deity: chimera-lts
Accept-Ranges: bytes
```

netcat

Netcat is a networking tool capable of reading and writing data across a network connection. This makes it possible to use netcat both as a client and as server. On many systems, netcat is started with the `nc` command instead of the program's full name.

Basic client/server To make a basic client-server connection, netcat can be told to listen on a port using the `-l` option:

```
$ nc -l 192.168.0.1 1234 > output.log
```

In above example, IP address is specified, but this is not required. When started without IP specified, netcat will be listening on `0.0.0.0:1234`. Any output is written to file output.log.

Then in another terminal, pipe some data to netcat that is connected to the destination IP and port.

```
$ echo "I'm connected as client" | nc 192.168.0.1 1234
```

Then again in the first (server) terminal, the data is displayed:

```
$ cat output.log
I'm connected as client
```

Beyond sending simple pieces of text, netcat can be used to copy files and directory structures.

Serving HTTP Netcat can be instructed to execute a program and redirect file descriptors when a connection is established. If compiled with DGAPING_SECURITY_HOLE option, netcat has the `-e <program>` option to specify which program to 'bind' to a connection. Even when the `-e` option is not available, a basic HTTP server can be created by redirecting file descriptors to our program.

Consider the following Bash script which implements a simple HTTP server:

```
#!/usr/bin/env bash

base_uri='/tmp'

respond(){
        [ -d "${base_uri}/${uri}" ] && uri+=/index.html
        if [ -f "${base_uri}/${uri}" ]
                then
                        printf '%s\r\n'\
                                "HTTP/1.1 200 OK"\
                                "Date: $(date '+%a,%e %b %H:%M:%S GMT')"\
                                "Server: myserver"\
                                "Content-Length: $(stat -c'%s' ${base_uri}${uri})"\
                                "" | tee >(cat - >&2);
                        cat <"${base_uri}/${uri}" | tee >(cat - >&2);
                else
                        printf '%s\r\n'\
                                 "HTTP/1.1 404 Not Found"\
                                "Date: $(date '+%a,%e %b %H:%M:%S GMT')"\
```

```
                                "Server: myserver"\
                                "Content-Length: 30"\
                                ""\
                                "<html><b>Not Found</b></html>" | tee >(cat - >&2);
        fi
        unset request method uri version
        exit 0
}

## MAIN ##

read -r request
read -r method uri version <<<"$request"
[ -n "$method" ] && [ -n "$uri" ] && [ -n "$version" ] || echo "HTTP/1.1 400 Bad Request"
echo $request >&2
while read -r header
do
        header=${header%%$'\r'}
        echo $header >&2
        [ -z "$header" ] && { respond; break; }
done
```

Stdout output acts as the response to the client. Output to file descriptor 2 (`>&2`) is written to the server terminal. This only covers reprentation of a web page, or an error page in case of a missing file.

To launch this with netcat compiled with DGAPING_SECURITY_HOLE option:

```
$ while true; do netcat -lp 1234 -e ./httpd.sh; done
```

Or when `-e` is not available:

```
$ mkfifo /tmp/httpipe
$ while true; do cat /tmp/httpipe | ./httpd.sh | nc -l 192.168.0.1 1234 > /tmp/httpipe ; done
```

This creates a pipe that we use to redirect data to and from the httpd script. Now we can test it with either of the above shown tools:

```
$ curl 192.168.0.1:1234/
<html>Hello there world!</html>

$ nc 192.168.0.1 1234
GET /notexisting.html HTTP/1.1

HTTP/1.1 404 Not Found
Date: Tue, 9 Jun 18:39:59 GMT
Server: myserver
Content-Length: 30

<html><b>Not Found</b></html>

$ telnet 192.168.0.1 1234
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
GET /index.html HTTP/1.1

HTTP/1.1 200 OK
Date: Tue, 9 Jun 18:44:53 GMT
Server: myserver
```

```
Content-Length: 32

<html>Hello there world!</html>
Connection closed by foreign host.
```

Apache, nginx

Apache and Nginx are the two most used open-source webservers on the internet. Though they share several features, their architectures differ significantly.

Apache relies on a process-based model, either with a single or multiple threads per process that handle a connection, whereas Nginx operates event-driven with multiple single-threaded processes that all handle connections asynchronously. Nginx' event-driven approach requires less resources, making it faster, especially for static content. For dynamic content however, Nginx needs to proxy requests to an external processor for execution as it has no programming language support. Therefore Nginx is often used as a front-end/proxy server in combination with other application servers, possibly Apache.

Both web servers are extensible with modules (security, compression, proxy, ...). Where Apache is able to dynamically load and enable modules at runtime, Nginx can only load modules that have been precompiled into the executable.

HTML

Hyper**T**ext **M**arkup **L**anguage is the markup language used to create web pages. An HTML page is made up of a tree of elements and text. Elements are identified by tags that are enclosed by angle brackets. The root element is the `<html>` which on its turn contains the `<head>` element with information about the document, and `<body>` elements with the visible content of the page.

HTML pages can reference to other pages by using hyperlinks identified by `<a>` tags. The developers of HTML, W3C, have the full specification on their website: http://www.w3.org/html/.

Virtual hosting

Virtual hosting is a technique where mutiple websites are hosted on a single web server. The server's resources are shared amongst all configured hosts, but each host has its own IP address or domain name. There are two mechanisms for implementing virtual hosts:

IP-based

In IP-based virtual hosting each virtual host is configured with its own dedicated IP address. The web server is configured to serve multiple IP addresses, either on multiple physical network interfaces or on virtualized interfaces on a single physical interface. Based on the IP on which a client connects, the web servers determines which of the configured virtual hosts to serve to the client.

Name-based

In name-based virtual hosting a web server serves multiple host names on a single IP address. While a client is connecting to a webserver, the destination address is resolved to an IP address either by DNS or local lookup table. As several virtual hosts may be hosted on the same IP, the client needs to state which website it is requesting for by setting the `Host` header. The web server inspects the `Host` header field in the HTTP request (mandatory as of HTTP/1.1) and routes the request to the correct virtual host.

HTTP 201 (Application Servers & Frameworks)

Java

Tomcat

JBoss

Jetty

node.js / Server-side Javascript

Perl

PHP

Ruby

Rails

Sinatra

Python

Django

Flask

SMTP 101

Tools: Speaking smtp with telnet/netcat

Risks / Dangers / Open relays

Tools to test/confirm operation as an open relay

Protocol

MX routing

Envelope

Message Headers

Message Body

MTAs (forwarding, smart hosts, etc.)

Postfix

Sendmail

SMTP 201

Anti=Spam

Blacklisting

Whitelisting

Greylisting

SpamAssassin?

Reliable Mail Delivery

Basic Etiquette

Authentication (SPF, DKIM)

Bounce Management

Feedback Loops

Reputation Management

Advanced routing

Hosted services = why they are good/bad/ugly

Identity Management 101

LDAP

LDAP (Lightweight Directory Access Protocol) is an application protocol used for accessing and maintaining directory services. Directory Services uses a database-styled system to correlate information about objects in that directory with metadata about that object.

Directory services provide information about a unique ID, in much the same way that an ISBN number identifies a book in a library. That unique ID can be a user, a computer, or a group, or any number of other objects, depending on how the directory index (or schema, as it is referred to) has been specified.

The metadata for that object can include things such as a Display Name, Address, Date Created, Date Modified, Home Directory, etc... Since LDAP can be extended by modifying or adding to the schema, it is a very flexible format and serves as the base of most modern directory services.

Active Directory

Active Directory is Microsoft's implementation of LDAP, coupled with Kerberos encrypted authentication security. One benefit of Active Directory is Group Policy, which is a flexible and granular system for controlling a great many user and computer settings based on flexible criteria, such as Organizational Unit or Group membership.

Active Directory Domain Services Port Requirements can be found on Technet.

Active Directory works best with Windows Domain Controllers serving as both DHCP and DNS servers, in an AD-integrated DNS zone. While DNS or DHCP can be handled by alternative systems such as BIND, this is an advanced configuration, and should not be attempted in most scenarios.

The Active Directory schema has been modified with each release of Windows Server, and will often be modified when deploying core Microsoft server applications such as Exchange or Certificate Services. The overall state of the domain is referred to as the Domain Functional Level - this may require consideration when determining requirements of a project or feature implementation.

Active Directory is typically managed through a variety of tools, including:

GUI Tools

- Active Directory Users & Computers (to manage the users, computers, and groups in a domain)
- Active Directory Sites & Services (to manage the replication of AD to different sites/servers)
- adsiedit (a useful tool for viewing the attributes of objects within the domain)

Command Line Tools

- Powershell - through the Active Directory cmdlets
- ldp - an extremely low-level tool for interacting with LDAP directly, not recommended for most uses

OpenLDAP

Directory389

NIS

Active Directory 101

What is Active Directory?

Active Directory is a Directory Service created by Microsoft. It is included with most Windows Server operating systems.

Almost all Active Directory installations actually include several separate but related components; although the term "Active Directory" technically refers only to the directory service, in general use it refers to the entire constellation of parts.

What is Active Directory used for?

Active Directory is primarily used to store directory objects (like users and groups) and their attributes and relationships to one another. These objects are most commonly used to control access to various resources; for instance, an Active Directory might contain a group which grants its members permission to log into a certain server, or to print to a specific printer, or even to perform administrative tasks on the directory itself.

Active Directory also provides a useful configuration management service called *Group Policy*, which can be used to manage computers which connect to the domain in order to install packages, configure software, and much more.

You mention "separate components"; what is Active Directory composed of?

Most Active Directory installations have a few distinct parts which all work together:

- a *directory database* which stores the actual directory information
- a *Kerberos key distribution center* which helps manage user passwords and other security resources
- a *DNS server*, which maps IP addresses to hostnames and back
- a *DHCP server*, which grants dynamic IP addresses to hosts as they join the network
- one or more *Global Catalogs*, which cache parts of the directory database and help speed up some common queries

Also, Active Directory is designed in such a way that it can be run on multiple computers at the same time, which coordinate between themselves to ensure that their data is always consistent; this process is called "replication".

What specific services does Active Directory provide?

Group Policy

DNS

Kerberos Key Distribution Center

DHCP

Best Practices for managing an Active Directory installation

Cleaning up Unneeded Objects

Making Backups

Replication Health

Active Directory 201

Detailed Breakdown of Active Directory Components/Services

NTDS, the database

Kerberos KDC

DNS

DHCP

Global Catalog

The Global Catalog (GC) holds a subset of attributes for all objects in a multi-domain forest. It can be used to search all objects in the forest. For example, when searching for the a specific user, the GC can be searched to find users not only in the current domain but also in other domains in the forest. The attributes included in the GC's subset of object attributes are those required by the schema and those most commonly used in searches. Attributes can be added and removed from the Global Catalog using the Active Directory Schema snap-in. Alternatively, they can be added with PowerShell using the following command:

```
Set-ADObject "yourAttribute" -Replace @{"isMemberOfPartialAttributeSet"=$true}
```

When Active Directory Domain Services is installed, the GC is added to the first domain controller in the forest. In order to avoid traversing a WAN to gather GC information, it is recommended that each Active Directory site have at least one Global Catalog server. It can be added to additional domain controllers with Active Directory Sites and Services by following the steps in this link: https://technet.microsoft.com/en-us/library/cc755257.aspx

It can also be done with a PowerShell command:

```
Set-ADObject "CN=NTDS Settings,CN=yourDC,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configurat
```

Universal groups can contain objects from any domain in the forest. The GC holds the membership for all universal groups. As part of the authorization process, the authenticating DC retrieves the security identifier for all the member's security groups, which includes universal groups. Therefore, if the environment has more than one domain, the global catalog server must be queried during authentication in order to find objects from the universal groups in other domains.

FSMO Roles

FSMO stands for Flexible Single Master Operation. FSMO roles are also referred to as Operations Master roles. They are various functions performed by Domain Controllers (DCs) in Active Directory.

There are 5 FSMO roles. Three of the roles exist for every domain in the forest and two apply to the entire forest. While it is possible to have 5 domain controllers each with a separate FSMO role, it is common to have multiple roles on one server. The forest-wide roles are Schema Master and Domain Naming Master. The domain-wide roles are PDC Emulator, RID Master, and Infrastructure Master.

Schema Master

This role is forest-wide. This domain controller can update the schema. Given that the schema is not frequently updated, availability of this DC is less important than it is for other roles. By default, the role is held on the first server promoted to a DC in the forest.

Domain Naming Master

This role is also forest-wide. It controls changes to the forest namespace. It can add, remove, rename, and move domains. Since domain name changes are infrequent, this role is rarely used. By default, the role is held on the first server promoted to a DC in the forest.

PDC Emulator

This role is domain-wide. PDC stands for primary domain controller. It holds the latest password hashes and lockout information for all accounts. Password changes are immediately forwarded to the PDC from other DCs. Also, it is the authoritative time source for the domain. This role most directly affects users. Thus, the availability of the DC with the role is critically important.

The name emulator came from its ability to emulate the functionality of a Windows NT 4.0 PDC, which was useful in a mixed environment with Windows NT 4.0 BDCs. Extended support for Windows NT 4.0 server ended in 2004 and as a result, it is unlikely to be seen in modern environments.

RID Master

This role is domain-wide. A RID is a relative identifier used as part of the SID. The SID is used to uniquely identify objects. It is similar in function to the GUID but used for security. As SIDs must be unique in the domain, the RID master maintains the global pool of unique RID values for the domain, a subset of which are given out to other DCs. In past versions of Active Directory, rolling back a DC on a virtual machine could cause RID reuse. However, newer versions of Active Directory are able to detect and prevent this event when a virtual machine is restored.

Infrastructure Master

This role is domain-wide. It is used to help manage cross-domain references. Thus, if the forest only holds one domain, the role is not relevant as there are no cross-domain references. In order to update the cross-domain references, it compares the data to the Global Catalog (GC). If all DCs have the GC, then the placement of this role is irrelevant as it will already have the updated data.

Advanced Active Directory Maintenance

Remote Filesystems 101

NFSv3

iSCSI

SAMBA/CIFS

Remote Filesystems 201

GlusterFS

NFSv4

Netatalk / AFP

S3

Programming 101

Shell scripting basics

Specifying the interpreter

Shell scripts will typically start with `#!/bin/sh` or `#!/bin/bash`. This line, affectionally known by various names including hashbang and shebang, is treated as a directive to run the rest of the script with the given interpreter.

This directive, combined with setting the execute bit via `chmod` tells the system that the file is meant to be executed, rather than simply a file containing text.

The same line may be used instead to specify another shell on the system, such as `ksh`, `tcsh`, `zsh`, or another interpreter entirely, such as Perl, Python, or Ruby.

Portability Considerations

If you only target Linux systems, you may never run into much in the way of shell portability concerns, and you can usually specify the interpreter to run your script as `#!/bin/bash` safely. However, many systems other than Linux (and some limited Linux environments such as recovery modes and embedded systems) will only have the standard Bourne shell installed as `/bin/sh`, and if bash is installed at all, it will be in another location.

Bash is backwards-compatible with scripts written for the Bourne shell, but adds new extensions, which will not work in the original `/bin/sh` that's found on BSD, Solaris, and other Unix systems. Anything that needs bash should call it specifically, anything which does not should simply refer to `#!/bin/sh`

Caution must be used in writing scripts to specify the correct interpreter. `#!/bin/sh` is virtually guaranteed to be a *POSIX*-compatible shell - that is, a shell which is compatible with the original Bourne shell - but there is no guarantee that this shell will bash. The term "bashism" commonly refers to features of bash not contained in the *POSIX* `/bin/sh` specifications. The bash manual contains a list of these functions.

Aside from the `/bin/bash` vs `/bin/sh` considerations, it's not guaranteed that bash will be located at `/bin/bash` - it may commonly be found at `/usr/bin/bash`, `/usr/local/bin/bash`, and many other locations on systems other than Linux - since you won't know beforehand where it's located, a common trick is to specify the interpreter as `#!/usr/bin/env bash`, using the `env` utility as a way to search the path for bash. This technique is also highly recommended for calling other shells and interpreters, as this allows them to be found anywhere in the path, rather than at some fixed location which may change between systems.

A less common portability consideration, but once still worth mentioning, is that of limited environments, such as systems on which `/usr` has not yet been mounted, or recovery environments. If a script must run under such conditions, it is best to aim for strict *POSIX* compatibility, and specify your interpreter as `#!/bin/sh`, so as not to rely on utilities which may be on unmounted filesystems.

Variables

There are two kinds of variables you can use in your scripts: user-defined and built-in. Quite obviously, user-defined variables are created by the user (you) and built-in variables are automatically set by the shell you use.

- User-defined variables

To create a variable, all you need to do is assign it a value.

```
$ ops_var=1
```

NOTE: When using a BASH shell or writing a BASH script, make sure that you don't leave spaces in the expression; doing so will lead to an error. Always make sure you don't have spaces in variable assignment statements in BASH.

To echo out the value of this variable, use `$ops_var` like so:

```
$ echo $ops_var
1
```

The same variable can be assigned different values in the same script. Creating a script called `ops_script.sh`

```
#!/usr/bin/env bash

# First an integer value
```

```
ops_var=1
echo $ops_var

# Then a string
ops_var="Hello"
echo $ops_var
```

When this script is executed:

```
$ ./ops_script.sh
1
Hello
```

NOTE: From this point on, assume that the same `ops_script.sh` is going to be used. I also won't be typing `#!/usr/bin/env bash` every time, but know that it's present at the top of the script.

Variables can be used in other strings by calling them with curly braces `{ }` around the variable name.

```
ops_var="Yoda"
echo "${ops_var}, my name is"

$ ./ops_script.sh
Yoda, my name is
```

You can use variables to store user input and use it later on in the script. For example, if you want to ask the user for their name and echo it back to the screen:

```
print "Hello, what is your name?: "
read name

echo "Hello, ${name}"
```

I'm going to supply the name "Yoda" in this case.

```
$ ./ops_script.sh
Hello, what is your name?: Yoda
Hello, Yoda
```

By default, all variable values are treated as strings, unless the operation they are used for explicitly uses them as another data type. Assume you have two variables that you assign integer values to, and you want to add them together.

```
first_var=1
second_var=2
result=${first_var}+${second_var}

echo ${result}
```

You would expect the output of this to be 3, but this is not the case with bash.

```
$ ./ops_script.sh
1+2
```

What happened here was that both values were treated as string and the expression `${first_var}+${second_var}` got evaluated as the string "1+2". To actually add these two numbers, the operation needed is a little different.

```
first_var=1
second_var=2
result=$(( ${first_var} + ${second_var}))
```

```
echo ${result}
```

Here, the `$(( ))` tells bash that anything that goes inside these double parentheses needs to be evaluated as an arithmetic operation.

```
$ ./ops_script.sh
3
```

- Built-in variables

The second kind of variables that you can use in your scripts are the ones that the shell automatically populates for you when it is started up. These variables store information specific to your shell instance, so if you and a co-worker are both logged on to the same server, both of you might not see the same values for all built-in variables.

Here are some examples of built-in variables:

```
# Home directory of the logged-on user.
$HOME

# Name of the computer you are currently logged on to
$HOSTNAME

# Present working directory
$PWD
```

The above variables contain some information relative to the shell itself. But there are other built-in variables you can use within your script which store information like exit statuses of scripts/commands and number of arguments passed to a script.

```
# Exit status of the previously executed script/command
$?

# Number of arguments passed to the script
$#

# Value of the 1st argument passed to the script
${1}
```

These variables let your script take in parameters which can be then used throughout the script.

For example, I will write a script that prints the number of parameters received, and use the first one in a string

```
# Print the number of arguments passed
echo "Number of arguments: $#"

# Use the first argument in a string
echo "First argument passed was: ${1}"
```

I'll run this script a couple of times with different arguments to show how this works

```
$ ./ops_script.sh hello world
Number of arguments: 2
First argument passed was: hello

$ ./ops_script.sh car truck bike scooter
Number of arguments: 4
First argument passed was: car
```

Control Statements

tests / conditionals loops

functions

arrays

style

Redirection

I/O

Pipes

stderr vs. stdout

/dev/null and /dev/zero

Regular Expressions

Sed & awk

GIGO

Validating input

Validating output

Trapping & handling exceptions with grace

Programming 201

Common elements in scripting, and what they do

Syntax

Variables

Common data structures

Rosetta stone? One man's hash is another's associative array is another man's dict(ionary)?

Functions

Objects

C (A very basic overview)

The main loop

Libraries & Headers

#include

The Compiler

The Linker

Make

Lab: Open a file, write to it, close it, stat it & print the file info, unlink it. Handle errors.

Ruby

Ruby is a very user-friendly, flexible language and fun to use. To quote from the Ruby website, Ruby is described as:

> A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

The creator of Ruby, Yukihiro "Matz" Matsumoto, took various parts of his favourite languages (Perl, Smalltalk, Ada, Lisp and Eiffel) to create Ruby.

Reading and writing Ruby code is amazingly easy and fun. Once you learn the basics, it is amazing how much can be achieved in so little and concise code. A very simple example would be how iterations or loops are done in Ruby:

```
for(int i = 0; i < 3; ++i) {
    std::cout << "Hello"
}
```

You will see this for Ruby:

```
> (1..3).each { puts "Hello" }
Hello
Hello
Hello
=> 1..3

> (1..3).each do
  puts "Hello again"
> end
Hello again
Hello again
Hello again
=> 1..3
```

Ruby is a very good tool to write scripts.

Although this will be not covered here in detail, a very important thing to keep in mind is that in Ruby, **everything is an object**. This means that you can treat everything i.e. numbers, strings, classes, objects themselves, etc. as objects. Even the simplest of Ruby code will use this principle:

```
> 3.times { puts "hello" }
hello
hello
hello
=> 3
```

```
> "michael".capitalize
=> Michael
```

Strictly speaking, there will be cases where the above statement is not true in Ruby. For example, in Ruby, functions are not first class objects. In some languages like Javascript and Python, functions are first class objects. In these languages, a function can be treated like an object, i.e. they have attributes, they can be referenced and passed as parameters, etc.

Running Ruby Code

Ruby scripts are usually text files with `.rb` extension. You can run your Ruby scripts as follows:

```
$ ruby script.rb
```

You can run ad-hoc Ruby code in an interactive session called the Interactive Ruby or `irb` in short.

```
$ irb
1.9.3-p448 :001>
```

All Ruby examples in this topic will start with >, short for 1.9.3-p448 :XXX>. It means that it is running inside an irb session. *1.9.3-p448* is the Ruby version the author was running while writing this topic. The XXX are line numbers.

Syntax

- Conditionals
- Symbols
- Blocks

Variables

Common data structures

- Arrays

Arrays in Ruby are ordered collections of heterogenous items. Items can be added, inserted, removed from an array. Arrays are indexed starting from 0.

```
> empty_ary = []
=> []
> str_ary = ["Pune", "Mumbai", "Delhi"]
=> ["Pune", "Mumbai", "Delhi"]
> num_ary = [1, 2, 3.14, 10]
=> [1, 2, 3.14, 10]
> mix_ary = ["this array has", 3, "items"]
=> ["this array has", 3, "items"]
> arr_in_ary = [1, 2, [3, 4], 5]
=> [1, 2, [3, 4], 5]
> str_ary.each { |city| puts city }
Pune
Mumbai
Delhi
=> ["Pune", "Mumbai", "Delhi"]
> num_ary[0]
=> 1
```

```
> num_ary[2]
=> 3.14
```

Notice how arrays are heterogenous, i.e. array elements can be of different types. And an array can have array as its element.

Array objects are instances of Array class. So all instance methods are accessible to array objects. Discussing every method is beyond the scope of this topic but here are a few examples:

```
num_ary = [1, 2, 3.14, 10]
> num_ary.first
=> 1
> num_ary.last
=> 10
> num_ary.length
=> 4
> num_ary.empty?
=> false
> empty_ary.empty?
=> true
```

It is highly recommended that one reads the Ruby Array API documentation.

- Hashes

Hashes in Ruby are ordered collection of unique keys and their values. A hash key can be of any object type. Values can be referenced by their keys.

```
> empty_hash = {}
=> {}
> device_hash = { samsung: "Galaxy S", apple: "iPhone"}
=> {:samsung=>"Galaxy S", :apple=>"iPhone"}
> device_hash[:samsung]
=> "Galaxy S"
> country_hash = { "America" => "Washington DC", "India" => "New Delhi", "Germany" => "Berlin" }
=> {"America"=>"Washington DC", "India"=>"New Delhi", "Germany"=>"Berlin"}
```

Hash objects are instances of Hash class. So all instance methods are accessible to hash objects. Discussing every method is beyond the scope of this topic but here are a few examples:

```
> country_hash["America"]
=> "Washington"
> country_hash["Sweden"] = "Stockholm"
=> "Stockholm"
> country_hash
=> {"America"=>"Washington DC", "India"=>"New Delhi", "Germany"=>"Berlin", "Sweden"=>"Stockholm"}
> country_hash.values
=> ["Washington DC", "New Delhi", "Berlin", "Stockholm"]
> country_hash.length
=> 4
> empty_hash.empty?
=> true
```

It is highly recommended that one reads the Ruby Hash API documentation.

Functions

Functions are used in Ruby to perform a specific task. In Ruby parlance, functions are generally termed as methods. Ideally, a single method should do a single task and no more. In Ruby, methods accept parameters and return a value.

A methods is enclosed inside `def` and the `end` keywords. Parentheses is optional in Ruby for passing parameters. The last line inside a Ruby method is returned by the method. Using `return` keyword is optional.

```
> def print_hello
    puts "hello"
  end
=> nil
> def sum(a, b)
    a + b
  end
=> nil
> def sum2 a, b
    return a + b
  end
=> nil
> print_hello
=> hello
> sum(2, 3)
=> 4
> sum 4, 6
=> 10
```

Objects and Classes

As mentioned above, in Ruby, **everything is an object**. Ruby also has a class called `Object`. It is the default root of all Ruby objects.

Ruby objects can have attributes and methods. An instance of Object class (and in general, to create an instance of any class) can be created as follows:

```
> obj = Object.new
=> #<Object:0x007fcba39874b8>
```

In Ruby, you can create your custom classes. These can used along with the classes that come with Ruby and its standard library.

Classes can have methods. Classes also have a special method called `initialize`. When a new object is created in Ruby using `new` method, an uninitialized object is first created and then `initialize` is called. Any parameters passed to `new` is passed to `initialize`.

An instance variable in Ruby is prepended by `@` symbol.

```
> class Student
    def initialize(name, age)
      @name = name
      @age  = age
    end

    def details
      puts @name
      puts @age
    end
  end
=> nil
> s1 = Student.new('Cathy', 20)
=> #<Student:0x007fcba39b78c0 @name="Cathy", @age=20>
> s1.details
Cathy
```

```
20
=> nil
```

Rubygems

Todo

Explain more about what rubygems are as well as http://rubygems.org

Databases

Python

Python is one of the most versatile languages you're going to use in your career. You will soon see that for almost everything you want to do, Python either has a something in its standard library or an amazing third-party module that you can import in seconds. But since this is a guide for operations engineers, I'll focus the discussion more towards Python's scripting capabilities.

NOTE: Before I start, I want to point out a series of documents called Python Enhancement Proposals, PEP for short. Like their title suggests, these are potential enhancements to the Python language that have been proposed by members of the community. There's a lot of them, and you don't have to go over every single one, but you can find some very useful tips and best-practices there.

Syntax

- Indentation

If you've ever written or read any code in C, C++, Java or C#, you're used to seeing curly braces ({}) pretty much everywhere. These compiled languages use curly braces to denote the start and end of functions, loops and conditional statements. Python, on the other hand, uses indentation to achieve the same goal. What this means is that where you see this in C++:

```
if (3>2) {
    // Do something
}
```

You will see this for Python:

```
if (3>2):
    # Do something
```

As you can see, Python didn't need curly braces to signify the start or end of the if conditional; a simple indent does the job. Now when it comes to indentation, PEP8 says that you should use 4 spaces to indent your code. Keep in mind that this specifically means spaces and not tabs. Fortunately for you, most text editors today can automatically convert tabs to spaces so you don't have to hit four spaces every time you want to indent a line. However, if you are dealing with some legacy code that uses 8 space tabs, feel free to continue doing so.

Indentation is by far the most important part of Python's syntax you should keep track of. If there's two lines in your code where one uses 4 spaces and another uses one 4-space tab, Python's going to give you errors when you try to run your script. Be consistent with your indentation.

- Conditionals

Conditionals refer to `if, else` statements where you're checking if some condition is met and then taking action based on whether it is or not. Python supports conditionals just like any other language, with the only exception being indentation as explained above. A complete conditional block would look like this:

```
# Check if the variable 'num' is greater than or less than 5
if (num > 5):
    print "Greater"
else:
  print "Less"
```

You can even have 'else if' conditions, which in Python are used as `elif`

```
# Check if the variable 'num' is 2 or 5
if (num == 2):
    print "Number is 2"
elif (num == 5):
    print "Number is 5"
else:
    print "Number is neither 2 nor 5"
```

- Boolean Operations

Python can perform all of the standard boolean operations:`and`, `or` and `not`. The operations can be used as statements of their own:

```
>>> (3 > 2) and (3 < 4)
True
>>> (2 > 3) or (3 > 4)
False
>>> not (2 > 3)
True
```

and even in conditionals:

```
if not ((2 < 3) or (3 > 4)):
    print "Neither statement is true"
```

Variables

Variables in Python work just like in any other language. They can be assigned values like this:

```
times = 4
name = "John"
```

They can be used in almost any statement.

```
>>> print times
4
>>> times + times
8
```

You might have noticed that the variable didn't have to be created with a specific type before being assigned a value. Python allows you to assign any value to a variable and will automatically infer the type based on the value it is assigned. This means that the value assigned to a variable can be replaced with another value of a completely different type without any issues.

```
>>> times = 4
>>> print times
4
```

```
>>> times = "Me"
>>> print times
'Me'
```

However, if you try to perform an operation with two variables that have values of conflicting types, the interpreter will throw an error. Take this example where I will try to add a number and a string.

```
>>> times = 4
>>> name = "John"
>>> times + name
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

As you can see here, the interpreter threw a TypeError when we tried to add an integer and a string. But there is a way around this; Python lets you type cast variables so their values can be treated as a different type. So in the same example, I can either try to treat the variable `times` as a string, or the variable `name` as an integer.

```
>>> str(times) + name
'4John'
>>> times + int(name)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'John'
```

Here you can see that when we cast `times` as a string and added it to name, Python concatenated the two strings and gave you the result. But trying to cast `name` as an integer threw a ValueError because 'John' doesn't have a valid base 10 representation. Remember, almost any type can be represented as a string, but not every string has a valid representation in another type.

Common data structures

Out of the box, Python implements a few major data structures.

- Lists

Lists in Python are the equivalent of arrays in other languages you may be familiar with. They are mutable collections of data that you can append to, remove from and whose elements you can iterate over. Here's some common operations you can perform with lists:

```
>>> to_print = [1, 4]
>>> to_print.append('Hello')
>>> to_print.append('Hey')
>>> to_print
[1, 4, 'Hello', 'Hey']
>>> for i in to_print:
...     print i
...
1
4
Hello
Hey
>>> to_print[1]
4
>>> to_print[-1]
'Hey'
>>> to_print[-2:]
['Hello', 'Hey']
```

```
>>> to_print.remove(4)
>>> to_print
[1, 'Hello', 'Hey']
```

Just like arrays in other languages, Python's lists are zero-indexed and also support negative indexing. You can use the `:` to get a range of items from the list. When I ran `to_print[-2:]`, Python returned all items from the second last element to the end.

You may have also noticed that I had both numbers and strings in the list. Python doesn't care about what kind of elements you throw onto a list. You can even store lists in lists, effectively making a 2-dimensional matrix since each element of the initial list will be another list.

- Dictionary

Dictionaries are a key-value store which Python implements by default. Unlike lists, dictionaries can have non-integer keys. Items of a list can only be referenced by their index in the list, whereas in dictionaries you can define your own keys which will then serve as the reference for the value you assign to it.

```
>>> fruit_colours = {}
>>> fruit_colours['mango'] = 'Yellow'
>>> fruit_colours['orange'] = 'Orange'
>>> fruit_colours
{'orange': 'Orange', 'mango': 'Yellow'}
>>> fruit_colours['apple'] = ['Red', 'Green']
{'orange': 'Orange', 'mango': 'Yellow', 'apple': ['Red', 'Green']}
>>> fruit_colours['mango']
'Yellow'
>>> for i in fruit_colours:
...     print i
...
orange
mango
apple
>>> for i in fruit_colours:
...     print fruit_colours[i]
...
Orange
Yellow
['Red', 'Green']
```

You should be able to see now that dictionaries can take on custom keys. In this example, my keys were names of fruits, and the value for each key was the colour of that particular fruit. Dictionaries also don't care about what type your keys or values are, or whether the type of a key matches the type of its value. This lets us store lists as values, as you saw with the colour of apples, which could be red and green.

An interesting property about dictionaries that you might have noticed, is that iterating through the dictionary returned only the keys in the dictionary. To see each value, you need to print the corresponding value for the key by calling `fruit_colours[i]` inside the for loop where `i` takes on the value of a key in the dictionary.

Python implements a lot more data structures like tuples, sets and dequeues. Check out the Python docs for more information these: http://docs.python.org/2/tutorial/datastructures.html

Functions

Functions in Python work exactly like they do in other languages. Each function takes input arguments and returns a value. The only difference is syntax, you define functions with the keyword `def`, and don't use curly braces like in Java, C, C++ and C#. Instead, function blocks are separated using indentation.

```
>>> def square(x):
...     result = x*x
...     return result
...
>>> square(3)
9
```

You can even call functions within other functions

```
>>> def greet(name):
...     greeting = "Hello "+name+"!"
...     return greeting
...
>>> def new_user(first_name):
...     user = first_name
...     print "New User: "+user
...     print greet(user)
...
>>> new_user('Jack')
New User: Jack
Hello Jack!
```

Objects

Version Control

Git

SVN

CVS

API design fundamentals

RESTful APIs

JSON / XML and other data serialization

Authentication / Authorization / Encryption and other security after-thoughts.

:) https://github.com/ziliko/code-guidelines/blob/master/Design%20an%20hypermedia(REST)%20api.md

Continuous Integration

Hardware 101

Hardware Types

Rackmount

Rackmount refers to servers mounted in 19-inch relay racks. The server's height is measured in U or rack units (1.75 inches) and servers typically range in height from 1U to 4U(though they can be larger). The rack they are installed

into is typically 42U high, but you rarely see installations with more that 40 servers in a rack. Servers are secured to the rack via specialty rails.

Rackmount servers are the most common physical hardware installation in a datacenter. Individual rackmount systems can usually meet most density requirements, but in large installations physical cable management can become an issue. In addition systems management can be a problem in large installations. Rackmount servers usually are less expensive than other solutions, and modern rackmount servers typically include some type of remote-management.

Blades

A blade is a server on a removable card which connects to a chassis for power, network, video, and other services. The blade chassis is typically a rackmount unit, but sizes of this are usually much larger, using 8U or more depending on the number of blades the chassis supports.

Blades were originally hoped to solve server density issues, and some products have the ability to house four servers in only 2U of space. In recent years blades have been used to solve physical cable management issues in complex environments, since there are very few cables required to connect the systems to other pieces of infrastructure.

SANs

A Storage Area Network or SAN is a network that provides access to centralized block storage. Common choices for the network fabric are Fiber Channel and Ethernet. Solutions built around Ethernet usually rely on an additional protocols to implement access, and may require special configurations for high throughput (thing like large frames). The most common Ethernet based approach uses iSCSI. These networks are typically separate and dedicated to providing access to storage.

SAN Array

A SAN Array is a dedicated array of disks that has been designed to connect to a SAN. SAN Arrays will often support Ethernet or Fiber Channel, and some vendors will support both network types within the same unit.

Most SAN Arrays have two main components, a head unit and an array. The head unit controls a collection of arrays, and the SAN configuration. Most SANs come with two head units for redundancy. The Array unit holds some number of disks, and typically connects directly to the head units. The head unit is usually connected to the network fabric.

SANs installations vary from vendor to vendor, but they generally can be installed in standard 19-inch relay racks, or come in a custom cabinet. Power requirements also vary from vendor to vendor, but it its not atypical that they require dedicated power circuit due to the large number of spinning disks.

NAS

A Network-Attached Storage device or NAS is a specialty server designed to provide storage to other servers. Unlike SANs, a NASs will typically implement an standard network protocol like SMB or NFS to expose the storage to the client. The client will present these to its users as standard network share.

NASs and SANs are almost indistinguishable from each other physically. They will typically have head units and array units, and there will typically be two head units for redundancy. They will typically install into a 19-inch rack, or come with a custom cabinet. Lastly, they will typically need a dedicated circuit due to the large number of spinning disks.

Basic server architecture

CPU

RAM

Motherboard

Bus

Disk controllers

Network devices

BIOS/UEFI

PCI-E Cards

Disk management

Arrays

RAID/ZFS

Logical Volume Management

Performance/Redundancy

RAID/ZFS

Power

Electrical overview

110v/220v, amperage, 3 phase power

UPS

common plug types

common circuit types

Troubleshooting

Remote management (IPMI, iLO, DRAC, etc)

Common MCE log errors

System burn-in

Datacenters 101

Power budgets

A/B power and cascading outages

Cooling budgets

You will be judged by the tidiness of your rack

Machine and cable labeling

Traditional naming conventions

Datacenters 201

Networking many racks

Power

N+1, N+2 power

Fused vs usable

calculations

Cooling

N+1, N+2

The cooling exhausts for redundant or load-sharing aircos should be located in enough distance from each other that they do not influence each others effectiveness.

Cooling efficiency

Surrounding temperature and load both influence how good an airco can work. For example it will work best if it's not fully loaded (has to move as many units of thermal heat as it's specified for) and if the outside temp is somewhere below 30Deg Celsius. (Hopefully someone knows a smart formula for that) Their effectiveness drops if they're located too close to places where hot air is not really moving, i.e. ground level or non-raised on a rooftop, or on a roof section that is surrounded by higher parts of the building. Spraying water with a sprinkler can help.

Cooling failures

Very important is to gather information on the duration of a cooling failure so far. The surrounding datacenter structures delay a failure's effect by absorbing heat. That means two things: 1. At some point the walls have gained a temperature close to the ambient. From that moment, they will not absorb more heat, resulting in a massive increase of the rate at

which the room is heating up 2. A sliding delay that means the room will take somewhat as long as it took to heat up to cool down again, even if you already shut down everything that consumes power.

Generally you need to assess a few things: - is the temperature currently still a tolerable operating temperature - how quickly is it rising - how soon will the fault be fixed and when will a tolerable temperature be reached - how long is the expected time of overheating

- what damage to expect from quick heat-up and cool-down (i.e. tape backups in progress at the time of failure will be probably lost because of the stretching effects on the media. Disks can fail by the dozen at the same time)
- what are the absolutely critical systems running, and how much power do they use
- how many not absolutely critical systems are running, and how much power will it save to turn them off.
- How fast can you shut them down? (i.e. on Unix, that could mean init 2, wait, flip the switch)
- can you shut down enough to keep the most critical systems running?
- can you also shut down enough to massively flatten the temperature curve?

Generally, the goal is to flatten the effects so that the critical infrastructure gets away with a very and constant slow hike up and down at less than 2-3 deg / hr. This is also to be respected when the cooling is fixed again, it should run at emergency power only till the pivot is taking off the curve and the the power needs to be constantly decreased so the ambient takes a slow fall instead of going from 55C to 21C in 30 minutes. (Dangers: Humidity, microscopic tears on PCB)

Physical security and common security standards compliance requirements

Suggested practices

Access control

Security should require an ID from anyone coming onsite, and not stop at having you fill a form where you just write anything you want.

Techs should need to bring a ticket ID, and this ticket ID plus the tech's name should be announced by the vendor to security, plus(!) a callback to a defined number known to security. So, a double handshake with a token. This is really important to avoid incidents like suddenly missing two servers because "there were some techs that replaced it"

Most critical DCs will have security accompanying you to the server. Some will keep the sec person around while the tech is working. The really smart ones train their sec staff so that they will know *which* server the tech needs to go to and *which* disk is to be replaced as per the original ticket. (If you think the security people can't handle that, then just ask yourself who's to blame since it does work for other DCs)

Datacenters 301

Power

PUE (Power Usage Effectiveness)

3 phase power

Note you can't use a wire clamp to measure a 3-phase power cable, you need to measure each phase by itself.

DC power

48V (telco land, NEBS standard) and other versions that are being tried, (12, 24V). Grounding requirements, power bars. Devices often have different plugs. Most of the time still using A+B circuits Reduced conversions == Reduced losses. Amplified by most larger servers (i.e. HP C7000, SuperDome) using 48V internally. Even more interesting with large UPS banks since those also use 48V.

Increasing cooling efficiency

CFM

Hot aisle / cold aisle design fundamentals

Hot aisle / cold aisle containment

Chimneys

Design Options

Raised floor (18in & 36in)

Virtualization 101

Intro to virtualization technologies

What problems is virtualization good at solving, what isn't it good at.

Hardware assisted vs Paravirtualization

Hypervisors - overview, some comparison

Container-based virtualization (Jails, Containers/Zones, OpenVZ)

The Cloud

The different types of "cloud"

Unfortunately "cloud" is a term which is overloaded in the industry. This course refers to cloud in the Infrastructure as a Service (IaaS) sense, not the hosted applications sense.

There are currently two types of IaaS cloud being deployed. These are "public clouds", which are services such as Amazon's AWS, Rackspace's OpenStack based cloud offering or HP's OpenStack based cloud. There are a variety of companies of various sizes offering these public cloud services. These services are run in the hosting company's datacenter, and an administrator's access to the infrastructure is limited to launching and configuring virtual machines, normally through a web-based dashboard or an API.

The second type of cloud is "private clouds". These are conceptually similar to public clouds, but run in your own datacenter. The point here is that you retain complete control over the environment – if some of your virtual machines need access to unusual hardware, or store data which you're unwilling to move offsite, then private cloud might be a good choice. The open source OpenStack system is often used to build these private clouds, and there are various vendors offering private cloud solutions, including RackSpace, Piston, and Canonical.

Cloud providers comparison. Pros & Cons

Cloud management tools and APIs

Virtualization 201

Managing virtualized infrastructures (Private clouds)

Balancing CPU, Memory, and IO for guests

Leveraging virtualization for development

Leveraging virtualization for production

Security implications of virtualization

Logs 101

Common system logs & formats

There are generally three places where a process may be expected to send logs: stderr, a file, or syslog.

Standard Error

stderr is a file descriptor opened automatically (along with stdin and stdout) for any new process on a Unix system that is intended as a simple way for developers to provide feedback to a user without assuming any fixed format. stderr is always exposed to a process as file descriptor 2. If a process is launched from a shell attached to a tty, stderr usually appears inline with the rest of the program's output. In the shell, stderr may be redirected in order to differentiate it from stdout or to send it to a file.

The *find* command often writes messages to stderr upon encountering permissions issues. We'll use it as an example.

```
$ mkdir noperms
$ chmod 000 noperms/
$ find noperms/
noperms/
find: 'noperms/': Permission denied
```

The name of the directory *noperms/* is first written to stdout, then a permission denied error is written to stderr. Suppose we expected a large number of these errors to occur and wanted to append them in a log file, separate from the info written to stdout. Remember: stderr is always file descriptor 2.

```
$ find noperms/ 2>>find.log
noperms/
$ cat find.log
find: 'noperms/': Permission denied
```

It is important to note that we used the append >> operation here, rather than a single > which would overwrite find.log every time this command was run. This sort of redirection is a feature of the shell that is used quite commonly in init scripts and cron tasks to capture the output of both short lived commands and long running daemons.

Log files

The authors of many daemons add the ability to log directly to a file, rather than depending on the shell to provide redirection of stderr. There is no standard format or convention for a daemon to write it's own logs beyond opening a file and appending lines to it. Some daemons provide extremely flexible log configuration whereas others might only allow you to set a file path.

The *tail* command's *–follow* or *-f* flag is an extremely useful tool for viewing new lines appending to a file in realtime. For example, if you were running a web server like nginx or Apache that was configured to send it's access log to a file, you could use tail to see new requests.

```
$ tail -f /var/log/nginx/access.log
127.0.0.1 - - [17/Aug/2013:02:53:42 -0700] "GET / HTTP/1.1" 200 151 "-" "curl/7.22.0 (x86_64-pc-linux
127.0.0.1 - - [17/Aug/2013:02:53:43 -0700] "GET / HTTP/1.1" 200 151 "-" "curl/7.22.0 (x86_64-pc-linux
127.0.0.1 - - [17/Aug/2013:02:53:43 -0700] "GET / HTTP/1.1" 200 151 "-" "curl/7.22.0 (x86_64-pc-linux
```

With the *-f* option, tail won't exit on it's own, it will continue to wait for new lines to be written to the log file and write them to the terminal until it receives a signal or encounters an error.

It is important to understand that after a file is opened for writing, the process writing the file only refers to that file by its file handle, which is a number assigned by the kernel. If that file is renamed with *mv* or deleted with *rm*, writes to that file handle will still succeed. This can sometimes lead to counter-intuitive situations where log messages are being written to a file that's been renamed for archival or to an inode that no longer has a filename associated with it. Some daemons provide mechanisms for closing and reopening their log files upon receiving a signal like SIGHUP but quite a few don't.

Syslog

Logging directly to files can add a lot of complexity to an application as close attention has to be paid to the use of file handles, log directory permissions, timezones, etc. Syslog was created to provide a simple logging interface to application developers while offloading the tasks of sorting and storing logs to a separate daemon.

Every message sent to syslog has three pieces of information: facility, priority, and message. The facility tells syslog the type of information in the message. For example, login attempts are logged to the "auth" facility. The priority tells syslog the importance of the message. Syslog provides a fixed set of facilities and priorities that a program may use. Some examples of facilities are: auth, user, daemon, kern. Examples of priorities are debug, info, warn, critical.

All modern Linux distributions ship with a syslog daemon and most of them are pre-configured to write messages to various files in */var/log/*, depending on their facility or priority. While the exact names of these files are not consistent across different Linux distributions, a few common ones like */var/log/auth.log* and */var/log/kern.log* almost always exist. If you haven't already, take a look at the files in */var/log/* on your system to get a sense of the types of log messages available in these files.

Example log from a mail server (redacted to protect the guilty):

```
$ tail /var/log/mail.log
Aug 17 10:25:42 front01 postfix/smtpd[25791]: connect from unknown[115.77.XXX.XXX]
Aug 17 10:25:43 front01 postfix/smtpd[25791]: NOQUEUE: reject: RCPT from unknown[115.77.XXX.XXX]: 554
Aug 17 10:25:43 front01 postfix/smtpd[25791]: lost connection after RCPT from unknown[115.77.XXX.XXX]
Aug 17 10:25:43 front01 postfix/smtpd[25791]: disconnect from unknown[115.77.XXX.XXX]
```

One of the advantages of using a syslog daemon is that the format of log lines can be configured in a single place and standardized for all services using syslog on a single host. In this example, every line starts with a timestamp, the server's hostname, the name of the program and a PID. While the name of the program is set when the connection to syslog is first opened, the rest of these fields are generated by the syslog daemon itself and added to every line.

Many different syslog implementations exist with a variety of configuration mechanisms and design philosophies. Most current Linux distributions ship with a syslog daemon that implements some superset of the original Unix

syslogd's functionality. The following examples will use rsyslogd, which is currently included in Ubuntu Linux and according to it's manpage "is derived from the sysklogd package which in turn is derived from the stock BSD sources."

/etc/rsyslog.d/50-default.conf (truncated):

```
#
# First some standard log files.  Log by facility.
#
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
#daemon.*                       -/var/log/daemon.log
kern.*                          -/var/log/kern.log
#lpr.*                          -/var/log/lpr.log
mail.*                          -/var/log/mail.log
#user.*                         -/var/log/user.log

#
# Logging for the mail system.  Split it up so that
# it is easy to write scripts to parse these files.
#
#mail.info                      -/var/log/mail.info
#mail.warn                      -/var/log/mail.warn
mail.err                        /var/log/mail.err
```

Lines beginning with # are comments. Each line has two fields: a filter and an action. The filter is a comma-separated list of *facility.priority* where * is used as a wildcard, meaning match anything. The action is commonly just the name of a file, which causes all messages that match the filter to be written to that file. The actions in this example starting with - invert the behavior and cause messages matching the filter to be excluded from the given file. Many other flags and options are available here for configuring the exact behavior of log formatting and writing to places other than files.

As soon as you start to manage more than a couple of servers, you start to think about ways to aggregate the logs from all of those servers in a single place so that you don't have to login to each one individually to find an issue. Remote log aggregation is also often used to provide an audit trail for security events or a source of data that can be fed into a metrics system like Graphite or Ganglia. There is a standard protocol for sending syslog events over a network to another host over UDP port 514.

Configuring remote logging:

```
auth.*                          @10.0.0.2
```

As UDP is connectionless and makes no delivery guarantees, syslog messages sent to a remote host using this standard protocol can be dropped, delayed, or intercepted without any real indication to the user. For these reasons, many syslog daemons implement different extensions and mechanisms for transporting this stream reliably. The simplest option is to replace UDP with TCP to provide a reliable transport layer. When configuring syslog aggregation, attention and care should be paid to security as syslog messages are often used as an audit trail and need to be protected against eavesdropping and manipulation. Read your syslog daemon's documentation to understand what options are supported.

Log Rotation

No matter which logging option you choose, logging directly to files or using syslog, log files grow large and unwieldy over time and become difficult to use, for example identifying specific events.

To handle this problem, log files are rotated on a regular basis, by making a copy of the current log files and creating fresh log files. The old logs can be archived, compressed, mailed to an address or removed at predetermined intervals.

The Logrotate application eases management of systems that generate large numbers of log files. It allows automatic rotation, compression, removal, and mailing of log files. The log files may be handled at intervals (daily, weekly and monthly) or when they grow too large. It is usually scheduled to run daily.

Everything about the log files to be handled by logrotate as well as the actions to be carried out on them is read from the logrotate configuration files. The main configuration file is `/etc/logrotate.conf`. Applications can also create configuration files in the `/etc/logrotate.d` directory, logrotate automatically includes all configuration files in this directory.

```
# sample logrotate configuration file
compress

/var/log/messages {
    rotate 5
    weekly
    postrotate
        /sbin/killall -HUP syslogd
    endscript
}

"/var/log/httpd/access.log" /var/log/httpd/error.log {
    rotate 5
    mail foo@bar.org
    size=100k
    sharedscripts
    postrotate
         /sbin/killall -HUP httpd
    endscript
}
```

Lines beginning with # are comments and can appear anywhere in the configuration file. The first few lines set global options. In this example logs are compressed after rotation.

The next section defines how to handle the log file `/var/log/messages`. The log file is rotated weekly and removed after going through 5 rotations. The postrotate option defines a command to execute after the log file is rotated but before it is compressed. Postrotate is usually used to force daemons to reload their configurations so they will log to the new log file.

Log file names can be quoted or not quoted. Quoting allows matching file names with spaces in them. The second section defines how to handle two files, `/var/log/httpd/access.log` and `/var/log/httpd/error.log`. These logs files are rotated when they grow over 100k in size. The old log log files are mailed to foo@bar.org (uncompressed) after going through 5 rotations. The sharedscripts options means that the command for postrotate should be run only once no matter how many log files match. In this case although two files are handled, the command `/sbin/killall` is executed once.

There a lot more options available for logrotate, you can get a full list by checking the logrotate man page.

Logs 201

Centralized logging

Syslogd, Rsyslog

Benefits for developers

Log parsing

Search & Correlation

(Splunk, Logstash, Loggly, etc)

Databases 101 (Relational Databases)

What is a Database?

A database is a program or library that helps you store data. They usually impose some sort of structure to the data to assist with querying and filtering. The most common structure databases use to store data is a table, and most databases will use multiple tables to store data.

A table is composed of rows representing an item, and columns represent some attribute of that item. Much like using a spreadsheet for a task list, where a row would be a task, and you may have a column for a task name, and a second column to determine whether it has been completed.

What is a Relational Database?

A relational database is a database in which data is organized into tables, usually with some form of relationship between them. For example, a table containing customer names and addresses, and a table containing sales transactions. Rather than repeat customer details every time a transaction is done, the data is stored once in one table, and then a unique reference is stored in the sales transactions table, linking each transaction to the customer. This approach makes a relational database very efficient for storing data, since reused data can be stored in a single table and referenced in other tables.

Data can be retrieved in many forms by combining multiple tables into new tables using operations like `JOIN`. The composed table will have new rows that are a combination of its parent tables.

Why We Use Databases?

We use databases to store data. They help provide us guarantees that data is stored and that it exists in the correct format. In addition most databases are heavily optimized making data retrieval very fast.

Because of these attributes, we use databases a lot. Most e-commerce sites use databases to keep inventory and sales records. Doctors offices use databases to store medical records, and the DMV uses databases to keep track of cars. Lawyers use databases to keep track of case law, and many websites use databases to store and organize content. Databases are everywhere, and you interact with them daily.

What is SQL?

SQL or Structured Query Language is a *domain specific language* used for accessing databases. It has a declarative syntax, meaning you declare the structure you want returned, the sources, and constraints in a single statement. The database's query parser and planner will determine a how to retrieve your data from this statement.

SQL shell

Many relational databases provide an interactive *CLI* for interacting with the database. For example, MySQL provides the `mysql` command, Postgresql provides `psql`, and Oracle provides `sqlplus`. These programs give you the ability to compose queries, and diagnose issues with the software.

todo:: Add example of connecting with each command.

MySQL

To connect to a MySQL database from the CLI your command would usually take this form:

```
$ mysql -u username -p -h server-address
Password:
```

Of these flags, `-u` = username, `-p` = password, and `-h` = hostname. You may provide the password on the command prompt: `-ppassword` (no space after flag!). We strongly advise against this, as this may leave the password visible in your shell history.

Creating databases

Most database platforms allow you to create a new database using the `CREATE DATABASE` SQL query. It can be executed via a connection to the server, or via the SQL shell.

A MySQL example:

```
mysql> CREATE DATABASE example_database;
```

Specialized Create Database Commands

Some database platforms have specialized *GUI* tools, or CLI tools. For example Postgresql has a UNIX command `createdb` that creates databases. In many cases these commands are just wrappers around the standard SQL statements.

MySQL

```
$ mysqladmin create example_database
```

Postgresql

```
$ createdb example_database
```

Some platforms, like MySQL support multiple databases per instance, while other platforms like Oracle support one database per instance. You should check the documentation for your particular vendor to see what is supported.

Creating users

Users can be created in most databases using the the `CREATE USER` statement.

```
mysql> CREATE USER username;
```

Specialized Create User Commands

Some relational databases provide additional ways of creating users like specialized command line programs.

MySQL

MySQL does not support creation of users via the `mysqladmin` command.

Postgresql

```
$ createuser username
```

Create Tables

Tables are organized into rows and columns. Data is stored inside these tables. In order to host this information we need to create a table. We do this with the `CREATE TABLE` statement.

Standard Syntax is:

```
CREATE TABLE table_name (
column1 datatype(size),
column2 datatype(size),
column3 datatype(size)
);
```

Here is an example. This will create a table called "Users" with 2 columns.

```
CREATE TABLE Users (
name varchar(50),
address varchar(50)
);
```

Alter Table

The `ALTER` statement will allow you to modify the design of your current database. For example, if you have a table called "Users" which contains Names and Addresses, but you need need to add an age column, you could do so with:

```
ALTER TABLE Users ADD age int;
```

Standard Syntax is:

```
ALTER TABLE table_name ADD column_name datatype
```

If we need to modify a current table, we can also do so with the `ALTER` statement. If in the previous example you wanted to change the datatype of age from an integer to varchar, you would do so with:

```
ALTER TABLE Users MODIFY age varchar(50);
```

Standard Syntax is:

```
ALTER TABLE table_name MODIFY column_name datatype
```

If you now realize you don't need the age column at all, you can drop it all together with:

```
ALTER TABLE Users DROP age;
```

Standard Syntax is:

```
ALTER TABLE table_name DROP column_name
```

Drop Table

If you have created a table and need to remove it from your database, you can with the `DROP` statement:

Standard Syntax is:

```
DROP table table_name
```

If we want to drop our table we created earlier (Users) we would use this command:

```
DROP table Users;
```

Data Type

There are different types of data that we will store in our database. It can be simple text or a number, sometimes it is a mix of these things. The table below lists some common Standard SQL commands that are also in MySQL. It is important to note this is not a complete list.

Data type	Description
char(n)	Fixed width character string, can hold letters and numbers-Max of 8,000 characters
varchar(n)	Variable width character string, can hold letters and numbers- Max of 8,000 characters
text	Variable width character string. Value is represented internally by a separately allocated object
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647
float	A small number with a floating decimal point
decimal	An exact numeric data value, allowing for a fixed decimal point
date	Date formated as YYYY-MM-DD
datetime	Date and time combination formated as YYYY-MM-DD HH:MM:SS
timestamp	Stores a unique number that gets updated every time a row gets created or modified

Granting privileges

Privileges can be granted using the SQL GRANT statement. These statements are persisted by the RDBMS when issued. The typical command format is:

```
GRANT [PRIVILEGE] on [OBJECT] to [USER];
```

The standard SQL privileges are:

Privilege	Description
ALL	Allows user all privileges
ALTER	Allows user to alter schema objects
CREATE	Allows user to create schema object like tables
DELETE	Allows user to delete from an object
EXECUTE	Allows user to execute a store procedure or function
INSERT	Allows user to add new data to an object
REFERENCES	Allows user to create a referential table constraint
SELECT	Allows user to read from an object
TRIGGER	Allows user to create a trigger
TEMPORARY	Allows user to create temporary tables
UPDATE	Allows user to update existing data in an object

Below is an example granting a user SELECT privileges on a table

```
GRANT SELECT ON TABLE example_table TO user;
```

You can also grant multiple privileges in a single statement.

```
GRANT SELECT,INSERT ON TABLE example_table TO user;
```

Many databases stray from the SQL standard here, and it is important to read your database's documentation when granting privileges. There may be additional privileges not listed here and syntax can vary significantly.

Removing Privileges

Privileges are removed with the SQL `REVOKE` statement. It follows a similar command format like grant:

```
REVOKE [PRIVILEGE] on [OBJECT] FROM [USER];
```

Basic normalized schema design

A normalized schema is a database with a table and column structure designed to reduce data redundancy. Typically data is placed into tables with a unique identifier, or primary key, and then is referenced by id in any tables that wish to use that data.

Suppose we have two types of records in a database; one for a city's population and one for a city's average temperature. We could simply create the tables like so:

City Population:

City	Population
San Francisco	812,826

City Temperature:

City	Temperature
San Francisco	73 F

A normalized version of this would have three tables instead of two.

City ID and Name:

City_id	Name
1	San Francisco

City ID and Temperature:

City_id	Temperature
1	73 F

City ID and Population:

City_id	Population
1	812,826

The advantage of this design is that it prevents you from having to enter data multiple times, and generally reduces the storage cost of a row. If San Francisco changed its name you would only need to update a single row instead of two tables like the first example. SQL allows you to replace the `id` with a name using a `JOIN` statement when the data is retrieved, making it functionally identical to the two table example.

Select, Insert, Update and Delete

SELECT

The `SELECT` statement is the standard way you read from a table in an SQL database. You can use it to retrieve a set of data, and perform aggregations on them. The standard syntax is:

```
SELECT [column1,column2|*] FROM [TABLE];
```

By adding a `WHERE` statement, you can have the database filter results:

```
SELECT user_id, user_name FROM users WHERE user_id = 1;
```

You can join tables using a `JOIN` statement. In this example we're temporarily assigning an alias of 'u' for the table users and an alias of 'a' for the table addresses:

```
SELECT user_id, user_name FROM users u JOIN addresses a
ON u.user_id = a.user_id  WHERE user_id = 1;
```

Count the rows in a table by using an aggregation:

```
SELECT COUNT(1) FROM users;
```

Order by a column:

```
SELECT * FROM users ORDER BY user_name;
```

INSERT

The `INSERT` statement is used to add additional data into a table. It can be used to insert data either a row at a time or in bulk. The standard syntax is:

```
INSERT INTO table (column1, column2, column3) VALUES (value1, value2, value2)
```

The column list is optional, if you don't specify which columns you're inserting data into, you must provide data for all columns.

For example, to insert a single row:

```
INSERT INTO users (user_name,user_phone) VALUES ("Joe Bloggs","555-1234");
```

Or in bulk:

```
INSERT INTO users (user_name,user_phone)
VALUES ("John Smith","555-5555"),("Tom Jones","555-0987");
```

Inserting in bulk like that is typically much quicker than using separate queries as the query planner only has to execute once, and any indexes are updated at the end.

UPDATE

`UPDATE` is the SQL statement for updating existing data in a table. It should almost always be used with a conditional statement. The standard syntax is:

```
UPDATE [TABLE] SET [COLUMN] = {expression}, {COLUMN2={expression}, ...}
[WHERE condition]
[ORDER BY ...]
[LIMIT count];
```

Without a `WHERE` condition, the statement will apply to all the rows in a table.

Here is a simple example of an `UPDATE` statement:

```
UPDATE users SET user_name = 'Jim Smith' WHERE user_name = 'James Smith';
```

DELETE

`DELETE` is the SQL statement for removing rows from a table. The standard syntax is:

```
DELETE FROM [TABLE]
[WHERE condition]
[ORDER BY ...]
[LIMIT count] ;
```

Note: Without a `WHERE` condition the statement will apply to **all** the rows of a table.

Here is a simple example of a DELETE statement:

```
DELETE FROM users WHERE user_name = 'James Smith';
```

Pro Tips

- Before doing a write query, run it as a read query first to make sure you are retrieveing exactly what you want. If your query is:

  ```
  UPDATE users SET disabled=1 WHERE id=1;
  ```

  Run this first to validate you will be affecting the proper record:

  ```
  SELECT disabled FROM users WHERE id=1;
  ```

- use a `LIMIT` on `UPDATE` and `DELETE FROM` queries to limit damage imposed by an erroneous query

  ```
  UPDATE users SET disabled=1 WHERE id=1 LIMIT 1;
  ```

  ```
  DELETE FROM users WHERE id=1 LIMIT 1;
  ```

- If your database supports transactions, run `START TRANSACTION` first then run your query and check what it has done. If you're happy with what you see then run `COMMIT` and finally `STOP TRANSACTION`. If you realize you've made a mistake, you can run `ROLLBACK` and any changes you've made will be undone.

Databases 201

Database Theory

ACID

CAP theorem

High Availability

Document Databases

MongoDB

CouchDB

Hadoop

Key-value Stores

Riak

Cassandra

Dynamo

BigTable

Graph Databases

FlockDB

Neo4j

Application Components 201

Message Queue Systems

In a distributed system a Message Queue Systems can provide a basic infrastructure to higher level functions like

- decoupling processes or hosts from each other
- parallel processes execution
- distribute events and information like changes to data, entries from logfiles and statistics for monitoring and notification
- data-streaming and file-transfers to multiple hosts via multicast and unicast
- replace polling mechanism with an event orientated system
- network wide accessible job queues
- message rerouting on failure / escalation schemes

- verification of results (best of three)
- asynchronous remote procedure calls
- network wide mutex
- network or distributed system wide message bus

Message Queue Systems are more like peer2peer networks then client-server applications. They can be split up into message brokers and routers or brokerless message queuing systems. [0MQ] A message as understood by the system is everything that can be represented as a bytestream. Properties like type and timestamp may are added to message. [Bok]

Further Readings

- http://c2.com/cgi/wiki?MessageQueuingArchitectures
- https://en.wikipedia.org/wiki/Message-oriented_middleware and https://en.wikipedia.org/wiki/Message_oriented_middleware
- https://en.wikipedia.org/wiki/Queueing_theory
- https://en.wikipedia.org/wiki/Message_queue
- http://www.slideshare.net/mwillbanks/art-of-message-queues

Message Brokers

Message brokers represent a message queue systems that relies on a central system to route messages to its destination. Messages are send to exchanges and received from queues. Queues resists on the broker, also some protocols implement a client site queue for multipart transactions and prefetching. Message Brokers tend to become a critical system in an otherwise distributed environment.

RabbitMQ

RabbitMQ is an open source Message Broker written in erlang. Its queuing protocol is the Advance Message Queuing Protocol. [wiki] [specs] It provides plugins for different queuing protocols like STOMP and advanced setups. Installations scale from single host as an applicationmessagebus, to multi cluster installations in different networks. Messages are send to different type of exchanges. While Direct, Fanout, Topic and Header Exchanges are part of the core system, others plugins are rooted in the community. No prior understanding of the erlang programming language is needed to setup, configure and operate the message broker. The documentation on the projects homepage reads a fair amount of different configurations with explanations.

To setup a cluster a minimum of two nodes is required. All nodes of a cluster must be in the same IP network segment. It is possible to send huge messages via AMQP to a RabbitMQ node. The maximum message size is limited by the amount of available RAM on the node. Depending on the durability of a message and the policy of queues, the messages are synchronized to other nodes. A disc-node ensures a message is stored to a disc before delivering. Memory based nodes provide a much higher message throughput. To establish an encrypted communication between nodes, IPSec or TLS can be used. Authentication is possible via an internal database, LDAP, SASL and PKI client certificates. With a PKI in place consumers and publisher can share the same authorization. No preshared password among all processes is needed, but different private keys are mandatory. This is accomplished by using the commonname of the dn in the x509 client certificate as username. Different applications can be separated by the concept of virtual hosts in a similar way the Apache web server does. Configuration is provided by webGUI or commandline tools. Programming libraries and tools for a wide range of environments are available.

The installation from the Debian repository leads to a configuration with a single host. Similar installation instructions are provided for Windows, Solaris and other operation systems.

```
curl 'https://www.rabbitmq.com/rabbitmq-signing-key-public.asc' | sudo apt-key add -
echo 'deb http://www.rabbitmq.com/debian/ testing main' | sudo tee /etc/sources.d/rabbitmq.list
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install rabbitmq-server
```

To backup and restore configuration data the management plugin should be configured.

```
{ rabbitmq_management, [
    {  listener, [ { port, 15672 },
                   { ssl, true },
                   { ssl_opts, [ { cacertfile, "/etc/ssl/certs/cacert.crt" },
                                 { certfile,   "/etc/ssl/certs/node1.crt" },
                                 { keyfile,    "/etc/ssl/private/node1.key" } ]}
                ]} // configured listener
]} // configured rabbitmq_management
]. // EOC

rabbitmqadmin export rabbit.config
rabbitmqadmin -q import rabbit.config
```

Show a detailed report about queues, users and connections

```
rabbitmqctl report
```

Apache ActiveMQ

Memory Caches

Memcached

Redis

Specialized Caches

Varnish

nginx+memcached

Load Balancing

Why do we use load balancers?

Scalability

The most basic scalability need is to allow one address handle a load that's too much for one physical server. A secondary scalability factor is offloading or reducing underlying tasks, such as SSL termination and re-using already-established TCP links between the load balancer and the servers to reduce the time taken to send data to the user. Smart load balancers that are able to poll and react to client load can seamlessly distribute work among heterogeneous machines.

Reliability

Simple health checking alone increases reliability. A load balancer that simply polls an endpoint like status/health can detect a machine or application that has failed and remove it from the pool of active servers. More advanced health checking can remove partially failed systems that are exhibiting poor or intermittent performance.

Agility

A load balancer can assist in making application development faster too. Load balancers could be configured to route requests using higher level information about the connection, such as certain URLs served by certain types of servers, or sending everybody who logs in as an employee to an internal staging server instead of production. Finally, a load balancer permits various different strategies for introducing new deployments in stages.

Application implications

Backend

An application server behind a load balancer is typically referred to as a "backend" or "backend server".

Pools

A collection of backends is referred to as a pool.

Balancing algorithms

Session Affinity

Session affinity or session stickiness, is when the load balancer applies an algorithm to the incoming connection to direct it to a single server. This is typically done for HTTP applications by setting cookies. In general TCP applications often use IP addresses to determine the server to direct the connection to.

Local & ISP caching

SSL Termination and Offloading

SSL Termination is when the load balancer established an SSL connection between the client and the load balancer and a non-encrypted connection between the load balancer and backend server.

Terminating SSL on the load balancer eliminates the need to distribute your certificate and key amongst all the servers. When using hardware load balancers they typically have special hardware acceleration which is much more performant compared to terminating connections on the backend server.

Balancing vs. Failover (Active / Passive)

Often load balancing is used as a high-availability technique, by allowing multiple backends to service a request if one node should become unavailable. It differs from failover configuration because all nodes generally participate in servicing clients. In failover configurations a single (active) node handles all requests until an issue arises, and the secondary (passive) node takes over all of the incoming traffic. Failover configurations are usually not configured for scaling purposes.

Health Checks

Most load balancers have some ability to test backend servers for availability, these are called health checks. They can be as simple as whether hosts are responding on the port the application is bound to, or complex configurations that test special URIs, or response times. If a server fails a health check the load balancer will temporarily remove the node from the pool, until it successfully passes the health check.

Non-HTTP use cases

Native SMTP, or general TCP for things like RabbitMQ, MongoDB, etc

Software

Apache

Apache has the ability to load balance using `mod_proxy_balancer`, and `mod_jk`.

Mod_proxy_balancer is purpose-built load balancing solution. It supports the HTTP, FTP, and AJP protocols. There is basic support for sessions stickiness's, weighted round-robin, and can remove unhealthy backends from the balancing pool. It lacks support for customizable health checks, other TCP protocols. Support for AJP is provided by `mod_proxy_ajp` and support for FTP is provided by `mod_proxy_ftp`.

`Mod_jk` supports the AJPv1.2 and AJPv1.3 protocols. It's a purpose build connector to support the Java application servers, like Apache Tomcat. It supports weighted round-robin, session stickiness, session counting, traffic based routing, and backend busyness (server with the least connections). It supports health checks using the method built into AJP protocol.

Nginx

Nginx reverse proxy implementation supports load balancing for HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached. The default method is round-robin, but session persistence/stickiness (`ip_hash`) and least connections (`least_conn`) methods are available. Flags for weighted load balancing and passive server health checks can also be set on any of the backends.

Nginx load balancing documentation

HAProxy

HAProxy is a general TCP load balancing server that is highly configurable. It will generally support any TCP based protocol, and has special modes for HTTP, RDP, MySQL, and Postgresql protocols. It has support for multiple types of health check including URL based, traffic-based health, and external checks via the `httpchk` options. It has several load balancing algorithms: round robin, static round-robin, least connections, source hashing, URI hashing, URI parameter, and RDP-cookie.

Hardware

BIG-IP

BIG-IP has purpose-built hardware load balancers. They support protocols in layers 2, 4, and 7 of the OSI model. They allow for very complex configurations, and support writing special TCL programs to modify the load balancing behavior. The product supports SSL termination and offloading, with additional licensing.

Netscaler

Multi-dc

Anycast

DNS GSLB

- A GSLB (Global Site Load Balancer) at the most simplistic level is a health checking DNS server.
- Most often used to load balance between geographically dispersed data centers.
- Generally has health check mechanisms similar to load balancers which are used to return an IP address (as part of the DNS lookup) of a host that is currently available to service the request.
- Conceptually provides coarse-grained round robin and affinity balancing algorithms by setting the time to live (TTL) of the DNS lookup for an appropriate duration.

CDN's

(cparedes: I'd argue that it's valid in some contexts, depending on what you're load balancing)

Monitoring, Notifications, and Metrics 101

History: How we used to monitor, and how we got better (monitors as tests)

Perspective (end-to-end) vs Introspective monitoring

Metrics: what to collect, what to do with them

Common tools

Syslog (basics!)

Syslog-ng

Nagios

Nagios is a monitoring tool created by Ethan Galstad. It was originally released as a opensource project named NetSaint in 1999. Due to trademark issues it was later renamed Nagios. Since then it has become one of if not the most common monitoring tool for production environments.

Nagios is primarily an alerting tool that will notify an administrator or group of administrators if a service enters a critical or warning state if desired. It has a basic web interface that allows for acknowledgment of issues and scheduling downtime.

Nagios is highly configurable and extensible due to its reliance on external commands to perform almost every task. For example, every availability test in Nagios is a standard executable, and notifications are generated by an external command. Because of this Nagios does not restrict the administrator to using a particular language to extend the system, and often times plug-ins and tests are written in a variety of languages.

The feature set of Nagios is pretty extensive. It supports service and host altering hierarchies via parenting, so you can reduce the number of alerts when a critical service or host fails. It supports active and passive checks. It has basic scheduling for on-call periods, and supports time periods in which you can disable alerting.

Since Nagios is so configurable, it can often be difficult to configure for the uninitiated. It can use many files for configuration, and a single syntax error will prevent the system from starting. Additionally, the open-source version does not natively support adding and removing hosts dynamically; the configuration needs to be modified, and the server reloaded to add or remove a host.

Graphite

Graphite is an open-source monitoring tool that let's you store time-series based data and graph them.

Graphite consists of three components:

- Carbon: A daemon which listens for time-series data.
- Whisper: Similar to RRD, it's a time series database.
- Webapp: A Django based webapp which helps visualize beautiful graphs on the data collected.

Graphite does not collect metrics for you, however there are three methods using which you can send data to Graphite:

- PlainText
- Pickle
- AMQP

Apart from the out-of-box web-app that Graphite comes with you can even create your own applications on top of Graphite by retrieving data in form of JSON, CSV or raw-data format.

Ganglia

Ganglia is a highly scalable distributed monitoring solution for high performance systems like clusters and grids.

Ganglia leverages the following technologies:

- XML for data representation.
- XDR for compact and portable data transport.
- RRD for data-storage.
- PHP and RRDTool for data visualization.

Ganglia is organized using the following conventions:

- Grid: Consists of clusters.
- Clusters: Consists of hosts/nodes. This is a logical grouping of machines and metrics like database servers, qa servers, etc.
- Host/Node: Typically a machine/server.

Ganglia monitoring suite consists of three main components:

- gmond: Daemon which needs to sit on every single node which needs to be monitored, gather monitoring statistics, send as well as receive the stats to and from within the same multicast or unicast channel.
- gmetad: daemon that polls gmonds periodically and stores their metrics into a storage engine like RRD. It can poll multiple clusters and aggregate the metrics. It is also used by the web frontend in generating the UI.
- ganglia-web: It should sit on the same machine as gmetad as it needs access to the RRD files.

To get a feeling of what features Ganglia has, you can look at the demo at http://ganglia.wikimedia.org/latest/

Munin

RRDTool / cacti

Icinga

SNMP

Simple Network Management Protocol or SNMP, is a monitoring and management protocol. It is the standard way of monitoring on switches, routers, and other networking equipment. SNMP relies on an agents which when contacted by a management system return the information requested. The data provided by the agent uses Object Identifiers or OIDs that provide information about the current system. OIDs can contain anything from strings identifying information about the system, to total number of frames received by the Ethernet controller. Devices and systems often are provided with MIBs or Management Information Base these help the management system identify the information contained in the OID. Lastly, management systems request information by providing a community string, for example Public. These community strings allow the agent to determine what information is appropriate to return to the requester, and whether the requesting system has read-only or read-write access.

There are three commonly used versions of the protocol, SNMPv1, SNMPv2c and SNMPv3. SNMPv3 is the only cryptographically secure version of the protocol. Most devices will have support at least two versions of SNMP.

Collectd

Collectd collects system-level metrics on each machine. It works by loading a list of plugins, and polls data from various sources. The data are sent to different backend (Graphite, Riemann) and can be used to trigger alerts with Nagios.

Sensu

Sensu was written as a highly configurable, Nagios replacement. Sensu can be described as a "monitoring router", since it connects check scripts across any number of systems with handler scripts run on one or more Sensu servers. It is compatible with existing Nagios checks and additional checks can be written in any language similar to writing Nagios checks. Check scripts can send alert data to one or more handlers for flexible notifications. Sensu provides the server, client, API and dashboard needed to build a complete monitoring system.

Diamond

Diamond is a Python daemon that collects system metrics and publishes them to Graphite (and others). It is capable of collecting cpu, memory, network, i/o, load and disk metrics. Additionally, it features an API for implementing custom collectors for gathering metrics from almost any source.

Logster

Logster project was created at Etsy as a fork of ganglia-logtailer .

Logster is a utility for reading log files and generating metrics in:

- Graphite
- Ganglia

- Amazon CloudWatch

It is ideal for visualizing trends of events that are occurring in any type of logs:

- Application
- System
- Error logs

For example, you might use Logster to graph the number of occurrences of HTTP response code that appears in your web server logs.

Logster maintains a cursor, via logtail on each log file that it reads so that each successive execution only inspects new log entries.

A simple,1 minute crontab entry for logster would allow you to generate near real-time trends for anything you want to measure from your logs.

This tool is made up of a framework script, Logster, and parsing scripts that are written to accommodate your specific log format.

Sample parsers are included in the distribution, which essentially read a log file line by line, applying a regular expression to extract useful data from the lines you are interested in, and then aggregate that data into metrics that will be submitted to Ganglia or Graphite or Amazon CloudWatch.

Do take a look through the sample parsers, which should give you some idea of how to get started writing your own.

Monitoring, Notifications, and Metrics 201

Dataviz & Graphing

Graphite, StatsD

What are Graphite and StatsD?

In order to understand what your application is doing and to get a better overview over key functionality it is indispensable to gather metrics about those areas. A popular combination to achieve this is StatsD with the Graphite storage and graphing backend. This chapter will introduce those tools, how to install and configure them and how they can be used to gather insight into your application's metrics.

Graphite

The Graphite project is a set of tools for collecting and visualizing data, all written in Python. It consists of a file based database format and tools to work with it called *whisper*, a set of network daemons to collect data - the *carbon* daemons - and a Django based web application to display data.

Whisper database format

Graphite at its core uses a disk-based storage system called *whisper*. Every metric is stored in its corresponding `.wsp` file. These are fixed sized databases, meaning that the amount of data a file can hold is pre-determined upon creation. In order to achieve this, every database file contains so called *archives* which store *(timestamp, value)* pairs with varying granularity and length. For example a possible archive setup could be to store a data point every 10 seconds for 6 hours, after that store 1 minute data for a week and for long time archival use a data point every 10 minutes for 5 years.

This means once your data is older than the time period stored in the archive, a decision has to be made how to fit the data into the coarser slots in the next archive. In our example setup when going from the highest precision (10 seconds) to the next archive (1 minute) we have 6 values at our disposal to aggregate. The database can be configured to aggregate by average, sum, last, max or min functions, which uses the arithmetic mean, the sum of all values, the last one recorded or the biggest or smallest value respectively to fill the value in the next archive. This process is done every time a value is too old to be stored in an archive until it is older than the lowest resolution archive available and isn't stored anymore at all.

An important setting in this context is the *x-files-factor* you can set on every database. This describes how many of the archives slots have to have a value (as opposed to `None` which is Python's `NULL` value and recorded when no value is received for a timestamp) to aggregate the next lower archive to a value and not also insert `None` instead. The value range for this setting is decimal from 0 to 1 and defaults to 0.5. This means when we use the default setting on our example archives, at least 3 of the 6 values for the first aggregation and 5 of the 10 values for the second one have to have actual values.

The disk format to store all this data is rather simple. Every file has a short header with the basic information about the aggregation function used, the maximum retention period available, the *x-files-factor* and the number of archives it contains. These are stored as 2 *longs*, a *float* and another *long* thus requiring 16 bytes of storage. After that the archives are appended to the file with their *(timestamp, value)* pairs stored as a *long* and a *double* value consuming 12 bytes per pair.

This design makes it possible to store a lot of datapoints relatively easy and without consuming a lot of disk space (1 year's worth of 1 minute data for a metric can be stored in a little more than 6MB). But it also means that some considerations about usage have to be made upfront. Due to the database's fixed size, the highest resolution archive should match the rate at which you send data. If data is sent at lower intervals the archive ends up with a lot of `None` values and if more than one metric are received for a timestamp, the last one will always overwrite previous ones. So if a low rate of events during some times is expected, it might make sense to tune down the *x-files-factor* to aggregate even if only one of ten values is available. And depending on the type of metrics (counters for example) it might also make more sense to aggregate with the *sum* function instead of the default aggregation by averaging the values. Thinking about this before sending metrics makes things a lot easier since it's possible to change these settings afterwards, however to keep the existing data, the whisper file has to be resized which takes some time and makes the file unavailable for storage during that time.

In order to understand whisper files a little bit better, we can use the set of scripts distributed with whisper to take a look at a database file. First install whisper from the PyPi distribution:

```
% sudo pip install whisper
```

And create a whisper database with a 10 second retention for 1 minute by using the `whisper-create.py` command:

```
% whisper-create.py test.wsp 10s:1minute
Created: test.wsp (100 bytes)

% whisper-info.py test.wsp
maxRetention: 60
xFilesFactor: 0.5
aggregationMethod: average
fileSize: 100

Archive 0
retention: 60
secondsPerPoint: 10
points: 6
size: 72
offset: 28

% whisper-fetch.py test.wsp
```

```
1354509290      None
1354509300      None
1354509310      None
1354509320      None
1354509330      None
1354509340      None
```

The resulting file has six ten second buckets corresponding to the retention period in the create command. As it is visible from the `whisper-info.py` output, the database uses default values for *x-files-factor* and aggregation method, since we didn't specify anything different. And we also only have one archive which stores data at 10 seconds per value for 60 seconds as we passed as a command argument. By default `whisper-fetch.py` shows timestamps as epoch time. There is also `--pretty` option to show them in a more human readable format, but since the exact time is not important all examples show epoch time. For updating the database with value there is the handy `whisper-update.py` command, which takes a timestamp and a value as arguments:

```
% whisper-update.py test.wsp 1354509710:3
% whisper-fetch.py test.wsp
1354509690      None
1354509700      None
1354509710      3.000000
1354509720      None
1354509730      None
1354509740      None
```

Notice how the timestamps are not the same as in the example above, because more than a minute has past since then and if we had values stored at those points, they wouldn't be show anymore. However taking a look at the database file with `whisper-dump.py` reveals a little more information about the storage system:

```
% whisper-dump.py test.wsp
Meta data:
aggregation method: average
max retention: 60
xFilesFactor: 0.5

Archive 0 info:
offset: 28
seconds per point: 10
points: 6
retention: 60
size: 72

Archive 0 data:
0: 1354509710,          3
1: 0,          0
2: 0,          0
3: 0,          0
4: 0,          0
5: 0,          0
```

In addition to the metadata `whisper-info.py` already showed, the dump command also tells us that only one slot actually has data. And in this case the time passed doesn't matter. Since slots are only changed when new values need to be written to them, this old value will remain there until then. The reason why `whisper-fetch.py` doesn't show these past values is because it will only show valid data within a given time (default 24h) until *max retention* from the invoked point in time. And it will also fetch the points from the retention archive that can cover most of the requested time. This becomes a bit more clear when adding a new archive:

```
% whisper-resize.py test.wsp 10s:1min 20s:2min
Retrieving all data from the archives
```

```
Creating new whisper database: test.wsp.tmp
Created: test.wsp.tmp (184 bytes)
Migrating data...
Renaming old database to: test.wsp.bak
Renaming new database to: test.wsp

% whisper-info.py test.wsp
maxRetention: 120
xFilesFactor: 0.5
aggregationMethod: average
fileSize: 184

Archive 0
retention: 60
secondsPerPoint: 10
points: 6
size: 72
offset: 40

Archive 1
retention: 120
secondsPerPoint: 20
points: 6
size: 72
offset: 112

% whisper-fetch.py test.wsp
1354514740      None
1354514760      None
1354514780      None
1354514800      None
1354514820      None
1354514840      None
```

Now the database has a second archive which stores 20 second data for 2 minutes and `whisper-fetch.py` returns 20 second slots. That's because it tries to retrieve as close to 24h (the default time) as possible and the 20 second slot archive is closer to that. For getting data in 10 second slots, the command has to be invoked with the `--from=` parameter and an epoch timestamp less than 1 minute in the past.

These commands are a good way to inspect whisper files and to get a basic understanding how data is stored. So it makes sense to experiment with them a bit before going into the rest of the Graphite eco-system.

The carbon daemons

In order to make whisper files accessible to be written to from other network services, the Graphite project includes the *carbon* daemon suite. The suite consists of a *carbon-cache*, *carbon-relay* and *carbon-aggregator* daemon, which are all based on the Twisted framework for event-driven IO in Python.

The *carbon-cache* daemon is the most crucial of them as it provides the basic interface to the whisper backend and a scalable and efficient way for a large number of clients to store metrics. In order to minimize write delay for a big number of metrics depending on the disk seek time (each metric has its own file) the daemon employs queuing. Every metric has its own queue and an incoming value for a metric gets appended to it. A background thread then checks the queues for data points and writes them consecutively to the storage file. This way cost of an expensive disk seek gets amortized over several metric values that are written with one seek.

The daemon relies on two config files, `carbon.conf` for general configuration and `storage-schemas.conf` for whisper storage configuration. The general configuration file contains settings like network configuration (*carbon-*

cache can listen on different sockets like plain TCP and UDP or even AMQP), cache sizes and maximum updates per second in its `[cache]` section. These settings are very useful when tuning the carbon daemon for the hardware it's running on, but to get started the default settings from the example config files will suffice. The storage schemas configuration file contains information about which metrics paths are using which retention archives and aggregation methods. A basic entry looks like this:

```
[default_1min_for_1day]
pattern = .*
retentions = 60s:1d
```

Each section has a name and a regex pattern which will be matched on the metrics path sent. The pattern shown above will match any pattern and can be used as a catch-all rule at the end of the configuration to match uncaught metrics. The `retentions` section is a comma separated list of retention archives to use for the metrics path in the same format that `whisper-create.py` expects them.

In order to get a basic carbon cache instance running (default listener is TCP on port 2003), install it from PyPi and copy the example config files:

```
% cd /opt/graphite/conf
% cp carbon.conf.example carbon.conf
% cp storage-schemas.conf.example storage-schemas.conf
% /opt/graphite/bin/carbon-cache.py start
Starting carbon-cache (instance a)

% netstat -an | grep 2003
tcp4       0      0  *.2003                 *.*                    LISTEN
```

The default installation creates its directories in `/opt/graphite` but this can also be changed within the configuration. After the carbon daemon has been started, metrics can just be recorded by sending one or more values in the format `metric_path value timestamp\n`:

```
% echo "test 10 1354519378" | nc -w1 localhost 2003
% whisper-fetch.py /opt/graphite/storage/whisper/test.wsp |
tail -n 3
1354519260      None
1354519320      10.000000
1354519380      None
```

All metrics paths that are sent are relative to the `/opt/graphite/storage/whisper` directory and will be stored there. The interface also supports sub folders, which can be created by separate the metrics path with dots:

```
% echo "this.is.a.test 10 1354519680" | nc -w1 localhost 2003
% whisper-fetch.py /opt/graphite/storage/whisper/this/is/a/test.wsp| tail -n 3
1354519560      None
1354519620      None
1354519680      10.000000
```

This is all that's needed to collect metrics over the network. As mentioned before, the carbon suite contains two more daemons `carbon-relay` and `carbon-aggregator`. These can be used to improve the performance of the system under higher load.

`carbon-relay` acts as a router between different `carbon-cache` or `carbon-aggregator` instances. The daemon reads the `[relay]` section of the `carbon.conf` configuration file where the most important sections are the interface and TCP port to listen to via the `LINE_RECEIVER_*` settings, the `DESTINATIONS` property, a comma separated list of `ipaddress:port` pairs of available carbon daemons and the type of relaying to use. The `carbon-relay` can be operated in rule based or consistent hashing based relaying mode. In the second case, consistent hashing is employed to balance the metrics between available destinations. When using the relay based approach, the relay daemon needs a `relay-rules.conf` configuration file of the form:

```
[name]
pattern = <regex>
destinations = <list of destination addresses>
```

This follows the storage schema configuration file format and will route any metric matching the pattern to the given destinations. There also has to be exactly one section which additionally has the property `default = true` which is used as the catch all rule if no other rule has matched a metric path.

The Graphite web application

Normalizing the Metrics

To easily navigate within hundreds of metrics, it's important to normalize the name. Here are a few naming schemes:

- `<ENV>.<CLUSTER>.<SERVER>.metric`
- `<ENV>.<APPLICATION-TYPE>.<APPLICATION>.metric`
- `<ENV>.APPLICATIONS.EVENTS.<APP-NAME>.deploy`

Here a couple rules to choose an appropriate scheme:

- always put the most common part on the left of the name
- differentiate them by type (e.g.: hosts / applications)

Of course, you're free to adopt different schemes. The most important rule is to be consistent when naming your metrics.

To achieve this, a common solution is to have a small proxy between the tool reporting metrics and Graphite. This could be an HTTP proxy (like documented by Jason Dixon, or a simple script that listens on a port, and rewrites the metric.

Using this pattern, you'll get more control over the format and paths chosen by developers or operations.

StatsD

StatsD is a network daemon listening for statistics and sends the aggregation to a backend. In our case we will see how it works with Graphite.

Setting it up and make it show pretty graphs

StatsD is a simple daemon listening for metrics. The first implementation was done by Etsy, and is written for node.js. but other implementation exists (Python, Ruby, C, etc).

To install the one by Etsy, you will need to install node.js (if it's not packaged for your OS, you can follow the instructions). Then, to actually run StatsD:

```
% git clone git://github.com/etsy/statsd.git
% cd statsd
% $EDITOR /etc/statsd/config.js
% node stats.js /etc/statsd/config.js
```

A basic configuration file will be similar to this:

```
{
  graphitePort: 2303,
  graphiteHost: "localhost",
  port: 8125,
  graphite: {
    legacyNamespace: false,
    prefixTimer: "aggregate",
    globalPrefix: ""
  }
}
```

Concepts

StatsD listens on the UDP port 8125 for incoming statistics. As for Graphite, the protocol is line based. You send a string similar to `name:1|c`. The first element (name) is the name of the statistic, the colon acts as a separator with the value (1), and the pipe separates the value with the type (c, for *counter*).

StatsD stores statistics in buckets. A value is attached to the statistic, and periodically (by default it's every 10 seconds), the statistics are aggregated and send to the backend.

A few types are supported, and we will now see them in detail.

Counter

The *counter* is the most basic type.

```
% echo "my.statistic:1|c" | nc -w 1 -u localhost 8125
```

This will add 1 to the statistic named "my.statistic". After the flush the value for this statistic will be 0. It's also possible to specify to statsd that we are sampling:

```
% echo "my.statistic:1|c|@0.1" | nc -w 1 -u localhost 8125
```

Timing

```
% echo "my.timer:43|ms" | nc -w 1 -u localhost 8125
```

This type is somewhat mis-named, since you can report more than time based metrics. You give it times in milliseconds, and it will compute the percentiles, average, standard deviation, sum, lower and upper bounds for the flush interval.

Gauges

Gauges are arbitrary values.

```
% echo "my.statistic:23|g" | nc -w 1 -u localhost 8125
```

Gauges can be useful when you have a script that runs periodically and you want to report a value (e.g: count the number of rows in a database). The number is final, there's no additional processing.

However, there's a few things to know about gauges:

- if you send multiple values for the same gauge between 2 flushes, only the most recent one will be kept

- if you're sending a gauge for the same metric from two different places, only one of them will be kept
- if there's no new value during the time period, it will send the one from the previous period to Graphite

Sets

Schema

When using graphite, you have to be sure that the smallest time retention in Graphite is the same as the interval between two flushes in StatsD. If you're sending to Graphite two data points in the same time period, it will overwrite the first one.

Management interface

A management interface is listening (by default) on the TCP port 8126. A few commands are supported:

- `stats` will output statistics about the current process
- `counters` will dump all the current counters
- `timers` will dump all the current times

The `stats` output looks like this:

```
% telnet localhost 8125
stats
uptime: 334780
messages.last_msg_seen: 0
messages.bad_lines_seen: 1590
graphite.last_flush: 9
graphite.last_exception: 334780
END
```

Where:

- `uptime` is the number of seconds elapsed since the process started
- `messages.last_msg_seen` is the number of seconds since the last message received
- `messages.bad_lines_seen` is the number of badly formatted line received
- `graphite.last_flush` is the number of seconds elapsed since the last flush to Graphite
- `graphite.last_exception` is the number of seconds elapsed since the last exception thrown by Graphite while flushing

What have we done and where to go from here

This list is a suggestion of things you can collect and measure.

Events

Every time you push an application or use your configuration manager to push changes, you could send an event to statsd. Something as simple as

```
% echo "<ENV>.APPLICATIONS.EVENTS.<APP-NAME>.deploy:1|c" | nc -w 1 -u localhost 8125
```

Now, in graphite, you can use the formula `drawAsInfinite` to represent this event as a vertical line.

Caveats

Size of the payload

When you're sending statistics to StatsD, you have to be careful about the size of the payload. If the size is greater than your network's MTU, the frame will be dropped. You can refer to this documentation to find the size that might work best for your network.

Dashboard: Info for ops and info for the business

Tasseo

Tasseo is a Graphite specific live dashboard. It is lightweight, easily configurable and provides a near-realtime view of Graphite metric data. It is a ruby based Sinatra and javascript application.

Dashing

Dashing is a dashboard framework allowing you to build your own custom dashboards. Dashboards can be created with premade widgets, or custom widgets can be written using scss, html and coffeescript. Data bindings allow reuse and manipulation of data from a variety of sources.

GDash

GDash is another dashboard for Graphite. Dashboards are created using a simple DSL.

Third-party tools

Datadog

Boundry

NewRelic

Librato Metrics

Circonus

Geckoboard

Business Continuity Planning

Backups

Backups are a critical part of preserving an organization's information. RAID, clustering and other real-time copies of data do not help when the data is deleted, the buildings containing these live sources are compromised by weather, human intervention, or facilities failures.

Could the organization survive if part or all of its information were lost?

Remember, RAID is not a backup solution. RAID protects against the loss of individual drives, but if data gets corrupted or deleted, RAID will replicate that corruption across all RAID stripes. Making backups on separate media is the best way to protect yourself against both "hard" and "soft" data loss.

Designing a backup policy

Operations needs to work with organizational leaders to develop a backup policy that protects the organization's information while balancing cost and other factors important to the organization.

Operations engineers need to be able to understand organizational needs in order to evaluate backup solutions, strategies, and current best practices. They must then translate this knowledge into recommended solutions which management can use to make a sound business decision. The ability to assemble a cohesive plan, including a cost-benefit analysis of available options, can help the organization make the right decision for the business on an appropriate backup policy.

Deciding what to backup

Consider cost, time, and staff resources when deciding what to backup. Does it make sense to backup every client hard drive when a system can be re-imaged over the network? Does the configuration management solution maintain client customizations? If so, operations may not need to backup the operating system on clients and can focus on user data. Ideally, the goal is not to restore clients starting at the operating system level.

Where is organizational data stored? Is data local on each client or does the organization use a file server? Do users actually store their data on the file server? Some users have had bad experiences using the network and move all of their data locally, or refuse to store anything on the server in case the network becomes unavailable. If users travel frequently, they may store all of their data on a laptop. This poses additional backup challenges when laptops are not always on the organization's network to talk to the backup server.

Some organizations perform incremental backups on all clients to protect against users who decide where best to store their data.

What about data stored on servers? Do all servers need to be backed up? What is the available backup window and will the organization's network push enough data to a backup server to fit within that window? How will taking the backup affect each application's performance?

Ultimately, deciding what to backup is a business decision. What information is critical to the business and how much is the organization willing to invest to protect that information? There may also be laws requiring the organization to retain certain data that will influence this decision.

When not to backup

Todo

restoring? what not to backup, regulations on same, how to store them (PCI)

Retention periods

Is there a risk to having a long retention period?

Todo

The idea of such a risk may not be immediately clear to a beginning ops person.

What is the cost to retain backups for a 30-days, 6-months, or 1 year?

Is there a business requirement for keeping backups for a specific length of time?

Are there laws for maintaining electronic records for a specific period of time?

If the organization is required to adhere to Freedom of Information Act (FOIA) law, sometimes long retention periods are a detriment. FOIA requests can cover any kind of data including individual emails on a specific subject. If the information doesn't benefit the organization, then it might be a liability due to labor-intensive searches through ancient backups.

Todo

How does FOIA affect what information an organization needs to have available? Assume the reader is a civilian and doesn't know how FOIA affects an organization.

Offsite backups

How sensitive is the organization's information? Storing a copy of the organization's crown jewels in an insecure location can expose the organization to loss, unauthorized modification, or theft of intellectual property.

Is the organization required to encrypt offsite data?

Does the organization have a second geographically distributed location that could house an off-site backup of the organization's information?

Designing a Backup System

Backup type and frequency

What kind of information does the organization rely on to do business? Do hourly changes need to be captured or can the organization survive with backups every 12-hours or once per day?

- Full backups
- Incremental backups
- Replication
- Snapshots
- Bare-metal restore vs data only
- online/offline

Todo

media – should someone address the state of backup media? Some places are still doing tape. What about orgs who rely on standalone consumer-grade disks for client backups (e.g. Time Machine)? Risks, cost to maintain.

Cost of backups

What is the cost of not doing backups?

Verification

Test backups. If data cannot be restored then what was the point of backing it up in the first place.

Recovery testing

How long does it take to restore the largest backup set?

Integrity of backups

Completeness of backups

Security implications

Todo

Using backups to restore to a known "good" state after an incident just serves to put the machine in a known vulnerable state (security hole that was exploited is now back in operation)

Todo

can be used to restore system state that can be useful in a post mortem after an incident (say the attacker covered their tracks but backups were able to capture a rootkit before it was removed or before logs were tampered with)

Recovery basics

Secure data destruction

Information Lifecycle Management in relation to backups

Main goal of backups is restore system state including data in case of issues and ILM, have data available for functional reasons other than uptime.

Main items to cover in this chapter are:

Archiving

Data replication

Outages

When outages happen (and they will) it is incredibly useful to have a point person that will handle communications while the admin focuses on resolving the issue. A common communication forum (conference call, irc, etc) that affected users can jump in get a quick status and pass along any information is useful. On a larger scale you would want to have a status page that is constantly updated.

Once an outage takes place as an Admin you have to primary functions:

1. Restore service.
2. Document what caused the failure.

Depending on the failure and the impact #2 may actually be more important than #1. Assuming you have a proper DEV / QA / PROD / DR environment setup you should fail over to your DR systems. Then take your time investigating the PROD systems to identify root cause (and build a test so you can replicate the problem at will) and come up with a proper resolution that can be documented and isn't considered a "band-aide" fix. Apply the change up through DEV

& QA and the then to your PROD, assuming all goes well fail back from DR and run your replicated failure test to show PROD isn't going to fail from that issue again. Then you are safe to make the change on your DR side.

Post-outage you need to document what the problem was, what the root cause was, and what the steps were to fix. Depending on who your users are you may wish to provide this information to them as well to keep the informed.

Postmortems

Disaster Recovery

Disaster Recovery is a process for securing business continuity in the event of a disaster. The severity of the disaster may differ, as well as how to resolve it. However, the goal for a Disaster Recovery, or the Disaster Recovery Plan should always be to get to the state of business as usual as soon as possible.

Planning

The most important thing about Disaster Recovery is preparation and planning. This is because when a disaster actually happens, you will not be able to change any prerequisites but are forced to work with what you already have. If your plan was broken or didn't actually address all the critical aspects of your IT, your Disaster Recovery-environment will end up broken as well. A misconfigured backup job can always be reconfigured during normal operations but when the environment is already down; that ship has sailed.

To make sure we cover those needs we will split this section into a couple of subsections.

Business Needs

The first question we need to ask ourselves when creating a Disaster Recovery Plan, or DRP, is what the needs of the business actually is. In most cases, an organization has some sort of critical infrastructure or systems that need to be available at all time to be able to perform the core business. This might be an Enterprise Resource Management-system (ERP), a document collaboration or versioning system, production system etc.

The size of the organization as well as the grade of information technology adaptation all need to be taken into consideration when creating a plan for Disaster Recovery.

Defining these business needs can't be the sole work of the IT Department or an Operations Engineer. It needs attention from all levels of the organizational hierarchy. One method for identifying the needs might be interviews with colleagues that work in the core business.

Questions that need to be answered by every role in the core process(es):

- In your every day work, what activities do you need to be able to perform to fulfill the goals of your role?
- How do you do these?

Using the answers to the questions, we then need to identify each system or infrastructural component that is needed to realize that goal. The identified objects gives us a list of what we actually need to secure in the case of a real disaster.

This can be done in many ways, for example:

- Backup/Restore
- Data Mirroring
- Off-Site Mirrored environment
- Failover

So, if we were to create a disaster recovery plan for Acme Inc, we would first need to identify what components we need to consider putting into the DRP. Usually, you have some kind of system (or excel spreadsheet for smaller companies) containing all the IT environment components. If so, this information will make your work a lot easier.

The components may contain, among other things:

- SQL Servers (sql-srv-001 and sql-srv-002)
- Domain Controllers (dc-srv-001 - primary, dc-srv-002 & dc-srv-003 - used for load balancing)
- Fileservers (stgsrv001)
- Application servers (app-srv-001, app-srv-002)

We will then, together with the rest of the organization, try to map these components to the bussiness activities we've identified earlier. For example:

- The application server **app-srv-001** is running our ERP, which is needed to be able to place orders from customers.
- The SQL Server **sql-srv-001** contains the data from the ERP which means that it's a prerequisite for the application server.
- The domain controller is needed so the users are able to sign in to the ERP. However, since 002 and 003's main purposes is load balancing, recovering **dc-srv-001** will be our main objective.
- The fileserver **stg-srv-001** is used to store copies of the order receipts produced in the ERP.

What we've concluded from this activity is that we need to recover four components to be able to use the ERP. Note however, that in reality, the IT environment's usually alot more complex then the one used in this example.

The identified objects then need to be ranked to determine in what order they need to be recovered to minimize downtime. In this example, this would most likely be as follows:

- dc-srv-001
- sql-srv-001
- stg-srv-001
- app-srv-001

Prioritizing Recovery Components

In the example above, the number of assets to prioritize is low, which might suggest that there is no need for making a priority list. However, one thing that you'll learn by doing a couple of disaster recovery excercises is that no matter how small the scope, stakeholders will always try to direct your effors to the assets mosts relevant to them. For example, the CEO might want you to prioritize the ERP System over the Domain Controller, which might very well be correct, but as the list of assets grow longer the number of stakeholders wanting to influence your decisions will as well.

As you might have guessed, it may be a good idea to actually prioritize your asset list, if not out of necessity then at least to circumvent any issues that might occur because of differing opinions on what needs to be prioritized. A great way to start out is to create a spreadsheet consisting of the following columns:

- **Asset Identification** A name, a FQDN or an IP. Whatever helps you identifying the asset.
- **Business Priority** Non-Essential, Essential, Critical. This should be decided by either the board or the senior management team.
- **Tiebreaker/Sequential numbering** A sequential numbering which will break ties in case multiple assets have the same priority. Should also be decided by the board or the senior management team.

- **Business Impact** A textual description of what would happen if this asset where to be unavailable.
- **Exceptions** Any exceptions to the priority above. A company that is doing billing once every month might not feel that the billing system is critical during any other period then the billing period. This will of course reflect your real time prioritization if (when) a disaster occurs.

The finished product should, after a signoff from your department manager and the senior management team, be published on your company's intranet, available for anyone. This is very important as lack of transparency is one of the most common prejudices about IT Departments.

Todo

shared resources, bussiness needs.

Disaster Recovery Plans

Todo

How to create a plan from the material we gathered in the planning phase.

Todo

Pros and cons on separating the disaster recovery manual from the technical recovery manual.

Disaster Recovery Simulations

Todo

Strategies when simulating. Defining testing scopes. Measuring.

Considerations

Todo

Limiting the scope to core business

Todo

Expanding the scope in the disaster recovery environment vs. going back to production before expanding

---

Execution

Todo

Communication

Architecture 101

How to make good architecture decisions

Patterns and anti-patterns

Introduction to availability

Introduction to scalability

Architecture 201

Service Oriented Architectures

Fault tolerance, fault protection, masking, dependability fundamentals

Fail open, fail closed

Perspective: node, network, cluster, application

Caching Concerns

Static assets

Data

Eviction and replacement policies and evaluation

Approaches

(TTL, purge-on-write, no-purge versioning, constantly churning cache versus contained, working set sizing)

Crash only

Synchronous vs. Asynchronous

Business continuity vs. Disaster Recovery

Designing for Scalability: Horizontal, Vertical

Simplicity

Performance

Tiered architectures

MTTR > MTBF

http://www.kitchensoap.com/2010/11/07/mttr-mtbf-for-most-types-of-f/

Configuration Management 101

A Brief History of Configuration Management

Configuration management as a distinct sub-discipline of operations engineering has roots in the mid-1990s. Prior to then, even sites with a large number of users like universities and large ISPs had relatively few Unix systems. Each of those systems was generally what today's operations community calls a "snowflake system" (after the phrase "a precious and unique snowflake"). They were carefully hand-built to purpose, rarely replaced, and provided a unique set of services to their users.

The rise of free Unix-like Operating Systems and commodity x86 hardware, coupled with the increasing demands to scale booming Internet services meant the old paradigms of capturing configuration were no longer adequate. Lore kept in text files, post-bootstrap shell scripts, and tales told around the proverbial campfire just didn't scale. Administrators needed automation tools which could stamp out new machines quickly, plus manage configuration drift as users made changes (deliberately or accidentally) that affected the functioning of a running system.

The first such tool to gain prominence was CFEngine, an open-source project written in C by Mark Burgess, a CS professor at Oslo University. CFEngine popularized the idea of *idempotence* in systems administration tasks, encouraging users to describe their system administration tasks in ways that would be convergent over time rather than strictly imperative shell or perl scripting.

Todo

a specific example of convergent over time might help

In the early 2000s, the systems administration community began to focus more intensely on configuration management as distributed systems became both more complex and more common. A series of LISA papers and an explosion in the number and sophistication of open-source tools emerged. Some highlights and background reading:

- Steve Traugott's isconf3 system and paper "Bootstrapping an Infrastructure" provided a concrete model for repeatable, scalable provisioning and config management.
- CERN released and wrote about Quattor which they used to build and administer high-performance compute clusters at larger scale than most sites at the time had dealt with.

- Alva Couch from Tufts University and Paul Anderson from University of Edinburgh, laid out theoretical underpinnings for configuration management in a joint session at LISA'04
- Narayan Desai's bcfg2 system provided a hackable Python CM project with early support for advanced features like templating and encrypted data
- Recapitulating Luke Kanies' departure from cfengine to start Puppet, Adam Jacob created Chef in 2008 to address fundamental differences with Puppet (primarily execution of ordering and writing user code in Ruby vs a DSL).

By 2008, provisioning and configuration management of individual systems were well-understood (if not completely "solved") problems, and the community's attention had shifted to the next level of complexity: cross-node interactions and orchestration, application deployment, and managing ephemeral cloud computing instances rather than (or alongside) long-lived physical hardware.

A new crop of CM tools and approaches "born in the cloud" began to emerge in the 2010s to address this shift. SaltStack, Ansible, and Chef-v11 built on advances in language (Erlang and Clojure vs Ruby and Python), methodology (continuous deployment and orchestration vs static policy enforcement), and the component stack (ZeroMQ and MongoDB vs MySQL).

Whatever specific configuration management tooling operations engineers encounter as an operations engineer, ultimately the technology exists to enable business goals – short time-to-restore in the face of component failure, auditable assurance of control, low ratio of operators per managed system, etc. – in a world whose IT systems are moving, in the words of CERN's Tim Bell, "from pets to cattle".

Idempotence

Convergent and Congruent systems

Direct and Indirect systems: ansible, capistrano

(mpdehaan: Are we talking about deployment here? Then let's start a deployment section. What does direct/indirect mean? How about not addressing tools in 101 and talking about concepts, so as to make a better tools section? Ansible operates in both push and pull topologies, so I'm guessing that is not what is meant about direct/indirect?)

Chef

Chef (adam: I'm biased here, but I would do Chef in 101, puppet and cfengine in 201, but it's because I want junior admins to get better at scripting, not just because I'm a dick.) (Magnus: this goes back to why Ruby will be so much more for new guys coming in today like Perl was for a lot of us in the 90's)

Configuration Management 201

Ansible

Ansible is a configuration management, deployment, and remote execution tool that uses SSH to address remote machines (though it offers other connection types, including 0mq). It requires no server software nor any remote programs, and works by shipping small modules to remote machines that provide idempotent resource management. While implemented in Python, Ansible uses a basic YAML data language to describe how to orchestrate operations on remote systems.

Ansible can be extended by writing modules in any language you want, though there is some accelerated module writing ability that makes it easier to do them in Python.

To prevent documentation drift, see Ansible documentation site.

Puppet

As system administrators acquire more and more systems to manage, automation of mundane tasks is increasingly important. Rather than develop in-house scripts, it is desirable to share a system that everyone can use, and invest in tools that can be used regardless of one's employer. Certainly doing things manually doesn't scale.

This is where Puppet comes to rescue. Puppet is a Configuration Management Tool. It is a framework for Systems Automation. An OpenSource software written in Ruby that uses Declarative Domain Specific Language (DSL).

Puppet usually uses an agent/master (client/server) architecture for configuring systems, using the Puppet agent and Puppet master applications. It can also run in a self-contained architecture, where each managed server has its own complete copy of your configuration information and compiles its own catalog with the Puppet apply application.

Puppet Agent/Master Communication

Before being applied, manifests get compiled into a document called a "catalog", which only contains resources and hints about the order to sync them in. With puppet apply, the distinction doesn't mean much.

In a master/agent Puppet environment, though, it matters more, because agents only see the catalog.

Running Puppet in agent/master mode works much the same way, the main difference is that it moves the manifests and compilation to the puppet master server. Agents don't have to see any manifest files at all, and have no access to configuration information that isn't in their own catalog.

How Do Agents Get Configurations ?

Puppet's agent/master mode is pull-based. Usually, agents are configured to periodically fetch a catalog and apply it, and the master controls what goes into that catalog.

By using this logic, manifests can be flexible and describe many systems at once. A catalog describes desired states for one system. By default, agent nodes can only retrieve their own catalog and they can't see information meant for any other node. This separation improves security.

This way, one can have many machines being configured by Puppet, while only maintaining our manifests on one (or a few) servers.

Installation

Puppet has an installing puppet page on how to get it installed and ready to use, please refer to it after deciding what OS and deployment type to use.

The examples shown here use CentOS installation with standalone architecture and Puppet apply application.

Manifests

Puppet programs are called "manifests", and they use the .pp file extension. These programs comprise of resource declarations, described below.

Resources

System's configuration can be imagined as a collection of many independent atomic units, called "resources".

Example of puppet resouces can be a specific file, a directory, a service.

Anatomy of a Resource

In Puppet, every resource is an instance of a resource type and is identified by a title, it has a number of attributes (which are defined by the type), and each attribute has a value.

Puppet uses its own language to dsecribe and manage resources:

```
type { 'title':
   argument  => value,
   other_arg => value,
}
```

This syntax is called a resource declaration.

Example of file resource type.

```
file{ '/tmp/example':
  ensure   => present,
  mode     => '600',
  owner    => 'root',
  group    => 'hosts',
}
```

Resource Type

As mentioned above, every resource has a type.

Puppet has many built-in resource types, and you can install even more as plugins. Each type can behave a bit differently and has a different set of attributes available.

The full list of different puppet resource types can be found at the Puppet Type Reference.

Puppet Apply

It is used below to test small manifests, but it can be used for larger jobs too. In fact, it can do nearly everything an agent/master Puppet environment can do.

'apply' is a Puppet subcommand. It takes the name of a manifest file as its argument, and enforces the desired state described in the manifest.

Try applying the short manifest above:

```
# puppet apply /root/examples/file-1.pp
notice: /Stage[main]//File[testfile]/ensure: created
notice: Finished catalog run in 0.05 seconds
```

Package/File/Service

The package/file/service pattern is one of the most useful idioms in Puppet. This is a pattern constantly seen in the production Puppet code.

Below is an example of a manifest that uses this pattern to install and configure ssh for Enterprise Linux - based Linux systems.

```
package { 'openssh-server':
  ensure => present,
  before => File['/etc/ssh/sshd_config'],
```

```
}
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode => 600,
  source => '/root/examples/sshd_config',
}
service { 'sshd':
  ensure => running,
  enable => true,
  subscribe => File['/etc/ssh/sshd_config'],
}
```

The package resource makes sure the software and its config file are installed, the config file depends on the package package resource, and the service subscribes to the changes in the config file.

CFEngine 3

"In 2008, after more than five years of research, CFEngine 3 was introduced, which incorporated promise theory as 'a way to make CFEngine both simpler and more powerful at the same time', according to Burgess." https://en.wikipedia.org/wiki/CFEngine

"If you are looking for a fast and highly scalable configuration management tool for your IT infrastructure, you should give CFEngine a try. Though the functionality it offers is quite similar to that offered by other popular tools such as Puppet and Chef, CFEngine has a much smaller footprint, both in terms of memory and CPU utilization, and is generally faster because it is written in C and thus runs natively on the OS.", CFEngine tutorial on DigitalOcean.com

To learn more see: CFEngine Primer, CFEngine Tutorial, CFEngine Learning Center,

SaltStack

SaltStack or just **Salt**, is a configuration management and remote execution tool written in Python. Salt uses ZeroMQ to manage communication between master and minions, and RSA keys to handle authentication. This chapter will explain the basics on how to get started with it.

Salt is a centralized system, which means there is a main server (also referred here as *master*) which manages other machines connected to it or itself (also referred here as *minions*). This topology can be further split using Salt Syndic, please refer to Salt documentation for more details on this topic.

In examples below we will be using the master + 1 minion setup. The approximate time you will need to work through all the content is about 10 minutes.

Prerequisites:

- access to 2 Linux/Solaris/FreeBSD/Windows machines in the same network
- basic understanding of command line instructions
- basic understanding of YAML file format

Installation

Salt has a dedicated page on how to get it installed and ready to use, please refer to it after deciding what OS you will be using. These examples are shown on an Ubuntu installation with Salt installed from a project personal package archive.

To set-up the environment you can use virtual machines or real boxes, in the examples we will be using hostnames **master** and **slave** to refer to each one.

At this point, you should install the latest version on both machines with the directions provided above, and have a command line session open on both your **master** and **slave** machines. You can check what version are you using on master with:

```
root@master:~# salt --version
salt 0.10.3
```

and on slave with:

```
root@slave:~# salt-minion --version
salt-minion 0.10.3
```

Configuration

A minimum configuration is required to get the slave server to communicate with master. You will need to tell it what IP address and port master uses. The configuration file can typically be found at `/etc/salt/minion`.

You will need to edit the configuration file directive `master:  salt` replacing `salt` with master IP address or its hostname/FQDN.

Once done, you will need to restart the service: **salt-minion**. On most Linux distributions you can execute `service salt-minion restart` to restart the service.

Authentication keys for master/slave are generated during installation so you don't need to manage those manually, except in case when you want to preseed minions.

To add the slave to minions list, you will have to use the command `salt-key` on master. Execute `salt-key -L` to list available minions:

```
root@master:~# salt-key -L
Unaccepted Keys:
slave
Accepted Keys:
Rejected:
```

To accept a minion, execute `salt-key -a <minion-name>`:

```
root@master:~# salt-key -a slave
Key for slave accepted.
```

```
root@master:~# salt-key -L
Unaccepted Keys:
Accepted Keys:
slave
Rejected:
```

Once the minion is added, you can start managing it by using command `salt`. For example, to check the communication with slave, you can ping the slave from the master:

```
root@master:~# salt 'slave*' test.ping
slave: True
```

Remote execution

In order to understand how Salt does its configuration management on minions, we'll take look at the `salt` command line tool. Let's take our previous command and inspect the parts of the command:

```
root@master:~# salt 'slave*' test.ping
                    ^ ^
                    |_|___________________
                    target  function to execute
```

target is the minion(s) name. It can represent the exact name or only a part of it followed by a wildcard. For more details on how to match minions please take a look at Salt Globbing.

In order to run target matching by OS, architecture or other identifiers take a look at Salt Grains.

Functions that can be executed are called Salt Modules. These modules are Python or Cython code written to abstract access to CLI or other minion resources. For the full list of modules please take a look this page.

One of the modules provided by Salt, is the **cmd** module. It has the **run** method, which accepts a string as an argument. The string is the exact command line which will be executed on the minions and contains both the command name and command's arguments. The result of the command execution will be listed on master with the minion name as prefix.

For example, to run command `uname -a` on our slave we will execute:

```
root@master:~# salt slave cmd.run 'uname -a'
slave: Linux slave 2.6.24-27-openvz #1 SMP Fri Mar 12 04:18:54 UTC 2010 i686 GNU/Linux
```

Writing configuration files

One of the Salt modules is called `state`. Its purpose is to manage minions state.

> Salt configuration management is fully managed by states, which purpose is to describe a machine behaviour: from what services are running to what software is installed and how it is configured. Salt configuration management files (`.sls` extension) contain collections of such states written in YAML format.

Salt states make use of modules and represent different module calls organised to achieve a specific purpose/result.

Below you can find an example of such a **SLS** file, whose purpose is to get Apache Web server installed and running:

```
install_apache:
  pkg:
    - installed
    - name: apache2
  service:
    - running
    - name: apache2
    - watch:
      - pkg: apache2
```

To understand the snippet above, you will need to refer to documentation on states: pkg and service. Basically our state calls methods `pkg.installed` and `service.running` with argument `apache2`. The `watch` directive is available for all states and creates a dependency, as well as restarts the apache2 service if the apache package changes or gets updated.

Back to `state` module, it has a couple of methods to manage these states. In a nutshell the state file form above can be executed using `state.sls` function. Before we do that, let's take a look where state files reside on the master server.

Salt master server configuration file has a directive named `file_roots`, it accepts an YAML hash/dictionary as a value, where keys will represent the environment (the default value is `base`) and values represent a set/array of paths on the file system (the default value is `/srv/salt`).

Now, lets save our state file and try to deploy it.

Ideally you would split state files in directories (so that if there are also other files, say certificates or assets, we keep those organised). The directory layout we will use in our example will look like this:

```
/srv/salt/
|-- apache
|   `-- init.sls
`-- top.sls
```

When creating new states, there is a file naming convention. Look at `init.sls`, it is the default filename to be searched when loading a state. This is similar to Python or default web page name `index.html`.

So when you create a new directory for a state with an `init.sls` file in it it translates as the Salt state name and you can refer to it as that. For example, you do not write `pkg: new_state.init`, write just `pkg: new_state`.

Now to deploy it, we will use the function `state.sls` and indicate the state name:

```
root@master:~# salt slave state.sls apache
slave:
----------
    State: - pkg
    Name:      apache2
    Function:  installed
        Result:    True
        Comment:   Package apache2 installed
        Changes:   apache2.2-bin: {'new': '2.2.14-5ubuntu8.10', 'old': ''}
                   libapr1: {'new': '1.3.8-1ubuntu0.3', 'old': ''}
                   perl-modules: {'new': '5.10.1-8ubuntu2.1', 'old': ''}
                   ssl-cert: {'new': '1.0.23ubuntu2', 'old': ''}
                   apache2-utils: {'new': '2.2.14-5ubuntu8.10', 'old': ''}
                   libaprutil1-ldap: {'new': '1.3.9+dfsg-3ubuntu0.10.04.1', 'old': ''}
                   apache2-mpm-worker: {'new': '2.2.14-5ubuntu8.10', 'old': ''}
                   make: {'new': '3.81-7ubuntu1', 'old': ''}
                   libaprutil1: {'new': '1.3.9+dfsg-3ubuntu0.10.04.1', 'old': ''}
                   apache2: {'new': '2.2.14-5ubuntu8.10', 'old': ''}
                   libcap2: {'new': '1:2.17-2ubuntu1', 'old': ''}
                   libaprutil1-dbd-sqlite3: {'new': '1.3.9+dfsg-3ubuntu0.10.04.1', 'old': ''}
                   libgdbm3: {'new': '1.8.3-9', 'old': ''}
                   perl: {'new': '5.10.1-8ubuntu2.1', 'old': ''}
                   apache2.2-common: {'new': '2.2.14-5ubuntu8.10', 'old': ''}
                   libexpat1: {'new': '2.0.1-7ubuntu1.1', 'old': ''}

----------
    State: - service
    Name:      apache2
    Function:  running
        Result:    True
        Comment:   The service apache2 is already running
        Changes:
```

You can see from the above that Salt deployed our state to **slave** and reported changes.

In our state file we indicated that our service requires that the package must be installed. Following the same approach, we can add other requirements like files, other packages or services.

Let's add a new virtual host to our server now using the `file` state. We can do this by creating a separate state file or re-using the existing one. Since creating a new file will keep code better organised, we will take that approach.

We will create a new `sls` file with a relevant name, say `www_opsschool_org.sls` with the content below:

```
include:
  - apache
```

```
extend:
  apache2:
    service:
      - watch:
        - file: www_opsschool_org

www_opsschool_org:
  file.managed:
  - name: /etc/apache2/sites-enabled/www.opsschool.org
  - source: salt://vhosts/conf/www.opsschool.org
```

Above, we include the existing Apache service state and extend it to include our configuration file. The `watch` requisite indicates that the Apache service is dependent on the `www_opsschool_org` file state and will restart the Apache service if the file changes.

Below is the directory listing of the changes we did:

```
/srv/salt/
|-- apache
|   `-- init.sls
|-- top.sls
`-- vhosts
    |-- conf
    |   `-- www.opsschool.org
    `-- www_opsschool_org.sls
```

Using the newly created state file, we can try and deploy our brand new virtual host:

```
root@master:~# salt slave state.sls vhosts.www_opsschool_org
slave:
----------
    State: - file
    Name:      /etc/apache2/sites-enabled/www.opsschool.org
    Function:  managed
        Result:    True
        Comment:   File /etc/apache2/sites-enabled/www.opsschool.org updated
        Changes:   diff: New file

----------
    State: - pkg
    Name:      apache2
    Function:  installed
        Result:    True
        Comment:   Package apache2 is already installed
        Changes:
----------
    State: - service
    Name:      apache2
    Function:  running
        Result:    True
        Comment:   Started Service apache2
        Changes:   apache2: True
```

Salt reports another successful deploy and lists the changes as in the example above.

All this time, you were probably wondering why there is a file `top.sls` and it was never used?! Salt master will search for this file as indicated in the configuration of your install. This file is used to describe the state of all the servers that are being managed and is deployed across all the machines using the function `state.highstate`.

Let's add our state files to it to describe the high state of the `slave`.

```
base:
  'slave*':
    - vhosts.www_opsschool_org
```

Where `base` is the default environment containing minion matchers followed by a list of states to be deployed on the matched host.

Now you can execute:

```
root@master:~# salt slave state.highstate
```

Salt should output the same results, as nothing changed since the last run. In order to add more services to your slave, feel free to create new states or extend the existing one. A good collection of states that can be used as examples can be found on Github:

- https://github.com/saltstack/salt-states – Community contributed states
- https://github.com/AppThemes/salt-config-example – WordPress stack with deployments using Git

See also:

For the full documentation on available states, please see Salt States documentation.

Capacity Planning

(Seems reasonable to have the Statistics for Engineers course be a pre-req for this course)

Fundamentals of capacity planning

Resource usage investigation and exploration

- Examples: CPU:req/sec ratio, memory footprint:req/sec ratio, disk consumption per user/per sale/per widget, etc.
- Application:Infrastructure metric relationships
- 2nd order capacity (logging, metrics+monitoring systems, ancillary systems)

Finding ceilings

- Discovering resource limits
- Comparing different hardware/instance profiles - production load versus synthetic
- Benchmarking: pitfalls, limitations, pros/cons
- http://www.contextneeded.com/system-benchmarks
- Multivariate infra limits (multiple resource peak-driven usage) Ex: web+image uploads, caching storage+processing, etc.
- Architecture analysis (anticipating the next bottleneck)

Forecasting

Linear and nonlinear trending and forecasting (“steering by your wake”)

Details of automatic forecasting and scaling

Seasonality and future events

- Organic growth approaches (bottom-up infra driven, top-down app driven)
- inorganic growth events (new feature launch, holiday effects, “going viral”, major public announcement)
- Provisioning effects on timelines, financial tradeoffs

Diagonal scaling

(vertically scaling your already horizontal architecture)

Reprovisioning and legacy system usage tradeoffs

Statistics For Engineers

Normal distributions

Percentiles, histograms, averages, mean, medians

Software Deployment 101

Software deployment vs configuration management

definition of a host’s role or state

Running services

inetd

Shared containers vs self-contained binaries

Package management

Configuration files

Source based / binary packages

Building packages

Packaging new tools

Software Deployment 201

Running services

daemontools

god

angel

runit

monit

supervisord

circus

Soft Skills 101

The term "soft skills" seems to imply that these skills are somehow less important than technical skills. In reality, soft skills are often specifically sought-after by hiring managers. These skills are also important for operations people seeking to advance to senior engineering positions.

As much as technical people would like to believe that operations is a purely technical profession, it is really about serving people. Operations, as the title indicates, is about making things work for people. Operations people design, build, and maintain services for people to use. It is all in a day's work for an operations professional to translate, educate, inform, reason, persuade, and generally act as a liaison between technology and the people who use it.

Soft skills at the 101 level encompass communication skills, time management, project management, and a basic understanding of DevOps from an operations perspective. Soft skills at the 201 level lead into general business skills including positioning, budgeting and the financial process, using metrics effectively, demonstrating impact, risk management, managing customer preference, and thinking strategically.

Communication basics

Audience analysis is the first step to effective communication. Perform a basic audience analysis by answering some simple questions:

- Is the audience technical or non-technical?
- How much do they know about the topic?
- How much do they care about the topic?
- What is the intended message for this audience?

Before sending one email or setting up a meeting, answer these questions. People are inundated with communication from email, voicemail, Twitter, social media, internal web/wikis, text, IM, and meeting requests.

Communicating internally

Internal customers could be people who use general computing and call a helpdesk for support, the organization's software developers or engineering team, senior management, or researchers, students, faculty, or others. The type of customers depends upon the type of organization and the industry.

Working with internal customers could be as simple as being the "Ops" side of a DevOps team or it could mean supporting a wide range of technologies used by customers at varying levels of technical understanding.

When operations focuses on a specific project or works with a specific team, such as engineering or software development, communication is generally specific to that work. It can take on the form of meetings, video conferencing, chat sessions, and emails between team members. A communications culture tends to develop in these scenarios as team members figure out the best way to coordinate with one another.

When operations focuses on more general IT support, communication becomes more complicated for the operations people. Factors such as audience analysis play a larger role in successful communication with customers. Operations faces a potentially wide array of communications scenarios:

- Announcing outages to general staff in a large organization
- Announcing upcoming maintenance to a set of staff impacted by a service outage
- Broadcasting a technical idea to a non-technical audience
- Contacting internal customers impacted by a security issue or vulnerability (e.g. Run this update. Install this patch.)
- Asking middle management across the organization to weigh in on a potential service change
- Offering a seminar, workshop, or class to assist customers with a new or modified service for a general audience
- Offering a seminar, workshop, or class to assist customers with a new or modified service for a non-technical audience
- Presenting the service catalog in a question-and-answer session
- Meeting with senior management to address an operations problem, budget shortfall, request more resources, or propose an architectural change
- Meeting with customers to address service problems
- Meeting with specific groups of customers to collect requirements for a special project
- Requesting feedback from customers either individually or as a group
- Meeting with customers who are engaged in the subject matter
- Meeting with customers who are disengaged or in attendance because it is mandatory

This list spans a wide range of communication modes, communication types, customers, and outcomes.

- **communication modes** email, meetings, larger presentations, surveys
- **communication types** persuasive communication, instructional, informational
- **diverse customer backgrounds** management, administrative staff, technical staff, IT-savvy, non-IT-savvy, interested, disinterested
- **desired outcomes** management decision, increased understanding, increased abilities, increased awareness

Communicating externally

Communicating with external customers can offer additional challenges. If the external customers are customers of the organization, there is the possibility that dealings with them could result in a complaint to upper management.

Reduce complaints by considering how to communicate with these external customers. When communicating about a service outage, consider timing of the outage, duration, and impact of the outage on these external customers. Are most external customers in the same time zone? If so, then the maintenance window could be outside of traditional working hours. If external customers include international people in varying timezones, the outage window may be the one that impacts core customers the least.

Communicate the timing of service outages with management. It is best if management knows that external customers are about to be impacted by operations. Include a justification for the maintenance: why is it necessary, why this outage window, why this duration, plan B in case the outage goes beyond the outage window, method of communication with external customers? All of these pieces of information may not be necessary if operations already supports external customers on a regular basis.

There is significant breadth and depth required to effectively communicate.

Communication Modes

Let's start by covering the two most common modes of communication for operations: email and meetings.

Communicating via email

Before sending that email to the entire organization, who really needs to know the information? People already get a lot of email; for most it is information overload. How many of customers already complain about too many emails? Don't get filtered, Make communication count.

Here are some best practices when using email to communicate:

- Shorter is better.
- Make the subject descriptive (e.g. "www.co.com outage, May 10 - 6-8 pm")
- Put the most important information at the top of the message (e.g. deadline, action, outage dates). People generally skim the first few lines to determine if the information pertains to them. Starting with a lengthy background risks alienating people before they read the important part of the message.
- State the audience at the top of the email (e.g. "All Macintosh Users") to let them know the message is directed at them.
- Consider including a link to an internal site with a lengthier writeup if needed.
- Limit the recipient list to only those people who need or want the information (management, administrative customers, developers, people impacted.). Build a list if necessary to avoid spamming the entire organization.

Sometimes email is the best way to communicate and sometimes not. Decide when to use email and when to communicate another way.

Consider email appropriate in some situations:

- Attemping to reach a large audience
- The message or action is simple
- The message needs to reach them now
- Need to document the distribution of the information.
- Following up on a previous conversation, request, or action

Consider email less effective in other situations:

- Conversing back-and-forth with people to define or understand a complex issue
- Creating something new

- Drawing it on a whiteboard would provide better enlightenment
- There is a potential for much confusion or questions about the issue
- Asking for a management decision on a technical issue from non-technical management
- Trying to teach people

Sometimes email can be used in combination with other methods:

- After a meeting, send an email to the attendees to summarize action items or decisions. This can be an important tool to remind management of a decision made months earlier.
- Announce the time and location of a seminar or training class.
- Share status of an action taken as the result of a discussion or meeting.

Some common effective uses of email include the following:

- Notification of outages
- Warn of IT security threats (e.g. raise awareness of increased phishing attacks)
- Document a decision made by management in a meeting.
- Document the outcome of actions taken
- Provide status on previous assignments
- Announce training, seminars, and presentations by operations
- Provide customers with a link to access a new or modified service

The dreaded meeting

If customers think they get too much email, some of them also think they also attend too many meetings. Some people, especially managers, have corporate calendars that resemble a tetris game. Coordinating an effective and productive meeting follows a simple formula.

Have a purpose. Need a decision? Need a decision now? Need to inform? Need need to persuade?

Be prepared! Consider audience and be prepared to answer questions relevant to their interest in the topic. Some of this is covered in more depth at the Soft Skills 201 level.

Communicate at the right level. Leave out technical jargon if meeting with a non-technical audience. Consider simplified explanations, diagrams, and framing the content to address concerns that the audience cares about. Operations is the translator of technical information when meeting with a non-technical audience. Take that role seriously.

Set a duration. Decide how much time is needed to present the topic and answer questions. Make it as short as possible. Some organizations default all meetings to one hour.

Consider what the audience gets out of the meeting. Should the audience increase their knowledge or understanding on the topic? Maybe they have no interest in the topic but are the final decision maker due to funding levels, type of money, policy, role within the organization, or other factors.

Stick to the agenda. Do not let the audience take the meeting off course. In a 1:1 meeting, the audience might ask for IT support for an unrelated problem. Agree to put someone on the problem after the meeting, then go return to the scheduled topic. In a larger meeting, audiences can tangent into related areas or even unrelated areas. Be prepared to steer the meeting back on topic.

Summarize Summarize the outcome in the last few minutes of the meeting. It can be good to send an email to summarize decisions made in the meeting in order to document the outcome.

Required meetings

Sometimes attendees are mandated to attend meetings:

- Committees where members are selected by the organization to represent a subset of people. Committees are often too large and unproductive. The saying "languishing in committee" describes this cultural phenomenon.
- Management meetings where all members of a management team are required to meet at regular intervals to review topics that may or may not be relevant to everyone in the room.
- Training where all employees of an organization are required to complete a minimum set of hours on a particular topic.

The operations person tasked with leading one of these types of meetings may find a less than enthusiastic audience. Apply the best practices above and attempt to make these meetings productive. Even without being the chairperson, sometimes keeping a meeting on topic and looking for areas to be productive can reduce inefficiencies.

Alternative meeting styles

Meetings do not always require scheduling a conference room for an hour or more and everyone arriving with a laptop or a legal pad. Consider stand up meetings or even short 10-minute slots on a manager's calendar to provide a quick status update or respond to a question that is best answered in person.

Special cases for operations

There are some special communication challenges that operations engineers face.

Communicating planned and unplanned outages

Managing maintenance windows in the organization involves more than choosing a date and time that works for operations.

Consider working around important events within the organization. It takes extra planning and outreach to learn about these events, but it is one way operations demonstrates that it is savvy to the organization's needs. Wouldn't it be good to know if the organization is about to roll out the next version of a product, perform a year end close-out, host a big conference, or stage a demo to an important external stakeholder. There are no extra points for doing this, but operations avoids losing respect within the organization for being unaware of the organization's core business.

For outages that may impact a large percentage of the customers or a critical service, it is a good practice to notify the organization more than a week in advance. This serves a dual purpose: it alerts people who might be out of the office the week before the actual outage and it provides lead time to reschedule in case someone responds with a critical activity that would conflict with the outage. Send a reminder the day before or the day of the outage for customers who missed the first message.

To send a followup email, simply forward the original email with a short note at the top reminding people of the time and services impacted.

Example: Planned outage notification

```
All file cluster users,

Save your work before 7:00 pm Friday, January 10th for a planned
outage of the file cluster.

The file cluster will be taken off-line for scheduled maintenance
at 7:00pm Friday, January 10th. We expect the outage to last until
```

```
10:00 pm.

Notify operations immediately if this interferes with time-critical work.

[provide a way to notify operations]
```

Fielding customer complaints

In the world of operations, customer complaints are a given. Operations can't please everyone all the time. Every operations person has dealt with unhappy customers so it is good to develop strong people skills.

It is important to face customer complaints, not avoid them. Occasionally we have a customer who is a chronic complainer and the operations staff dive under their desks when that person walks in the office. A complaint should be treated as an opportunity to hear a customer's perception of services. Complaints can be turned into opportunities for improvement and can be a path to creating a lasting relationship with customers.

People are often at their worst when reporting a complaint; emotions are high due to lost data, a service outage, or frustration trying to use technology. Now is not the time for operations to get emotional or defensive about the work. Instead of reacting, follow these steps to adeptly manage customer unhappiness and maybe increase customer respect for operations as a whole.

- Listen without judgment
- Rephrase the concern so to confirm understanding
- Agree to investigate if it isn't something resolvable now
- Leave the customer with the assurance that someone will get back to him/her with a solution or feedback.
- Get back to the customer even if it is to say
  - It was a one-off problem and here is why
  - We found a problem internally and it is now resolved
  - We are improving our processes to reduce the likelihood of it happening again
  - Or an explanation that simply provides feedback to the customer.
- And don't forget to thank the customer for taking the time to provide feedback

The reason to close the feedback loop is to show the customer that operations did something as a result of the complaint. The customer will know that someone in operations was concerned enough to investigate and potentially resolve the root cause of the complaint. It could have been inconsistencies in operation's internal procedures or a skills gap. That's a bonus for operations and the customer should know that the communication had a positive impact.

Try these techniques with chronic complainers. Sometimes all they want is to be heard. Bring in IT operations management if someone is repeatedly impacting operations with complaints or becomes abusive, This advice stands if operations feels like the above techniques are not working. Escalation to the next person in the management chain is a valid procedural step in any of these instances.

Todo

It might be interesting to put together an exercise where the student interacts with a fictional customer in some different scenarios. Depending on what the student does, the customer is happy or complains to the operations person or escalates the complaint up the management chain. How does the student respond? Could have multiple scenarios with different customers (a customer who causes his own problem then gets in the way, a customer who cannot wait, a customer who tries to fix the problem and makes it worse, a customer who uses the opportunity to speak to an operations person to dump 10 other requests on that person. This idea came to me from a series of books my kid has where you make a decision on page 10 that leads to to either page 26 or page 40. Your decision could end the story

or take you in a new direction. The books are full of these decision points so the story is rarely the same twice, kinda like customer support!

Time Management

Time management is a critical skill for the operations professional. Customer service requests and trouble tickets are up against project work and infrastructure maintenance and enhancements. How does one person prioritize and accomplished?

Recommended reading:

- Tom Limoncelli's book Time Management for System Administrators
- Tom Limoncelli's Time Management Wiki

Tom Limoncelli also teaches a Time Management tutorial at the USENIX LISA conference and sometimes the LOPSA community conferences: Lopsa-East and Cascadia

Todo

does this section need a real writeup or are references to Tom's work enough?

Project Management

Project management is a necessary skill for any mid-level operations person. Start with small projects and work the way up to larger ones.

Be aware that project customers, or stakeholders, will often not know what they truly want from a project or they ask for the moon. Review the project management triangle (good, cheap, fast: pick two).

Henry Ford is credited with saying about his customers "If I had asked customers what they wanted, they would have said faster horses." Whether or not he said it, it still captures the essence of requirements gathering for operations projects. The operations professional is the technology expert. The stakeholders know they want a certain output or service. They may not know what that looks like or how to achieve it. The challenge is to extract requirements from the stakeholders then realize that these may not be the real or complete requirements.

Enter project management. Project management should help to frame the scope, resources, goals, and outcomes for the project. Let's look at two different project management methodologies as they apply to operations.

Waterfall

Waterfall is a hierarchical form of project management that was adapted from other industries for the software development world. In waterfall, think of the phases of a project as a cascading waterfall. Each phase must be completed before moving onto the next phase. The entirety of the project is scoped from beginning to end including milestones and and final deliverables.

Technologies change, requirements change and scoping a large project over a long period of time with what are commonly incomplete requirements or faulty assumptions by stakeholders leads operations down a path of delivering an incomplete or inaccurate solution at the end. Waterfall breaks down in practice because it requires a promise of delivery that may be several years out.

Also, by requiring each phase a project to complete before moving onto the next phase, bugs and issues are often not discovered until late in the project. This causes delays and sometimes large amounts of refactoring or re-architecting to go back and resolve these issues.

Detractors of the waterfall method point to its rigidity and lack of testing during the development phase. One of the issues in operations and development work is that stakeholders may not have a solid grasp of requirements until they see a working prototype, or iterations of working prototypes during the implementation of the product. It is common for stakeholders in a project not to know what technology can deliver until they see it. Many operations teams are moving to Agile methods for several reasons and one of them is because agile development allows stakeholders to see working bits of the product before the end and to modify requirements before it's too late.

Agile

Agile is a project management methodology. Agile started in 2001 when a group of software developers created the Agile Manifesto. The Agile Manifesto outlines the 12 principles of agile. Agile is seen most often in the software development world but it has crept into operations because of the obvious benefits over waterfall. Common implementations of Agile include: Scrum, Kanban, and the hybrid Scrumban that was created to meet more operational needs. The idea behind Agile is continuous release or delivery of a product. Instead of creating one big outcome at the end of a project, Agile allows a team to release a partially completed project for stakeholder review and requirements tweaking. Another big benefit of Agile methodologies is the discovery of problems early in the product development cycle when refactoring can be done immediately before the end product is set in a particular architectural direction that would make it costly to change.

Some documented benefits of agile include the following:

- Reduced process overhead
- Improved team and stakeholder communication and collaboration
- Errors and bugs are fixed in development instead of waiting till the product is "complete" to address them.
- Stakeholders see the product as it is shaped and have the ability to adjust requirements during development
- Project teams are empowered
- Can easily be combined with DevOps methodology to improve effectiveness of development-into-operations
- If done well, can increase work output of teams (increased velocity)
- Everyone on the project can easily see where the project stands (e.g. Scrum board or Kanban wall)

One thing to remember when implementing an Agile solution: adapt it as needed. Each of the following has its own simple framework, but organizations can use some or all of the implementation and even combine Agile methods to achieve success.

Scrum

Scrum is the more prescriptive of the included methods. Scrum is recognizable by Scrum boards, user stories, timeboxed sprints, cross-functional teams, Scrum Master and Product Manager roles, the burndown chart used for tracking project status, and the Scrum meetings: daily stand-up, and retrospectives.

Some of the limiting factors of Scrum for operational teams include timeboxing and tracking the burndown velocity of the team.

Scrum board - An electronic or physical board that is used to track project status, actions that are in progress, upcoming work, and completed work. A basic Scrum board will have three columns: Todo, In Progress. Done. Items in todo are the up and coming work, items in "In Progress" are currently being worked during this sprint. Done is fairly self-explanatory. Assignments can be tracked by sticky note on a white board or via an electronic Scrum board. The Scrum board also has rows. These are referred to as swimlanes. Rows can be labeled with project names and it common to have the very first swimlane titled "unplanned work" for operations tasks that fall on the team.

Electronic Scrum board - Electronic Scrum board software can be great if the team is geographically distributed. All members of the team can see and update the board from remote locations. The downside of electronic versions is

getting the team to keep the application open and updated. Burndown can also be computed automatically making it easier for management to see progress.

Physical Scrum board - Often a whiteboard with a grid made of electrical tape. The swimlanes and tasks are marked by sticky notes. The team names can be post-it flags or some other marker. The downsides to a physical board include manual tracking of burndown, stickies falling off the board onto the floor (hint: Buy the Post-It super sticky notes or use tape or magnets), and lastly distributed teams cannot see the board easily. The upside to a physical board is visibility. The board can be placed in a prominent location where the operations staff can see it every day. This makes for easy daily stand-ups. It also allows members of the team to walk up to the board and have conversations with other members of the team about the work in progress.

Sprint - A sprint is a duration of time defined by the team when the work will be done between Scrum meetings. Work is chunked into pieces small enough to fit within the sprint window. A sprint window might be a week, two weeks, four weeks, or whatever length of time seems to fit the team. During the sprint, operations staff focus on the work agreed upon at the beginning of the sprint. Organizations can define how unplanned work will be dealt with during a sprint. Sometimes it is helpful to be able to tell a customer that we can prioritize that project request in two weeks at our next sprint meeting instead of feeling like operations has to drop everything for a last minute request. Sprints are somewhat rigid and can break down with operations because the work doesn't neatly fit within a timeboxed window. The team will also provide time estimates for each task.

Daily Standup - This is a short daily meeting with the team at the Scrum board (virtual or physical). The person in the Scrum master role leads the daily stand-up by asking each team member a few questions:

- What are you working on?
- Are there any impediments?
- Do you need anything to be successful?

Each member of the operations team now knows what is expected of him/her for the day. Balance the expected work output with other team efforts such as trouble tickets and outside projects.

Burndown - The burndown tracks estimates of time with the actual time spent working on a project's tasks. The resulting chart will show a project approaching 0 as the level of effort needed to complete the project winds down. Teams get better at estimating with experience. Burndown can also demonstrate if a project is taking longer than planned or is ahead of schedule. Building a burndown chart can involve a spreadsheet or graphing application. It is common to build formulas in excel that will automatically update a pivot chart showing the project tracking. Some burndown charts are very complex and others are simple. The organization has to decide how fancy to get with this tool.

User stories - In Agile software development, user stories can be feature requests, bugs, or modules the team plans to code for a product release. In operations, user stories can be small or large projects. Smaller projects are usually broken down into smaller more easily digestible pieces otherwise a project can park in a swimlane for an inordinately long time bringing down team morale and potentially impacting productivity. Teams should see positive outcomes and accomplishments across the swimlanes.

Cross-functional teams - In a development environment, a cross-functional team could include developers, testers, management, and operations. The purpose is to introduce DevOps to software development by including roles that have a stake in the project at different levels. In operations, a cross-functional team could include people from systems administration, networking, security, and management.

Kanban

Kanban is a much less prescriptive Agile implementation. Kanban can be recognized by a similar task board to Scrum but often there are more columns. Kanban's strength is the work in progress (WIP) limit. Kanban doesn't require roles, timeboxing, or burndown tracking like Scrum.

Because there is no timeboxed sprints, work continuously moves across the swimlanes on the Kanban board. Daily stand-ups are critical in Kanban because there isn't a touchpoint at the end of a sprint to review completed work effort.

Kanban boards can have several additional columns to assist in the management of this continuous work flow. An example Kanban board may have "Coming soon" "Review" "Available" "In progress" "Acceptance" "Completed." The purpose of these additional columns is to enable teams to pull work into the "In progress" column as they finish other work. The "In progress" column and other columns will have what is called a WIP limit. There are a few schools of thought regarding WIP limits. Each organization must experiment with the WIP limit until a sweet spot is found for operations.

In Kanban for operations, the columns can be varied across teams or organizations. These columns are only provided as an example. The organization needs to find the Kanban workflow that works best for the team. There are several good resources that explain various ways of configuring a Kanban board. Sticking with the current example, let's review the columns in an example Kanban board to understand their purpose.

- Coming soon - these are tasks, projects, or user requests. They are un-prioritized and may be big or small.
- Review - These are tasks that are prioritized by management or the team during the daily stand-up. They are put "in the hopper" as work items that should be reviewed and possibly broken into smaller pieces if they are too large. The downside of too large is similar to Scrum when the user stories were too broad. If an in progress items its in the active queue too long, it takes up a WIP slot and can make it difficult to understand if the team is making progress on that item.
- Available - This item has been reviewed, broken into a reasonably sized task and approved by management or the team to be pulled into the active column at the next opportunity.
- In progress - Similar to Scrum, these are the tasks being worked actively by the team.
- Acceptance - When someone on the team considers a task complete, s/he moves it to this column. Acceptance means it is discussed at the next daily stand-up and possibly accepted as done by the team. Acceptance can also mean stakeholder acceptance. This could also be a testing phase for something that is rolling toward production. If something idles too long in this column, it will hold up other work because of the WIP in progress limits placed on this column.
- Completed - These are tasks that are accepted as completed and put into production.
- Impediments - Some boards might include a small section of a column to identify impediments. Impediments are tasks that cannot begin because of outside forces. Usually management intervention is required to resolve the impediment. By separating these tasks on the board, the team sends a message to management that this work requires outside intervention to move forward.

Work in Progress (WIP) limits WIP limits define the maximum number of tasks that can appear in that column on the Kanban board. The two schools of thought that seem to pervade are:

- 2n-1 - where n = the number of people on the operations team. The reason for this is to enable team members to work together on some tasks but to give enough tasks so team members stay busy.
- n-1 - where n = the number of people on the operations team. The reason for this is to encourage collaboration on the team and not to overwhelm them with too many tasks. If someone on the team completes all of their work, that person should be able to pull the next available task from the "Available" column.

What is the risk of having a WIP limit too low or too high? A high WIP limit might mean the team is taking on too much at one time. Each member of the team may get overwhelmed with the amount of work. Consider these are reviewed daily in the stand-up meetings and team members can pull new work from the "Available" column when current work moves to "Acceptance." High WIP limits mean that team members are less likely to work together on projects or tasks because each person has his/her own work to complete. A WIP limit that is too low could create a bottleneck, disallowing a team member from pulling new work into the "In Progress" queue because other people on the team have hit the WIP limit with their own work. The WIP limit is a sweet spot that the organization needs to discover through experimentation.

Whenever there is a bottleneck in Kanban, the team can refocus its efforts on the item stuck in the flow in order to unblock progress across the board. WIP limits force this to occur because a column with a WIP limit of 3 on the

acceptance column will not allow any tasks to move to that column if there are already 3 items waiting for acceptance. It is a way to keep work moving across the board.

Scrumban

Scrumban is a hybrid of the two previously mentioned methodologies. Operations teams seem to embrace Kanban or Scrumban because of the flexibility of daily re-prioritizing and the WIP limits that keep the team from getting overwhelmed.

A Scrumban implementation would take elements from both Scrum and Kanban. For example, operations might decide to define some roles, keep the review and retrospectives, hold the daily standup from Scrum while enforcing WIP limits and implement continuous work flow from Kanban.

Agile Toolkit

jira

The Tao of DevOps

What is DevOps

DevOps seeks to include the IT operations team as an important stakeholder in the development process. Instead of developers solely coding to meet the stakeholder's requirements on time and on budget, they are also held responsible for how easily it deploys, how few bugs turn up in production, and how well it runs. Developers also focus on providing software operations can asily support once it's in production. Instead of bringing operations into the conversation after the product is complete, the DevOps methodology includes operations in the development stream.

Development's view:

- Roll a product out to meet customer specifications within a certain timeframe
- Continuous delivery means recurring change as bugs are fixed and features added
- Fast changing environments are needed to support dev
- Agility is key

Operation's view:

- Supporting the product for customers
- Keeping a handle on IT security
- Planning for deployment to production state
- Changes are slow/incremental
- Consistent environments are needed to support operations
- Stability is key

Why DevOps is important

In organizations where DevOps is not a priority, development is often viewed as customer-focused by trying to solve problems and deliver solutions while operations is viewed as a barrier to development's mission. By combining these two often competing mindsets, both sides can be satisfied. The result is a product that potentially has fewer bugs,

higher availability, increased security, and a process for improved development over the life of the product that works for both the developers and the operations people.

It is also possible to implement a DevOps methodology in a pure operations teams. In this scenario the operations team is also Development because they stand up a webserver, provision virtual machines, or code configuration management systems. In this case, operations needs to wear both the development and operations hats by meeting customer needs while also addressing security and supportability of the solution.

What isn't DevOps

A person cannot be a DevOp. You don't hire a DevOp.

The importance of Documentation

What to document

- Runbooks? SOP? (cparedes: might be worthwhile even though we want to automate SOP's away as much as possible - what should we check at 2 AM? What do folks typically do in this situation if automation fails?)
- Architecture and design (cparedes: also maybe talk about *why* we choose that design - what problems did we try to solve? Why is this a good solution?) How to manage documentation

Documentation through Diagrams

Anecdote At one job we had a single network engineer. He had a habit of walking up to a whiteboard to explain something to the systems folks. He would proceed to draw what we considered a hyper-complex-looking diagram showing the current or future state of some networking solution. We could never keep his configurations in our heads like he did and he wasn't always around when we had a question. One of us figured out that we should take a picture of the whiteboard after he finished drawing. These pictures went into the operations wiki. They weren't beautiful but they saved us time when we could easily refer back to the pictures we took.

Diagrams don't always have to be professional visio-quality to count as documentation.

Functional diagrams

Technical diagrams

Working with other teams

Soft Skills 201

Soft skills at the 201 level attempt to inject higher level business awareness and practices into an otherwise sound technical operations person to create a senior operations engineer.

Soft skills at the 201 level include positioning, budgeting and the financial process, using metrics effectively, demonstrating impact, risk management, managing customer preference, and thinking strategically.

Business Acumen in Operations

What is business acumen? Business acumen as a leadership competency simply defined as a general understanding of business principles that leads to an organization's success. While operations professionals do not need to be senior

executives, development of business acumen as applied to operations can help to bridge the gap between the organization's senior leadership and the operations team. Business acumen as applied to operations works on multiple levels. In many organizations, operations is a service unit within the larger organization but it also serves the needs of the organization as a whole. The savvy operations person will look at operations within that context, applying the following skills to appropriately position operations and act with the best interests of the greater organization in mind. This also helps when trying to make the organization DevOps friendly.

Distilling the definition of business acumen for operations yields the following important skillsets:

- Understand the role of operations within the context of the organization to correctly position operations.
- Think broadly about decisions and act decisively.
- Support and promote change as needed.
- Develop basic business skills that allow operations to communicate within the executive suite.

Understanding the role of operations

Under any of the operations professions, the most fundamental role of the operations person is to deliver services to a set of customers. To build upon this further, the operations person maintains existing IT infrastructures, translates customer requirements into tangible and actionable solutions, assists in the protection of customer information and services, and advises stakeholders on application of technology under existing limitations of time, money, or capabilities.

By thinking of operations as a business unit instead of a forgotten office within the organization, the operations engineer is already thinking at the correct level to assess how to support the needs of the organization.

Understand how the organization competes within its industry. Commercial entities, non-profits, educational institutions, government agencies all measure success in some way. For commerce, it will be sales and profit. For educational institutions, it might be numbers of incoming students and retention rate of students. For a non-profit it might be the number of people willing to give to support the work of the organization and the number of people who use its services.

All of this leads to correct positioning of operations within the organization.

- What are the core competencies of operations and how do they serve the internal business units and the organization as a whole?
- What core competencies should operations develop in order to better support the organization's mission?

Maintaining Existing IT Infrastructures

The most visible role of Operations is to maintain the status quo. For the system administrator this means maintaining servers and processes such as logging, monitoring, backups, authentication, or naming services. For the network administrator it means maintaining routers, switches, the edge network, gateways, or the relationship with the corporate Internet Service Provider (ISP). A security engineer might be responsible for maintaining a vulnerability scanning capability, incident response policy and processes, intrusion detection systems, firewalls, and a customer security awareness training program. Operations may also be responsible for maintaining access to internal services (e.g. financial systems, corporate content management systems, procurement systems, etc.) that may impact the various business units within the organization. These roles are distinct but there is sometimes overlap between them in smaller organizations where fewer people serve in multiple roles.

Translating Customer Requirements

Operations roles are customer service positions. These careers require a level of customer interaction because the services delivered by the Operations professional must be driven by customer needs. In this case, customer is used to

mean the business, organization, or other entity that is employing the Operations professional. Some questions to ask to help the Operations person understand requirements from the customer perspective:

- What is the core mission of this organization?
- How does Operations support, hinder, or allow the organization to innovate for the mission?
- Who are the core customers (internal, external, or both)?
- What does the organization need from the Operations professionals?
- Why should this organization come to these Operations people for this service or solution? What is the value proposition for Operations within this organization?
- How could Operations provide more value: higher level of competitiveness, faster service delivery, stronger security, or other benefit that aligns with the mission?

Translating customer requirements is key to focusing the efforts of Operations. Operations work can be a slippery slope where the professionals are spreading themselves too thin on projects and deliverables that do not serve the organization's mission. One way to focus the efforts of Operations is to answer these questions and to ensure that the Operations organization, whether insourced or outsourced, is delivering services that provide the most value.

Protection of Information and Services

Often the Operations professionals in an organization are the people who most completely understand the technical risk to organizational assets from an IT perspective. Senior management within an organization will usually understand risks related to financials, competition, manufacturing, etc. but they often do not understand IT enough to make an informed decision. Operations professionals are the ones with the deep technical expertise required to comprehend risks, threats, vulnerabilities, and countermeasures. They use this expertise to express their concerns in a way suitable for senior management. This is another area where the Operations professional is communicating with the organization's leaders to advise on appropriate actions to address IT security where it makes sense for the organization.

Areas where organizations need the Operations professional to advise on IT security could include threats to data from internal and external sources, hardware failure, site availability or resilience, data preservation, and information integrity. Again, these areas are dependent on the organization's mission.

For example: an ecommerce organization will most likely want strong site availability and protection of customer personal information. The operations professionals might build a site with high resilience and availability including use of Content Delivery Networks (CDNs); strong encryption (not only for the ecommerce session but also data at rest); role-based access for internal employees accessing customer information, to reduce access to only those people who need it. Organizational leaders often do not understand how these solutions are implemented so it is up to the Operations professional to communicate the threat, solution, cost, impact to the organization of implementing the solution.

Advising within Current Limitations

The Operations professional who advises an organization must also consider limitations that impact the potential solution. Cost, timing, expertise within the organization, available time of the people who would implement the solution, or IT security issues may be considerations. For example, decision makers within the organization will need to know what is possible and at what cost so they can decide how to spend the organization's money. Good, fast, or cheap (pick two). It may be the operations professional's responsibility to explain this concept from an IT perspective.

Thinking broadly

Broad thinkers can look at a problem from the viewpoint of other people and business units within the organization. Instead of insular thinking, they approach problems with a broad-minded perspective. How do decisions impact other

areas of the organization and, alternatively, how does the organization view this particular issue? Those with strong acuity for business will see the big picture and be able to understand the implications of a decision on more than just operations.

In some cases it may not be a problem, but an opportunity that injects potential life into an organization or recalibrates it. Business leaders, stakeholders and customers often don't understand what technology can do for them. Operations should understand the organization well enough to see where technology can support innovation. This leads into change as a constant.

What would it take to make this happen? What are the missing ingredients for success?

Promoting Change

The operations world changes rapidly, more rapidly than other sectors. Operations people cannot afford to cling to a specific operating environment, hardware platform, or technical solution because the industry has already started moving toward the next innovation.

Once operations understands how the organization competes to stay viable in its marketplace, operations can leverage technology to support those needs. Operations may be the first business unit to grasp the importance of a technology innovation that would improve the mission work of the business.

Identifying that change is only the first step. Next operations must be able to demonstrate the benefit of the innovation to the organization's leaders in a meaningful way to promote change.

Building basic business skills

Basic business skills include simple tasks such as learning to use Excel to build a basic budget and navigating internal business systems such as procurement, capital expenditures (CapEx) and contracts. Some skills are the same everywhere (e.g. Excel) and some require study of the internal organization (e.g. procurement). Understanding CapEx means being able to understand what is or isn't a capital expenditure (e.g. some hardware purchases may be) within your organization and knowledge of your organization's depreciation process.

Budgeting and Financial Skills

A basic knowledge of Excel includes formulas, formatting for readability, using multiple worksheets and importing external data. More advanced Excel knowledge includes use of macros, pivot tables and pivot charts.

Some operations folks use other Excel-like programs such as OpenOffice or LibreOffice spreadsheet programs. Use caution when using something that the senior leaders do not use. If the whole organization has adopted LibreOffice as the standard spreadsheet application, that works. The problem occurs when the boss wants to share the spreadsheet with some of the organization's senior leaders and the file format doesn't translate exactly or the file is unreadable to them. In this case, try to bridge the gap between operations and the executive suite by using the same tools. Formats do not always translate between two different spreadsheet programs.

Building a basic budget requires institutional knowledge. How is employee labor computed? Understand operations' income and where it comes from. Are any employees billable to other projects? Is there a flat budgetary structure with a single cost center for all labor or are there multiple cost centers. Is there any income that has special restrictions? How are purchases handled: things such as parts, services, software, contractor services? Does operations have to account for overages or money not spent at the end of the fiscal year?

Generally, organizations have financial people who can provide reports for various cost centers. If operations fits neatly within one or more cost centers, these reports can help build a budget. If operations is combined with other projects or business units, then the work of separating operation's budget becomes a bit more complex. Starting with these reports is a good first step.

To really understand how these reports work, understand how operations is paid and how it spends within the organization.

How is operations funded?

Where does operation's base funding originate?

- Is Operations billable or do they have constant funding from year-to-year?
- Does someone need to request this money or is it always there?
- How are pay increases funded?
- Is there only one source of money or are there multiple income streams?

Does everything come out of one cost center or are there multiple cost centers?

- If multiple, are they broken down by project, type of expenditure (labor, contractors, services, supplies)?

Is any of the money special?

- Does it expire?
- Does it come with strings/hooks to specific projects or billables?

How does operations spend?

- How are employee salaries computed to include benefits and overhead?
- How are contractors paid?
- Are there special rules for obligations? In some organizations, some kinds of money must be allocated up front and cannot be reclaimed even if not spent until after the contract or service has completed or the fiscal year has ended.
- How do operational purchases work within the organization (parts, services, software, training, travel, supplies)? Who pays for these purchases? Who tracks these expenses?
- Does the organization have a CapEx process and where does that money originate? Does depreciation impact the budget?
- Are there any hidden costs?
  - Service fees from internal organizations?

Answering these questions and looking at reports from within should provide most of the answers. Operations may have to implement tracking to get some answers if they aren't easily identified in the reports.

Why would any sane operations person want to go through all of this to assemble a budget?

- Operations is understaffed and wants to ask senior management to hire more people
- There has been staff turnover and operations needs to fill those positions. How much is available and what opportunities exist to do something different?
- Senior management is asking hard questions about the operations budget (e.g. why do we spend so much on operations, where does the money go?).
- Operations is considering a student hire or contractor to help with some short-term work but operations cannot move forward until demonstrating that they are spending wisely.

Budgeting for impact

Just putting numbers in a spreadsheet isn't budgeting. What do the numbers show? Is operations spending too much on senior people? Equipment? Vendor maintenance? Where is the majority of spending (commonly it is labor)? An easy to present budget can also help to understand if operations is well managed.

Take that same view of the budget that gave visibility into operations and use it to support a request or a claim to senior management.

As an example: consider a senior person leaving the organization. Operations needs to fill that slot with a new person to avoid getting overwhelmed.

- Does this vacant position present an opportunity?
- Does operations need to hire someone with specialized experience in a new area?
- Could operations benefit from hiring two junior level people using the same salary slot as the former senior person? Does that work mathematically within the organization's hiring rules?
- Could operatoins reduce the overall cost of operations to help the organization by hiring one junior person and growing that person?
- Could operations hire a junior person and use the remaining money to refresh hardware or invest in a new technology to help the organization?

See how to make some of these arguments mathematically in a spreadsheet. The part that is missing is the "why" and that's where the impact comes in. Senior management may believe that operations needs to reduce overall costs. This is when operations needs non-numerical supporting evidence to persuade management that operations does need to hire a specialist or make the case for an apprentice that would achieve a cost savings but would reduce capabilities until the person came up to speed within the operations team. Budget decisions have consequences: make sure those impacts are clearly illustrated within the numbers but also be prepared to explain the non-monetary impacts. This includes risks to the organization such as reduction in capabilities.

When preparing for a big budget presentation where operations is asking for a decision that will impact operations, consider the following supporting strategies:

- Enlist customer support. Customers are asking for improved capabilities, better response, new technology. How can they provide input to management that operations needs more or different resources to serve them better?
- Find out if there are any new initiatives within the organization that would rely on specific expertise or additional operations resources. This demonstrates a tangible need (e.g. Project X will require 50% of someone from operations to implement their technical plan).

Using these additional supports requires knowing the organization and having a good relationship with the customers. Ideally, customers come to operations in the planning stages of new projects in order to get feedback on potential technology issues before they begin work. That makes this step a bit easier. If not, then begin reconnaissance by talking to project leaders or middle management within the organization.

When researching organizational needs, start with some basic questions:

- Planning anything new in the next year?
- What projects is the group starting?
- What technologies are not in use that would make the unit more productive?
- Does operations provide the right level of support to the division?

Exercise:

Choose a budget scenario from above or make up your own.

- How would you build a basic budget to persuade senior management on your issue?

- What would be important to highlight?
- What non-monetary supporting information would help your cause?

The cost-benefit analysis

The cost-benefit analysis, or CBA, provides senior management with concise proof that operations has done its homework when proposing a solution.

The first step in the CBA process is to know the audience. The higher up the organizational chain, the less detail required. Before presenting a CBA to management, prove that the solution is the best one.

Before detailing the cost of a solution, operations needs to know existing expenditures without it. What is the cost of not doing anything? This is where the benefits of performing a solution would need to outweigh the status quo.

Building a case

Everything in a CBA should be represented in the same units, the most common being money. Consider benefits to the solution in terms of savings, efficiency, increased income to the organization.

Cost should include anything that explicitly adds to the total cost of the solution:

- Employee labor
- Contractor costs
- Maintenance fees
- Up-front costs and licensing
- Hardware
- Depreciation
- Facilities costs (outfitting a space)
- Provisioning or migration costs
- Networking

Benefits should include anything that is an outcome of the solution:

- Increased productivity
- Increased organization efficiency
- Increased income to the organization
- Increased capabilities that enhance the organization in another way

Putting it all together

Todo

Might give an example here. Need to write more explaining how to assemble the pieces.

Exercise

Put together a CBA for a recent project or task you worked on or encountered:

- How would you estimate costs that are not known?

- How do you monetize benefits that are not explicitly monetary?
- What does the result tell you?
- How could you sell this idea to non-technical people using the CBA?

Navigating the capital expenditure process

The Capital expenditure (CapEx) process is used by organizations to purchase assets that have value across multiple tax years. In operations CapEx usually means new equipment or equipment that extends the useful life of existing equipment beyond the existing tax year.

CapEx allows an organization to depreciate an asset over the estimated useful lifespan of that asset. How is this valuable? On the organization's balance sheet, only part of the total expense is counted for a specific tax year. The amount of the expense depends on the type of depreciation used.

Straight Line Depreciation

With straight line depreciation, assets are depreciated at an equal amount each year. A piece of equipment with an estimated useful lifespan of 4 years would be depreciated 25% per year on the organization's expense sheet.

Accelerated Depreciation

Accelerated depreciation usually frontloads the depreciation costs. This method may more accurately reflect the value of equipment because there is a greater depreciation at the beginning of the cycle. An example of accelerated deprecation might require a piece of equipment to be depreciated over 4 years at a rate of 40 percent per year. There would be a greater expense in the first year, calculating 40% of the total value of the asset as depreciation. In the second year, compute 40% of the remaining value, and so on until the fourth year at $0.

An analogy to help explain Accelerated depreciation might be the purchase of a new car. The car depreciates the moment it leaves the lot. Even if the owner were to sell the car soon after purchasing it, the car has already significantly decreased in value.

Building a business case

Todo

write this section.

Distilling information for impact

This skill goes hand-in-hand with budget but it is also an excellent standalone skill. Operations deals with complex implementation of technology. To the non-technical person, the architectural diagram on the whiteboard looks like a Rube Goldberg machine.

The further up the management chain, the more distilled information should get. Senior leaders do not usually need or want deep technical detail. When presenting a complex solution, it is fine to have one diagram that is completely unintelligible to them as long as it is only used to to demonstrate that operations did more than throw a blade in a rack and spin it up to achieve the solution. The most important part of the presentation is the part where operations answers the questions in the heads of senior leaders even before they ask them.

What are their questions?

- What are we trying to accomplish?
- What do we do today and how is this better?
- How do we know this is the best solution?
- Do we have the right people to make it happen?
- How much will it cost?
- How long will it take?
- What is the benefit if we do it?
- What is the risk if we don't do it?
- How do we know if it worked?

Exercise

Take an idea you have and use the questions above to try to build a case for senior management to fund this idea.

Specific Examples

Below are some specific examples to demonstrate the importance of soft skills in operations. In each example, soft skills closed the deal because they enabled the operations person to see the situation from other perspectives and communicate the needs of operations in terms of the organization as a whole.

Selling system changes and new proposals

Negotiating budgetary constraints vs. need/want requirements

Evaluating a product offering

Labs exercises

Bare-Metal Provisioning 101

Install CentOS by hand

Install the same node using kickstart

Bare-Metal Provisioning 201

Setup a basic cobbler server

Build a profile

Kickstart a node

Change the profile, re-kickstart the node

Cloud Provisioning 101

Setup a free-tier account with AWS

Spawn a Linux node from the AWS Console

Cloud Provisioning 201

Spawn a Linux node using the AWS API

Attach an Elastic IP and EBS to it

Database 101

Install and start up MySQL

Create basic relational database / tables, using a variety of field types

Grant and revoke privileges

Install Riak

Write (or provide, probably depends on where this fits in relation to scripting tutorial?) basic tool to insert and retrieve some data.

Database 201

Spawn up second VM/MySQL install

Set up Master->Slave replication

Deliberately break and then fix replication

Set up Master<->Master replication

Introduction to percona toolkit

Set up Riak Cluster, modify the tool from 101 to demonstrate replication.

Database 301

Galera cluster

- Introduction to variables and their meaning. Tuning MySQL configuration (use mysqltuner.pl as a launch point?), pros and cons of various options.
- Introducing EXPLAIN and how to analyse and improve queries and schema.
- Backup options, mysqldump, LVM Snapshotting, Xtrabackup.

Automation 101

Do you need it? How much and why?

http://www.kitchensoap.com/2012/09/21/a-mature-role-for-automation-part-i/

- Talk basic theory and approach in terms of Idempotency and Convergence.
- Write a bash script to install something as idempotently as possible.
- Discuss Chef and Puppet and while reflecting on the bash script.

Automation - Chef 201

Setup an Opscode account

Setup your workstation as a client to your Opscode account

Download the build-essential cookbook, and apply it to your workstation

Automation - Chef 301

Setup a chef-repo

Write your own cookbook

Automation - Chef 302

Setup your own Chef Server

Write your own resources/providers

Write sanity tests for your code

Automation - Puppet 201

Install Puppet

Install a Forge module using the module tool

Apply it to your local machine

Automation - Puppet 301

Install Puppet

Create your own module

Apply it to your local machine

Package Management 101

Setup a basic YUM or APT repository and put some packages in it

Setup a local mirror of CentOS (or what have you)

Setup a client to install from it

Package Management 201

Build a simple RPM or deb

FPM

Build automation fleets

koji

D

Version Control with Git 101

Open a GitHub account

Create a new repository called 'scripts'

Place a useful shell script in it

Commit and push

Make a change, commit and push

Create a branch, make a change, commit, and push

Create a pull request and merge the branch into the master branch

- Read Chapters 1-3 of the Pro Git book online
- Work through Code School's Try Git online

DNS 101

Install Bind

Configure one zone

Show DNS resolution for an A and a CNAME record in the configured zone

HTTP 101

Install Apache

Configure a virtual host

Display a simple web page

Learning and the Community

Learning and strategies for improvement

Certification

Certification is a broad, overarching topic that could easily span an entire section. The goal here is to briefly describe some of the most useful and common aspects of certification and their relation to operations specific roles. There are a wide variation of certifications, some bear more weight and clout than others. Therefore it is important to get a basic understanding of the various certification options available if you choose to pursue certification to help strengthen your career in ops and increase your hiring eligibility with prospective employers.

There is some disagreement as to the value that certifications provide but it is important to make readers aware of the common arguments for and against certifications for purposes of completeness.

One such argument against certifications is that they merely test trivial knowledge which does not translate well into IT ability by memorizing facts and using test dumps in order to pass certain tests. The counter argument to that philosophy is that many of the higher level certs are much more difficult to obtain than by simply doing brain dumps, as they are typically a combination of written and lab type scenarios. Often times HR departments will use certifications as a filter and many IT managers and hiring staff believe experience to be a much stronger indicator of performance and ability than certifications. Another example for preference of certified individuals are government roles or otherwise top level security positions that lean heavily towards higher level security certifications.

Generally, there are a few scenarios that you will see with regards to certification.

- The first scenario, and most common for those just starting out, is the group who use certification as a way to get their foot in the door or to launch their careers in ops.

The types of certifications that these individuals target are typically the entry level, easiest to obtain certifications which will yield the greatest improvement of job prospects for those that obtain them.

- The second scenario that is typical is one where certain certifications are required either for an individuals' place of employment or are used as a requirement or prerequisite to obtain a position at a company.

Many times, Managed Service Providers (MSP's) require a certain level of certification amongst their employees to retain their status as a certified partner with some technology companies, like Cisco or Microsoft for example. These companies are required to have various levels of certification for their employees in certain specific areas and are also required to continually have their employees become re-certified periodically to retain their status as a "partner" to keep various perks and relationships associated with the vendors they certify and partner with.

The following is a partial list of some of the most popular certifications followed by a brief description of the certification and how it can help with a career in a specific area of IT operations.

Comptia

A+ - Designed as an entry level certification, typically geared towards desktop support and help desk type positions. This certification demonstrates basic ability to troubleshoot hardware and software related problems.

Network+ - This is an entry level network certification. It covers topics in the realm of networking and is designed for those looking to go into careers as network administrators, network technicians, help desk and desktop support roles.

Security+ - Perhaps the most popular and trusted Comptia certification, the Security+ covers a number of broad topics in IT security including network security, compliance topics, cryptography, access controls and identity management. This certification is geared towards those in security roles, such as security engineers, consultants, systems/network administrators and security administrators.

Linux+ - Another introductory, entry level certification offered by Comptia. This certifications covers topics that are commonly seen in the Linux world, such as filesystems, different shell commands, package management, scripting and some other administrative topics unique to Linux environments. Those who obtain the Linux+ typically qualify for roles like junior network administrators, junior Linux administrators, web administrators and Linux based DBA's.

One nice things about the Comptia certs is their vendor neutral position in the industry. The certifications are great for building foundation building with no prior knowledge or experience and are nice stepping stones into jobs as well as other, more advanced certifications.

Cisco

CCNA -

CCNP -

CCIE -

CCDE -

CCIA -

Microsoft

MCITP -

MCSE -

MCM -

Redhat

RHCSA -

RHCE -

VMWare

VCP -

This list is only meant to be used as a guideline, and is by no means comprehensive. There are many other IT certifications out there and are simply too numerous to cover. The certifications and their requirements are in constant change and a number of the certifications listed here have gone through numerous revisions, some of which are currently going through revision. This is one downside to certification. Because the industry changes so rapidly these certifications can quickly become irrelevant if the necessary time is not taken to keep current with your knowledge and industry technologies.

There is one final note on certification that should be emphasized here. There are many resources available on the topic of certifications, a quick Google search will more than likely yield a large number of results. One good site worth mentioning is the TechExams site. Here you will find many valuable resources and materials to help with pretty much all aspects of certifications. Their forums are an excellent place to exchange ideas and get quick feedback as well.

Explicit vs Tacit knowledge

Explicit knowledge can be defined as that gained from books or listening to a lecture. Basically, this type of knowledge would be some form of reading or auditory resource. It is typically easy to transfer to others, an example would be a manual for driving and operating a car.

Tacit knowledge can be defined and described as knowledge gained from experience, action and practice. It is typically difficult to transfer this type of knowledge to other people. A good example would be flying a plane.

Let's start off by making a distinction between different types of knowledge. The practice of system administration relies heavily on both types of learning so just one type of experience is not enough to be great in this field. Essentially,

the two knowledge types work hand in hand. So for example, reading a ton of books, while useful in its own right will not be nearly as effective as reading books and then applying the knowledge gained from hands on experience. Likewise, if somebody never bothers to pickup a book and relies entirely on hands experiences they will not be as knowledgeable as someone who incorporates both types of knowledge. It is in the opinion of many in the field that much more can be learned from hands on experience than by books alone.

Types of learning

There has been a good deal of research done on this subject but for the purposes of this post I would like to boil this all down to what are considered the three primary or main styles of learning. Types of learning play an important role because they work hand in hand with explicit and tacit knowledge. Each one of these different styles represents a different sort of idiom to the learning experience. So here they are:

- Visual
- Auditory
- Kinesthetic

It can be argued that employing a good variety of learning and study methods would be the most appropriate way to develop your skills as a sysadmin or any other career related in the operations field. Instead of saying that one is better than another, one should employ all of these types learning in their own life. Take a look at yourself and figure out how you learn best and then decide which method(s) are the most and least helpful and then decide how to make these styles work to your advantage.

For example, having a piece of reading material as a reference or as an introduction is great. If the subject material is difficult and isn't easily understood, a good next step to take is internalizing things by listening to or watching others. Finally, with a good enough understanding about a topic, quickly put things into your own experiences. It is much more easy to remember things when you are able to experience them yourself.

Learning strategies

It is important to highlight some of the major tactics that can be utilized when attempting to learn a new subject. Here are some different strategies and techniques for learning new and difficult to understand information. Many of these strategies work together or in tandem so they may described more than once.

The Feynman technique - This is as close to the end all be all that there is when it comes to learning. Everybody is probably familiar with this one, but am guessing the name is unfamiliar. This technique is used to explain or go through a topic as if you were teaching it to somebody else that was just learning about it for the first time. This basically forces you to know what you're talking about. If you get stuck when trying to explain a particular concept or idea, make a note of what you are struggling with and research and relearn the material until you can confidently explain it. You should be able to explain the subject simply, if your explanations are wordy or convoluted you probably don't understand it as well as you think.

Reading - This is a great technique to get an introduction to a topic by reading up on (and bookmarking) what information you feel to be the most important, whether it be official documentation, RFC's, books, magazines, respected blogs and authors, etc. It is a good idea to take very brief notes when something looks like it would useful so to try it out yourself.

Watching/Listening to others - After getting a good idea from reading about a subject it is good to reinforce this by either watching demonstrations, videos, listening to podcasts, lectures or anything else that will show how to get a better idea of how to do something. An example of this would be to put on a podcast. It kills time as well as improves knowledge at the cost of nothing. Very efficient! The same with videos and demonstrations, the only thing holding you back is the motivation.

Try things for yourself - Sometimes this can be the most difficult approach but definitely can also be the most rewarding, there is nothing better than learning things the hard way. Try things out for yourself in a lab or anywhere that you can practice the concepts that you are attempting to learn and understand.

Take notes - This is important for your own understanding of how things work in a way that you can internalize. Take notes on simple things like commands you know you won't remember, related topics and concepts or even just jotting down keywords quickly to Google for later on. This goes hand in hand with the reading technique described above, just jotting down very simple, brief notes can be really useful.

Communicate with others - There are plenty of resources out there for getting help and for communicating and discussing what you learn with others. /r/sysadmin would be a good starting point. IRC channels are another great place to ask questions and get help, there are channels for pretty much any subject you can think of out there. There are good sysadmin related channels at irc.freenode.net, if you don't already utilize IRC, take a look.

Come back later - Give your brain some time to start digesting some of the information and to take a step back and put the pieces together to begin creating a bigger picture. If you have been working on learning a new concept or subject and felt overwhelmed and feel stuck, take a break. Do something completely different or think about something else entirely and came back to the subject later on with a fresh perspective. Sometimes these difficult subjects just take time to fully understand so taking breaks and clearing your head can be very useful.

Sleep on it - Have you ever heard of the term before? This may sound crazy but sometimes if there is a particular problem that you're struggling with, think about it before going to sleep. By blocking out all outside interference and noise it is much easier think about, come up with fresh perspectives and ideas and often times you will wake up with an answer the next morning.

Break stuff - One of the best ways to incorporate a number of these techniques is to intentionally break stuff in your own setups. Triple check to be sure that nothing important will get broken first and then go ahead and give it a try. A much deeper and more intimate relationship with the way things work, why they work and how they get broken occurs when things get broken. The great thing about using this method is that it is almost always useful for something in the future, whether it be the troubleshooting skills, the Googling skills or the specific knowledge in the particular area that needed to be fixed.

Practice, practice, practice - There is just no way around it. To get better at something one must dedicate time and be prepared to practice like an absolute maniac. For operations roles and system administration this can partially come from practical job experience but it also comes from dedicated study and lab time. The hands on component is where most experience and time will come from and becoming better doesn't just happen, it takes cultivation and time, just like with any other skill. Stick with it and never stop learning and improving on your skillset through practice and experience.

Things to keep in mind as you learn how to be an engineer

General Tips for improvement

These simple tips can go a really long way. There is no magical instant way to improve yourself. If you take nothing else, just remember the following. The best way to see results and really work on yourself starts by changing your habits, working hard and being consistent.That might not be what you are looking for, but it has been proven to be true time and again that even by making just these few adjustments can go a long way in becoming better at what you do.

Exercise - Just doing a Google search will reveal all the information on the massive benefits of proper exercise. Even just this one tip can make a huge difference in the way you think and feel. It is not recommended to completely change the way you live your life when starting out, especially if you are sedentary. Just make a simple change as something to start with and work from there. There are many benefits of exercising and working your body regularly will help you improve your mind.

Sleep - This is probably the most important thing to remember when you are trying to work on hacking your mind and improving yourself. 8 hours of sleep seems to be the general rule of thumb, and it should not be overlooked when you are evaluating yourself and your goals for getting to where you want to be. If you want to wake up early, you need to

go to sleep early, it really is as simple as that. It is also important to be consistent on your sleep schedule so your body can get used to when it should slow down and when it should speed up (even on weekends!). For example, getting in a routine of winding down at a certain time, say 9 pm every night by reading a book for an hour to train your body that it is time to sleep. Read until say 10 pm every night if you want to wake up at 6 am to get the sleep consistency your body needs, also giving your body enough time to repair and heal itself to get up and going.

Diet - Also important. Everybody is different so please take this advice at face value. As with anything else, it is not recommended to go all out and make completely polar changes to every eating habit at once. You will crash and burn like many others.So while it may work for some you generally will be safer and more likely to make a lasting impact if you take things slowly. Work on one thing at a time and gradually make the changes to improve your diet and health. As an example, start by cutting out something small, like cutting out a particular type of food that isn't exactly healthy. Not entirely, but even just cutting back is a good first step. Basically doing something is better than doing nothing.

Golden rules for careers in ops

- A sysadmin is there to support the customer(s)
- Leave your ego outside
- Listen
- Learn to communicate tech ideas to non-techies
- Learn how to triage
- Take time to document
- Start with the known and move to the unknown
- It isn't just backup, it's also restore
- Learn to do post mortems

Other skills that can help you

- Be personable
- Learn to code
- Learn patience
- Learn to budget

Where to look for help in the community

Local user groups

- Local LOPSA Groups:
- BayLISA
- Back Bay LISA

Mailing lists and forums

- lopsa-tech@lopsa.org
- lisa@usenix.org
- systers [women in computing]
- ops-education Google group

IRC Channels

irc.freenode.net

- #OpsSchool
- #lopsa
- ##infra-talk

Membership organizations

- USENIX LISA Special Interest Group (SIG)
- League of Professional System Administrators (LOPSA)

Conferences and Events

- USENIX Large Installation System Administration (LISA)
- Velocity
- Scale
- Ohio LinuxFest
- Lopsa-East
- Cascadia
- VMWorld
- Toorcon
- Blackhat
- Derby Con
- Def Con
- Schmoocon
- TechEd
- Redhat Summit
- Cisco Live!
- HP Discover
- EMC World

Subreddits

- /r/sysadmin
- /r/linuxadmin
- /r/commandline
- /r/networking
- /r/netsec
- /r/vim
- /r/python
- /r/programming
- /r/learnprogramming

Podcasts

- Security Now! - Security
- The Linux Action Show! - Linux focused podcast
- Techsnap - Various IT topics
- Hak 5 - Hacking and security related
- Podnutz Pro - SMB system administration
- Windows Weekly - Windows news
- Packet Pushers - Networking
- RunAs Radio - Various IT topics
- The UC Architects - Exchange, Lync
- The PowerScripting Podcast - Powershell
- FLOSS Weekly - All things Open Source
- Stack Exchange Podcast - Various Programming and administration topics
- Healthy Paranoia - Security
- My Hard Drive Died - Anything and everything Hard Drive related

RSS Feeds

ServerFault

Sign up and participate. Ask your own questions, but also answer questions that look interesting to you. This will not only help the community, but can keep you sharp, even on technologies you don't work with on a daily basis.

Books (and concepts worth "Googling")

- Time Management for System Administrators, Thomas Limoncelli
- The Practice of System and Network Administration, Thomas Limoncelli
- Web Operations, John Allspaw and Jesse Robbins
- The Art of Capacity Planning, John Allspaw
- Blueprints for High Availability, Evan Marcus and Hal Stern
- Resilience Engineering, Erik Hollnagel
- Human Error, James Reason
- To Engineer is Human, Henry Petroski
- To Forgive Design, Henry Petroski

University Programs that teach Operations

A list of System Administration or Ops related classes or degree granting programs. It would be well worth your time to compare their syllabi, course outcomes, exercises etc.

- http://www.hioa.no/Studier/TKD/Master/Network-and-System-Administration/
- http://www.hioa.no/Studier/Summer/Network-and-system-administration
- http://www.cs.stevens.edu/~jschauma/615/
- http://goo.gl/4ygBn
- http://www.cs.fsu.edu/current/grad/cnsa_ms.php
- http://www.rit.edu/programs/applied-networking-and-system-administration-bs
- http://www.mtu.edu/technology/undergraduate/cnsa/
- http://www.wit.edu/computer-science/programs/BSCN.html
- http://www.his.se/english/education/island/net–og-kerfisstjornun/

LOPSA has an educator's list

Aleksey's report on university programs

See also

A list of System Administration or Ops related classes or degree granting programs. It would be well worth our time to compare their syllabi, course outcomes, exercises etc.

- http://www.hioa.no/Studier/TKD/Master/Network-and-System-Administration/
- http://www.hioa.no/Studier/Summer/Network-and-system-administration
- http://www.cs.stevens.edu/~jschauma/615/
- http://goo.gl/4ygBn
- http://www.cs.fsu.edu/current/grad/cnsa_ms.php
- http://www.rit.edu/programs/applied-networking-and-system-administration-bs

- http://www.mtu.edu/technology/undergraduate/cnsa/
- http://www.wit.edu/computer-science/programs/BSCN.html
- http://www.his.se/english/education/island/net–og-kerfisstjornun/

In addition, Usenix SAGE (now LISA) used to have a sage-edu@usenix.org mailing list, but I don't know if that is still active. LOPSA has https://lists.lopsa.org/cgi-bin/mailman/listinfo/educators

http://www.verticalsysadmin.com/Report_on_programs_in_System_Administration__25-June-2012.pdf

Reading List

This is a list of additional material should you wish to dive deeper on particular topics.

Book	Author (last name)	Description
Network Warrior	Donahue	Picks up where CCNA leaves off. Lots of real-world networking advice.
DNS & BIND	Liu & Albitz	One of the best books available on DNS
The Practice of System and Network Administration	Limoncelli	Focused more on the traditional IT role, but plenty for modern sysadmins to learn
The Practice of Cloud System Administration	Limoncelli	Focused on modern system administration and devops-y/cloud-y topics
Time Management for System Administrators	Limoncelli	Learn how to better manage your time. A classic book.
Web Operations	Allspaw	Full of lots of practical advice on web operations. Limoncelli's Cloud System Administration is kind of like an updated and more in-depth version of this.
Release It!	Nygard	Focused more on the developer side of of application design/deployment, but sysadmins will learn a lot from this book.
UNIX and Linux System Administration Handbook	Nemeth, et el	A classic book on UNIX/Linux system administration. Does not cover modern tooling/practices, but rather more in-depth teaching of Linux/UNIX operating systems
The Phoenix Project	Kim	A parable about DevOps, that does a great job explaining what DevOps is all about.
Continuous Delivery	Humble & Farley	Discusses continuous delivery practices, anti-patterns, and solutions. One of the few books that brings together many related topics (CM, VCS, etc) into a cohesive idea of continuous delivery.

Contributions

Ops School is a community driven effort, and we always need people to contribute. Currently we need people who are able to fill in areas in our documentation - whether it's a little or a lot, everything helps.

How we'll organize work

This is a living document. We write headings and their breakdowns as bullet points. We turn each bullet point into the material we want people to read. That's right. The syllabus IS the course. For now, at least, until we find a better way.

You should start writing the actual material to be taught right into this syllabus. We'll worry about formatting and things later. As you write, remember that "101" material is aimed at people who are working up to being junior sysadmins. They're mostly blank slates.

How to contribute

You can find the documentation source on GitHub and send pull requests:

https://github.com/opsschool/curriculum

Please fork the repo to your own account, create a branch, make your changes there, and issue a pull request from your fork/branch to `opsschool:master`. Be descriptive in your commit message for anything non-obvious.

If all of this git and GitHub stuff is a little overwhelming, take a look at GitHub's documentation. If you have still have questions after reading that, please feel free to join #opsschool on irc.freenode.net and ask your questions there. We'll be glad to help you get your patches merged–but please remember that many of us are busy ops professionals, so we may not respond immediately.

If you'd like to join the mailing list, email avleen@gmail.com.

Rewards for contributions

We have a special reward for reaching either of the following goals with your contributions:

- 10 pull requests totaling 50 or more lines changed
- 1 recorded Ops School video

For completing either of these, you will receive a dozen freshly baked cookies, made fresh for you by a professional baker, and a 3-inch die-cut sticker of the Ops School logo.

Once you reach either of these, please fill in the OpsSchool rewards form and we will get your reward over to you!

Ops School Videos

In collaboration with O'Reilly Media, we are filming the Ops School curriculum to provide another method of learning for students. Filming happens approximately every 3 months in New York or Silicon Valley.

O'Reilly is publishing the videos online, in individual form and in package form. Any profits which would be made by Ops School are donated to non-profit organisations which aid in learning for low-income persons.

Video can be short (10-20 mins), or longer. Depending on the content in the video, some presenters choose to use slides, while others prefer screen-casting live demonstrations. Some videos may not need any supporting material at all!

The next scheduled filming is at the Santa Clara Hyatt, from Monday June 17th 2013 to Friday June 21st 2013. Filming slots are available all day.

The topics which have already been filmed are:

- Boot Process 101
- DNS 101
- Filesystems 101
- MySQL 101
- Nagios 101

- Networking 101
- Outage Investigation 101
- Package Management 101
- Shells 101
- SSH 101
- Application Level Monitoring
- Careers In Operations
- Operable Service - What People Expect From an Operable Service
- Productivity Tools and Techniques
- The `/proc` Filesystem
- Web Flow - The Life of a Web Request

If you are interested in filming, please contact Avleen Vig (avleen#gmail.com) or open an issue on the Github project.

How to write sections

In order to help students learn as much as possible, we are taking the following approach to the curriculum (this isn't a hard-and-fast rule, but an encouraged guideline wherever possible):

- Approach your writing in three steps:
  1. Tell students what they're going to learn
  2. Teach them what they need to know
  3. Tell them what they have learnt
- As much as possible, treat this as an interactive exercise. For example if you are writing about virtual machines, don't just write about virtual machines. Have the student create a virtual machine, and then explain what just happened. Don't tell students that package managers install packages, have them install a few packages and then explain how the package manager did its thing.

Please read the *Conventions* topic for some style guideline and preferences.

Overwriting existing content

There are times when you will want to update or replace sections of text written by others. When doing so, follow these guidelines to ensure your changes are integrated smoothly:

- Submit your pull request
- Reference the original author in your commit

We'll wait a few days (up to one week) to let the original author comment, if we feel there may be anything contentious in the commit. For most simple and straightforward commits, we'll simply accept the commit.

Credits

If you contribute to this document and would like your name in lights (or, at least, written down somewhere) please add it here along with an email address and any affiliation:

Name	Company
Avleen Vig <avleen@etsy.com>	Etsy, Inc
Patrick McDonnell	
Michael Rembetsy <mcr@etsy.com>	Etsy, Inc
Magnus Hedemark <magnus@yonderway.com>	Wireless Generation
Ariel Jolo <ajolo@sysarmy.com.ar>	sysARmy
Ryan Frantz <ryanleefrantz@gmail.com>	Crabby Admins (www.crabbyadmins.org)
Mike Fiedler <miketheman@gmail.com>	Datadog
Nathan Milford <nathan@milford.io>	Outbrain, Inc.
Patrick Cable <pc@pcable.net>	
Benjamin Krueger <benjamin@seattlefenix.net>	Sourcefire, Inc
Mårten Gustafson <marten.gustafson@gmail.com>	
Phil Hollenback <philiph@pobox.com>	
Adam Jacob <adam@opscode.com>	Opscode, Inc.
Mark Imbriaco <mark@github.com>	GitHub
James Turnbull <james@lovedthanlost.net>	Puppet Labs
Scott Nyce <snyce@codetwit.com>	TiVo, Inc.
Christian Paredes <cp@redbluemagenta.com>	Amazon
Jan Schaumann <jschauma@netmeister.org>	
Stephen Balukoff <sbalukoff@bluebox.net>	Blue Box Group
Evan Pettrey <jepettrey@gmail.com>	LOPSA
Khalid Maqsudi <khalid7621@hotmail.com>	Ashford.com
Paul Graydon <paul@paulgraydon.co.uk>	
Harold “Waldo” Grunenwald	
Martin Gehrke <martin@teamgehrke.com>	LOPSA
John Boris <jborissr@gmail.com>	LOPSA
John Dewey <john@dewey.ws>	AT&T
Carolyn Rowland <unpixie@gmail.com>	
Jordan Dea-Mattson <jdm@dea-mattson.com>	Numenta, Inc.
Sean Escriva <sean.escriva@gmail.com>	Heavy Water Ops
Adam Compton <comptona@gmail.com>	
Franck Cuny <franck@lumberjaph.net>	SAY Media
Chris Nehren <cnehren@omniti.com>	OmniTI
Brian Rak <dn@devicenull.org>	
Divij Rajkumar <drajkuma1@gmail.com>	
Aslan Brooke <aslandbrooke@yahoo.com>	ZynxHealth.com, InfoQ.com
Glen Kaukola <gkaukola@cs.ucr.edu>	
Spencer Krum <krum.spencer@gmail.com>	UTi Worldwide Inc.
Jeremy Grosser <jeremy@synack.me>	Uber Technologies, Inc.
Hugo Landau <hlandau@devever.net>	
Konark Modi <modi.konark@gmail.com>	MakeMyTrip.com
Josh Reichardt <josh.reichardt@gmail.com>	thepracticalsysadmin.com
Ben Reichert <ben@benreichert.com>	
Simon Aronsson <simon.aronsson@gmail.com>	itshale.com, simonaronsson.se
Andrew Langhorn <andrew@ajlanghorn.com>	
Abubakr-Sadik Nii Nai Davis <dwa2pac@gmail.com>	
Mike Julian	
Bram Verschueren	

Conventions

This set of documents uses a number of conventions throughout for ease of understanding. Contributors should be familiar with everything described here, and optimally contributions will be compared to what's here for uniformity.

Style Guide

See *Style Guide* for more.

Sample Network

A sample network makes explaining networking topics easier for both the teacher and the student. The teacher doesn't need to spend time creating an example network to explain a topic, and the student doesn't need to learn a new network for every section. To this end, the curriculum has adopted a default sample network. The goal is to construct a hypothetical network that resembles what one is likely to find in a well-constructed production environment. This means firewalls, load balancers, separation of concerns between different machine roles, and so on.

Topography

The network is hosted in a datacenter, with the subnet of 10.10.10.0 and a subnet mask of 255.255.255.0. This yields an IP address range of 10.10.10.0 through 10.10.10.255. In front of everything else on the network are two firewall machines, fw-1 and fw-2. These dual-homed machines have the LAN addresses 10.10.10.1 and 10.10.10.2, respectively. The network also has two DNS servers, dns-1 and dns-2, occupying 10.10.10.3 and 10.10.10.4. In between the firewalls and the rest of the LAN are two load balancers, lb-1 and lb-2. These machines share the rest of the public address space and have the internal LAN addresses 10.10.10.5 and 10.10.10.6.

Beyond this, there are two machines that function as jumphosts into the network (independent of any VPN or the like). These machines also host the source repositories, configuration management, and similar services. Their LAN addresses are 10.10.10.7 and 10.10.10.8, and their hostnames jump-1 and jump-2. In addition, there is an LDAP server, ldap-1, at 10.10.10.9.

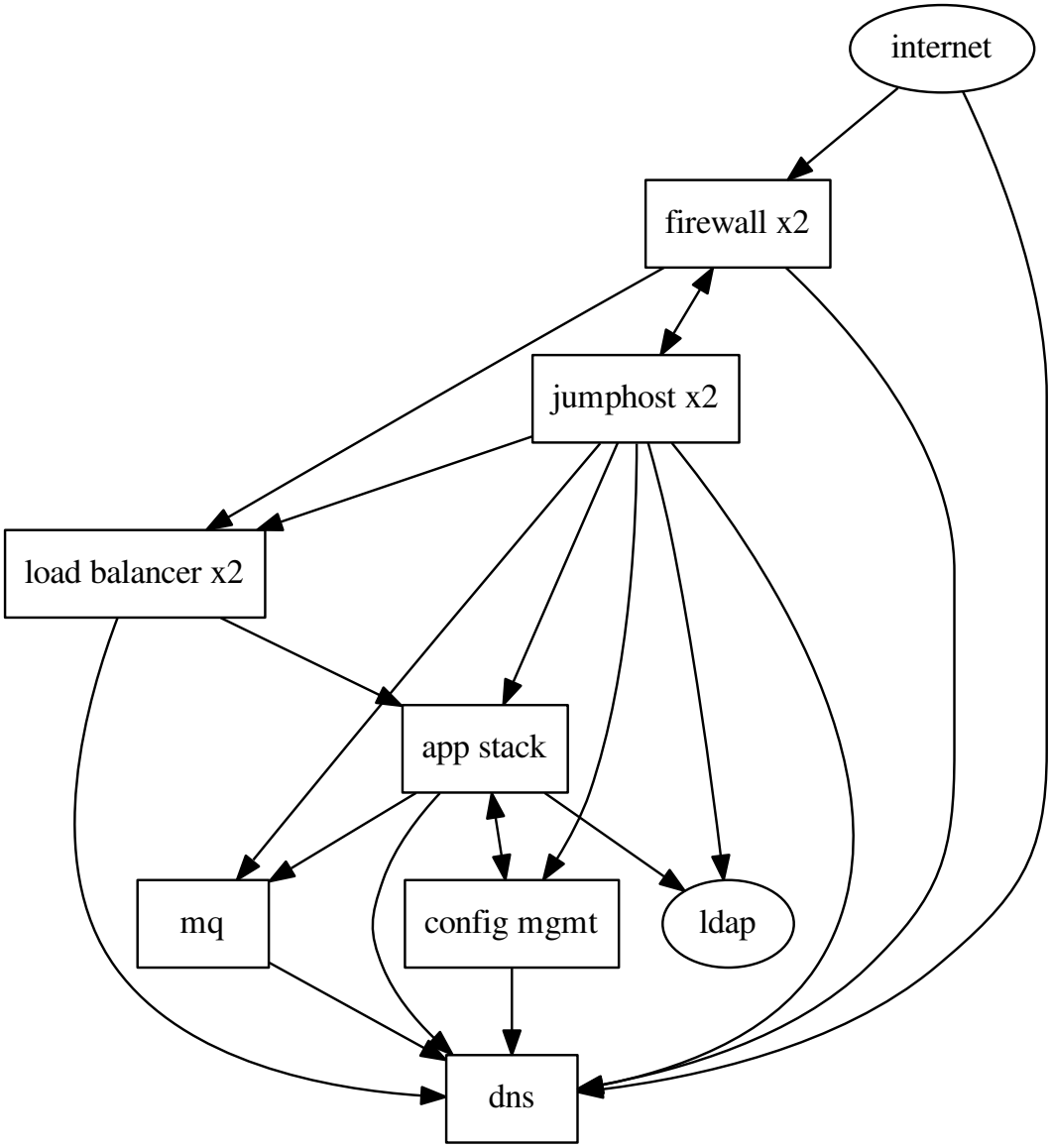
In terms of the application stack, there are four Web servers (10.10.10.10-10.10.10.13), two app servers (10.10.10.14-10.10.10.15), two database servers (10.10.10.16-10.10.10.17), two DNS servers (10.10.10.18-10.10.10.19), and a message queue (10.10.10.20). The Web server hostnames are web-1 through web-4; the app servers, app-1 and app-2; the databases, db-1 and db-2; the message queue, mq-1.

Todo

Mainly for @apeiron: Simplify the topography description, possibly with use of a table to describe IP assingments

---

This is a lot to conceptualize all at once; a diagram will elucidate the architecture.



Style Guide

When writing documentation, we should strive to keep the style used consistent, clear and understandable across all topics.

These documents are written in reStructuredText with Sphinx directives.

The style used in writing here should follow the guidelines in the following sections. This document attempts to conform to the guidelines themselves, and may be used as a reference. You may view this document's source by clicking "Show Source" on the left-hand menu bar.

Editing

This section attempts to guide the author on ways to edit the source documentation and preserve project style so many editors can collaborate efficiently. Guides on spacing, heading layout and topic structure will be established here.

Spacing

- Do not add two spaces after terminal punctuation, such as periods.

  The extra space will be ignored when rendered to HTML, so strive for consistency while editing.

- Remove trailing whitespace from your lines.

  Almost any modern text editor can do this automatically, you may have to set your editor to do this for you. Reasoning is that:

  1. it makes git unhappy
  2. there's no usefulness in those extra characters
  3. when others edit with whitespace-removing editors, their commit can get polluted with the removals instead of the content

  Basically, don't leave a mess for someone else to clean up.

- Start sentences on new lines.

  Start each new sentence of the same paragraph on the immediately following line. Historically, lines had been requested to be wrapped at 80 characters. In our modern day and age of powerful text editors this may no longer be very useful, rather asking the author to provide concise text is more valuable. If a sentence is very long, consider how you might rewrite it to convey the idea across multiple sentences. In general, try to consider how long the line would appear in a text editor at 1024 pixels wide, 12pt font (around 140 characters total).

- Sectional spacing should follow these guidelines:
  - 1 blank line after any section heading, before content starts
  - 2 blank lines after an H1/H2 section ends
  - 1 blank line after any other section ends

  When in doubt, leave one blank line.

- Leave one (1) blank line at the end of every file.

Showing Examples

Very frequently, we may want to show the reader an example of code or console output. Sphinx provides language-specific highlighting, so instead of using a standard literal block (`::`), use the more explicit `code-block::` syntax.

An example:

```
.. code-block:: ruby

require 'date'
today = Date.today
puts "Today's date is #{today}"
```

This would render so:

```
require 'date'
today = Date.today
puts "Today's date is #{today}"
```

This highlighted version is superior, and more easily understood.

Please use the language being displayed for the correct syntax. For examples of console output, use `console`. When showing pseudo-code or items that do not fit a language or console, use `none`.

When showing shell commands, plese use the following standard prompts for user and root:

```
user@opsschool ~$ uptime
 14:17:34 up 12:05, 16 users,  load average: 0.34, 0.16, 0.05

root@opsschool ~# uptime
 14:18:15 up 12:05, 16 users,  load average: 0.22, 0.15, 0.06
```

These prompts can be set with the following PS1 entries:

```
export PS1='user@opsschool \w$ '
export PS1='root@opsschool \w# '
```

Glossary

BIOS A set of computer instructions in firmware that control input and output operations.

CLI Command Line Interface

Domain Specific Language A special purpose programming language dedicated to a particular problem area, e.g. SQL is a domain specific language for retrieval of data from a database.

EPEL Common acronym for Extra Packages for Enterprise Linux. These are typically included in Fedora Linux, and provided for RedHat, CentOS, and other RPM-based distributions. The project's homepage is here: FedoraProject:EPEL.

GUI Pronounced "gooey", this is an acronym for a Graphical User Interface. This is distinctly different from a *CLI*, in that the GUI typically can contain more complex visual interactions. Read more here: Wikipedia:GUI.

MAC Pronounced as "mack" and often used as a noun referring to a network device's Media Access Controller (MAC) address. A MAC address is a globally unique number assigned to each interface in an Ethernet network and used to direct Ethernet frames between source and destination devices.

OSI Open Systems Interconnection (OSI) is an effort to develop standards-based computer networking protocols in order to allow networking equipment from different vendors to interoperate without relying on implementation of competing proprietary protocols. The OSI is best known for the development of the standard seven-layer OSI model for describing layers of abstraction into which the various networking protocols are categorized.

POSIX An acronym for "Portable Operating System Interface", is a family of standards specified by the IEEE for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems. Read more here: Wikipedia:POSIX.

RFC The RFC documents (Request for Comments) are a series of Internet standards, best common practices, and related documents describing how networked computers communicate. This document series provides the standards for how the Internet and many other related technologies interrelate and interoperate.

VLAN An acronym for "Virtual Local Area Network"; a single physical switch can be divided into multiple "virtual" switches via the configuration of VLANs, and assigning individual switchports into a given VLAN. Each VLAN represents a Ethernet broadcast domain (boundary); best practices dictate that a single IP (Layer 3) network be

mapped to a single VLAN. Packets must be routed between VLANs (either by an outboard router, or by an internal router in what is known as a "Layer 3 switch.")

CHAPTER 2

TODO List

Todo

restoring? what not to backup, regulations on same, how to store them (PCI)

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 74.)

Todo

The idea of such a risk may not be immediately clear to a beginning ops person.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 81.)

Todo

How does FOIA affect what information an organization needs to have available? Assume the reader is a civilian and doesn't know how FOIA affects an organization.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 98.)

Todo

media – should someone address the state of backup media? Some places are still doing tape. What about orgs who rely on standalone consumer-grade disks for client backups (e.g. Time Machine)? Risks, cost to maintain.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 131.)

Todo

Using backups to restore to a known "good" state after an incident just serves to put the machine in a known vulnerable state (security hole that was exploited is now back in operation)

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 159.)

Todo

can be used to restore system state that can be useful in a post mortem after an incident (say the attacker covered their tracks but backups were able to capture a rootkit before it was removed or before logs were tampered with)

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/backups.rst, line 161.)

Todo

Check this section. I think i've got it down, but I'm not super familiar with this part.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/boot_process_101.rst, line 323.)

Todo

a specific example of convergent over time might help

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/config_management.rst, line 31.)

Todo

shared resources, bussiness needs.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 109.)

Todo

How to create a plan from the material we gathered in the planning phase.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 114.)

Todo

Pros and cons on separating the disaster recovery manual from the technical recovery manual.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 115.)

Todo

Strategies when simulating. Defining testing scopes. Measuring.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 120.)

Todo

Limiting the scope to core business

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 124.)

Todo

Expanding the scope in the disaster recovery environment vs. going back to production before expanding

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 125.)

Todo

Communication

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/disaster_recovery.rst, line 129.)

Todo

Explain "What is Ops School?"

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/introduction.rst, line 5.)

Todo

Mainly for @apeiron: Simplify the topography description, possibly with use of a table to describe IP assingments

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/meta/conventions.rst, line 60.)

Todo

Mention spec files and roughly how RPMs are put together.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/package_management_101.rst, line 165.)

Todo

Then introduce FPM and tell them not to bother with spec files yet.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/package_management_101.rst, line 166.)

Todo

Explain more about what rubygems are as well as http://rubygems.org

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/programming_201.rst, line 318.)

Todo

Discuss how authentication works. Touch on `/etc/(passwd|group|shadow)`, hashing. What are groups? Lead in to the users/groups permissions model and how permissions are based on the user/group/other bits.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/security_101.rst, line 7.)

Todo

What is PKI? What uses it? Why is it important?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/security_201.rst, line 106.)

Todo

stat command

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shell_tools_101.rst, line 247.)

Todo

vmstat command

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shell_tools_101.rst, line 252.)

Todo

strace command

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shell_tools_101.rst, line 257.)

Todo

ulimit command

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shell_tools_101.rst, line 262.)

Todo

Only talk about replacing text for now? It's the most common / needed piece of sed at this level.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shell_tools_101.rst, line 619.)

Todo

Tighten up the above sentence. It's wordy and doesn't seem to make the point I want it to make.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/shells_101.rst, line 30.)

Todo

It might be interesting to put together an exercise where the student interacts with a fictional customer in some different scenarios. Depending on what the student does, the customer is happy or complains to the operations person or escalates the complaint up the management chain. How does the student respond? Could have multiple scenarios with different customers (a customer who causes his own problem then gets in the way, a customer who cannot wait, a customer who tries to fix the problem and makes it worse, a customer who uses the opportunity to speak to an operations person to dump 10 other requests on that person. This idea came to me from a series of books my kid has where you make a decision on page 10 that leads to to either page 26 or page 40. Your decision could end the story

or take you in a new direction. The books are full of these decision points so the story is rarely the same twice, kinda like customer support!

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/soft_skills_101.rst, line 355.)

Todo

does this section need a real writeup or are references to Tom's work enough?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/soft_skills_101.rst, line 375.)

Todo

Might give an example here. Need to write more explaining how to assemble the pieces.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/soft_skills_201.rst, line 433.)

Todo

write this section.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/soft_skills_201.rst, line 487.)

Todo

"What is Development" Section needs more developer perspective.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ops-school/checkouts/latest/sysadmin_101.rst, line 211.)

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

[Byobu] http://byobu.co/

[Bok] http://sardes.inrialpes.fr/papers/files/Bouchenak08a.pdf

[0MQ] https://en.wikipedia.org/wiki/%C3%98MQ

[wiki] https://en.wikipedia.org/wiki/AMQP

[specs] http://www.amqp.org/resources/download