treebuilder Documentation

Release 0.1

nate skulic

May 12, 2012

CONTENTS

CHAPTER ONE

WELCOME!

Treebuilder is a tool designed for rapid development of system images.

In addition to building images, live or otherwise, it can be used as a type of distributed installation tool (such as kickstart).

NOTE : Api/names might change. Still working out the verbage.

PDF Version of this documentation: http://media.readthedocs.org/pdf/treebuilder/latest/treebuilder.pdf

Contents:

1.1 Treebuilder Features

Build live images, install/upgrade systems over the network or via disk/stick and define configuration all from one tool.

- Lightweight text-mode installer included
- Installer capable of retrieving configuration and packages from a network/server
- Build live installer images (iso)
- Custom repository integration - Integrate your own repositories into the installer or installed system
- Full machine description:
  - disk: partitioning, raid
  - package installation
  - firewall
  - authentication (pam/ssh)
  - network devices - specify match expressions to configure network devices
  - general machine configuration including: hostname, timezone, selinux, users/password and keyboad
- Hooks for scripting machine configuration pre and post processing
- Grub2 and latest boot support

1.2 Installing treebuilder

1.2.1 Requirements

- PyYAML
- reparted

- progressbar == 2.2
- distutils/setuputils

Treebuilder also requires some dependencies not found in the python package index, including:

- libuser-python
- dbus-python
- yum

1.2.2 Releases

Stable releases will be available as rpm packages shortly. Pushing the tool into other distros is being considered. Fedora: maintainer/sponsor needed.

1.2.3 Using pip/easy_install

A common way to install python packages is using pip (or the outdated easy_install):

```
easy_install https://bitbucket.org/nateskulic/treebuilder/get/tip.tar.gz
```

```
pip install https://bitbucket.org/nateskulic/treebuilder/get/tip.tar.gz
```

1.2.4 From source

Treebuilder can be obtained from https://bitbucket.org/nateskulic/treebuilder:

```
hg clone https://bitbucket.org/nateskulic/treebuilder
```

Using a virtualenv

As many of the packages treebuilder depends on are not available through pypi but rather through distribution specific packages, make sure to use the –system-site-packages with virtualenv to include these packages. And example of a proper virtualenv command is below:

```
virtualenv .virtual --distribute --system-site-packages
```

Enter the virtual environment:

```
source .virtual/bin/active
```

Installing

Like any standard python package, distutils and setup.py can be used to install treebuilder. There are numerous way to setup and distribute package using setupyools/distutils and it is recommend that you consult their documentation.

For example, the following command will install a link into your .virtual/lib/pythonx.x/site-packages/treebuilder to the you have checked out treebuilder:

```
./setup.py develop
```

1.3 Using treebuilder

1.3.1 General usage

You must first define a configuration file. Some example configurations are in the examples directory.

To run the commands use:

```
sudo ./buildtree.py [path to yaml]
```

```
sudo ./buildtree.py examples/installer/iso.tree
```

1.3.2 Usage

usage: buildtree.py [-h] [-r ROOT] [-l LOGFILE] [-s SKIP] [-b BUILD] [–list-tools] [–list-commands] [-d] [-v] [-w] config_file

positional arguments: config_file Yaml configuration file.

optional arguments:

-h, --help show this help message and exit

-r ROOT, --root ROOT Override the install_root configuration.

-l LOGFILE, --logfile LOGFILE Output to a logging file.

-s SKIP, --skip SKIP Skip to a command. Valid commands: install.RunPreScripts, install.ZeroMBR, install.CreatePartitions, install.CreateRaid, install.FormatPartitions, yum.Prepare, yum.Install, yum.RunTxn, yum.RemovePackages, setup.Fstab, setup.RootPassword, setup.KeyboardLayout, setup.Authconfig, setup.SELinux, setup.TimeZone, setup.Network, setup.Firewall, setup.Hostname, setup.SysLog, installer.EmbedInstaller, installer.InstallTreebuilderService, installer.EmbedTree, install.RunPostScripts, yum.Mirror, yum.CreateRepo, installer.BuildInstallerImage, install.InstallBootloader, install.RaidPostInstall, install.Cleanup

-b BUILD, --build BUILD Build directory

--list-tools List tools and their help.

--list-commands List commands and their help.

-d, --debug Debug mode. Outputs heavier logging.

-v, --verbose Verbose/debug ui mode. Allow verbose debug message to pass up to the ui.

-w, --wait Dont exit right away. Wait for a input/keypress.

1.3.3 Included trees

- Treebuilder installer
- Fedora - minimal, gnome (possibly cinnamon) desktop, live desktop/rescue image (ALL COMING SOON. TESTING/HELP APPRECIATED).

treebuilders commands can depend on one another.

1.3.4 Building trees

1.3.5 Executing commands directly

-e

1.3.6 Jumping to commands and Modifying the sequence

-j -s

1.3.7 Creating a treebuilder installer

Create an installer image

First you must generate an install.iso:

```
sudo ./buildtree.py examples/installer/iso.tree
```

The iso.tree specifies that it should embed examples/minimal.tree and its required packages into the iso.

An install.iso should be generated in the current directory after running this command.

Run the installer image

Booting this iso will begin the installation as specified in examples/installer/iso.tree (and the included examples/minimal.tree).

An alternate tree file can be specified at kernel boot by specifying the kernel option tree.config as an absolute url to a tree file. This allows tree files (and everything they require) to reside in a remote location.

If testing load the output iso into virt-manage or qemu.

Treebuilder developers use qemu for testing treebuilder.

First create a disk image (or two+ if testing raid); noted as disk1.img in the following:

```
qemu-img create disk1.img 10G
```

An approriate command to test an image would be:

```
qemu-system-x86_64 -cdrom install.so -hda disk1.img -append "console=ttyS0 console=tty0 debug" -n
```

Or if testing the kernel/initrd directly:

> qemu-system-x86_64 -kernel isoroot/isolinux/vmlinuz -initrd isoroot/isolinux/initrd.img -append “console=ttyS0 console=tty0 debug” -nographic -m 700

The above command will output the kernel message to your console.

A *killall qemu-system-x86_64* might be required due to the console takeover.

Real testing should happen on real machines.

If you just want to update the tree included with the iso, and rebuild the iso then use:

> buildtree.py path/to/tree.tree -v -e treebuilder.EmbedTree -e treebuilder.Image

Structure of a Treebuilder installer iso

- isolinux/isolinux.cfg - Syslinux configuration file.
- isolinux/initrd.img - Treebuilder root initramfs image.
- isolinux/vmlinuz - Matching kernel for treebuilder root.
- Packages - A rpm repository of packages to install.
- Packages/repodata - A directory containing the repository metadata.
- install.tree - A tree included with iso. This tree will be installed when the iso is booted.

1.3.8 TODO: Create live images

- Live/rescue disks

1.3.9 Notes

Note: a root environment is required due to the way yum/rpm install packages. This might be fixed later. Overriding the system.chroot tool to use mock instead of an actual chroot; and hacking through rpm/yum should fix this. possibly needed is user mounting capability.

1.4 Configuration File

1.4.1 Description

Treebuilder *.tree files have the following properties:

- Are standard yaml files.
- Are in a mostly declarative syntax.
- Are CASE SENSITIVE.
- Can refer to other files and paths relative to the tree file or in another remote location.
- Can "inherit" from other tree files to allow for less duplication and manageable configurations.

It is recommended that one uses two space indentation for tree/yaml files so that everything lines up easily.

1.4.2 Examples

The default base treebuilder installer yaml:

```
# A Base install tree
# Contains a configuration suitable for working with treebuilder
install_root: /mnt/sysroot
locales: ['en_US','en']
yum:
  prepare:
    release: 16
    architecture: x86_64
    repos:
    - name: Fedorareleasever-basearch
      #url: "http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everything/$
      #type: url
      url: "https://mirrors.fedoraproject.org/metalink?repo=fedora-$releasever&arch=$basearch"
      type: metalink
    - name: Fedorareleasever-basearch-Updates
      #url: "http://download.fedoraproject.org/pub/fedora/linux/updates/$releasever/$basearch/"
      #type: url
      url: "https://mirrors.fedoraproject.org/metalink?repo=updates-released-f$releasever&arch=$ba
      type: metalink

## AT THIS POINT EVERYTHING IS MOUNTED UNDER install_root
setup:
  root_password: root

  keyboard:
    layout: us

  authconfig:
    USESHADOW: yes
    PASSWDALGORITHM: sha512

  hostname:
    name: installed

  timezone: UTC

  selinux:
    mode: disabled

  network:
    devices:
    - match: ['class=ethernet']
      bootproto: 'dhcp'
```

And a minimal install tree from which it derives:

```
# treebuilder - A minimal system image
include:
  - base.tree

partitions:
- disk: sda
  partitions:
  - name: bios-boot
    type: primary
    size: 1
    flags:
      bios_grub: True

  - name: boot
    type: primary
    size: 500

  - name: swap
    type: primary
    size: 1000

  - name: root
    type: primary
    size: grow

format:
- mount: /boot
  fs: ext4
  device: sda2

- mount: swap
```

```
  fs: swap
  device: sda3

- mount: /
  fs: ext4
  device: sda4

bootloader:
  location: mbr
  ignore: []
  disks: ['sda']

yum:
  install:
    packages:
      - fedora-release
      - kernel
      - python
      - vi
      - dhclient
      - grub2
      - e2fsprogs
      - parted
      - yum
      - htop
      - libuser-python


setup:
  fstab:
    include:
    - UUID=%(uuid./dev/sda4)s /                   ext4    defaults,discard,noatime        1 1
    - UUID=%(uuid./dev/sda2)s /boot                   ext4    defaults,discard,noatime        1 2
    - UUID=%(uuid./dev/sda3)s swap                  swap    defaults 0 0
    - none       /dev/pts  devpts  gid=5,mode=620  0 0
    - none       /dev/shm  tmpfs   defaults        0 0
    - none       /proc     proc    defaults        0 0
    - none       /sys      sysfs   defaults        0 0
```

1.5 Commands

Commands can be defined in any of the input, processing or output sequences.

The hardcoded command sequence is:

```
install.RunPreScripts
install.ZeroMBR
install.CreatePartitions
install.CreateRaid
install.FormatPartitions
yum.Prepare
yum.Install
yum.RunTxn
yum.RemovePackages
setup.Fstab
setup.RootPassword
setup.KeyboardLayout
setup.Authconfig
setup.SELinux
setup.TimeZone
setup.Network
setup.Firewall
setup.Hostname
setup.SysLog
installer.EmbedInstaller
installer.InstallTreebuilderService
installer.EmbedTree
install.RunPostScripts
yum.Mirror
yum.CreateRepo
installer.BuildInstallerImage
install.InstallBootloader
install.RaidPostInstall
install.Cleanup
```

Commands expect configuration values in specific places in the tree file. Documentation under each command defines an example configuration and its expected values.

Run the program with –list-commands to output a nicely formatted documentation of all possible commands.

1.5.1 Built-in Commands

General Commands

class `install.`**`Raid`**

Creates raid devices from components.

Example:

```
CreateRaid:
    devices:
    - device_name: md0
      level: 1
      components: ['sda2', 'sdb2']
      metadata: 1.1
```

class `install.`**`InstallBootloader`**

sets up the bootloader on specified disks

can specify the target disk(s), the disks to skip, additional grub2-install parameters, and the path future boot partition is mounted to.

creates a grub on the mbr (for now) of all so specified disks, or all disks.

class `install.`**`RunPostScripts`**

class `install.`**`RunPreScripts`**

class `setup.`**`Authconfig`**

simply copying yaml lines to authconfig file.

install_root + any lines to copy should be in the config param

class `setup.`**`Firewall`**

Punches holes in the firewall. Incoming traffic is blocked by default. Outgoing traffic is permitted unless one specifies 'incoming_*' options.

One can specify ports to open either by service names, or explicitly by port numbers.

'ports' will open both tcp and udp outgoing traffic on listed ports. 'udp' and 'tcp' will open only udp or tcp ports, respectively.

'incoming_ports', 'incoming_udp', 'incoming_tcp' will do the equivalent for incoming ports.

'services' option sets relevant outgoing tcp & udp ports. 'protocols' option enables additional protocols, beyond tcp, udp.

class setup.**Fstab**
Creates an fstab in the install root.

Args include - list - A list of lines to write into the fstab file.

class setup.**Hostname**
sets the hostname

Args name - string - the desired hostname

class setup.**KeyboardLayout**
sets the keybord layout and other keyboard parameters.

if given 'name' in the parameter dictionary, will use it for both KEYTABLE and LAYOUT fields in the cfg file. any (further) fields will be copied from the yaml.

if MODEL is not specified, defaults it to pc105+inet

class setup.**Network**
finds network devices, creates fixed names for them, ordered by bus, creates ifcfg files by spec, creates static resolv.conf, if any explicit domain and dns servers

class setup.**RootPassword**
Sets the root password to the argument name password.

class setup.**SELinux**
selects SELinux mode

Args

mode - string - should be either permissive, enforcing or disabled.

class setup.**TimeZone**
selecting the timezone.

selects the timezone. relevant file is expected to exist in /usr/share/zoneinfo. also updates the sysconfig file, albeit I gather that has no effect on the zone selection itself.

Yum Commands

The commands are for working with the yum package manager.

class yum.**Prepare**
Prepares for typical install root package transactions.

Bootstraps a yum/rpm environment in the install root (/var/lib/{rpm,yum}, etc.).

An example:

```
PrepareYum:
    release: 16
    architecture: x86_64
    repos:
    # Affix ourself to the release repo so we arent a moving target with updates
    # If needed, be selective about packages from updates.
    - name: Fedorareleasever-basearch
      url: http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everything
      mirrorlist: https://mirrors.fedoraproject.org/metalink?repo=fedora-$releasever&arch=$ba

    # Treebuilder needs some of its own tools not in the distros repo
    - name: Treebuilder
      url: file://./repo
```

Args:

- cache_dir - str - optional. path to a cache directory
- repos - list - a list of repo defitions. Each repo defintion

- requires at least name and a url or mirrorlist.

class `yum.`**`Install`**

Installs packages using yum.

Args packages - list - a list of package globs

class `yum.`**`Remove`**

Prepares for typical install root package transactions.

Packages and their files defined in yum.Install will be stripped depending on how they are negated.

Args:

repos - list - a list of repo defitions. Each repo defintion

requires at least name and a url or mirrorlist.

class `yum.`**`Mirror`**

The mirror command copys packages from repositories into a single destination.

This is useful if you need to create install disks with all the packages inside.

This is typically used to create the "Packages" directory in the treebuilder installer image.

An example:

```
yum:
    Mirror:
        packages_from: ../minimal.tree
        destination: isoroot/Packages
        release: 16
        architecture: x86_64
        resolve: True
        repos:
        - name: Fedorareleasever-basearch
          url: http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everyt
          mirrorlist: https://mirrors.fedoraproject.org/metalink?repo=fedora-$releasever&arch

        - name: Fedorareleasever-basearch-Updates
          url: http://download.fedoraproject.org/pub/fedora/linux/updates/$releasever/$basear
          mirrorlist: https://mirrors.fedoraproject.org/metalink?repo=updates-released-source
```

Args

•destination - Destination folder to put downloaded packages

•release - release variable

•architecture - architecture variable.

•repos - list - A list of dictionaries specifying the repositories to download packages from.

•resolve - boolean - If true, resolve package dependencies.

•packages - list - A list of package globs

•packages_from - path to a tree to pull a pacakge list from

Repos list item

name - string - name of the repository

one of url or mirrorlist optional

url - specifies the url of the repository

mirrorlist - specifies the location of a mirrorlist for the repository

“Live” Disk Commands

class `live.`**`buildiso`**

class `live.`**`BuildLiveImage`**

Builds a live image

Args

disklabel - string - label stored in .discinfo

rootlabel - string - Label of the live root filesystem

isolabel - string - Label of the iso

isolinuxcfg - string - relative or absolute path of isolinux.cfg to include

syslinux_source - relative or absolute path to where syslinux binaries can be found (eg isolinux.bin, vesamenu)

Installer Commands

Installer commands are defined to help build the treebuilder installer

class `treebuilder.`**`EmbedInstaller`**

Embeds the treebuilder installer into the install_root.

class `treebuilder.`**`Image`**

Builds a live installer image

1.5.2 Writing commands

Treebuilder can be easily extended by adding additional commands.

Matching names from args in the tree file are passed to the respective command as arguments. The default Command implementation makes the dictionary available in Command.args (self.args) for the subsequent call of the commands run method.

Commands are registered by subclassing the Command class and defining the __register__ property. This allows commands to be discovered by treebuilder easily.

TODO: in your configuration define command_search_paths to a python package (with an __init__.py) where commands can be found. Treebuilder will automatically search the directory for any classes deriving from Command.

Command api

class `treebuilder.command.`**`Command`**(*args*)

The command class is used to encapsulate various “commands” that treebuilder can peform.

Subclasses of this are automatically registered using metaclasses.

`SIGNAL_COMMAND_FINISH` **= <treebuilder.signals.Signal instance at 0x2e08368>**

This signal is called just after the command is finished.

`SIGNAL_COMMAND_PROGRESS` **= <treebuilder.signals.Signal instance at 0x2e08320>**

This signal is called just before the command is started.

Its sig looks like:

handle_progress(command, percent=None, level=None, message=’ ‘)

`SIGNAL_COMMAND_START` **= <treebuilder.signals.Signal instance at 0x2d711b8>**

This signal is called just before the command is started.

run (*config*)
Run the command, using the passed configuration object.

1.6 Tools

A tool registry is defined so that tools can be defined and possibly re-defined anywhere.

1.6.1 Tool Api

treebuilder.tool.**register_tool** (*name*)
This is a decorator intended to allow registration of tools plugins under tools. in the global plugin registry

treebuilder.tool.**get_tool** (*name*)
Returns a tool defined by name.

treebuilder.tool.**get_tools** ()
Returns a dictionary of all registered tools.

1.6.2 Available Tools

TODO: tool documenter required.

Refer to treebuilder/tools for now. Anything decorated with register_tool can be used by commands.

Run the program with –list-tools to output a nicely formatted documentation of all tools.

1.7 Treebuilder Development

Most installations can be generalized into three stages: input/gather/pre-processing, processing/install and post-processing/output. Treebuilder was designed with this in mind.

Commands, and their tools, are automatically searched by treebuilder and gathered into a registry (refer to treebuilder.plugins, treebuilder.tool, treebuilder.command).

A standard yaml parser reads the config file and passes off the various stages to a treebuilder.command.Runner. The Runner loops through the command list, performing each command as specified, pulling in matching args defined in the tree file.

Roughly, the execution order of treebuilder looks something like:

- Input stage - optional. this stage prepares for the processing stage. The entire installation (configuration, packages and files) can be made available via local or remote sources.
  - Configuration - make a configuration file available (from local or remote). read the configuration file and find out what files and packages are needed. if a remote configuration file is specified download it into memory.
  - Packages - download them into a cache if needed.
  - Files - if the configuration file is remote, the required files will need to be downloaded to the machine. prepare them for copy (possibly templatize).
- Pre command stage scripts - run python scripts before the command stage
- Processing stage - a series of commands are run in order to produce the tree
  - **Storage**
    * Prepare storage - commands to prepare the disks, partitions and filesystems
  - Install packages

  - Configure system - various commands to configure the system.
  - Install bootloader - commands to configure and install the bootloader
- Post command stage scripts - run python scripts after the processing stage in a optionally chrooted environment
- Output stage - optional. this stage are used to create live disk, pxe or initrd images.
  - Output commands - optional. these commands are used to create live disk, pxe or initrd images.

1.7.1 Core API

class `treebuilder.command.`**`Runner`**

Runs the specified stage defined in config.

config - the global configuration object. commandconflist - should be a list of dicts representing commands.

`treebuilder.tool.`**`register_tool`**`(`*name*`)`

This is a decorator intended to allow registration of tools plugins under tools. in the global plugin registry

`treebuilder.tool.`**`search_tools`**`()`

`treebuilder.command.`**`search_commands`**`()`

1.7.2 Treebuilder installer development

Create TWO virt-manager or virtualbox virtual machines. One to test the installer and one to test the installed. This can be done easily using virt-manager and shared disks.

Make sure these virtual machines can access the network and can talk back and forth with your machine. A bridged connection usually suffices. These matchines should have matching devices (including macs). “virsh edit” is the only way to accomplish this libvirtd.

To work on the treebuilder installer:

1. Clone the treebuilder source
2. On your machine build an installer image (install.iso):

   ```
   buildtree.py examples/installer/iso.tree
   ```

3. Boot the install.iso generated by the above in one of the virtual machines. After booted, ctrl+alt+F2 to enter a command shell and ctrl+alt+F3 to enter a process manager (htop).
4. By default treebuilder will begin the installation as specified by the configuration file embedded in the iso. (TODO: a way to stop this from boot. For now: Stop the default installer process (killall python or find and kill the process in htop))
5. Connect to the treebuilder source you checked out:

   ```
   mkdir /mnt/tb
   sshfs root@<your ip>:<path to treebuilder source> /mnt/tb
   ```

6. Develop on your machine and test on the virtual machine:

   ```
   /mnt/tb/buildtree.py /mnt/tb/examples/minimal.tree -d -dui
   ```

7. As soon as the install is finished, you may unmount /mnt/sysroot/proc, /mnt/sysroot/dev, /mnt/sysroot/sys, /mnt/sysroot/boot and /mnt/sysroot and boot the disk in the second image; while the installer is still running.
8. If you are not satisified, run step 6 again to reinstall.

1.7.3 Building a ui around treebuilder

Treebuilder uses a type of signals framework for notifying a calling program of status and progress.

These signals can be found under treebuilder/command.py and treebuilder/tool.py.

`Command.`**`SIGNAL_COMMAND_START`** **= <treebuilder.signals.Signal instance at 0x32f9368>**
    This signal is called just before the command is started.

`Command.`**`SIGNAL_COMMAND_PROGRESS`** **= <treebuilder.signals.Signal instance at 0x34034d0>**
    This signal is called just before the command is started.

    Its sig looks like:

    handle_progress(command, percent=None, level=None, message='')

`Command.`**`SIGNAL_COMMAND_FINISH`** **= <treebuilder.signals.Signal instance at 0x3403518>**
    This signal is called just after the command is finished.

`Tool.`**`SIGNAL_TOOL_START`** **= <treebuilder.signals.Signal instance at 0x32eccb0>**
    This signal is called when a tool is started

`Tool.`**`SIGNAL_TOOL_PROGRESS`** **= <treebuilder.signals.Signal instance at 0x32eccf8>**
    This signal is called when a tool makes progress.

    Its sig is handle_progress(tool, progressdata=None, level=None, message='')

`Tool.`**`SIGNAL_TOOL_FINISH`** **= <treebuilder.signals.Signal instance at 0x32ecdd0>**
    This signal is called when a tool is finished

Treebuilder has a built in command line/terminal "user interface". An example of how it connects and uses those signals can be found in treebuilder/ui/cli/handlers.py.

Useful notes: http://fedoraproject.org/wiki/Anaconda/Stage2DevelopmentGuide http://fedoraproject.org/wiki/Anaconda/Stage1DevelopmentGuide

1.7.4 Source Overview

- docs - this documentation
- examples - example trees and configurations
- treebuilder -
- treebuilder/commands - commands are sorted here
- treebuilder/tools - tools that commands use
- treebuilder/ui - user interfacing layers

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*