Linux and Shell Programming with Bash

Release #9a25cc0, 2018-12-31

Matt Harasymczuk

2018-12-31

Introduction

Author

name Matt Harasymczuk

email matt@astrotech.io

www http://www.astrotech.io

facebook https://facebook.com/matt.harasymczuk

linkedin https://linkedin.com/in/mattharasymczuk

slideshare https://www.slideshare.net/astrotech/presentations

github https://github.com/astromatt

Tip: The most up-to-date version of this book is always at http://linux.astrotech.io

Other books from this series

Python and Machine Learning http://python.astrotech.io

DevOps and Development Tools Ecosystem http://devops.astrotech.io

GIT and GIT Flow in CI/CD http://git.astrotech.io

Agile, Scrum, Kanban, XP, Lean http://agile.astrotech.io

IT Software Architecture, Cloud, Microservices and Processes http://arch.astrotech.io

Linux and Shell Programming with Bash http://linux.astrotech.io

CHAPTER 1

Introduction

1.1 About this book

1.1.1 Zapotrzebowanie uczestnika

- umieć stworzyć backlog i wiedzieć jak priorytetyzować zadania dla zespołu
- rozumieć estymacje zespołu
- znać zasady Scrum dotyczące tworzenia i utrzymywania produktów
- rozumieć różnicę między Project Managerem a Product Ownerem
- umieć połączyć rozwój oprogramowania z utrzymaniem
- wiedzieć jak pracować w kilka zespołów nad jednym produktem
- móc szybko i precyzyjnie szacować projekty dla klientów zarówno wewnętrznych jak i zewnętrznych
- zarządzać funkcjonalnościami produktu
- umieć określić hipotezę przydatności funkcjonalności i ją potwierdzić na podstawie danych z testów
- jak tworzyć i czytać wykresy: Burndown Chart, Velocity Chart, Version Report, Epic Report, Cumulative Flow Diagram, Control Chart
- wiedzieć jak tworzyć Kryteria Akceptacyjne i jak wypracować Definicję Ukończenia (Definition of Done)

1.1.2 Tematyka szkolenia

Obszar procesowy

- Scrum jako ramy tworzenia produktu
- Projekt a Produkt
- Fundamenty Scrum i główne zasady
- Multidyscyplinarne i samo-organizujące się zespoły
- Łączenie rozwoju i utrzymania oprogramowania
- Czym różnią się Epic, User Story, Task, Requirement

- Cykl życia aplikacji, podejście SDLC (Waterfall i Scrum)
- Praca wielu zespołów nad jednym produktem
- Jak wykrywać marnotrawstwa i zastosować technikę Continuous Improvement

Obszar wartości biznesowych

- Zwiększanie wartości dla klienta
- Zarządzanie backlogiem produktu
- Szacowanie backlogu, określanie priorytetów
- Praktyki i technologie wspierające dostarczanie wartości biznesowych (wprowadzenie)
- Tworzenie i czytanie wykresów: Burndown Chart, Velocity Chart, Version Report, Epic Report, Cumulative Flow Diagram, Control Chart
- Elementy Lean Startup dla Product Ownerów, tj. pętla Build - Measure - Learn

Warsztat na prawdziwym produkcie

- Rozbicie na epiki i podział na User Stories, Tasks, Requirements
- Trzy iteracje refinementu, dekompozycji i estymacji
- Określanie Kryteriów Akceptacyjnych
- Określenie pracochłonności, wartości biznesowej, priorytetów MoSCoW (i dlaczego to ma sens)
- Rozplanowanie sprintów z zakresem produktu
- Wykorzystanie systemów elektronicznych wspierających proces
- Wykorzystanie wersji i release stream

1.2 Agenda

1.2.1 Agile Bootcamp

Tab.1.1.: Agile Bootcamp day 1 agenda

Time	Title	Agenda
09:00-12:00	Introduction	1. What is Linux and why? 2. Unix family tree 3. Linux family tree 4. Which distribution
12:00-13:00	Lunch	
13:00-17:00	Bash programming workshop	1. Variables 2. Scopes 3. Files

Tab.1.2.: Agile Bootcamp day 2 agenda

Time	Title	Agenda
09:00-12:00	Introduction	1. What is Linux and why? 2. Unix family tree 3. Linux family tree 4. Which distribution 5. Short discussion
12:00-13:00	Lunch	
13:00-17:00	Bash programming workshop	1. Variables 2. Scopes 3. Files

1.3 VIM

1.3.1 Opening files to edit

1.3.2 Writing files

CHAPTER 2

Linux

2.1 Directory Structure

Fig.2.1.: Linux directory tree

Tab.2.1.: Directory Structure

Path	Description
/	Main directory
/bin	Buil-in executable files
/boot	Boot files and kernel
/etc	Configuration directory
/etc/init.d	Runtime scripts
/dev	Devices and drivers
/home	User files
/lib	Shared libraries
/opt	Optional applications
/root	Superuser home directory
/sbin	Superuser built-in binary files
/srv	Optional services
/tmp	Temporary files (removed on startup)
/usr	User installed files
/usr/bin	Application executable files
/usr/lib	Applications data files
/usr/local/bin	User installed applications executable files
/usr/local/sbin	Superuser installed applications executable files
/usr/sbin	Application superuser executable files
/usr/src	Application source codes
/var	Installed applications files
/var/lock	Application lock files
/var/log	Applications and system log files
/var/pid	Application PID files
/var/spool	System spool files (crontab, mail, printer)

2.2 Basic Commands

Tab.2.2.: Buit-in commands

Command	Description
ls	List
cd	Change Directory
cat	Displays file
cp	Copy
mv	Move
rm	Remove
man	Manual Pages
clear	Clears terminal
pwd	Shows Present Working Directory
env	Show all environmental variables
echo	Displays text
tail	Last -n lines from file
head	First -n files from file
grep	Regual Expressions tool (parses input for regexp)
crontab	Automatic tasks
sudo	Switch user and execute command
apt install	installs application (on Debian based systems)
apt search	searches for application (on Debian based systems)
history	Last executed commands
sed	Stream Editor
awk	Parses lines
uniq	Remove duplicated lines
sort	Sorts input
wc	Counts characters and lines
export	Set environment variable
chown	Change Owner
chmod	Change Permissions (mods)
du	Disk Usage
df	Disk Free (space)
file	Show file type and metadata
whoami	Shows user login
which	Shows path to executable
find	Finds file in the filesystem
locate	Locates file (from updatedb database)
updatedb	Scans filesystem and create database for locate
dmesg	Debugging Messages
locale	Localization
touch	Creates empty file
alias	Creates user defined alias
mc	Midnight Commander
su	Switch user
rsync	Syncronizes two directories
ssh	Secure Shell Connection

2.2.1 `cd`

- `cd ~`
- `cd -`
- `cd`

- `cd ..`

2.2.2 `ls`

- `ls -lh`
- `alias l='ls -lAh --color=auto'`

2.3 Environmental Variables

- `/usr/bin/env`
- `/etc/environment`

Tab.2.3.: Environmental Variables

Name	Description
PWD	Present Working Directory
UID	User ID
HOME	User Home Directory
PATH	Executable Search Path
SHELL	Current Shell
TERM	Current Terminal (character mapping)
PS1	Prompt
LANG	System Language
HOSTNAME	Hostname
IFS	Inter Field Separator
UMASK	Permission mask for new files

2.4 Users and groups

2.4.1 Files

- /etc/passwd
- /etc/shadow
- /etc/group

2.4.2 whoami

2.4.3 UID

2.4.4 HOME

2.4.5 useradd vs. adduser

2.5 Permissions

2.5.1 Understanding Permissions

Tab.2.4.: Understanding Permissions

Permission	Octal	Binary	Description
—	0	000	Cannot read, execute or modify
–x	1	001	Can execute
-w-	2	010	Can write (modify)
-wx	3	011	Can modify and execute
r–	4	100	Can read
r-x	5	101	Can read and execute
rw-	6	110	Can read and write
rwx	7	111	Can read, write and execute

2.5.2 Changing Permissions

chmod

chown

chgrp

2.5.3 UMASK

2.5.4 Sticky bit

2.5.5 ACL

2.6 SSH

2.6.1 Connecting

2.6.2 Private Key

- `~/.id_rsa`
- `~/.id_rsa.pub`

2.6.3 Authorized Keys

2.6.4 Known Hosts

2.6.5 Port Forwarding

2.6.6 Reverse Tunnel

-L

-R

2.6.7 Config and host aliases

2.6.8 SSHd

Disabling password authentication

2.7 Crontab

```
$ crontab -e
$ crontab -l
$ sudo crontab -e
```

2.7.1 Przykładowy crontab

```
5 4 * * *       /bin/echo 'five past four a.m.'
*/10 * * * *    /bin/echo 'every ten minutes'
5-10 4 * * *    /bin/echo 'every minute from 5-10 past four a.m.'
* 4 * * *       /bin/echo 'every minute at 4 a.m.'
0 14 * * *      /bin/echo 'at 2 p.m.'
0 0 1 * *       /bin/echo 'at midnight of first day of month'
0 0 1 JAN *     /bin/echo 'at midnight of first day of January'
0 0 1 1 *       /bin/echo 'at midnight of first day of January'
0 0 * * SAT,SUN /bin/echo 'at midnight on weekends'
0 0 * * 0,6     /bin/echo 'at midnight on weekends'

@daily          /bin/echo 'at midnight'
@weekly         /bin/echo 'at midnight on Sunday'

45 04 * * * /usr/bin/updatedb
45 04 * * * /usr/sbin/chkrootkit && /usr/bin/updatedb
00 06 * * * env DISPLAY=:0.0 gui_appname
00 01 * * * ubuntu /home/ubuntu/script.sh
```

2.7.2 Editing `crontab`

```
export EDITOR=/usr/bin/vim
```

Variables

```
PATH=/usr/sbin:/usr/bin:/sbin:/bin
```

Special characters

- * any value
- , value list separator
- – range of values
- / step values

Crontab formatting

- minute: 0-60
- hour: 0-23
- day of month: 0-31
- month: JAN-DEC / 0-12
- day of week: SUN-SAT / 0-7 (Sunday = 0 or 7)

Short notation

Tab.2.5.: Short notation

Notation	Meaning
@yearly	Run once a year, `0 0 1 1 *`
@annually	Same as @yearly
@monthly	Run once a month `0 0 1 * *`
@weekly	Run once a week `0 0 * * 0`
@daily	Run once a day `0 0 * * *`
@midnight	Same as @daily
@hourly	Run once an hour `0 * * * *`
@reboot	Run once, at startup

2.7.3 Allowing/Denying User-Level Cron

- /etc/cron.allow
- /etc/cron.deny

2.7.4 Files and Directories

- /etc/crontab
- /var/spool/crontab/
- /etc/cron.d/
- /etc/cron.daily/
- /etc/cron.hourly/
- /etc/cron.weekly/
- /etc/cron.monthly/

2.7.5 Other

- z jakiego użytkownika są uruchamiane
- przekierowanie outputu stdout i stderr
- dostawanie maili

2.8 Logs

2.8.1 `dmesg`

2.8.2 `/var/log`

2.8.3 `/var/log/syslog`

2.8.4 `/var/log/messages`

2.9 Filesystem

2.9.1 Symlinks

2.9.2 File types

- no extension
- .filenames (starting with dot)
- `file`

2.9.3 Size

`du -h` `df -h`

2.9.4 Disk partitioning

parted

gparted

druid

2.9.5 Checking integrity

`fdisk`

2.9.6 Mounting devices

`mount`

Devices

- /dev/

Mount points

- /etc/fstab
- /etc/mtab

Filesystems

2.10 Booting

2.10.1 LiveCD

RamFS

2.10.2 GRUB

Kernel

Initramfs

Splashscreen

Multiple OSes

Hard disk naming convention

2.10.3 Services and Daemons

/etc/rc.d

/etc/init.d/

Systemd

System-V

Init-d

```
service (start | stop)
```

```
servicectl (start | stop)
```

2.11 Devices

/dev/sda /dev/sda1 /dev/sdb1

/dev/random /dev/urandom

2.12 Networking

- /etc/hosts localhost
- 127.0.0.1

- ::1
- /etc/hosts
- /etc/resolv.conf
- /etc/network/interfaces
- /etc/if-up-down/

2.12.1 Built-in

```
ifconfig
```

```
ip
```

```
route
```

```
netstat
```

```
iptables
```

2.12.2 Additional

```
nc
```

```
wireshark
```

```
nmap
```

```
tcpdump
```

2.13 Processes

2.13.1 Spawning - &

2.13.2 Listing

```
ps aux
```

```
lsof
```

```
top
```

```
htop
```

2.13.3 PID

PID files

```
pidof
```

```
/var/spool/pid
```

2.13.4 Locks

2.13.5 Killing

```
kill
```

```
kill -9
```

```
killall
```

```
Ctrl-c
```

```
Ctrl-d
```


2.13.6 Priorities

- `xdm`
- `kdm`
- `gdm`

2.14.4 Desktop Environment

- `gnome`
- `kde`
- `fluxbox`
- `fvwm`
- `xfce`

CHAPTER 3

Bash

3.1 Interpreter

3.1.1 Configuration files

- ~/.profile
- ~/.bashrc
- ~/.bash_logout
- /etc/bashrc

3.1.2 Locale

- `$LANG`
- /etc/locale

3.1.3 Autocompletion

3.1.4 New lines

- "\n"

3.1.5 `#!/bin/bash`

- A.K.A shebang or hashbang
- Interpretes script as `/bin/bash` source code

3.1.6 `bash -x`

- shows execution steps

3.1.7 Comments

- # at the beginning of the line

3.1.8 Inline comments

- # in the middle of the line

3.2 Variables

3.2.1 single quotes

```
$ name='José Jiménez'
$ echo 'My name is $name'
My name is $name
```

3.2.2 double quotes

```
$ name="José Jiménez"
$ echo "$name"
My name is José Jiménez
```

3.2.3 Script arguments

$0 - Script name `$1...` ‘‘$9‘‘ - positional parameter number 1 to 9 `$@` - all parameters

3.3 Arrays

3.3.1 Declaration

`ARRAY=()` Declares an indexed array ARRAY and initializes it to be empty. This can also be used to empty an existing array.

`ARRAY[0]=` Generally sets the first element of an indexed array. If no array ARRAY existed before, it is created.

`declare -a ARRAY` Declares an indexed array ARRAY. An existing array is not initialized. `declare -A ARRAY` Declares an associative array ARRAY. This is the one and only way to create associative arrays.

3.3.2 Storing values

`ARRAY[N]=VALUE` Sets the element N of the indexed array ARRAY to VALUE. N can be any valid arithmetic expression.

`ARRAY[STRING]=VALUE` Sets the element indexed by STRING of the associative array ARRAY.

`ARRAY=VALUE` As above. If no index is given, as a default the zeroth element is set to VALUE. Careful, this is even true of associative arrays - there is no error if no key is specified, and the value is assigned to string index “0”.

`ARRAY=(E1 E2 ...)` Compound array assignment - sets the whole array ARRAY to the given list of elements indexed sequentially starting at zero. The array is unset before assignment unless the += operator is used. When

the list is empty (`ARRAY=()`), the array will be set to an empty array. This method obviously does not use explicit indexes. An associative array can not be set like that! Clearing an associative array using `ARRAY=()` works.

`ARRAY=([X]=E1 [Y]=E2 ...)` Compound assignment for indexed arrays with index-value pairs declared individually (here for example X and Y). X and Y are arithmetic expressions. This syntax can be combined with the above - elements declared without an explicitly specified index are assigned sequetially starting at either the last element with an explicit index, or zero.

`ARRAY=([S1]=E1 [S2]=E2 ...)` Individual mass-setting for associative arrays. The named indexes (here: S1 and S2) are strings.

`ARRAY+=(E1 E2 ...)` Append to ARRAY. `ARRAY=("${ANOTHER_ARRAY[@]}")` Copy ANOTHER_ARRAY to ARRAY, copying each element.

3.3.3 Getting values

`${ARRAY[N]}` Expands to the value of the index N in the indexed array ARRAY. If N is a negative number, it's treated as the offset from the maximum assigned index (can't be used for assignment) - 1

`${ARRAY[S]}` Expands to the value of the index S in the associative array ARRAY.

`"${ARRAY[@]}" ${ARRAY[@]}` `"${ARRAY[*]}" ${ARRAY[*]}` Similar to mass-expanding positional parameters, this expands to all elements. If unquoted, both subscripts * and @ expand to the same result, if quoted, @ expands to all elements individually quoted, * expands to all elements quoted as a whole.

`"${ARRAY[@]:N:M}" ${ARRAY[@]:N:M}` `"${ARRAY[*]:N:M}" ${ARRAY[*]:N:M}` Similar to what this syntax does for the characters of a single string when doing substring expansion, this expands to M elements starting with element N. This way you can mass-expand individual indexes. The rules for quoting and the subscripts * and @ are the same as above for the other mass-expansions.

3.3.4 Metadata

`${#ARRAY[N]}` Expands to the length of an individual array member at index N (stringlength)

`${#ARRAY[STRING]}` Expands to the length of an individual associative array member at index STRING (stringlength)

`${#ARRAY[@]}` `${#ARRAY[*]}` Expands to the number of elements in ARRAY

`${!ARRAY[@]}` `${!ARRAY[*]}` Expands to the indexes in ARRAY since BASH 3.0

3.3.5 Destruction

`unset -v ARRAY` `unset -v ARRAY[@]` `unset -v ARRAY[*]` Destroys a complete array `unset -v ARRAY[N]` Destroys the array element at index N `unset -v ARRAY[STRING]` Destroys the array element of the associative array at index STRING

3.4 Conditionals

3.4.1 `if`

```
name="José Jiménez"

if [ $imie == "José Jiménez" ]; then
    echo "My name José Jiménez"
fi
```

3.4.2 `if` and `else`

```
name="José Jiménez"

if [ $imie == "José Jiménez" ]; then
    echo "My name José Jiménez"
else
    echo "I am someone else"
fi
```

3.4.3 Short version - `&&` and `||`

```
$ name="José Jiménez"
$ [ $imie == "José Jiménez" ] && echo "My name José Jiménez" || echo "I am someone␣
↪else"
My name José Jiménez
```

3.4.4 Case (A.K.A. switch)

```
case $( arch ) in   # $( arch ) returns machine architecture.
                    # Equivalent to 'uname -m' ...
  i386 ) echo "80386-based machine";;
  i486 ) echo "80486-based machine";;
  i586 ) echo "Pentium-based machine";;
  i686 ) echo "Pentium2+-based machine";;
  *    ) echo "Other type of machine";;
esac

exit 0
```

```
echo; echo "Hit a key, then hit return."
read Keypress

case "$Keypress" in
  [[:lower:]]   ) echo "Lowercase letter";;
  [[:upper:]]   ) echo "Uppercase letter";;
  [0-9]         ) echo "Digit";;
  *             ) echo "Punctuation, whitespace, or other";;
esac      #  Allows ranges of characters in [square brackets],
          #+ or POSIX ranges in [[double square brackets.
```

3.5 Loops

3.5.1 For

```
for i in `seq 1 10`; do
    echo $i
done
```

```
for i in $( ls ); do
    echo item: $i
done
```

Warning: IFS='\n'

Inline for

- `for a in *; do echo $a; done`

3.5.2 While

```
COUNTER=0

while [  $COUNTER -lt 10 ]; do
    echo The counter is $COUNTER
    let COUNTER=COUNTER+1
done
```

```
while [ $# -gt 0 ]; do    # Until you run out of parameters . . .
  case "$1" in
    -d|--debug)
          # "-d" or "--debug" parameter?
          DEBUG=1
          ;;
    -c|--conf)
          CONFFILE="$2"
          shift
          if [ ! -f $CONFFILE ]; then
            echo "Error: Supplied file doesn't exist!"
            exit $E_CONFFILE     # File not found error.
          fi
          ;;
  esac
  shift   # Check next set of parameters.
done
```

3.5.3 Until

```
COUNTER=20

until [  $COUNTER -lt 10 ]; do
    echo COUNTER $COUNTER
    let COUNTER-=1
done
```

3.6 Pipe

3.6.1 Pipe - `|`

3.6.2 `awk`

3.6.3 `sed`

3.6.4 `sort`

3.6.5 `uniq`

3.7 Stdout and Stderr

3.7.1 `>` and `>>`

3.7.2 `<` and `<<`

3.7.3 `1 > /dev/null`

3.7.4 `2 > &1`

3.8 Network

3.8.1 `wget`

3.8.2 `curl`

3.9 Regular Expressions

3.9.1 Grep

3.9.2 Egrep

3.10 Parameter expansion

3.10.1 Simple usage

- `$PARAMETER`
- `${PARAMETER}`

3.10.2 Indirection

- `${!PARAMETER}`

3.10.3 Case modification

- `${PARAMETER^}`
- `${PARAMETER^^}`
- `${PARAMETER,}`
- `${PARAMETER,,}`
- `${PARAMETER~}`
- `${PARAMETER~~}`

3.10.4 Variable name expansion

- `${!PREFIX*}`
- `${!PREFIX@}`

3.10.5 Substring removal (also for filename manipulation!)

- `${PARAMETER#PATTERN}`
- `${PARAMETER##PATTERN}`
- `${PARAMETER%PATTERN}`
- `${PARAMETER%%PATTERN}`

3.10.6 Search and replace

- `${PARAMETER/PATTERN/STRING}`
- `${PARAMETER//PATTERN/STRING}`
- `${PARAMETER/PATTERN}`
- `${PARAMETER//PATTERN}`

3.10.7 String length

- `${#PARAMETER}`

3.10.8 Substring expansion

- `${PARAMETER:OFFSET}`
- `${PARAMETER:OFFSET:LENGTH}`

3.10.9 Use a default value

- `${PARAMETER:-WORD}`
- `${PARAMETER-WORD}`

3.10.10 Assign a default value

- `${PARAMETER:=WORD}`
- `${PARAMETER=WORD}`

3.10.11 Use an alternate value

- `${PARAMETER:+WORD}`
- `${PARAMETER+WORD}`

3.10.12 Display error if null or unset

- `${PARAMETER:?WORD}`
- `${PARAMETER?WORD}`

3.11 Multiprocessing

3.11.1 Process - `...&`

3.11.2 Subprocess - `$(...)`

3.11.3 Return codes from last command `$?`

CHAPTER 4

Appendices

4.1 Glossary

4.2 Copyright

4.2.1 MIT License