# Linux Guidebook 2

By Joseph Colton

DRAFT: May 23, 2019

# Quick Table of Contents

# Complete Table of Contents

# Preface

Thank you for reading the preface of the Linux Guidebook 2. This book is intended to be a resource that you can use to help you better understand Linux and the abilities that it provides.

The original version of this book, the Linux Guidebook, was written in 2002 and was written to help people who were working with Red Hat Linux 7.1 (Before Red Hat Enterprise Linux). I decided that it was necessary to write a new version as I started to notice that the commands I was instructing people to use were not working as often. This has been especially difficult with the change from sysvinit to systemd.

This version of the book is based on CentOS 7 which is the first CentOS version to come with systemd. CentOS is part of the Red Hat Linux family, so most of the content will continue to be useful for users in the Red Hat family for years to come. Many of the major Linux distributions have started using systemd, so this book will be helpful where a lot of older documentation for other distributions will now be obsolete. While Debian and source code based Linux distribution users can still benefit from much of this book, some of the commands will be Red Hat family specific.

# Copyright

# Linux Distributions

Since Linux is free and most of the software for Linux is free there has to be someone to put the operating system and the software packages together. Red Hat is currently the most popular company that makes a Linux distribution, but there are many more, including CentOS.

There are three main branches or families of Linux distributions. The three branches are the Red Hat Family, the Debian Family and the source code family. The way I identify them is by their package management systems. While there are many distributions, the one thing that they have in common is that they all use a Linux kernel.

In addition to the Linux kernel, most distributions also use the GNU utilities. Because the GNU term in the distribution title is a political one, you will see some distributions listed as GNU/Linux and some with just Linux in the name.

You can find many of the popular distributions at http://distrowatch.com/.

## *Red Hat Family*

Distributions in the Red Hat Linux family use the Red Hat Package Management system or (RPM) packages. Not all distributions in the Red Hat family are derived from Red Hat, but most were at some point. In addition to the RPM packages and utility, most of the Red Hat family has adopted the Yellow Dog Update Management tool (YUM). This family includes the following well known distributions as well as many others:

- Red Hat Enterprise Linux - http://www.redhat.com/en
- Fedora - https://getfedora.org/
- CentOS - https://www.centos.org/
- openSUSE - https://www.opensuse.org/en/
- Mageia - https://www.mageia.org/en/

## *Debian Family*

The Debian family was started with support from the Free Software Foundation. They wanted a Linux distribution that represented the community supported free software model. At one point Debian was the only Linux distribution that was really considered free by Richard Stallman the founder of the Free Software movement and author of the GNU General Public License (GPL).

The Debian family of distributions use packages with the .deb file extension. Like the yum utility for the Red Had family, the Debian family has a tool, apt-get, that automatically finds and installs dependencies.

Because of difficulty in working with a community of developers some have forked the debian project and have created derivative distributions including the well known Ubuntu distribution. The following are a few well known Debian based Linux distributions:

- Debian GNU/Linux - https://www.debian.org/

- Linux Mint - http://www.linuxmint.com/
- Ubuntu - http://www.ubuntu.com/

## *Source Code*

The Free Software movement started with developers who decided it was better to share code than close it up. All Linux distributions are based on a collection of source code projects. The problem with pre-packaged software distributions is that all of the compile options have already been made. This means packages are optimized to run a particular way. Additionally, since the packages are pre-built, they sometimes list requirements that are not accurate. Updating packaged distributions can be difficult because you are using a curated collection of software that someone decided would work best for you.

Slackware started by bundling software on sets of floppy disks that you could then use to build your system. That was great because there were not a lot of options and people did not mind the compile time. Slackware later made pre-compiled binary packages, but did embrace the package management systems with forced dependencies.

Some source code distributions have tried to avoid the pre-compile even more. Gentoo was a leader in providing a list of packages and a place to download them from as well as letting you decide what compile options you wanted when building your system. After the Gentoo founder left to work for Microsoft in their Linux labs for a few years the project started to die. Arch Linux has taken up most of the Gentoo market and helps people who want to fully customize their systems.

For those really interested in learning more than either Gentoo or Arch has to offer, there is the Linux from Scratch option. They have manuals that walk you through the full installation process from building your static system to building the dynamic libraries and making a fully working system.

Any of the advanced source code versions of Linux will make the users full super users, but the learning curve is very steep. The following are a few source code based Linux distributions:

- Slackware Linux - http://www.slackware.com/
- Arch Linux - https://www.archlinux.org/
- Gentoo Linux - https://www.gentoo.org/
- Linux from Scratch - http://www.linuxfromscratch.org/

## *SystemD*

SystemD was created as a replacement for the older init system. The init process was the first process executed in user space. This process has the responsibility of starting the rest of the system including services, firewalls, driver modules, terminals, and even the GUI.

The init process is not just Linux, it actually comes from the Unix world and is commonly referred to the System V init. The init process had a long list of services that it started up on boot time. As each service was started, init would move on to the next one. In a parallel processing world, this did not make much sense.

The biggest initial differences between init and systemd for new Linux users is probably server start and stop syntax. Below are some examples of starting, stopping, and restarting services using init and

systemd.

## Command Differences

Commands to start, stop, and restart httpd using init and the service command:

```
service httpd start
```

```
service httpd stop
```

```
service httpd restart
```

Same commands, but using systemd and the systemctl command:

```
systemctl start httpd.service
```

```
systemctl stop httpd.service
```

```
systemctl restart httpd.service
```

If you wanted to add or remove the httpd service to or from your list of services to start at boot time using init and a Red Hat type system you used chkconfig to issue the following command:

```
chkconfig httpd on
```

```
chkconfig httpd off
```

Doing the same thing with systemd:

```
systemctl enable httpd.service
```

```
systemctl disable httpd.service
```

The new systemd also has the additional option you can figure out below:

```
systemctl is-enabled httpd.service
```

## Behind the Scenes

**Init**

When the commands to start and stop services were issued in the past with init, the service command worked as a front to start and stop scripts in the **/etc/init.d/** directory. The /etc/init.d/ directory was the repository for all scripts that started and stopped services.

The new systemd enable and disable was nothing more than making symbolic links with init. With the init system, each script contained information about the start and stop order that was parsed out so that scripts could be stopped and started in correct orders. The run levels in init were stored in /etc/rc#.d/ directories. For example run level 3 would have symbolic links in the /etc/rc3.d/ directory.

Each run level directory contained symbolic links that started with either the letters S or K. S stood for start and K stood for kill. After the instructional letter we had a number that indicated the order the service should start or stop in. The idea was that if you ran all of the scripts in alphabetical order you would kill processes, then start processes in the order they sorted.

The symbolic links pointed to the actual scripts stored in the /etc/init.d/ directory.

**Systemd**

Services that are now started and stopped contain scripts in the /usr/lib/systemd/system/ directory. When you enable a service, you create a symbolic link from your run level directory to the location of the scripts. The run level directories are all in the /etc/systemd/system/ directory. For example, the multi-user level (same as past run levels 3-5) directory is **/etc/systemd/system/multi-user.target.wants/**

Instead of just numbers in the scripts, the scripts contain information about what they really need to start. As you might expect, the **httpd.service** lists **network.target** as an item in the After field.

When a service is disabled, all that really happens is that the symbolic link is removed. This does not immediately stop the service, but does prevent the service from starting when the machine is rebooted.

# CentOS 7 Installation Walk Through

The easiest way to test out Linux is to use a Live DVD. The idea is that you download an ISO image from the Linux distribution's website, but the ISO image to a blank DVD, then with the DVD in the drive, reboot your machine and using the boot menu, select the DVD device. You are then usually presented with the options of loading the Live DVD or performing an installation.

Some Linux distributions focus more on installation and not on Live DVDs. The Red Hat family of Linux distributions tends to be mostly installation focused. The Debian family more commonly has the Live DVD options.

## *Boot Options*

When you initially boot your CentOS 7 installation DVD, you are given just under a minute to select a boot option. You can use the up or down arrows to select the different options. The options are as follows:

- Install CentOS 7
- Test this media & install CentOS 7
- Troubleshooting

We are going to want to select the Install CentOS 7 option, but what do the other options do?

The test option is good when you are using a physical DVD and are not performing a virtual machine type of installation. At times when you burn the DVD, there are mistakes. You may have successfully created a DVD and might be able to read it correctly with a full operating system, but for some reason, during the installation process, the device drivers are a little bit less comfortable rereading problem sections of the disc, and crash. There are also times when the disc is not handled properly and gets damaged.

The test option checks to make sure the media looks correct and you are able to read the whole disc. It tries to reduce the number of times you get half way through the installation only to discover that your disc has a scratch or other bad section. After the test is complete, your installation starts.

The troubleshooting option actually gives you more options. You can test memory, boot from your local disk drive, boot into a rescue environment, or even install CentOS 7 with lesser graphical options.

- Select Install CentOS 7
- Press Enter to continue

This will tell your machine to start booting the installation environment.

### *Language Selection*

When your CentOS 7 installation environment has booted, you are greeted with a welcome message and asked to select the language you would like to use. I am going to assume that you are using English (United States).

- Select the English category
- Select the English (United States) option
- Click Continue

### *Localization, Software, and System*

CentOS 7 has grouped a lot of tasks that need to be completed in any order onto a single screen. From this screen you can decide which items you would like to complete and in which order. There are a few items that are marked with an orange triangle icon that indicate they are required before you begin the actual installation. After making selections, sometimes the system needs to verify that the options you selected do not require more configuration. During this multi-second check you might see additional orange triangle icons. They disappear once you system has decided they are not required.

### Date & Time

The Date & Time option allows you to set the clock for your system and decide which time zone your system will use.

- Verify/Update the Region/City or Click the map
- Verify/Update the date
- Verify/Update the time

If you already had networking set up, you could enable the Network Time Protocol (NTP).

### Keyboard

I will assume you are using an English (US) keyboard, but if you have something different, now would be the time to configure it. If you configure multiple keyboards, you can also decide which key/key combination you will use to switch between keyboard layouts.

- Click Done

### Language Support

I will assume you are using English (United States), but if that is not the case or if you would like to add additional language support, select the language category and select the additional language.

- Click Done

## Installation Source

Because you are performing an installation from a DVD, you do not need to even look at this category, but you do have additional options. We want to keep all of the default options.

One option is to configure your machine to perform an installation from the network. This can actually be a lot faster than a DVD in some cases. Because network based installation requires access to the network, you would have to have a server in place with the configuration files and have your network configured before you can begin installation.

Your network based options are using the http, https, ftp, and nfs protocols. If all of the files from the DVD are copied to a directory on a server somewhere they you just need to select the protocol to access the directory and the directory to start your installation.

You can also add additional repositories at this time. These are sites with additional software packages that you might like to use.

- Click Done

## Software Selection

The software selection tries to help you get the correct software for the way you intend to use the system. Many people assume client and server machines are different in both hardware and software. This is not actually the case, well, not completely. You can use any of the base systems to run your machine however you want. There are sometimes memory and disk drive limitations, but that is rarely the case for normal machines.

The Base Environment is just a grouping of packages that are usually used in a given environment. The Minimal Install is the best option when you are concerned with security and do not want extra services running. The GNOME Desktop is probably the best option for users who plan to use the system as a regular web browsing client system. The Development and Creative Workstation option works great if you would like to have your development libraries there for creating software. Once again, any of the Base Environments can have any packages installed, you just have to manually add the packages.

After selecting the Base Environment, you can select additional add-ons. These are package sets that are common for the selected Base Environment. They are not tied to the Base Environment in any restrictive way, so you can change your base after selecting add-ons and they will still be selected during installation.

- Select “GNOME Desktop”
- Click Done

## Installation Destination (Automatic)

Before you can begin your installation, you have to select a destination. Just selecting the Installation Destination will usually automatically select the default options, but this is a step you must make.

You can see a check mark on a disk drive in your list of Local Standard Disks. This is the intended installation destination. You can click the disk drive to unselect it and/or select something else. You will need to make sure something has the check mark before you can continue.

In addition to your local disks, you can additionally add specialized or network disks. These specialized disks would include things like RAID and SAN options.

Partitioning is usually done well, so you can ignore that. Some people do have specialized servers and want to make sure some of the data is stored in different partitions. If you select the "I will configure partitioning" option then you will be greeted with an additional screen when you click Done. In this screen you will have to configure all of the required partitions.

The partitions a functioning Linux system needs are a root or / partition and a SWAP partition. Everything else is really optional, even though some distributions try to force separation for directories such as /boot and/or the /home directory.

Standard partitioning means the partition start and stop locations are fixed and difficult to move. The LVM type options make things more fluid and add an additional layer or partitioning. The LVM partitioning scheme allows for you to add and remove blocks of unused space to and from partitions. LVM also allows expansion onto additional disk drives. LVM is now the default.

- Select "Automatically configure partitioning"
- Click Done

## Installation Destination (Configure Partitions)

If you would like to create your own partitions you can do so. First you need to tell the installer that you want to configure your own partitions.

- Select "I will configure partitioning"
- Click Done

At this point you will be presented with dialog boxes for creating partitions. The boxes initially ask you for the mount point and size, but later allow you to make changes including changing the file system and partition labels.

In creating partitions there are three partitions I would strongly recommend. First, you need to have a root file system which is your initial / directory. Second, you want to have a SWAP partition so that your virtual memory works correctly. Third, you should have a /boot partition so that your kernel, initial ram disk, and boot loader all have a place to hang out.

The rule for SWAP used to be that it should be twice the size of your RAM, but because the amount of RAM systems have has gone up, it does not always make sense to have that much SWAP memory. A good idea is to think about the most memory intensive things you might do on your system, then make sure your RAM plus SWAP is enough to handle it and have a little excess.

Here is a listing of the mount points to consider and a little information about why you might consider doing each as a separate partition:

- The / directory – You just need this directory, so plan on making it. Anything not stored in a different partition will default to being in this root partition.
- The /boot directory – Having this directory as a separate partition ensures the kernel has a place and that there is room for updates to your kernel. It is usually a good idea to have about 512 MB of space so that you do not run out.
- The /home directory – If your user accounts are stored in a separate partition then you can reformat and reinstall your main system without messing with the home directory files. If you do keep the user files in a separate partition you will still need to remember that the users also have file information stored in the /etc directory which includes passwords and group assignments.
- The /var directory – Systems which receive email messages or generate logs tend to fill up the var partition. If your /var directory is part of the main / root directory then you could run into a situation where the logs or spam messages fill up your whole system and causes bad damage. One way to prevent this is to have the /var directory as a separate partition so that your system can continue to function will full logs.

## Network & Hostname

When performing an installation it is best to have an IP address and hostname. You can set your hostname in the hostname box. You should use your fully qualified hostname. Anything less than a fully qualified name works, but can cause problems later from things like mail servers that use the hostname of your system to function.

When you configure the interface, the first thing long time users of Linux might notice is that the name of their network device is not eth0. This can be confusing, but this is the way things are going. With the switch to SystemD, that was one of the changes. Enjoy your new device name.

After selecting the configure button, you can decide things like the IP address in the IPv4 settings tab. You can do things like clone MAC addresses, etc. The one thing that you will probably want to do is configure the device, under the General tab, to “Automatically connect to this network when it is available”.

After configuring the interface, you probably want to Save, then select the “ON” option to the right of the interface name. This increases you chance of getting an interface running at boot time.

- Click Done

## Begin Installation

Up until this point you have not made any actual changes to the system. When you press the “Begin Installation” button, things start to happen, beginning with partitioning and formatting devices.

- Click Begin Installation

## *Passwords*

While the installation is happening, you are prompted to create passwords and users. You probably want to have a little bit of both.

### Root Password

Set your root password to something you will remember. If the password you select is considered weak, you will be asked to press Done twice before you are allowed to leave that screen. Do not forget the password.

- Create a password
- Click Done

### User Creation

It is best to have a user account to use in a GUI environment. Create a standard user account. You should put in the full name and decide if you want to use the automatically created username or create one of your own. The same rules for the root password also apply here. Remember your regular user password.

- Create a user
- Click Done

### Reboot

After completing your installation, you will be able to press the Reboot button. This will bring you into your freshly installed Linux distribution.

## *Initial Setup*

Once you boot, you are asked to accept the license. You must accept the license to continue.

### License Agreement

Read through the EULA. Basically, you just agree that you have no rights and the software is GPLv2 licensed.

- Check “I accept the license agreement”
- Click Done

## Finish

You are now done with the main part of installation.  Only a little bit more.

- Click Finish Configuration

### *Kdump*

You think you have finished everything, but there is one more step.  What if you are a kernel developer and want to have access to memory of the kernel when your system crashes?  Okay, you probably do not care.  Go ahead and disable the Kdump to save memory for things that you like to do.

- Uncheck Enable kdump
- Click Forward
- Reboot the system

Now, you really are done.

### *System Update*

Assuming you have your network up and running, now would be a good time to update your system.  It is important to remember to update so that you can stay current with software and to fix problems.  When updating, it is safest if you are not GUI mode if you have a lot of updates.  To switch to a pure command line mode you can use the **Ctrl-Alt-F2** key combination.  To switch back to GUI you can use the **Ctrl-Alt-F1** key combination.  You can actually, use Ctrl-Alt-F2 through Ctrl-Alt-F6 for text mode.  In text mode you can type the user name and password to login.  Run the following as root on the command line to update your system:

```
yum update
```

Normally after a new install there are a lot of packages that can already be updated.  This is because the installation media is not always up to date.

## Command Line Interface

Many users will naturally feel more comfortable using a graphical user interface, but this reduces your options. On Linux and UNIX-like systems the graphical user interface is built upon the command line. Developers naturally find it easier to develop programs that run on the command line and do not require the extra graphical components. Since many users like the GUI, some developers like to create a graphical interface to the command line programs below. Since the developers themselves prefer the command line, the graphical interface is usually an after thought and does not implement everything available from the command line.

Even if the GUI provides all of the functionality that you desire, it is still quite likely that it does not provide as much detail when errors occur or something unexpected happens. Most Linux users eventually make the switch to the command line and use the GUI to launch terminal emulators and graphical applications such as web browsers, office products, and games.

If your system does not have a graphical user interface installed, then you will have to start with the command line. If you do have a graphical user interface, you can launch a terminal emulator or use a key sequence to switch to the command line.

### *Installing the GUI*

Assume for a moment that you did not install the GUI during the initial installation. If you want to have the GUI on your system you need to make sure you first have at least 1GB of RAM, then you have to install the software and get it running. For all software installations you need to either log in as the "root" user or you need to use either the "su" or "sudo" commands to get root permissions.

You can check the available memory by using the following command to view information in the kernel directly:

```
cat /proc/meminfo
```

If you have enough RAM you should go ahead and perform the GUI installation.

As the root user you can use the "yum" command to install software. Use the following command as root to install the GUI components required for the GNOME desktop:

```
yum -y groupinstall "GNOME Desktop"
```

If you performed a minimal install, this will install hundreds of packages. As long as you type to command above correctly you should have a full GUI installed with the GNOME environment.

Now, if you want to configure the system to start in the GUI you can use the following command:

```
systemctl set-default graphical.target
```

This command configures Linux so that your graphical user interface will start at boot time. This does not start the GUI right now. To start the GUI it is probably easiest to just type the "reboot" command to restart your system and make sure everything starts correctly.

If you later decide you want to disable the default graphical interface, you can use the following command to switch to the text interface default again:

```
systemctl set-default multi-user.target
```

For switching between the GUI and command line interface for the full system without changing the boot default the following commands might be useful:

```
systemctl isolate graphical.target
```

```
systemctl isolate multi-user.target
```

If you decide that you want to just switch between the GUI and command line without changing how the system is running, you can use the Ctrl-Alt-F1 through Ctrl-Alt-F7 key sequences to switch. Normally the GUI on CentOS 7 runs at Ctrl-Alt-F1, but on some versions of Linux it can be found other places such as Ctrl-Alt-F7.

## *Bash Key Combinations*

When you are using a Linux command line, the default program that is running is called **bash**. Bash is a command shell. There are multiple shells available, but bash is the default found on most Linux and even Mac OS X machines.

The shell take commands you type in and parses them to figure out what you actually want to do. At the most basic level, the shell takes your command line and uses the first argument, or everything before the first space character as the command name. After the first space, the rest of the line is divided into arguments usually using the space character as a deliminator.

You can use the **Ctrl-a** and **Ctrl-e** key sequences to jump to the front or back of a command line. Ctrl-a takes you to the front of the line. Ctrl-e takes you to the back of the current command line. The **Ctrl-b** and **Ctrl-f** keys take you back or forward a single character. Ctrl-b takes you back a single character and Ctrl-f takes you forward a single character. You can also use the left and right arrow keys to accomplish the same task.

You can edit the line where ever your cursor is placed. To erase everything after your cursor, you can use the **Ctrl-k** key sequence. To erase everything before the cursor you can use the **Ctrl-u** key sequence.

You can scroll up and down through your history of commands using the **Ctrl-p** and **Ctrl-n** key combinations. Ctrl-p takes you to your previous command and Ctrl-n takes you back to the next command after the one you are currently displaying. You can also use the up arrow in place of the Ctrl-p to see previous commands and the down arrow in place of the Ctrl-n key sequence to see the next

command after the current one.

The **Ctrl-l** key combination will clear the screen and display only the current command you are typing. This can be useful if you find the bottom of the terminal outside of the visible part of the screen.

If you are in the middle of displaying output from a program or want to stop the display of information to the screen, you can press the **Ctrl-s** key sequence. To once again allow text to write to the terminal screen you can press the **Ctrl-q** key sequence.

If you decide you want to quit the current command, you can press the **Ctrl-c** key sequence. This works in any programs that do not intentionally catch the Ctrl-c break sequence.

## *Home Directory Configuration Files*

Each user has a home directory. This directory contains the basic configuration files for when a user logs into their Linux system. You can see these files from the terminal using the **ls** command, but since they start with a period "." they are officially "hidden". You can see all files using the **-a** option with the **ls** command.

```
ls -al
```

Two useful files for configuration are the **.bash_profile** and the **.bash_logout** files. You can can set environmental variables such as the **PATH** in the .bash_profile. If you had some binary executable programs in the **/opt/software/bin** directory you could add them to the path, by editing the .bash_profile file with an editor like **nano**.

```
nano .bash_profile
```

When inside you could add the new path to the end of the **PATH** variable. To add a new directory to the end, you need a colon, then the path.

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin:/opt/software/bin
```

Note, that at the end of the .bash_profile file there is a line that exports the **PATH** variable back into the environment. This makes it so the environment variable is changed after the program finishes. Also, the changes will only take place after the user logs in again.

If you want commands to execute when you log out, you could add them to the **.bash_logout** file. A common example of something you might want to do is add the **clear** command to the end of the file. If this command is in place then the terminal screen will be cleared when you logout. The following is an example of the contents with the clear command added to the end.

```
# ~/.bash_logout
clear
```

### *File/Directory Navigation*

When you start at the command line you are at some location. Usually you start in your home directory. Sometimes your location is displayed on the command line as part of the prompt, but not always.

If you want to know where you are at any point, you can use the **pwd** command. This command prints the working directory.

## Directory Structure

Unlike Windows where each device has a different letter and root starting place, UNIX-like systems such as Linux have all devices attached to a single tree of directories. The top most directory is called the root directory or the / directory. Sometimes this causes problems because we also have the /root directory which is the root user's home directory.

Now assume you are logged in as a user "alice" and you are in the home directory. The normal place for alice to have her home directory would be "/home/alice". If user alice wanted to move into a different working directory she could use the "cd" command to change directories.

The cd command can be used to move into relative or absolute locations. Since alice is in the /home/alice directory, relative directories would be in relation to that directory. Absolute directories would be based on the root and are not tied to the directory you start in when you enter the command.

## Relative Directories

From a starting directory you can move into the same directory as you are in, you can move to a parent directory, or to a child directory.

To move to the directory you are already in, you can use the single period with the cd command. The following command will move you from a directory into that same directory, effectively not moving you at all:

```
cd .
```

This command may seem pointless, but the single period does have a purpose and can be very useful for some later commands.

To move into a parent directory you can use the ".." directory name. This double period expression means the parent directory to the current directory.

If alice used the following command in her home directory she would move into the "/home" directory:

```
cd ..
```

Using the same command would move her from the /home directory into that directory's parent directory. The parent for the /home directory is the / directory.

If you are in the /home directory you can move to child directories by using the cd command with the child directory name to change directory into the child directory. From the the /home directory alice could use the following command to get back into her home directory:

```
cd alice
```

Moving a single directory at a time can take a lot of commands to get from one place to another and is not the most efficient way to move around the system. If you know where you are at, you can combine command arguments to move more quickly.

If alice were in her own home directory of /home/alice and wanted to get into bob's home directory (and had permissions), she could use the following command:

```
cd ../bob
```

This command would move from the /home/alice directory into the /home directory, then from there into the bob child directory to end up in the /home/bob directory.

## Absolute Directories

When using absolute references to directories, you need to base the directory you are going to on the root directory. All absolute directories start with a leading slash character.

If user alice wanted to go to her home directory of /home/alice, she could, from any location, use the following command to get there:

```
cd /home/alice
```

Additionally, assuming she had permissions, she could get into user bob's directory using a similar incantation:

```
cd /home/bob
```

For both of these commands it does not matter where your starting directory is. You can additionally combine relative elements and absolute elements in a single command. There are many times when programs and scripts have a base directory they are working with, but from there they move to locations in relation to the base. The following are valid commands that could potentially end up in alice's home directory:

```
cd /home/alice
cd /home/bob/../alice
cd /home/alice/.././alice
cd /home/alice/public_html/..
```

The thing that is common among all of the previous commands is that they all start with the leading / slash and they all end in the /home/alice directory.

### *Directory and File Management*

When using Linux, it is common to need to create, delete, or even move or rename directories or various types of files. This is usually fairly easy to do using the command shell, but does take a little bit of getting used to. Following are some descriptions of tasks and how to accomplish them.

## Command: cat

Sometimes you want to see the contents of a text file. You could use an editor, but if you want to save time it is sometimes easier to just list the contents of the file on the screen. The **cat** command allows you to see the contents of a file easily.

If you wanted to see the contents of filename.txt you could use the following command.

```
cat filename.txt
```

In addition to just seeing the contents of a single file, you can see the contents of multiple files by either putting them all on the command line as arguments or using something like the star symbol to match multiple files. The following commands can be used to see multiple files.

```
cat file1.txt file2.txt file3.txt
```

```
cat *.txt
```

If for some reason you forget to pass a filename as an argument then the cat command assumes you are going to be giving it something from standard in or the keyboard. Sometimes the cat command can use this behavior to create files. Assuming you do not make mistakes you could create a file message.txt using the following command. When you are done, press the Ctrl-d key combination to indicate the end of input.

```
cat > message.txt
```

## Command: reset

If you go around displaying random files, you are likely going to display the contents of a binary file. Sometimes binary files happen to have characters that have special meaning to the shell and they mess up your view. If this happens, you can use the reset command.

Just type the reset command to see if you can clear up the fonts and make everything readable again.

```
reset
```

## Command: pwd

Directories are special files that contain information about other files. When working with directories, it is usually important to know where on the system you are currently working. You can use the **pwd** command at the command prompt to print the working directory.

```
pwd
```

The command will then print the absolute path for the working directory. If you discover that you are not in the directory you wish to operate in, you can use the **cd** command to change directories.

## Command: mkdir

You can create and delete directories using the **mkdir** and **rmdir** commands. All these commands do is work with the creation and deletion of directories. Other non-directory files are not affected and can even prevent directories from being deleted when the directories are not empty.

The **mkdir** command creates directories. Normally you navigate to a directory and from there create a sub-directory. If the user **alice** wanted to create a directory called **bin** in her home directory the following dialog would work.

```
bash$ cd /home/alice
bash$ mkdir bin
```

Additionally, since Alice is moving to her home directory, she could have used the **cd** command without any arguments or with just the tilde ~ character.

```
cd
```

```
cd ~
```

If Alice were in a different directory on the system and wanted to create the **/home/alice/bin/** directory

without moving she could have used the absolute path to the directory instead of the relative path. The following command would accomplish her task.

```
mkdir /home/alice/bin
```

Sometimes users want to create more than one directory in a directory path. If Alice wanted to create the directory **/home/alice/docs/presentations/** she could first create the **/home/alice/docs/** directory, then create the **/home/alice/docs/presentations/** directory. There is, however, an option flag that can be used to tell the **mkdir** command that you want to create all of the required directories to make a directory path work. If you use the **-p** option the **mkdir** command will create the whole path.

```
mkdir -p /home/alice/docs/presentations
```

This command will create both the docs/ and presentations/ directories all in a single command. The -p option also helps in cases when you do not know if a directory has been created. If the directory already exists, no error is reported.

## Command: rmdir

The **rmdir** command is used to remove empty directories. Because of the danger in removing directories that have files inside this command will refuse to delete non-empty directories. To remove an empty directory use that directory as an argument. If Alice wanted to remove her /home/alice/bin/ directory she could issue either of the following commands.

```
rmdir /home/alice/bin
```

```
cd /home/alice
rmdir bin
```

The first one uses the absolute location of the directory to remove it. If alice uses the absolute location then it does not matter where she is working on the system. The second command set moves to Alice's home directory, then removes the directory using the relative name of the directory.

Earlier, Alice created multiple directories with a single command using the **-p** option. If Alice has not put anything into the directories she can once again remove all of the directories with a single command using the **-p** option. The following command set could be used to remove all of the created directories at once.

```
Cd /home/alice
rmdir -p docs/presentations
```

This command is equivalent to removing the **docs/presentations** directory, then removing the **docs** directory.

## Command: touch

The **touch** command is commonly used to create empty files, however, the intended purpose was to change the timestamps on files. If you use the touch command with a single argument of a file name then the file will either have the modification timestamps updated or the file will be created if it is missing.

To create multiple files you could use the following command.

```
touch file1.txt file2.txt file3.txt
```

This will cause the creation of file1.txt, file2.txt, and file3.txt. Now on to the intended purpose. To change the timestamp for one of the newly created files we could use the touch command and set the stamp. The following command would change the timestamp for file1.txt to December 31, 1999 at 11:59 PM.

```
touch -t 199912312359 file1.txt
```

The timestamp format is YYYYMMDDHHMM. So we can see that we set the year to 1999, the month to 12, the day to 31, the hour to 23 and the minutes to 59. Now, if we wanted to set the modification timestamp of file2.txt to also be the same date as file1.txt we could use the same command again and just change the file name, but we have an additional option. Instead of setting the date using the time stamp format we can use the first file as the reference file. To copy the timestamp from the first file we can use the following command.

```
touch -r file1.txt file2.txt
```

## Command: rm

To remove files we can use the **rm** command. The rm command takes the file name as an argument and deletes the file. To delete the file file1.txt we could use the following command.

```
rm file1.txt
```

The rm command does not delete directories normally, but if you use the **-r** option then the command will remove directories and files recursively. The following could be used to delete Alice's home directory and all files in that directory including sub-directories.

```
rm -r /home/alice
```

When you are using the rm command there are times when you are prompted to accept each file you are going to delete. This prompting is caused by the **-i** option. You might not remember typing the **-i** option, and that is expected, but since problems often happen where root deletes too many files, there is an **alias** on many machines that causes the rm command to actually run "**rm -i**" instead of rm by itself.

To override the -i option you can use the -f option or you can avoid the alias by using the absolute name of the file instead. The following two commands would both ignore the -i option when deleting files.

```
/usr/bin/rm file1.txt
```

```
rm -f file1.txt
```

The first one never had the -i option and the second one overrides it with the -f (force) option.

It is common to see Internet posts where someone pretending to give advise will tell someone to run the following command. (Note: the following command is not a good command to run).

```
rm -rf /
```

## Command: unlink

When you remove a file, all you are really doing is unlinking it. The **unlink** command will also delete the file, but it will do it a different way. The unlink command removes a link to the file, which causes the link count to go to zero. When the link count is zero then the space is marked available and the file is effectively gone. You can delete files using the unlink command.

```
unlink file1.txt
```

Normally, you would not delete a file by unlinking it using the **unlink** command.

## Command: mv

To move files and directories or to rename them we can use the **mv** command. The mv command can be very fast because if the source and target destination locations are on the same file partition then the name just changes. If, however, the locations are on different partitions then the file is copied to the target destination and the source file is deleted.

To rename the file **alice.txt** to **bob.txt** we could use the following command.

```
mv alice.txt bob.txt
```

We can also rename directories this way. To rename the **alice/** directory to **bob/** we could use the following command.

```
mv alice bob
```

There is something to remember. If the **bob** directory already exists then instead of renaming the directory the **alice** directory will be moved into the **bob** directory. If you remember this, then moving files should be pretty easy. As long as the destination directory exists then the files and directories will be moved instead of renamed. The reverse is also true. If you wanted to move a file into a directory, but the directory does not exist then a file with the intended directory name will likely be created instead.

To move the file alice.txt into the directory /home/alice/ we could use either of the following commands.

```
mv alice.txt /home/alice
```

```
mv alice.txt /home/alice/
```

These two commands look about the same, but they are different. If the directory **/home/alice/** exists than both commands will have the same result. If, however, the **/home/alice/** directory does not exist then the first command would move the file **alice.txt** to the /home/ directory and call it **alice**. The second command would notice that the directory **/home/alice/** does not exist and would report an error. Because of this behavior, it is recommended that when moving something to a new directory that you end the directory name with a trailing slash character.

## Command: cp

The **cp** command is used to copy files. To copy file in your current directory to a different file name also in your current directory you can use the cp command. The following copies the **file1.txt** file to the file **file2.txt**.

```
cp file1.txt file2.txt
```

If you want to copy a file from one location to a different location then you can have the second argument a directory instead. To copy the file **eve.txt** to the directory **/home/eve/** we could use either of the following commands.

```
cp eve.txt /home/eve/
```

```
cp eve.txt /home/eve
```

Just like with the mv command, if the destination directory does not exist then an error is reported if you include the trailing slash. If you are missing the trailing slash then a file with the name is created instead.

If your source is a directory then the command will assume you meant to recursively copy the directory, but since you did not specify the **-r** option it will refuse. To recursively copy a directory tree to a new location you can use the **-r** option. To copy the contents of the **/home/** directory to a directory **/homebackup/** you could use either of the following commands.

```
cp -r /home/ /homebackup/
```

```
cp -r /home/* /homebackup/
```

The first command will copy everything including the /home/ directory into the /homebackup/ directory. This would result in a /homebackup/home/ directory being created that had all of the contents of the /home/ directory.

The second command would copy all normal files and directories in the /home/ to the /homebackup/ directory. If there were no files or directories that started with a dot "**.**" then the command would have the desired result. If you do have files that start with "." then they would be missing. There is a temptation to use an additional command accidentally copies the "**..**" directory as well, so be careful. Remember you can always move files and directories after they have been copied.

## Command: ln

The **ln** command can be used to create links. This command can create both symbolic links and hard links. When files are created there is a place in memory that is marked as used and a file entry is created in a directory that points to the location in memory. The entry that points to the place in memory is called a hard link.

When files are removed, they are unlinked. When a file no longer has any links the space is marked as unused and the file is mostly gone. You can still recover the contents of the file until the space is overwritten, but it can be difficult to find that space.

To create an additional hard link to the data for an existing file you can use the ln command with the first argument being the existing file and the second argument being the new file or link name. If you have an existing file called old.txt and want to create a new link called new.txt then you could use the following command.

```
ln old.txt new.txt
```

You can instead opt to create symbolic link to the old file. A symbolic link is like a shortcut to the file. It does not link to the data, it instead has the address of the file that contains the data. The following command would create a symbolic link called new2.txt that points to the file old.txt.

```
ln -s old.txt new2.txt
```

If you were to perform a directory listing and look at only those three files in long format then you would see something like the following.

```
lrwxrwxrwx 1 alice alice 7 Nov  7 16:49 new2.txt -> old.txt
-rw-rw-r-- 2 alice alice 0 Nov  7 16:48 new.txt
-rw-rw-r-- 2 alice alice 0 Nov  7 16:48 old.txt
```

Looking at the original file **old.txt** and the new hard link called **new.txt** you can see that both have the number 2 in the column for links after the permissions. Additionally, they have the same date and time. If you were to use the **touch** command to change the timestamp on old.txt then the stamp that shows for new.txt would also change. If you used the **chmod** command to change permissions for one of the hard linked files then both would change.

The symbolic link is different. The symbolic link has its own permissions, link count, and timestemp. If you delete the original file old.txt then the symbolic link would still point to the file old.txt, even though it would be gone.

To remove links you can use the **rm** command or the **unlink** command.

## Output/Input Redirection

You can redirect output and input at the bash shell. Normally input to programs comes from standard in (**stdin**) which has a file descriptor symbolic link **/dev/stdin** which points to **/proc/self/fd/0**. When you use the print or echo commands in programs the output is usually sent to standard out (**stdout**) which has a file descriptor symbolic link **/dev/stdout** which points to **/proc/self/fd/1**. When programs print errors, they are usually sent to standard error (**stderr**) which has a file descriptor symbolic link **/dev/stderr** which points to **/proc/self/fd/2**.

To redirect standard output from a program you can use the greater than character "**>**" and follow it with a file name. The following command would print the string "**Hello World**", but would redirect the output to the file **hello.txt** instead of printing it on the screen.

```
echo Hello World > hello.txt
```

This is a quick way to create small files. If you want to create a slightly larger **message.txt** text file and feel you can do so without making any mistakes you could use the following command.

```
cat > message.txt
```

When you are done typing you can press the **Ctrl-d** key combination to end input. This would tell the **cat** command that you are done and the **cat** command would end. When the cat command ends the redirection would end and the file **message.txt** would be closed.

Sometimes you run a command that produces a lot of error text. When you are in this situation is it often desirable to redirect the standard error output, but you are not interested in what was printed. To completely ignore the standard error output it is common to send it to **/dev/null**. The **/dev/null** device is a file that does not have any contents and never will.

You can redirect standard error using the number 2 for the file descriptor followed by the greater than character "**2>**" and the place you want to put the standard error output. The following command would redirect all error messages and just display the standard out data from the grep command.

```
grep -i hostname /etc/* 2> /dev/null
```

Sometimes you want to write scripts that perform things without your involvement. To get the command line input from a file instead of a keyboard, use the less than character "**<**". For example, if you wanted to create a script that would automatically erase all partitions on a device, you might create a text file **fdisk.txt** with the following data.

```
d
4
d
3
d
2
d
w
q
```

The file would delete each partition, write the partition table, then quit. You could then use this script with the **fdisk** command to destroy the partition table of the **/dev/sda** device using the following command.

```
fdisk /dev/sda < fdisk.txt
```

## *Piping Output*

Redirecting output and input are great, but sometimes you want to take the output from a given command and pass that in as the input for a different command. In these situations you can redirect the output to a file, then redirect the input so it comes in from the file. Two commands works, but you can do it directly on a single line in a single command using the pipe character "|".

The following is an example of using the **find** command to search for file names in the /etc/ directory tree, then pass them through the **grep** command to search for the "**passwd**" string.

```
find /etc/ | grep passwd
```

If you have multiple processes you want to run the data through, you can just append them with the pipe command.

```
find /etc/ | grep passwd | sort
```

The pipe character works great, so you can probably just use it and be happy. However, there is another method that you can use to redirect to a process like the pipe character. You can redirect using the greater than character "**>**", but you do not have to redirect to a file. You can use a second greater than character after a space to a process listed within parenthesis ( ). You can also write the above command using the following syntax.

```
find /etc/ > >(grep passwd)
```

You might note that there are a lot of errors that display on the screen. You can redirect standard error using the "**2>**" character combination to a file or through a process.

```
find /etc/ > >(grep passwd) 2> /dev/null
```

You also have the option of taking the standard error and passing it through a process. Unfortunately, there is no "**2|**" character combination. You have to use this second method. The following would just perform word count using the **wc** command on the standard error. The standard out is being thrown away.

```
find /etc/ 2> >(wc) > /dev/null
```

## *File Permissions*

File permissions are an important part of any UNIX-like environment. You have read, write, and execute permissions for each set of users, the owner, the files group, and everyone else on the system. If you take a long listing of a directory you can see what these permissions are set to:

```
bash$ ls -al
total 48
drwx------    4 bob      bob          4096 Feb 26 10:33 .
drwxr-xr-x    5 root     root         4096 Feb 26 10:33 ..
-rw-r--r--    1 bob      bob            24 Feb 26 10:33 .bash_logout
-rw-r--r--    1 bob      bob           224 Feb 26 10:33 .bash_profile
-rw-r--r--    1 bob      bob           124 Feb 26 10:33 .bashrc
-rw-r--r--    1 bob      bob          5450 Feb 26 10:33 .canna
drwxr-xr-x    2 bob      bob          4096 Feb 26 10:33 Desktop
```

```
-rw-r--r--    1 bob      bob           747 Feb 26 10:33 .emacs
drwxr-xr-x    3 bob      bob          4096 Feb 26 10:33 .kde
-rw-r--r--    1 bob      bob          3728 Feb 26 10:33 .screenrc
-r--r--r--    1 bob      bob          1019 Feb 26 10:33 .wl
```

In this directory the file ".emacs" has a string of 10 characters at the beginning of the line. The first character is where you would have information about what type of file this is. The "d" character stands for directory. After the first character there are nine characters left. These nine characters are split into three groups of three each. The first three characters are the permissions for the owner. The next three are the permissions for the files group owner, and the last set of permissions are the permissions for everyone else on the system. (Everyone else on the system includes the users like apache the web server.)

Each set of three characters are split into three permissions read, write, and execute. These are marked with a letter if the permission is available or marked with a dash if the permission is not set.

## Command: chmod

The **chmod** command can be used to change the permissions of files on your system. Because learning the chmod command is so important, we will be talking about it in more detail in the following sections as well.

Basically, the chmod command takes two arguments one for the permissions and one for the file. The permissions are expressed in either a text readable format or in the octal representations of the binary values that are actually stored in the file system.

```
chmod PERMISSIONS FILE(S)
```

Some people are more comfortable using the text representations for the file permissions and some are more comfortable using the octal/binary representations. Continue reading and decide what makes more sense to you.

## Command: chown

In addition to permissions, you sometimes want to change the ownership of a given file. To change the ownership you can use the **chown** command. To change the ownership of a single file we just pass the owner's user name and the file name as arguments.

The following command will change the ownership of a file **/tmp/information.txt** to being owned by the user **charles**.

```
chown charles /tmp/information.txt
```

Now, sometimes we want to change both the user and the group ownership for a file. To change the

file to being owned by both the user charles and the group charles we could type the following command.

```
chown charles:charles /tmp/information.txt
```

The first charles in the above command is the user name and the second one if the group name.

This command needs to be used in situations where the root or apache users create files in places where they should not be creating files. It is sometimes easiest to just change the ownership to the owner of other files in the same directory.

If you find that you need to change not only a single file, but all files in a directory or even all files in a directory tree then you can use wildcards and/or the -R switch for recursion. If you were in the home directory for the user dave and wanted to make sure all files in that directory and all sub-directories were owned by dave you could use the following command.

```
chown dave:dave -R /home/dave
```

This command is powerful and can make a mess of your system, so make sure you are in the right place and are using the correct command when you use wildcards or use recursion.

## Read Permissions

Read permissions allow the user to view the contents of the file. If you have read permissions, but you do not have execute permissions you can copy the file into another directory and change the permissions of the file. You can also change the permissions of the file if you have permissions for that directory. You can give everyone read access to the file by typing:

```
chmod a+r text1.txt
```

You can give permissions to only the user, group, or others by using “u”, “g”, or “o” instead of “a”. This is what you would do to give read permissions to the group:

```
chmod g+r text1.txt
```

If you wanted to take away everyones read permissions you would replace the plus sign with a minus sign:

```
chmod a-r text1.txt
```

You can also set the file by typing the binary values of each bit. We will talk about that later in this

section.

## Write Permissions

Write permissions allow the user to write information to the file. The user may or may not have permissions set to view the contents of this file. If you are creating a log that you do not intend others to read, but you would like them to be able to add to you could create a write only file. If you wanted people to be able to read the file, but not modify it you could take away the write permissions and make it a read only file. Note that if a user has permissions to write to a file, then the user can erase the file also. To give write access to everyone type:

```
chmod a+w text2.txt
```

If you wanted to take away everyone's write permissions you would replace the plus sign with a minus sign:

```
chmod a-w text2.txt
```

## Execute Permissions

Execute permissions give the user the right to run the code in the file. This does not always allow the code to be run. If the code is a shell script then the user needs to have read permissions in order to run the program. This gets a little bit complex. Usually execute permissions go with read permissions for executable programs. Directories also have an execute bit set if you are allowed to enter the directory. You may enter, but not see anything there unless the read bit is set. To give execute permissions to a file type:

```
chmod a+x text3.txt
```

If you wanted to take away everyone's execute right you would replace the plus sign with a minus sign:

```
chmod a-x text3.txt
```

I hope that you are beginning to see a pattern. "chmod" is not really that hard to use.

## Expressing Permissions in Binary

The three sets of three characters are really 9 binary bits that mark 1 for yes you have the right and 0 for no, you do not. These bits are displayed in octal. If you do not know what that means then just assume that they are represented by three digits from 0 to 7, each digit representing a different set of

three bits. If you wanted to give read, write, and execute rights you could represent that in binary as 111 or in octal as 7. To convert from octal assume that the first bit stands for 4, the second for 2, and the last for 1. If you can do this then you will be able to figure out what permissions to use. For a file that is read only for everyone you would use 444. For a file that was write only for everyone you would use 222. If the file was execute only for everyone the permissions would be set to 111. If you want the owner to have read, write, and execute and everyone else to have read and execute rights you would use 755. Web pages would usually be set to 644 so that the owner can read and write, but everyone else can only read them:

```
chmod 644 index.html
```

For CGI files, which are actually scripts that are executed by the web server and display the HTML contents of a web page, you would have to use 755 so that the world (which includes the Apache user) can read and execute them:

```
chmod 755 results.cgi
```

I mentioned that there are only 9 binary bits to set permissions with, but that is not exactly correct. There are other bits that you can use to do things like setuid and setgid. With setuid the executable runs as the owner. With setgid the executable runs as the group owner. This can be accomplished by putting either a 4 (for setuid) or a 2 (for setgid) in front of the other permissions. If you wanted "useradd" to run as root, allowing anyone to create users you could change permissions like this:

```
chmod 4755 /usr/sbin/useradd
```

If you then looked at the directory in list format, the line with "useradd" would look something like this:

```
-rwsr-xr-x    1 root  root   52348 Mar  9  2015 /usr/sbin/useradd
```

Notice the "s" instead of the "x" in the first three letters. The "s" stands for setuid.

## *Text Editors*

There are three text editors that I think you should know about. Vi/Vim, Nano/Pico, and Emacs are all well known editors. They are available on Linux distributions, various versions of UNIX, and even other well known operating systems. Text editors are sometime hard to learn, but can be very powerful and even necessary for system administrators.

## Command: vi

The vi editor is one of the oldest and most available editors around. You can find vi on almost any and all UNIX and Linux systems.

Vi has multiple different modes. You start in command mode. From here you can enter text mode or do things like save or edit your document. In the vi editor you must move out of command mode and into insert mode before editing.

```
vi textfile.txt
```

Once in the editor, here are a few command mode incantations:

```
:w                  Save the file
:q                  Exit vi
:wq                 Write the file and quit
:q!                 Quit without saving
x                   Delete the current character
i                   Enter Insert Mode
a                   Enter Insert Mode after the current character
```

When you are in insert mode you can exit and return to command mode by pressing the Escape (ESC) key.

## Command: nano

The University of Washington created a mail reader called pine. With it they also created a text editor called pico. Pico was easy for starters to get used to.

Unfortunately, the University of Washington decided they wanted to aggressively protect their intellectual property, so their software became less compatible with open source environments. People who still wanted to use pico created an alternative called **nano** which looked and felt like pico.

Older Linux distributions tend to have the **pico** editor and newer ones seem to have **nano**. Unfortunately, the Red Hat family decided to go back to Vi and do not usually have nano installed by default in the minimal install. Fortunately, it does appear in the GNOME Desktop install.

```
nano textfile.txt
```

Here are some of the options in nano:

```
Ctrl-o              Save the file
Ctrl-x              Exit nano
Ctrl-k              Cut a line
Ctrl-u              Paste cut lines
Ctrl-t              Spell Checker
```

There is one point that I probably mention about nano so that you are not surprised. When you are editing files with long lines, nano has the habit of splitting lines to make sure they are easier to read. If the file is a configuration file, this might be a bad thing. If you want nano to resist the urge to split long lines you can use the -w option.

```
nano -w sample.conf
```

**Line Numbers**

When you are making changes to configuration files it is common to make small typographical mistakes. The error logs and program crash reports normally try to give you as much information as possible to help you fix the problem. This information normally includes line numbers. If you want the Nano editor to report line numbers, you can turn on the status bar. This is done by editing the **/etc/nanorc** global configuration file and uncommenting the **set const** line.

```
set const
```

## Command: emacs

GNU Emacs is a powerful text editor created by Richard Stallman as part of the free software movement. This editor and vi had religious debates about which was better for a long time.

Since vi is the default editor on Red Hat and CentOS systems today, I would guess that vi won. It also probably helps that vi was widely available on closed source UNIX systems for a long time as well. If you want to try out Emacs, you can install it and give it a try.

```
emacs textfile.txt
```

Below are some of the commands you can use in editing files:

```
Ctrl-x Ctrl-s        Save the file
Ctrl-x Ctrl-c        Exit Emacs
Ctrl-k               Cut to the end of a line
Ctrl-y               Paste cut lines
Ctrl-h t             Emacs Tutorial
```

If you want to learn emacs take the tutorial and have fun. The tutorial is really quite long.

Sometimes, people who use emacs run into an issue where Emacs wants to open up in a graphical window, but there is not graphical environment available. To prevent emacs from running a graphical display you can use the -nw option for no window.

```
emacs -nw textfile.txt
```

### *Other Useful Utilities*

There are many other command line utilities that users regularly use. Here are some of the common ones.

## Command: locate

When you know what a file is called, but do not know where to find it, the **locate** command comes in very useful. It used to be common for the locate command to be installed on a new Linux system, but that is becoming less common. To install and initialize the locate command use the following commands as root.

```
yum install mlocate
updatedb
```

The **updatedb** command takes a little bit of time, but once it completes, you can use the locate command. Here is a dialog of searching for the yum.conf file.

```
bash# locate yum.conf
/etc/yum.conf
/usr/share/man/man5/yum.conf.5
```

It this point it is still up to you to decide which file is the one that you want, but at least you have more to work with.

## Command: find

Sometimes you do not want to install the locate command or do not have the right access rights. In those cases, you can sometimes just use the **find** command instead.

With find you can pass it a directory name and it will list all files, directories, and everything in those directories all of the way down the tree. To see all of the files in the /etc/ directory and sub-directories you could use the following command.

```
find /etc/
```

That will probably give you a lot more files and directories than you can easily sort through, but it should give you everything you are authorized to see.

## Command: grep

The grep command is good for searching the contents of files in a directory or output from a previous command. The above find command listed all files in the /etc/ directory. What if you knew you wanted to find a file in the /etc/ directory that contained the letters “ifcfg”? You could use the following command.

```
find /etc/ | grep ifcfg
```

The find command lists all files and directories in /etc/, but that list is too long. The grep command filters that list. The pipe character which is usually found above the Enter key on your keyboard takes the output from one command and passes it as input to another command. In this case, the output from the find command is being passed as input to the grep command. Here is an example dialog.

```
bash# find /etc/ | grep ifcfg
/etc/dbus-1/system.d/nm-ifcfg-rh.conf
/etc/sysconfig/network-scripts/ifcfg-lo
/etc/sysconfig/network-scripts/ifcfg-ens33
```

In addition to searching for file names from the find command, you can also search for files with contents. Assume you want to change your computer's hostname, but are unsure which file to edit. If your hostname is example.com you could use the following command to search.

```
grep example.com /etc/*
```

I assure you that somewhere in that output would be the answer, but you might need to filter out the error statements. Commands generate standard output and standard error output. Most of that output from that command is standard error. Here is the command with filtering to get rid of the standard error.

```
bash# grep example.com /etc/* 2> /dev/null
Binary file /etc/aliases.db matches
/etc/hostname:example.com
```

If your file contained the name Example.com instead of example.com you would need to use a -i case insensitive switch. Here is the command with insensitivity to case.

```
grep -i example.com /etc/* 2> /dev/null
```

## Command: wget

System administrators regularly find themselves in situations where they need to download a single

file, but do not have a web browser. If you know the URL of the file you want to download, you can often use the **wget** command. You first need to make sure the **wget** command is installed, then you can use it with a URL as the single argument to download the file.

To install the wget program use the following command.

```
yum install wget
```

After wget has been installed you can use it to download pages and packages. If you wanted to download a package called software.1.2.3.tar.gz from example.com, you might use a command like the following.

```
wget http://example.com/software.1.2.3.tar.gz
```

If you find that the file to download is big and you do not have a stable network connection, you might have to download the file with more than one command. When a download stops in the middle, you can cancel the command and restart the download with the **-c** option to continue instead of starting new.

To continue the previous download command you would use something like the following.

```
wget -c http://example.com/software.1.2.3.tar.gz
```

If you needed to download a whole website, you could try using the -r option for recursive. To download everything on example.com you might use a command like the following.

```
wget -r http://example.com/
```

## Command: tar

The **tar** command was originally created to work with backup tapes archives. The idea is to take multiple files and create a single file that contains all of the original files and their locations. Tar files are easy to identify because they have the **.tar** file extension. To create a tar file, you can use the options cvf. The following would create a .tar file called **data.tar** that contained a directory **data/** with all of the sub-directories and files.

```
tar cvf data.tar data/
```

If you then wanted to extract the **data.tar** file you could use the following command.

```
tar xvf data.tar data/
```

You will probably notice that the main difference in the options is that the "**c**" character option was used to create the archive file and the "**x**" character option is used to extract the files.

**Tar with Compression**

The tar command does not compress files. This makes the tar command run faster, but means that more space is consumed. If you wanted to compress files, you can use built-in character options to compress. The two most common compression types on Linux machines are the gzip and bzip2 types.

To create tar archives with the **gzip compression** you need to add the "**z**" character option. Files creates using this option usually have either the **.tar.gz** or **.tgz** file extension. The following command could be used to create a **data.tar.gz** file that contains the **data/** directory.

```
tar cvfz data.tar.gz data/
```

If you then wanted to extract the **data.tar.gz** file you could use the following command.

```
tar xvfz data.tar.gz data/
```

To create tar archives with the **bzip2 compression** you can add the "**jp**" character options. Files creates created using the bzip2 compression usually have the **.tar.bz2** file extension. The following command could be used to create a **data.tar.bz2** file that contains the **data/** directory.

```
tar cvjpf data.tar.bz2 data/
```

If you then wanted to extract the **data.tar.bz2** file you could use the following command.

```
tar xvjpf data.tar.bz2
```

To create tar archives with the **xz compression** you can add the "**J**" character option. Files created using the xz compression usually have the **.tar.xz** file extension. The following command could be used to create a **data.tar.xz** file that contains the data/ directory.

```
tar cvfJ data.tar.xz data/
```

If you then wanted to extract the **data.tar.xz** file contents you could use the following command.

```
tar xvfJ data.tar.xz
```

## Commands: gzip / gunzip

While **gzip** compression is usually performed using the **tar** command, it is possible to do it without using the tar command. The gzip program does not create archives and only compresses and uncompresses individual files. Files compressed with the gzip compression have the **.gz** file extension. If you had a file called **data.tar** and wanted to compress it using **gzip**, you could use the following command.

```
gzip data.tar
```

The resulting file would be called **data.tar.gz** and the original file would no longer be present. To uncompress the file, you could use the following command.

```
gunzip data.tar.gz
```

## Commands: bzip2 / bunzip2

The **bzip2** utility was created to work like the **gzip** utility. It is usually considered better compression, but is a little newer than the gzip compression, so it is not as common with older projects. Files compressed with the bzip2 compression have the **.bz2** file extension. If you had a file called **data.tar** and wanted to compress it using **bzip2**, you could use the following command.

```
bzip2 data.tar
```

The resulting file would be called **data.tar.bz2** and the original file would no longer be present. To uncompress the file, you could use the following command.

```
bunzip2 data.tar.bz2
```

## Commands: xz / unxz

The **xz** utility was also created to work like the **gzip** utility. Files compressed with the xz compression have either the **.xz** file extension or the **.lzma** file extension. If you had a file called **data.tar** and wanted to compress it using **xz**, you could use the following command.

```
xz data.tar
```

The resulting file would be called **data.tar.xz** and the original file would no longer be present. To uncompress the file, you could use the following command.

```
unxz data.tar.xz
```

## *CLI Special Keys and Combinations*

When you are using the Linux command line there are times when you want to use a command, but are not sure what the command is or how to use it. Fortunately, the Bash shell helps users who are new to Linux and other UNIX-like systems.

## Tab Completion

When you are typing at the terminal, you can press the tab key and the bash shell will attempt to complete the command, directory, or file name. If you are typing a command and you do not know how to complete the command, you can press the tab key once to and the terminal will complete as mush of the command as is obvious. If there is only one possible command then it will display that command. If there are more than one command you can press the tab key again to see what the options are. The following is a dialog for commands that start with the letters "user".

```
bash# user[tab][tab]
useradd     userdel     userhelper  usermod     usernetctl  users
```

The tab completion indicates that there are six commands that start with the letters "user". If you were to add an additional letter such as the letter "a" and press tab, there would only be a single command that would work, so the terminal would complete the command "useradd".

Tab completion can help you when you are typing long file names so that you do not make as many mistakes and it is easier to remember. Assume for a moment that you wanted to see the contents of the messages log file, but you were not sure if the file was messages or message. You could type the following to see what possibilities were.

```
bash# cat /var/log/m[tab][tab]
maillog   messages
```

At this point you would know the file was messages and would be able to add the letter "e" and press the tab key once again to complete the file name.

## Stop (Ctrl-s) and Continue (Ctrl-q) Output

In the early days of UNIX-like machines it was common to want to stop the output of a command so that you would have the option of reading it. This is great, but it can catch a few new users by surprise. The command to stop the output is **Ctrl-s**. To continue the output again you can use **Ctrl-q**. The problem occurs when a new user is at the terminal and they instinctively press the Ctrl-s key combination because they like to save their document. The output stops and it seems like the terminal window is suddenly frozen. Users might think that their terminal window has crashed and do something that they do not want to do.

Once someone explains the Ctrl-s and Ctrl-q combinations and their symptoms users might add the Ctrl-q key combination to their troubleshooting steps to solve the frozen terminal window problem.

To practice this command, type in a command that produces a lot of output, then use the Ctrl-s and Ctrl-q key combinations to practice stopping and starting the output. Press the **Ctrl-c** key combination to break out of the command and return to the prompt.

```
find /
[Ctrl-s]
[Ctrl-q]
[Ctrl-c]
```

## Suspending a Process (Ctrl-z)

Some processes take a long time to complete. Since you do not get the terminal back until the command completes, you usually just wait. The **Ctrl-z** key combination makes it possible to get back to the shell without breaking out of the process. This key combination actually stops the command and removes it from the list of processes getting CPU time.

**Jobs**

You can then get a list of processes running in your terminal using the **jobs** command. Each process in the list has a number, a status, and the command line used to start it. Below is a dialog showing the Ctrl-z command and the jobs.

```
bash$ sleep 60
[Ctrl-z]

[1]+  Stopped                 sleep 60
bash$ jobs
[1]+  Stopped                 sleep 60
```

Once a process has been stopped this way, it does not continue until you tell it to start up again. The sleep command above was going to sleep for 60 seconds, then it would return to the terminal.

**Foreground**

To return to the command you can use the **fg** command. The fg command returns the command back to the foreground. If there is more than one stopped process or job, you can use the number of the job to return to it. The previous job was number 1, so to return to number 1 we could use the following command.

```
fg 1
```

The command might immediately complete if 60 seconds have passed, or it will continue to wait until the rest of the 60 seconds are up before ending.

**Background**

If you decide that you want the terminal back, but also want the command to run in the background, you can use the **bg** command. Like the fg command, the bg command can also have the job number passed to it. To continue the previous command in the background, use the following command.

```
bg 1
```

## Starting in the Background (&)

In addition to sending a process to the background using the **bg** command, you can also start a process in the background. To start a process in the background, use the ampersand "**&**" after the command. The following two command (combinations) have the same effect.

```
sleep 60 &
```

```
sleep 60
[Ctrl-z]
bg
```

Once a process has been started in the background, it will continue to run and will also send output to the terminal. Because most users who start background processes do not want to see the results, they usually redirect the output somewhere using the greater than "**>**" redirection symbol. The following command would create a file with all file names on the system listed in it. Standard error is sent to /dev/null to avoid seeing it using the "**2>**" redirection symbols.

```
find / > allfiles.txt 2> /dev/null &
```

## *Root Access / Super Users*

On every Linux system, you have a root user account. The root user has full access to the file system, the networking stack, and a few other things. In addition to the root user, there are power users who assist root by running the commands normally reserved exclusively for root.

There are a couple of ways that a standard user can elevate privileges to the root level. The first option is to switch user accounts. The second is to run the sudo command and get root privileges for a single command.

## Command: su

The **su** command switches to another user account. The following is the basic syntax for switching into the root user with the root user's environment.

```
su -
```

The su command tells the system you want to switch. The minus sign indicates you want the environment of the user. After the minus sign, you can have an additional argument of the user name, but in this case it was left blank. When left blank, the assumption is that you want the root user.

If you want to remain in the same directory and have the same environment, but have root access, then you could use the su command without any arguments.

```
su
```

Sometimes when you have root privileges you want to drop down into another user account and run commands as that user. If root wanted to switch to the **alice** user, one of the following commands would be executed.

```
su alice
```

```
su - alice
```

## Command: sudo

The **sudo** command is used to execute a command as a different user. Normally, people use the command and just execute as root, but you can execute commands as a different user as well.

In order to execute commands as root using the sudo command, you to be authorized. The authorization comes in a two step process. First, you normally need to be added to the **wheel** group. You can see which groups you are a member of by looking at the **/etc/group** file or by using the **id** command. The following command shows which groups a user alice is part of:

```
id alice
```

Once the user is part of the wheel group, you need to make sure the **/etc/sudoers** file lists the wheel group as authorized to run commands. There is a line in the **/etc/sudoers** file that may or may not be commented out. Make sure the following line is not commented out with a hash mark:

```
%wheel ALL=(ALL) ALL
```

At this point, you should be able to execute commands as root. The following is an easy command that tests sudo configurations (as long as it is run by the non-root user that is a member of the wheel group):

```
sudo ls /root
```

# Networking

Networking on any modern operating system is very important. Linux is probably most famous for being a good operating system for running networking services.

## *Configuring Networking*

You can configure the network in a couple of different ways. You can use the GUI applets to configure the network, you can configure the network using the text line utilities, you can configure it using the network scripts, and you can manually configure the network by using commands listed in the viewing configuration section.

## Configuring the Network using the GUI

After logging into a GNOME Desktop you have the option of making configuration changes. In the upper right hand corner of the desktop, there is a power button icon and possibly other icons that look like they are related to settings. When you use the left mouse button over one of those icons, you can usually bring up a drop down menu showing an icon with two tools crossed.

The crossed tools icon takes you to a window labeled, "**All Settings**". From this window you can make many different configuration setting changes. The networking options are all configured using the "**Network**" icon in the "**Hardware**" section.

The best way to configure the networking is to first, make sure network interface you want to configure is selected on the left hand side. Usually, you will want to make sure "**Wired**" is selected. Then you want to make sure the network is turned off. There is a button you can use to toggle the network status in the upper right hand corner of the window. Finally, click the gear in the lower right hand corner of the network window to bring up the settings window.

Normally, you will want to configure your network interface for IPv4 networking. Select the IPv4 option on the left hand side of the settings screen. On the left you should see a drop down option where you can select "**Automatic (DHCP)**" and "**Manual**". There might be other options, but these are the two that matter.

**Automatic (DHCP)**

If you select the DHCP option, then the interface will attempt to get the IP address and other information by broadcasting and looking for a DHCP server. If a server responds, it should be able to configure the IP address, subnet mask, default gateway, and DNS server information. You also still have the option of configuring a DHCP server manually if you like.

**Manual**

If you select the manual option then you can configure the IP address manually. There are 4 pieces of information you need to know in order to have your networking work properly. You need to know an IP address, a subnet mask, a default gateway, and a DNS server.

Put the IPv4 address in the "**Address**" field, the subnet or network mask in the "**Netmask**" field, the default gateway in the "**Gateway**" field. Then jump down to the DNS section and put IP address of the DNS server in the "**Server**" field.

After you are done either configuring the interface for "**Automatic (DHCP)**" or "**Manual**" configuration, you are ready to apply the settings with the "**Apply**" button. Apply the settings.

When the settings have been applied, you should then turn the interface back on. If you did not turn it off previously, you will want to turn it off, then back on again.

Make sure the IP Address, Default Route (Gateway), and DNS server address are all displayed. If they are all there and the information is correct, you should be done here.

## Configuring the Network using the Command Line

You can configure the networking using the **nmtui** command in a terminal. This command will bring up a Curses interface. Curses is a development library used to provide text based menus you can control using the arrow keys.

```
nmtui
```

The options in the nmtui command all trigger calls to external configuration programs. These programs can be run individually using the **nmtui-connect**, **nmtui-edit**, and **nmtui-hostname** commands.

The **nmtui-connect** command provides an interface for starting and stopping the network interface cards. After making configuration changes, it is usually a good idea to restart the network settings and get the configuration files reloaded.

```
nmtui-connect
```

The **nmtui-edit** command provides an interface for configuring the settings for the interfaces. You can edit fields by using the arrow keys to select them, then editing the text. If you want to change options in drop down menus or check boxes, you can use the space key.

```
nmtui-edit
```

Once everything has been configured you can save and exit. Remember that changing the settings with the nmtui-edit command does not restart the network. To make sure the network has been restarted you can either use the nmtui-connect command or you can issue the following command.

```
systemctl restart network
```

The **nmtui-hostname** command provides an interface for changing your hostname. This command updates the contents of the **/etc/hostname** file.

## Configuring the Network using text files

You can make manual configuration changes using the text files that store the settings. The text files are stored in the **/etc/sysconfig/network-scripts/** directory. Normally, the local loopback interface will be called **ifcfg-lo** and a wired interface would be called something like **ifcfg-ens32**. You can use the **ls** command to see what the interface files are called.

```
ls -l /etc/sysconfig/network-scripts/ifcfg-*
```

### Using DHCP

To set your interface to use DHCP you need to make sure the file has the **BOOTPROTO** variable set to "dhcp".

```
BOOTPROTO="dhcp"
```

If you previously had a statically or manually configured IP address, you will want to comment out any lines that assign a static IP address. The **IPADDR**, **PREFIX**, and **GATEWAY** variables and permutations of them are used to assign addresses. The **DNS** variable is used to assign a DNS server address. You can comment them out with the hash mark.

```
#IPADDR0="10.10.10.10"
#PREFIX0="16"
#GATEWAY0="10.10.0.1"
#DNS1="8.8.8.8"
```

### Manual Static Network Addresses

If you decide you want to manually configure an IP address using the text file, you can edit the same file and make sure you turn off DHCP using the BOOTPROTO variable with a value of "none".

```
BOOTPROTO="none"
```

You will then need to assign IP address, prefix, and default gateway using the **IPADDR**, **PREFIX**, and **GATEWAY** variables. Normally, you would also give them a number like 0. You also need to assign the DNS server address using the **DNS** variable.

```
IPADDR0="10.10.10.10"
PREFIX0="16"
GATEWAY0="10.10.0.1"
DNS1="8.8.8.8"
```

After making changes to either do DHCP or manual configuration, you want to make sure the

**ONBOOT** variable is set to "yes". This variable tells the machine that you want the interface starting automatically at boot time.

```
ONBOOT="yes"
```

Now all you need to do is start or restart your network and see if everything works correctly. The following command can be used to restart your network.

```
systemctl restart network
```

### *Viewing IP Configurations*

Your minimal installation of CentOS will contain the newer network configuration viewing utilities. Because these tools are fairly new they are not listed in most configuration examples on the web. This can be a bit difficult to work around. Here are a few of the basic troubleshooting tools available with a minimal installation.

## Command: ip

You can see a list of your devices using the "**ip addr**" command. Here is a sample of the output from that command:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 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
           valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
           valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:12:34:56:78:90 brd ff:ff:ff:ff:ff:ff
    inet 192.168.219.130/24 brd 192.168.219.255 scope global dynamic ens33
           valid_lft 1746sec preferred_lft 1746sec
    inet6 fe80::20c:29ff:fe56:7890/64 scope link
           valid_lft forever preferred_lft forever
```

You can see that there are two Ethernet interfaces listed above "lo" and "ens33". These are the local loopback and the external interface.

The local loopback interface has two addresses assigned. For all IPv4 communication we can use the inet address/mask of 127.0.0.1/8. For IPv6 communication we use inet6 with an address/mask pair of ::1/128. The ::1 address is just shorthand for the full address which is 0000:0000:0000:0000:0000:0000:0000:0001.

The ens33 interface also has two addresses. The first, inet, is 192.168.219.130/24. The second for inet6 is fe80::20c:29ff:fe56:7890/64. The fe80 prefix indicates that this address is a link-local address and not for Internet routing.

The ip command has additional options that can be used. Sometimes you want to know where your default gateway is located. The "**ip route**" command helps us here. Here is the output:

```
default via 192.168.219.2 dev ens33  proto static  metric 1024
192.168.219.0/24 dev ens33  proto kernel  scope link  src 192.168.219.130
```

The output indicates that our default gateway is 192.168.219.2 and that address is available through the ens33 interface.

## Command: routel

If the "ip route" command is not giving you enough information you can try the "routel" command to list the routing tables. Here is an example of the output from the routel command:

```
         target            gateway          source    proto    scope    dev tbl
        default      192.168.219.2                   static          ens33
 192.168.219.0/ 24                 192.168.219.130   kernel     link  ens33
      127.0.0.0          broadcast       127.0.0.1   kernel     link     lo local
     127.0.0.0/ 8            local       127.0.0.1   kernel     host     lo local
      127.0.0.1              local       127.0.0.1   kernel     host     lo local
127.255.255.255          broadcast       127.0.0.1   kernel     link     lo local
  192.168.219.0          broadcast 192.168.219.130   kernel     link  ens33 local
192.168.219.130              local 192.168.219.130   kernel     host  ens33 local
192.168.219.255          broadcast 192.168.219.130   kernel     link  ens33 local
            ::1              local                   kernel              lo
            ::/ 96     unreachable                                       lo
::ffff:0.0.0.0/ 96     unreachable                                       lo
    2002:a00::/ 24     unreachable                                       lo
   2002:7f00::/ 24     unreachable                                       lo
   2002:a9fe::/ 32     unreachable                                       lo
   2002:ac10::/ 28     unreachable                                       lo
   2002:c0a8::/ 32     unreachable                                       lo
   2002:e000::/ 19     unreachable                                       lo
   3ffe:ffff::/ 32     unreachable                                       lo
        fe80::/ 64                                     kernel            ens33
        default        unreachable                     kernel              lo unspec
            ::1              local                       none              lo local
fe80::20c:29ff:fe56:7890     local                       none              lo local
        ff00::/ 8                                                        ens33 local
        default        unreachable                     kernel              lo unspec
```

Once again, the information we wanted about the gateway is right here, but there is a lot more information that is presented. The target of "default" has a gateway of 192.168.219.2 which is our default gateway. This is where we can stop, but then we see the rest of the routes loaded into the routing table. In reality, there are multiple routes to different places and we want to use the best route. If we were to send traffic to our local networks we would not need to send anything to the default gateway, but those routes have to be listed in the routing table, even though they feel like common sense. Some of the routes listed are for local networks.

## Command: routef

If you wanted to flush the route tables you can use the "routef" command. This will drop some of the routes including the default gateway. If you are directly connected you will not likely notice anything until you try to connect outside of your local networks.

### *Hostnames*

The hostname of a machine can either not be important or can be very important depending on what it is used for. Some servers require a hostname in their configuration file and also require that the configuration file match the actual hostname of the machine. For this reason and for keeping track of machines is it a good idea to set a hostname for the machine early.

## Command: hostnamectl

A newer command for setting the hostname is **hostnamectl**. You can use the command without any arguments to get information about the system, including the hostname.

```
[root@example ~]# hostnamectl
   Static hostname: example.com
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 783c87bf36884280a0dc2cc660c23a89
           Boot ID: 50a682c409d74620833d86c1933f711e
    Virtualization: vmware
  Operating System: CentOS Linux 7 (Core)
       CPE OS Name: cpe:/o:centos:centos:7
            Kernel: Linux 3.10.0-123.el7.x86_64
      Architecture: x86-64
```

Once you decide to set the hostname, you can use the same command with the **set-hostname** option. This will change both the **/etc/hostname** file and the active hostname of the system.

```
bash# hostnamectl set-hostname newhost.example.com
```

## Command: hostname

You can display the hostname for your machine using the hostname command. The following is an example dialog showing the hostname command without any arguments:

```
bash$ hostname
example.com
```

There are two older methods for setting your hostname on the command line. If you want to just set your hostname in memory, you can use the hostname with a single argument of the new hostname:

```
hostname newhost.example.com
```

Unfortunately, when the machine reboots the hostname will be lost. To set the hostname permanently you need to put it in a configuration file. Normally, you just put a single line in the /etc/hostname file. The single line is the new hostname you want. You can use nano or another editor to change the contents of the hostname file:

```
nano /etc/hostname
```

The following dialog shows the contents of the /etc/hostname file when viewed with the **cat** command:

```
bash$ cat /etc/hostname
newhost.example.com
```

## Service: network

If you make changes to configuration files or mess up the network in other ways such as flushing the routing tables, you can restart the network to get the configuration file information reloaded.

It used to be that everyone used the "/etc/init.d/network restart" command to restart the network. Then people started using the "service network restart" command. Now with the change to SystemD we are using the "systemctl restart network.service" command.

If you use either of the earlier commands, they still work with CentOS 7, but will likely have problems working in the future. Here is the command again:

```
systemctl restart network.service
```

After restarting your network, it is good to immediately check to see if the settings took correctly. You can use the "ip addr" command or use an older troubleshooting command like "ping" to see if you can still communicate out.

## Directory: /etc/sysconfig/network-scripts

Your basic IP address and gateway information is stored in files in the /etc/sysconfig/network-scripts/ directory. A quick directory listing should indicate multiple files.

## File: /etc/sysconfig/network-scripts/ifcfg-lo

The local loopback interface configuration file will be called ifcfg-lo. The following is as example of the contents of that file:

```
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

Each line contains a variable name and value pair. This file is a lot less complex than the files for external interfaces. Most of the information makes sense to normal network administrators. The variables that are not networking are DEVICE, ONBOOT, and NAME.

DEVICE indicates the name of the device. This should match the portion of the filename after the "ifcfg-". The ONBOOT variable is not as clear as the name indicates. When the network is restarted, the device will also start/restart if the value is set to "yes". If the value is set to "no" then the device will not start automatically. At boot time the network service is usually started, so, in a way, this variable name is mostly accurate. The NAME variable indicates the name of the interface that is in a more human readable format.

## File: /etc/sysconfig/network-scripts/ifcfg-enXXX

When Linux distributions switch to using systemd, they give up the old eth0 names, but get a more predictable naming scheme. On older Linux systems there were times when a simple kernel change would change the order of ethernet devices being loaded.

The following is an example of the contents of the ifcfg-ens33 file:

```
HWADDR="00:12:34:56:78:90"
TYPE="Ethernet"
BOOTPROTO="dhcp"
DEFROUTE="yes"
PEERDNS="yes"
PEERROUTES="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_PEERDNS="yes"
IPV6_PEERROUTES="yes"
IPV6_FAILURE_FATAL="no"
NAME="ens33"
UUID="2a15e489-a343-48cf-a9c6-d8d493d2d207"
ONBOOT="yes"
```

As you might have guessed, this configuration is for a dynamic address instead of static. The static addresses have same different keywords. Here is an example static configuration:

```
TYPE="Ethernet"
BOOTPROTO="none"
```

```
DEFROUTE="yes"
IPV4_FAILURE_FATAL="yes"
IPV6INIT="no"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_PEERDNS="yes"
IPV6_PEERROUTES="yes"
IPV6_FAILURE_FATAL="no"
NAME="ens33"
UUID="2a15e489-a343-48cf-a9c6-d8d493d2d207"
ONBOOT="yes"
HWADDR="00:12:34:56:78:90"
IPADDR0="10.10.10.10"
PREFIX0="16"
GATEWAY0="10.10.0.1"
DNS1="8.8.8.8"
DOMAIN="example.com"
```

The first thing to note is that the BOOTPROTO for dynamic and static is a different value. Additionally, the static configuration contains actual configuration information.

### *DNS Name Resolution*

Name Resolution is required to have a normal networking experience. While it is technically possible to do many networking tasks without DNS name resolution, it is difficult.

## File: /etc/resolv.conf

Information about which DNS servers you are using and what your default search domain is are listed in the /etc/resolv.conf file. You can view the contents of the **/etc/resolv.conf** file using the **cat** command:

```
cat /etc/resolv.conf
```

Here is an example /etc/resolv.conf file:

```
#Some comments
nameserver 8.8.8.8
search example.com
```

There are three lines in the file. The first line is an example comment. You can add as many comments as you like as long as you start them with the hash mark.

The second line indicates the IP address of the DNS server that you are using. You can have multiple nameserver lines each with the IP address of a DNS server. When resolving names, the computer will try using the first nameserver first, but if that server cannot be reached, it will proceed to the next one in the file. Once a nameserver is reached, it will ask that name server for resolution. If the nameserver it talks to does not know about the hostname or IP address you are trying to resolve, it will not ask any

additional servers listed in the resolv.conf file.

The last line indicates what your default search domain is. If I wanted to look up www.example.com, I could use either www or www.example.com because if the domain is missing, the computer will attempt an additional search using the search domain listed in the resolv.conf file.

### *DNS Records*

The Domain Name Service or DNS is a hierarchical system of records stored in a distributed database. We start at the root DNS servers as the dot "." zone and work our way through the top level domains (TLDs) to the domains, then sub-domains. Eventually we get to the host records.

Each of the record types is important and serves a purpose, but most users never need to understand what is happening behind the scenes.

Normally, when you type a URL into a web browser, your computer browser extracts the hostname from the URL and asks it's DNS server to look the name up. At this point I am going to explain this assuming no DNS servers involved have any stored cache. If any data is cached, the process gets shorter.

Your DNS server will look up a list of root servers and ask one of them for the A record for the hostname. The root server will then respond by telling you to talk to the servers in charge of that TLD and will give you a list of servers that serve that TLD with their A and NS records.

Your DNS server will then ask one of the TLD servers for the hostname. The TLD server will respond with a list of servers assigned to administer the domain that the hostname is a part of and will give you A records and NS records for those DNS servers administering the domain.

Your DNS server will then contact a server administering the domain and ask for the A record for the hostname. If the server knows the A record, it will respond with the A record. If the A record is part of a sub-domain, it might send your server along with the NS records for the sub-domain and A records for servers administering that sub-domain.

Eventually, your DNS server will probably get the A record and will return it to you. If it fails, it will let you know. The A record will tell your computer what the IP address is of the computer in the URL.

With the IP address, your browser will try to connect to the web server and will tell the remote server the hostname and path of the web page it is requesting. Hopefully, at this point, after all of this work, you will get the contents of the web page.

## DNS Record Types

A Records: These are records that convert names into IPv4 addresses.

AAAA Records: These are records that convert names into IPv6 addresses.

PTR Records: These are records that convert IP addresses into names. It is a little bit more complex than this because the IP addresses are pretty complex and are listed in order where the largest grouping is the piece closest to the right. For example, the IP address 10.11.12.13 would be listed as 13.12.11.10.in-addr.arpa.

NS Records: These records are for assigning responsibility for a TLD, domain, or sub-domain to a

name. These records translate a name into another name. For example, the NS record for example.com. might point to dns.example.com.

MX Records: These records list the mail exchange server for a given domain. There can be multiple mail exchange servers and each is listed with a priority number with the lowest priority being the server you are supposed to talk to. For example, the MX records for example.com. might be mail.example.com. with a priority of 10 and mail2.example.com. with a priority of 20.

TXT Records: These records are used for various things. Originally, they were probably just used for comments, but have recently been used to identify the servers authorized to send mail on behalf of a client domain. For example, SPF records can indicate that 10.10.10.10 is authorized to send mail for example.com.

### *CLI Tools*

DNS resolution can easily be done using the BIND utilities dig and nslookup. To use these utilities you need to have the bind-utils package installed.

```
yum install bind-utils
```

## Command: nslookup

The nslookup command provides a powerful, yet still simple interface for making DNS queries. Assuming you were using the Google name server 8.8.8.8 that is listed in the resolv.conf a little earlier in this, using the nslookup command with www.example.com as an argument would produce the following results:

```
bash$ nslookup www.example.com
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   www.example.com
Address: 93.184.216.34
```

The server and the address listed immediately in the returned text are the address and port number of the DNS server performing the query. The “Non-authoritative answer” is the results of the actual query. This tells use that the DNS entry for www.example.com. is the IP address 93.184.216.34.

What nslookup does not tell us is that the record we requested is an A record and that this is the results of that record. Most users, however, are not concerned with the type of record that was returned.

Performing a reverse DNS query is very easy with nslookup. Here is a query using the IP address as the argument instead:

```
bash$ nslookup 93.184.216.34
Server:         8.8.8.8
```

```
Address:        8.8.8.8#53

** server can't find 34.216.184.93.in-addr.arpa.: NXDOMAIN
```

Once again, the DNS server is 8.8.8.8, but the hostname tied to the IP address is not listed. In the error message, it says something interesting. It says it cannot find 34.216.184.93.in-addr.arpa. This kind of looks like the IP address, but the number is backwards and it has other words attached.

When you perform an nslookup command on the IP address there are a couple of things to note:

- You are performing a PTR record query
- IPv4 addresses are in the in-addr.arpa. domain
- The records always have the largest grouping on the right

If instead of having nslookup automatically translate the IP address into a PTR record query you wanted to do it yourself, you could use the following command:

```
bash$ nslookup -type=PTR 34.216.184.93.in-addr.arpa
Server:         8.8.8.8
Address:        8.8.8.8#53

** server can't find 34.216.184.93.in-addr.arpa: NXDOMAIN
```

If you wanted to look up other record types you can just list them on the command line as the type value.

## Command: dig

The dig command is like the nslookup command, but it forces you to do a little bit more work for anything more than an A record. Here is the results of the simple query for the A record of example.com:

```
bash$ dig example.com

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-25.P1.el5_11.2 <<>> example.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53219
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 4

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            79254   IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.com.            79254   IN      NS      b.iana-servers.net.
example.com.            79254   IN      NS      a.iana-servers.net.
```

```
;; ADDITIONAL SECTION:
b.iana-servers.net.     1645    IN      A       199.43.133.53
b.iana-servers.net.     110518  IN      AAAA    2001:500:8d::53
a.iana-servers.net.     1645    IN      A       199.43.132.53
a.iana-servers.net.     110518  IN      AAAA    2001:500:8c::53

;; Query time: 3 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Sep  8 10:16:19 2015
;; MSG SIZE  rcvd: 181
```

The results of the command told you which server you used and even told you what the IP address is, but a lot more is displayed.

All of the results are in a format that can be used directly by the BIND software. In BIND, the semicolon is used to indicate comments. Every line that starts with a semicolon can be read and understood, but might not translate directly into a BIND command.

In the question section we have the commented line that indicates that we were looking for the A record for example.com.

The answer section tells us that the A record for example.com. is 93.184.216.34 and that we have 79254 seconds until that record expires. When the record expires, your DNS server is supposed to forget about the entry and query the servers again if it needs another copy.

The authority section tells us who the two name servers for the example.com. domain are. The first one listed is the one that we most likely got the information from, but that is not always the case.

The additional section tells us information about connecting to the name servers. We can see that we are given both the A and AAAA records for the name servers. The A records are the IPv4 addresses and the AAAA records are the IPv6 addresses.

The last commented section tells us information about the query itself. We can see the query took 3 milliseconds. We got our information form the 8.8.8.8 DNS server talking to port 53. We can also see a date stamp and the message size.

Performing reverse look ups is not as simple as just using the IP address as the argument for the dig command. You can either request the PTR record or you can tell the command that you want to have dig translate the IP address into the PTR and perform the look up.

The following are the command line commands to perform the reverse look up. The results are not given.

```
dig -x 93.184.216.34
```

```
dig -t PTR 34.216.184.93.in-addr.arpa.
```

Once again, as with the nslookup command, you can look up other record types as well.

## Command: host

Sometimes you want something even easier than nslookup or dig. The host command is pretty easy to use. Using our www.example.com hostname we will perform a look up to see what the host command returns as results.

```
bash$ host www.example.com
www.example.com has address 93.184.216.34
www.example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

This command does not tell use where the information came from. Additionally, it is not just looking for the A record, it also looks up the AAAA record and tells us the results of both.

As with the nslookup and dig commands, you can also perform a reverse look up. Here are the results of a reverse lookup:

```
bash$ host 93.184.216.34
Host 34.216.184.93.in-addr.arpa. not found: 3(NXDOMAIN)
```

Okay, the reverse of the IP address still does not have a hostname. If you wanted to look up individual records you can use the -t option like with the dig command. Here we look up the AAAA and A records to show the results:

```
bash$ host -t AAAA www.example.com
www.example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

```
bash$ host -t A www.example.com
www.example.com has address 93.184.216.34
```

## *Old LAN Configuration Tools*

If you have been using Linux or UNIX like systems such as Mac OS X for a while, then you might be familiar with commands such as **ifconfig**. Unfortunately for many people, the ifconfig command has been listed as obsolete and is no longer installed by default on minimal systems.

If you would like to use the ifconfig command, and find you do not have it on your system, you can install the package that provides it. The following command should get you the ifconfig command again:

```
yum install net-tools
```

Note, the ifconfig command is not the same as the **ip** command and is obsolete, so you probably still need to learn how to use the ip command eventually.

# Disk Partitions

In order to fully use a hard disk you need to first initialize it by creating a partition table, create partitions on the disk, then put file systems in the partitions. This whole process used to be pretty simple because there were few ways of doing this.

As operating system and file system technology has developed and advanced, the number of file systems has grown so that there are different file system types for all kinds of needs. Additionally, the size of drives has grown which has caused additional partition table types.

## *Partitioning*

There are two main partition table types out there. The old one is Master Boot Record (MBR) and the new one is GUID Partition Table (GPT). The partition table style you have depends on the operating system you have and the size of the disk drive you use.

Master Boot Record (MBR) allows 4 standard partitions, but can be extended to allow one to be marked as an extended partition which allows further divisions into logical partitions. Each partition has a start and stop place and has a file system type ID associated with it. Additionally, you can mark partitions with a boot flag that is used by some operating systems to decide what to boot and use.

The major problem with MBR is that it was developed in a day when people had a hard time imagining that hard drives could get so large. Because each partition has a start and stop address stored in memory, the addresses need to fit there. These addresses are a fixed length that does not allow addressing the whole disk if you have a drive larger than 2 terabytes in size.

As the hard drive technology advanced, GPT was developed to address this issue. As GPT was developed to address the MBR limitations, UEFI was also developed to address limitations in BIOS. Because of this, some operating systems require UEFI in order to boot from GPT initialized disks. Fortunately, Linux has GPT support built in and can boot from GPT disks easily.

## *Boot Loaders*

When computers first started booting from hard drives and floppy disks the idea was they would just load the first section into memory and start executing instructions. This worked great when the number of instructions needed to get your machine up and going was small.

As operating systems got bigger, they created simple boot loaders that basically had the single task of loading the operating system from the hard drive and jumping into the newly loaded memory. These boot loaders only needed to locate the operating system data and read it from the file system into memory.

Boot loaders have continued to become more complex as newer partition table styles, file system types, RAID, encryption, and other advances have made everything more complex. Older Linux systems used the Linux Loader or LILO boot loader. Most Linux distributions now use the GNU Grand Unified Boot loader or GNU GRUB.

Boot loaders are normally loaded at the beginning of the disk, but can be put into a partition instead if you have multiple boot loaders in use. It is common to have multiple boot loaders if you have a dual boot system where Linux and another operating system have different boot loader preferences.

### *File Systems*

The file system organizes space and makes it usable. In order for a file system to organize that space, it needs to take some space for itself. File systems consume space keeping track of which parts of the partition are used and available, keeping track of permissions and other file and directory attributes, keeping track of transactions or journaling, and other things.

In addition to the space they consume, they also allocate space in sizes that do not exactly match the file requirements. Any space the file system has allocated to a file that is not used is still allocated and unavailable to other files.

Linux has historical roots with the Minix operating system acting as a guide. The main file systems that come with Linux Ext2, Ext3, and Ext4 are all based on the Minix file system. Additional file systems are available on Linux depending on the options that were selected when building the Linux kernel and kernel modules.

For a listing of the file systems your system claims to know how to create, you can use the mkfs commands. The mkfs command is actually a front end program that runs many other programs that start with "mkfs." To get a list of the file systems, type mkfs. Then press the **tab** key twice.

```
bash$ mkfs[tab][tab]
mkfs          mkfs.cramfs  mkfs.ext3    mkfs.minix
mkfs.btrfs    mkfs.ext2    mkfs.ext4    mkfs.xfs
```

Each of these file systems has different characteristics and abilities. For example, the ext2 file system was the default for many years because it just worked. It also had to perform a file system check regularly, but that was okay, right? The problem with ext2 was it had problems when it lost power in the middle of a write. The ext3 file system introduced journals. With a journal the file system keeps track of transactions by marking information about the write before and after it happens. This way it can tell if everything happened according to plans or if it needs to back out of something or complete it.

## Command: mkfs

To format a partition we can use the mkfs command and pass it arguments of the file system and the device. If we wanted to create a basic ext2 file system on /dev/sda2 we would use the following command:

```
mkfs -t ext2 /dev/sda2
```

If we wanted to directly use the formatting tool mkfs.ext2 instead of the shell program we could instead use this command:

```
mkfs.ext2 /dev/sda2
```

## Command: tune2fs

If, instead of formatting a file system you want to convert an Ext2 file system to Ext3 by adding a journal, this can be accomplished using the **tune2fs** command. The tune2fs command is used to query information about the Ext2/Ext3/Ext4 family of file systems and to make changes.

You can see a full list of the options made available by tune2fs by looking at the manual page.

```
man tune2fs
```

The most common action that requires the **tune2fs** command would be to take an Ext2 file system and add a journal to make it an Ext3 file system. If you wanted to convert a device **/dev/sda1** from Ext2 to Ext3 you could use the following command.

```
tune2fs -j /dev/sda1
```

If you do not know the device, you can sometimes look at the file system table stored in **/etc/fstab**, use the **df** command, or you can use a partitioning utility such as **fdisk** to list the partitions.

## Command: fdisk

To make file systems you need to know what devices are available. This can be done using by getting a listing from the fdisk command. As a root user you can use the -l option with fdisk to get a listing like the following:

```
bash# fdisk -l

Disk /dev/sda: 42.9 GB, 42949672960 bytes, 83886080 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 label type: dos
Disk identifier: 0x0009d919

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     1026047      512000   83  Linux
/dev/sda2         1026048    83886079    41430016   8e  Linux LVM

Disk /dev/mapper/centos_test-root: 38.2 GB, 38243663872 bytes, 74694656 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 /dev/mapper/centos_test-swap: 4177 MB, 4177526784 bytes, 8159232 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
```

Okay, this shows us that the machine I am looking at has a device **/dev/sda** that is a 42.9 GB disk. The disk is divided into two partitions **/dev/sda1** and **/dev/sda2**. We can see that the first partition is a standard Linux partition and starts at sector 2048 and ends at sector 1026047. This gives us about 500 MB of space.

The second partition is Linux LVM partition. LVM stands for Linux Volume Management. It allows for easier movement of blocks between volumes. We can see that second partition takes up from sector 1026048 to 83886079. Of that we can also see that two volumes have been created. One volume is taking 74694656 sectors and is mapped to the device /dev/mapper/centos_test-root and the other is taking 8159232 sectors and is mapped to /dev/mapper/centos_test-swap. We can also see the sizes.

Using the fdisk command we could edit the partition table of /dev/sda using the following command as root:

```
fdisk /dev/sda
```

This takes you to a menu driven dialog that you can use to modify the partition table in memory, the write it to the disk. Once it has been written to the disk, the kernel will need to reread it in order to use it properly.

## Command: partprobe

To cause the kernel to reread the partition table use the **partprobe** command. This command updates the kernel, but there will still be a few random processes that might still remember the old partition table. This command is good for immediately after partitioning, but ultimately, you will want to reboot the system:

```
partprobe
```

## Command: df

Using the df command we can see where these devices are mapped, well at least some of them. Here is a sample output from the df command:

```
bash# df
Filesystem                    1K-blocks    Used Available Use% Mounted on
/dev/mapper/centos_test-root   37329092 2208504  35120588   6% /
devtmpfs                        1923808       0   1923808   0% /dev
tmpfs                           1933316       0   1933316   0% /dev/shm
tmpfs                           1933316  197032   1736284  11% /run
tmpfs                           1933316       0   1933316   0% /sys/fs/cgroup
/dev/sda1                        508588  147636    360952  30% /boot
```

If you want the information in a more readable format, you can use the -h option:

```
bash# df -h
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/centos_test-root     36G  2.2G   34G   6% /
devtmpfs                        1.9G     0  1.9G   0% /dev
tmpfs                           1.9G     0  1.9G   0% /dev/shm
tmpfs                           1.9G  193M  1.7G  11% /run
tmpfs                           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda1                       497M  145M  353M  30% /boot
```

At this point, you might be wondering, where is the **/dev/mapper/centos_test-swap** device listed? It is probably easiest if you just look at the File System Table to see where everything is supposed to be mapped.

## Command: du

Using the **du** you can see the estimated disk usage for any given directory. To use the command, just pass the directory as the argument. The du command will then traverse the directory tree all of the way down to the sub-directories and add them up as it returns. When the command completes it will return the total for the directories.

The following command can be used to see how much space is used by the **/etc/** directories.

```
du /etc/
```

Sometimes the output of the **du** program is not easy to read, so people use the **-h** switch to show the numbers in easy to understand numbers.

```
du -h /etc/
```

### *File System Table*

The File System Table tells the operating system how to mount partitions and where they are supposed to be mapped and the arguments used in mapping them. Below we have a sample file system table:

```
bash# cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Thu Mar 19 23:55:29 2015
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/centos_test-root              /       xfs     defaults        1 1
UUID=0800954e-8de5-45ab-a09d-a6b2baa8579b /boot   xfs     defaults        1 2
/dev/mapper/centos_test-swap              swap    swap    defaults        0 0
```

The **cat** command was used to display the contents of the file. We can see a number of comments that

are not really the main part of the file. Only the last three lines are important.

Each of the lines contains a device, a mount point, a file system, options, and two numbers which are for file system check and backup. The lines are processed in order and partitions are mounted on the tree as each line is processed.

We can see that **/dev/mapper/centos_test-root** is mapped to the / directory, but we already knew that from the **df** command. We can also see that the **/dev/mapper/centos_test-swap** device is being used as swap or virtual memory.

The last partition we want to look at is the middle of the last three lines. The device listed is **UUID=0800954e-8de5-45ab-a09d-a6b2baa8579b**, but we know that **/boot** is really **/dev/sda1**, so what is the UUID thing? Well, you could use /dev/sda1 there and it would work well, right until you added another hard drive to the system. If you added an additional hard drive, would the new drive be / dev/sda or would it be /dev/sdb? If it became /dev/sda, then our current drive would probably become / dev/sdb and the /dev/sda1 would become /dev/sdb1 instead. That could really complicate things. To make things easier, each partition is given an ID that can be used to positively identify it even if the order of the drives changes.

If you need to lookup an ID to see what device it is connected to, you can take a look at the information stored in the **/dev/disk/** directory. In that directory there is a list of other directories with symbolic links to the devices.

```
[root@test ~]# ls -l /dev/disk/
total 0
drwxr-xr-x. 2 root root  60 Aug 21 11:56 by-id
drwxr-xr-x. 2 root root 100 Aug 21 11:56 by-label
drwxr-xr-x. 2 root root 120 Aug 21 11:56 by-path
drwxr-xr-x. 2 root root 100 Aug 21 11:56 by-uuid
```

Changes to the /etc/fstab file do not affect the whole system immediately. As you change the /etc/fstab file, it is best to verify the changes work. You can verify changes by either unmounting and remounting the partition or by rebooting your system.

## Command: blkid

When you create partitions or make changes, it is important to know either the device, the label, or the UUID. In reality, you should probably know all of them or at least know how to find them. The **blkid** command helps you find some of this information.

The following example shows sample output from the **blkid** command on a machine with standard partitions.

```
[root@server ~]# blkid
/dev/block/8:2: LABEL="/" UUID="6dc080b3-134e-418b-acc2-973fc62b5a7f" TYPE="xfs"
/dev/block/8:1: LABEL="/boot" UUID="7c3d481d-78af-4a37-b696-8a4fecd63ae2" TYPE="ext3"
/dev/block/8:3: LABEL="swap" UUID="5bc5641c-744e-4c70-a9df-36876b8ace0e" TYPE="swap"
```

The following example shows sample output from the **blkid** command on a machine with LVM partitions.

```
[root@server ~]# blkid
/dev/mapper/server-root: UUID="ddf92e68-79b6-43fb-9ee2-dca1089f60a4" TYPE="xfs"
/dev/sda2: UUID="ep4c3f-UK2D-5QHY-Y0fK-0OOt-RG1A-TOxb5E" TYPE="LVM2_member"
/dev/sda1: UUID="0800954e-8de5-45ab-a09d-a6b2baa8579b" TYPE="xfs"
/dev/mapper/server-swap: UUID="6985599d-6725-49bc-a73a-a05bc95aa686" TYPE="swap"
```

## Command: mount

The mount command normally takes two basic arguments. The first is the device and the second is the mount point. For example, to mount **/dev/sda1** at **/boot** you would use the following command:

```
mount /dev/sda1 /boot
```

You can additionally add information such as the file system type and other options as well. If your file system is listed in the **/etc/fstab** file you can supply just one of the argument instead of both. If there is a line for the above listed device in the **/etc/fstab** file then either of these lines would also work:

```
mount /dev/sda1
```

```
mount /boot
```

In both of these cases the mount command looks at the **/etc/fstab** file for the device/mount point pair, the file system and options to use in issuing this command.

## Command: umount

Once a file system has been mounted, you can unmount it using the umount command. You can issue the umount command with either the device or the mount point to unmount the device. For a device /dev/sda1 mounted at the mount point /boot either of the following commands would unmount the device:

```
umount /dev/sda1
```

```
umount /boot
```

## *SWAP*

SWAP partitions and SWAP files are places on a disk that are dedicated for virtual memory or swap memory. Whenever a system needs to flush active processes from memory in order to load additional processes, the memory that is not currently in use can be written out to SWAP memory to make room in active memory for active processes.

In older documentation it was normally written that you should have twice as much SWAP as you had RAM. This makes the assumption that you will be using your SWAP memory. In reality, if you have a slow hard drive, you do not want to use SWAP. If you have a solid state drive then it might make more sense to use the SWAP.

## Command: mkswap

Since SWAP is not a normal file system type of partition, it does not use the same commands for creation and mounting as file systems. To create SWAP partitions we use the mkswap command as root. If we wanted to turn /dev/sda5 into a swap partition we would first need to make sure the type flag for the partition was set to Linux swap (82), then we could use the mkswap command.

```
mkswap /dev/sda5
```

Once the partition has been formatted as swap, it is ready to be listed as swap.

## Command: swapon

You can see the list of swap partitions actively being used as swap by the kernel through the /proc/ kernel interface. Use the following command to see the list of swap partitions being used:

```
cat /proc/swaps
```

If you do not have anything there, your list will be non-existent. Once you activate a swap partition with the swapon command the file should automatically be populated. The following command should activate a formatted swap partition located on /dev/sda5.

```
swapon /dev/sda5
```

After activating the swap partition here is a sample dialog for looking at the /proc/swaps virtual file.

```
bash# cat /proc/swaps
Filename          Type            Size    Used    Priority
/dev/sda5         partition       4128760 0       -2
```

## Command: swapoff

If you want to make changes to partition sizes, etc. you might need to turn off a swap partition. The swapoff command does this. To turn off the /dev/sda5 swap partition you would use the following command:

```
swapoff /dev/sda5
```

As you might expect, you can take a look at the /proc/swaps virtual file to see that your change is reflected in the kernel list of swaps.

# Scripts and Scripting

System administrators regularly run into situations where they need to automate a process. The easiest way to do this is often by writing a script that does everything for you on the command line. In addition to learning how to work with the system, it is a good idea to learn how to write these scripts. Unfortunately, I will not be covering script writing in this book. There are, however, multiple scripting tutorials on the Internet you can work through.

## *Basic Scripts*

UNIX like systems have a simple way of writing scripts. If you write a text file and make it executable, then naturally, most shells will just run through the text file and execute each line as if it were entered on the command line.

If I wanted a script called **script.sh** that moved into the home directory and performed a long directory listing and saved the output to a file called **dir.txt** then the following script would do this:

```
cd /home/
ls -al > dir.txt
```

The next steps would be to make the script executable, then actually run the script. The following commands would do that:

```
chmod 755 script.sh
./script.sh
```

You might remember, but just in case, the period slash before the script name above had a purpose. You can run scripts and programs in your path and also scripts and programs that you tell the shell the directory they are located. The period indicates the directory you are currently in. So when we type ./script.sh the shell is looking in the current directory for a file called script.sh to execute.

## *Bash Shell Scripts*

The file **script.sh** we just wrote is a working script, but you might want to tell the shell which interpreter you would prefer that we use. This can be done easily using the first line of the text file. If the first line starts with the hash bang characters "**#!**" then the next part is interpreted as the path of the executable to run. The file contents are then passed to the executable as standard input. Let's look at the following **hello.sh** file:

```
#!/bin/bash
# My hello world program
echo "Hello World!"
```

Assuming you set the execute bit and executed the script by typing **./hello.sh**, here is the order of things that would happen:

1. The shell would start reading the text file
2. The shell would parse out the #! and execute the /bin/bash program
3. The shell would send the contents of the hello.sh file to the /bin/bash program as input

This order is a little bit different from the standard binary programs. When binary programs are executed, the is no second interpreter program that is executed. In order to execute a binary program you only need to have the execute bit set. In order to execute a script you need to have both the execute bit set and the read bit set because if you cannot read the script you cannot hand the script to the interpreter for execution.

Shell scripts and scripting languages all tend to have the hash mark as a comment character because they need a way to filter out the first line of the file and still continue to run. Both the first and second lines of **hello.sh** start with hash marks, so they are treated by the /bin/bash interpreter as comments. Only the last line of the program is actually treated as code.

The last line runs the echo command which prints the next argument to standard out. This script, when executed, would display the text "Hello World!" on the command line or terminal interface.

## *Scripting Languages*

Most Linux distributions currently ship with Python 2 as the default scripting language. Python 3 is a nice version, but is not really compatible with all of the scripts that are regularly used on Linux machines, so Linux distributions have opted to stick with Python 2 for now.

Many distributions additionally ship with Perl and a few other scripting languages, but most have tried to convert all of their scripts to Python.

In general you will find Python scripts that start with either of the following lines:

```
#!/usr/bin/python
```

```
#!/usr/bin/env python
```

These two lines are slightly different. The first one tries to start the Python interpreter that is located at /usr/bin/python and does not work if that is not the name of the Python interpreter. The second one needs the env program to be located at /usr/bin/env to run, but then it uses the first program called python that is available in your path. If your Python interpreter is located at /usr/lib/bin/python then the second line will have a higher success rate.

Most successful system administrators write scripts in shell script or a higher level scripting language like Perl, Python, or Ruby.

Contrary to common belief, it usually does not matter what file extension a script file ends with. Take the following script for example:

```
#!/usr/bin/python
print "Hi!"
```

Because the shell reads in the first line to find the interpreter, it does not matter what the file extension is. Whither I name this script **script3.sh** or **script4.py** or even something illogical such as **system.exe** or **butterfly.jpg** it will run the same as long as the execute and read permission bits are set.

Where the extension does matter is outside of the shell. If you have files being executed by something like a web server, then in order to identify the file types and present them in a timely manner, you tend to have extensions that are tied to MIME types.

## *Python Hello World Script*

Sometimes when you are working with a new language it is helpful to have a basic hello world script to use as an example to get started. The following is text of a hello world Python script. Save the contents below as **hello.py**, set the execute bit, the execute the script.

```
#!/usr/bin/python
# Python Hello World program
print("Hello World!")
```

For Python 2 the parentheses are not required, but since they are required in Python 3 I recommend putting them in. The print statement in Python automatically adds a newline to the end, so you do not need to manually add it.

## *Perl Hello World Script*

The following is a sample Perl hello world program. You just need to type these lines in an editor, save as **hello.pl**, set the execute bit on the file, and run the file.

```
#!/usr/bin/perl
# Perl Hello World program
print "Hello World!\n"
```

Unlike Python, Perl does not add the newline, so the program above explicitly adds it to make sure the program reacts the way you would expect.

## *Scripting Tutorial Links*

To learn how to write scripts, I recommend the following links:

- Shell Scripts – http://linuxcommand.org/
- Python 2 – https://docs.python.org/2/
- Perl – https://www.perl.org/learn.html

- Ruby – https://www.ruby-lang.org/en/documentation/

# Software Installation

Software comes to your system in many forms. Originally all software was binary, then assembly languages were created to make the binary more human readable. A major break through in computer technology came when the compilers and higher level programming languages were invented. Compilers could take the same source code and convert it to multiple different architectures as long as a compiler was written to do the work.

## *Compiling Source*

Linux distributions usually have the GNU GCC Compiler as the default compiler. The minimal installation of Cent OS does not come with a compiler. It is assumed that you will be able to accomplish most of your tasks without a compiler by downloading pre-built packages and using them instead.

If you discover that you need a compiler, you should install the gcc compiler. In addition to gcc, you might also need the GNU GCC C++ compiler. The gcc package used to contain both the C and C++ compilers, but they are packaged separately now. The following should get you both the C and C++ compilers.

```
yum install gcc gcc-c++
```

## C Hello World Program

In order to start writing C programs on a Linux machine it is good to know how this is accomplished. With a text editor open **hello.c** and type the following text.

```
#include <stdio.h>

/* C Hello World Program */
int main(int argc, char **argv) {
  printf("Hello World!\n");
  return 0;
}
```

Once you have your code typed in and saved, you are ready to compile. The gcc compiler will default to creating a program called **a.out** if you do not specify the file name. The following are two commands you could type to compile the source code, then execute the resulting binary program.

```
gcc hello.c
./a.out
```

If you decide that you want to specify a name instead of using the default **a.out** you can use the **-o** option on the command line. The following command will take the **hello.c** source file and compile it to a resulting binary called **hello**.

```
gcc hello.c -o hello
```

There is a temptation to type **hello.c** for both the source and the resulting binary file. Resist this urge. If you do, you will not have a source code file in the end.

After creating the hello binary file you can execute it using the following command.

```
./hello
```

You might have noticed that we did not have to set the execute bit on the file. The compiler automatically takes care of setting permissions.

## C++ Hello World Program

You can compile most C programs using the g++ compiler as if they were C++ programs, but you can also choose to use the newer C++ syntax instead. The following source is for a simple **hello.cpp** file.

```
#include <iostream>

// C++ Hello World Program
int main(int argc, char **argv) {
  std::cout << "Hello World!\n";
  return 0;
}
```

After creating the hello.cpp source code file, it is time to compile the source code. The gcc compiler requires the gcc-c++ package to compile C++ source. Make sure it is installed. Use either of the following command sets to compile and run the program.

```
g++ hello.cpp
./a.out
```

```
g++ hello.cpp -o hello
./hello
```

## Makefiles

Because it can sometimes become difficult when many source code files are involved in building a package or multiple packages scripts were used to keep track of changes using date stamps and build newer packages when needed. The following is an example **Makefile** that will build the two hello world programs we looked at above.

```
BIN=helloc hellocpp
```

```
all: $(BIN)

helloc: hello.c
        gcc hello.c -o helloc

hellocpp: hello.cpp
        g++ hello.cpp -o hellocpp

clean:
        rm -f $(BIN)
```

Make sure the file is saved as **Makefile** so that the **make** command has a source file to read. Once the file has been created, you can use the **make** command to build the packages.

```
make
```

Most Makefiles are more complex than this. When the **make** command is executed, the make program reads the **Makefile** and sees that the first option is **all**. After the all is a list of binary programs that need to be checked. For each one listed below we can see which source code file they are dependent on. If the source code is newer than the binary program than the compile command listed below is executed. For helloc we see the compile command starts with gcc. For hellocpp the command starts with g++.

When you execute the make command you can specify which option you want to run. The all option is the default because it is first, but you could specify it if you wanted. You can also specify only one of the binary programs, or even the last option, **clean**, which will remove the binary programs.

Here are the possible incantations you could have used with this Makefile.

```
make
```

```
make all
```

```
make helloc
```

```
make hellocpp
```

```
make clean
```

When you download source code packages from the Internet, it is common for the programs to come with a **Makefile** or a **configure** program that builds the Makefiles before compiling.

## *Red Hat Package Manager (RPM)*

The Red Hat Package Manager keeps track of pre-built packages installed on your machine and helps

to make sure you do not install pre-built packages without the required dependencies. You can administrate the rpm database using the **rpm** command.

## Command: rpm

Each of the packages is a file that comes with the .rpm extension. You can view the list of all the files using the rpm command with the -qa options. The file names can be divided into NAME-VERSION-RELEASE.ARCHITECTURE.rpm to determine version and other information. The following dialog shows the name of an installed package:

```
bash# rpm -qa | grep nano
nano-2.3.1-10.el7.x86_64
```

The **rpm -qa** command will normally show hundreds or thousands of packages, but by piping the output through the **grep** command we can filter out results we are not interested in. This filters the results to only packages with **nano** in the name somewhere.

We can see that we have Nano version 2.3.1 installed. We can also see that the release number is 10. Additionally, we can see that the package was built for the el7.x86_64 architecture. The el7 indicates Enterprise Linux 7 from Red Hat Enterprise Linux (RHEL) 7. The CPU architecture is for the x86_64.

If we wanted to install packages using the rpm command we could use either of the following commands:

```
rpm -i nano-2.3.1-10.el7.x86_64.rpm
```

```
rpm -ivh nano-2.3.1-10.el7.x86_64.rpm
```

The first command just installs the package. The second one installs the package but does so with a progress bar to indicate where it is at in the process.

The only real problems here are the following:

- You need to have the nano package on your machine to install it. Before you can do that, you need to find it.
- You need to have all of the dependencies for nano installed before you invoke the installation command.

Both of these problems can be handled with the Yellow Dog Update Manager (YUM) which we will talk about in just a little bit.

If instead of installing a package that you do not have on your system you want to update, you can use either of the following commands:

```
rpm -U nano-2.3.1-10.el7.x86_64.rpm
```

```
rpm -Uvh nano-2.3.1-10.el7.x86_64.rpm
```

If the package is missing the rpm command will perform an installation instead of an upgrade. Once again, for this command to work you need the file.

Now that you can install packages, what about the uninstallation process? You can uninstall with the -e for erase option. The following command would be used to uninstall the nano package:

```
rpm -e nano-2.3.1-10.el7.x86_64
```

### *Yellow Dog Update Manager (YUM)*

Trying to find all of the dependencies for a package can take a long time. There are times when whole trees of dependencies must be installed before a single desired package. Fortunately, the rpm command has a nice front-end utility called **yum** that simplifies all of this.

## Command: yum

The **yum** command has a configuration file at **/etc/yum.conf** that lists where package information is kept. Each collection of packages is called a repository. Linux distributions usually have multiple repositories that they can pull software packages from. Each repository also has a configuration file which is stored in the **/etc/yum.repos.d/** directory.

When you use the yum utility to install a package, it looks through its cache of information from each repository and then selects which packages are required by the package you request. Yum can download the packages and install them.

To search for packages you might want to install use the **search** option with the yum command. If I wanted to install the GNU GCC Compiler, I could try either of the following searches.

```
yum search GNU GCC Compiler
```

```
yum search gcc
```

Because the first search is so detailed, it will not likely return the name of the package containing the gcc compiler. The second search will get the gcc compiler and anything with gcc in the name or description. You will probably need to perform a few different searches to get the best results.

Once you know which package you want to install, you can use the **install** option. The following would install the gcc package and all dependencies.

```
yum install gcc
```

Many groups of people make additional repositories that you can download and install additional software. One common repository that people like to install is the **epel-release** repository. This repository provides less known software packages and packages such as games that are less common on servers. To install the EPEL repository use the following command.

```
yum install epel-release
```

This command will create an additional .repo file in the **/etc/yum.repos.d/** directory. When you perform your next **yum** command the repo will download a list of files.

When you need to update your system you can use the yum command as well. To update you can use the update option. The following command updates your system.

```
yum update
```

Sometimes you want to remove a package that was installed with **yum**. To remove a package you can use either the **remove** or **erase** option. The following could be use to remove the gcc compiler.

```
yum remove gcc
```

When you remove a package it is possible and even likely that the dependencies that might have been installed when you installed the package will remain in place.

During the Linux installation process you have the option of selecting multiple groups of packages to install. Now that your system is already installed those groups are not as obvious, but can still be installed. To install a group of software packages you can use the group options in yum. To see a list of the available groups use the **grouplist** option. The following command would show you the available groups to install.

```
yum grouplist
```

If you decide you would like to install one of these groups you can use the **groupinstall** option. If you decided you wanted the GNU GCC Compiler and all of the packages related to it you might opt into installing the Development Tools group. Here is how you would install that group.

```
yum groupinstall "Development Tools"
```

Groups installed this way can also be removed. To remove a group of packages you can use the **groupremove** option. To remove the Development Tools group use the following command.

```
yum groupremove "Development Tools"
```

# Building Kernel/Modules

Building a kernel sounds difficult, but it is really not that complex if you know what you are doing. Before you do anything, it is probably a good idea to update your system, then reboot into your newest kernel. Sometimes machines that are not fully updated have strange issues.

## *Getting the Kernel Headers*

The kernel source code is usually stored in the **/usr/src/** directory and sub-directories. When you install the **kernel-devel** package, a branch of your kernel's source code headers is put into the **/usr/src/kernels/** directory. Navigate to that directory and look at the kernel source code headers available to work with. (***Note**: kernel-devel does not provide the full kernel source code, just the headers*).

When you move into one of the kernel sub-directories you will see there is a **Makefile**. To use this Makefile for kernel development, for the next step, you will need to have the **gcc** and **ncurses-devel** packages installed. To install the gcc package, it is probably best to get the whole "**Development Tools"** group. Then install the **ncurses-devel** package.

```
yum groupinstall "Development Tools"
yum install ncurses-devel
```

With them installed you are ready for the **menuconfig** make option. Run the following command to get into a kernel configuring menu.

```
make menuconfig
```

This makes it possible to look around and set building options. Options listed with a "*" are marked to be built into the kernel. Options with an "m" are marked to be built as kernel modules. If the option is marked with a space then it is marked for not being included. Because we do not have a full kernel source code we cannot build the kernel, but we can see the options.

## *Building Vanilla Kernel/Modules*

Many Linux distributions, including CentOS, use modified copies of the kernel source code. Copies of the source code that have not been modified are referred to as vanilla. Follow the instructions for getting the kernel headers. After you have the required packages for viewing the **menuconfig** make option you need to install the following packages.

```
yum install hmaccalc zlib-devel binutils-devel elfutils-libelf-devel
```

Next you need to get a copy of the vanilla kernel source code. You can find many different versions of the kernel source code at https://www.kernel.org/ and a few other sites. Because many of the packages on your system are built against a specific version of the kernel, it is probably best to get the version of kernel that matches your current system. You can see your kernel version using the **uname** command

with the **-r** option.

```
uname -r
```

Navigate to your **/usr/src/kernels/** directory. While in that directory extract your kernel files. The kernel source is large and the extract will take a while. Once the extraction is complete you can move into the newly created kernel source tree.

Your current kernel has a lot of configuration options already set. If you want to use those options, you can copy the **.config** file from a different kernel headers directory tree or you can get a copy from the **/boot/** directory. The following command should get you a copy from your **/boot/** directory.

```
cp /boot/config-`uname -r` .config
```

Note that the above command uses the backtick characters. If the command is successful, you will already have your base system configuration options in place. Now you are ready to go into the **menuconfig** and then build your **kernel** and **modules**.

Run the following command to change the default options and set a few of your own.

```
make menuconfig
```

After you have make all of the changes you want to make, you are ready to build your kernel. You can build your kernel with the all make option. Build your kernel with the following command. (***Note**: Building a kernel can take anywhere from minutes to multiple hours*).

```
make all
```

After building your kernel, it is a good idea to build kernel modules. You can build them with the modules make option. Build the kernel modules with the following command.

```
make modules
```

# Scheduled Tasks

Once system administrators have scripts created to do their work, then the next step is getting the operating system to run the scripts when you are not there. You might want scripts to run hourly, daily, on weekends, on the first Friday or the month of something else. To make this possible there are services such as Cron and even systemd that make things easier.

## *Service: crond*

Cron is a nice service that has been around for a long time and provides a way to create tasks that will run at regular intervals. By default, the crond.service is enabled and running on new Linux systems. Most people just use cron instead of making changes.

You can start, stop, restart, enable, and disable the service with crond.service using systemctl.

```
systemctl start crond.service
```

Each user can have a cron table or crontab. The crontab files are stored in the **/var/spool/cron/** directory. Each file is named the same as the user who created it and contains one line for each scheduled task.

The crontab task line format looks a little like the following:

```
0 3 * * * /home/alice/bin/backup.sh
```

There are 6 fields. The first 5 fields are used to identify the days and times when the task should be run. The sixth field is the actual command line that needs to be executed. Note: cron does not operate with the same environment as the user, so everything needs to be specific including the full path to the executable and in cases of scripts, all parts of the script should assume no prior environment including path.

The first five fields are as follows: **minute**, **hour**, **day of month**, **month**, and **day of week**. Let's look at the example from above. The values we see are zero and three. The minutes is set to zero and the hours is set to three, so this indicates that the script is supposed to be executed everyday at 3:00 AM.

Maybe you want something to run every first and third Sunday at 2:15 AM. This gets a little complicated, but can be done. We already know how the file would list 2:15 AM, so now we just need to figure out the days of the week and figure out how to do the first and third Sunday.

Sunday is either the 0 or 7 day of the week. If we wanted just Sunday, then we can put the day of the week in as either 0 or 7. The first Sunday is always something in the 1-7 day of the month range and the third Sunday is in the 15-21 day of the month range. We can use ranges with dashes and list multiple numbers or ranges with a comma. Here is what our resulting crontab line could look like:

```
15 2 1-7,15-21 * 0 /home/alice/bin/backup.sh
```

Now that we have a general idea how to create crontab lines, we need to know how to actually edit the crontab file.

## Command: crontab

To edit the crontab file, we can use the **crontab** command with the -e option for edit. Here is the command:

```
crontab -e
```

The only major issue you will run into is that your crontab file editor is set to default to Vi. Now, if you are not familiar with Vi, I recommend you brush up a little beforehand.

In addition to the user crontab files, you also have system global cron files. The easiest way to find them is to navigate to the /etc/ directory and look for anything that starts with cron. There is a system crontab file located at **/etc/crontab** that can be edited to run task as the system. Additionally, if you have regular tasks, they can be put in directories such as **/etc/cron.hourly/**, **/etc/cron.daily/**, and **/etc/cron.monthly/** for hourly, daily, and monthly execution.

# Managing Processes

Working with processes on Linux machines can be an important skill. You can see running processes in a number of ways. To display a list of processes running in your current terminal, you can use the **ps** command. With options, you can see more than just your own processes. If you want a more interactive and active display, you can use the **top** command.

## Command: ps

To display your own processes in your current terminal, use the **ps** command without any options.

```
ps
```

If you want more than just your immediate processes, you can add additional command line switch characters. One common set of options is to use "**aux**" to display all processes on the system. The following would list all processes on the system.

```
ps aux
```

## Command: top

The **top** command actively shows all processes running on the system. By default, the most active processes are listed at the top and less active processes are lower on the list. If, at any point, you want to leave the top command, you can press the "**q**" key.

You can sort the list of processes by different columns. Since you start sorting by CPU you can change to the column to the right using the greater than "**>**" character and to the column to the left using the less than "**<**" character. Unfortunately, it is not always easy to see which column you are sorting by.

If you see a process that you want to kill, you can press the "**k**" key. The top command will then allow you to type the process ID (PID) of the process. After you type the process ID it will prompt you for the signal number to send. The default is to send signal 15, which tells the process to terminate, but does not force it to stop.

## Command: kill

The **kill** command is used to send signals to processes. The name implies that the command is used to kill or terminate a process, however, it can do a lot more. To get a list of possible codes the kill command can send, use the **-l** (lowercase L) option.

```
kill -l
```

To kill a process, you need to know what the process ID (PID) of the process is. You can then send the **SIGTERM(15)** terminate signal using just the kill command and the PID. To kill process 1234 you

would use the following command.

```
kill 1234
```

Sometimes the process does not want to die and other times the process has crashed and does not remember how to handle the signal interrupt. If you want to increase your chances of getting the process to terminate, you can send the **SIGKILL(9)** signal instead. Since this is not the default, you need to instruct the kill command to use SIGKILL. Either of the following will do the job.

```
kill -9 1234
```

```
kill SIGKILL 1234
```

If you send the SIGKILL signal instead of the default SIGTERM the process is not told to terminate, it is killed from outside by the system. This prevents the process from catching the interrupt and doing something else instead.

Some processes, especially services and daemons use the **SIGHUP(1)** signal to let them know that changes have happened in the configuration files. When those processes are sent the SIGHUP signal, they read their configuration files again and continue to run on the new configurations.

## Kernel /proc/ Directory

The **/proc/** directory is an active interface into the insides of the kernel. The kernel keeps track of what is happening on the system.

**Processes**

In the /proc/ directory there is a sub-directory for each process. For an example to get you started, take a look at the following dialog.

```
bash$ sleep 10000 &
[1] 701
bash$ cd /proc/701
bash$ cat cmdline
sleep10000bash$
```

That might look a little confusing, but it should make sense. We started a background process that would sleep for 10000 seconds. When the process went to the background we were told it was PID **701**. We went into the **/proc/701/** directory to start looking at information. In there we looked at the file called **cmdline** which tells the command line that was typed to start the command.

In addition to the cmdline file, there are multiple other files that help us know how the process was started and what information it had when it was started.

**Memory and CPU Information**

You can see memory and CPU information in the /proc/ directory.  To view your current memory you can use the following command.

```
cat /proc/meminfo
```

To see your CPU information you can use the following command.

```
cat /proc/cpuinfo
```

# System Users

## *Users and Groups*

Linux systems use users and group accounts to keep track of ownership and permissions. Each file or directory has a user and group owner. Permissions are then assigned to give rights to the owner, the members of the group, and everyone else. This is a simple approach to security that is easy to understand and simple to implement in the Linux kernel.

The list of users and groups are stored in the **/etc/passwd** and **/etc/group** files respectively. The user files are then traditionally stored in the /home/ directory. In addition to the list of users, we also have a list of encrypted copies of passwords stored in the **/etc/shadow** file.

## File: /etc/passwd

The /etc/passwd file stores important information about users. The following are four sample lines from the /etc/passwd file.

```
root:x:0:0:root:/root:/bin/bash
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
alice:x:1000:1000:Alice:/home/alice:/bin/bash
bob:x:1001:1001:Bob:/home/bob:/bin/bash
```

You will probably notice that there is one line for each user. The user line can be broken into seven parts each separated by a colon. They are the username, the password, the user ID (uid), the group ID (gid), the name or comment for the user, the home directory, and the shell.

The user name listed in this file is what is displayed when you perform a directory listing and see the owner of files. If a file is owned by uid **48** then the owner will be listed as **apache**. Additionally, when the user logs into the system it will use the name listed here in exactly the same case as the file.

The password here is usually the letter 'x' because we are not listing the passwords in this file anymore. In the past the user names and passwords were stored in the same /etc/passwd file, but as methods for password breaking were discovered, people realized regular users should not see the encrypted password. Since there is important information in the /etc/passwd file that users need to know, such as which user name is tied to each uid, the encrypted passwords were moved to the **/etc/shadow** file and an 'x' was placed in the password field.

The uid field contains an integer number. This number is the user's ID number.

The gid field contains an integer number as well. This number is the group's ID number and can also be found in the /etc/group file. Using this number you can look up the default group that each user is a member of. On most Linux systems each user has their own group and tend to have group names and gid numbers that match their user account and uid.

The name field is not required, but does make it easier when authenticating at a graphical login screen. Normally it is set to be the full name of the user using the account. When the user is not a normal person, there is sometimes a comment in the field that indicates what the account is used for.

The home directory indicates where the user account should start when logging in. For normal users the default is to have a directory in the /home/ directory that matches the name of the user. For security reasons, some services run as different users than root. They then are usually assigned a user to run as and have one of their directories listed as their home directory.

The last field, the shell, lists which terminal shell the user uses. If the user does not need to type commands at the shell then the shell is usually listed as **/sbin/nologin** to prevent any access should the service be compromised.

The /etc/password file is not normally edited directly by root or other users. Instead it is edited using commands such as **useradd**, **usermod**, and **userdel**.

## File: /etc/group

The /etc/group file stores important information about groups. The following are four sample lines from the /etc/group file.

```
root:x:0:
wheel:x:10:alice,bob
alice:x:1000:
bob:x:1001:
```

Here we have four groups listed. The root group belongs to the root user. The wheel group is usually reserved for users who are considered super-users. Alice and Bob both have their own groups which match their user names and, in this case, numbers.

Each line had four columns. The first column is the name of the group. The second is the group password. The third is the group ID number (gid). The fourth is the additional members of the group.

The first three columns are probably pretty easy to understand, so I am going to just focus on the last column. In the /etc/passwd file we have the gid for the default group the user is a member of. Because users can be members of multiple groups, they need to be listed somewhere. The last column in each group line provides a place where additional users can be listed as members of that group.

Alice is a member of the alice group, but she is also a member of the wheel group.

You can make modifications to group membership and even add, delete, and modify groups by directly editing this file, but usually this is handled by commands such as **groupadd**, **groupmod**, and **groupdel**.

## File: /etc/shadow

The **/etc/shadow** file contains the passwords for users on the system. This file cannot be read by regular users. In order to read this file you need to be root or run a program that executes as the root user. Only root can read this file.

Here are two sample lines from the **/etc/shadow** file.

```
sshd:!!:16719::::::
bob:$6$cLzh55.B$uJDtCX1v8Th3Yg7UY3dLzZW1hatTsqy7scnuAGI4uI97jkycAiNdctEOipnUW2pEd
J.GmD2joav64HSTH7qmK/:16726:0:99999:7:::
```

Each line in the file is dedicated to a single user. In the case of bob, there are two lines, but that is because of the line wrap. The lines are formed out of nine columns separated by colons. The columns are the user name, the encrypted password, the date of the last password change, the minimum password age, the maximum password age, the password warning period, password inactivity period, account expiration date, and a reserved field for future use.

The **sshd** user does not appear to have a valid encrypted password. This is because the account for sshd is not authorized for login or authentication.

The bob user does have an encrypted password. This encrypted password is very long, even though bob's actual password of "*aloha123*" is very short.

The dates listed are usually in the number of days since the epoch. The epoch is the date **January 1, 1970 UTC**. In the case of minimum and maximum password ages, the numbers are also in days, but they are since the last password change. 0 indicates no minimum and 99999 indicates no maximum age. Normally you would not need to look at the number fields, but if you are interested in more information you can look at the man pages for the shadow file.

```
man 5 shadow
```

The passwords are normally set using the **passwd** command. It will be discussed more when you get to that section.

## Directory: /etc/skel/

When new accounts are created, the users need to start with some files. As the users run various programs and use different features, their home directories start to collect files and directories.

The initial directory given to each user upon account creation is the **/etc/skel/** directory. All files and directories contained in the /etc/skel/ directory are copied over to the new home directory and are assigned to the new user.

When the contents of the /etc/skel/ directory are modified, the contents of the existing users home directories does not change. This is not an overlay directory, so changes to the skel directory only affect future account creations.

If you were running a web server and wanted users to be able to host web pages of their own, you might add the **public_html** directory to the /etc/skel/ directory before creating the user accounts. This would make creating user home pages a little bit easier.

## Command: useradd

The useradd command is used to add users to your system. When creating an account there are two things that you want to know. First, what is the user name that you want to use. Second, what is the name of the user who you are creating an account for. In reality, you can create the user account without a name/comment, but it makes things easier in the future if the accounts are identified.

The following commands could be used to create an account for Alice.

```
useradd alice
```

```
useradd -c "Alice" alice
```

The biggest difference between the two is that in the second command the comment for the alice account is set to “Alice”. If Alice had a last name, we would probably include it in with her first name.

## Command: groupadd

The **groupadd** command makes it easy to create new groups. Normally, when new user accounts are created they come with a new group named the same thing as the user name. When you use the groupadd it is usually because you want to create a special group.

To create a group you could just invoke the command with the group name as the single argument. The following command would create a group called **spies**.

```
groupadd spies
```

While this would do the job, we might want to assign a special gid to this group. If we wanted to create the group spies and assign it the gid of 123 we could use the following command.

```
groupadd -g 123 spies
```

## Command: usermod

The usermod command can be used to change the options that you might have missed when you created the user accounts. Additionally, it can be used to lock and unlock accounts.

To change the comment or name associated with an account you can use the -c option. If Bob wanted to change his comment from “Bob” to “Robert” the following command could be used.

```
usermod -c "Robert" bob
```

To change the group memberships you can use the -g or -G commands. The -g option is for the default group and the -G option is for all supplementary groups. If we wanted to add alice and bob to the wheel group we could use the following commands.

```
usermod -G wheel alice
usermod -G wheel bob
```

You do need to keep in mind that if they are already members of other groups you might be removing

them from those groups. To be safe it is usually a good idea to use the **id** command beforehand to see what other groups they are already members of.

The check which groups Bob is a member of we can use the following command.

```
id bob
```

If you do not want to check current supplemental group memberships you can use the -a switch to append the new supplemental groups instead of replacing them. If we were to use the -a switch the commands above would look like the following.

```
usermod -G wheel -a alice
usermod -G wheel -a bob
```

If we wanted to lock accounts we can use the -L switch. To unlock those accounts we can use the -U switch. To lock, then unlock Eve's account we might use the following commands.

```
usermod -L eve
usermod -U eve
```

If you use the commands fast enough then eve may never even notice.

## Command: groupmod

The **groupmod** command is used to change the name of a group. This is not a common action and will be rarely used. Before you change the name of a group it is best to know the gid of that group. If the group ID is **500** and you wanted to change the name of the group to "**groupies**" you could use the following command.

```
groupmod -g 500 -n groupies
```

## Command: userdel

The **userdel** command is used to delete user accounts. To delete the user account pass the user login name as the only argument to the command. To delete the user "**charles**" we could use the following command.

```
userdel charles
```

The only real problem with that command is that it leaves the user files on the system. These files are still owned by the **uid** of the former owner, but that owner no longer exists. It is possible that the next user created will be assigned the same number and will then have ownership of the files. If this is a

problem you can either delete the files or change the owner of the files to be some other user such as root.

If you want the home directory of the user deleted at the time you delete the account you can use the **-r** switch option. To delete the user "**charles**" and all of the files in his home directory you can use the following command.

```
userdel -r charles
```

## Command: groupdel

The **groupdel** command is used to delete groups. You just pass the name of the group as an argument to the groupdel command to delete the group. To delete a group "**games**" you could use the following command.

```
groupdel games
```

## Command: passwd

The passwd command us used to set and reset password. If you are logged in as a standard user you can use the passwd command to reset your own password, but you cannot reset a different user password unless you are logged in as the root user.

To change your own password just type passwd. The following is a dialog of changing your own password.

```
bash$ passwd
Changing password for user alice.
Changing password for alice.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

In this dialog, you are prompted for the current password. As you type it in, there are no echoed characters to indicate you are typing a password into the terminal. After you press Enter you are prompted for the new password twice.

As long as the system is comfortable with your password it will reset it.

If you are logged in as the root user, either directly, or by using su or sudo you can reset the passwords for other users on the system. Because the passwords are stored as encrypted hash values, the passwords cannot be easily recovered. The standard procedure is to reset passwords when they are forgotten or when accounts are created.

To reset the password for the user alice as root you would use the following command.

```
passwd alice
```

You would not be prompted for the current password, but would still be asked to supply the new password twice. The following is a dialog showing the process.

```
bash# passwd alice
Changing password for user alice.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

## *Quotas*

Disk quotas can be very helpful if you want to prevent users from taking all of the space on your system. The quotas can be tied to either users or groups. While creating disk quotas is not really difficult, it does require a couple of steps.

## Setting up Quotas

First, you need to make sure the file system is mounted with quota support. You need to edit the **/etc/fstab** file and add either the **usrquota** or **grpquota** option. The **usrquota** option is for user quotas and the **grpquota** option is for group quotas. Normally the mounting options column just lists the keyword defaults. To add user quotas to **/dev/sda3** we might have a resulting line in the **/etc/fstab** file that looks like the following.

```
/dev/sda3  /home  ext3   defaults,usrquota   0 0
```

Second, you need to create a file at the base of the file system that will be used to keep track of the quotas. If you wanted to create a user quota file at the base of the **/dev/sda3** file system that was mounted at **/home** you would create a file called **aquota.user** in the root of the file system. The file then needs to have permissions set to 600 and be owned by root. The following commands would accomplish this.

```
touch /home/aquota.user
chmod 600 /home/aquota.user
```

Third, you need remount the file system and repair the quota file. To remount the file system you can either reboot or run the following command.

```
mount -o remount /home
```

Repairing the quota file can be done with the **quotacheck** command. You can use the following command to repair your quota files.

```
quotacheck -vguma
```

Last, you need to turn your quotas on. You can use the **quotaon** command to turn the quotas on and the **quotaoff** command to turn them back off again. To turn on all quotas you could use the following command.

```
quotaon -av
```

If you wanted to just turn on the quotas for the **/home** file system you could use the following command instead.

```
quotaon -v /home
```

To turn the quotas off for all file systems you can use the **-av** switch options with the quotaoff command.

```
quotaoff -av
```

To just turn off the quotas for the **/home** file system you could use the following command.

```
quotaoff -v /home
```

## Command: edquota

You can set quotas for an individual user using the **edquota** command. To edit the quotas for the user **bob** we could use the following command.

```
edquota -u bob
```

You will be put into a text file where you can make changes to the amount of space bob is allowed to use. There are 6 fields each in 1k blocks. **Soft** and **hard** are the number of allowed **blocks** and **inodes**. When the user reaches the hard number all write operations are blocks. When the user reaches the soft limit a warning is sent to the user, but writes are still allowed. If either number is set to 0 then the limits are neither set nor enforced.

## Command: quota

You can see the quotas and usage using the **quota** command. You see quota reports for individual users using the **-u** switch option. The following command could be used to see how much space **bob** is

using.

```
quota -u bob
```

# Printing

Most Linux distributions use the CUPS system for printing. CUPS stands for Common Unix Printing System and operates as a service on TCP port 631. When CUPS is running, it is usually configured from a web browser. Normally, you just point a local browser to the following URL:

```
http://localhost:631/
```

Configuration is pretty easy, but you need to know where the printer is and how the printer wants to be communicated with. When configuring CUPS, you usually want to add printers and manage jobs. When CUPS prompts you for a username and password, it will accept the root user and password.

## *Running CUPS*

The CUPS service is usually configured to run as part of the default configuration. If it is not running, you can change its state using the cups.service.

## Service: cups

You will need to work with the cups.service to get CUPS started. Additionally, you might want to also enable the service if you want to make sure it runs after rebooting your machine. To start the CUPS service you would use the following command.

```
systemctl start cups.service
```

If you want to have the CUPS service start when you reboot your machine, you will need to make sure the service has been enabled. The following command will enable cups.service so that it will start at boot time.

```
systemctl enable cups.service
```

If you later decide that you do not want to run the CUPS service you would then need to **disable** the cups.service. If you are not sure of the status you can use either the **status** or **is-enabled** options.

## CUPS Firewall Rules

If you are not sharing your printers then you will not need to give access to anyone through the firewall. However, if you do want to share printers, you would allow communication to the CUPS port which is TCP port 631. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-port=631/tcp
```

If you wanted to make the rule permanent, you would additionally add the following line.

```
firewall-cmd --zone=public --permanent --add-port=631/tcp
```

# System Diagnostics

There are many different tools and methods for system diagnostics and troubleshooting. It is helpful to know more than a single method for diagnosing a problems and solving it.

## *System Logs*

The system logs are stored in the **/var/log/** directory. For security reasons many of these files can only be read by the root user, so standard accounts might not be able to effectively troubleshoot issues.

There are a couple of standard log files that you should be aware of. These files contain important troubleshooting information that will make your lives easier.

**/var/log/messages**

The messages file is the default file for most system logging events. When you start and stop services it is logged in the messages file. When things fail to start it is often logged here. Sometimes you have problems with .cgi or .php that reach a level where they are logged here. Sometimes you can look at the users and actions they have taken here in the /var/log/messages file.

**/var/log/secure**

Whenever you log into the system, or even attempt a login, it is logged in the secure log. When users or groups are created, deleted, etc. it is logged in the secure file. There are even logs in the secure file for every time a password is changed. When you suspect issues related to accounts, the /var/log/secure file is the first one that you should look at.

**/var/log/boot.log**

As you boot the system you start multiple different services. Because of the nice pretty screens and the boot speeds it is difficult to see the status of each service that was started. If you want to go back and look at what happened at boot time, you can look at the contents of the boot.log file.

**/var/log/dmesg**

In addition to services, hardware needs to be activated and started. The dmesg file contains a lot of important messages that can help you if you are trying to troubleshoot hardware issues. You can look at what the kernel did during boot time by reading through the /var/log/dmesg file.

After you have completed your boot of the system the dmesg messages are still generated, but are not stored in the /var/log/dmesg file. You can see a complete list of the messages by running the **dmesg** command.

**/var/log/audit/audit.log**

All kinds of things on your system are logged to the audit.log file. One common reason people go to the audit.log file for information is that it feels like SELinux might be blocking something. If you are worried about SELinux causing problems and want to verify it, you can search the audit.log file for any entries containing the sub-string “AVC”.

If you have the **setools-gui** package installed you can also use the **seaudit** command to start a GUI program that can open the audit.log files and look for SELinux exceptions.

**/var/log/yum.log**

Sometimes you install a lot of packages, but cannot remember which packages you installed and when

you installed them. The yum.log file keeps track of installed packages complete with date and time stamps so that you can figure out which packages where installed when. This is especially useful when you want to know if a packages was installed before or after you started seeing a problem.

If you just want to check for the existence of a package you can use the “rpm -qa” command and filter by piping the results to grep.

## Kernel Interface

The kernel has a filesystem style interface you can use to look different system characteristics. If you navigate to the **/proc/** directory and look around, you will find a lot of zero byte files that actually appear to contain information. When these files are viewed, the kernel retrieves the information and displays it to you.

There are some variables in here that can be modified as well. If you decide to modify a boolean value in a file you can do so by redirecting the output from a program like **echo** to the file name of the variable. If you are not comfortable using the echo command to change values, you can also do it with the **sysctl** command or by making configuration changes in the **/etc/sysctl.conf** file or by making your own .conf file in the **/etc/sysctl.d/** directory.

An example of a file in the /proc/ directory that you can change would be the file that tells the kernel that you want to allow traffic to pass through your machine. The /proc/sys/net/ipv4/ip_forward file contains a boolean value that incates if you will allow forwarding of traffic. To change the value you could use the following echo command.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

You could accomplish that same results using the sysctl command by starting after sys and replacing the slashes with dots and assigning a value.

```
sysctl net.ipv4.ip_forward=1
```

You could also put the line in the sysctl.conf file and reload the file. The line in the sysctl.conf file would look like this.

```
net.ipv4.ip_forward=1
```

You would reload the file using the following command.

```
sysctl -p
```

## Hardware Utilities

In addition to the system logs and the basic kernel interface, there are multiple utilities that can be used to perform system diagnostics, optimization, and configuration when working with hardware.

## Command: uname

The **uname** command is used for printing basic system information. When the command is executed without any arguments it normally just prints the operating system, which would be "Linux".

```
uname
```

If you wanted to display all information provided by the uname command you could use the **-a** switch. Below is a sample dialog.

```
bash$ uname -a
Linux example.com 3.10.0-229.1.2.el7.x86_64 #1 SMP Fri Mar 27 03:04:26 UTC 2015 x86_64
x86_64 x86_64 GNU/Linux
```

There are many examples on the Internet where someone uses the uname command. The most common is probably with the **-r** switch option. The -r switch shows the currently running version of the kernel. This is important when you are building something that needs to know the kernel version.

```
uname -r
```

## Command: lsusb

The **lsusb** command gives information about which usb devices are connected to your system. The basic usage of the command is without any options.

```
lsusb
```

If that does not give you enough information, you can add the **-v** switch. The -v switch gives a lot more information than most people want to see, but it does tell you a lot about the connected devices.

```
lsusb -v
```

You can filter out the output to only list some devices. If you are interested in a device 002 on bus 001 you can use the **-s** option. The following command would give verbose information about the device 002 on bus 001.

```
lsusb -v -s 1:2
```

## Command: lspci

The **lspci** command lists PCI buses and PCI bus connected devices. The basic command displays each item on a separate line.

```
lspci
```

If you want more information you can add the **-v** switch. If that is not enough, you can add another "**v**". If you still want more information, add a third "**v**" to the switch. The following are verbose options.

```
lspci -v
```

```
lspci -vv
```

```
lspci -vvv
```

If you are trying to build an optimized kernel, you can use the **-k** switch to see which kernel driver is being used by each PCI device. After you know which kernel driver each piece of hardware is using, you can switch that kernel from a module to being built-in.

```
lspci -k
```

## Command: lsmod

The **lsmod** shows the status of kernel modules. This command can be useful when you are trying to figure out which kernel drivers are being used. Sometimes you get into situations where there is more than one driver that can work with your hardware and the best one is not chosen. If this happens, one troubleshooting step would be to figure out which driver is being loaded.

```
lsmod
```

The loading of modules is usually automatic, but they can be manually loaded using the **insmod** command or the **modprobe** utility front end.

## Command: modprobe

The **modprobe** command is used to add and remove modules from the kernel. It used to also have the ability to see all modules available on you system with the **-l** switch option, but that feature disappeared in the switch from CentOS 6 to Cent OS 7.

To add modules you can use the **-a** switch option. The modprobe command then tries to load all

modules listed after the -a option. To load the e1000 kernel module you could use the following command.

```
modprobe -a e1000
```

The **-r** switch option is used to remove kernel modules. To remove the e1000 kernel module we could use the following command.

```
modprobe -r e1000
```

## Command: insmod

The **insmod** command is used to load kernel modules into the kernel. If you know the name of the module you pass it as an argument to the insmod command. To load the **e1000** networking kernel module you could use the following command.

```
insmod e1000
```

## Command: rmmod

The **rmmod** command is used to remove modules from the kernel. This is not always a safe thing to do, but usually works. To remove the **e1000** networking kernel module from the kernel we could use the following command.

```
rmmod e1000
```

# The Grub2 Boot Loader

During the boot process, you are briefly presented with the Grub2 Boot Loader screen. This software package is loaded by the BIOS and is responsible for making sure the operating system loads. On CentOS 7 machines there is usually a partition or directory available at **/boot** that contains all of the active boot files including the boot loader, the kernel, and the initial RAM disk.

## *Securing Grub2 with a Password*

When you are presented with the Grub2 screen, you can edit one of the boot options to cause it to boot without prompting for a root password. If you want to prevent people from making edits during the boot process, you can install a password in the **/boot/grub2/grub.cfg** file.

The first step in installing a password is to create the password. The **grub2-mkpasswd-pbkdf2** command is good for creating password hashes. Use the command to generate a password.

```
grub2-mkpasswd-pbkdf2
```

Once you have the password, copy it and edit the **/etc/grub.d/40_custom** file. You want to skip over all of the included lines and add the password and other instructions to the end.

The first thing you want to do is list the superusers who are allowed to make changes. This is done with the "**set superusers="root"**" command. After this you add line you can add a password. The syntax for the password is "**password_pbkdf2 USERNAME PASSWORDHASH**". For the USERNAME we are going to use root since we just added it to the list of superusers. For the PASSWORDHASH we use the password hash generated by the **grub2-mkpasswd-pbkd2** command.

Here are example lines added to the **/etc/grub.d/40_custom** file:

```
set superusers="root"
password_pbkdf2 root
grub.pbkdf2.sha512.10000.5A95BA6ED2798986FF84BD0FCB11D4A2592ED6DB2DE04D04418504C99C658D99
549FD1E3A17B17A473EF2E504D9D51723A01EFBC17BE61B4814E6DE721690E35.AEFA872C24FAEF32DDF76004
0429873271E607EC06F7124EE47EFA04B31C0531F983EC47E0D6C46C84352D736794710172073E4F42473E87F
3EF09DBBAC7A5B9
```

Once the file has been updated, it is time to rebuild the **/boot/grub2/grub.cfg** file using the **grub2-mkconfig** command. You can regenerate the file using the following line.

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

# Password Recovery

At times you may have problems with your system and need to take steps to recover the data or accounts. Normally, root can reset the passwords of any users, but what if the user you need the reset the password for is the root user? In times like this you need to go through the root password recovery process.

## *Root Password Recovery*

It is very important to always remember your root password. However, there are times when you may have forgotten a root password or the person who knew the password is no longer available. Because passwords are normally stored as password hashes instead of the actual password, you cannot really recover the passwords, instead you are going to replace the password.

## Cent OS 7 Systems

With Cent OS 7 there have been a lot of changes and the method of root password recovery has also changed. After booting you start with the grub boot loader screen. Use the up and down arrows to select the label to boot and to stop the countdown for default booting. Use the following instructions:

- With the OS selected, press “e”
- Scroll down and select the line that starts with “linux16” or “linuxefi”
- At the end of the line add a space and the following text “rd.break enforcing=0”
- Press Ctrl-x to boot the system

Your system should boot without SELinux enabled and will break after initramfs has completed doing what it wants to do. You should be given a root command prompt without being prompted for a root password.

Your file system will now be available, but will be mounted as read-only. In order to make changes we will need to remount the file system as read-write. The following command should take care of remounting:

```
mount -o remount -rw /sysroot
```

Now, the file system is mounted, but it is now mounted in a way that is easy to work with. We can use the chroot command to change where your terminal claims your / directory is at. Use the following command to change into the chroot environment:

```
chroot /sysroot
```

Now you will be in your file system's root directory and will be able to do most things that you could do in a fully running system. There will be small quirky things related to the kernel if you look around too much.

Now we want to change the root password. The following command should take care of this:

```
passwd root
```

Because the system uses SELinux and you edited a file, you probably just broke the context of the file. If you were to reboot right now, you would probably have a mess on your hands. We want SELinux to relabel the system to clear up and problems you just created. To cause the system to relabel the SELinux contexts we can create a file called **.autorelabel** at the root directory.

```
touch /.autorelabel
```

After doing this, we can exit the chroot environment and reboot the system. Use the following commands:

```
exit
reboot
```

The system will reboot and SELinux will take some time to relabel the security contexts for files on the main file system. You will eventually be presented with a login screen. At this point you should be able to log into the new system using the new root password.

If your system boots to a black screen without any text, then you most likely did not create the **.autorelabel** file correctly. To fix this, reboot and do the steps again.

## Older Linux Systems (grub)

After rebooting the system you start with the grub boot loader screen. Use the up and down arrows to stop the boot process. This allows time to select the label for the kernel you would like to boot. You then need to edit the options. Use the following steps to boot into single user mode:

- With the OS Selected, press “e”
- Select the line that starts with “kernel” and press “e”
- Add the word “single” to the end of the line and press Enter
- Press “b” to boot the system

When the system boots, it should drop you to a command prompt. You should not be prompted for a root password, but should have full root access. It is now time to change the root password. Use the following command to change the root password:

```
passwd root
```

After the password has been changed you may reboot the system and use the new root password.

## Older Linux Systems (LILO)

When you reach the boot loader screen you will be presented with a prompt where you can select your kernel to boot. Normally, people type “linux” and then boot the system.

At this point you are going to instruct the Linux kernel to load the init system, then drop into single user mode. At the prompt type the following:

```
linux single
```

This should bring you to a root prompt without requiring a password. At this point you can reset the root password.

```
passwd root
```

You can then either reboot your system using the reboot command or jump into the run level you want using the init command. Either of the following might do what you want.

```
reboot
```

```
init 5
```

### *Standard User Password Recovery*

If a normal user forgets their password the root user can just reset it with the **passwd** command. If the user alice forgot her password the root user could use the following command to reset her password.

```
passwd alice
```

# Services and Daemons

For a Linux to be fully functional it needs to have multiple services running in the background. These services are sometimes referred to as daemons. Because services run in the background and have the title daemon, the process that starts the background service commonly ends with the letter 'd'. Examples of background services include sshd, httpd, firewalld, mariadb, and postfix.

## *Service Status*

The services are controlled using the systemctl command. This command is used to control the systemd system. When the command is issued without any arguments, the active services and other items controlled by systemd are displayed along with their status. The output is displayed in a **more** type of window that allows you to view a single screen of information at a time. When you are done looking at the displayed information you can press the '**q**' key to exit back to the shell.

```
systemctl
```

If you want to know about additional services that are not active, you can use the **--all** argument option.

```
systemctl --all
```

If you know which service you are interested in learning about or for a more detailed version of the status information on a given service you can use the **status** option with the **systemctl** command. Just replace the keyword ***SERVICE*** below with the actual service name.

```
systemctl status SERVICE
```

The following dialog displays information about the **sshd** service.

```
bash# systemctl status sshd
sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since Sat 2015-08-08 12:09:55 PDT; 2 months 30 days ago
 Main PID: 1350 (sshd)
   CGroup: /system.slice/sshd.service
         +-- 1350 /usr/sbin/sshd -D

Nov 05 12:27:07 example.com sshd[4332]: Accepted password for backup from 10.10.10.10
port 53348 ssh2
Nov 05 12:27:30 example.com sshd[4364]: Accepted publickey for backup from 10.10.10.10
port 53349 ssh2: DSA d6:5f:92:cd:86:2c:b8:cc:...35:96:56
Nov 05 12:29:40 example.com sshd[4403]: Connection closed by 10.10.10.10 [preauth]
Hint: Some lines were ellipsized, use -l to show in full.
```

There is a lot of information displayed here, so we will go through the important parts. The first line shows that we ran the command "**systemctl status sshd**" at the root bash shell prompt.

After running the command, we are told that the **systemctl** command is displaying information about the **sshd.service** service and tells us that the service is called, "**OpenSSH server daemon**"

We are told that the service script is located at **/usr/lib/systemd/system/sshd.service** and that the service is currently **enabled**. Because it is enabled, the service will start automatically at boot time.

We can see that the service is currently active and running. The service has been running since Saturday August 8, 2015 at 12:09:55 PDT, when it was started. The service has been running for 2 months and 30 days.

The main PID or Process ID for the initial sshd service is 1350. This means that when the OpenSSH server was started, the service was assigned the pid or 1350. After starting it likely spawned (started) additional processes, but the initial process was and still is 1350.

After all of this status information, the service logs are displayed. We can see that the server is running on example.com and that there is a user called backup that logged in multiple times. It initially logged in using a password, then later logged in using a key (probably the authorized_keys file in the .ssh directory).

Normally, if your server is not very active you will see information about the service being started and listening to the IPv4 and IPv6 addresses. If you are listening on all IPv4 ports you will see the address 0.0.0.0 displayed. For any IPv6 address you will see :: displayed. Both IPv4 and IPv6 usually run on TCP port 22.

## Command: netstat

Sometimes when you are trying to figure out which services are running, it is best to see which ports are in the LISTEN state. An incantation that works pretty good is the following:

```
netstat -tapu
```

The order of the argument characters does not matter, so it is a good idea to arrange them in an order that will produce a word you can remember. Since tapu is the formal Polynesian cloth created from the insides of the mulberry tree bark, it is easy for me to remember.

Basically, the "t" indicates we are interested in things using TCP. "u" is for the UDP protocol. "a" indicates we want to start with all services, both listening and non-listening. The "p" indicates that we also want to know which process performed the bind operation that started the network connection.

Sometimes you do not want to see all of the names of services and protocols, just the services and port numbers that are being used. In this situation, the following argument options work well.

```
netstat -tuna
```

The 'n' displays numbers instead of names. If you wanted to add back the processes, you could add the "p" option, but then you would need to figure out how to order the letters to create a new word. Maybe

something like “-atnap” or “-at -nap”?

## Command: nmap

Looking at the listening processes from the inside of you machine can be a little tricky. Sometimes there are processes that are listening and appear open, but really are not accessible because they are either not listening on all interfaces or they are being blocked by a firewall or other device.

The **nmap** utility is very useful in troubleshooting which ports are available on the outside. To get the nmap utility you can use the yum command to install it.

```
yum install nmap
```

After nmap has been installed you can use it with an argument of the IP address or hostname of the machine to see which ports are open. To see what arguments and options are available, it is a good idea to run the nmap program without any arguments and see the help menu.

```
nmap
```

To look at your machine internally, you can use the nmap command with an argument of localhost.

```
nmap localhost
```

This will indicate some ports are open that are only listening on your local loopback, so make sure you interpret the returned data appropriately. You can also use the IP address of your machine to look at your machine. This can sometimes bypass the firewall, but does not always. Replace the IP address displayed below with your own IP address. (Note: scanning a machine with a firewall up might take a long time, if it is blocking ping request then you might need to add the -P0 option).

```
nmap 10.10.10.10
```

Once you have examined your server from a local point of view, it is good to use the nmap from a remote machine and scan your machine. The following is a dialog showing what a DNS server might display when performing a remote scan.

```
bash$ nmap 10.10.10.10
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2015-11-07 13:52 PST
Interesting ports on 10.10.10.10:
Not shown: 1676 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
111/tcp  open  rpcbind
2049/tcp open  nfs
```

```
Nmap finished: 1 IP address (1 host up) scanned in 6.658 seconds
```

We can see that ports 22, 53, 111, and 2049 are all open. If you knew ports 22 and 53 were supposed to be open, but were not quite sure why ports 111 and 2049 are open, then you could investigate and turn off and disable unused services.

## *Start/Stop Services*

You can start and stop services using the **start** and **stop** options using **systemctl**. The **sshd** service is a great service that allows administrators the ability to easily remotely manage their servers. Because it is such a great tool for managing remote servers, it is often the target of attacks.

If you wanted to start the sshd OpenSSH server daemon you could issue the following command.

```
systemctl start sshd
```

Normally, the sshd service is enabled be default, so instead of starting the service, you would likely be stopping the service. To stop the sshd service you can issue the following command.

```
systemctl stop sshd
```

Sometimes you want to continue using the service, but need to make changes to configuration files. Since the configuration files are loaded when the service is started, you usually need to restart the service to load the new configuration files. The following two possibilities work for restarting a service.

```
systemctl stop sshd
systemctl start sshd
```

```
systemctl restart sshd
```

The difference between the options is usually only minimal, but there can be large consequences. If you stop, then start a service, it is not available between the two commands. In reality, it might not be available during a restart either, but it can be worse with a stop/start combination.

If you are remotely logged into a server using **sshd** and decide to restart the sshd service, the stop and start options would not work since you cannot start the service remotely if you are using that service to send the start option.

OpenSSH server daemon does allow you to use the restart option while logged in and you can usually maintain your connection. If the configuration file has an error and the service does not start, you will likely lose your connection.

Some services, in addition to the start, stop, and restart options also have a reload option. If you use the

reload option, the service usually just reloads the configuration files and continues to run without issue. The following would attempt to reload the sshd configuration files.

```
systemctl reload sshd
```

The start, stop, restart, and reload all run silently unless they decide to report issues. System administrators who are used to the older init type of [OK], [Failed] reports can get a feeling of insecurity and might need to check the status of services to feel comfortable.

## *Enable/Disable Services*

Even if you are able to start and stop services, this does not have a lasting effect. When the machine reboots it will need to reload the services. If the services are marked as enabled then they will be start, but if they are not marked as enabled then they will not be started.

When we were looking at the status of a service we saw if it was enabled or disabled. If we wanted to just check to see if a service is enabled without getting all of the status information we can use the **is-enabled** option. The following could be used to determine if the sshd service is enabled.

```
systemctl is-enabled sshd
```

The command returns a single line with either the text "**enabled**" or "**disabled**". At this point you can decide how you want to make changes to the enabled/disabled status. To enable a service you can use the **enable** option. The following would enable the **sshd** service.

```
systemctl enable sshd
```

When you issue the command you can see that a symbolic link is created. There is a new symbolic link file in the **/etc/systemd/system/multi-user.target.wants/** directory created that points to the service in the **/usr/lib/systemd/system/** directory.

When the system then later starts, it only needs to look in the directory of the run level and see which symbolic links exist. It can them parse each script and start the services as part of the system start-up process.

To disable a service you only need to use the **disable** option with the service name. The following command would disable **sshd** from starting automatically when the system boots.

```
systemctl disable sshd
```

When the command is issued, the system will then remove the symbolic link that exists from the run level directory to the service scripts directory.

Neither the enable nor the disable command options will actually start or stop a service. They only

mark the service for future booting status.  To start or stop the service you will still need to use the start or stop options with the systemctl command.

# Secure Shell (SSH)

Secure Shell (SSH) is an encrypted network protocol that provides secure communicate over an insecure network. Many network administrators use ssh to log into remote servers and perform administration tasks at the command line. In addition to command line access, ssh can also be used to transfer files securely. Advanced ssh users can use the protocol to tunnel other protocols and even forward X Window applications from a server to a local client.

## *SSH Server Configuration*

The SSH server has a configuration file called **sshd_config** that is stored in the **/etc/ssh/** directory. This file contains a list of all of the possible directives and their settings. Lines that start with the hash mark are commented and are not being set. Directives without a hash mark are being overridden from their default values.

SSH lists the directives with their default values, and then comments them out. If you want to change a directive from the default value you need to uncomment the line and then change the value. If you do not do both the uncommenting and changing of values then the directive will remain unchanged.

## *SSH Client Configuration*

The SSH client has a configuration file called **ssh_config** that is also stored in the **/etc/ssh/** directory. This file follows the same basic idea at the server configuration file in that default directive values are listed and commented out.

## *Running the SSH Server*

You can start the services using the following command:

```
systemctl start sshd.service
```

On many CentOS 7 systems the services will already be running, so this might not be required. If you need to make sure the services are started automatically at boot time you can enable the services. Once again, the OpenSSH server should be running be default.

```
systemctl enable sshd.service
```

If you need to stop the services, you can use the following commands.

```
systemctl stop sshd.service
```

When you make configuration changes, you can either restart the OpenSSH server or reload the

configurations using the reload option. Either of these commands so take care of loading new configurations.

```
systemctl restart sshd.service
```

OR

```
systemctl reload sshd.service
```

If the configuration changes are more than just configuration files, such as when you upgrade the OpenSSH software, you will want to make sure you restart the service.

## Command: ssh

The **ssh** command is used to connect to remote machines and provides a terminal. To log into a remote we can use the **ssh** command and pass the username and hostname as an argument with the two values separated with the at sign. To log into **alice**'s account on the **example.com** server we might use the following command.

```
ssh alice@example.com
```

If you have never communicated with this server before, you will be presented with information about the key and asked if you want to add it to your list of known hosts. The **known_hosts** file is stored in the user's **.ssh/** directory. After accepting the key, the system will attempt to authenticate you with either a key or a password.

When you are done communicating with the remote system you can type the **exit** command to leave and return to your own system.

## Command: scp

The **scp** command is used to securely copy files using the ssh protocol. The **scp** command works basically the same as the **cp** command, but allows for remote machines to be used instead. If **alice** wanted to copy a file called **report.txt** to **bob**'s home directory on the server **files.example.com** then she could use the following command.

```
scp report.txt bob@files.example.com:.
```

If she wanted to copy the same file to **/var/www/html/** then she could use the following command. This assumes bob has permissions to write to the directory.

```
scp report.txt bob@files.example.com:/var/www/html/
```

### *Key Based Authentication*

Normally people use user names and passwords to authenticate with remote servers. This works, but sometimes you want to use keys instead or in addition to the passwords. The SSH protocol allows for connections to remote servers using keys and optionally a passphrase.

## Command: ssh-keygen

The **ssh-keygen** command creates RSA public and private key pairs. When you run the command you are prompted for file names and a passphrase. If you want a client machine to be able to connect to a remote server without typing a password or passphrase then you will need to leave the passphrase blank. Also, if you change the name of the files being created then you might not have everything work as expected.

The following is a dialog showing the creation of the SSH keys.

```
bash$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/alice/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alice/.ssh/id_rsa.
Your public key has been saved in /home/alice/.ssh/id_rsa.pub.
The key fingerprint is:
ac:b9:35:a4:d4:42:73:89:db:93:2b:5c:61:7d:75:23 alice@example.com
The key's randomart image is:
+--[ RSA 2048]----+
|             E...|
|       . o   ....|
|      + = . .    |
|     . O o .     |
|      + S        |
|     o B o       |
|      * +        |
|       + .       |
|      .          |
+-----------------+
```

After the command completes you should have two new files. The **id_rsa** file contains the private key and the **id_rsa.pub** file contains the public key. If you want to be able to log into a remote machine using your key, you need to get the contents of your **id_rsa.pub** file into the **authorized_keys** file in the **.ssh/** directory inside the user home directory. If you are trying to log into the **root** account you would need to get your key into the **/root/.ssh/authorized_keys** file.

## Command: ssh-copy-id

In order to get your public key copied over to the server, you can use the **ssh-copy-id** command. This program opens a connection to the server, prompts you for any required passwords, then updates the **authorized_keys** file and puts your public key in that file.

Since the file is updated, you should be able to run the following command twice in a row with different results. The first time will prompt you for a password. The second time will skip the password prompting and will either directly let you in, or if you set a passphrase for your public/private key pair, it will prompt you for the passphrase.

```
ssh-copy-id charles@example.com
```

### *Security*

Because SSH allows someone to have direct command line access on your system, it is important to make sure it is configured correctly and the correct people are able to connect. There are a couple of decisions you can make that will affect how safe and secure your SSH server and ultimately your whole system will be.

## Configurations

The most commonly attacked service is probably SSH. Most of the time it is just random people on the Internet scanning for machines and automatically trying a list of user and password combinations that are known to be vulnerable.

#### Disabling Root Logins

Because many systems have easy to guess root passwords, it is common to attempt a login using the root account.

If you want to prevent someone from guessing the root password, it is usually best to disable root SSH logins. This does not disable to account, but prevents people from logging using root from remote machines. Once root logins are disabled, you would have to login with a normal user and then use **su** or **sudo** to escalate permissions as needed.

You can disable root logins by editing the **/etc/ssh/sshd_config** file and find the following line.

```
#PermitRootLogin yes
```

Uncomment that line by removing the leading hash mark and change the value **yes** to **no**. The resulting line should look like the following.

```
PermitRootLogin no
```

After changing the configuration file, restart the **sshd.service** daemon to make sure the new configurations are loaded correctly.

```
systemctl restart sshd.service
```

## Firewall

The SSH service operates on TCP port 22. Because users are expected to use SSH, the service is running and the firewall ports are open by default. If you wanted to add the service to the firewall, you could do so. The following commands would allow the services through the firewall.

```
firewall-cmd --zone=public --add-service=ssh
```

If you wanted to make sure the SSH service is allowed through the firewall the next time you boot your system or restart your firewall, you can add the **--permanent** option to the command line.

```
firewall-cmd --zone=public --add-service=ssh --permanent
```

## SELinux

SSH should be able to run normally, but sometimes people move files into directories instead of copying them and the files retain their original SELinux security context. If you get into this situation, you can use the **restorecon** command on any files to reset their values.

All of the files in the **.ssh/** directory and the directory itself should all have "ssh" as a substring in the context type. If you have something like "admin" or "home" as a substring, you will want to change the context.

## *Troubleshooting*

The SSH service is usually installed and running by default on CentOS machines, so many of the following steps can be may not seem necessary when troubleshooting, however, they are listed below for the sake of completeness.

- Verify your IP address is correct (ip addr)
- Verify services are running (netstat)
- Verify the firewall is not in the way (firewall-cmd --list-all)
- Verify SELinux is set correctly. (ls -alZ)
- Verify remote host is up (ping)
- Verify remote ports are open (nmap)
- Check the logs (/var/log/secure)

- If you get the error: “Agent admitted failure to sign using the key.” try running the command **ssh-add** to add your keys to the key chain.

# Network File System (NFS)

The **Network File System** (NFS) protocol was originally developed by Sun Microsystems. With the NFS protocol you can export directories and mount them on different machines as if they were local. This is commonly used for shared storage directories and also for exporting the **/home/** directory in lab or workstation types of situations.

To make this all work, Sun used the **Open Network Computing Remote Procedure Call** system. Because of this, in order to use NFS, you need to have both **nfs-server** and **rpcbind** services running on the server machine.

Additionally, the nfs-mountd service provides assistance in figuring out which directories are being exported and connecting with them.

## *Running NFS*

You can start the services using the following command:

```
systemctl start rpcbind.service
systemctl start nfs-server.service
systemctl start nfs-mountd.service
```

On many CentOS 7 systems the services will already be running, so this might not be required. If you need to make sure the services are started automatically at boot time you can enable the services.

```
systemctl enable rpcbind.service
systemctl enable nfs-server.service
systemctl enable nfs-mountd.service
```

If you need to stop the services, you can use the following commands.

```
systemctl stop nfs-server.service
systemctl stop rpcbind.service
systemctl stop nfs-mountd.service
```

## *Exporting*

When you are trying to export directories, there are two places you can create configuration entries. The most common place for configurations is the **/etc/exports** file. If you do not like listing all of your exported directories in the same file then you can put them in separate text files with the **.exports** file extension in the **/etc/exports.d/** directory. When you are ready to export the directories you can use the **exportfs** command.

## File: /etc/exports

Each line in the **/etc/exports** file indicates a directory to be exported or shared. The format of the line

is simple, first the directory is listed, then there is some white space, then the machine and options used when exporting.

If you wanted to export the **/data/** directory to all machines with normal read/write permissions you could use the following line in the /etc/exports file.

```
/data/ *(rw)
```

Even though the above line exports the directory as read/write, it is not always possible for the client users to read and write the files. When exporting, NFS also exports file ownership and permissions. Since files all have **uid** and **gid** values associated with them, the files appear owned by the matching uid and gid of the client machine.

System administrators exporting files this way usually try to synchronize the **/etc/passwd** files or use a service such as **NIS** or **LDAP** to make sure the uid and gid values for their users match even on different machines.

For safety, some uid and gid values are changed in the export process. To keep things safe, the files owned by root (uid of 0) are usually changed to a different uid value. This is called **root_squash**. To prevent this behavior, you could use the **no_root_squash** option in addition to the **rw** option. To export the **/data/** directory to all users with the uids preserved for even the root user, you could use the following line.

```
/data/ *(rw,no_root_squash)
```

You can also control the machines you export a directory to. Instead of the "*" wildcard, you can use a hostname or an IP address. You can also use IP address ranges. If we wanted to export the /data/ directory to all machines in the **192.168.0.0/24** network then we could use the following line.

```
/data/ 192.168.0.0/24(rw)
```

There are multiple other options for exporting that are available by viewing the man pages for the exports file.

```
man exports
```

## Command: exportfs

Once you have your **/etc/exports** file or **/etc/exports.d/** files created, it is time to export the directories. You can export the directories using the **exportfs** command. The easiest incantation of this command is using the **-a** switch option for exporting all directories.

```
exportfs -a
```

If you want to see the current export list, you can use the **-s** switch option. The output text format it will be compatible with the /etc/exports file.

```
exportfs -s
```

### *NFS Clients*

For the NFS protocol to really be useful, you need to have clients using it. On a client machine you can mount NFS shares using either the command line or a line in the **/etc/fstab** file. The command line is great for testing configurations. After you know you can mount the file system successfully, it is a good idea to make the configuration permanent by placing a line in the /etc/fstab file so that you do not need to manually mount the system anymore.

To mount a file system using NFS the following conditions must be met.

- The file system must be exported by the server
- You must have a local mount point
- You need to have the required services running

It is good to know how to verify each of these steps so that you can mount the network file system and also be able to troubleshoot problems in the future.

## Command: showmount

The **showmount** command does what the name implies, it shows you possible NFS mount points. You can use it with the **-e** switch option and a hostname or IP address to see which directories are being exported and are available for mounting. If you were on a client machine and wanted to mount a share exported from **server.example.com** you might check the possible mount points using the following command.

```
showmount -e server.example.com
```

The return text might look something like the following.

```
Export list for server.example.com:
/data *
```

With the above results we would know that the machine **server.example.com** was exporting the **/data/** directory.

## Mounting an NFS share

To mount an NFS share you first need to have a mount point. It is still common for system

administrators to put their own mount points in the **/mnt/** directory. This is not a required location, but it is common practice.

One possible mount point for the exported **/data/** directory to be mounted at would be a **/mnt/data/** directory on the client machine. The client machine should already have a /mnt/ directory so you should be able to create a /mnt/data directory easily using the following command.

```
mkdir /mnt/data
```

After creating the directory you should be able to mount the /data/ on server.example.com to the /mnt/data mount point using the following command.

```
mount server.example.com:/data /mnt/data
```

When mounting the NFS file system, you can add additional options using the **-o** switch options. You can also specify that the file system is NFS using the "**-t nfs**" arguments. There are defaults, so these are usually used when you are not using defaults. The following command would mount the same directory with the same default options.

```
mount -t nfs -o rw server.example.com:/data /mnt/data
```

## Mount NFS Shares with /etc/fstab

You can have your machine automatically mount NFS directories by adding a line in the **/etc/fstab** file. The following line could be added to the **/etc/fstab** file to mount the exported **/data/** directory from **server.example.com** on a client machine with a mount point of the **/mnt/data/** directory.

```
server.example.com:/data       /mnt/data     nfs    defaults      0 0
```

In the above line, **server.example.com:/data** is considered the device. The mount point is **/mnt/data**. The file system type is **nfs**. The options are the **defaults**. The last two zeros are there because we do not perform file system checks or do backups of remote file systems.

The only things you can really change are the arguments used when mounting the file system.

## NFS Firewall Rules

In order to really use NFS you will want to make sure the firewall is open for external connections. Since you need multiple services in order to run NFS and connect to outside servers you will also need the services available on the outside. The following commands would allow the services through the firewall.

```
firewall-cmd --zone=public --add-service=rpc-bind
firewall-cmd --zone=public --add-service=nfs
firewall-cmd --zone=public --add-service=mountd
```

If you wanted to make the rules permanent, you would additionally add the following lines.

```
firewall-cmd --zone=public --permanent --add-service=rpc-bind
firewall-cmd --zone=public --permanent --add-service=nfs
firewall-cmd --zone=public --permanent --add-service=mountd
```

## NFS Troubleshooting

Sometimes you cannot mount NFS shares because something is configured incorrectly. Sometimes you can mount the share, but things do not work correctly. Below are a couple of troubleshooting questions you can ask yourself to make sure everything is configured the way it is supposed to be.

- Are the IP addresses on the client and server both configured correctly?
- Can the client and server both ping each other?
- If you have a hostname in the /etc/exports file, can you resolve that hostname with either your /etc/hosts file or the dns?
- If using a hostname in the mount command, can you resolve that hostname?
- Has rpcbind.service been started? Is it still running?
- Has nfs-server.service been started? Is it still running?
- Is the firewall allowing both the rpc-bind and nfs services through?
- Can you see the services using nmap against the server from the client machine?
- Is the share being exported correctly from the server?
- Does “showmount -e SERVER” show you the share in the exported list?
- Has the mount point been created on the client machine?
- Is the mount command being run as the root user or with a command like sudo which gives root privileges?
- Was the /etc/exports file configured to share with root_squash or similar options?
- Are the directory permissions on the exported directory set correctly?

# Samba (SMB)

The Samba program is a free software implementation of the SMB/CIFS protocol. Samba is probably most well known for and used for the ability to provide directory shares, but it does a lot more. In addition to sharing directories with Windows and Linux clients Samba servers can also run as Active Directory domain controllers and as a member.

## *Samba Installation*

To install Samba you will need to install the samba package. Use the following line to install Samba.

```
yum install samba
```

## *Samba Configuration*

The main Samba configuration is the **/etc/samba/smb.conf** file. The file comes prepopulated with active configurations and some that are commented out. There are two different characters used to indicate the line they lead is a comment. The hash mark "**#**" usually indicates a comment that really is a comment. The semicolon "**;**" indicates a comment that is really an inactive configuration option.

## Global Section

Each section starts with a name in braces "[]". The first section is the global section. Since Samba allows your machine to pretend it is a regular Microsoft Windows system, you need to have a few of the variables that those machines have.

Normally you would configure a workgroup, server string, and netbios name. If you were setting up the only server for a bakery and Samba was going to be used as the shared location for company documents you might configure the machine name as such.

```
[global]
workgroup = bakery
server string = Main Server
netbios name = Server
```

All of your client workstations would then join the "bakery" workgroup and should be able to see the machine listed as "Server". Unfortunately, Windows never really did figure out the whole finding other machines in a timely manor, so to speed things up a bit, you might also consider running your Samba machine as a WINS server. If you decide to run as a WINS server you will need to turn on wins support. To do so you would need to uncomment the following line.

```
wins support = yes
```

To actually use your Samba machine as a WINS server you will need to configure the client workstations to use your server's IP address as their WINS server.

## Share Definitions

There are a couple of share definitions already listed in the configuration file that you can use as examples. Some such as the **[homes]** and **[printers]** definitions have special meaning.

If you want users of your server to be able to view their home directories you can leave the **[homes]** share definition alone and just activate the SELinux boolean variable later. If you you do not want it, you should probably disable it using the semicolon ";" to comment out the commands and share name.

If you want to be able to use your machine as a print server for client machines then you might want to keep the **[printers]** definition. The rest of the configurations are earlier in the configuration file, but generally, Samba forwards your print requests to the CUPS print service when allowed to. If you want to disable this section you can comment all of the command directives out using the semicolon ";" character before each line of the definition.

To create new definitions you need to create a new section with the name in braces "[]", then list the options below it. Continuing with our bakery example from earlier we want to create a share called documents that shares the files in the **/documents** directory. The following would take care of this.

```
[documents]
comment = Bakery Shared Documents
path = /documents
public = yes
writable = yes
printable = no
```

After creating this share definition you would then need to make sure the **/documents** directory exists and has SELinux context and permissions set correctly. Samba shares should have the context type set to **samba_share_t** with **chcon**. You can experiment with different permissions to see what works for you. You can be sure the directory permissions will not be a problem if the directory is set to 777 using **chmod**.

## Samba Users

To connect to Samba shares you need to have a Samba user. If you active the home directories then you can use the regular users on your system, however, you will need to set a Samba password for each one. To test, it is probably easiest if you create a password for the nobody user and use that. To set the nobody Samba password use the following line.

```
smbpasswd -a nobody
```

For each additional user you would use the same **smbpasswd** command with the -a option. Since the home directories are turned on by default, each regular user will at least have access to their home directories when connecting to the share.

## *Running Samba*

In order to run Samba you will need to configure and start the Samba service, make any required

changes to the firewall rules, and change the SELinux contexts of shared directories. Once all of this is done, clients should be able to connect.

## Service: smb

You will need to work with the smb.service to get Samba started. Additionally, you might want to also enable the service if you want to make sure it runs after rebooting your machine. To start the Samba service you would use the following command.

```
systemctl start smb.service
```

Every time you make configuration changes you will need to either **stop** and **start** or **restart** the smb service. If you want to have the Samba service start when you reboot your machine, you will need to make sure the service has been enabled. The following command will enable smb.service so that it will start at boot time.

```
systemctl enable smb.service
```

If you later decide that you do not want to run the Samba service you would then need to **disable** the smb.service. If you are not sure of the status you can use either the **status** or **is-enabled** options.

## Samba Firewall Rules

To use Samba you will need to make sure the service is available from the outside. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-service=samba
```

If you wanted to make the rule permanent, you would additionally add the following line.

```
firewall-cmd --zone=public --permanent --add-service=samba
```

If instead of the service you would like to allow the port numbers you can add both of them using the following commands.

```
firewall-cmd --zone=public --add-port=139/tcp
firewall-cmd --zone=public --add-port=445/tcp
```

After adding the ports to the active firewall, you would then probably add them to the permanent firewall with the permanent option.

```
firewall-cmd --zone=public --permanent --add-port=139/tcp
firewall-cmd --zone=public --permanent --add-port=445/tcp
```

## Samba SELinux Configuration

When you create Samba shares, you need to make sure the SELinux context for those directories is set correctly. The Samba server is allowed to read directories that have been marked as the **samba_share_t** type. To make a directory readable you can use the following **chcon** command.

```
chcon -t samba_share_t /directory
```

If you decide to allow users to use their own home directories you will need to active them for reading. There is already a set of disabled SELinux rules for reading user home directories, but you need to switch the boolean switch with the **setsebool** command to activate them. The following command will make it possible for Samba to read user home directories.

```
setsebool -P samba_enable_home_dirs on
```

There are additional steps and boolean switches for more complex configurations such as domain controllers. Some of this is listed in the comments of the Samba configuration file.

### *Accessing Samba Shares*

From a client machine you can see which Samba shares are available if you know the address of the server and a username/password set to connect. The **smbclient** command can connect to and query the server. Replace the SERVER and USERNAME in the following command to see the shares.

```
smbclient -U USERNAME -L SERVER
```

You will be prompted for a password before you can see the shares. This password is the one you set using the **smbpasswd** command. If you do not see the shares you intended to share, verify the configuration and make sure the smb service has been restarted.

## Mounting Samba Shares Manually

When mounting you always need a mount point. When you decide to mount a Samba share you will additionally need a Samba user account on the server machine. The device name is made up of the server address, and the share name. In the options you would put the username and possibly password. If you skip the username, the mount command will assume you mean the root user. The following command could be used to mount a Samba share.

```
mount //SERVER/SHARE /mnt/point -o user=USERNAME
```

Remember that you need to replace SERVER, SHARE, and USERNAME with real values and you also need to have a mount point. This will prompt you for a password before mounting the directory share. To not be prompted for a password you can add it into the command.

```
mount //SERVER/SHARE /mnt/point -o user=USERNAME,password=PASSWORD
```

If you feel uncomfortable putting the password on a command line you can put it in a text file for credentials. The credentials file has the following format.

```
username=USERNAME
password=PASSWORD
```

You would then use the credentials option when mounting. If the credentials text file were stored at **/etc/samba/credentials.txt** you could use the following line.

```
mount //SERVER/SHARE /mnt/point -o credentials=/etc/samba/credentials.txt
```

## Mounting Samba Shares using /etc/fstab

Once you are comfortable in knowing you are able to successfully mount a Samba share, you can take it one step further and configure your machine to automatically mount the Samba share at boot time by putting it in the **/etc/fstab** file.

You can add the share by appending a line like the following to the end of the **/etc/fstab** file.

```
//SERVER/SHARE /mnt/point cifs credentials=/etc/samba/credentials.txt 0 0
```

Once you have added the line, you should try manually mounting the share using just the device name or the mount point. The mount command should be able to figure out what you mean by looking up the entry in the **/etc/fstab** file. Either of the following commands should work.

```
mount //SERVER/SHARE
```

Or

```
mount /mnt/point
```

## *Samba Troubleshooting*

If you are having problems with Samba you can ask yourself the following questions to help diagnose the problem.

- Are the IP addresses on the client and server both configured correctly?
- Can the client and server both ping each other?
- Did you install the samba package before trying to start the server?
- Did you start the smb service?
- If you use “netstat -tuna” are both tcp/139 and tcp/445 listed as open and listening?
- If you use nmap against localhost do you see both netbios-ssn and microsoft-ds listed as open?
- Is the firewall allowing both tcp/139 and tcp/445 or the samba service through?
- Can you see the services using nmap against the server from the client machine?
- Did you create the desired share(s)? Are they uncommented?
- Did you create the share directory/directories?
- Is the SELinux context for the directories set correctly?
- Are the permissions on the directory set correctly on the server side?
- Linux Client: Does “smbclient -U nobody -L SERVER” show the share(s)?
- Linux Client: Does “smbclient -U nobody //SERVER/SHARE” connect to the share?
- Linux Client: Has the mount point been created on the client machine?
- Linux Client: Is the mount command being run as the root user or with a command like sudo which gives root privileges?
- Windows Client: Did you set the WINS server address? (Usually on WINS tab of IPv4 “Advanced” settings)
- Windows Client: Does typing “\\SERVER\SHARE” in the run field or File Explorer URL do anything? (Note: SERVER can be either the name or IP address)

# Apache Web Server (HTTP)

The Apache Web Server has been a very important service on Linux machines for a long time. For a number of years the Apache server software was running on more then three quarters of the web servers on the Internet. In recent years there have been a number of decent competitors which have come out and started to compete, but Apache is still strong and doing well.

## *Apache Installation*

In order to run the Apache Web Server you need to install the **httpd** package. If you want to run https you would also want to install the **mod_ssl** and possibly **openssl** to generate a private key and Certificate Signing Request. Because of recent security concerns, many companies are starting to run their servers as https and have the http requests redirected to the https service.

To install the above listed packages you can run the following command.

```
yum install httpd mod_ssl openssl
```

## *Basic Apache Configuration*

The basic configuration files are found in the **/etc/httpd/** directory. In that directory there are two important directories the **conf/** and **conf.d/** directories. The **/etc/httpd/conf/** directory contains a file **httpd.conf** which includes a directive to load all of the files ending with **.conf** that are in the **/etc/httpd/conf.d/** directory.

There are a lot of different configuration changes that you can make in the httpd.conf file, so it is usually a good idea to browse through the file and get an idea of the possibilities. The httpd.conf file included with the Apache packages is full of well written comments and documentation.

## User Home Directory SELinux Configurations

In order to use user home directories with Apache you will need to configure them to be available. The first step is to set the boolean value in SELinux that manages access to the home directories. The following **setsebool** command should do this.

```
setsebool -P httpd_enable_homedirs true
```

If you already have **public_html** directories in place you might have to change their context to make everything work. The following **chcon** command will change the directory tree of a given USERNAME to the correct SELinux context type of **httpd_user_content_t** for static pages.

```
chcon -R -t httpd_user_content_t /home/USERNAME/public_html
```

Keep in mind that CGI scripts will still need to be configured to use an appropriate context type. CGI scripts in the homedirs will require the **httpd_user_script_exec_t** type in order to run.

Once you have the directories and they have the correct context, you will need to make sure the Apache web server is able to get to them. Since the Apache server does not run as root, it will no have automatic access to all directories. It will need at least execute access on the user's home directory and will need both read and execute access on the public_html directory inside in order to see the default pages. The following commands will give Apache access for USERNAME.

```
chmod 711 /home/USERNAME
chmod 755 /home/USERNAME/public_html
```

### *HTTPS Configuration*

In addition to running as a standard HTTP web server on tcp port 80, Apache can also run the HTTPS protocol on tcp port 443. Before running HTTPS you need to make sure you have the **mod_ssl** and **openssl** packages installed.

```
yum install mod_ssl openssl
```

## Generating Keys, Requests, and Certificates

In order to start the HTTPS part of your server, you need to have a private key you can use to sign messages going out. You also need to have a signed public certificate that clients can use to verify your identity and decrypt your signed messages.

To create your initial key you need to use the **openssl** command. This command can help you create your own key, the request for signing, and even self signed certificates if you so desire. Use the following command to create your key. Remember to replace the **HOSTNAME** with your actual hostname.

```
openssl genrsa -out HOSTNAME.key 2048
```

Once you have the key, it is time to generate the **Certificate Signing Request (CSR)**. The request is usually sent to a **Certificate Authority** for signing. There are many different signing authorities that will usually sign your certificate for a fee. The following command will generate a CSR file that you can present to the Certificate Authority along with your money for your certificate.

```
openssl req -new -key HOSTNAME.key -out HOSTNAME.csr
```

If you decide that you do not want to pay for a certificate, you can generate and sign your own. A self signed certificate will likely result in a browser warning the user that the sign is not trusted. If this is not a big concern, you can go ahead and get your certificate signed by yourself.

The following command will take your Certificate Signing Request and sign it with your own key. The end result will be a Certificate that can be used on your website.

```
openssl x509 -req -days 365 -in HOSTNAME.csr -signkey HOSTNAME.key -out HOSTNAME.crt
```

If you did all of the steps correctly, you should now have three files that you can use. You will have a **.key**, a **.csr**, and a **.crt** file. You will not need the **.csr** file for further configuration, but it is probably a good idea to keep it around.

If you wanted to skip the whole CSR generation and just create a self signed certificate and key with a single command, the following command should do the trick. Note that the "\" character can be omitted if you do not press the enter key after it.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout HOSTNAME.key \
    -out HOSTNAME.crt
```

## Installing the Certificate

Once you have your key and your certificate, it is time to install them and get the server up and going. The key and the certificate are supposed to be in specific locations on your system and the configuration file indicates where. Edit the ssl configuration information. This is sometimes in the **/etc/httpd/conf/httpd.conf** file, but will more likely be in the **/etc/httpd/conf.d/ssl.conf** file, which is imported into the httpd.conf file when Apache loads.

You will want to look for the **SSLCertificateFile** and **SSLCertificateKeyFile** directives. Both of the directives should indicate a localhost default file. You will want to take a note of the location of the files and then replace the localhost portion of the file name with your hostname or name of your key and certificate files. The resulting directives would look something like the following, except with hostnames instead of the word HOSTNAME.

```
SSLCertificateFile /etc/pki/tls/certs/HOSTNAME.crt
```

```
SSLCertificateKeyFile /etc/pki/tls/private/HOSTNAME.key
```

You now need to get the files into the correct location. Since SELinux is very touchy with the Apache web service, you need to make sure the files are copied into the correct location and not moved. The following commands should do the job.

```
cp HOSTNAME.crt /etc/pki/tls/certs/
cp HOSTNAME.key /etc/pki/tls/private/
cp HOSTNAME.csr /etc/pki/tls/private/
```

If you find you moved the files instead of copying them, you might not have the correct SELinux

context type of **cert_t**. You can use the **restorecon** command to try to fix their SELinux contexts.

```
restorecon /etc/pki/tls/certs/HOSTNAME.crt
restorecon /etc/pki/tls/private/HOSTNAME.key
```

Both files will have the SELinux **cert_t** type and should have permissions set to 600.

## Disabling Weak Protocols

Some of the encryption protocols available in Apache are insecure. If you allow client web browsers to tell your web server that they are unable to do complex encryption, you might have a situation where your web server is asked to use a protocol that has known vulnerabilities. If you wanted to disable both SSL version 2 and SSL version 3 you can edit the /etc/httpd/conf.d/ssl.conf file and make sure the SSLProtocol directive looks like the following.

```
SSLProtocol all -SSLv2 -SSLv3
```

### *Starting Apache*

The Apache web server already has a basic configuration file in place that will allow you to serve pages. After you have made any additional changes you can go ahead and start the httpd.service, make any required changes to the firewall rules, and change the SELinux contexts of any new directories you designate for serving pages. Once all of this is done, client browsers should be able to connect.

## Service: httpd

You will need to work with the **httpd.service** to get Apache started. Additionally, you might want to also **enable** the service if you want to make sure it runs after rebooting your machine. To start the Apache service you would use the following command.

```
systemctl start httpd.service
```

Every time you make configuration changes you will need to either **stop** and **start**, **restart**, or **reload** the **httpd.service**. To restart the Apache web service run the following command.

```
systemctl restart httpd.service
```

If you want to have the Apache service start when you reboot your machine, you will need to make sure the service has been enabled. The following command will enable **httpd.service** so that it will start at boot time.

```
systemctl enable httpd.service
```

If you later decide that you do not want to run the Apache service you would then need to **disable** the httpd.service. If you are not sure of the status you can use either the **status** or **is-enabled** options.

## Apache Firewall Rules

To use Apache to the fullest you will need to make sure the services are available from the outside. The following firewall rule commands will make this possible.

```
firewall-cmd --zone=public --add-service=http
firewall-cmd --zone=public --add-service=https
```

If instead of opening up for the http and https services you want to open the TCP port numbers you can use these lines instead.

```
firewall-cmd --zone=public --add-port=80/tcp
firewall-cmd --zone=public --add-port=443/tcp
```

If you wanted to make the rules permanent, you would additionally add the **--permanent** option. The following lines do this for the services.

```
firewall-cmd --zone=public --permanent --add-service=http
firewall-cmd --zone=public --permanent --add-service=https
```

## Apache SELinux Configuration

The Apache service is a very public facing service that is commonly a target of malicious actors. To reduce the risks associated with using Apache it has been locked down using SELinux. When you add additional files or modules to Apache it is usually safest if you copy files instead of moving them because you want to inherit the new context instead of bringing the old one along.

If you get into a situation where you are unsure of what the context of a file is supposed to be you can use the following table to get you started.

| Absolute Directory | SELinux Context |
|---|---|
| /etc/httpd/conf/ | httpd_config_t |
| /usr/lib/httpd/modules<br>/usr/lib64/httpd/modules | httpd_modules_t |
| /var/log/httpd/ | httpd_log_t |

| | |
|---|---|
| /var/www/cgi-bin/ | httpd_sys_script_exec_t |
| /var/www/html/ | httpd_sys_content_t |

You can change the context of a directory or file using the **chcon** program with the correct context. You can additionally use the **restorecon** command if you do not want to type the context. After using **restorecon** you want to still verify the directory to see if it was set correctly.

If you create CGI scripts in directories other than /var/www/cgi-bin/ you will need to make sure they have the correct type. For normal directories you will need to assign them the SELinux type of **httpd_sys_script_exec_t** in order to allow them to run.

## *Apache Troubleshooting*

If you are having problems with Apache Web Server you can ask yourself the following questions to help diagnose the problem.

- Are the IP addresses on the client and server both configured correctly?
- Can the client and server both ping each other?
- Did you install the required packages package before trying to start the server?
- Did you start the httpd service?
- If errors were reported at start time, did you check the /var/log/messages file?
- If you use “netstat -tuna” are both 80/tcp and 443/tcp listed as open and listening?
- If you use nmap against localhost do you see both http and https listed as open?
- Is the firewall allowing 80/tcp and 443/tcp or the http and https services through? You can use the “firewall-cmd --list-all” command to verify.
- Can you see the services using nmap against the server from the client machine?
- Is the SELinux context for the directories set correctly?
- Are the permissions on the directory set correctly on the server side? Can the apache user get into directories and read files?
- Are .cgi files set to the SELinux context type of httpd_sys_script_exec_t?
- Can you run .cgi files from command line?

# File Transfer Protocol (FTP)

The File Transfer Protocol (FTP) is one of the oldest ways people used for transferring files between machines over the Internet. To use FTP you need both a client and a server. The most common server on Linux machines is **vsftpd** which stands for Very Secure FTP Daemon.

## *FTP Server Installation*

To get a copy of the FTP client and the Very Secure FTP Daemon server you should install the **ftp** and **vsftpd** packages. The following line can be used to accomplish this.

```
yum install ftp vsftpd
```

## *FTP Server Configuration*

Most of the configuration files for the **vsftpd** server are found in the **/etc/vsftpd/** directory.

The public files available for downloading are stored in the **/var/ftp/pub/** directory.

## *Starting the FTP Server*

The **vsftpd.server** script is used to **start**, **stop**, **restart**, and check the **status** of the service. To start the service you can use the following command.

```
systemctl start vsftpd.service
```

If you want to have the service automatically start at boot time you can **enable** the service. To disable the service you would use the **disable** option. The following command will enable the **vsftpd** service so that it will start automatically at boot time.

```
systemctl enable vsftpd.service
```

## FTP Firewall Rules

To use FTP you will need to make sure the service is available from the outside. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-service=ftp
```

If you wanted to make the rule permanent, you would additionally add the following line.

```
firewall-cmd --zone=public --permanent --add-service=ftp
```

## FTP SELinux Configuration

Public files available via FTP have the **public_content_t** type.

### *FTP Troubleshooting*

The FTP service is not as complex as some services and tends to work pretty good using the default installation, however, if you are having trouble, there are a couple of troubleshooting questions you can ask yourself.

- Are the IP addresses on the client and server both configured correctly?
- Can the client and server both ping each other?
- Did you install the required packages package before trying to start the server?
- Did you start the vsftpd service?
- If errors were reported at start time, did you check the /var/log/messages file?
- If you use “netstat -tuna” is 21/tcp open and listening?
- If you use nmap against localhost do you see ftp (port 21) listed as open?
- Is the firewall allowing 21/tcp or the ftp service through? You can use the “firewall-cmd --list-all” command to verify.
- Can you see the service using nmap against the server from the client machine?
- Is the SELinux context for the directories set correctly?

# Trivial File Transfer Protocol (TFTP)

The Trivial File Transfer Protocol (TFTP) is a less secure and easier protocol, than FTP, for transferring files. The TFTP protocol uses UDP port 69, and is much quicker in setting up communication because it does not require the TCP handshake, and does not require passwords.

## *TFTP Server Installation*

In order to get TFTP running on your system, there are two packages you should consider. The server package is **tftp-server** and the client is called **tftp**. The following line can be used to install both packages.

```
yum install tftp-server tftp
```

## *TFTP Server Configuration*

The TFTP server is installed as part of **xinetd**, so the configuration file is also found in a related directory. You can edit the configuration file at **/etc/xinetd.d/tftp**. The most important change required will be to change the **disable** option from “yes”, to “no”.

For some reason, the server start options found in the **/etc/xinetd.d/tftp** file are ignored part of the time. If you want to be able to upload files to the server instead of just downloading them, you will want to configure the server options. The server also runs as the user nobody, so you might want to change that. The server start up options are found in the **/usr/lib/systemd/system/tftp.service** file.

To run the service as a different user, it is usually a good idea to create a service specific user. To create a **tftpd** user you could use the following command.

```
useradd -c "TFTP User" tftpd
```

It is also a good idea to change the ownership of the TFTP file directory to be owned by the new tftpd user. The following command will accomplish this.

```
chown tftpd:tftpd /var/lib/tftpboot/
```

With the user created, you will probably want to edit the **tftp.service** file. Find the **ExecStart** line and change it to the following.

```
ExecStart=/usr/sbin/in.tftpd -c -p -u tftpd -s /var/lib/tftpboot
```

The additional **-c** option tells the server to allow the creation of new files in the directory. The **-p** option ignores some of the permissions. The **-u** option allows you to assign the user that will be used when the TFTP server is running.

The files served using the TFTP server are stored in the **/var/lib/tftpboot/** directory. This directory and the files in it should have the SELinux context type of **tftpdir_rw_t**. This type allows the TFTP server to both provide files and allow uploads of files.

### *Starting the TFTP Server*

The **tftp.service** script is used to **start**, **stop**, **restart**, and check the **status** of the service. To start the service you can use the following command.

```
systemctl start tftp.service
```

If you want to have the service automatically start at boot time you can **enable** the service. To disable the service you would use the **disable** option. The following command will enable the **tftp** service so that it will start automatically at boot time.

```
systemctl enable tftp.service
```

## TFTP Firewall Rules

To use TFTP you will need to make sure the service is available from the outside. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-service=tftp
```

If you wanted to make the rule permanent, you would additionally add the following line.

```
firewall-cmd --zone=public --permanent --add-service=tftp
```

## TFTP SELinux Configuration

Public files available via FTP have the **tftpdir_rw_t** type.

### *TFTP Troubleshooting*

The TFTP service is not as complex as some services and tends to work pretty good using the default installation, however, if you are having trouble, there are a couple of troubleshooting questions you can ask yourself.

- Are the IP addresses on the client and server both configured correctly?
- Can the client and server both ping each other?
- Did you install the required packages package before trying to start the server?

- Did you start the tftp service?
- If you use “netstat -tuna” is 69/udp open and listening?
- Is the firewall allowing 69/udp or the tftp service through? You can use the “firewall-cmd --list-all” command to verify.
- Is the SELinux context for the files and directories set correctly?

# MariaDB Database

The MySQL database has been a very important part of the Open Source community for a long time. MySQL has, however, had a rough history. Like many commercial companies, the founder sold rights to the software to get more money. Eventually, MySQL fell into the hands of Oracle. As part of Oracle, MySQL has started to feel more like a closed product. The MariaDB database was written as a drop in replacement for the MySQL database by the original author of MySQL. Because of this history, many of the commands will feel the same and share the same names.

## *MariaDB Installation*

To use MariaDB you need to have the **mariadb** package installed (**MariaDB-client** for updated systems). This package provides a client, but does not provide the server. To get the server you need to install the **mariadb-server** package (**MariaDB-server** for updated systems).

After getting these two packages installed, if you want to install additional features you will want to remember that many add-on libraries were written originally for MySQL, so they will probably have MySQL in their name instead of MariaDB.

To install the MariaDB client and server use the following line.

```
yum install mariadb mariadb-server
```

## *Starting the MariaDB Server*

Many commands used for basic configuration of the MariaDB service require that the server be running before they are executed. The MariaDB server is started using the **systemctl** command on the **mariadb.service** script. To start the MariaDB server you can use the following line.

```
systemctl start mariadb.service
```

In addition to starting the server, you can also **stop**, **restart**, and get the **status** using the **systemctl** command.

## *MariaDB Configuration*

After you have started your MariaDB server, you can use the following commands to set the root password using **mysqladmin**. (Note: This is not the root password for your Linux system, just the password used when connecting to the database when logged in as the root user.)

```
/usr/bin/mysqladmin -u root password 'PASSWORD'
/usr/bin/mysqladmin -u root -h HOSTNAME password 'PASSWORD'
```

Remember to replace the **HOSTNAME** to your own hostname and **PASSWORD** with a password that

you can remember. If you want to later change the password, you can use the same above commands with the **-p** option. This will prompt you for the current password before you change the password.

## Configuration File

If you want to make any configuration changes, you can edit the main configuration file. You can look at the **/etc/my.cnf.d/server.cnf** file and see if anything needs to be changed. Normally, the file can remain unchanged.

## MariaDB Database Files

The actual database files for MariaDB are stored in the same place that MySQL stored them. You can find the database files by navigating to the **/var/lib/mysql/** directory. Each database is stored in a directory with a name that matches the name of the database.

If you wanted to backup the database files you could stop the database and backup the files. However, this is not always a good solution since you might miss some of the files. A safer way to copy the databases is to use the **mysqldump** command. To dump a database you could use the following command. Remember to replace **DATABASE** with the actual database name.

```
mysqldump DATABASE > DATABASE.sql
```

You might have to define the user or set the password at the time of dumping the database. To prompt for a password you could use the following command.

```
mysqldump DATABASE -p > DATABASE.sql
```

## *MariaDB Security*

To keep the MariaDB database server safe, it is usually best to only allow local connections. If you are using the MariaDB server for a back-end database for a web server then you can probably just configure the web pages to use the local database if the web server is on the same machine. If you need to connect to a remote machine you will need to configure the firewall to allow remote connections and you will also need to configure SELinux to allow services such as httpd on the client to make network connections.

## MariaDB Firewall Rules

Usually MariaDB can be fully functional without being visible from the outside. If you need to make the service available to outside machines you will need to add a firewall rule. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-service=mysql
```

In addition to adding the service you could instead have added the port number. MariaDB listens on TCP port number 3306. To add a rule to the firewall to based on the port number you could use the following line.

```
firewall-cmd --zone=public --add-port=3306/tcp
```

If you wanted to make the rule permanent, you would additionally add the **--permanent** option as in the following line.

```
firewall-cmd --zone=public --permanent --add-service=mysql
```

## MariaDB SELinux Configuration

The directories and files used by MariaDB use a number of different SELinux contexts. Because MariaDB is a drop in replacement for MySQL the contexts are usually labeled with "mysqld" in the name. Here are a list of files with their context.

| Absolute Directory/File | SELinux Context |
|---|---|
| /var/lib/mysql/ | mysqld_db_t |
| /var/lib/mysql/mysql.sock | mysqld_var_run_t |
| /var/log/mariadb/mariadb.log | mysqld_log_t |

If you want Apache to be able to connect to your MariaDB server using locahost, you do not need to do anything. If you want to allow Apache to connect to a remote MariaDB server you will need to set the SELinux boolean value. The following command will allow httpd to connect to a remote MariaDB server.

```
setsebool -P httpd_can_network_connect on
```

## *MariaDB Troubleshooting*

The MariaDB service usually works, but there are times when troubleshooting is necessary. The following are a list of troubleshooting questions you can ask if you are having problems with the MariaDB server.

- Did you install the required packages package before trying to start the server?
- Did you start the mariadb service?
- If errors were reported at start time, did you check the /var/log/messages file?

- Is the SELinux context for the directories set correctly?
- If you use “netstat -tuna” is 3306/tcp open and listening?
- If you use nmap against localhost do you see mysql (port 3306) listed as open?
- Remote Connections: Is the firewall allowing 3306/tcp or the mysql service through? You can use the “firewall-cmd --list-all” command to verify.
- Remote Connections: Can you see the service using nmap against the server from the client machine?
- Remote Connections: Are the IP addresses on the client and server both configured correctly?
- Remote Connections: Can the client and server both ping each other?

# Postfix Mail Server (SMTP)

With advancements in web servers and browsers, people have started to move away from email technology a little bit. Those who still use email regularly are less likely to understand how it works on the back end.

Traditionally, email is generated in a client program. The client programs can be anything that created the initial email document. Not too long ago, people used desktop and server based applications as their mail clients. Alpine, Mozilla Thunderbird, and Microsoft Outlook are examples of mail applications that communicate with mail servers to get and send their messages. Most modern email accounts use a web based email client. Online services such as Gmail, Hotmail, and Yahoo all provide a web based email client.

The email client talks directly to the local email server to send mail out. They also communicate with email servers to bring in the new email messages. It is the mail servers that take care of sending and receiving the actual email messages.

The default mail server on CentOS is now Postfix. This is not the only mail server, but it is a lot easier to configure than Sendmail, the previous mail server. Postfix also has a better security track record than Sendmail.

## *Running Postfix*

You can start the Postfix services using the following command:

```
systemctl start postfix.service
```

Since on many CentOS 7 systems the service will already be running so this might not be required. If you need to make sure the service is started automatically at boot time you can enable the service.

```
systemctl enable postfix.service
```

If you need to stop the service, you can use the following command.

```
systemctl stop postfix.service
```

## *Listening on all Interfaces*

Because Linux machines need a way to communicate with the administrator, they have the mail server listening on localhost by default. If you were to take a CentOS 7 machine and run the following **netstat** command, you would find that Postfix is listening on **127.0.0.1:25** and **::1:25**, both local loopback addresses.

```
[root@mailhost ~]# netstat -tanp | grep master
```

```
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      1099/master
tcp6       0      0 ::1:25                  :::*                    LISTEN      1099/master
```

To change this, you need to edit the main Postfix configuration file found at **/etc/postfix/main.cf** to listen on all interfaces. Find the section with the **inet_interfaces** directive and comment out the "localhost" option and uncomment the "all" option. The resulting four lines should look like the following.

```
inet_interfaces = all
#inet_interfaces = $myhostname
#inet_interfaces = $myhostname, localhost
#inet_interfaces = localhost
```

After saving the file, you should be able to restart postfix.service and have a mail server that is listening on all interfaces. You can restart the service using the following command.

```
systemctl restart postfix
```

After the restart, make sure your server is listening on all interfaces. You should see that the server is listening on 0.0.0.0:25 and :::25. These addresses indicate that you are listening on all interface addresses for both IPv4 and IPv6.

```
[root@mailhost ~]# netstat -tanp | grep master
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      2046/master
tcp6       0      0 :::25                   :::*                    LISTEN      2046/master
```

## *Accepting Mail*

The Postfix server should accept mail normally for anything with a To: address that goes to the hostname. If your hostname is **mailhost.example.com** then you would be able to accept mail for **Alice** as long as she used the email address **alice@mailhost.example.com**. If you wanted to receive email instead for the domain name, you can configure that.

To allow Alice to receive email at **example.com** and have her email address be **alice@example.com** you would want to change the internal hostname as understood by Postfix. You can do this with the **myhostname** directive in the **/etc/postfix/main.cf** file as seen in the example below.

```
myhostname = example.com
```

Once again, every time you make a change to the server configuration file, you should either **restart** or **reload** the service. The following command will restart your Postfix service.

```
systemctl restart postfix.service
```

## *SMTP Protocol*

If you want to test the SMTP protocol, you can do so by hand. This will also show you how to create fake or spam messages. The easiest way to test the protocol is using a command line tool such as **telnet** that allows you to connect directly to the server port. You can install telnet using the following command.

```
yum install telnet
```

Once you have the telnet client you want to figure out which server you want to connect to. Assume you want to test sending an email message to **alice@example.com**. The example.com domain might receive mail on their example.com server or they might have a mail exchange (MX) record that indicates a different server to connect to. To check for an MX record we could use either of the following commands.

```
nslookup -type=MX example.com
```

```
dig -t MX example.com
```

If there are MX records, they will have priority numbers. The number will be right before the hostname of the server identified in the MX record. If there are MX records then you want to connect to the server with the lowest priority number. If there are no MX records then you connect directly to the domain name of the email address. If example.com does not have an MX record then you would use the following line to connect to the mail server.

```
telnet example.com 25
```

Once you are connected you need to walk through the SMTP protocol dialog. The commands are in order HELO, MAIL FROM, RCPT TO, DATA, and QUIT. The following dialog will show you how to send alice@example.com an email message from santa@northpole.gov

```
bash$ telnet example.com 25
Trying 10.10.10.10...
Connected to example.com.
Escape character is '^]'.
220 example.com Ready
HELO spammer.com
250 example.com Ok
MAIL FROM: <santa@northpole.gov>
250 Ok
RCPT TO: <alice@example.com>
250 Ok
DATA
354 Enter mail, end with "." on a line by itself
```

```
To: "Alice" <alice@example.com>
From: "Santa Claus" <santa@northpole.gov>
Subject: Naughty or Nice List

Dear Alice,
After reviewing the naughty and nice list it has been determined that you do not qualify
for good presents.
I am shipping you a box of coal to warm your heart.
Sincerely,

Santa
.
250 Ok
QUIT
221 example.com Closing transmission channel
Connection closed by foreign host.
```

The DATA section contains a header, a blank line, then the body of the message. The message knows to end when you put a single dot on a line by itself. At that point you just type QUIT to end the message.

If at any point you receive errors, you can look at those errors and try to figure out what happened and how to fix it.

## *Mail Server Firewall Rules*

Postfix runs by default on many Linux machines and handles internal messaging. Because it normally handles internal things, it does not need outside access. If you need to make the service available to outside machines you will need to add a firewall rule. The following firewall rule command will make this possible.

```
firewall-cmd --zone=public --add-service=smtp
```

If you wanted to make the rule permanent, you would additionally add the following line.

```
firewall-cmd --zone=public --permanent --add-service=smtp
```

## *Postfix Troubleshooting*

The **postfix** service is configured to work on default systems, it is usually only when you decide to listen to outside interfaces that you have problems. The following are a list of troubleshooting questions you can ask if you are having problems with the postfix server.

- Does you mail server have an IP address and hostname that match what is stored in the DNS server?
- Is the mail server running? (netstat)
- Are you listening on all interfaces? (netstat, nmap)
- Is the firewall blocking incoming connections? (firewall-cmd --list-all)

- Have you changed the SELinux context on any files?  Are they set correctly?  (ls -Z, ls -dZ)
- Can client machines see the server?  (ping, nmap)
- Is the SMTP open on the server?  (nmap)
- Do the logs indicate there are problems? (/var/log/maillog, /var/log/messages)
- Is there anything in the mail directory that indicates a problem?  (/var/spool/mail)

# Dovecot Mail Services

The **dovecot** service provides many different interfaces for requesting mail from the server while on the client machines. These services include the IPv4 and IPv6 versions of **pop3**, **imap**, **imaps**, and **pop3s**. All of the services I just listed are configured to run on the default machine.

## *Running Dovecot*

Before you can run Dovecot, you need to have it installed. If you do not already have Dovecot installed, you can install it using the following command.

```
yum install dovecot
```

Once you have Dovecot installed, you can **start** the Dovecot services using the following command:

```
systemctl start dovecot.service
```

If you need to make sure the service is started automatically at boot time you can **enable** the service.

```
systemctl enable dovecot.service
```

If you need to **stop** the service, you can use the following command.

```
systemctl stop dovecot.service
```

You can get a listing of all of the Dovecot services running using the **netstat** command. The following command will show you all the Dovecot services and ports.

```
[root@cs240u-2-00 postfix]# netstat -tap | grep dovecot
tcp        0      0 0.0.0.0:pop3            0.0.0.0:*               LISTEN      3053/dovecot
tcp        0      0 0.0.0.0:imap            0.0.0.0:*               LISTEN      3053/dovecot
tcp        0      0 0.0.0.0:imaps           0.0.0.0:*               LISTEN      3053/dovecot
tcp        0      0 0.0.0.0:pop3s           0.0.0.0:*               LISTEN      3053/dovecot
tcp6       0      0 [::]:pop3               [::]:*                  LISTEN      3053/dovecot
tcp6       0      0 [::]:imap               [::]:*                  LISTEN      3053/dovecot
tcp6       0      0 [::]:imaps              [::]:*                  LISTEN      3053/dovecot
tcp6       0      0 [::]:pop3s              [::]:*                  LISTEN      3053/dovecot
```

## *Dovecot Firewall Rules*

Since Dovecot needs to provide services to client machines, it needs to have openings in the firewall. The services you decide to run within Dovecot will help you determine which firewall rules are needed.

The following firewall services can be added to open up all of you service ports.

```
firewall-cmd --zone=public --add-service=pop3
firewall-cmd --zone=public --add-service=pop3s
firewall-cmd --zone=public --add-service=imap
firewall-cmd --zone=public --add-service=imaps
```

If you wanted to make the rules permanent, you would additionally add the following lines.

```
firewall-cmd --zone=public --permanent --add-service=pop3
firewall-cmd --zone=public --permanent --add-service=pop3s
firewall-cmd --zone=public --permanent --add-service=imap
firewall-cmd --zone=public --permanent --add-service=imaps
```

You only need to add firewall rules for the services that you want to make available to the outside for clients to connect to and download their mail.

# Network Information Service (NIS)

Authentication services are useful in situations where you would prefer to maintain a single set of user accounts and passwords. This is especially useful in lab scenarios where users can sit down at any computer and use their username and password to authenticate. It is also useful when implementing single sign-on types of services.

The Network Information Service makes managing user accounts and passwords from a central server pretty easy. The NIS protocol used to be called YP which is short for Yellow Pages, but someone was not happy with the trademark violation and the name had to be changed. Because the YP service was so established at the time, many of the commands retail their original names.

## *NIS Master Server Setup*

In order to get a NIS server running, you need to make sure you have the software installed. The NIS server requires the **ypserv** package to be installed. Use the following command to install the **ypserv** package.

```
yum install ypserv
```

After you have the server software installed, you need to make sure you set the NIS Domain Name. This can be done with either the **nisdomainname** or the **ypdomainname** commands. You do not need to use both. To set the NIS domain name to **example.com** I would use one of the following commands.

```
nisdomainname example.com
```

```
ypdomainname example.com
```

You can also update the **/etc/sysconfig/network** file and add a line that looks like the following.

```
NISDOMAIN=example.com
```

To verify that the NIS domain name was properly set, you can use the above commands without any arguments. The machine should then respond with the domain name. The following dialog is an example.

```
[root@server ~]# nisdomainname
example.com
```

When you have the ypserv package installed and the domain has been set, you are ready to initialize NIS. The **ypinit** command is the easiest way to get your initial configuration files created and put into

place. The **ypinit** command is usually found in the **/usr/lib/yp/** or **/usr/lib64/yp/** directory depending on your server architecture. Use the commands that is appropriate for your machine.

On 32 bit architecture:

```
/usr/lib/yp/ypinit -m
```

On 64 bit architecture:

```
/usr/lib64/yp/ypinit -m
```

Because NIS can be distributed, you can have multiple servers running the whole NIS system. The server you create first is the master and any additional servers running copies of the NIS database are slaves. You can add additional slaves if you want to, but it is not required. In the **ypinit** dialog you will be prompted for additional slave servers. When done, press the **Ctrl-d** key combination.

## *NIS Slave Server Setup*

If you added additional slave servers, you will need to log into each of them, install the **ypserv** software and tell them to use your master server. If your master NIS server was **server.example.com** you would run one of the following lines on each of the slave servers.

On 32 bit architecture:

```
/usr/lib/yp/ypinit -s server.example.com
```

On 64 bit architecture:

```
/usr/lib64/yp/ypinit -s server.example.com
```

## *Running NIS Servers*

NIS uses the rpcbind service as part of normal operation. While this service is normally installed and running, you might want to make sure. You can start and enable it using the following lines.

```
systemctl start rpcbind
systemctl enable rpcbind
```

Once you have the NIS servers installed and configured, you are ready to start the **ypserv** service. The following command will start your NIS server.

```
systemctl start ypserv
```

If you want the server to start automatically every time you boot your machine you would want to enable the service. Use the following command to enable the **ypserv** service.

```
systemctl enable ypserv
```

If you make any configuration changes or change passwords directly on the master server, you will need to make sure the master server has the new configuration files loaded. This can be a little bit complex since the files loaded into the server are first compiled. To do this, move to the configuration file directory and rebuild the files, then restart the server. The following commands should take care of this for you.

```
cd /var/yp/
make
systemctl restart ypserv
```

If you ever need to restart the **ypserv** service you can just run the restart command.

```
systemctl restart ypserv
```

If you need to stop the service or disable it, you should first consider the client machines. Will they still be able to authenticate or will you cause problems? The following commands will stop the service and disable it from starting automatically at boot time.

```
systemctl stop ypserv
systemctl disable ypserv
```

## *Firewall*

You probably want to configure your firewall so that the client machines that can connect to your NIS server. Enabling the firewall on the server is a bit difficult. First, the port does not seem to want to stay in the same place using the default configuration. Second, the NIS server requires that the **rpc-bind** service also be able to communicate through the firewall.

First, we will change the default port to be static ports. To do this, we need to edit the **/etc/sysconfig/network** file and add the following line.

```
YPSERV_ARGS="-p 834"
```

If you want to then add the ports to the firewall you can use the following lines. I also included the tcp and udp 111 ports for the **rpc-bind** service:

```
firewall-cmd --add-port=tcp/834
firewall-cmd --add-port=udp/834
firewall-cmd --add-port=tcp/111
firewall-cmd --add-port=udp/111
```

After adding these rules, do it all over, but with the **--permanent** option to make the rules stick.

If you instead prefer to add services, you will need to first create the **ypserv** service in the firewall since it does not exist in the current list of **firewalld** services. Using an editor, create a file called **/usr/lib/firewalld/services/ypserv.xml** and put the following contents into it.

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>ypserv</short>
  <description>NIS Server</description>
  <port protocol="tcp" port="834"/>
  <port protocol="udp" port="834"/>
</service>
```

When you have the file created, you will need to restart the **firewalld** service to make sure it is aware of the new service definition.

```
systemctl restart firewalld
```

After the firewall has been restarted, you are ready to add the firewall service. To enable the new **ypserv** firewall service you can use the following lines to add your new service and the **rpc-bind** service.

```
firewall-cmd --add-service=ypserv
firewall-cmd --add-service=rpc-bind
```

To make your rules permanent remember to also add the **--permanent** option.

## *NIS Client Machine Setup*

Before you configure a NIS client machine it is usually a good idea to setup your server and make sure it is running. The easiest way to configure the client machine is during the installation process. You can tell the server to use NIS for authentication and configure before you even boot the first time.

If you already have a running client machine and want to switch it over to using NIS you will first need to install the NIS client software. The **yp-tools** and **ypbind** packages are the most useful. You can install them using the following command.

```
yum install yp-tools ypbind
```

Once you have the NIS client software installed you should go ahead and set your NIS domain name. If you were using the **example.com** domain name you could use either of the following commands.

```
nisdomainname example.com
```

```
ypdomainname example.com
```

Now you can configure your machine to use the NIS service using the **authconfig-tui** command.

```
authconfig-tui
```

Select the "**Use NIS**" option and [**tab**] over to the Next button. Set your domain to the NIS domain you configured earlier. Set the server to the machine that is running the NIS server.

When the configuration has completed, you might notice the following line appearing in your **/etc/yp.conf** file.

```
domain example.com server server.example.com
```

After configuring your machine to use the NIS server for authentication, it is probably a good idea to make sure your **ypbind** service is running and enabled. Use the following lines to make sure it is up and ready to authenticate users.

```
systemctl start ypbind
systemctl enable ypbind
```

NIS uses **rpcbind** to find the server and connect correctly, so if you are having problems with your client, you should make sure that the **rpcbind** service is up and running as well. You can turn it on and enable it using the following lines.

```
systemctl start rpcbind
systemctl enable rpcbind
```

If you want to run some of the troubleshooting tools, it is usually a good idea to install the **ypbind** package and configure it on the server as well as clients.

# Domain Name Service (DNS)

The Domain Name System/Service is a large scale distributed database that contains name and address mappings. When networks first used names to refer to IP addresses, they used the hosts file. This file is still in use and is available at /etc/hosts, however, as more and more computers received IP addresses and were given names, this file grew to an unmanageable size. DNS was an answer to the distribution problem of the data stored in that single file.

## *Installing BIND DNS Services*

The most commonly used DNS server on Linux machines is BIND. You can get a copy of BIND using the following command.

```
yum install bind
```

## *Starting the BIND DNS Service*

The BIND DNS service is called **named** and is controllec with systemctl using the **named.service**. To start the BIND server you can use the following line.

```
systemctl start named.service
```

In addition to starting the server, you can also **stop**, **restart**, and get the **status** using the **systemctl** command.

## *Configurations*

The BIND service has a zone file for each domain it is providing information and also a main configuration file that controls how zones are loaded and shared. The first file to look at is the main configuration file.

## File: /etc/named.conf

The main BIND configuration is found in the **/etc/named.conf** file. This file is broken up into multiple sections which include an options section and zone sections. In addition to the sections directly in the file it can also include other files using the include command.

The include directive takes a single argument of a file to be included. Normally the following default **include** files are in your named.conf file.

```
include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
```

### Section: options

In the options section there are a lot of variables that are set that control how the system works. Initially you want to configure how your DNS server is going to be listening. The default configuration

is to only listen on the local loopback addresses. You can change these by changing the IP address to a different address or using the keyword "**any**" instead. The following are the listening lines when listening on any interface.

```
listen-on port 53 { any; };
listen-on-v6 port 53 { any; };
```

After you have decided where your server is going to listen, it is important to indicate where your zone files are going to be stored. This is set with the **directory** directive. Normally this is set to the **/var/named/** directory.

```
directory "/var/named";
```

The directory directive also gets prepended to all later zone file locations. Keep this in mind when creating zone files.

In addition to the directory directive you also need to configure a few other files that will be used for caching, and statistics.

```
dump-file          "/var/named/data/cache_dump.db";
statistics-file    "/var/named/data/named_stats.txt";
memstatistics-file "/var/named/data/named_mem_stats.txt";
```

After configuring these files there are a few other directives that are important. First, you are listening on the interfaces you have chosen, but who are you allowing queries from? If you want to serve DNS to everyone then it is probably best to allow queries from anyone.

```
allow-query { any; };
```

If you are accepting queries from anyone, you need to decide which types of queries you are going to process. The next few directives deal with your relationships with other DNS servers. Are you going to allow clients to query your DNS server for information that you do not contain? Basically, if you do not have the information in your zone files, are you going to go out and ask other servers or are you just going to tell the client to figure it out on their own.

The act of resolving the queries for your clients is called recursion. If you want to allow **recursion** then you would set the directive value to "**yes**".

```
recursion yes;
```

If you do not know about the queried information the default location you would search for the information is at the root servers. From there you would work your way down to the desired servers.

If you have closer servers you want to query instead you can set them in the **forwarders** directive.

```
forwarders { 8.8.8.8; };
```

If you are acting as a primary or master DNS server then there would be servers that would need to request zone transfers from your server. Allowing zone transfers can sometimes be dangerous if you do not trust the client getting the information. The danger of revealing your whole zone is that social engineers and hackers can use the information to figure more information about your infrastructure and potentially vulnerable servers.

If you have a secondary or slave server also providing DNS services you would configure your server to **allow-transfer** from that client DNS server. Assuming the secondary DNS server was 10.10.10.9 you would use the following directive.

```
allow-transfer { 10.10.10.9; };
```

If you put it all together, you would have an **options** section that looks like the following.

```
options {
    listen-on port 53 { any; };
    listen-on-v6 port 53 { any; };
    directory    "/var/named";
    dump-file    "/var/named/data/cache_dump.db";
    statistics-file     "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    allow-query { any; };
    recursion yes;

    forwarders { 8.8.8.8; };
    allow-transfer { 10.10.10.9; };
};
```

**Section: zones**

The zone definitions tell BIND which zone is being served, the relationship the server has with that zone, the file the zone information is stored in, and also in some cases who has the information if a zone transfer is required.

**Forward Primary Zone**

The following is what a forward zone definition would look like if the server is acting as the primary.

```
zone "example.com" {
    type master;
    file "data/example.com.zone";
};
```

Because the directory directive in the options section has the value of **/var/named** the actual location of the zone would be in the **/var/named/data/example.com.zone** file.

### Forward Secondary Zone

If the DNS server is acting as a secondary and the primary is at 10.10.10.10 then you might have a definition like the following.

```
zone "example.com" {
    type slave;
    file "slaves/example.com.zone";
    masters { 10.10.10.10; };
};
```

The zone information would be stored in the **/var/named/slaves/example.com.zone** file.

### Reverse Primary Zone

The reverse zones provide IP to hostname lookups. The following is how you would create a zone file for the 10.10.0.0/16 zone.

```
zone "10.10.in-addr.arpa" {
    type master;
    file "data/10.10.zone";
};
```

The zone information would be stored in the **/var/named/data/10.10.zone** file.

### Reverse Secondary Zone

If the DNS server is acting as a secondary and the primary is at 10.10.10.10 then you might have a definition like the following.

```
zone "10.10.in-addr.arpa" {
    type slave;
    file "slaves/10.10.zone";
    masters { 10.10.10.10; };
};
```

The zone information would be stored in the **/var/named/slaves/10.10.zone** file.

## *Forward Zones*

Forward zones contain information mapping where you start with a name and go to either an IP address or some other information. Forward zones contain most of the DNS information with record types

such as A, AAAA, CNAME, MX, TXT, and NS.

The following is a simple example forward zone file for the example.com domain.

```
$TTL 3h;
@   IN     NS     server.example.com.
@   IN     SOA    example.com. admin.example.com. (
                  2016030101; serial
                  10800; refresh
                  3600; retry
                  604800; expire
                  86400; default ttl
)
server            IN     A      10.10.10.10
```

In the forward zone file there are a couple of things to note. The default time to live for records is set using the $TTL objective to a value of 3 hours. We are saying that the DNS server that serves this example.com domain is server.example.com.

Then we get into the SOA record. The SOA record indicates that we are working with the example.com domain and that the email address of the administer is admin@example.com (written with a dot in place of the at sign).

The SOA record also has a serial number of the database and a bunch of times. The serial number is used when zone transfers happen. If the serial number of the database is higher than the serial number in the client DNS server acting as a secondary/slave, then the database is downloaded, if not, it is assumed that the database has not changed and nothing is downloaded.

The rest of the times refer to how often your secondary server is supposed to try to update record information. Probably the most important number would be the refresh value which is the number of seconds before the secondary is supposed to try to download a new copy of the records.

## A Records

The A records are used to translate a name into an IP address. The A records are what the DNS servers are most known for and also the primary reason you want a DNS server. The following are examples of A records.

```
server     IN     A      10.10.10.10
server2    IN     A      10.10.10.9
```

Because we are in the example.com domain, each entry here is a host for a machine in the example.com domain. The server line indicates that server.example.com is mapped to the IP address 10.10.10.10.

## AAAA Records

With advancements in networking and the oncoming change to IPv6, it was decided that there needed to be a record type that could contain the IPv6 addresses. The AAAA records serve the same purpose as the A records, except for IPv6 instead of IPv4. The following are examples of AAAA records.

```
server     IN     AAAA   ::1
server2    IN     AAAA   ::1
```

## CNAME Records

CNAME records are used to create aliases for your hosts. When you create an alias you need to set it a name instead of an IP address. If you have a trailing dot then the name you assign the alias to is assumed to be an absolute name, if not, then it is assumed to be relative to the domain you are working in. The following are examples that both point to the same server.

```
www        IN     CNAME  server.example.com.
mail       IN     CNAME  server
```

## MX Records

MX or Mail Exchange records indicate which mail server you are supposed to use for the domain. Normally, they are assigned to the domain you are working in, so the record line starts with the at sign. If you are uncomfortable with the at sign, you can type out the whole domain instead. When writing the whole domain, do not forget the trailing dot.

Each MX record has a priority number as well. If you have multiple mail servers that are allowed to receive mail and have a preference as to which one get the mail then you would set the priority number for that server to be the lowest. If you do not care which one receives mail or if you want to load balance then you would set the priority numbers to the same value. In the example below, we will set our priority number to **10**.

The last field is the name of the server that is to receive the mail. In the examples below the server is **mail.example.com**, so I can either write mail because I am in the **example.com** domain or I can type out the whole name with a trailing dot. The following three lines are equivalent.

```
@   IN     MX     10     mail
```

```
example.com.      IN     MX     10     mail
```

```
example.com.      IN     MX     10     mail.example.com.
```

## TXT Records

The TXT records are now commonly used to indicate which machines are allowed to send mail for the domain. This is done by creating an SPF entry in the TXT record. The following is an example record that would let other mail servers know that anyone in the 10.10.10.0/24 subnet is allowed to send mail from the example.com domain.

```
@   IN     TXT    "v=spf1 ip4:10.10.10.0/24 ~all"
```

## NS Records

NS records indicate which machines server as name servers. You can also use them to designate sub-domain DNS servers as well.

```
@   IN     NS     server.example.com.
```

## Setting the Origin

When you create a forward or reverse zone, the SOA record is used to indicate what the zone is. Sometimes you create very broad zones such as example.com, but want to create some records for a sub-domain such as sub.example.com inside the main domain zone instead of creating a separate sub-domain zone file. To do this, you have two options. You can list the records with their sub-domain or use an **$ORIGIN** directive. Without the $ORIGIN directive your entries would look similar to the following.

```
host1.sub  IN     A      192.168.1.1
host2.sub  IN     A      192.168.1.2
```

With the $ORIGIN directive you can change the scope and skip listing the sub-domain name for each entry.

```
$ORIGIN sub.example.com.
host1      IN     A      192.168.1.1
host2      IN     A      192.168.1.2
```

## Generating Records

Using the **$GENERATE** you can create a set of similar records. Imagine for a moment that you wanted to generate A records for a group of IP addresses assigned to your DHCP server pool. If the addresses in the pool were 192.168.0.50-100 and the format of the A record was dhcp-X.example.com, then you could use a $GENERATE statement like the following.

```
$GENERATE 50-100 dhcp-${0,2} IN A 192.168.0.$
```

### *Reverse Zones*

Reverse zones provide a look up mechanism to figure out which hostnames are assigned to which addresses. While it would be ideal if you could always determine the hostname at an IP address, it is not always available or even possible. Each IP address, configured so, is usually assigned a single hostname. These are in the reverse zone files.

The following is an example reverse zone file for the 10.10.10.0/8 subnet.

```
$TTL 3h;
@       IN      NS      server.example.com.
@       IN      SOA     example.com. admin.example.com. (
                2016030101; serial
                10800; refresh
                3600; retry
                604800; expire
                86400; default ttl
)

; Some Comment
$ORIGIN 10.10.10.in-addr.arpa.
10  IN     PTR    server.example.com.
```

## PTR Records

PTR or pointer records are used to move from an IP address to a hostname. In the above reverse zone example, there is a single PTR record within the 10.10.10.0/24 address range as defined by the $ORIGIN command with the 10.10.10.in-addr.arpa. option.

Each PTR line should complete the full IP address and tell the hostname it resolves to. We are only missing the last octet, so we list it. The following is what we would list for the 10.10.10.10 address that resolves to server.example.com.

```
10  IN     PTR    server.example.com.
```

### *Security*

DNS servers are commonly a target of Internet attacks. If you can take down a DNS server or replace entries then you can cause a lot of damage to a domain or group of domains. Because of the information that DNS servers provide, they really need to be up and available all of the time. Most Internet services and communication relies on DNS being up, working, and accurate.

## Firewall

The DNS server uses UDP port 53 for normal lookups and TCP port 53 for zone transfers. To add the services you can use the following command:

```
firewall-cmd --zone=public --add-service=dns
```

If you wanted to make those rules permanent you would add the **--permanent** option.

```
firewall-cmd --zone=public --add-service=dns --permanent
```

If you wanted to do the same thing by ports you could use the following lines:

```
firewall-cmd --zone=public --add-port=53/tcp
firewall-cmd --zone=public --add-port=53/udp
```

And for permanent rules the following lines:

```
firewall-cmd --zone=public --add-port=53/tcp --permanent
firewall-cmd --zone=public --add-port=53/udp --permanent
```

## SELinux

There are multiple SELinux contexts for files used by named. The configuration files usually have the **named_conf_t** SELinux type. The zone files will usually have the **named_conf_t, named_cache_t**, or **named_zone_t** types. If you find that you have the incorrect context type for a file, you can use the **restorecon** command to reset the context type for the file. The following are examples.

```
restorecon /etc/named.conf
restorecon /var/named/
```

### *Client Programs*

To verify DNS configurations, you can use the nslookup and dig commands. The easiest of these commands is the nslookup command. The following are examples of queries you can make using the nslookup command.

```
nslookup example.com
nslookup 192.168.1.1
```

The dig command is a little bit more complex than the nslookup command. It also provides a lot more information. Here are the above two queries using dig.

```
dig example.com
dig -x 192.168.1.1
```

## *Troubleshooting*

DNS configurations can be pretty clean and simple, but sometimes you have problems and it is not always obvious what is not working correctly. The following are a list of steps you can take in troubleshooting your DNS servers.

- Make sure DNS server is set. (/etc/resolv.conf)
- Make sure you can talk to DNS (ping, nslookup, dig)
- Make sure records display correctly (nslookup, dig)
- Try performing a manual zone transfer (type: AXFR)
- Trace DNS hierarchy (dig +trace)
- Make sure the firewall is correct (firewall-cmd --list-all)
- Check logs (/var/log/messages)
- Verify service is running (netstat -tunap | grep named)
- Verify SELinux context (ls -Z)

# Dynamic Host Configuration Protocol (DHCP)

The DHCP protocol makes it much easier to manage a network full of client machines. Since every machine communicating on a network needs an IP address it ideal to have an easy way to assign these numbers. DHCP makes it possible to assign a range of addresses and have the server automatically manage assigning addresses and keeping track of leases and renewals.

## *Installing DHCPv4*

To use a DHCP server you first need to install it. The DHCPv4 service uses a package called **dhcp**. This package provides the server.

To install the DHCP server, use the following line.

```
yum install dhcp
```

## *Starting the dhcpd Service*

The DHCPv4 server is started using the **systemctl** command on the **dhcpd.service** script. To start the DHCP server you can use the following line.

```
systemctl start dhcpd.service
```

In addition to starting the server, you can also **stop**, **restart**, and get the **status** using the **systemctl** command.

## *Configurations*

The configuration file for the DHCP server is found in the **/etc/dhcp/** directory. There are two main files **dhcpd.conf** and **dhcpd6.conf**. The following is an example of the dhcpd.conf file.

```
option domain-name "example.com";
option domain-name-servers 8.8.8.8;
default-lease-time 300;
max-lease-time 1200;

subnet 10.10.0.0 netmask 255.255.0.0 {
  range 10.10.0.100 10.10.0.150;
  option routers 10.10.0.1;
}
```

This file is made up of some global configurations and then some subnet specific configurations. Normally, you can put the global configuration directives inside of a subnet if you want to override them for that specific subnet.

### Static DHCP Addresses

Within a subnet definition you can assign a static address using the following command.

```
host printer { hardware ethernet 00:11:22:33:44:55; fixed-address 10.10.0.15; }
```

### Additional Subnets

If you want your DHCP server to serve more than one subnet, you need to make sure it has a subnet definition for the subnet it is in (even if not assigning addresses for the subnet). Then add additional subnet definitions for each subnet you want to assign addresses for.

The following configuration example would take the previous definition and add additional information for the 192.168.0.0/24 subnet.

```
option domain-name "example.com";
option domain-name-servers 8.8.8.8;
default-lease-time 300;
max-lease-time 1200;

subnet 10.10.0.0 netmask 255.255.0.0 {
  range 10.10.0.100 10.10.0.150;
  option routers 10.10.0.1;
}

subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.50 192.168.0.150;
  option routers 192.168.0.1;
}
```

## Routers

Since DHCP only responds to requests it can hear, it is important to make sure the requests get to the required service. DHCP requests only go as far as the local area network, so routers need to be configured to forward requests if you want to allow a DHCP server to serve a network it is not part of. With Cisco routers you can configure the ip helper address.

The following dialog on a Cisco router will show you how this is done assuming the DHCP server is at 10.10.10.10.

```
Router(config)# ip helper-address 10.10.10.10
```

## *Clients and Leases*

When machines are assigned addresses, they get an entry in the leases file. This file is usually contained in the **/var/lib/dhcpd/** directory and is called **dhcpd.leases**. The following is an example of a simple lease.

```
lease 192.168.0.221 {
        starts 2 2016/03/26 02:23:59;
        ends 3 2016/03/27 02:23:59;
        hardware ethernet 33:44:55:66:77:88;
        uid 01:23:45:67:89:ab:cd;
        client-hostname "alice";
}
```

### *Security*

There are a few security concerns with DHCP servers. The biggest concern is for rogue DHCP servers giving out addresses when they are not supposed to. If you configure a DHCP server, it is best to make sure it is only giving out addresses when it is supposed to. You also want to make sure information in the configuration is correct because once you give out information it will be used until the client comes back to renew the lease.

In addition to rogue DHCP servers, there is a risk when you assume a MAC address can positively identify a machine. You cannot always be sure that the machine requesting the IP address is who they say they are. Try not to give too much access based on MAC addresses and assigned IP addresses.

## Firewall

The DHCP server uses UDP port 67 for communication. To allow communication to the service you can either add the dhcp service or add the port. The client uses the dhcpv6-client service in firewalld and is already activated by default. To add both service rules you could use the following lines.

```
firewall-cmd --zone=public --add-service=dhcpv6-client
firewall-cmd --zone=public --add-service=dhcp
```

If you wanted to make those rules permanent you would add the **--permanent** option.

```
firewall-cmd --zone=public --add-service=dhcpv6-client --permanent
firewall-cmd --zone=public --add-service=dhcp --permanent
```

## SELinux

SELinux should be configured correctly, so you do not need to do much. If you want to verify some files, the following are some file/directory names and contexts. Note: Some files might not exist yet. You can check the context of a file using “ls -Z” and the context of a directory using “ls -dZ”.

| File/Directory | SELinux Context |
|---|---|
| /etc/dhcp/ | dhcp_etc_t |
| /etc/dhcp/dhcpd.conf | dhcp_etc_t |

| | |
|---|---|
| /var/lib/dhcpd/dhcpd.leases | dhcpd_state_t |
| /var/log/messages | var_log_t |

## *Troubleshooting*

The DHCP service usually works when you are serving just your own LAN, but there are times when troubleshooting is necessary. The following are a list of troubleshooting questions you can ask if you are having problems with the DHCP server.

- Does the server have a static IP address?
- Is the DHCP service running?
- Do the client machines have a driver that works with the NIC?
- Do the client machines have network cables that connect all the way back to the DHCP server through the network infrastructure?
- Is the firewall allowing connections? (firewall-cmd --list-all)
- Are client machines getting IP addresses from the DHCP server? (ip addr, ifconfig, ipconfig)
- Are you seeing the server status and/or dhcp request dialog in /var/log/messages file?
- Are lease records being created? (cat /var/lib/dhcpd/dhcpd.leases)
- If you are serving a subnet outside your LAN, do you have a subnet definition for the subnet?
- Are routers properly forwarding DHCP requests for other subnets?
- Are the switches allowing your machine to send DHCP OFFER packets?
- Are ports on the switch configured to assume a computer instead of a switch? Is spanning-tree preventing address assignments?

# Source Code Repository: Subversion

When programmers are creating source code it is common to run into situations where the solution to a problem is not obvious. Sometimes the changes required will take a lot of work and will modify multiple files. In order to make this operation safer, it is usually best to save a copy of working code before starting your changes. Sometimes you want to just look back and see what a previous version of source code looked like. For these situations it is useful to have a version control software package.

Subversion is a software package that makes it easier to keep track of changes to code and make working together easier.

## *Subversion Server Configuration*

With the high number of open source projects available on the Internet there are a number of examples of how to check out a subversion project. Now, it is time to look at the subversion server configuration and setup.

In order to use subversion you need to install the **subversion** package. Most source repository sites use http or https based URLs, so to use those you would also want the **httpd** and **mod_dav_svn** packages. The following line should get you what you need.

```
yum -y install subversion httpd mod_dav_svn
```

Once you have the software installed, it is time to figure out where you want to store your repositories. I like the **/var/svn/** directory, so my examples will assume you are using that directory. You will first need to create the directory.

```
mkdir /var/svn/
```

In that directory you can create a repository using the **svnadmin create** command. In order to create a repository for the hello world source code you could first move into the **/var/svn/** directory, then use the following command.

```
svnadmin create hello
```

## File Level Permissions

I would, at this point, recommend setting everything up so that you can use Apache. To do so, you will need to make Apache the owner and set the SELinux context on the files. First, set the owner to Apache.

```
chown -R apache:apache /var/svn/
```

Now, for the SELinux context type. Normally, in order for Apache to see files you want to set the **httpd_sys_content_t** type, but since you want to be able to both read and write the files you want to instead use the **httpd_sys_rw_content_t** type. You want to change the context type of the files and also make sure new files in the SVN repos have the correct context. Use the **semanage** command to set the default type, then change the existing files.

```
semanage fcontext -a -t httpd_sys_rw_content_t '/var/svn(/.*)'
```

In order to change the existing files you can use either of the following commands.

```
chcon -R -t httpd_sys_rw_content_t /var/svn
```

OR

```
restorecon -R /var/svn
```

At this point you have a repository that could be used for clients using Secure Shell, but your Apache server is still not configured to allow SVN checkouts.

## Configuring Apache for SVN

When you installed the **mod_dav_svn** package you should have automatically gotten the **/etc/httpd/conf.modules.d/10-subversion.conf** file that tells Apache to load the required modules. Now, we need to tell Apache where the repositories are stored.

In order to make things as clean as possible, create a subversion specific file called **/etc/httpd/conf.d/subversion.conf** with the following contents.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthName "Subversion repositories"
  AuthType Basic
  AuthBasicProvider file
  AuthUserFile /var/svn/.htpasswd
  Require valid-user
</Location>
```

This file will tell Apache that you are designating the **/svn** path on the server to the repositories found in the **/var/svn/** directory. It also sets up basic security for Apache so that you need a user name and password in order to checkout content. The file **/var/svn/.htpasswd** will store the user names and passwords.

Before you can actually use the repository, you need to create a user. In order to create a user Alice we will use the **htpasswd** command. The following dialog is an example.

```
bash# htpasswd -c /var/svn/.htpasswd alice
New password:
Re-type new password:
Adding password for user alice
```

We need to make sure the Apache service is up and running. You will likely also want to make sure the Apache service starts at boot time. The following commands should take care of these.

```
systemctl restart httpd.service
systemctl enable httpd.service
```

### *Firewall Configuration*

In order to allow client machines to connect to the server we need to configure the server allow external connections to the port(s) we use. The Apache service uses the **http** filewall service.

```
firewall-cmd --permanent --add-service http
firewall-cmd --reload
```

### *Subversion Clients*

When you checkout a project on a client machine, it is normally a good idea to create a directory you can put all of your projects into. When you are in the directory you want to use you can checkout the repositories.

## Secure Shell Checkout

If you have not configured or cannot configure web based authentication, you still have the option of using SSH to checkout a project. Normally, if you are using SSH it is a good idea to configure key based authentication. You can jump to the SSH chapters or just use the following commands that assume you are using the **root** user at **server.example.com**.

```
ssh-keygen
ssh-copy-id root@server.example.com
```

Now that you can connect to the server easily. To checkout the above **hello** project using the user **root** from **server.example.com** you could use the following command.

```
svn checkout svn+ssh://root@server.example.com/var/svn/hello
```

## HTTP Based Checkout

If you configured the server to provide web based access you can use the URL to check out the project. Since we created the **alice** user and put her information in the **.htpasswd** file we can use a command like the following to checkout a project.

```
svn checkout --username alice http://server.example.com/svn/hello
```

This will prompt you for Alice's password that you created earlier when you were configuring the server. After entering the password you can decide if you want to save the password locally for later use.

## Client Commands

When you have the repository, you can start using the SVN commands to add files, update files, and commit changes. There are multiple other commands, but these are the basic commands to get you started.

After you have created some files in the repository, you can add it to the project using the **svn add** command. The following would add the **hello.py** command to the repository.

```
svn add hello.py
```

After adding the file, the next step is to commit the file to the repository. You have to think about a comment you can include when sending the file. The following **svn commit** example should give you an idea of what you can do.

```
svn commit -m "Created the initial hello.py program"
```

At this point the file will be uploaded to the repository. If you have not saved passwords you will probably be prompted for passwords. Sometimes multiple communication events happen and you are prompted for the password multiple times. Just enter the password as many times as you are asked for it.

If you have multiple machines using the same code you might need to download the changes that were made by someone else or a different machine. To download updates you can use the following **svn update** command.

```
svn update
```

# Source Code Repository: Git

The Git version control system is similar to Subversion in purpose, but offers more. The Subversion version control system was mostly a client/client server where each user checked out a copy from the server and sent updates to the server.

With Git, the same client/server system exists internally and when you are ready, you can push your server information to another server repository or pull updates down to your local machine from the server repository.

When you decide to use Git, you can decide if you want to have a central server or if you want to run a local only copy without the push and pull requests.

## *Local Only Git Repository*

You can create a local Git repository from an existing project. To do so, you first navigate into the base directory of the project and issue the following command to create the repository.

```
git init .
```

This will create a **.git** directory in the current directory and will allow you to then add files commit them to the project. Each file that you want to add to the project needs to be added. If you had created a "Hello World" project and you had a source code file called **hello.c** you could mark it for adding to the project repo using the following **add** command.

```
git add hello.c
```

Once the file has been marked for inclusion in the project, you need to actually add it. Adding files can be done using the commit command. I recommend also using a comment that indicates what you did.

```
git commit -m "Added hello.c"
```

Once the file has been committed to the project repository, it is mostly safe. If the file were to be deleted, you could restore it using the following command.

```
git checkout hello.c
```

Sometimes there are files created automatically, or you have files you want to exclude. You can mark files to ignore using a **.gitignore** file. This file is a text file that lists the files you want to ignore. You can list files individually or use wildcards.

### *Remote Repositories*

Ideally, you would want to configure Git to have a remote repository so multiple people can make changes, commit those changes, and push and pull to synchronize their repositories with a central server repository.

When setting up a central repository that others will use to synchronize with, you normally want a repository that will not have any files directly committed to it and will just service push and pull requests. This is done by creating a bare repository. Many public repositories are configured using the **bare** option.

## Creating the Bare Repository

Create the bare repository by first creating the directory, then initializing the directory as a git repository. To create a server repository in the **/data/hello** directory we could use the following commands.

```
mkdir /data/hello
cd /data/hello
git --bare init
git update-server-info
```

There are a couple of different ways clients can connect, so let's make sure we can connect over the web, just in case we decide to do so.

```
git config http.receivepack true
```

At this point, we are ready for client machines to connect and use the repository.

## Git Client Configurations

In order for a git client to be able to successfully contribute with push and pull commands, a couple of global variables need to be set. You need to identify who you are and how you are going to communicate with the server. All contributors should have both a name and an email address.

The following commands will take care of setting **Bob**'s global variables up to make contributions.

```
git config --global user.name "Robert"
git config --global user.email "bob@example.com"
git config --global push.default simple
```

Once Bob is configured with global variables, he needs to know how he will connect to the repository. There are a couple of different ways to connect. He can connect locally to his remote repo, he can connect over SSH, or he can connect using the web.

## Local Connection to a Remote Repository

To connect locally to a "remote" repository you need to know the directory the files are stored in. You can use the **git clone** command and pass it the directory. Since the server repo is stored in /data/hello we can clone that.

For Bob to clone a local copy, he will need to decide where he will put his copy. He has a directory called /home/bob/projects so he decides to put it there. The following will clone the project into his local directory.

```
cd /home/bob/projects
git clone file:///data/hello
```

Now, Bob can go into the hello directory and create files, commit changes, and push the changes to the central server repository. There is one little issue. Bob might not have rights to actually make changes on the global repository. This is pretty easy to overcome if he is running as root, but as a standard user, this can be a bit of an issue.

It is probably better to connect using a remote protocol so that rights can be protected and the correct authorization can all happen. For that, we could use SSH or the web.

## Secure Shell Connection to a Remote Repository

In order to connect over SSH you need to make sure you are able to connect to the server machine and you have your client settings taken care of. First, make sure you have followed the Git Client Configurations section from above.

### Secure Shell Key-Based Authentication

If you want to connect without typing a password each time you push or pull updates you should configure SSH to use keys. You can generate a set of keys on the client using the **ssh-keygen** command. You can use use all of the default settings.

```
ssh-keygen
```

After you have created the keys you can install them on the server using the **ssh-copy-id** command. The following command will install the keys in the **root** account on **server.example.com**.

```
ssh-copy-id root@server.example.com
```

Cloning the Repository

Now that you have key-based authentication configured, you can clone the repository in **/data/hello/** on **server.example.com** using the **root** account using the following command.

```
git clone ssh://root@server.example.com/data/hello
```

Once the repository has been cloned, you can test it to make sure you are able to add files, commit them, and push the changes.

```
cd hello
touch README.md
git add .
git commit -m "added README.md"
git push
```

At this point you should see information about what you pushed and version numbers changing. You can pull your results to see if you are up to date and can check the logs as well.

```
git pull
git log
```

## HTTP/HTTPS Connection to a Remote Repository

In order to allow connections using HTTP or HTTPS you need to make sure your Git server is on a machine with Apache installed and you have WebDAV installed. (*Note: WebDAV is usually installed with Apache on CentOS 7*).

### WebDAV Configuration

If your repositories are all stored in the **/data/** directory, then you will need to make those files available over the web. You can do this by creating a directory entry in Apache configuration files. We are going to create a file called **git.conf** and put it in the **/etc/httpd/conf.d/** directory. The file should contain information about the directory and how to access it. We are going to use the alias **/repos/** to redirect us to the **/data/** directory. Here is an example **git.conf** file.

```
<Directory "/data">
  Options Indexes FollowSymLinks
  AllowOverride None
  Require all granted
</Directory>

Alias /repos /data
```

Now, we want to use WebDAV on this directory, so we will add some additional lines to the **git.conf** file. We want to use authentication, so we need a password file as well. To use **/data/.htpasswd** as our

password file, we could use the following lines.

```
<Location /repos>
   DAV on
   AuthType Basic
   AuthName "Git Repositories"
   AuthUserFile /data/.htpasswd
   Require valid-user
</Location>
```

In order to Apache to recognize the changes you will need to either restart or load the Apache web server.

Now that you have a password file, it is probably a good idea to get a password put into that file. We can use the **htpasswd** command to install the password. To create an account for **bob**, we could use the following command. We will also set the password to **aloha123**.

```
htpasswd -c /data/.htpasswd bob
```

The next thing you need to worry about is file system level access. If the **/data/** directory and any projects held within are not owned by the user that the Apache web server is running as, you might have permission issues. Typically, Apache runs as the **apache** user, so make sure all of the files are owned by Apache.

If you are using SELinux, you want to make sure the context type allows read/write access. Change the context type for the **/data/** directory and sub-directories and files to **httpd_sys_rw_content_t**.

```
chcon -t httpd_sys_rw_content_t /data/ -R
```

Make sure the firewall allows communication as well, since you probably want to make your repositories available to client machines.

### Web Client Configuration

If you have not already done so, make sure your client machine can connect to the server over the web ports. You can use a GUI web browser of install a package like **links** to connect. Make sure you have followed the Git Client Configurations section from above.

Since we created a project on **server.example.com** called **hello** in the **/data/** directory, which is being identified as **/repos/** over the web, we would use the following **clone** command.

```
git clone http://server.example.com/repos/hello/
```

You should be prompted for a user name and password. Since we are using the user **bob** with the password of **aloha123** we would just need to make type those when prompted.

If we want to speed up the authentication process, we can create a file called **~/.netrc** with the following lines.

```
machine server.example.com
login bob
password aloha123
```

Only bob needs to see the file, so if you are worried about security, you can take away the rights of other users to see the contents.

```
chmod 600 ~/.netrc
```

### Adding Committing and Pushing

When you have the directory cloned, you can go in and add files and make changes. Many people start with a README.md file. The following will create an empty file, add it to the project, commit it, then push it to the server.

```
cd hello
touch README.md
git add .
git commit -m "added README.md"
git push
```

At this point you should see information about what you pushed and version numbers changing. You can pull your results to see if you are up to date and can check the logs as well.

```
git pull
git log
```

### Insecure Web Clients

If you are trying to self-sign a certificate for use over HTTPS you might not pass the certificate validation. In order to avoid this problem you could either get a valid certificate, or ignore the certificate validation. The following command will ignore validation.

```
git config --global http.sslVerify false
```

# Configuration Management: Ansible

Ansible is a Configuration Management software package that makes it easy to execute a set of commands on a group of machines remotely using push requests. This can be really nice because the client machines do not need to have a lot of configuration to be ready to receive commands.

## *Ansible Server Configuration*

To configure Ansible, you need to first install the **epel-release** repository. After installing the EPEL repository, you will want to get **python-pip** for communication with Windows clients and then the **ansible** package. For Windows clients you will want to have the **pywinrm** package installed. The following lines should get your basic software installed.

```
yum install epel-release
yum install python-pip
yum install ansible
pip install pywinrm
```

Ansible uses SSH with key based authentication to send commands from the server to Linux clients. In order to make this happen, you need to generate a public/private key pair on the server and the client needs to have the server's public key listed in the **authorized_keys** file.

Generate the keys using the **ssh-keygen** command. Do not change any of the values, just press enter at each prompt.

```
ssh-keygen
```

You can copy the public key to the client machine using the **scp** command. The following command will copy the server's key to the client machine's root user directory. Remember to replace **CLIENTIP** with the IP address of the client machine.

```
scp /root/.ssh/id_rsa.pub root@CLIENTIP:/root/server.pub
```

When prompted, type the client machine's root password. After this is done, you just need to add the client machine's IP address to the list of hosts.

Edit the **/etc/ansible/hosts** file and add the client's IP address on a single line by itself. This will get the client into the list of "all" clients. At this point you are ready to configure the client machine.

## *Ansible Client Configuration*

Log into the Ansible client machine as root. You need to have a correctly configured .ssh/ directory. If you already have one, great, if not, it is probably easiest to create one using the **ssh-keygen** command like you did with the server machine.

After you have a working **/root/.ssh/** directory, you are ready to add the server's public key to your **authorized_keys** file. You can most safely do this using the cat command to append the contents of the **/root/server.pub** file to the **/etc/.ssh/authorized_keys** file using the following command.

```
cat /root/server.pub >> /root/.ssh/authorized_keys
```

Remember to make sure you use both greater than signs. If you only have one you will overwrite the contents of the authorized_keys file. This may or may not be a problem.

## *Testing Ansible Server/Client Communication*

You can test the Ansible server's ability to send communication to the client machines using the following commands. This sends a ping message to all clients.

```
ansible all -m ping
```

This command should ping all client machines listed in the **/etc/ansible/hosts** file. If you created groups you can use the group name in place of the "all" option. For examples you can take a look at the contents of the **/etc/ansible/hosts** file.

If you want to send a command to execute on the client machines you can use the **-a** switch. The following command will reboot all of your Ansible client machines.

```
ansible all -a '/sbin/reboot'
```

Note, when sending commands, Ansible likes to wait until the command completes before reporting on the status of the command. The reboot command does not give you the satisfaction of a successful response because it reboots before you get a status reply.

# Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol makes it possible to centrally manage a number of client and server machines and collect information from them using the same protocol. In addition to clients and servers, many network appliances and network infrastructure also have the SNMP protocol available and in many cases running by default.

## *Installing net-snmp*

In order to use the SNMP protocol on a Linux machine you need to install the software and start it. The **net-snmp** package provides a nice SNMP server and the **net-snmp-utils** package provides great testing and troubleshooting utilities. Go ahead and install the packages using the following command.

```
yum install net-snmp net-snmp-utils
```

At this point, you could technically turn on the SNMP server and be ready to go. Sometimes it is better to make a few security changes.

## *Configuration*

SNMP uses two default passwords which are referred to as community strings. New, and often older, SNMP enabled devices normally have both of these two community strings set to the default values. For read-only information the default community string is "**public**" for read-write information the default community string is "**private**". If you want to allow random people to read information from your servers and other devices then you can leave these values alone.

The **net-snmp** server is also configured to use "**public**" as the default community string for read-only information collection. Configuration changes should be made to the **/etc/snmp/snmpd.conf** file. To change the default string you will need to find the line in that file that defines the community string as "**public**".

```
com2sec notConfigUser default public
```

You then need to change the password from public to something more secure. To change it to "**aloha123**" you would change the line to look like the following line.

```
com2sec notConfigUser default aloha123
```

The SNMP server also supports both the SNMP version 1 and version 2c protocols. If you want to disable one of these, you can turn it off by commenting it. To disable SNMPv1 you could comment the following line.

```
# group notConfigGroup v1 notConfigUser
```

### *Running snmpd*

Once you have your **/etc/snmp/snmp.conf** file configured the way you want, you are ready to start the **snmpd** service. To start the service you can use the following command.

```
systemctl start snmpd
```

If you want to **stop** or **restart** the service you can use those options. If you want to have the snmpd service running at boot time you can use the enable option.

```
systemctl enable snmpd
```

## Firewall

With the snmpd service running you should be able to connect internally, but if you want to connect from external machines, you will need to open the firewall. The following line will open the firewall for SNMP communication.

```
firewall-cmd --add-service=snmp
```

If you want the firewall rules to persist over restarts of the firewall or machine, you will need to add the **--permanent** option as well.

```
firewall-cmd --add-service=snmp --permanent
```

### *Testing SNMP*

In the **net-snmp-utils** package are two nice utilities for testing SNMP devices. The **snmpget** command performs a single query of a SNMP value and the **snmpwalk** command tries to collect a lot of information by requesting multiple information values.

Here are some sample queries that can be performed on **localhost** assuming your community string is "**aloha123**" and you want to use version **2c** of the protocol.

To get a single value you could use the following command:

```
snmpget localhost -v 2c -c aloha123 sysName.0
```

To get all values within **system** you could use the following command:

```
snmpwalk localhost -v 2c -c aloha123 system
```

# Configuring rsyslog

CentOS 7 and other Red Hat Linux based systems use **rsyslog** for systems logs. By default, the rsyslog service is running and logging information in the **/var/log/** directory. Configuration files for the rsyslog service are found at **/etc/rsyslog.conf** and in the **/etc/rsyslog.d/** directory.

## *Configure for Receiving Log Messages*

The default method of sending logs to a remote syslog server is over **UDP** port **514**. In order to have your rsyslog server listen on that port, you will need to edit the **/etc/rsyslog.conf** file and uncomment the following lines.

```
$ModLoad imudp
$UDPServerRun 514
```

Once fully configured, you can use commands like "**logging IPADDR**" or "**logging server IPADDR**" on a Cisco friendly router or switch to send logging messages to your syslog server.

## *Running snmpd*

Once you have your **/etc/rsyslog.conf** file configured the way you want, you will need to make sure the service is restarted. To start the reservice you can use the following command.

```
systemctl restart rsyslog
```

If you want to **stop** or **start** the service you can use those options. If you want to change if the rsyslog service is running at boot time you can use the **enable** or **disable** options.

## Firewall

When we configure the rsyslog service to listen on UDP port 514, we also need to make sure the firewall is allowing connections to that port. The **syslog** firewall service will make that port available.

```
firewall-cmd --add-service=syslog
```

If you want the firewall rules to persist over restarts of the firewall or machine, you will need to add the **--permanent** option as well.

```
firewall-cmd --add-service=syslog --permanent
```

# Firewalls

A very important step in securing your system is making sure only needed services are installed and running on the system and that services are only binding to required interfaces. While many services do this well, there are situations where the services cannot keep themselves safe. Firewalls can help bridge the gap by adding additional security using a standard interface that you can become familiar with and rely on.

Because of the modular nature of Linux distributions there are multiple options for firewalls. Some of the firewalls are older and more documented on the Internet and some of them are newer and less documented. Just because websites list instructions which specify commands using a particular firewall does not mean that those rules will work for your situation.

## *FirewallD*

The current firewall used by default on CentOS Linux machines is FirewallD. This firewall software package has a command line feel similar to the rest of systemd. While firewalld is the default firewall, you still have the ability to use older firewalls if you want to.

To check the status of the firewall, you can use the systemctl status command for the firewalld.service service.

```
systemctl status firewalld
```

Your firewall should be running by default, but if not, you can easily start it using the following command.

```
systemctl start firewalld
```

## Command: firewall-cmd

The **firewall-cmd** command is used to make most configuration changes to the firewall. The configurations can be either to the active configuration or to the permanent configurations. The active configurations take effect immediately, but disappear when the service is restarted. The permanent changes do not take effect until the service has been restarted.

You can see the current active configuration using the following command.

```
firewall-cmd --list-all
```

If you wanted to see the permanent configuration you could add the **--permanent** option and use the following command instead.

```
firewall-cmd --permanent --list-all
```

Additionally, the permanent firewall configurations are stored in XML files in the **/etc/firewalld/zones/** directory. Each file is named with the zone it contains information for. The standard configurations are put in the public zone. To see the XML of the configuration file you could use the cat command. To see the public.xml file use the following command.

```
cat /etc/firewalld/zones/public.xml
```

The following commands show you information about the zones used in firewalld and additional zone options.

```
firewall-cmd --get-active-zones
```

```
firewall-cmd --get-all-zones
```

## Service Based Rules

When configuring rules, you need to know where you are installing the rule and what the rule affects. You can create rules for individual IP addresses and ports or services. To get list of services you can use the following command.

```
firewall-cmd --get-services
```

Each of these services is actually defined in XML files in the **/usr/lib/firewalld/services/** directory. Below is a sample dialog showing a query for a list of services.

```
bash# firewall-cmd --get-services
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns ftp high-
availability http https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps libvirt
libvirt-tls mdns mountd ms-wbt mysql nfs ntp openvpn pmcd pmproxy pmwebapi pmwebapis
pop3s postgresql proxy-dhcp radius rpc-bind samba samba-client smtp ssh telnet tftp tftp-
client transmission-client vnc-server wbem-https
```

Each of these can be used instead of specifying actual port numbers. This can be useful, especially when the service uses more than one standard port. The **samba** service is an example that uses more than one port to operate. You can allow services by adding them to the list of those made publicly available with the **--add-service** command line option.

The following dialog shows what would be allowed if you added the **samba** service to the firewall.

```
bash# cat /usr/lib/firewalld/services/samba.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>Samba</short>
  <description>This option allows you to access and participate in Windows file and
printer sharing networks. You need the samba package installed for this option to be
useful.</description>
  <port protocol="udp" port="137"/>
  <port protocol="udp" port="138"/>
  <port protocol="tcp" port="139"/>
  <port protocol="tcp" port="445"/>
  <module name="nf_conntrack_netbios_ns"/>
</service>
```

The following is an example command line to make the **samba** service available on the public interface.

```
firewall-cmd --zone=public --add-service=samba
```

If you want to later remove that service from the firewall rules you can use the **--remove-service** option. The following line would remote the samba service.

```
firewall-cmd --zone=public --remove-service=samba
```

At any time you can query the firewall for a list of services using the **--list-services** command line option.

```
firewall-cmd --zone=public --list-services
```

These services have been added to the active portion of the firewall. They are only in memory and will not be there when the firewall restarts. To make the rules permanent, you will need to add the **--permanent** option as well. To add, remove, and check the permanent rules for **samba** you could use the following commands.

```
firewall-cmd --permanent --zone=public --add-service=samba
```

```
firewall-cmd --permanent --zone=public --remove-service=samba
```

```
firewall-cmd --permanent --list-services
```

## Port Based Rules

If you do not have a service for a given port or if you have deviated from the standard port assignments, you might have to create a port based rule to allow communication through the firewall.

You can add allowed ports using the **--add-port** command line option.

The following line is an example that you can use to allow **http** traffic by port number (tcp port 80) instead of by the service http.

```
firewall-cmd --zone=public --add-port=80/tcp
```

Just like with services, you can also remove ports from the allow list. You can remove ports with the **--remove-port** command line option. The following command would remove tcp port 80 from the list of allowed ports.

```
firewall-cmd --zone=public --remove-port=80/tcp
```

At any time you can get a list of the approved ports using the **--list-ports** command line option.

```
firewall-cmd --zone=public --list-ports
```

Once again, these rules are only in memory and not permanent. To make them permanent, use the **--permanent** option. The add, remove, and listing commands to show a permanent port 80 would look like the following.

```
firewall-cmd --permanent --zone=public --add-port=80/tcp
```

```
firewall-cmd --permanent --zone=public --remove-port=80/tcp
```

```
firewall-cmd --permanent --list-ports
```

## Rich Rules

Most configurations can be handled with the services and ports, but sometimes you want a more complex rule. Rules that restrict access based on source and destination ports. If you wanted to deny everyone on the 10.0.0.0/8 network from connecting to your TCP port 80, you could use the following command.

```
firewall-cmd --add-rich-rule='rule family=ipv4 port port=80 protocol=tcp source
address=10.0.0.0/8 drop'
```

You would also need to make sure the whole command above is on a single line.

There are a lot of nice rules you can create and you can see the documentation by looking at the following manual page.

```
man firewalld.richlanguage
```

## NAT Masquerade

Network Address Translation (NAT) has been an important part of networking for a long time. One of the important features of Linux firewalls is the ability to perform network address translation. On Linux machines NAT is commonly referred to as masquerading addresses.

Assume you have two interfaces, one inside and one outside. If the inside interface is called ens32 and the outside interface is called ens34 you could use the following lines to configure masquerading.

```
firewall-cmd --direct --add-rule ipv4 nat POSTROUTING 0 -o ens32 -j MASQUERADE
firewall-cmd --direct --add-rule ipv4 filter FORWARD 0 -i ens34 -o ens32 -j ACCEPT
firewall-cmd --direct --add-rule ipv4 filter FORWARD 0 -i ens32 -o ens34 -m state --state
RELATED,ESTABLISHED -j ACCEPT
```

The first line lists the ens32 interface as the outside interface to be used for the masquerade IP address. The second line allows forwarding through the firewall from the inside to the outside for everything. (Note: the kernel might still not allow routing). The last line allows incoming packets from the outside to be translated into internal addresses.

To configure the kernel to allow routing see the IP Forwarding section of Network Routing below.

## Blocking ICMP Pings

ICMP packets provide a lot of Internet functionality. They are used for many different things including the well known ping tests and lesser known flow control information messages. You can get a list of ICMP types using the following command.

```
firewall-cmd --get-icmptypes
```

The two types used by the ping command are the ICMP **echo-request** and the **echo-reply**. If you wanted to block the echo-reply you could use one of the following commands.

```
firewall-cmd --zone=public --query-icmp-block=echo-reply
firewall-cmd --zone=public --add-icmp-block=echo-reply
```

## *IPTables*

The iptables firewall was the default on many Linux distributions for a long time. The iptables firewall was replaced by firewalld as the default with the change from Cent OS 6 to Cent OS 7. Even though it is no longer the default, it can still be installed and used. If you have a mixed environment with both newer and older Linux distributions then chances are good that you will need to configure iptables at some point.

## *IPChains*

IPTables is based on IPChains, but added some stateful elements and made things a little bit more complex. In older versions of RedHat and similar the ipchains firewall rules were stored in the **/etc/sysconfig/ipchains** file.

Rules added to one of the following three chains: input, output, and forward. Anything entering an interface is considered input. Anything exiting an interface is considered output. Anything passing through the interface such as routed traffic or NAT traffic is considered part of the forward chain.

The following are some example firewall rules.

```
-A input -s 0.0.0.0/0 -d 0.0.0.0/0 21:21 -p 6 -j DENY
-A input -s 0.0.0.0/0 -d 0.0.0.0/0 69:67 -p 17 -j DENY
-A forward -s 10.0.0.0/8 -d 0.0.0.0/0 -j MASQ
```

# Network Routing

Many people think of Linux machines as infrastructure servers, but they can do a lot more. In addition to running servers, Linux machines can act as gateway firewalls, NAT routers, and more.

## *IP Forwarding*

There are a few variables you can change to make a difference in how the kernel acts. In the **/proc/sys/net/** directory there are a few files that affect networking. One common change people make is to enable IP forwarding. This is used when your machine is acting as a router.

To enable IP forwarding you can change the value of the variable in the **/proc/sys/net/ipv4/ip_forward** file to be 1 instead of 0. The following command makes the change in kernel memory only.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

You can additionally use the **sysctl** command to change the value.

```
sysctl -w net.ipv4.ip_forward=1
```

To make the changes permanent you need to edit the **/etc/sysctl.conf** file and add the configuration change. You need to make sure the following line is added to the /etc/sysctl.conf file.

```
net.ipv4.ip_forward = 1
```

After making sure the line is in the file you can use the following command to use the **sysctl.conf** file.

```
sysctl -p /etc/sysctl.conf
```

## *Static Routing*

Machines use routing and routes to make decisions about where to send packets. Normal machines only know about their local networks and how to get to the default gateway. Sometimes there are additional routers that your machine is aware of and can use. You can see all of your routes using the "**ip route**" command.

```
ip route
```

If you want to statically add an additional route you can add some options to the ip route command. To add a route to the 172.16.0.0/24 subnet from your current LAN using 192.168.0.10 as the router to get there you could use the following command.

```
ip route add 172.16.0.0/24 via 192.168.0.10
```

# Security Enhanced Linux (SELinux)

The SELinux additions were brought to us by the National Security Agency (NSA). For a while people were concerned that the NSA had hidden backdoor code to allow them to take over the machines. It is always difficult to trust spy agencies. Eventually the SELinux code has come to be trusted and used by many different Linux distributions.

SELinux provides modifications to the kernel that allow administrators to define the possible areas that a process should be allowed to touch. If you have a service running as the root user then that process could potentially cause severe damage if it went rogue. SELinux enforces protections at the kernel level that block operations that otherwise root would have been allowed to perform.

SELinux creates a mandatory access control system by labeling files, directories, and processes with contexts. It then prevents processes from even seeing files not allowed by the SELinux rules.

## File: /etc/sysconfig/selinux

This file contains the main controlling options for running SELinux. The most important option for users new to SELinux is the SELINUX variable. This variable can be set to enforcing, permissive, or disabled.

When SELINUX is set to **enforcing** all policies are in place and are followed. This means that anything that tried to do operations which are not permitted will be prevented. System administrators who are not aware of SELinux can spend hours trying to troubleshoot simple problems where servers such as Apache cannot see a file. This is by design, not the frustration, but the limitations which make the system more secure from outside attacks from worms and other malware that might compromise the service. Each SELinux violation will be logged with the Access Vector Cache (AVC) label. To set SELinux to enforcing, edit the /etc/sysconfig/selinux file and make sure the following line is present.

```
SELINUX=enforcing
```

If you have changed the line, go ahead and reboot the system for the changes to take affect, or use the **setenforce** command to change the value.

When SELINUX is set to **permissive** all policies are in place and are looked at, but not enforced. SELinux will continue to log all instances when the system would have blocked something. This can be very useful in situations where you are not sure where files are not labeled with the correct context and want to know.

When SELINUX is set to **disabled** all policies are ignored and the system administrators do not have to worry about SELinux being the cause of services failing.

This file is read when the SELinux system starts, but is not read again while SELinux is running. If you want to make changes to the state of SELinux at boot time this file is the one to edit. If you want to make more immediate changes then the **setenforce** command is more for you.

## Command: setenforce

When you want to make real time changes to the state of SELinux the setenforce command is your friend. The following dialog shows the setenforce options:

```
bash# setenforce
usage:  setenforce [ Enforcing | Permissive | 1 | 0 ]
```

Usually the setenforce command uses either the 0 or 1 argument. The 1 argument turns on SELinux and the 0 argument turns it off. To turn off the SELinux system use the following command:

```
setenforce 0
```

To turn it back on, use the following command:

```
setenforce 1
```

## Command: getenforce

Sometimes you do not know what the state of SELinux is currently running at. If you are unsure and you suspect SELinux is causing you problems you can use the getenforce command.

```
getenforce
```

This command returns the status of the SELinux system state.

## File and Directory Security Contexts

When you use SELinux, each file and directory is assigned a security context. These contexts are made of three parts; a set user, set role, and set type. Using the **-Z** option with the **ls** command you can see the context of each file and directory. Below is a sample dialog showing the context of the / directory.

```
bash$ ls -Z /
lrwxrwxrwx. root      root      system_u:object_r:bin_t:s0       bin -> usr/bin
dr-xr-xr-x. root      root      system_u:object_r:boot_t:s0      boot
drwxr-xr-x. root      root      system_u:object_r:device_t:s0    dev
drwxr-xr-x. root      root      system_u:object_r:etc_t:s0       etc
drwxr-xr-x. root      root      system_u:object_r:home_root_t:s0 home
lrwxrwxrwx. root      root      system_u:object_r:lib_t:s0       lib -> usr/lib
lrwxrwxrwx. root      root      system_u:object_r:lib_t:s0       lib64 -> usr/lib64
drwxr-xr-x. root      root      system_u:object_r:mnt_t:s0       media
drwxr-xr-x. root      root      system_u:object_r:mnt_t:s0       mnt
drwxr-xr-x. root      root      system_u:object_r:usr_t:s0       opt
```

```
dr-xr-xr-x. root      root      system_u:object_r:proc_t:s0      proc
dr-xr-x---. root      root      system_u:object_r:admin_home_t:s0 root
drwxr-xr-x. root      root      system_u:object_r:var_run_t:s0   run
lrwxrwxrwx. root      root      system_u:object_r:bin_t:s0       sbin -> usr/sbin
drwxr-xr-x. root      root      system_u:object_r:var_t:s0       srv
dr-xr-xr-x. root      root      system_u:object_r:sysfs_t:s0     sys
drwxrwxrwt. root      root      system_u:object_r:tmp_t:s0       tmp
drwxr-xr-x. root      root      system_u:object_r:usr_t:s0       usr
drwxr-xr-x. root      root      system_u:object_r:var_t:s0       var
```

This forces the directory listing to be in the long format and includes the normal field columns, but also includes a column which displays the context. The first line of the directory listing contains the context " **system_u:object_r:bin_t:s0**". The context can be divided by splitting on the colon ":" into the user, role, and type and an additional "s0" level value.

## Command: chcon

You can set the context of a file or directory using the **chcon** command with the **-t** option followed by the new context. If you were using Samba and wanted to make a share called **/music**, you might use the following command to tell SELinux to allow Samba to access it.

```
chcon -t samba_share_t /music
```

This would only change the directory itself. To change the whole directory tree, you would use the **-R** recursive option.

```
chcon -R -t samba_share_t /music
```

If you wanted to change the user or role associated with a context, you could use the **-u** or **-r** options respectively.

## Command: restorecon

If you do not know what context a file is supposed to have, you can use the **restorecon** command. This command looks up the listed context for the area in the file system as sets the context. Because not all directories and files you will create are known, it is possible to set the context incorrectly. For normal use, however, the **restorecon** command should work well.

If somehow messed up your **/etc/passwd** file context, the following command should fix it.

```
restorecon /etc/passwd
```

If you have a lot of files to restore and do not want to do them individually, you can create an **.autorelabel** file in your root directory and have it done the next time you boot your system.

```
bash# touch /.autorelabel
bash# reboot
```

## Command: getsebool

SELinux has multiple sets of rules for controlling various behaviors in services and applications. To get a complete listing of the rule sets you can use the **-a** option with the **getsebool** command.

```
getsebool -a
```

If you want to get the boolean value of a single variable name you can pass the name of the variable as a single argument. To see if **httpd.service** is allowed to make back end network connections such as to remote databases we might check the **httpd_can_network_connect** variable. The resulting dialog might look like the following.

```
bash# getsebool httpd_can_network_connect
httpd_can_network_connect --> off
```

## Command: setsebool

If you want to set the boolean value of one of the pre-configured rules sets in SELinux, you can use the **setsebool** command to set it. The values are **on** and **off**, but can also be represented on the command line as **0** and **1** or **true** and **false**. Additionally, you can either set the variable value in just memory or permanently in the policy as well as in active memory.

To set a value in just active memory you can use the **setsebool** command and pass it the variable name and the new value to set. The following would set the **httpd_can_network_connect** variable value to **on**.

```
setsebool httpd_can_network_connect on
```

If I wanted to make it permanent, I would add the **-P** option as well.

```
setsebool -P httpd_can_network_connect on
```

## Command: semanage

The **semanage** command can be used to change the default context type behavior of files in a directory. For example, if you wanted to change the default SELinux context type for files and directories created in the **/var/www/html/** directory so that they were read-write by Apache, instead of just read only, you could use the following command to change the default context type to **httpd_sys_rw_content_t**.

```
semanage fcontext -a -t httpd_sys_rw_content_t '/var/www/html(/.*)'
```

If you are interested in viewing current policy rules, you can use the list option with the semanage command. The following command will display a long list of SELinux policy rules.

```
semanage fcontext --list
```

You can filter the list using the **grep** command. For example, to see only default context policies for the **httpd_sys_content_t** type you could use the following command.

```
semanage fcontext --list | grep httpd_sys_content_t
```

# Index