Tecmint.com
Linux Howto's Guide

Red Hat RHCSA/RHCE Certification Preparation Study Guide

redhat

RedHat RHCSA / RHCE 7

RHCSA (EX200) and RHCE (EX300) exams

* EX200 - Red Hat Certified System Administrator (RHCSA)
* EX300 - Red Hat Certified Engineer (RHCE)

Tecmint.com
Linux Howto's Guide

By TecMint.com Team

Table of Contents

Important notice

DISCLAIMER: We highly encourage you to become familiar with the tools used in this ebook and their man pages. Although we have done our best to cover the essential concepts associated with these certifications, do not assume that after going through this ebook you are fully qualified to pass the RHCSA and RHCE exams. This ebook –although it is as complete as possible- is intended as a starting point and not as an exhaustive guide system administration.

We hope you will enjoy reading this ebook as much as we enjoyed writing it and formatting it for distribution in PDF format. You will probably think of other ideas that can enrich this ebook. If so, feel free to drop us a note at one of our social network profiles:


http://twitter.com/tecmint


https://www.facebook.com/TecMint


https://plus.google.com/+Tecmint

Questions and other suggestions are appreciated as well – we look forward to hearing from you!

Chapter 1: Reviewing essential commands & system documentation

In this article we will explain how to enter and execute commands with the correct syntax in a shell prompt or terminal, and explained how to find, inspect, and use system documentation.

Interacting with the shell

If we log into a Linux box using a text-mode login screen, chances are we will be dropped directly into our default shell. On the other hand, if we login using a graphical user interface (GUI), we will have to open a shell manually by starting a terminal. Either way, we will be presented with the user prompt and we can start typing and executing commands (a command is executed by pressing the **Enter** key after we have typed it).

Commands are composed of two parts:

1) the name of the command itself, and

2) arguments.

Certain arguments, called *options* (usually preceded by a hyphen), alter the behavior of the command in a particular way while other arguments specify the objects upon which the command operates.

The **type** command (see Fig. 1) can help us identify whether another certain command is built into the shell or if it is provided by a separate package. The need to make this distinction lies in the place where we will find more information about the command. For shell built-ins we need to look in the shell's man page, whereas for other binaries we can refer to its own man page.

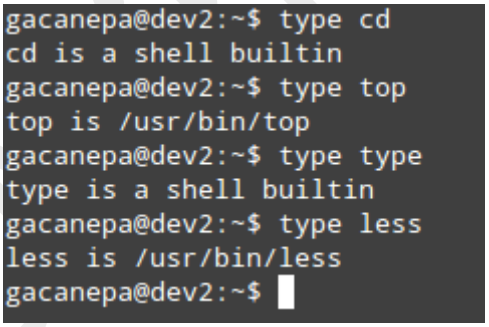


Figure 1: Using type to determine a file's type

In the examples above, **cd** and **type** are shell built-ins, while **top** and **less** are binaries external to the shell itself (in this case, the location of the command executable is returned by **type**).

Other well-known shell built-ins include (see Fig. 2):

1) echo: Displays strings of text.

2) pwd: Prints the current working directory.

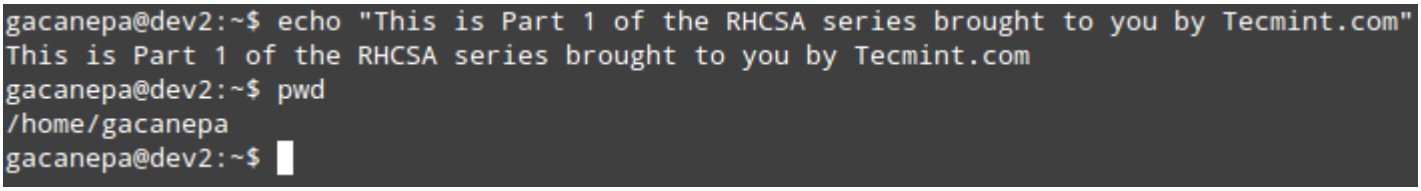


Figure 2: Other shell built-ins

3) exec: Runs an external program that we specify. Note that in most cases, this is better accomplished by just typing the name of the program we want to run, but the exec command has one special feature: rather than

create a new process that runs alongside the shell, the new process replaces the shell, as can verified by subsequent

```
ps -ef | grep [original PID of the shell process]
```

When the new process terminates, the shell terminates with it. Run **exec top** and then hit the q key to quit top. You will notice that the shell session ends when you do.

4) export: Exports variables to the environment of subsequently executed commands.

5) history: Displays the command history list with line numbers. A command in the history list can be repeated by typing the command number preceded by an exclamation sign. If we need to edit a command in history list before executing it, we can press Ctrl + r and start typing the first letters associated with the command. When we see the command completed automatically, we can edit it as per our current need.

This list of commands is kept in our home directory in a file called **.bash_history**. The history facility is a useful resource for reducing the amount of typing, especially when combined with command line editing. By default, bash stores the last 500 commands you have entered, but this limit can be extended by using the HISTSIZE environment variable (see Fig. 3):

```
gacanepa@dev2:~$ echo $HISTSIZE
500
gacanepa@dev2:~$ export HISTSIZE=1000
gacanepa@dev2:~$ echo $HISTSIZE
1000
gacanepa@dev2:~$
```

Figure 3: Command history size

But this change -as performed above- will not be persistent on our next boot.

In order to preserve the change in the HISTSIZE variable, we need to edit the .bashrc file by hand:

```
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
```

Keep in mind that these changes will not take effect until we restart our shell session.

6) alias: With no arguments or with the -p option prints the list of aliases in the form alias name=value on standard output. When arguments are provided, an alias is defined for each name whose value is given.

With alias, we can make up our own commands or modify existing ones by including desired options. For example, suppose we want to alias **ls** to **ls --color=auto** so that the output will display regular files, directories, symlinks, and so on, in different colors (see Fig. 4):

```
alias ls='ls --color=auto'
```

```
root@dev2:~# ls
backups  calcurse.txt  Release.key  sent  swapkeyfile  trace.scap
root@dev2:~# alias ls='ls --color=auto'
root@dev2:~# ls
backups  calcurse.txt  Release.key  sent  swapkeyfile  trace.scap
root@dev2:~# alias
alias ls='ls --color=auto'
root@dev2:~#
```

Figure 4: Using alias

Note that you can assign any name to your “new command” and enclose as many commands as desired between single quotes, but in that case you need to separate them by semicolons, as follows:

```
alias myNewCommand='cd /usr/bin; ls; cd; clear'
```

7) exit: The exit and logout commands both terminate the shell. The exit command terminates any shell, but the logout command terminates only login shells—that is, those that are launched automatically when you initiate a text-mode login.

If we are ever in doubt as to what a program does, we can refer to its man page, which can be invoked using the man command. In addition, there are also man pages for important files (**inittab**, **fstab**, **hosts**, to name a few), library functions, shells, devices, and other features.

Examples:

- **man uname** (print system information, such as kernel name, processor, operating system type, architecture, and so on)
- **man inittab** (init daemon configuration)

Another important source of information is provided by the info command, which is used to read info documents. These documents often provide more information than the man page. It is invoked by using the info keyword followed by a command name, such as

- **info ls**
- **info cut**

In addition, the /usr/share/doc directory contains several subdirectories where further documentation can be found. They either contain plain-text files or other friendly formats.

Make sure you make it a habit to use these three methods to look up information for commands. Pay special and careful attention to the syntax of each of them, which is explained in detail in the documentation.

Converting tabs into spaces with expand

Sometimes text files contain tabs but programs that need to process the files don't cope well with tabs. Or maybe we just want to convert tabs into spaces. That's where the **expand** tool (provided by the GNU coreutils package) comes in handy.

For example, given the file NumbersList.txt, let's run expand against it, changing tabs to one space, and display on standard output (see Fig. 5).

```
expand --tabs=1 NumbersList.txt
```

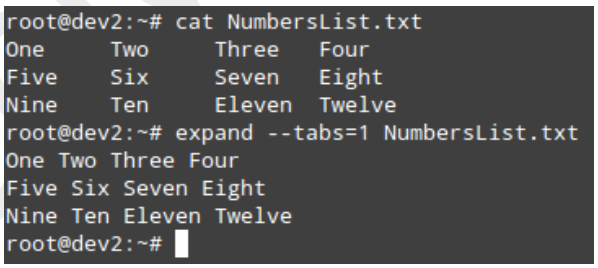


Figure 5: Using expand to convert tabs into spaces

The **unexpand** command performs the reverse operation (converts spaces into tabs).

Display the first lines of a file with head and the last lines with tail

By default, the head command followed by a filename, will display the first 10 lines of the said file. This behavior can be changed using the **-n** option and specifying a certain number of lines (see Fig. 6).

```
head -n3 /etc/passwd
tail -n3 /etc/passwd
```

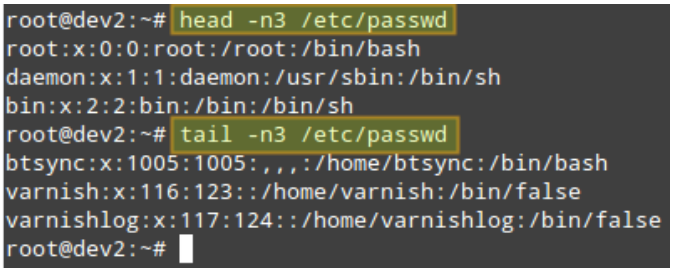


Figure 6: Using head and tail

One of the most interesting features of tail is the possibility of displaying data (last lines) as the input file grows (**tail -f my.log**, where my.log is the file under observation). This is particularly useful when monitoring a log to which data is being continually added.

Merging Lines with paste

The paste command merges files line by line, separating the lines from each file with tabs (by default), or another delimiter that can be specified (in the following example the fields in the output are separated by an equal sign). Refer to Figure 7 for details:

```
paste -d= file1 file2
```

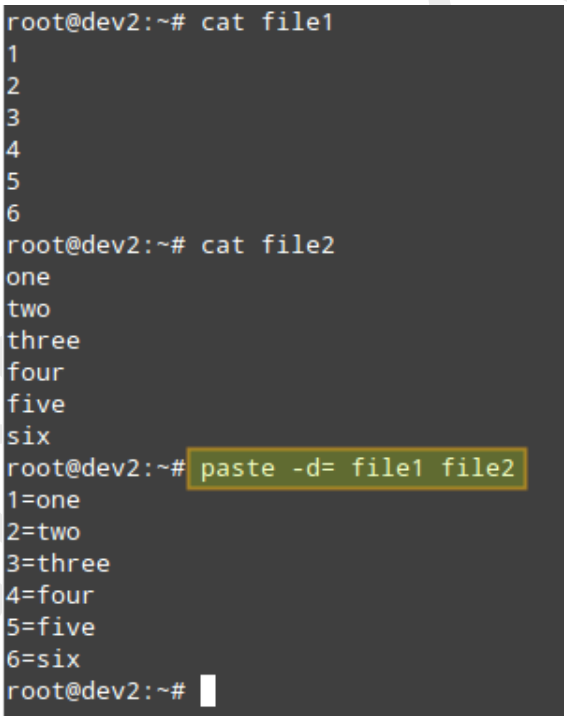


Figure 7: Using paste

Breaking a file into pieces using split

The **split** command is used split a file into two (or more) separate files, which are named according to a prefix of our choosing. The splitting can be defined by size, chunks, or number of lines, and the resulting files can have a numeric or alphabetic suffixes. In the following example, we will split bash.pdf into files of size 50 KB (-b 50KB), using numeric suffixes (-d). Refer to Fig. 8 for details:

```
split -b 50KB -d bash.pdf bash_
```

```
root@dev2:~# ls -lh | grep bash | grep -v grep
-rw-r--r-- 1 root root 290K Feb 21 12:18 bash.pdf
root@dev2:~# split -b 50KB -d bash.pdf bash_
root@dev2:~# ls -lh | grep bash | grep -v grep
-rw-r--r-- 1 root root  49K Feb 21 12:18 bash_00
-rw-r--r-- 1 root root  49K Feb 21 12:18 bash_01
-rw-r--r-- 1 root root  49K Feb 21 12:18 bash_02
-rw-r--r-- 1 root root  49K Feb 21 12:18 bash_03
-rw-r--r-- 1 root root  49K Feb 21 12:18 bash_04
-rw-r--r-- 1 root root  46K Feb 21 12:18 bash_05
-rw-r--r-- 1 root root 290K Feb 21 12:18 bash.pdf
root@dev2:~#
```

Figure 8: Using split

You can merge the files to recreate the original file with the following command:

```
cat bash_00 bash_01 bash_02 bash_03 bash_04 bash_05 > bash.pdf
```

Translating characters with tr

The **tr** command can be used to translate (change) characters on a one-by-one basis or using character ranges. In the following example we will use the same file2 as previously, and we will change 1) lowercase o's to uppercase, and 2) all lowercase to uppercase (see Fig. 9):

```
cat file2 | tr o O
cat file2 | tr [a-z] [A-Z]
```

```
root@dev2:~# cat file2
one
two
three
four
five
six
root@dev2:~# cat file2 | tr o O
One
twO
three
fOur
five
six
root@dev2:~# cat file2 | tr [a-z] [A-Z]
ONE
TWO
THREE
FOUR
FIVE
SIX
root@dev2:~#
```

Change only lowercase o's to uppercase

Change all lowercase to uppercase

Figure 9: Translating characters with tr

Reporting or deleting duplicate lines with uniq and sort

The **uniq** command allows us to report or remove duplicate lines in a file, writing to stdout by default. We must note that **uniq** does not detect repeated lines unless they are adjacent. Thus, **uniq** is commonly used along with a preceding sort (which is used to sort lines of text files). By default, sort takes the first field (separated by spaces) as key field. To specify a different key field, we need to use the **-k** option. Please note

how the output returned by **sort** and **uniq** change as we change the key field in the following example (see Fig. 10):

```
cat file3
sort file3 | uniq
sort -k2 file3 | uniq
sort -k3 file3 | uniq
```

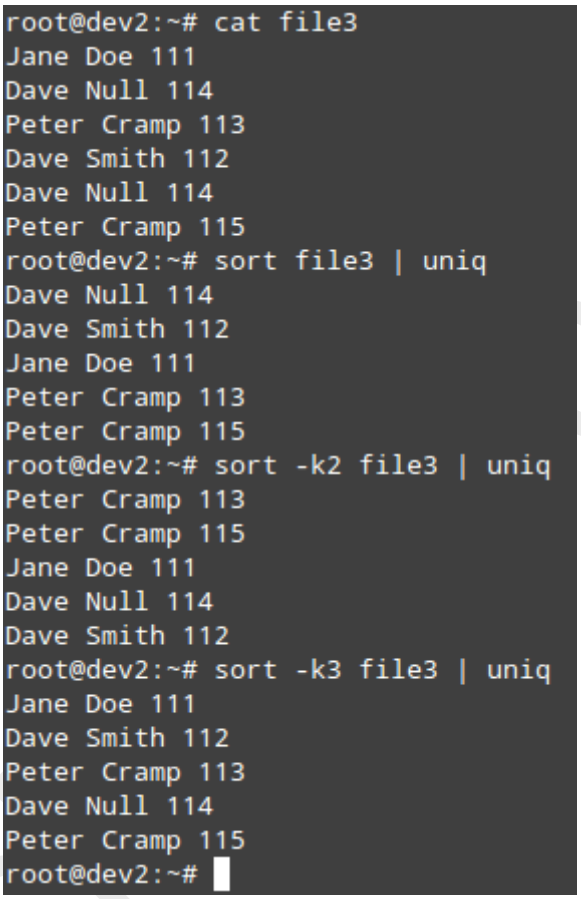


Figure 10: Using sort and uniq

Extracting text with cut

The **cut** command extracts portions of input lines (from stdin or files) and displays the result on standard output, based on number of bytes (-b), characters (-c), or fields (-f).

When using cut based on fields, the default field separator is a tab, but a different separator can be specified by using the -d option (see Fig. 11).

```
cut -d: -f1,3 /etc/passwd # Extract specific fields: 1 and 3 in this case
cut -d: -f2-4 /etc/passwd # Extract range of fields: 2 through 4 in this example
```

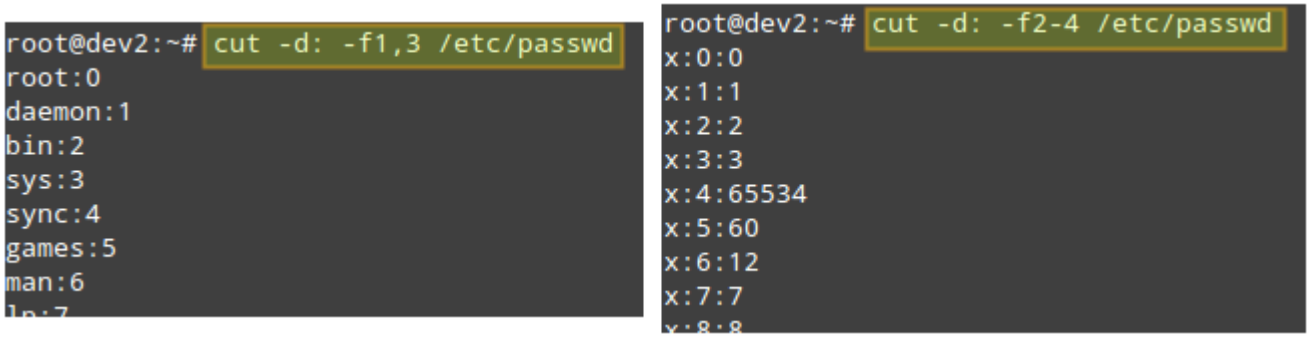


Figure 11: Extracting fields using cut

Note that the output of the two examples above was truncated for brevity.

Reformatting files with fmt

fmt is used to "clean up" files with a great amount of content or lines, or with varying degrees of indentation. The new paragraph formatting defaults to no more than 75 characters wide. You can change this with the **-w** (width) option, which set the line length to the specified number of characters.

For example, let's see what happens when we use **fmt** to display the /etc/passwd file setting the width of each line to 100 characters. Once again, the output shown in Fig. 12 has been truncated for brevity.

```
fmt -w100 /etc/passwd
```

Figure 12: Reformatting plain text files

Formatting content for printing with pr

pr paginates and displays in columns one or more files for printing. In other words, **pr** formats a file to make it look better when printed. For example, the following command

```
ls -a /etc | pr -n --columns=3 -h "Files in /etc"
```

shows a listing of all the files found in /etc in a printer-friendly format (3 columns) with a custom header (indicated by the -h option), and numbered lines (-n). See Figure 13 for more details.

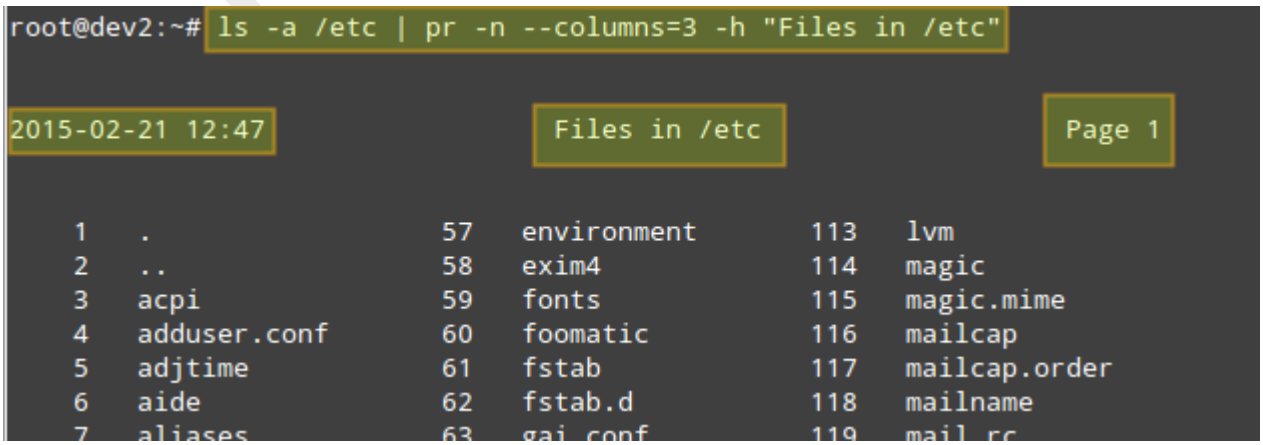


Figure 13: Formatting content for printing

Summary

In this article we have discussed how to enter and execute commands with the correct syntax in a shell prompt or terminal, and explained how to find, inspect, and use system documentation. As simple as it seems, it's a large first step in your way to becoming a RHCSA.

If you would like to add other commands that you use on a periodic basis and that have proven useful to fulfill your daily responsibilities, feel free to share them with the world by using the comment form below. Questions are also welcome. We look forward to hearing from you!

Chapter 2: File and directory management

In this article we will review some essential skills that are required to be used in the day-to-day tasks of a system administrator.

Create, delete, copy, and move files and directories

File and directory management is a critical competence that every system administrator should possess. This includes the ability to create / delete text files from scratch (the core of each program's configuration) and directories (where you will organize files and other directories), and to find out the type of existing files.

The **touch** command can be used not only to create empty files, but also to update the access and modification times of existing files (see Fig. 1):

```
[root@dodev01 ~]# touch testfile
[root@dodev01 ~]# ls -l
total 0
-rw-r--r-- 1 root root 0 Mar  9 18:20 testfile
[root@dodev01 ~]# echo "Adding some content..." > testfile
[root@dodev01 ~]# ls -l
total 4
-rw-r--r-- 1 root root 23 Mar  9 18:21 testfile
[root@dodev01 ~]# touch testfile
[root@dodev01 ~]# ls -l
total 4
-rw-r--r-- 1 root root 23 Mar  9 18:22 testfile
[root@dodev01 ~]#
```

Figure 1: Using touch to create empty files and to update access times

You can use **file [filename]** to determine a file's type (this will come in handy before launching your preferred text editor to edit it), as shown in Fig. 2:

```
[root@dodev01 ~]# file testfile
testfile: ASCII text
[root@dodev01 ~]#
```

Figure 2: Determining a file's type

and **rm [filename]** to delete it (see Fig. 3):

```
[root@dodev01 ~]# ls
testfile
[root@dodev01 ~]# rm testfile
rm: remove regular file 'testfile'? y
[root@dodev01 ~]# ls
[root@dodev01 ~]#
```

Figure 3: Deleting files

As for directories, you can create directories inside existing paths with **mkdir [directory]** or create a full path with **mkdir -p [/full/path/to/directory]**. Please refer to Fig. 4 for more details.

```
[root@dodev01 ~]# ls
[root@dodev01 ~]# mkdir dir1
[root@dodev01 ~]# ls
dir1
[root@dodev01 ~]# mkdir dir1/dir2/dir3
mkdir: cannot create directory ‘dir1/dir2/dir3’: No such file or directory
[root@dodev01 ~]# mkdir -p dir1/dir2/dir3
[root@dodev01 ~]# ls -lR
.:
total 4
drwxr-xr-x 3 root root 4096 Mar  9 18:40 dir1

./dir1:
total 4
drwxr-xr-x 3 root root 4096 Mar  9 18:40 dir2

./dir1/dir2:
total 4
drwxr-xr-x 2 root root 4096 Mar  9 18:40 dir3

./dir1/dir2/dir3:
total 0
[root@dodev01 ~]#
```

Figure 4: Creating directories and directory structure

When it comes to removing directories, you need to make sure that they’re empty before issuing the rmdir **[directory]** command, or use the more powerful (handle with care!) **rm -rf [directory]**. This last option will force remove recursively the **[directory]** and all its contents - so use it at your own risk.

Input and output redirection and pipelining

The command line environment provides two very useful features that allows to redirect the input and output of commands from and to files, and to send the output of a command to another, called **redirection** and **pipelining**, respectively.

To understand those two important concepts, we must first understand the three most important types of I/O (Input and Output) streams (or sequences) of characters, which are in fact special files, in the *nix sense of the word:

- **Standard input** (aka stdin) is by default attached to the keyboard. In other words, the keyboard is the standard input device to enter commands to the command line.
- **Standard output** (aka stdout) is by default attached to the screen, the device that “receives” the output of commands and display them on the screen.
- **Standard error** (aka stderr), is where the status messages of a command is sent to by default, which is also the screen .

In the following example, the output of **ls /var** is sent to stdout (the screen), as well as the result of ls /tecmint. But in the latter case, it is stderr that is shown (see Fig. 5):

```
[root@dodev01 ~]# ls /var
adm  cache  crash  db  empty  ftp  games  gopher  kerberos  lib  local  lock  log  mail
[root@dodev01 ~]# ls /tecmint
ls: cannot access /tecmint: No such file or directory
[root@dodev01 ~]#
```

Figure 5: Viewing file descriptors

To more easily identify these special files, they are each assigned a file descriptor, an abstract representation that is used to access them. The essential thing to understand is that these files, just like others, can be redirected. What this means is that you can capture the output from a file or script and send it as input to another file, command, or script. This will allow you to store on disk, for example, the output of commands for later processing or analysis.

To redirect stdin (fd 0), stdout (fd 1), or stderr (fd 2), the following operators are available (see Table 1):

Redirection operator	Effect
>	Redirects standard output to a file containing standard output. If the destination file exists, it will be overwritten.
>>	Appends standard output to a file.
2>	Redirects standard error to a file containing standard output. If the destination file exists, it will be overwritten.
2>>	Appends standard error to the existing file.
&>	Redirects both standard output and standard error to a file; if the specified file exists, it will be overwritten.
<	Uses the specified file as standard input.
<>	The specified file is used for both standard input and standard output.

Table 1: Redirection operators

As opposed to redirection, pipelining is performed by adding a vertical bar (|) after a command and before another one.

Remember:

- Redirection is used to send the output of a command to a file, or to send a file as input to a command.
- Pipelining is used to send the output of a command to another command as input.

Example 1: Redirecting the output of a command to a file

There will be times when you will need to iterate over a list of files. To do that, you can first save that list to a file and then read that file line by line. While it is true that you can iterate over the output of **ls** directly, this example serves to illustrate redirection (see Fig. 6).

```
ls -1 /var/mail > mail.txt
```

```
[root@dodev01 ~]# ls -1 /var/mail > mail.txt
[root@dodev01 ~]# cat mail.txt
dave
gacanepa
phil
[root@dodev01 ~]#
```

Figure 6: Redirecting output to a file

Example 2: Redirecting both stdout and stderr to /dev/null

In case we want to prevent both stdout and stderr to be displayed on the screen, we can redirect both file descriptors to /dev/null. Note how the output changes when the redirection is implemented for the same command (see Fig. 6):

```
[root@dodev01 ~]# ls /var /tecmint
ls: cannot access /tecmint: No such file or directory
/var:
adm  cache  crash  db  empty  ftp  games  gopher  kerberos  lib  loca
[root@dodev01 ~]# ls /var /tecmint &> /dev/null
[root@dodev01 ~]#
```

Figure 6: Redirecting to /dev/null

Example 3: Using a file as input to a command

While the classic syntax of the cat command is as follows:

```
cat [file(s)]
```

You can also send a file as input, using the correct redirection operator (see Fig. 7):

```
cat < mail.txt
```

```
[root@dodev01 ~]# cat < mail.txt
dave
gacanepa
phil
[root@dodev01 ~]#
```

Figure 7: Using a file as input to a command

Example 4: Sending the output of a command as input to another

If you have a large directory or process listing and want to be able to locate a certain file or process at a glance, you will want to pipeline the listing to grep. Note that we use to pipelines in the following example. The first one looks for the required keyword, while the second one will eliminate the actual grep command from the results. This example lists all the processes associated with the apache user (see Fig. 8):

```
ps -ef | grep apache | grep -v grep
```

```
[root@dodev01 ~]# ps -ef | grep apache | grep -v grep
apache   21004   671  0 Mar01 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   21666   671  0 Mar02 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22332   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22341   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22346   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22348   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22349   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22350   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22351   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache   22352   671  0 Mar03 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
[root@dodev01 ~]#
```

Figure 8: Using pipelining

Archiving, compressing, unpacking, and uncompressing files

If you need to transport, backup, or send via email a group of files, you will use an archiving (or grouping) tool such as **tar**, typically used with a compression utility like gzip, bzip2, or xz. Your choice of a compression tool will be likely defined by the compression speed and rate of each one. Of these three compression tools, gzip is the oldest and provides the least compression, bzip2 provides improved compression, and xz is the newest and provides the best compression. Typically, files compressed with these utilities have .gz, .bz2, or .xz extensions, respectively.

Table 2 shows the most commonly used tar commands and operation modifiers, respectively. You can use one or combine more at the same time. Note that the descriptions have been taken from **man tar**.

Command	Abbreviation	Description
--create	c	Creates a tar archive
--concatenate	A	Appends tar files to an archive
--append	r	Appends non-tar files to an archive
--update	u	Appends files that are newer than those in an archive
--diff or --compare	d	Compares an archive to files on disk
--list	t	Lists the contents of a tarball
--extract or --get	x	Extracts files from an archive
--directory dir	C	Changes to directory dir before performing operations
--same-permissions and --same-owner	p	Preserves permissions and ownership information, respectively.

--verbose	v	Lists all files as they are read or extracted; if combined with --list, it also displays file sizes, ownership, and timestamps
--exclude file	---	Excludes **file** from the archive. In this case, file can be an actual file or a pattern.
--gzip or --gunzip	z	Compresses an archive through gzip
--bzip2	j	Compresses an archive through bzip2
--xz	J	Compresses an archive through xz

Table 2: Common tar options

Example 5: Creating a tarball and then compressing it using the three compression utilities

You may want to compare the effectiveness of each tool before deciding to use one or another. Note that while compressing small files, or a few files, the results may not show much differences, but may give you a glimpse of what they have to offer (see Fig. 9).

```
tar cf ApacheLogs-$(date +%Y%m%d).tar /var/log/httpd/* # Create an ordinary tarball

tar czf ApacheLogs-$(date +%Y%m%d).tar.gz /var/log/httpd/* # Create a tarball and compress with gzip

tar cjf ApacheLogs-$(date +%Y%m%d).tar.bz2 /var/log/httpd/* # Create a tarball and compress with bzip2

tar cJf ApacheLogs-$(date +%Y%m%d).tar.xz /var/log/httpd/* # Create a tarball and compress with xz
```

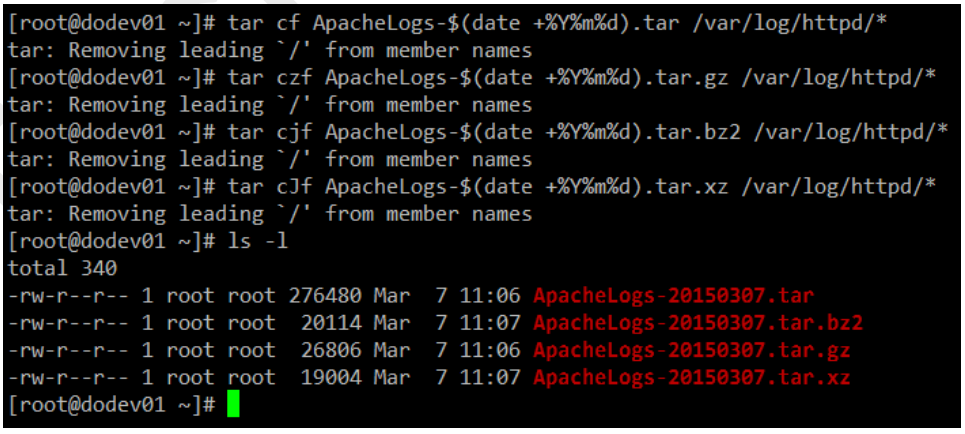


Figure 9: Using different compression methods

Example 6: Preserving original permissions and ownership while archiving and when restoring

If you are creating backups from users' home directories, you will want to store the individual files with the original permissions and ownership instead of changing them to that of the user account or daemon

performing the backup. The following example preserves these attributes while taking the backup of the contents in the /var/log/httpd directory:

```
tar cJf ApacheLogs-$(date +%Y%m%d).tar.xz /var/log/httpd/* --same-permissions --same-owner
```

Create hard and soft links

In Linux, there are two types of links to files: hard links and soft (aka symbolic) links. Since a hard link represents another name for an existing file and is identified by the same inode, it then points to the actual data, as opposed to symbolic links, which point to filenames instead.

In addition, hard links do not occupy space on disk, while symbolic links do take a small amount of space to store the text of the link itself. The downside of hard links is that they can only be used to reference files within the filesystem where they are located because inodes are unique inside a filesystem. Symbolic links save the day, in that they point to another file or directory by name rather than by inode, and therefore can cross filesystem boundaries.

The basic syntax to create links is similar in both cases:

```
ln TARGET LINK_NAME # Hard link named LINK_NAME to file named TARGET
ln -s TARGET LINK_NAME # Soft link named LINK_NAME to file named TARGET
```

Example 7: Creating hard and soft links

There is no better way to visualize the relation between a file and a hard or symbolic link that point to it, than to create those links. In the following screenshot you will see that the file and the hard link that points to it share the same inode and both are identified by **the same disk usage** of 466 bytes. On the other hand, creating a hard link results in an extra disk usage of 5 bytes. Not that you're going to run out of storage capacity, but this example is enough to illustrate the difference between a hard link and a soft link (see Fig. 10).

```
[root@dodev01 ~]# ls -li
total 4
913936 -rw-r--r-- 1 root root 466 Mar  9 16:16 rhcsa
[root@dodev01 ~]# ln rhcsa rhcsa_hardlink
[root@dodev01 ~]# ln -s rhcsa rhcsa_softlink
[root@dodev01 ~]# ls -li
total 8
913936 -rw-r--r-- 2 root root 466 Mar  9 16:16 rhcsa
913936 -rw-r--r-- 2 root root 466 Mar  9 16:16 rhcsa_hardlink
913933 lrwxrwxrwx 1 root root   5 Mar  9 16:16 rhcsa_softlink -> rhcsa
[root@dodev01 ~]#
```

Figure 10: Creating hard and soft links (symlinks)

A typical usage of symbolic links is to reference a versioned file in a Linux system. Suppose there are several programs that need access to file **fooX.Y**, which is subject to frequent version updates (think of a library, for example). Instead of updating every single reference to **fooX.Y** every time there's a version update, it is wiser, safer, and faster, to have programs look to a symbolic link named just foo, which in turn points to the actual fooX.Y. Thus, when X and Y change, you only need to edit the symbolic link foo with a new destination name instead of tracking every usage of the destination file and updating it.

Summary

In this article we have reviewed some essential file and directory management skills that must be a part of every system administrator's toolset. Make sure to review other parts of this series as well in order to integrate these topics with the content covered in this tutorial.

Feel free to let us know if you have any questions or comments. We are always more than glad to hear from our readers.

Chapter 3: Managing user accounts

Managing a RHEL 7 server, as it is the case with any other Linux server, will require that you know how to add, edit, suspend, or delete user accounts, and grant users the necessary permissions to files, directories, and other system resources to perform their assigned tasks.

Managing user accounts

To add a new user account to a RHEL 7 server, you can run either of the following two commands as root:

```
adduser [new_account]
useradd [new_account]
```

When a new user account is added to the system, by default the following operations are performed:

1) His/her home directory is created (/home/username unless specified otherwise).

2) The following hidden files are copied into the user's home directory, and will be used to provide environment variables for his/her user session. You can explore each of them for further details.

- **.bash_logout**
- **.bash_profile**
- **.bashrc**

3) A mail spool is created for the user.

4) A group is created and given the same name as the new user account.

The full account information is stored in the /etc/passwd file. This file contains a record per system user account and has the following format (fields are delimited by a colon):

```
[username]:[x]:[UID]:[GID]:[Comment]:[Home directory]:[Default shell]
```

- Fields **[username]** and **[Comment]** are self explanatory.
- The **x** in the second field indicates that the account is protected by a shadowed password (in /etc/shadow), which is needed to logon as **[username]**.
- The **[UID]** and **[GID]** fields are integers that represent the **U**ser **ID**entification and the primary **G**roup **ID**entification to which **[username]** belongs, respectively.

Finally,

- the **[Home directory]** indicates the absolute path to **[username]**'s home directory, and
- **[Default shell]** is the shell that is assigned to this user when he or she logins the system.

Group information is stored in /etc/group. Each line follows this pattern:

```
[Group name]:[Group password]:[GID]:[Group members]
```

where

- **[Group name]** is the name of group.
- an **x** in **[Group password]** indicates group passwords are not being used.
- **[GID]**: same as in /etc/passwd
- **[Group members]**: a comma separated list of users who are members of **[Group name]**.

After adding an account, you can edit the user's account information using the **usermod** command, whose basic syntax is:

```
usermod [options] [username]
```

Example 1: Setting the expiry date for an account

If you work for a company that has some kind of policy to enable account for a certain interval of time, or if you want to grant access to a limited period of time, you can use the --expiredate flag followed by a date in YYYY-MM-DD format. To verify that the change has been applied, you can compare the output of

```
chage -l [username]
```

before and after updating the account expiry date, as shown in the following image (see Fig. 1):

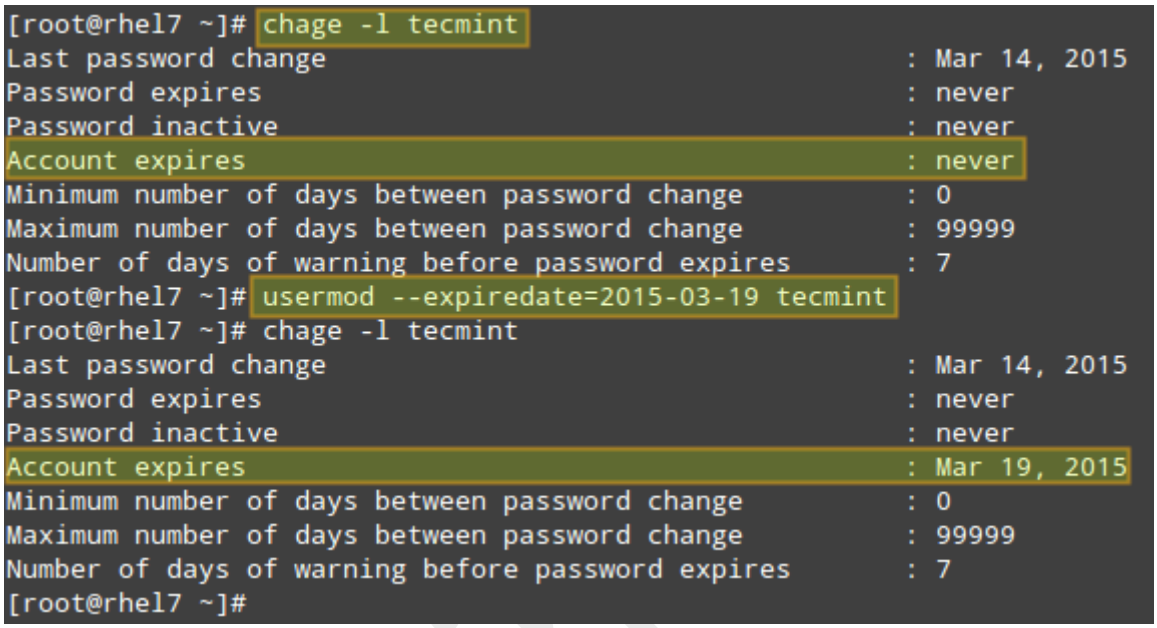


Figure 1: Setting the expiry date of an account

Example 2: Adding the user to supplementary groups

Besides the primary group that is created when a new user account is added to the system, a user can be added to supplementary groups using the combined **-aG**, or **--append --groups** options, followed by a comma separated list of groups.

Example 3: Changing the default location of the user's home directory and / or changing its shell

If for some reason you need to change the default location of the user's home directory (other than /home/username), you will need to use the -d, or --home options, followed by the absolute path to the new home directory.

If a user wants to use another shell other than bash (for example, sh), which gets assigned by default, use **usermod** with the **--shell** flag, followed by the path to the new shell.

Example 4: Displaying the groups an user is a member of

After adding the user to a supplementary group, you can verify that it now actually belongs to such group(s):

```
groups [username]
id [username]
```

The following image (Fig. 2) depicts Examples 2 through 4:

```
[root@rhel7 ~]# usermod --append --groups gacanepa,users --home /tmp --shell /bin/sh tecmint
[root@rhel7 ~]# grep -i tecmint /etc/passwd
tecmint:x:1001:1001::/tmp:/bin/sh
[root@rhel7 ~]# groups tecmint
tecmint : tecmint users gacanepa
[root@rhel7 ~]# id tecmint
uid=1001(tecmint) gid=1001(tecmint) groups=1001(tecmint),100(users),1000(gacanepa)
[root@rhel7 ~]#
```

Figure 2: Changing user account attributes

In the example above:

```
usermod --append --groups gacanepa,users --home /tmp --shell /bin/sh
tecmint
```

To remove a user from a group, omit the --append switch in the command above and list the groups you want the user to belong to following the --groups flag.

Example 5: Disabling account by locking password

To disable an account, you will need to use either the -l (lowercase L) or the --lock option to lock a user's password. This will prevent the user from being able to log on.

Example 6: Unlocking password

When you need to re-enable the user so that he can log on to the server again, use the **-u** or the **--unlock** options to unlock a user's password that was previously blocked, as explained in Example 5 above.

```
usermod --unlock tecmint
```

The following image illustrates Examples 5 and 6:

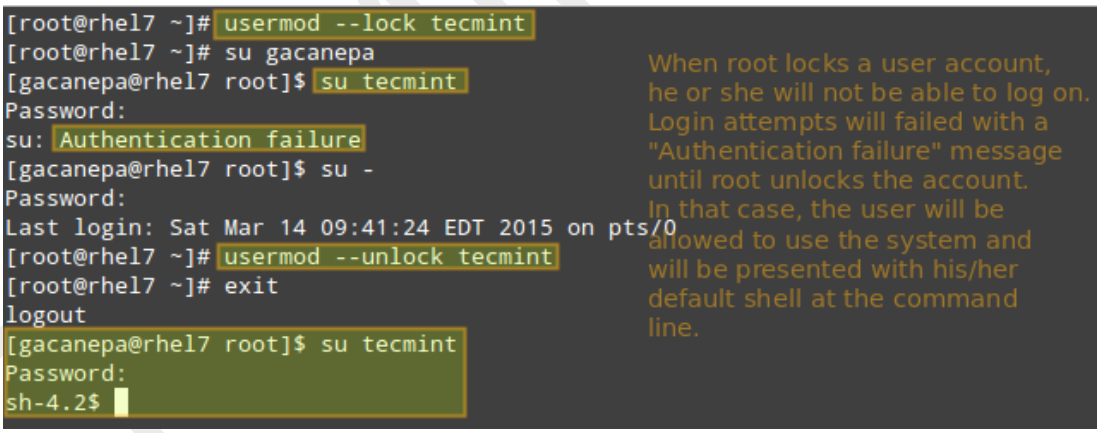


Figure 3: Locking and unlocking user accounts

Example 7: Deleting a group or an user account

You can delete a group with the following command:

Delete a group:

```
groupdel [group_name] # Delete a group
```

Delete a user account along with his/her home directory and mail spool:

```
userdel -r [user_name] # Remove user_name from the system
```

If there are files owned by **group_name**, they will not be deleted, but the group owner will be set to the GID of the group that was deleted.

Listing, setting, and changing standard ugo/rwx permissions

The well-known **ls** command is one of the best friends of any system administrator. When used with the **-l** flag, this tool allows you to view a list a directory's contents in long (or detailed) format. However, this command can also be applied to a single file. Either way, the first 10 characters in the output of **ls -l** represent each file's attributes.

The first char of this 10-character sequence is used to indicate the file type:

- - (hyphen): a regular file
- **d**: a directory
- **l**: a symbolic link
- **c**: a character device (which treats data as a stream of bytes, i.e. a terminal)
- **b**: a block device (which handles data in blocks, i.e. storage devices)

The next nine characters of the file attributes, divided in groups of three from left to right, are called the **file mode** and indicate the read (r), write(w), and execute (x) permissions granted to the file's owner, the file's group owner, and the rest of the users (commonly referred to as "the world"), respectively.

While the read permission on a file allows the same to be opened and read, the same permission on a directory allows its contents to be listed if the execute permission is also set. In addition, the execute permission in a file allows it to be handled as a program and run.

File permissions are changed with the **chmod** command, whose basic syntax is as follows:

```
chmod [new_mode] file
```

where **new_mode** is either an octal number or an expression that specifies the new permissions. In time, and with practice, you will be able to decide which method to change a file mode works best for you in each case. Or perhaps you already have a preferred way to set a file's permissions - so feel free to use the method that works best for you.

The octal number can be calculated based on its binary equivalent, which is calculated from the desired file permissions for the owner, the group, and the world, as follows:

The presence of a certain permission equals a power of 2 ($r=2^2$, $w=2^1$, $x=2^0$), while its absence equates to 0, as shown in Fig. 4:

r	w	x	r	-	-	r	-	-
4	2	1	4	0	0	4	0	0
4+2+1=7			4+0+0=4			4+0+0=4		

Figure 4: Setting permissions in octal form

To set the file's permissions as indicated above in octal form, type:

```
chmod 744 myfile
```

Please take a minute to compare our previous calculation to the actual output of **ls -l** after changing the file's permissions (see Fig. 5):

```
[root@rhel7 ~]# ls -l myfile
-rw-r--r--. 1 root root 0 Mar 13 20:49 myfile
[root@rhel7 ~]# chmod 744 myfile
[root@rhel7 ~]# ls -l myfile
-rwxr--r--. 1 root root 0 Mar 13 20:49 myfile
[root@rhel7 ~]#
```

Figure 5: Before and after setting permissions

Example 8: Searching for files with 777 permissions

As a security measure, you should make sure that files with 777 permissions (read, write, and execute for everyone) are avoided like the plague under normal circumstances. Although we will explain in a later tutorial how to more effectively locate all the files in your system with a certain permission set, you can -by now- combine **ls** with **grep** to obtain such information. In the following example, we will look for file with 777 permissions in the /etc directory only (see Fig. 6). Note that we will use pipelining as explained in Part 2 of this series:

```
ls -l /etc | grep rwxrwxrwx
```

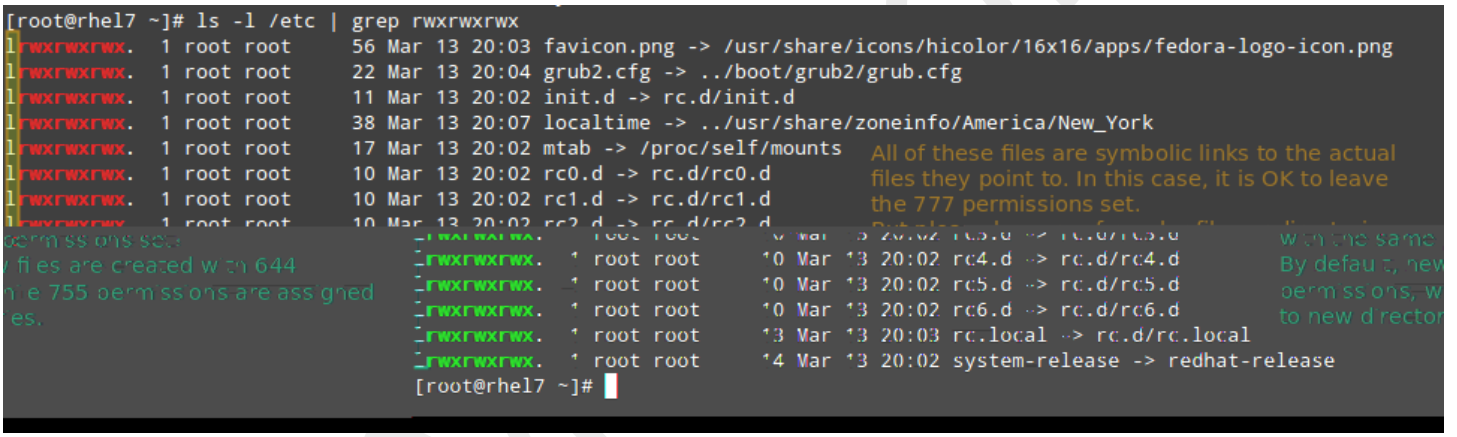


Figure 6: Searching for files with 777 permissions

Example 9: Assigning a specific permission to all users

Shell scripts, along with some binaries that all users should have access to (not just their corresponding owner and group), should have the execute bit set accordingly (please note that we will discuss a special case later):

```
chmod a+x script.sh
```

Note that we can also set a file's mode using an expression that indicates the owner's rights with the letter **u**, the group owner's rights with the letter **g**, and the rest with **o**. All of these rights can be represented at the same time with the letter **a**. Permissions are granted (or revoked) with the + or - signs, respectively (see Fig. 7).

```
[root@rhel7 ~]# ls -l script.sh
-rw-r--r--. 1 root root 0 Mar 13 21:21 script.sh
[root@rhel7 ~]# chmod a+x script.sh
[root@rhel7 ~]# ls -l script.sh
-rwxr-xr-x. 1 root root 0 Mar 13 21:21 script.sh
[root@rhel7 ~]#
```

Figure 7: Assigning individual permissions

A long directory listing also shows the file's owner and its group owner in the first and second columns, respectively. This feature serves as a first-level access control method to files in a system (see Fig. 8)

```
[root@rhel7 ~]# ls -la /home/gacanepa
total 12
drwx------. 2 gacanepa gacanepa  59 Mar 13 20:02 .
drwxr-xr-x. 3 root     root      21 Mar 13 20:07 ..
-rw-r--r--. 1 gacanepa gacanepa  18 Jan 29  2014 .bash_logout
-rw-r--r--. 1 gacanepa gacanepa 193 Jan 29  2014 .bash_profile
-rw-r--r--. 1 gacanepa gacanepa 231 Jan 29  2014 .bashrc
[root@rhel7 ~]#
```

Figure 8: Verifying file's owner and group owner

File ownership is handled by the **chown** command. The owner and the group owner can be changed at the same time or separately. Its basic syntax is as follows:

```
chown user:group file
```

where at least **user** or **group** need to be present, along with the colon.

Alternatively, you can also use chgrp to change a file's group owner only, as follows:

```
chgrp group file
```

Example 10: Cloning permissions from one file to another

If you would like to "clone" ownership from one file to another, you can do so using the **--reference** flag, as follows:

```
chown --reference=ref_file file
```

where the owner and group of **ref_file** will be assigned to **file** as well (see Fig. 9):

```
[root@rhel7 ~]# ls -l testfile
-rw-r--r--. 1 lp mail 0 Mar 13 21:54 testfile
[root@rhel7 ~]# ls -l yetanothertestfile
-rw-r--r--. 1 root root 0 Mar 13 21:54 yetanothertestfile
[root@rhel7 ~]# chown --reference=testfile yetanothertestfile
[root@rhel7 ~]# ls -l yetanothertestfile
-rw-r--r--. 1 lp mail 0 Mar 13 21:54 yetanothertestfile
[root@rhel7 ~]#
```

Figure 9: Cloning permissions from a file to another

Setting up setgid directories for collaboration

Should you need to grant access to all the files owned by a certain group inside a specific directory, you will most likely use the approach of setting the setgid bit for such directory. When the setgid bit is set, the effective GID of the real user becomes that of the group owner. Thus, any user can access a file under the privileges granted to the group owner of such file. In addition, when the setgid bit is set on a directory, newly created files inherit the same group as the directory, and newly created subdirectories will also inherit the setgid bit of the parent directory.

```
chmod g+s [filename]
```

To set the setgid in octal form, prepend the number 2 to the current (or desired) basic permissions.

```
chmod 2755 [directory]
```

Summary

A solid knowledge of user and group management, along with standard and special Linux permissions, when coupled with practice, will allow you to quickly identify and troubleshoot issues with file permissions in your RHEL 7 server. I assure you that as you follow the steps outlined in this article and use the system documentation (as explained in Part 1 of this series) you will master this essential competence of system administration.

Chapter 4: Text editors and regular expressions

Every system administrator has to deal with text files as part of his daily responsibilities. That includes editing existing files (most likely configuration files), or creating new ones. It has been said that if you want to start a holy war in the Linux world, you can ask sysadmins what their favorite text editor is and why. We are not going to do that in this article, but will present a few tips that will be helpful to use two of the most widely used text editors in RHEL 7: nano (due to its simplicity and easiness of use, specially to new users), and vi/m (due to its several features that convert it into more than a simple editor). I am sure that you can find many more reasons to use one or the other, or perhaps some other editor such as emacs or pico. It's entirely up to you.

Editing files with nano

To launch nano, you can either just type nano at the command prompt, optionally followed by a filename (in this case, if the file exists, it will be opened in edition mode). If the file does not exist, or if we omit the filename, nano will also be opened in edition mode but will present a blank screen for us to start typing (see Fig. 1):

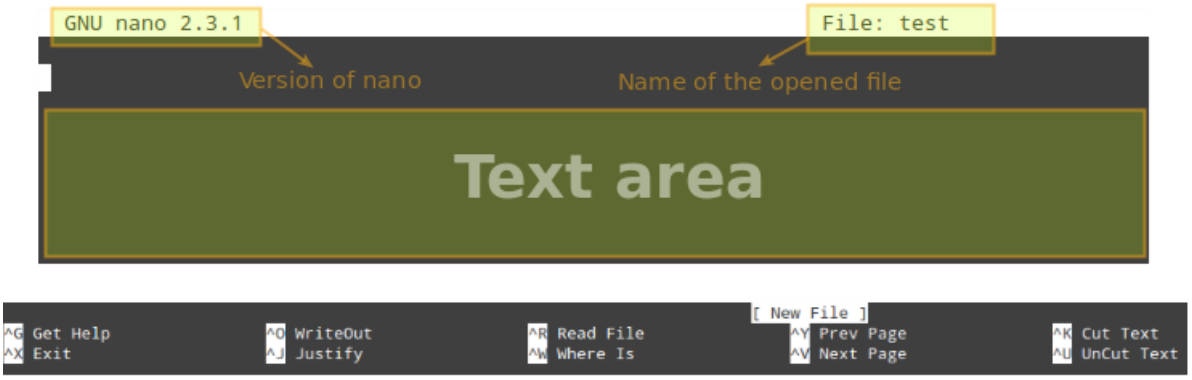


Figure 1: Nano initial screen

As you can see in the previous image, nano displays at the bottom of the screen several functions that are available via the indicated shortcuts (^, aka caret, indicates the Ctrl key). To name a few of them:

- **Ctrl + G**: brings up the help menu with a complete list of functions and descriptions (see Fig. 2):

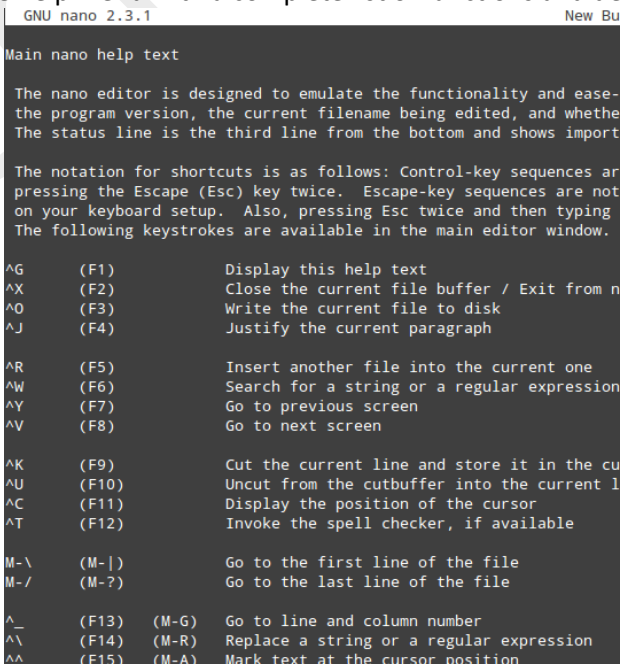


Figure 2: Nano help

- **Ctrl + O**: saves changes made to a file. It will let you save the file with the same name or a different one. Then press Enter to confirm (see Fig. 3)

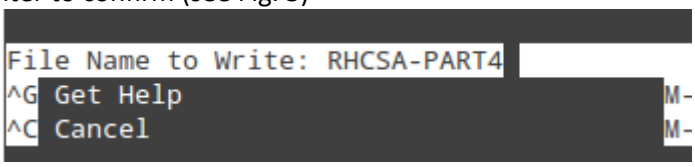


Figure 3: Saving changes to a file

- **Ctrl + X**: exits the current file. If changes have not been saved, they are discarded.
- **Ctrl + R**: lets you choose a file to insert its contents into the present file by specifying a full path (see Fig. 4):

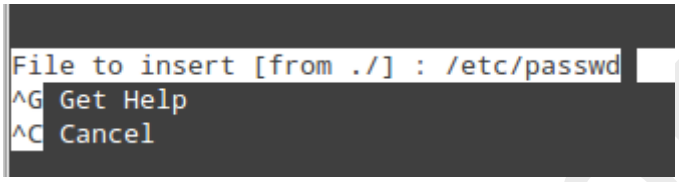


Figure 4: Inserting a file into the current window

will insert the contents of /etc/passwd into the current file.

- **Ctrl + K**: cuts the current line.
- **Ctrl + U**: paste.
- **Ctrl + C**: cancels the current operation and places you at the previous screen.

To easily navigate the opened file, nano provides the following features:

- **Ctrl + F** and **Ctrl + B** move the cursor forward or backward, whereas Ctrl + P and Ctrl + N move it up or down one line at a time, respectively, just like the arrow keys.
- **Ctrl + space** and **Alt + space** move the cursor forward and backward one word at a time.

Finally,

- **Ctrl + _ (underscore)** and then entering **X,Y** will take you precisely to Line **X**, column **Y**, if you want to place the cursor at a specific place in the document (see Fig. 5):

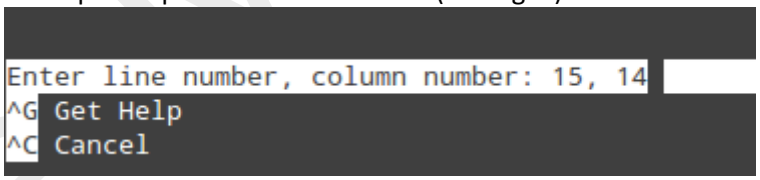


Figure 5: Go to a specific line and column

The example above will take you to line 15, column 14 in the current document.

If you can recall your early Linux days, specially if you came from Windows, you will probably agree that starting off with nano is the best way to go for a new user.

Editing files with vim

Vim is an improved version of vi, a famous text editor in Linux that is available on all POSIX-compliant *nix systems, such as RHEL 7. If you have the chance and can install vim, go ahead; if not, most (if not all) the tips given in this article should also work.

One of vim's distinguishing features is the different modes in which it operates:

- **Command mode** will allow you to browse through the file and enter commands, which are brief and case-sensitive combinations of one or more letters. If you need to repeat one of them a certain number of times, you can prefix it with a number (there are only a few exceptions to this rule). For

example, yy (or **Y**, short for ***yank***) copies the entire current line, whereas 4yy (or 4Y) copies the entire current line along with the next three lines (4 lines in total).

- In **ex mode**, you can manipulate files (including saving a current file and running outside programs or commands). To enter ex mode, we must type a colon (:) starting from command mode (or in other words, Esc + :), directly followed by the name of the ex-mode command that you want to use.
- In **insert mode**, which is accessed by typing the letter i, we simply enter text. Most keystrokes result in text appearing on the screen.
- We can always enter command mode (regardless of the mode we're working on) by pressing the Esc key.

Let's see how we can perform the same operations that we outlined for nano in the previous section, but now with vim. Don't forget to hit the Enter key to confirm the vim command!

To access vim's full manual from the command line, type **:help** while in command mode and then press Enter (see Fig. 6):

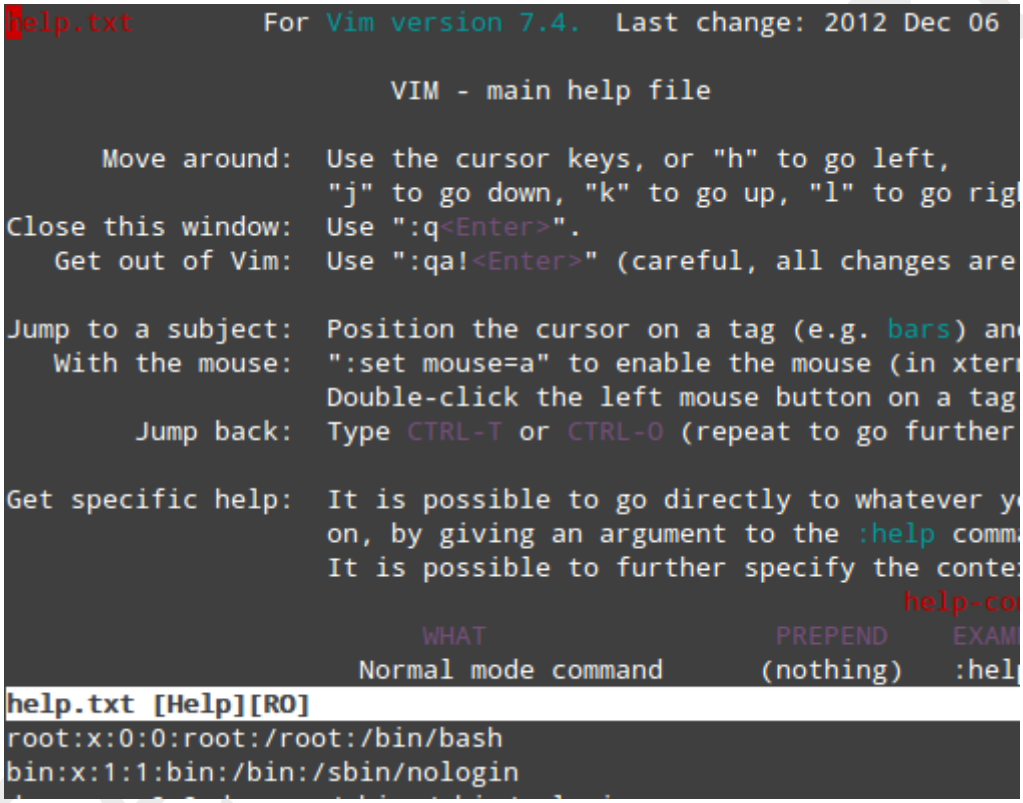


Figure 6: Vim help

The upper section presents an index list of contents, with defined sections dedicated to specific topics about vim. To navigate to a section, place the cursor over it and press **Ctrl +]** (closing square bracket). Note that the bottom section displays the current file.

1) To save changes made to a file, run any of the following commands from command mode and it will do the trick:

- **:wq!**
- **:x!**
- **ZZ** (yes, double Z without the colon at the beginning)

2) To exit discarding changes, use **:q!**. This command will also allow you to exit the help menu described above, and return to the current file in command mode.

3) Cut N number of lines: type **N**dd while in command mode.

4) Copy M number of lines: type **M**yy while in command mode.

5) Paste lines that were previously cutted or copied: press the **P** key while in command mode.

6) To insert the contents of another file into the current one:

:r filename

For example, to insert the contents of /etc/fstab, do (see Fig. 7):

Figure 7: Inserting the contents of an external file into the current file

7) To insert the output of a command into the current document:

:r! command

For example, to insert the date and time in the line below the current position of the cursor (see Fig. 8):

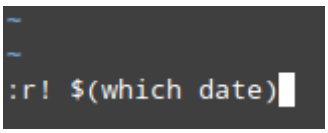


Figure 8: Inserting the output of a command into the current window

Analyzing text with grep and regular expressions

By now you have learned how to create and edit files using nano or vim. Say you become a text editor ninja, so to speak - now what? Among other things, you will also need how to search for regular expressions inside text.

A regular expression (also known as "regex" or "regexp") is a way of identifying a text string or pattern so that a program can compare the pattern against arbitrary text strings. Although the use of regular expressions along with grep would deserve an entire article on its own, let us review the basics here:

1) The simplest regular expression is an alphanumeric string (i.e., the word “svm”) or two (when two are present, you can use the | (OR) operator):

```
grep -Ei 'svm|vmx' /proc/cpuinfo
```

The presence of either of those two strings indicate that your processor supports virtualization (see Fig. 9):

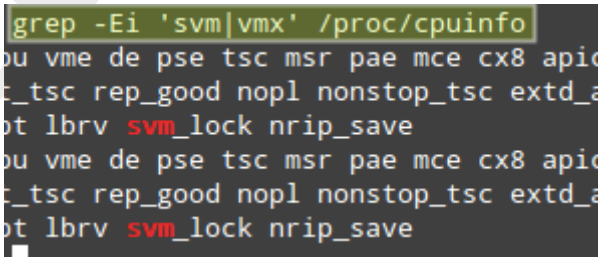


Figure 9: Analyzing text with grep and regexps

2) A second kind of a regular expression is a range list, enclosed between square brackets. For example, **c[aeiou]t** matches the strings **cat**, **cet**, **cit**, **cot**, and **cut**, whereas **[a-z]** and **[0-9]** match any lowercase letter or decimal digit, respectively. If you want to repeat the regular expression X certain number of times, type {X} immediately following the regexp.

For example, let’s extract the UUIDs of storage devices from /etc/fstab (see Fig. 10):

```
[root@rhel7 ~]# grep -Ei '[0-9a-f]{8}-([0-9a-f]{4}-){3}[0-9a-f]{12}' -o /etc/fstab
2d78977c-2a59-4962-bf1f-33d9b5f5076f
[root@rhel7 ~]# grep -i UUID /etc/fstab
UUID=2d78977c-2a59-4962-bf1f-33d9b5f5076f /boot                   xfs     defaults
[root@rhel7 ~]#
```

Figure 10: Using regular expressions

The first expression in brackets [0-9a-f] is used to denote lowercase hexadecimal characters, and {8} is a quantifier that indicates the number of times that the preceding match should be repeated (the first sequence of characters in an UUID is a 8-character long hexadecimal string).

The parentheses, the {4} quantifier, and the hyphen indicate that the next sequence is a 4-character long hexadecimal string, and the quantifier that follows ({3}) denote that the expression should be repeated 3 times. Finally, the last sequence of 12-character long hexadecimal string in the UUID is retrieved with [0-9a-f]{12}, and the -o option prints only the matched (non-empty) parts of the matching line in /etc/fstab.

3) POSIX character classes (see Table 1)

Character class	Matches...
[[:alnum:]]	Any alphanumeric [a-zA-Z0-9] character
[[:alpha:]]	Any alphabetic [a-zA-Z] character
[[:blank:]]	Spaces or tabs
[[:cntrl:]]	Any control characters (ASCII 0 to 32)
[[:digit:]]	Any numeric digits [0-9]
[[:graph:]]	Any visible characters
[[:lower:]]	Any lowercase [a-z] character
[[:print:]]	Any non-control characters
[[:space:]]	Any whitespace
[[:punct:]]	Any punctuation marks
[[:upper:]]	Any uppercase [A-Z] character
[[:xdigit:]]	Any hex digits [0-9a-fA-F]
[:word:]	Any letters, numbers, and underscores [a-zA-Z0-9_]

Table 1: POSIX character classes

For example, we may be interested in finding out what the used UIDs and GIDs (refer to Part 2 of this series to refresh your memory) are for real users that have been added to our system. Thus, we will search for sequences of 4 digits in /etc/passwd (see Fig. 11):

```
grep -Ei [[:digit:]]{4} /etc/passwd
```

```
[root@rhel7 ~]# grep -Ei [[:digit:]]{4} /etc/passwd
gacanepa:x:1000:1000:Gabriel A. Cánepa:/home/gacanepa:/bin/bash
tecmint:x:1001:1001::/tmp:/bin/sh
[root@rhel7 ~]#
```

Figure 11: Using POSIX character classes as regexps

The above example may not be the best case of use of regular expressions in the real world, but it clearly illustrates how to use POSIX character classes to analyze text along with grep.

Summary

In this article we have provided some tips to make the most of nano and vim, two text editors for the command-line users. Both tools are supported by extensive documentation, which you can consult in their respective official web sites.

Chapter 5: Understanding the boot process and process management

We will start this article with an overall and brief revision of what happens since the moment you press the Power button to turn on your RHEL 7 server until you are presented with the login screen in a command line interface. Please note that 1) the same basic principles apply, with perhaps minor modifications, to other Linux distributions as well, and 2) the following description is not intended to represent an exhaustive explanation of the boot process, but only the fundamentals.

The boot process

1) The POST (Power On Self Test) initializes and performs hardware checks.

2) When the POST finishes, the system control is passed to the first stage boot loader, which is stored on either the boot sector of one of the hard disks (for older systems using BIOS and MBR), or a dedicated (U)EFI partition.

3) The first stage boot loader then loads the second stage boot loader, most usually GRUB (GRand Unified Boot Loader), which resides inside /boot, which in turn loads the kernel and the initial RAM–based file system (also known as initramfs, which contains programs and binary files that perform the necessary actions needed to ultimately mount the actual root filesystem).

4) We are presented with a splash screen that allows us to choose an operating system and kernel to boot (see Fig. 1):

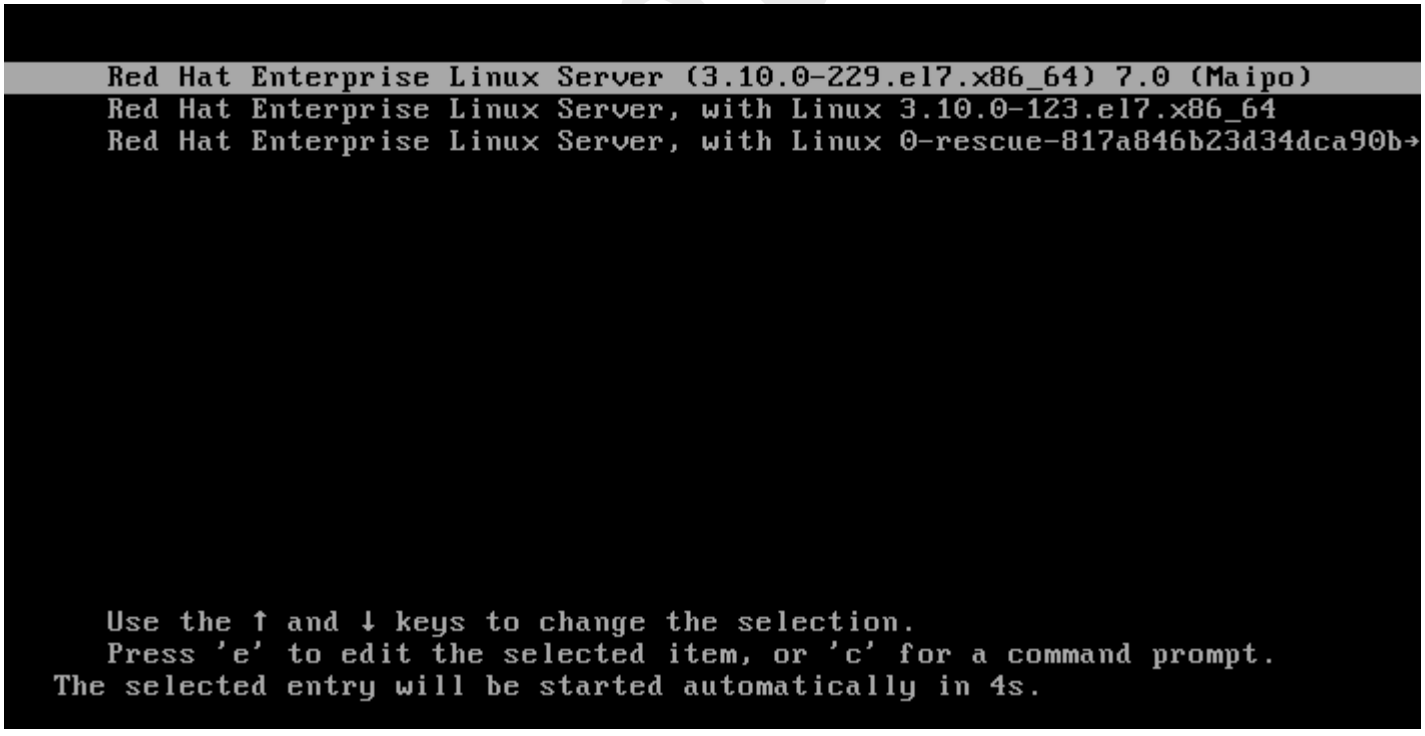


Figure 1: RHEL 7 Initial splash screen

5) The kernel sets up the hardware attached to the system and once the root filesystem has been mounted, launches process with PID 1, which in turn will initialize other processes and present us with a login prompt. Note that if we wish to do so at a later time, we can examine the specifics of this process using the **dmesg** command and filtering its output using the tools that we have explained in previous articles of this series (see Fig. 2).

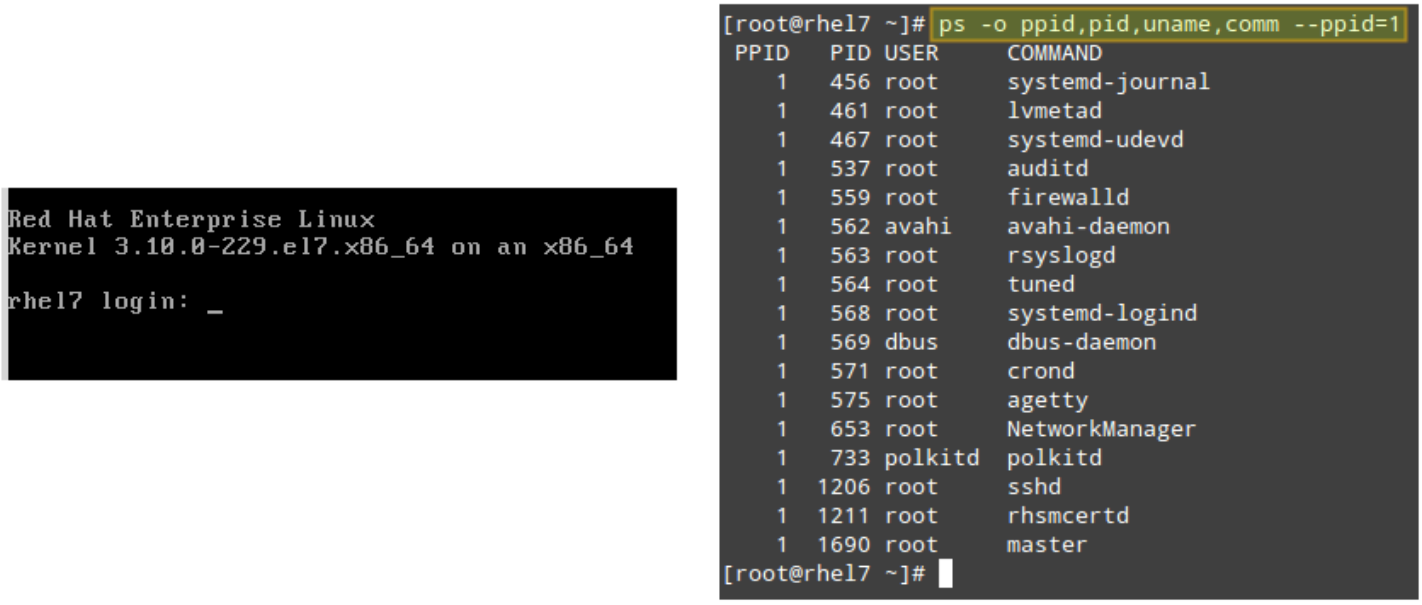


Figure 2: Viewing the child processes of the first process to launch

In the example above, we used the well-known **ps** command to display a list of current processes whose parent process (or in other words, the process that started them) is systemd (the system and service manager that most modern Linux distributions have switched to) during system startup:

```
ps -o ppid,pid,uname,comm --ppid=1
```

Remember that the -o flag (short for --format) allows you to present the output of ps in a customized format to suit your needs using the keywords specified in the **STANDARD FORMAT SPECIFIERS** section in man ps.

Another case in which you will want to define the output of **ps** instead of going with the default is when you need to find processes that are causing a significant CPU and / or memory load, and sort them accordingly (see Fig. 3 for an example):

```
ps aux --sort=+pcpu # Sort by %CPU (ascending)
ps aux --sort=-pcpu # Sort by %CPU (descending)
ps aux --sort=+pmem # Sort by %MEM (ascending)
ps aux --sort=-pmem # Sort by %MEM (descending)
ps aux --sort=+pcpu,-pmem # Combine sort by %CPU (ascending) and %MEM (descending)
```

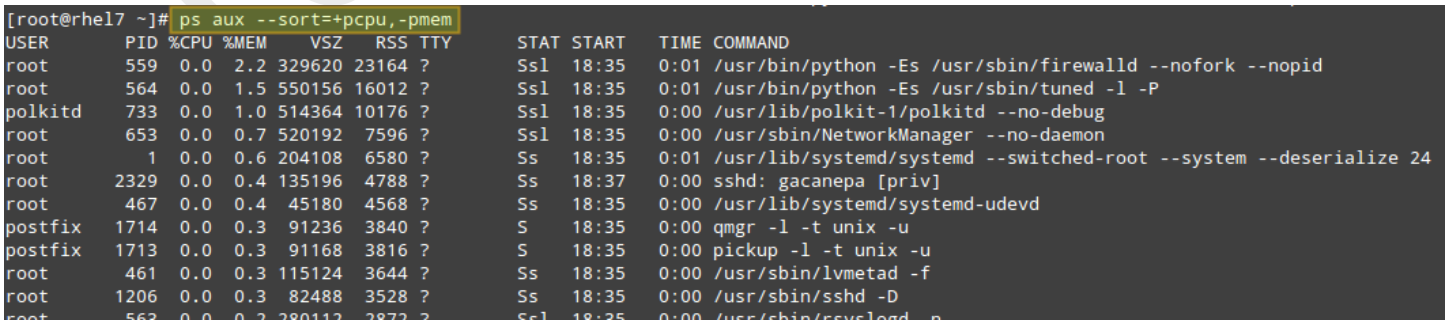


Figure 3: Customizing the output of ps

An introduction to systemd

Few decisions in the Linux world have caused more controversies than the adoption of systemd by major Linux distributions. Systemd's advocates name as its main advantages the following facts:

1) Systemd allows more processing to be done in parallel during system startup (as opposed to older SysVinit, which always tends to be slower because it starts processes one by one, checks if one depends on another, and then waits for daemons to launch so more services can start), and

2) it works as a dynamic resource management in a running system. Thus, services are started when needed (to avoid consuming system resources if they are not being used) instead of being launched without a valid reason during boot.

3) Backwards compatibility with SysVinit scripts.

Systemd is controlled by the systemctl utility. If you come from a SysVinit background, chances are you will be familiar with

a) the **service** tool, which -in those older systems- was used to manage SysVinit scripts, and

b) the **chkconfig** utility, which served the purpose of updating and querying runlevel information for system services.

c) **shutdown**, which you must have used several times to either restart or halt a running system.

Table 1 shows the similarities between the use of these legacy tools and **systemctl**:

Legacy tool	Systemctl equivalent	Description
service name start	`systemctl start name`	Start name (where name is a service)
service name stop	`systemctl stop name`	Stop name
service name condrestart	`systemctl try-restart name`	Restarts name (if it's already running)
service name restart	`systemctl restart name`	Restarts name
service name reload	`systemctl reload name`	Reloads the configuration for name
service name status	`systemctl status name`	Displays the current status of name
service --status-all	`systemctl`	Displays the status of all current services
chkconfig name on	`systemctl enable name`	Enable name to run on startup as specified in the unit file (the file to which the symlink points). The process of enabling or disabling a service to start automatically on boot consists in adding or removing symbolic links inside the /etc/systemd/system directory.
chkconfig name off	`systemctl disable name`	Disables name to run on startup as specified in the unit file (the file to which the symlink points)
chkconfig --list	`systemctl is-enabled`	Verify whether name (a specific service) is currently

name	name	enabled
chkconfig --list	systemctl --type=service	Displays all services and tells whether they are enabled or disabled
shutdown -h now	systemctl poweroff	Power-off the machine (halt)
shutdown -r now	systemctl reboot	Reboot the system

Table 1: Comparison between systemd and init-based system management commands

Systemd also introduced the concepts of **units** (which can be either a service, a mount point, a device, or a network socket) and targets (which is how systemd manages to start several related process at the same time, and can be considered -though not equal- as the equivalent of runlevels in SysVinit-based systems.

Summing up

Other tasks related with process management include, but may not be limited to, the ability to:

1) Adjust the execution priority -as far as the use of system resources is concerned- of a process: this is accomplished through the **renice** utility, which alters the scheduling priority of one or more running processes. In simple terms, the scheduling priority is a feature that allows the kernel (present in versions => 2.6) to allocate system resources as per the assigned execution priority (aka niceness, in a range from -20 through 19) of a given process.

The basic syntax of renice is as follows:

```
renice [-n] priority [-gpu] identifier
```

In the generic command above, the first argument is the priority value to be used, whereas the other argument can be interpreted as process IDs (which is the default setting), process group IDs, user IDs, or user names. A normal user (other than root) can only modify the scheduling priority of a process he or she owns, and only increase the niceness level (which means taking up less system resources). Refer to Fig. 4 for more details.

```
ps -efl | grep top | grep-v grep
```

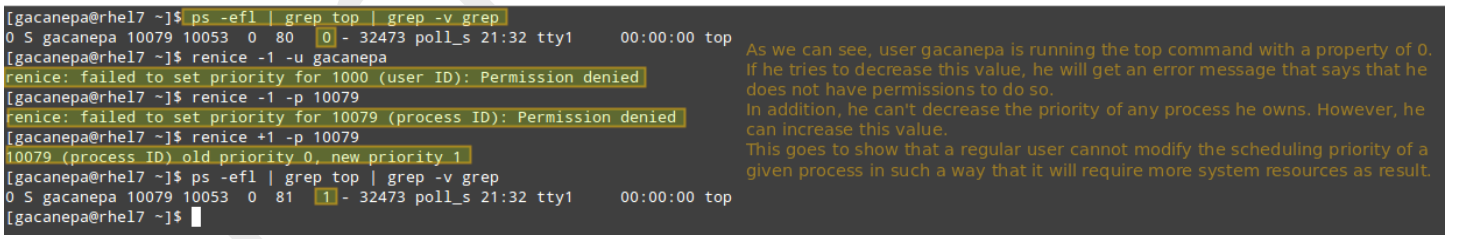


Figure 4: Adjusting niceness of a process

2) Kill (or interrupt the normal execution) of a process as needed: in more precise terms, killing a process entitles sending it a signal to either finish its execution gracefully (SIGTERM=15) or immediately (SIGKILL=9) through the kill or **pkill** commands. The difference between these two tools is that the former is used to terminate a specific process or a process group altogether, while the latter allows you to do the same based on name and other attributes.

In addition, **pkill** comes bundled with **pgrep**, which shows you the PIDs that will be affected should **pkill** be used. For example, before running

```
pkill -u gacanepa
```

it may be useful to view at a glance which are the PIDs owned by gacanepa (see Fig. 5):

```
pgrep -l -u gacanepa
```

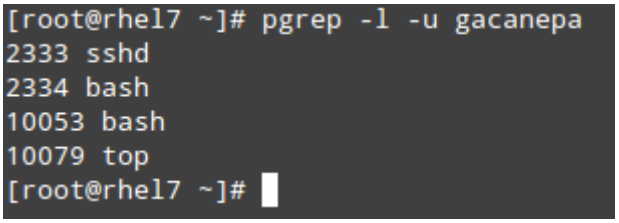


Figure 5: Process owned by a specific user

By default, both kill and pkill send the SIGTERM signal to the process. As we mentioned above, this signal can be ignored (while the process finishes its execution or for good), so when you seriously need to stop a running process with a valid reason, you will need to specify the SIGKILL signal on the command line:

Kill a process or a process group:

```
kill -9 identifier
kill -s SIGNAL identifier
```

Kill a process by name or other attributes

```
pkill -s SIGNAL identifier
```

Summary

In this article we have explained the basics of the boot process in a RHEL 7 system, and analyzed some of the tools that are available to help you with managing processes using common utilities and systemd-specific commands.

Chapter 6: Setting up and configuring system storage

In this article we will discuss how to set up and configure local system storage in Red Hat Enterprise Linux 7 using classic tools and introducing the System Storage Manager (also known as SSM), which greatly simplifies this task. Please note that we will present this topic in this article but will continue its description and usage on the next one (Part 7) due to vastness of the subject.

Creating and modifying partitions

In RHEL 7, **parted** is the default utility to work with partitions, and will allow you to:

- Display the current partition table
- Manipulate (increase or decrease the size of) existing partitions
- Create partitions using free space or additional physical storage devices

It is recommended that before attempting the creation of a new partition or the modification of an existing one, you should ensure that none of the partitions on the device are in use (`umount /dev/partition`), and if you're using part of the device as swap you need to disable it (`swapoff -v /dev/partition`). The easiest way to do this is to boot RHEL in rescue mode using an installation media such as a RHEL 7 installation DVD or USB (Troubleshooting → Rescue a Red Hat Enterprise Linux system) and Select **Skip** when you're prompted to choose an option to mount the existing Linux installation. You will then be presented with a shell prompt (as shown in Fig. 1):

Figure 1: Booting RHEL 7 in rescue mode

To start parted, simply type

```
parted /dev/sdb
```

where **/dev/sdb** is the device where you will create the new partition; next, type **print** to display the current drive's partition table (see Fig. 2):

```
[root@rhel7 ~]# parted /dev/sdb
GNU Parted 3.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Error: /dev/sdb: unrecognised disk label
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
(parted)
```

Figure 2: Printing partition tables

As you can see, in this example we are using a virtual drive of 5 GB. We will now proceed to create a 4 GB primary partition and then format it with the **xfs** filesystem, which is the default in RHEL 7. You can choose from a variety of file systems. You will need to manually create the partition with **mkpart** and then format it with **mkfs.fstype** as usual because mkpart does not support many modern filesystems out-of-the-box.

In the following example (see Fig. 3) we will set a label for the device and then create a primary partition (p) on /dev/sdb, which starts at the 0% percentage of the device and ends at 4000 MB (4 GB):

```
(parted) mklabel msdos
(parted) mkpart
Partition type?  primary/extended? p
File system type?  [ext2]?
Start? 0%
End? 4000
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  4000MB  3999MB  primary

(parted)
```

Figure 3: Creating device labels and primary partitions

Next, we will format the partition as xfs and print the partition table again to verify that changes were applied (refer to Fig. 4):

```
mkfs.xfs /dev/sdb1
parted /dev/sdb print
```

```
[root@rhel7 ~]# mkfs.xfs /dev/sdb1
meta-data=/dev/sdb1              isize=256    agcount=4, agsize=244096 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=976384, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
[root@rhel7 ~]# parted /dev/sdb print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  4000MB  3999MB  primary  xfs

[root@rhel7 ~]#
```

Figure 4: Verifying changes

For older filesystems, you could use the **resize** command inside parted to resize a partition. Unfortunately, this only applies to **ext2**, **fat16**, **fat32**, **hfs**, **linux-swap**, and **reiserfs** (if **libreiserfs** is installed). Thus, the only way to resize a partition is by deleting it and creating it again (so make sure you have a good backup of your data!). No wonder the default partitioning scheme in RHEL 7 is based on LVM.

To remove a partition with parted (see Fig. 5):

```
parted /dev/sdb rm 1
```

```
[root@rhel7 ~]# parted /dev/sdb print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  4000MB  3999MB  primary  xfs

[root@rhel7 ~]# parted /dev/sdb rm 1
Information: You may need to update /etc/fstab.

[root@rhel7 ~]# parted /dev/sdb print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End  Size  Type  File system  Flags

[root@rhel7 ~]#
```

When you create or remove a partition, you will get a reminder that you may need to update /etc/fstab.

To remove a partition, you will use the rm command of parted followed by the partition number (1 in this case)

Figure 5: Deleting partitions

The Logical Volume Manager

Once a disk has been partitioned, it can be difficult or risky to change the partition sizes. For that reason, if we plan on resizing the partitions on our system, we should consider the possibility of using LVM instead of the classic partitioning system. In LVM, several physical devices can form a volume group that will host a defined number of logical volumes, which can be expanded or reduced without any hassle.

In simple terms, you may find the following diagram (Fig. 6) useful to remember the basic architecture of LVM:

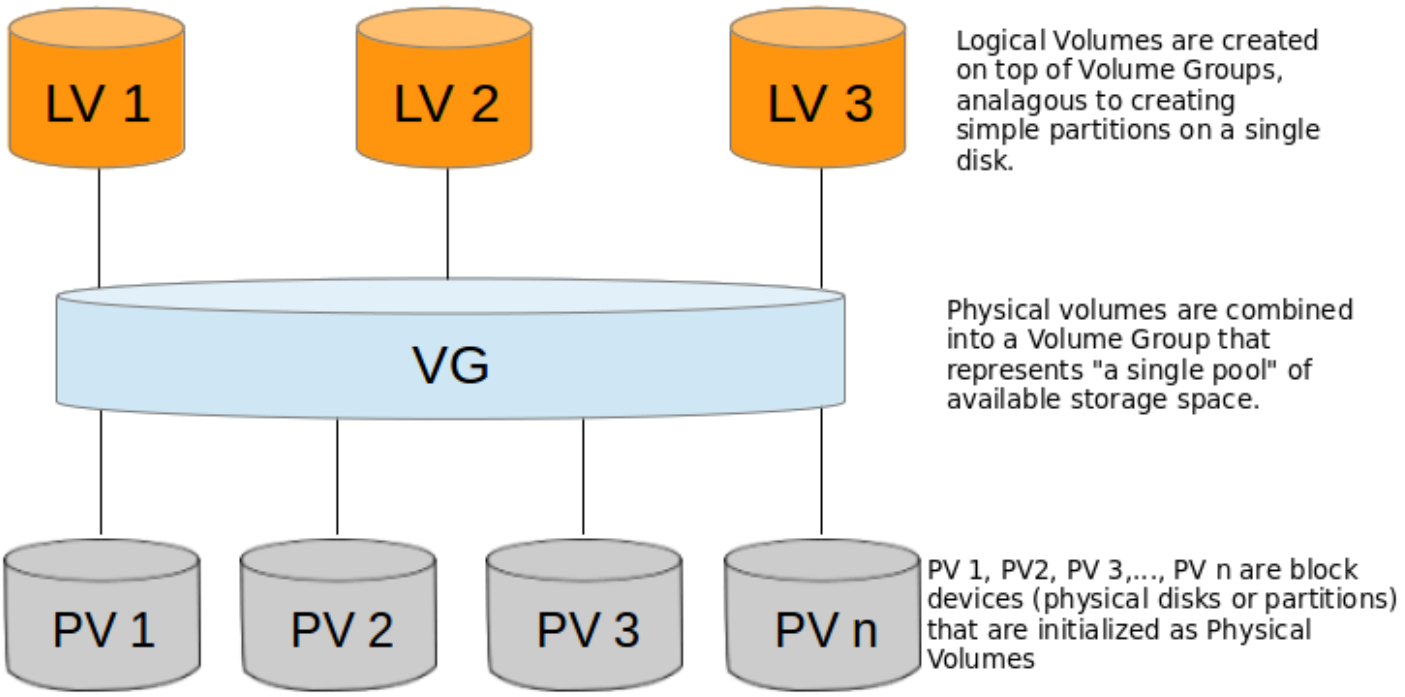


Figure 6: LVM basic architecture

Follow these steps in order to set up LVM using classic volume management tools. Since you can expand this topic reading the LVM series on Tecmint.com, I will only outline the basic steps to set up LVM, and then compare them to implementing the same functionality with **SSM**. Note that we will use the whole disks **/dev/sdb** and **/dev/sdc** as PVs but it's entirely up to you if you want to do the same.

1) Create partitions **/dev/sdb1** and **/dev/sdc1** using 100% of the available disk space in **/dev/sdb** and **/dev/sdc** (as shown in Fig. 7):

```
[root@rhel7 ~]# parted /dev/sdb print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  5369MB  5368MB  primary

[root@rhel7 ~]# parted /dev/sdc print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdc: 5369MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  5369MB  5368MB  primary

[root@rhel7 ~]#
```

Figure 7: Initial partitions

2) Create 2 physical volumes on top of **/dev/sdb1** and **/dev/sdc1**, respectively (see Fig. 8).

```
pvcreate /dev/sdb1
pvcreate /dev/sdc1
```

```
[root@rhel7 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
[root@rhel7 ~]# pvcreate /dev/sdc1
  Physical volume "/dev/sdc1" successfully created
[root@rhel7 ~]#
```

Figure 8: Creating physical volumes

Remember that you can use `pvdisplay /dev/sd{b,c}1` to show information about the newly created PVs.

3) Create a VG on top of the PV that you created in the previous step (see Fig. 9):

```
vgcreate tecmint_vg /dev/sd{b,c}1
```

```
[root@rhel7 ~]# vgcreate tecmint_vg /dev/sd{b,c}1
  Volume group "tecmint_vg" successfully created
[root@rhel7 ~]#
```

Figure 9: Creating a volume group

Remember that you can use `vgdisplay tecmint_vg` to show information about the newly created VG.

4) Create three logical volumes on top of VG tecmint_vg, as follows (see Fig. 10):

- vol01_docs → 3 GB:

```
lvcreate -L 3G -n vol01_docs tecmint_vg
```

- vol02_logs → 1 GB:

```
lvcreate -L 1G -n vol02_logs tecmint_vg
```

- vol03_homes → 6 GB:

```
lvcreate -l 100%FREE -n vol03_homes tecmint_vg
```

```
[root@rhel7 ~]# lvcreate -L 3G -n vol01_docs tecmint_vg
  Logical volume "vol01_docs" created.
[root@rhel7 ~]# lvcreate -L 1G -n vol02_logs tecmint_vg
  Logical volume "vol02_logs" created.
[root@rhel7 ~]# lvcreate -l 100%FREE -n vol03_homes tecmint_vg
  Logical volume "vol03_homes" created.
[root@rhel7 ~]#
```

Figure 10: Creating logical volumes

Remember that you can use `lvdisplay tecmint_vg` to show information about the newly created LVs on top of VG tecmint_vg.

5) Format each of the logical volumes with xfs (**do NOT use xfs if you're planning on shrinking volumes later!**):

```
mkfs.xfs /dev/tecmint_vg/vol01_docs
mkfs.xfs /dev/tecmint_vg/vol02_logs
mkfs.xfs /dev/tecmint_vg/vol03_homes
```

6) Finally, mount them:

```
mount /dev/tecmint_vg/vol01_docs /mnt/docs
mount /dev/tecmint_vg/vol02_logs /mnt/logs
mount /dev/tecmint_vg/vol03_homes /mnt/homes
```

Now we will reverse the LVM implementation and remove the LVs, the VG, and the PVs:

```
lvremove /dev/tecmint_vg/vol01_docs
lvremove /dev/tecmint_vg/vol02_logs
lvremove /dev/tecmint_vg/vol03_homes
vgremove /dev/tecmint_vg
pvremove /dev/sd{b,c}1
```

Now let's install SSM and we will see how to perform the above in ONLY 1 STEP!

```
yum update && yum install system-storage-manager
```

We will use the same names and sizes as before:

```
ssm create -s 3G -n vol01_docs -p tecmint_vg --fstype ext4 /mnt/docs
/dev/sd{b,c}1
ssm create -s 1G -n vol02_logs -p tecmint_vg --fstype ext4 /mnt/logs
/dev/sd{b,c}1
ssm create -n vol03_homes -p tecmint_vg --fstype ext4 /mnt/homes
/dev/sd{b,c}1
```

Yes! SSM will let you 1) initialize block devices as physical volumes, 2) create a volume group, 3) create logical volumes, 4) format LVs, and 5) mount them using only one command.

We can now display the information about PVs, VGs, or LVs, respectively, as follows (see Fig. 11 for further details):

```
ssm list dev
ssm list pool
ssm list vol
```

```
[root@rhel7 ~]# ssm list dev
----------------------------------------------------------
Device        Free      Used      Total  Pool        Mount point
----------------------------------------------------------
/dev/sda                         30.00 GB             PARTITIONED
/dev/sda1                       500.00 MB             /boot
/dev/sda2  0.00 KB  29.51 GB   29.51 GB  rhel
/dev/sdb                          5.00 GB
/dev/sdb1  0.00 KB   5.00 GB    5.00 GB  tecmint_vg
/dev/sdc                          5.00 GB
/dev/sdc1  0.00 KB   5.00 GB    5.00 GB  tecmint_vg
----------------------------------------------------------
[root@rhel7 ~]# ssm list pool
-----------------------------------------------------
Pool        Type  Devices     Free      Used     Total
-----------------------------------------------------
rhel        lvm   1        0.00 KB  29.51 GB  29.51 GB
tecmint_vg  lvm   2        0.00 KB   9.99 GB   9.99 GB
-----------------------------------------------------
[root@rhel7 ~]# ssm list vol
-----------------------------------------------------------------------------------------------------------
Volume                      Pool        Volume size  FS      FS size       Free  Type    Mount point
-----------------------------------------------------------------------------------------------------------
/dev/rhel/root              rhel           27.48 GB  xfs    27.46 GB   26.27 GB  linear  /
/dev/rhel/swap              rhel            2.03 GB                              linear
/dev/tecmint_vg/vol01_docs  tecmint_vg      3.00 GB  ext4    3.00 GB    2.73 GB  linear  /mnt/docs
/dev/tecmint_vg/vol02_logs  tecmint_vg      1.00 GB  ext4    1.00 GB  922.20 MB  linear  /mnt/logs
/dev/tecmint_vg/vol03_homes tecmint_vg      5.99 GB  ext4    5.99 GB    5.45 GB  linear  /mnt/homes
/dev/sda1                                 500.00 MB  xfs   496.67 MB  343.77 MB  part    /boot
-----------------------------------------------------------------------------------------------------------
[root@rhel7 ~]#
```

Figure 11: Listing LVM information using ssm

As we already know, one of the distinguishing features of LVM is the possibility to resize (expand or decrease) logical volumes without downtime.

Say we are running out of space in **vol02_logs** but have plenty of space in **vol03_homes**. We will resize **vol03_homes** to 4 GB and expand **vol02_logs** to use the remaining space:

```
ssm resize -s 4G /dev/tecmint_vg/vol03_homes
```

Run `ssm list pool` again and take note of the free space in **tecmint_vg** (see Fig. 12):

```
-----------------------------------------------------
Pool        Type  Devices     Free      Used     Total
-----------------------------------------------------
rhel        lvm   1        0.00 KB  29.51 GB  29.51 GB
tecmint_vg  lvm   2        1.99 GB   8.00 GB   9.99 GB
-----------------------------------------------------
```

Figure 11: Listing VG information

Then do:

```
ssm resize -s+1.99 /dev/tecmint_vg/vol02_logs
```

Note that the plus sign in front of the **-s** flag indicates that the specified value should be added to the present value.

Removing logical volumes and volume groups is much easier with ssm as well. A simple

```
ssm remove tecmint_vg
```

will return a prompt asking you to confirm the deletion of the VG and the LVs it contains (see Fig. 13):

```
[root@rhel7 ~]# ssm remove tecmint_vg
Do you really want to remove volume group "tecmint_vg" containing 3 logical volumes? [y/n]: y
Do you really want to remove active logical volume vol01_docs? [y/n]: y
  Logical volume "vol01_docs" successfully removed
Do you really want to remove active logical volume vol02_logs? [y/n]: y
  Logical volume "vol02_logs" successfully removed
Do you really want to remove active logical volume vol03_homes? [y/n]: y
  Logical volume "vol03_homes" successfully removed
  Volume group "tecmint_vg" successfully removed
[root@rhel7 ~]#
```

Figure 13: Removing volume groups and logical volumes

Managing encrypted volumes

SSM also provides system administrators with the capability of managing encryption for new or existing volumes. You will need the **cryptsetup** package installed first:

```
yum update && yum install cryptsetup
```

Then issue the following command to create an encrypted volume. You will be prompted to enter a passphrase to maximize security:

```
ssm create -s 3G -n vol01_docs -p tecmint_vg --fstype ext4 --encrypt luks /mnt/docs /dev/sd{b,c}1

ssm create -s 1G -n vol02_logs -p tecmint_vg --fstype ext4 --encrypt luks /mnt/logs /dev/sd{b,c}1

ssm create -n vol03_homes -p tecmint_vg --fstype ext4 --encrypt luks /mnt/homes /dev/sd{b,c}1
```

Our next task consists in adding the corresponding entries in **/etc/fstab** in order for those logical volumes to be available on boot. Rather than using the device identifier (/dev/something) we will use each LV's UUID (so that our devices will still be uniquely identified should we add other logical volumes or devices), which we can find out with the **blkid** utility:

```
blkid -o value UUID /dev/tecmint_vg/vol01_docs

blkid -o value UUID /dev/tecmint_vg/vol02_logs

blkid -o value UUID /dev/tecmint_vg/vol03_homes
```

In our case (see Fig. 14):

```
[root@rhel7 ~]# blkid -o value UUID /dev/tecmint_vg/vol01_docs
ba77d113-f849-4ddf-8048-13860399fca8
crypto_LUKS
[root@rhel7 ~]# blkid -o value UUID /dev/tecmint_vg/vol02_logs
58f89c5a-f694-4443-83d6-2e83878e30e4
crypto_LUKS
[root@rhel7 ~]# blkid -o value UUID /dev/tecmint_vg/vol03_homes
92245af6-3f38-4e07-8dd8-787f4690d7ac
crypto_LUKS
[root@rhel7 ~]#
```

Figure 14: Viewing the UUIDs of each logical volume

Next, create the **/etc/crypttab** file with the following contents (change the UUIDs for the ones that apply to your setup):

```
docs UUID=ba77d113-f849-4ddf-8048-13860399fca8 none
```

```
logs UUID=58f89c5a-f694-4443-83d6-2e83878e30e4 none
homes UUID=92245af6-3f38-4e07-8dd8-787f4690d7ac none
```

And insert the following entries in **/etc/fstab**. Note that **device_name (/dev/mapper/device_name**) is the mapper identifier that appears in the first column of **/etc/crypttab**.

```
# Logical volume vol01_docs:
/dev/mapper/docs      /mnt/docs       ext4  defaults        0     2
# Logical volume vol02_logs
/dev/mapper/logs      /mnt/logs       ext4  defaults        0     2
# Logical volume vol03_homes
/dev/mapper/homes     /mnt/homes      ext4  defaults        0     2
```

Now reboot (**systemctl reboot**) and you will be prompted to enter the passphrase for each LV. Afterwards you can confirm that the mount operation was successful by checking the corresponding mount points (see Fig. 15):

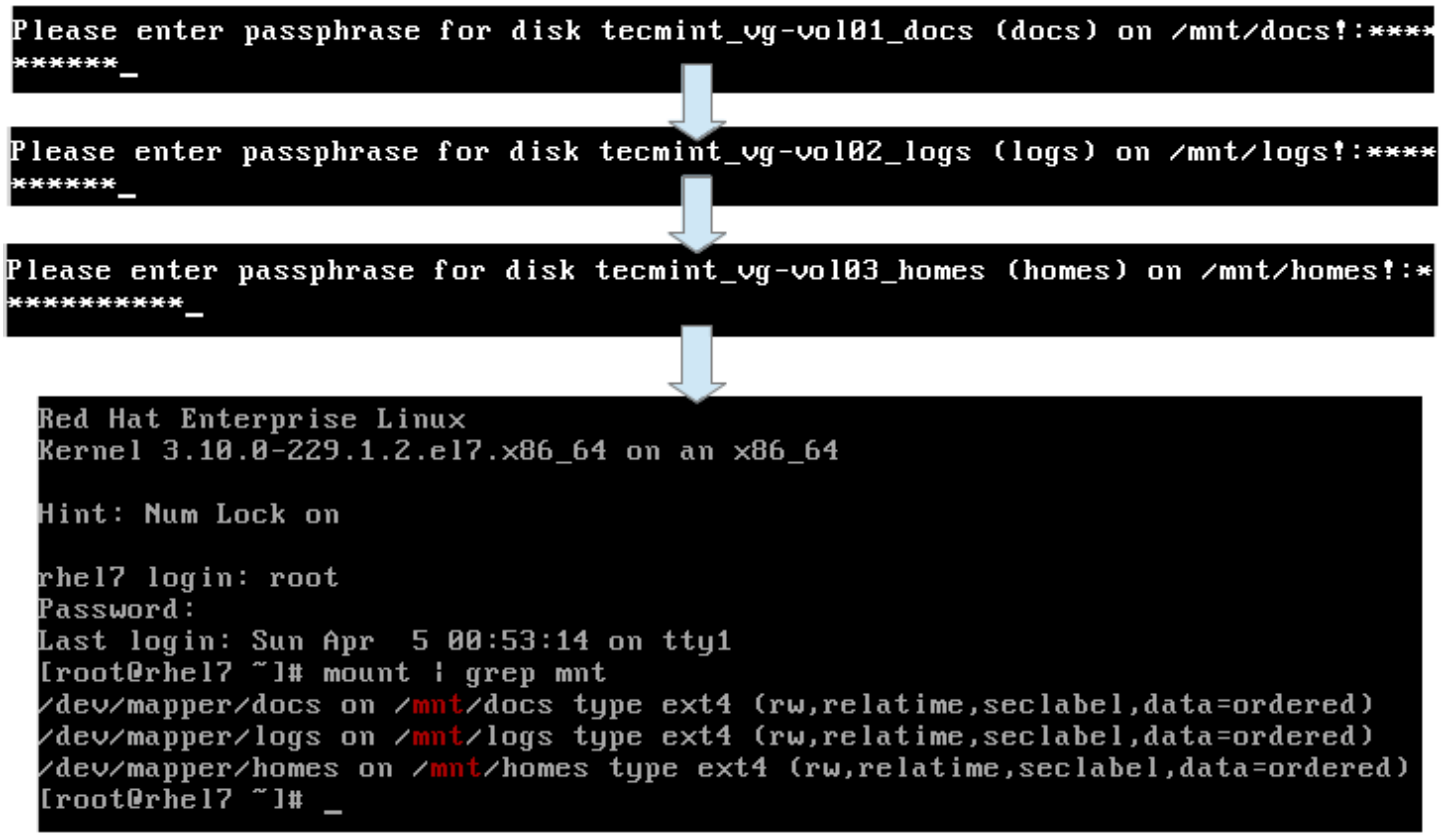


Figure 15: Mounting encrypted volumes during boot

Summary

In this tutorial we have started to explore how to set up and configure system storage using classic volume management tools and SSM, which also integrates filesystem and encryption capabilities in one package. This makes SSM an invaluable tool for his or her storage management tasks.

Chapter 7: File systems formats and mounting network shares

In the last article we started explaining how to set up and configure local system storage using parted and ssm. We also discussed how to create and mount encrypted volumes with a password during system boot. In addition, we warned you to avoid performing critical storage management operations on mounted filesystems. With that in mind we will now review the most used file system formats in Red Hat Enterprise Linux 7 and then proceed to cover the topics of mounting, using, and unmounting -both manually and automatically- network filesystems (CIFS and NFS), along with the implementation of access control lists for your system.

Prerequisites

Before proceeding further, please make sure you have a Samba server and a NFS server available (note that NFSv2 is no longer supported in RHEL 7). During this guide we will use a machine with IP 192.168.0.10 with both services running in it as server, and a RHEL 7 box as client with IP address 192.168.0.18. Later in the article we will tell you which packages you need to install on the client.

File system formats in RHEL 7

Beginning with RHEL 7, XFS has been introduced as the default file system for all architectures due to its high performance and scalability. It currently supports a maximum filesystem size of 500 TB as per the latest tests performed by Red Hat and its partners for mainstream hardware. Also, XFS enables **user_xattr** (extended user attributes) and **acl** (POSIX access control lists) as default mount options, unlike ext3 or ext4 (ext2 is considered deprecated as of RHEL 7), which means that you don't need to specify those options explicitly either on the command line or in **/etc/fstab** when mounting a XFS filesystem (if you want to disable such options in this last case, you have to explicitly use **no_acl** and **no_user_xattr**).

Keep in mind that the extended user attributes can be assigned to files and directories for storing arbitrary additional information such as the mime type, character set or encoding of a file, whereas the access permissions for user attributes are defined by the regular file permission bits.

Access control lists

As every system administrator, either beginner or expert, is well acquainted with regular access permissions on files and directories, which specify certain privileges (read, write, and execute) for the owner, the group, and "the world" (all others). However, feel free to refer to Chapter 3 if you need to refresh your memory a little bit.

However, since the standard ugo/rwx set does not allow to configure different permissions for different users, ACLs were introduced in order to define more detailed access rights for files and directories than those specified by regular permissions. In fact, ACL-defined permissions are a superset of the permissions specified by the file permission bits. Let's see how all of this translates as applied in the real world.

1) There are two types of ACLs: access ACLs, which can be applied to either a specific file or a directory), and default ACLs, which can only be applied to a directory. If files contained therein do not have an ACL set, they inherit the default ACL of their parent directory.

2) To begin, ACLs can be configured per user, per group, or per an user not in the owning group of a file.

3) ACLs are set (and removed) using **setfacl**, with either the **-m** or **-x** options, respectively.

For example, let us create a group named **tecmint** and add users **johndoe** and **davenull** to it:

```
groupadd tecmint
useradd johndoe
useradd davenull
usermod -a -G tecmint johndoe
usermod -a -G tecmint davenull
```

And let's verify (see Fig. 1) that both users belong to supplementary group **tecmint**:

```
id johndoe
id davenull
```

```
[root@rhel7 ~]# id johndoe
uid=1002(johndoe) gid=1002(johndoe) groups=1002(johndoe),1001(tecmint)
[root@rhel7 ~]# id davenull
uid=1003(davenull) gid=1003(davenull) groups=1003(davenull),1001(tecmint)
[root@rhel7 ~]#
```

Fig. 1: Listing the groups an user belongs to

Let's now create a directory called **playground** within **/mnt**, and a file named **testfile.txt** inside. We will set the group owner to **tecmint** and change its default ugo/rwx permissions to **770** (read, write, and execute permissions granted to both the owner and the group owner of the file):

```
mkdir /mnt/playground
touch /mnt/playground/testfile.txt
chmod 770 /mnt/playground/testfile.txt
```

Then switch user to johndoe and davenull, in that order, and write to the file:

```
echo "My name is John Doe" > /mnt/playground/testfile.txt
echo "My name is Dave Null" >> /mnt/playground/testfile.txt
```

So far so good. Now let's have user **gacanepa** write to the file - and the write operation will, which was to be expected.

But what if we actually need user gacanepa (who is not a member of group **tecmint**) to have write permissions on **/mnt/playground/testfile.txt**? The first thing that may come to your mind is adding that user account to group tecmint. But that will give him write permissions on ALL files were the write bit is set for the group, and we don't want that (see Fig. 2). We only want him to be able to write to /mnt/playground/testfile.txt.

```
[root@rhel7 ~]# touch /mnt/playground/testfile.txt
[root@rhel7 ~]# chown :tecmint /mnt/playground/testfile.txt
[root@rhel7 ~]# chmod 770 /mnt/playground/testfile.txt
[root@rhel7 ~]# su johndoe
[johndoe@rhel7 root]$ echo "My name is John Doe" > /mnt/playground/testfile.txt
[johndoe@rhel7 root]$ su davenull
Password:
[davenull@rhel7 root]$ echo "My name is Dave Null" >> /mnt/playground/testfile.txt
[davenull@rhel7 root]$ su gacanepa
Password:
[gacanepa@rhel7 root]$ echo "My name is Gabriel Cánepa" >> /mnt/playground/testfile.txt
bash: /mnt/playground/testfile.txt: Permission denied
[gacanepa@rhel7 root]$
```

Figure 2: Write permission denied to user not in a group

Let's give user **gacanepa** read and write access to **/mnt/playground/testfile.txt**.

Run as root

```
setfacl -R -m u:gacanepa:rwx /mnt/playground
```

and you'll have successfully added an ACL that allows **gacanepa** to write to the test file. Then switch to user **gacanepa** and try to write to the file again:

```
echo "My name is Gabriel Cánepa" > /mnt/playground/testfile.txt
```

To view the ACLs for a specific file or directory, use **getfacl** (see Fig. 3):

```
getfacl /mnt/playground/testfile.txt
```

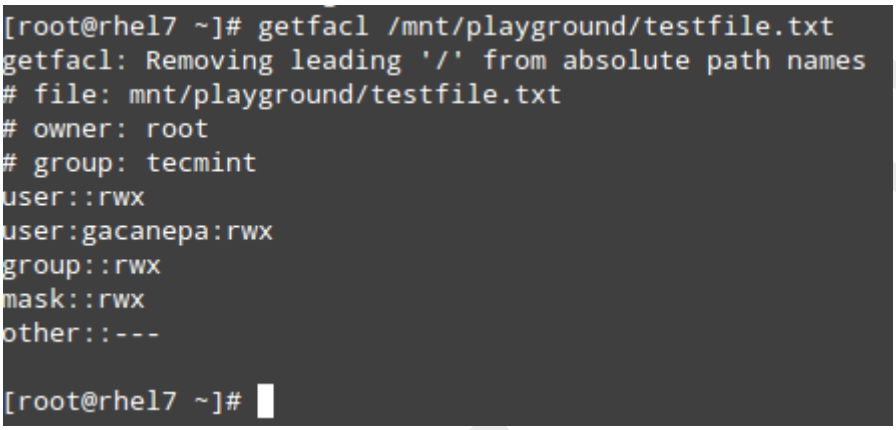


Figure 3: Viewing ACLs

To set a default ACL to a directory (which its contents will inherit unless overwritten otherwise), add **d:** before the rule and specify a directory instead of a file name:

```
setfacl -m d:o:r /mnt/playground
```

The ACL above will allow users not in the owner group to have read access to the future contents of the **/mnt/playground** directory. Note the difference in the output of `getfacl /mnt/playground` before and after the change (refer to Fig. 4):

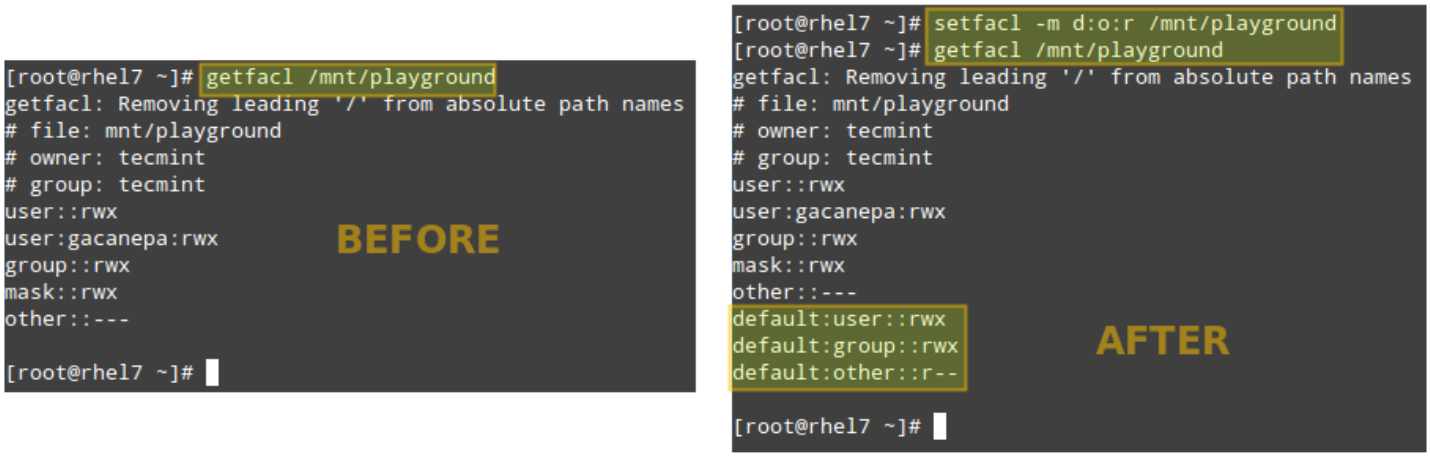


Figure 4: Setting ACLs

Chapter 20 in the official RHEL 7 Storage Administration Guide provides more ACL examples, and I highly recommend you take a look at it and have it handy as reference.

Mounting NFS network shares

To show the list of NFS shares available in your server, you can use the **showmount** command with the **-e** option, followed by the machine name or its IP address. This tool is included in the **nfs-utils** package:

```
yum update && yum install nfs-utils
```

Then do:

```
showmount -e 192.168.0.10
```

and you will get a list of the available NFS shares on 192.168.0.10 (see Fig. 5):

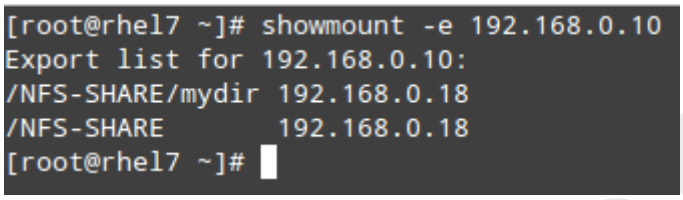


Figure 5: Listing available NFS shares

To mount NFS network shares on the local client using the command line on demand, use the following syntax:

```
mount -t nfs -o [options] remote_host:/remote/directory /local/directory
```

which, in our case, translates to:

```
mount -t nfs 192.168.0.10:/NFS-SHARE /mnt/nfs
```

If you get the following error message: "***Job for rpc-statd.service failed. See "systemctl status rpc-statd.service" and "journalctl -xn" for details.***", make sure the rpcbind service is enabled and started in your system first:

```
systemctl enable rpcbind.socket
systemctl restart rpcbind.service
```

and then reboot. That should do the trick and you will be able to mount your NFS share as explained earlier.

If you need to mount the NFS share automatically on system boot, add a valid entry to the **/etc/fstab** file:

```
remote_host:/remote/directory /local/directory nfs options 0 0
```

The variables **remote_host**, **/remote/directory**, **/local/directory**, and **options** (which is optional) are the same ones used when manually mounting an NFS share from the command line. As per our previous example:

```
192.168.0.10:/NFS-SHARE /mnt/nfs nfs defaults 0 0
```

Mounting CIFS network shares

Samba represents the tool of choice to make a network share available in a network with *nix and Windows machines. To show the Samba shares that are available, use the **smbclient** command with the **-L** flag, followed by the machine name or its IP address. This tool is included in the **samba-client** package:

You will be prompted for root's password in the remote host (see Fig. 6):

```
smbclient -L 192.168.0.10
```

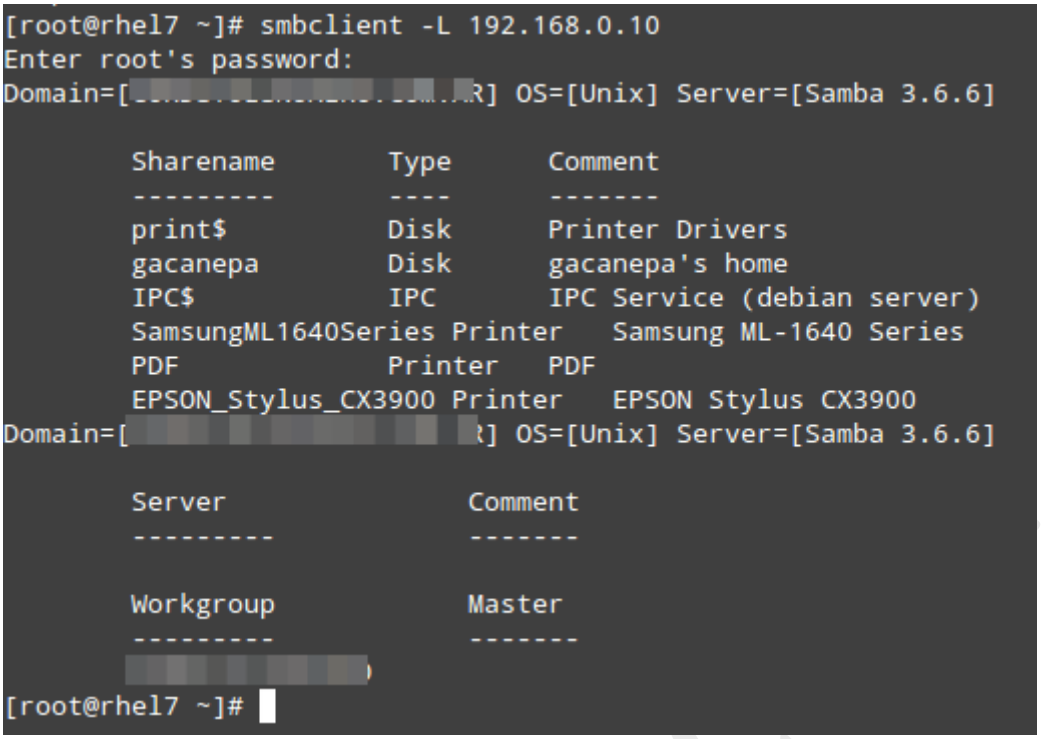


Figure 6: Listing CIFS shares

To mount Samba network shares on the local client using the command line on demand, use the following syntax:

```
mount -t cifs -o credentials=/path/to/credentials/file
//remote_host/samba_share /local/directory
```

which, in our case, translates to:

```
mount -t cifs -o credentials=~/.smbcredentials //192.168.0.10/gacanepa
/mnt/samba
```

where the **smbcredentials** file contains:

```
username=gacanepa

password=XXXXXX
```

and is a hidden file inside root's home (/root/) with permissions set to 600, so that no one else but the owner of the file can read or write to it.

Please note that the **samba_share** is the name of the Samba share as returned by `smbclient -L remote_host` as shown above.

Now, if you need the Samba share to be available automatically on system boot, add a valid entry to the **/etc/fstab** file as follows:

```
//remote_host:/samba_share /local/directory cifs options 0 0
```

The variables **remote_host**, **/samba_share**, **/local/directory**, and **options** (which is optional) are the same ones used when manually mounting a Samba share from the command line. Following the definitions given in our previous example:

```
//192.168.0.10/gacanepa /mnt/samba    cifs
credentials=/root/smbcredentials,defaults 0 0
```

Summary

In this article we have explained how to set up ACLs in Linux, and discussed how to mount CIFS and NFS network shares in a RHEL 7 client. I recommend you to practice these concepts and even mix them (go ahead and try to set ACLs in mounted network shares) until you feel comfortable. If you have questions or comments feel free to use the form below to contact us anytime. Also, feel free to share this article through your social networks.

Chapter 8: Network operations

As a system administrator you will often have to log on to remote systems to perform a variety of administration tasks using a terminal emulator. You will rarely sit in front of a real (physical) terminal, so you need to set up a way to log on remotely to the machines that you will be asked to manage. In fact, that may be the last thing that you will have to do in front of a physical terminal. For security reasons, using Telnet for this purpose is not a good idea, as all traffic goes through the wire in unencrypted, plain text.

In addition, in this article we will also review how to configure network services to start automatically at boot and learn how to set up network and hostname resolution statically or dynamically.

Installing and securing SSH communications

For you to be able to log on remotely to a RHEL 7 box using SSH, you will have to install the **openssh**, **openssh-clients** and **openssh-servers** packages. The following command not only will install the remote login program, but also the secure file transfer tool, as well as the remote file copy utility:

```
yum update && yum install openssh openssh-clients openssh-servers
```

Note that it's a good idea to install the server counterparts as you may want to use the same machine as both client and server at some point or another.

After installation, there is a couple of basic things that you need to take into account if you want to secure remote access to your SSH server. The following settings should be present in the **/etc/ssh/sshd_config** file.

1) Change the port where the sshd daemon will listen on from 22 (the default value) to a high port (~2000 or greater), but first make sure the chosen port is not being used. For example, let's suppose you choose port 2500. Use netstat in order to check whether the chosen port is being used or not:

```
netstat -npltu | grep 2500
```

If **netstat** does not return anything, you can safely use port 2500 for **sshd**, and you should change the **Port** setting in the configuration file as follows:

```
Port 2500
```

2) Only allow protocol 2:

```
Protocol 2
```

3) Configure the authentication timeout to 2 minutes, do not allow root logins, and restrict to a minimum the list of users which are allowed to login via ssh:

```
LoginGraceTime 2m
PermitRootLogin no
AllowUsers gacanepa
```

4) If possible, use key-based instead of password authentication:

```
PasswordAuthentication no
RSAAuthentication yes
PubkeyAuthentication yes
```

This assumes that you have already created already created a key pair with your user name on your client machine and copied it to your server previously.

Networking and basic name resolution

Every system administrator should be well acquainted with the following system-wide configuration files:

1. **/etc/hosts** is used to resolve names <---> IPs in small networks.

Every line in the /etc/hosts file has the following structure:

IP address - Hostname - FQDN

For example,

```
192.168.0.10     laptop     laptop.gabrielcanepa.com.ar
```

2. **/etc/resolv.conf** specifies the IP addresses of DNS servers and the search domain, which is used for completing a given query name to a fully qualified domain name when no domain suffix is supplied.

Under normal circumstances, you don't need to edit this file as it is managed by the system. However, should you want to change DNS servers, be advised that you need to stick to the following structure in each line:

nameserver - IP address

For example,

```
nameserver 8.8.8.8
```

3. **/etc/host.conf** specifies the methods and the order by which hostnames are resolved within a network. In other words, tells the name resolver which services to use, and in what order.

Although this file has several options, the most common and basic setup includes a line as follows:

```
order bind,hosts
```

which indicates that the resolver should first look in the nameservers specified in resolv.conf **and then** to the /etc/hosts file for name resolution.

4. **/etc/sysconfig/network** contains routing and global host information for all network interfaces. The following values may be used:

```
NETWORKING=yes|no

HOSTNAME=value
```

where value should be the Fully Qualified Domain Name (FQDN).

```
GATEWAY=XXX.XXX.XXX.XXX
```

where **XXX.XXX.XXX.XXX** is the IP address of the network's gateway.

```
GATEWAYDEV=value
```

In a machine with multiple NICs, value is the gateway device, such as enp0s3.

5. Files inside /etc/sysconfig/network-scripts (network adapters configuration files)

Inside the directory mentioned previously, you will find several plain text files named

ifcfg-***name***

where ***name*** is the name of the NIC as returned by `ip link show` (see Fig. 1):

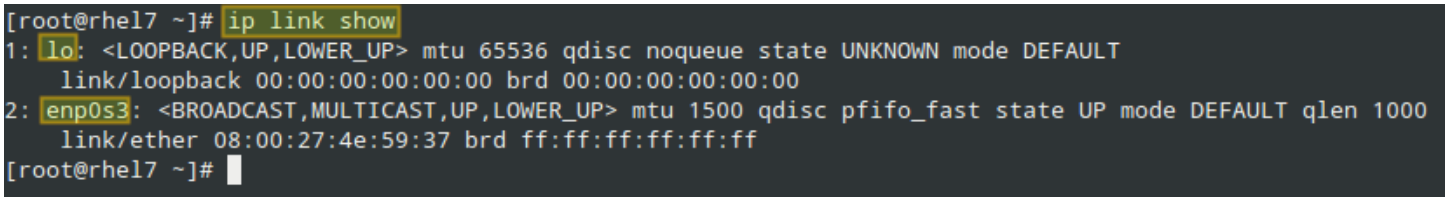


Figure 1: Listing network interface cards

For example (see Fig. 2):

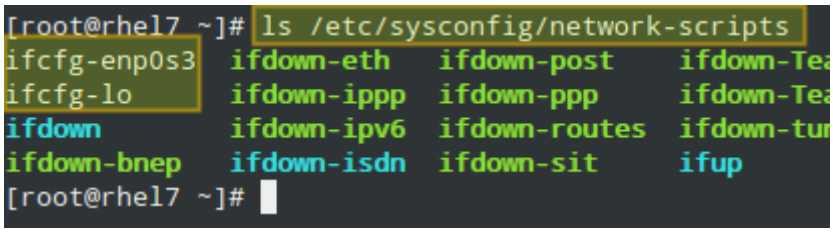


Figure 2: NIC configuration files

Other than for the loopback interface, you can expect a similar configuration for your NICs. Note that some variables, if set, will override those present in **/etc/sysconfig/network** for this particular interface. Each line is commented for clarification in this article but in the actual file you should avoid comments:

```
HWADDR=08:00:27:4E:59:37 # The MAC address of the NIC

TYPE=Ethernet # Type of connection

BOOTPROTO=static # This indicates that this NIC has been assigned a static IP. If this variable was set to dhcp, the NIC will be assigned an IP address by a DHCP server and thus the next two lines should not be present in that case.

IPADDR=192.168.0.18

NETMASK=255.255.255.0

GATEWAY=192.168.0.1

NM_CONTROLLED=no # Should be added to the Ethernet interface to prevent NetworkManager from changing the file.

NAME=enp0s3

UUID=14033805-98ef-4049-bc7b-d4bea76ed2eb

ONBOOT=yes # The operating system should bring up this NIC during boot
```

Setting hostnames

In Red Hat Enterprise Linux 7, the **hostnamectl** command is used to both query and set the system's hostname.

To display the current hostname, type

```
hostnamectl status
```

and press Enter (see Fig. 3):

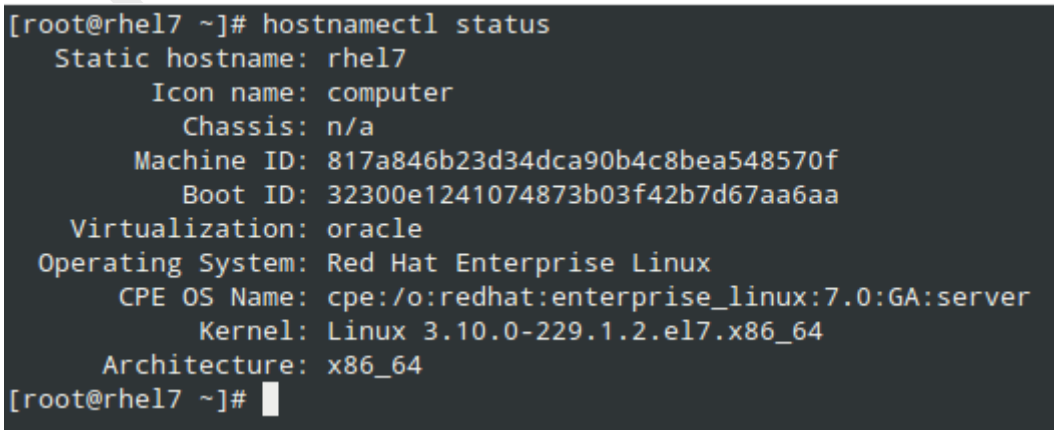


Figure 3: Viewing the system's hostname information

To change the hostname, use

```
hostnamectl set-hostname [new hostname]
```

For example,

```
hostnamectl set-hostname cinderella
```

For the changes to take effect you will need to restart the **hostnamed** daemon as shown in Fig. 4 (that way you will not have to log off and on again in order to apply the change):

```
systemctl restart systemd-hostnamed
```

```
[root@rhel7 ~]# hostnamectl status
   Static hostname: rhel7
         Icon name: computer
           Chassis: n/a
        Machine ID: 817a846b23d34dca90b4c8bea548570f
           Boot ID: 32300e1241074873b03f42b7d67aa6aa
    Virtualization: oracle
  Operating System: Red Hat Enterprise Linux
       CPE OS Name: cpe:/o:redhat:enterprise_linux:7.0:GA:server
            Kernel: Linux 3.10.0-229.1.2.el7.x86_64
      Architecture: x86_64
[root@rhel7 ~]# hostnamectl set-hostname cinderella
[root@rhel7 ~]# systemctl restart systemd-hostnamed
[root@rhel7 ~]# hostnamectl status
   Static hostname: cinderella
         Icon name: computer
           Chassis: n/a
        Machine ID: 817a846b23d34dca90b4c8bea548570f
           Boot ID: 32300e1241074873b03f42b7d67aa6aa
    Virtualization: oracle
  Operating System: Red Hat Enterprise Linux
       CPE OS Name: cpe:/o:redhat:enterprise_linux:7.0:GA:server
            Kernel: Linux 3.10.0-229.1.2.el7.x86_64
      Architecture: x86_64
[root@rhel7 ~]#
```

Figure 4: Changing the hostname

In addition, RHEL 7 also includes the **nmcli** utility that can be used for the same purpose. To display the hostname, run

```
nmcli general hostname
```

and to change it:

```
nmcli general hostname [new hostname]
```

For example (see Fig. 5),

```
nmcli general hostname rhel7
```

```
[root@rhel7 ~]# nmcli general hostname
cinderella
[root@rhel7 ~]# nmcli general hostname rhel7
[root@rhel7 ~]# nmcli general hostname
rhel7
[root@rhel7 ~]#
```

Figure 5: Using nmcli for hostname management

Starting network services on boot

To wrap up, let us see how we can ensure that network services are started automatically on boot. In simple terms, this is done by creating symlinks to certain files specified in the **[Install]** section of the service configuration files.

In the case of firewalld (**/usr/lib/systemd/system/firewalld.service**):

```
[Install]
WantedBy=basic.target
Alias=dbus-org.fedoraproject.FirewallD1.service
```

To enable the service:

```
systemctl enable firewalld
```

On the other hand, disabling firewalld entitles removing the symlinks as can be seen in Fig. 6:

```
systemctl disable firewalld
```

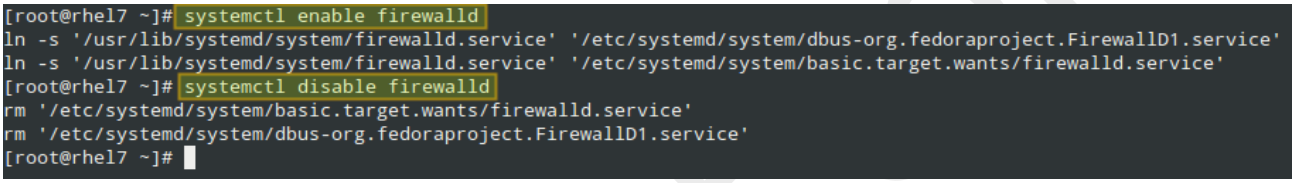


Figure 6: Enabling and disabling services

Summary

In this article we have summarized how to install and secure connections via SSH to a RHEL server, how to change its name, and finally how to ensure that network services are started on boot. If you notice that a certain service has failed to start properly, you can use `systemctl status -l [service]` and `journalctl -xn` to troubleshoot it.

Feel free to let us know what you think about this article using the comment form below. Questions are also welcome. We look forward to hearing from you!

Chapter 9: Setting up a web server and a FTP server

A web server (also known as a HTTP server) is a service that handles content (most commonly web pages, but other types of documents as well) over to a client in a network. A FTP server is one of the oldest and most commonly used resources (even to this day) to make files available to clients on a network in cases where no authentication is necessary since FTP uses username and password without encryption.

The web server available in RHEL 7 is version 2.4 of the Apache HTTP Server. As for the FTP server, we will use the Very Secure Ftp Daemon (aka **vsftpd**) to establish connections secured by TLS. In this article we will explain how to install, configure, and secure a web server and a FTP server in RHEL 7.

Installing the software

In this guide we will use a RHEL 7 server with a static IP address of 192.168.0.18/24. To install Apache and VSFTPD, run the following command:

```
yum update && yum install httpd vsftpd
```

When the installation completes, both services will be disabled initially, so we need to start them manually for the time being and enable them to start automatically beginning with the next boot:

```
systemctl start httpd
systemctl enable httpd
systemctl start vsftpd
systemctl enable vsftpd
```

In addition, we have to open ports 80 and 21, where the web and ftp daemons are listening, respectively, in order to allow access to those services from the outside:

```
firewall-cmd --zone=public --add-port=80/tcp --permanent
firewall-cmd --zone=public --add-service=ftp --permanent
firewall-cmd --reload
```

To confirm that the web server is working properly, fire up your browser and enter the IP of the server. You should see the test page (Fig. 1):

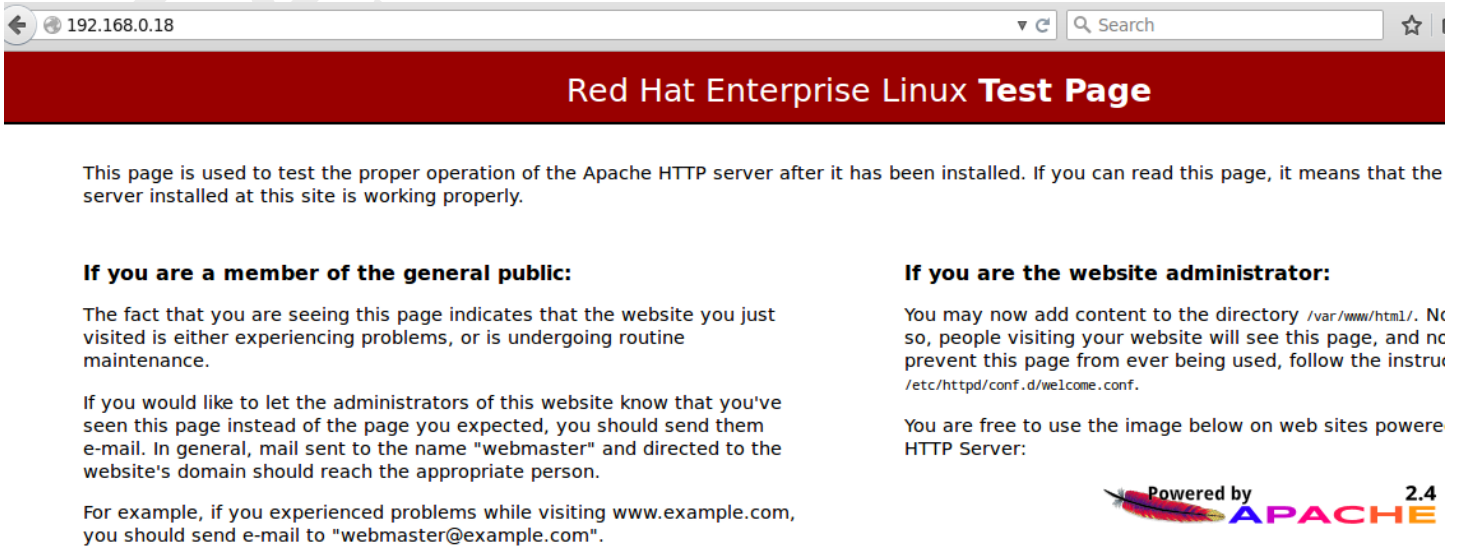


Figure 1: The Apache web server running in RHEL

As for the ftp server, we will have to configure it further, which we will do in a minute, before confirming that it's working as expected.

Configuring and securing the web server

The main configuration file for Apache is located in **/etc/httpd/conf/httpd.conf**, but it may rely on other files present inside **/etc/httpd/conf.d**. Although the default configuration should be sufficient for most cases, it's a good idea to become familiar with all the available options as described in the official documentation.

As always, make a backup copy of the main configuration file before editing it:

```
cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.$(date +%Y%m%d)
```

then open it with your preferred text editor and look for the following variables:

- **ServerRoot**: the directory where the server's configuration, error, and log files are kept.
- **Listen**: instructs Apache to listen on specific IP address and / or ports.
- **Include**: allows the inclusion of other configuration files, which must exist. Otherwise, the server will fail, as opposed to the **IncludeOptional** directive, which is silently ignored if the specified configuration files do not exist.
- **User** and **Group**: the name of the user/group to run the httpd service as.
- **DocumentRoot**: The directory out of which Apache will serve your documents. By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations.
- **ServerName**: this directive sets the hostname (or IP address) and port that the server uses to identify itself.

The first security measure will consist of creating a dedicated user and group (i.e. tecmint/tecmint) to run the web server as and changing the default port to a higher one (9000 in this case):

```
ServerRoot "/etc/httpd"
Listen 192.168.0.18:9000
User tecmint
Group tecmint
DocumentRoot "/var/www/html"
ServerName 192.168.0.18:9000
```

You can test the configuration file with

```
apachectl configtest
```

and if everything is ok, then restart the web server

```
systemctl restart httpd
```

and don't forget to enable the new port (and disable the old one) in the firewall:

```
firewall-cmd --zone=public --remove-port=80/tcp --permanent
firewall-cmd --zone=public --add-port=9000/tcp --permanent
firewall-cmd --reload
```

Note that, due to SELinux policies, you can only use the ports returned by

```
semanage port -l | grep -w '^http_port_t'
```

for the web server.

If you want to use another port (i.e. TCP port 8100), you will have to add it to SELinux port context for the httpd service (see Fig. 2):

```
semanage port -a -t http_port_t -p tcp 8100
```

```
[root@rhel7 ~]# semanage port -l | grep -w '^http_port_t'
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
[root@rhel7 ~]# semanage port -a -t http_port_t -p tcp 8100
[root@rhel7 ~]# semanage port -l | grep -w '^http_port_t'
http_port_t                    tcp      8100, 80, 81, 443, 488, 8008, 8009, 8443, 9000
[root@rhel7 ~]#
```

Figure 2: Adding ports to the SELinux port context for the httpd service

To further secure your Apache installation, follow these steps:

1) The user Apache is running as should not have access to a shell:

```
usermod -s /sbin/nologin tecmint
```

2) Disable directory listing in order to prevent the browser from displaying the contents of a directory if there is no **index.html** present in that directory. Edit **/etc/httpd/conf/httpd.conf** (and the configuration files for virtual hosts, if any) and make sure that the **Options** directive, both at the top and at Directory block levels, is set to **None**:

```
Options None
```

3) Hide information about the web server and the operating system in HTTP responses. Edit **/etc/httpd/conf/httpd.conf** as follows:

```
ServerTokens Prod
ServerSignature Off
```

Now you are ready to start serving content from your **/var/www/html directory**.

Configuring and securing the FTP server

As in the case of Apache, the main configuration file for Vsftpd (**/etc/vsftpd/vsftpd.conf**) is well commented and while the default configuration should suffice for most applications, you should become acquainted with the documentation and the man page (`man vsftpd.conf`) in order to operate the ftp server more efficiently (I can't emphasize that enough!).

In our case, these are the directives used:

```
anonymous_enable=NO
local_enable=YES
write_enable=YES
local_umask=022
dirmessage_enable=YES
xferlog_enable=YES
connect_from_port_20=YES
xferlog_std_format=YES
chroot_local_user=YES
allow_writeable_chroot=YES
listen=NO
```

```
listen_ipv6=YES
pam_service_name=vsftpd
userlist_enable=YES
tcp_wrappers=YES
```

By using **chroot_local_user=YES**, local users will be (by default) placed in a chroot'ed jail in their home directory right after login. This means that local users will not be able to access any files outside their corresponding home directories.

Finally, to allow ftp to read files in the user's home directory, set the following SELinux boolean:

```
setsebool -P ftp_home_dir on
```

You can now connect to the ftp server using a client such as Filezilla (see Fig. 3):

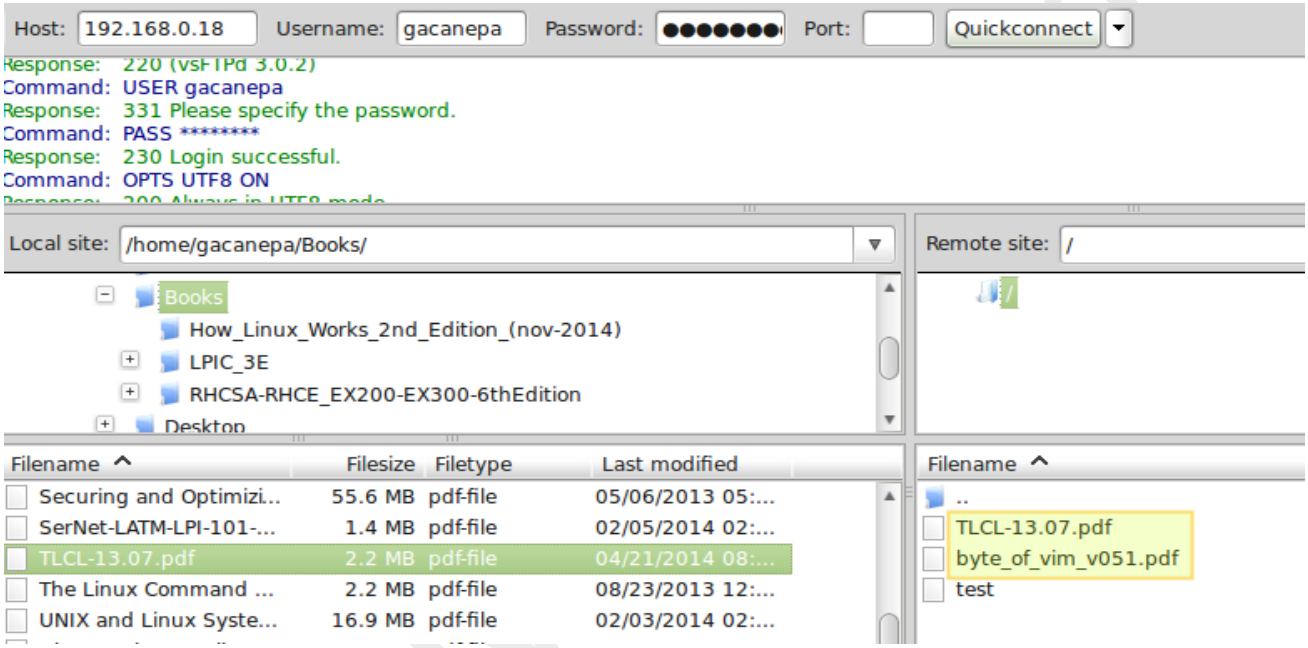


Figure 3: Connecting to our FTP server using FileZilla

Note that the **/var/log/xferlog** log records downloads and uploads, which concur with the above directory listing (see Fig. 4):

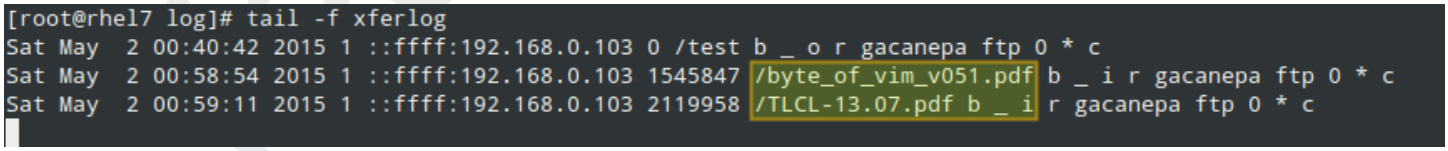


Figure 4: Checking the logs of the FTP server

Summary

In this tutorial we have explained how to set up a web and a ftp server. Due to the vastness of the subject, it is not possible to cover all the aspects of these topics (i.e. virtual web hosts). Thus, I recommend you also check other excellent articles in Tecmint.com about Apache and Vsftpd.

Chapter 10: Package management and system logs

In this article we will review how to install, update, and remove packages in Red Hat Enterprise Linux 7. We will also cover how to automate tasks using **cron**, and will finish this guide explaining how to locate and interpret system logs files with the focus of teaching you why all of these are essential skills for every system administrator.

Managing packages using yum

To install a package along with all its dependencies that are not already installed, you will use

```
yum -y install package_name(s)
```

where `package_name(s)` represent at least one real package name.

For example, to install **httpd** and **mlocate** (in that order), type

```
yum -y install httpd mlocate
```

Note that the letter **y** in the example above bypasses the confirmation prompts that **yum** presents before performing the actual download and installation of the requested programs. You can leave it out if you want.

By default, yum will install the package with the architecture that matches the OS architecture, unless overridden by appending the package architecture to its name. For example, on a 64 bit system, yum install *package* will install the x86_64 version of *package*, whereas yum install *package*.x86 (if available) will install the 32-bit one.

There will be times when you want to install a package but don't know its exact name. The search all or search options can search the currently enabled repositories for a certain keyword in the package name and / or in its description as well, respectively.

For example,

```
yum search log
```

will search the installed repositories for packages with the word **log** in their names and summaries, whereas

```
yum search all log
```

will look for the same keyword in the package description and url fields as well.

Once the search returns a package listing, you may want to display further information about some of them before installing. That is when the **info** option will come in handy (see Fig. 1):

```
yum info logwatch
```

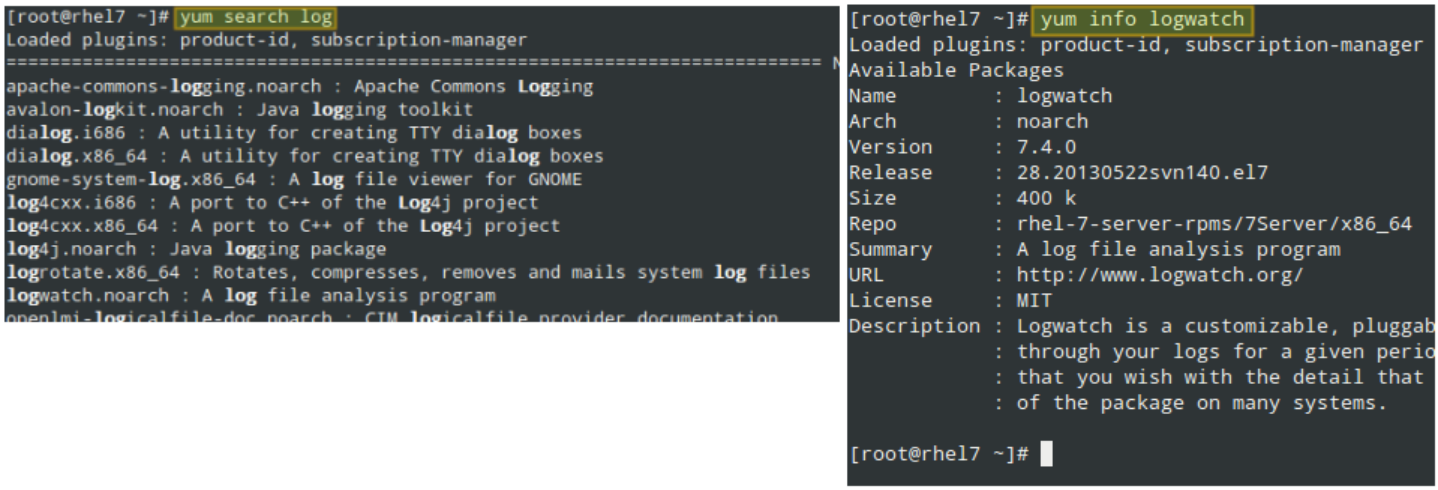


Figure 1: Viewing package information using yum

You can regularly check for updates with the following command:

```
yum check-update
```

The above command will return all the installed packages for which an update is available. In the example shown in the image below, only **rhel-7-server-rpms** has an update available (see Fig. 2):

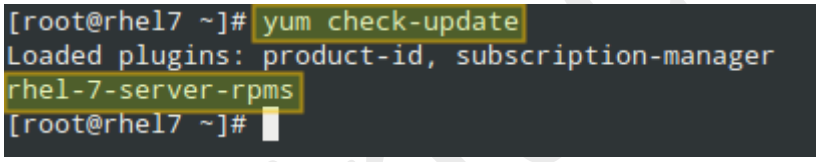


Figure 2: Checking for package updates using yum

You can then update that package alone with

```
yum update rhel-7-server-rpms
```

If there are several packages that can be updated, yum update will update all of them at once.

Now what happens when you know the name of an executable, such as **ps2pdf**, but don't know which package provides it? You can find out with `yum whatprovides "*/[executable] "`. For example (see Fig. 3),

```
yum whatprovides "*/ps2pdf"
```

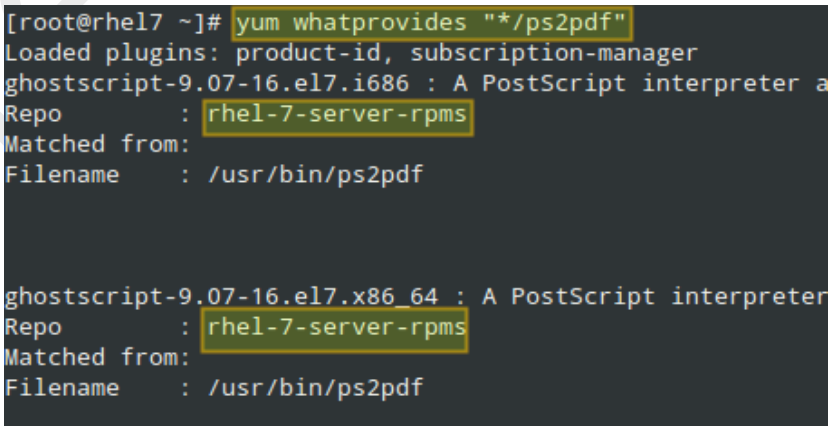


Figure 3: Finding out which package provides a specific utility

Now, when it comes to removing a package, you can do so with `yum remove package`. Easy, huh? This goes to show that **yum** is a complete and powerful package manager.

Good old plain RPM

RPM (aka **RPM Package Manager**, or originally **RedHat Package Manager**) can also be used to install or update packages when they come in form of standalone .rpm packages. It is often utilized with the **-Uvh** flags to indicate that it should install the package if it's not already present or attempt to update it if it's installed (**-U**), producing a verbose output (**-v**) and a progress bar with hash marks (**-h**) while the operation is being performed. For example,

```
rpm -Uvh package.rpm
```

Another typical use of rpm is to produce a list of currently installed packages with `rpm -qa` (short for **q**uery **a**ll), as can be seen in Fig. 4:

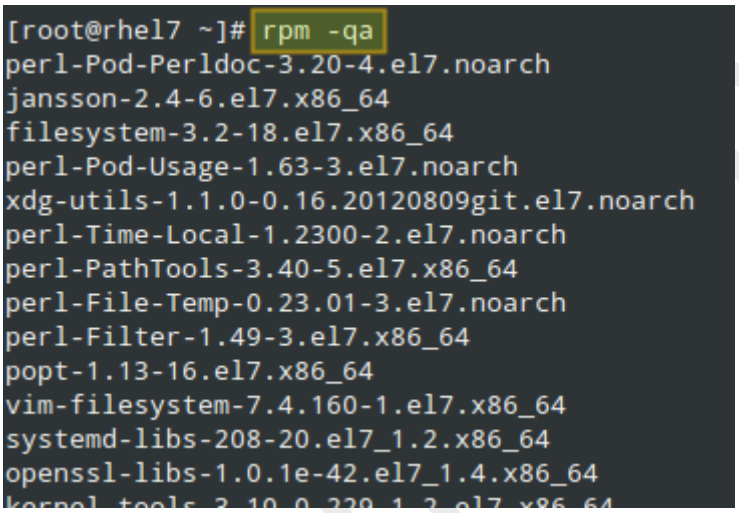


Figure 4: Listing installed packages using rpm

Scheduling tasks using cron

Linux and other Unix-like operating systems include a tool called **cron** that allows you to schedule tasks (i.e. commands or shell scripts) to run on a periodic basis. Cron checks every minute the **/var/spool/cron** directory for files which are named after accounts in **/etc/passwd**. When executing commands, any output is mailed to the owner of the crontab (or to the user specified in the **MAILTO** environment variable in the /etc/crontab, if it exists).

Crontab files (which are created by typing `crontab -e` and pressing Enter) have the following format (see Fig. 5):

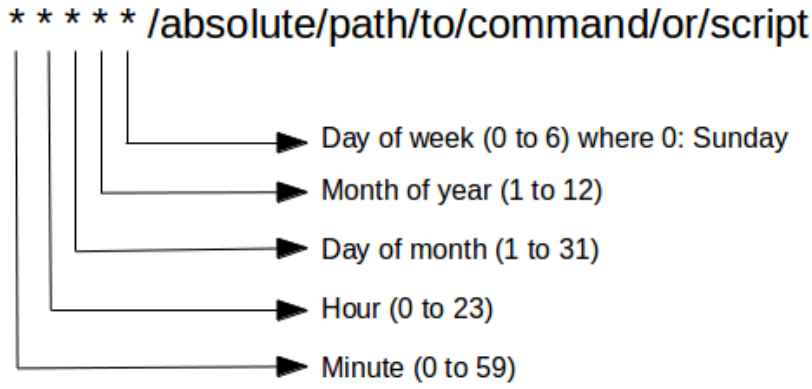


Figure 5: Crontab entries format

Thus, if we want to update the local file database (which is used by locate to find files by name or pattern) every second day of the month at 2:15 am, we need to add the following crontab entry:

```
15 02 2 * * /bin/updatedb
```

The above crontab entry reads, "***Run /bin/updatedb on the second day of the month, every month of the year, regardless of the day of the week, at 2:15 am***". As I'm sure you already guessed, the star symbol is used as a wildcard character.

After adding a cron job, you can see that a file named root was added inside **/var/spool/cron**, as we mentioned earlier. That file lists all the tasks that the crond daemon should run (see Fig. 6):

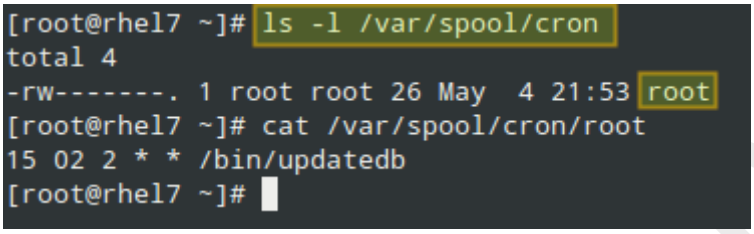


Figure 6: Listing cron jobs per user

In the above image, the current user's crontab can be displayed either using **cat /var/spool/cron/root** or **crontab -l**.

If you need to run a task on a more fine-grained basis (for example, twice a day or three times each month), cron can also help you to do that.

For example, to run /my/script on the 1st and 15th of each month and send any output to /dev/null, you can add two crontab entries as follows:

```
01 00 1 * * /my/script > /dev/null 2>&1
01 00 15 * * /my/script > /dev/null 2>&1
```

But in order for the task to be easier to maintain, you can combine both entries into one:

```
01 00 1,15 * *  /my/script > /dev/null 2>&1
```

Following the previous example, we can run /my/other/script at 1:30 am on the first day of the month every three months:

```
30 01 1 1,4,7,10 * /my/other/script > /dev/null 2>&1
```

But when you have to repeat a certain task every "x" minutes, hours, days, or months, you can divide the right position by the desired frequency. The following crontab entry has the exact same meaning as the previous one:

```
30 01 1 */3 * /my/other/script > /dev/null 2>&1
```

Or perhaps you need to run a certain job on a fixed frequency or after the system boots, for example. You can use one of the following string instead of the five fields to indicate the exact time when you want your job to run:

```
@reboot          Run when the system boots.
@yearly          Run once a year, same as 00 00 1 1 *.
@monthly         Run once a month, same as 00 00 1 * *.
@weekly          Run once a week, same as 00 00 * * 0.
@daily           Run once a day, same as 00 00 * * *.
@hourly          Run once an hour, same as 00 * * * *.
```

Locating and checking logs

System logs are located (and rotated) inside the /var/log directory. According to the Linux Filesystem Hierarchy Standard, this directory contains miscellaneous log files, which are written to it or an appropriate subdirectory (such as **audit**, **httpd**, or **samba** in the image below) by the corresponding daemons during system operation (see Fig. 7):

```
[root@rhel7 ~]# ls /var/log
anaconda        cron           dmesg      lastlog          maillog-20150504   messages-201
audit           cron-20150418  dmesg.old  maillog          messages           ppp
boot.log        cron-20150420  firewalld  maillog-20150418  messages-20150418  rhsm
btmp            cron-20150428  grubby     maillog-20150420  messages-20150420  samba
btmp-20150501   cron-20150504  httpd      maillog-20150428  messages-20150428  secure
[root@rhel7 ~]#
```

Figure 7: Locating logs inside /var/log

Other interesting logs are **dmesg** (contains all messages from kernel ring buffer), **secure** (logs connection attempts that require user authentication), messages (system-wide messages) and **wtmp** (records of all user logins and logouts).

Logs are very important in that they allow you to have a glimpse of what is going on at all times in your system, and what has happened in the past. They represent a priceless tool to troubleshoot and monitor a Linux server, and thus are often used with the tail -f command to display events, in real time, as they happen and are recorded in a log.

For example, if you want to display kernel-related events, type the following command:

```
tail -f /var/log/dmesg
```

Same if you want to view access to your web server:

```
tail -f /var/log/httpd/access.log
```

Summary

If you know how to efficiently manage packages, schedule tasks, and where to look for information about the current and past operation of your system you can rest assure that you will not run into surprises very often. I hope this article has helped you learn or refresh your knowledge about these basic skills.

Chapter 11: Firewall

In simple words, a firewall is a security system that controls the incoming and outgoing traffic in a network based on a set of predefined rules (such as the packet destination / source or type of traffic, for example). In this article we will review the basics of **firewalld**, the default dynamic firewall daemon in Red Hat Enterprise Linux 7, and **iptables service**, the legacy firewall service for Linux, with which most system and network administrators are well acquainted, and which is also available in RHEL 7.

A brief comparison between firewalld and iptables

Under the hood, both firewalld and the iptables service talk to the netfilter framework in the kernel through the same interface, not surprisingly, the iptables command. However, as opposed to the iptables service, firewalld can change the settings during normal system operation without existing connections being lost.

Firewalld should be installed by default in your RHEL system, though it may not be running. You can verify with the following commands (firewall-config is the package needed for the user interface configuration tool, see Fig. 1):

```
yum info firewalld firewall-config
```

```
[root@rhel7 ~]# yum info firewalld firewall-config
Loaded plugins: product-id, subscription-manager
Installed Packages
Name        : firewalld
Arch        : noarch
Version     : 0.3.9
Release     : 11.el7
Size        : 2.3 M
Repo        : installed
From repo   : rhel-7-server-rpms
Summary     : A firewall daemon with D-BUS interface providing a dynamic firewall
URL         : http://fedorahosted.org/firewalld
License     : GPLv2+
Description : firewalld is a firewall service daemon that provides a dynamic customizable
            : firewall with a D-BUS interface.

Available Packages
Name        : firewall-config
Arch        : noarch
Version     : 0.3.9
Release     : 11.el7
Size        : 103 k
Repo        : rhel-7-server-rpms/7Server/x86_64
Summary     : Firewall configuration application
URL         : http://fedorahosted.org/firewalld
License     : GPLv2+
Description : The firewall configuration application provides an configuration interface for
            : firewalld.
```

Figure 1: Displaying info about firewalld and firewall-config

And (see Fig. 2)

```
systemctl status -l firewalld.service
```

```
[root@rhel7 ~]# systemctl status -l firewalld.service
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Fri 2015-05-08 19:03:04 EDT; 15min ago
 Main PID: 633 (firewalld)
   CGroup: /system.slice/firewalld.service
           └─633 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid

May 08 19:03:00 rhel7 systemd[1]: Starting firewalld - dynamic firewall daemon...
May 08 19:03:04 rhel7 systemd[1]: Started firewalld - dynamic firewall daemon.
[root@rhel7 ~]#
```

Figure 2: Checking the running status of firewalld

On the other hand, the iptables service is not included by default, but can be installed through

```
yum update && yum install iptables-services
```

Both daemons can be started and enabled to start on boot with the usual systemd commands:

```
systemctl start firewalld.service | iptables-service.service
systemctl enable firewalld.service | iptables-service.service
```

As for the configuration files, the iptables service uses /etc/sysconfig/iptables (which will not exist if the package is not installed in your system). On a RHEL 7 box used as a cluster node, this file looks as follows (see Fig. 3):

```
[root@node01 ~]# cat /etc/sysconfig/iptables
# Generated by iptables-save v1.4.21 on Tue Apr 28 18:41:14 2015
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [262:28166]
-A INPUT -p tcp -m state --state NEW -m tcp --dport 7790 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 7789 -j ACCEPT
-A INPUT -m addrtype --dst-type MULTICAST -j ACCEPT
-A INPUT -p igmp -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 2224 -j ACCEPT
-A INPUT -p udp -m state --state NEW -m multiport --dports 5404,5405 -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Tue Apr 28 18:41:14 2015
[root@node01 ~]#
```

Figure 3: Viewing iptables rules

whereas firewalld store its configuration across two directories, /usr/lib/firewalld and /etc/firewalld (see Fig. 4):

```
[root@rhel7 ~]# ls /usr/lib/firewalld/ /etc/firewalld/
/etc/firewalld/:
firewalld.conf  icmptypes  lockdown-whitelist.xml  services  zones

/usr/lib/firewalld/:
icmptypes  services  zones
[root@rhel7 ~]#
```

Figure 4: Configuration directories of firewalld

We will examine these configuration files further later in this article, after we add a few rules here and there. By now it will suffice to remind you that you can always find more information about both tools with

```
man firewalld.conf
man firewall-cmd
man iptables
```

Using iptables to control network traffic

Let's jump in right into the examples.

Example 1: Allowing both incoming and outgoing web traffic

TCP ports 80 and 443 are the default ports used by the Apache web server to handle normal (HTTP) and secure (HTTPS) web traffic. You can allow incoming and outgoing web traffic through both ports on the enp0s3 interface as follows:

```
iptables -A INPUT -i enp0s3 -p tcp --dport 80 -m state --state
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o enp0s3 -p tcp --sport 80 -m state --state
ESTABLISHED -j ACCEPT
iptables -A INPUT -i enp0s3 -p tcp --dport 443 -m state --state
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o enp0s3 -p tcp --sport 443 -m state --state
ESTABLISHED -j ACCEPT
```

Example 2: Block all (or some) incoming connections from a specific network

There may be times when you need to block all (or some) type of traffic originating from a specific network, say 192.168.1.0/24 for example:

```
iptables -I INPUT -s 192.168.1.0/24 -j DROP
```

will drop all packages coming from the 192.168.1.0/24 network, whereas

```
iptables -A INPUT -s 192.168.1.0/24 --dport 22 -j ACCEPT
```

will only allow incoming traffic through port 22.

Example 3: Redirect incoming traffic to another destination

If you use your RHEL 7 box not only as a software firewall, but also as the actual hardware-based one, so that it sits between two distinct networks, IP forwarding must have been already enabled in your system. If not, you need to edit /etc/sysctl.conf and set the value of net.ipv4.ip_forward to 1, as follows:

```
net.ipv4.ip_forward = 1
```

then save the change, close your text editor and finally run the following command to apply the change:

```
sysctl -p /etc/sysctl.conf
```

For example, you may have a printer installed at an internal box with IP 192.168.0.10, with the CUPS service listening on port 631 (both on the print server and on your firewall). In order to forward print requests from clients on the other side of the firewall, you should add the following iptables rule:

```
iptables -t nat -A PREROUTING -i enp0s3 -p tcp --dport 631 -j DNAT --to 192.168.0.10:631
```

Please keep in mind that iptables reads its rules sequentially, so make sure the default policies or later rules do not override those outlined in the examples above.

Getting started with firewalld

One of the changes introduced with firewalld are zones. This concept allows to separate networks into different zones level of trust the user has decided to place on the devices and traffic within that network.

To list the active zones:

```
firewall-cmd --get-active-zones
```

In the example below, the public zone is active, and the enp0s3 interface has been assigned to it automatically. To view all the information about a particular zone (see Fig. 5):

```
firewall-cmd --zone=public --list-all
```

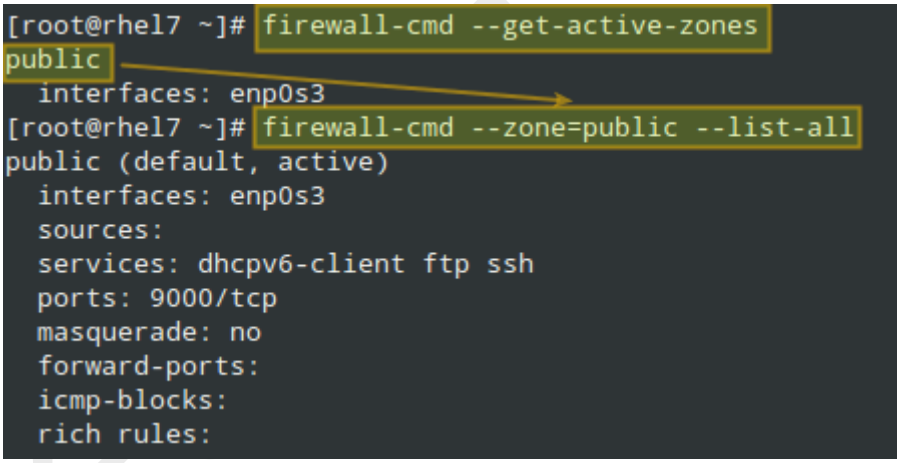


Figure 5: Viewing information about zones in firewalld

Since you can read more about zones in the RHEL 7 Security guide, we will only list some specific examples here.

Example 4: Allowing services through the firewall

To get a list of the supported services, use (see Fig. 6)

```
firewall-cmd --get-services
```

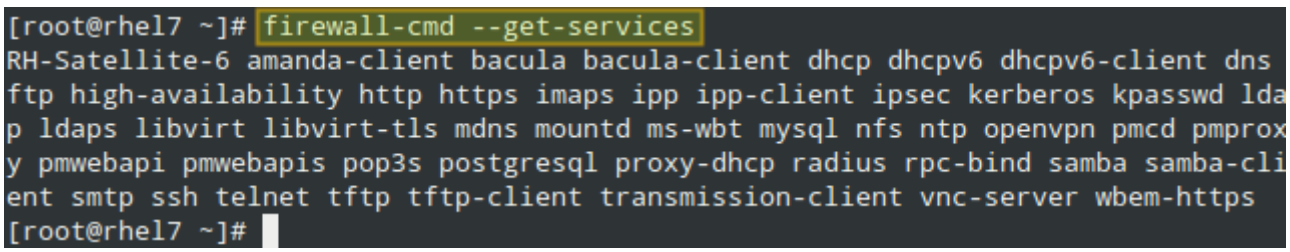


Figure 6: Supported services in firewalld

To allow http and https web traffic through the firewall, effective immediately and on subsequent boots:

```
firewall-cmd --zone=MyZone --add-service=http
firewall-cmd --zone=MyZone --permanent --add-service=http
firewall-cmd --zone=MyZone --add-service=https
firewall-cmd --zone=MyZone --permanent --add-service=https
firewall-cmd --reload
```

If --zone is omitted, the default zone (you can check with firewall-cmd --get-default-zone) is used.

To remove the rule, replace the word **add** with **remove** in the above commands.

Example 5: IP / Port forwarding

First off, you need to find out if masquerading is enabled for the desired zone:

```
firewall-cmd --zone=MyZone --query-masquerade
```

In the image below (Fig. 7), we can see that masquerading is enabled for the **external** zone, but not for **public**:

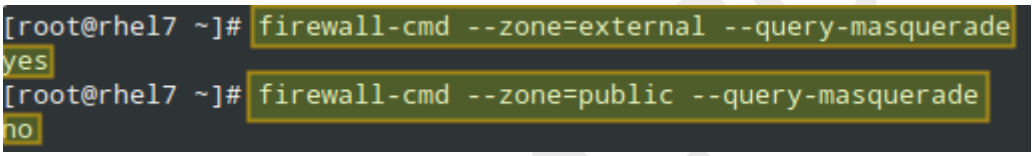


Figure 7: Viewing masquerading status

You can either enable masquerading for **public**:

```
firewall-cmd --zone=public --add-masquerade
```

or use masquerading in **external**. Here's what we would do to replicate Example 3 with firewalld:

```
firewall-cmd --zone=external --add-forward-
port=port=631:proto=tcp:toport=631:toaddr=192.168.0.10
```

And don't forget to reload the firewall.

```
firewall-cmd --reload
```

You can find further examples on Part 10 of the RHCSA series, where we explained how to allow or disable the ports that are usually used by a web server and a ftp server, and how to change the corresponding rule when the default port for those services are changed. In addition, you may want to refer to the firewalld wiki for further examples.

Summary

In this article we have explained what a firewall is, what are the available services to implement one in RHEL 7, and provided a few examples that can help you get started with this task. If you have any comments, suggestions, or questions, feel free to contact us.

Chapter 12: Automating RHEL installations using Kickstart

Linux servers are rarely standalone boxes. Whether it is in a datacenter or in a lab environment, chances are that you have had to install several machines that will interact one with another in some way. If you multiply the time that it takes to install Red Hat Enterprise Linux 7 manually on a single server by the number of boxes that you need to set up, this can lead to a rather lengthy effort that can be avoided through the use of an unattended installation tool known as **kickstart**. In this article we will show what you need to use this utility so that you can forget about babysitting servers during the installation process.

Preparing for a kickstart installation

To perform a kickstart installation, we need to follow these steps:

1. **Create a Kickstart file**, a plain text file with several predefined configuration options.
2. **Make the Kickstart file available on removable media, a hard drive or a network location**. The client will use the rhel-server-7.0-x86_64-boot.iso file, whereas you will need to make the full ISO image (rhel-server-7.0-x86_64-dvd.iso) available from a network resource, such as a HTTP of FTP server (in our present case, we will use another RHEL 7 box with IP 192.168.0.18).
3. Start the Kickstart installation

To create a kickstart file, login to your Red Hat Customer Portal account, and use the Kickstart configuration tool to choose the desired installation options. Read each one of them carefully before scrolling down, and choose what best fits your needs (see Fig. 1):

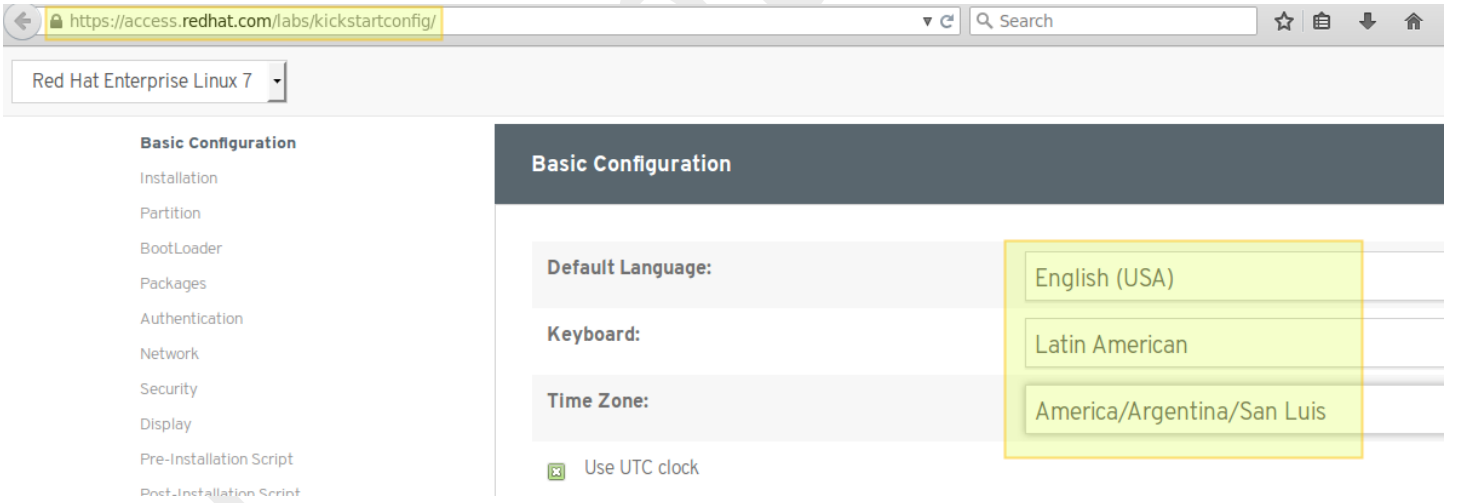


Figure 1: Kickstart configuration tool

If you specify that the installation should be performed either through HTTP, FTP, or NFS, make sure the firewall on the server allows those services.

Although you can use the Red Hat online tool to create a kickstart file, you can also create it manually using the following lines as reference. You will notice, for example, that the installation process will be in English, using the **latin american** keyboard layout and the **America/Argentina/San_Luis** time zone (you can change this as per your needs):

```
lang en_US
keyboard la-latin1
timezone America/Argentina/San_Luis --isUtc
rootpw $1$5sOtDvRo$In4KTmX7OmcOW9HUvWtfn0 --iscrypted
```

```
#platform x86, AMD64, or Intel EM64T
text
url --url=http://192.168.0.18//kickstart/media
bootloader --location=mbr --append="rhgb quiet crashkernel=auto"
zerombr
clearpart --all --initlabel
autopart
auth --passalgo=sha512 --useshadow
selinux --enforcing
firewall --enabled
firstboot --disable
%packages
@base
@backup-server
@print-server
%end
```

In the online configuration tool, use 192.168.0.18 for **HTTP Server** and /kickstart/tecmint.bin for **HTTP Directory** in the Installation section after selecting **HTTP** as installation source. Finally, click the ***Download*** button at the right top corner to download the kickstart file.

In the kickstart sample file above, you need to pay careful attention to

```
url --url=http://192.168.0.18//kickstart/media
```

That directory is where you need to extract the contents of the DVD or ISO installation media.

Before doing that, we will mount the ISO installation file in /media/rhel as a loop device (see Fig. 2):

```
mount -o loop /var/www/html/kickstart/rhel-server-7.0-x86_64-dvd.iso
/media/rhel
```

```
[root@rhel7 ~]# mount -o loop /var/www/html/kickstart/rhel-server-7.0-x86_64-dvd.iso /media/rhel
mount: /dev/loop0 is write-protected, mounting read-only
[root@rhel7 ~]#
```

Figure 2: Mounting the ISO installation file

Next, copy all the contents of /media/rhel to /var/www/html/kickstart/media:

```
cp -R /media/rhel /var/www/html/kickstart/media
```

When you're done, the directory listing and disk usage of /var/www/html/kickstart/media should look as follows (see Fig. 3):

```
[root@rhel7 ~]# ls /var/www/html/kickstart/media
addons  EFI  EULA  GPL  images  isolinux  LiveOS  media.repo  Packages  release-notes  repodata  RPM-GP
[root@rhel7 ~]# du -sch /var/www/html/kickstart/media
3.6G    /var/www/html/kickstart/media
3.6G    total
[root@rhel7 ~]#
```

Figure 3: Kickstart media directory listing

Now we're ready to kick off the kickstart installation.

Regardless of how you choose to create the kickstart file, it's always a good idea to check its syntax before proceeding with the installation. To do that, install the **pykickstart** package

```
yum update && yum install pykickstart
```

And then use the ksvalidator utility to check the file:

```
ksvalidator /var/www/html/kickstart/tecmint.bin
```

If the syntax is correct, you will not get any output, whereas if there's an error in the file, you will get a warning notice indicating the line where the syntax is not correct or unknown.

The installation

To start, boot your client using the **rhel-server-7.0-x86_64-boot.iso file**. When the initial screen appears, select Install Red Hat Enterprise Linux 7.0 and press the Tab key to append the following stanza and press Enter (refer to Fig. 4):

```
inst.ks=http://192.168.0.18/kickstart/tecmint.bin
```

Figure 4: Customizing the RHEL boot stanza

where **tecmint.bin** is the kickstart file created earlier.

When you press Enter, the automated installation will begin, and you will see the list of packages that are being installed (the number and the names will differ depending on your choice of programs and package groups, as seen in Fig. 5):

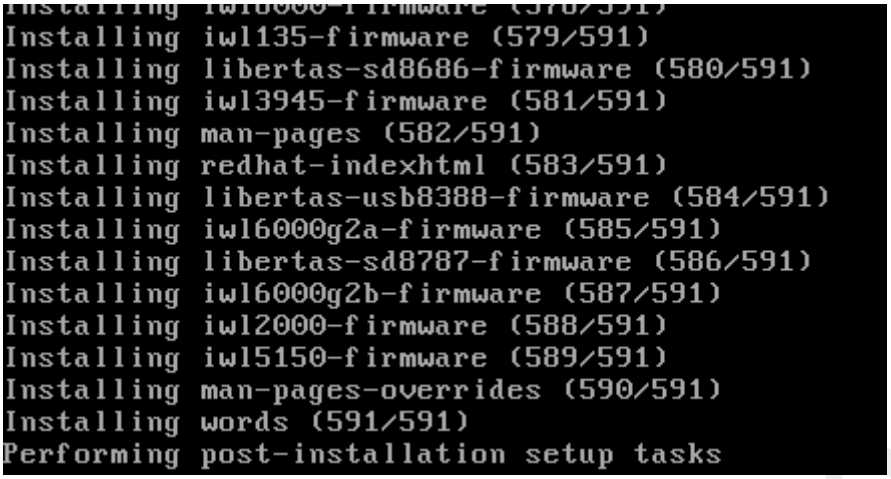


Figure 5: The automated installation

When the automated process ends, you will be prompted to remove the installation media and then you will be able to boot into your newly installed system (see Fig. 6):

Figure 6: RHEL is ready for use

Although you can create your kickstart files manually as we mentioned earlier, you should consider using the recommended approach whenever possible. You can either use the online configuration tool, or the **anaconda-ks.cfg** file that is created by the installation process in root's home directory. This file actually is a kickstart file, so you may want to install the first box manually with all the desired options (maybe modify the logical volumes layout or the file system on top of each one) and then use the resulting anaconda-ks.cfg file to automate the installation of the rest.

In addition, using the online configuration tool or the anaconda-ks.cfg file to guide future installations will allow you to perform them using an encrypted root password out-of-the-box.

Summary

Now that you know how to create kickstart files and how to use them to automate the installation of Red Hat Enterprise Linux 7 servers, you can forget about babysitting the installation process. This will give you time to do other things, or perhaps some leisure time if you're lucky.

Chapter 13: SELinux

In this book we have explored in detail at least two access control methods: standard ugo/rwx permissions and access control lists. Although necessary as first level permissions and access control mechanisms, they have some limitations that are addressed by Security Enhanced Linux (aka SELinux for short).

One of such limitations is that a user can expose a file or directory to a security breach through a poorly elaborated **chmod** command and thus cause an unexpected propagation of access rights. As a result, any process started by that user can do as it pleases with the files owned by the user, where finally a malicious or otherwise compromised software can achieve root-level access to the entire system.

With those limitations in mind, the United States National Security Agency (NSA) first devised SELinux, a flexible mandatory access control method, to restrict the ability of processes to access or perform other operations on system objects (such as files, directories, network ports, etc) to the least permission model, which can be modified later as needed. In few words, each element of the system is given only the access required to function.

In RHEL 7, SELinux is incorporated into the kernel itself and is enabled in Enforcing mode by default. In this article we will explain briefly the basic concepts associated with SELinux and its operation.

SELinux modes

SELinux can operate in three different ways:

- **Enforcing**: SELinux denies access based on SELinux policy rules, a set of guidelines that control the security engine.
- **Permissive**: SELinux does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode.
- **Disabled** (self-explanatory).

The getenforce command displays the current mode of SELinux, whereas setenforce (followed by a 1 or a 0) is used to change the mode to Enforcing or Permissive, respectively, during the current session only. In order to achieve persistence across logouts and reboots, you will need to edit the /etc/selinux/config file and set the SELINUX variable to either **enforcing**, **permissive**, or **disabled** (see Fig. 1):

```
[root@rhel7 ~]# getenforce
Enforcing
[root@rhel7 ~]# setenforce 0
[root@rhel7 ~]# getenforce
Permissive
[root@rhel7 ~]# setenforce 1
[root@rhel7 ~]# getenforce
Enforcing
[root@rhel7 ~]#
```

```
[root@rhel7 ~]# cat /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
```

Figure 1: SELinux modes

Typically you will use **setenforce** to toggle between SELinux modes (enforcing to permissive and back) as a first troubleshooting step. If SELinux is currently set to enforcing while you're experiencing a certain problem, and the same goes away when you set it to permissive, you can be confident you're looking at a SELinux permissions issue.

SELinux contexts

A SELinux context consists of an access control environment where decisions are made based on SELinux user, role, and type (and optionally a level):

- A SELinux user complements a regular Linux user account by mapping it to a SELinux user account, which in turn is used in the SELinux context for processes in that session, in order to explicitly define their allowed roles and levels.
- The concept of role acts as an intermediary between domains and SELinux users in that it defines which process domains and file types can be accessed. This will shield your system against vulnerability to privilege escalation attacks.
- A type defines an SELinux file type or an SELinux process domain. Under normal circumstances, processes are prevented from accessing files that other processes use, and and from accessing other processes, thus access is only allowed if a specific SELinux policy rule exists that allows it.

Let's see how all of that works through the following examples.

Example 1: Changing the default port for the sshd daemon

Changing the default port where sshd listens on is one of the first security measures to secure your server against external attacks. Let's edit the /etc/ssh/sshd_config file and set the port to 9999:

```
Port 9999
```

Save the changes, and restart sshd (see Fig. 2):

```
[root@rhel7 ~]# systemctl restart sshd
[root@rhel7 ~]# systemctl status sshd
sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: activating (auto-restart) (Result: exit-code) since Mon 2015-05-25 16
  Process: 3583 ExecStart=/usr/sbin/sshd -D $OPTIONS (code=exited, status=255)
 Main PID: 3583 (code=exited, status=255)

May 25 16:03:23 rhel7 systemd[1]: sshd.service: main process exited, code=exited
May 25 16:03:23 rhel7 systemd[1]: Unit sshd.service entered failed state.
[root@rhel7 ~]#
```

Figure 2: Changing the sshd port results in an error

As you can see, sshd has failed to start. But what happened?

A quick inspection of /var/log/audit/audit.log indicates that sshd has been denied permissions to start on port 9999 (SELinux log messages include the word "AVC" so that they might be easily identified from other messages) because that is a reserved port for the JBoss Management service (see Fig. 3):

```
cat /var/log/audit/audit.log | grep AVC | tail -1
```

```
[root@rhel7 ~]# cat /var/log/audit/audit.log | grep AVC | tail -1
type=AVC msg=audit(1432584371.456:568): avc:  denied  { name_bind } for  pid=3599 comm="sshd" src=9999 scont
u:object_r:jboss_management_port_t:s0 tclass=tcp_socket
[root@rhel7 ~]#
```

Figure 3: Inspecting the audit.log file

At this point you could disable SELinux (but don't!) as explained earlier and try to start sshd again, and it should work. However, the semanage utility can tell us what we need to change in order for us to be able to start sshd in whatever port we choose without issues.

Run

```
semanage port -l | grep ssh
```

to get a list of the ports where SELinux allows sshd to listen on (see Fig. 4).

```
[root@rhel7 ~]# semanage port -l | grep ssh
ssh_port_t                     tcp      22
[root@rhel7 ~]# getenforce
Enforcing
[root@rhel7 ~]# systemctl is-active sshd
activating
[root@rhel7 ~]# systemctl restart sshd
[root@rhel7 ~]# systemctl is-active sshd
activating
[root@rhel7 ~]# semanage port -a -t ssh_port_t -p tcp 9999
ValueError: Port tcp/9999 already defined
[root@rhel7 ~]# semanage port -l | grep 9999
jboss_management_port_t        tcp      4712, 4447, 7600, 9123, 9990, 9999, 18001
```

sshd will not start as long as the port where we told it to listen on is not allowed by the SELinux ssh_port_t context

Figure 4: Enabling SSH to listen on a different port

So let's change the port in /etc/ssh/sshd_config to Port 9998, add the port to the **ssh_port_t** context, and then restart the service (see Fig. 5):

```
semanage port -a -t ssh_port_t -p tcp 9998
systemctl restart sshd
systemctl is-active sshd
```

```
[root@rhel7 ~]# semanage port -a -t ssh_port_t -p tcp 9998
[root@rhel7 ~]# systemctl restart sshd
[root@rhel7 ~]# systemctl is-active sshd
active
[root@rhel7 ~]#
```

Figure 5: Changing the SELinux context to allow a change in the sshd port

As you can see, the service was started successfully this time. This example illustrates the fact that SELinux controls the TCP port number to its own port type internal definitions.

Example 2: Allowing httpd to send access sendmail

This is an example of SELinux managing a process accessing another process. If you were to implement mod_security and mod_evasive along with Apache in your RHEL 7 server, you need to allow httpd to access sendmail in order to send a mail notification in the wake of a (D)DoS attack. In the following command, omit the -P flag if you do not want the change to be persistent across reboots (see Fig. 6).

```
setsebool -P httpd_can_sendmail 1
```

```
[root@rhel7 ~]# semanage boolean -l | grep httpd_can_sendmail
httpd_can_sendmail   BEFORE   (off  ,  off)  Allow httpd to can sendmail
[root@rhel7 ~]# setsebool -P httpd_can_sendmail 1
[root@rhel7 ~]# semanage boolean -l | grep httpd_can_sendmail
httpd_can_sendmail   AFTER    (on   ,   on)  Allow httpd to can sendmail
[root@rhel7 ~]#
```

Figure 6: Managing SELinux booleans

As you can tell from the above example, SELinux boolean settings (or just **booleans**) are true / false rules embedded into SELinux policies. You can list all the booleans with **semanage boolean -l**, and alternatively pipe it to grep in order to filter the output.

Example 3: Serving a static site from a directory other than the default one

Suppose you are serving a static website using a different directory than the default one (/var/www/html), say /websites (this could be the case if you're storing your web files in a shared network drive, for example, and need to mount it at /websites).

a) Create an index.html file inside /websites with the following contents:

```
<html>
<h2>SELinux test</h2>
</html>
```

If you do

```
ls -lZ /websites/index.html
```

you will see that the index.html file has been labeled with the **default_t** SELinux type, which Apache can't access (see Fig. 7):

```
[root@rhel7 websites]# ls -lZ
-rw-r--r--. root root unconfined_u:object_r:default_t:s0 index.html
```

Figure 7: Viewing SELinux labels

b) Change the **DocumentRoot** directive in /etc/httpd/conf/httpd.conf to /websites and don't forget to update the corresponding Directory block. Then, restart Apache.

c) Browse to **http://<web server IP address>**, and you should get a 503 Forbidden HTTP response.

d) Next, change the label of /websites, recursively, to the **httpd_sys_content_t** type in order to grant Apache read-only access to that directory and its contents:

```
semanage fcontext -a -t httpd_sys_content_t "/websites(/.*)?"
```

e) Finally, apply the SELinux policy created in d):

```
restorecon -R -v /websites
```

Now restart Apache and browse to **http://<web server IP address>** again and you will see the html file displayed correctly (see Fig. 8):

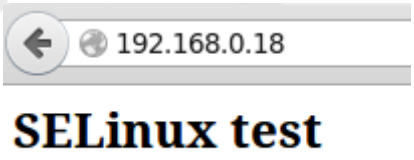


Figure 8: Serving static sites from a directory other than the default

Summary

In this article we have gone through the basics of SELinux. Note that due to the vastness of the subject, a full detailed explanation is not possible in a single article, but we believe that the principles outlined in this guide will help you to move on to more advanced topics should you wish to do so. If I may, let me recommend two essential resources: the NSA SELinux page and the RHEL 7 SELinux User's and Administrator's guide.

Chapter 14: LDAP

We will begin this article by outlining some LDAP basics (what it is, where it is used and why) and show how to set up a LDAP server and configure a client to authenticate against it using Red Hat Enterprise Linux 7 systems. As we will see, there are several other possible application scenarios, but in this guide we will focus entirely on LDAP-based authentication. In addition, please keep in mind that due to the vastness of the subject, we will only cover its basics here, but you can refer to the documentation outlined in the summary for more in-depth details.

For the same reason, you will note that I have decided to leave out several references to man pages of LDAP tools for the sake of brevity, but the corresponding explanations are at a fingertip's distance (**man ldapadd**, for example).

That said, let's get started.

Test environment

Our test environment consists of two RHEL 7 boxes:

- Server: 192.168.0.18. FQDN: rhel7.mydomain.com
- Client: 192.168.0.20. FQDN: ldapclient.mydomain.com

So… what is LDAP?

LDAP stands for **Lightweight Directory Access Protocol** and consists in a set of protocols that allows a client to access, over a network, centrally stored information (such as a directory of login shells, absolute paths to home directories, and other typical system user information, for example) that should be accessible from different places or available to a large number of end users (another example would be a directory of home addresses and phone numbers of all employees in a company).

Keeping such (and more) information centrally means it can be more easily maintained and accessed by everyone who has been granted permissions to use it.

The following diagram offers a simplified diagram of LDAP, and is described below in greater detail (see Fig. 1):

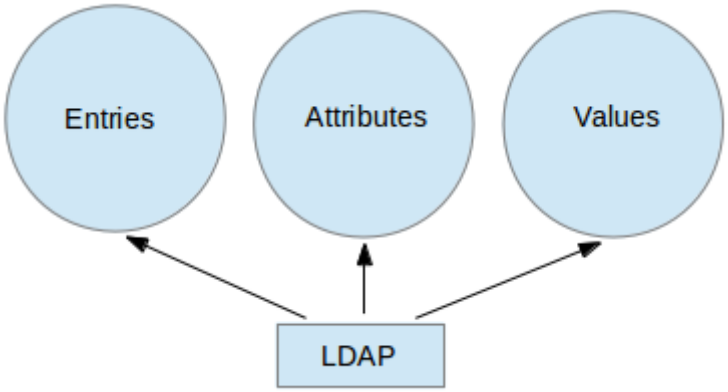


Figure 1: LDAP simplified diagram

- An **entry** in a LDAP directory represents a single unit or information and is uniquely identified by what is called a Distinguished Name.
- An **attribute** is a piece of information associated with an entry (for example, addresses, available contact phone numbers, and email addresses).
- Each attribute is assigned one or more **values** consisting in a space-separated list. A value that is unique per entry is called a Relative Distinguished Name.

That being said, let's proceed with the server and client installations.

Installing and configuring a LDAP server and a LDAP client

In RHEL 7, LDAP is implemented by **OpenLDAP**. To install the server and client, use the following commands, respectively:

```
yum update && yum install openldap openldap-clients openldap-servers
yum update && yum install openldap openldap-clients nss-pam-ldapd
```

Once the installation is complete, there are some things we look at. The following steps should be performed on the server alone, unless explicitly noted:

1) Make sure SELinux does not get in the way by enabling the following booleans persistently, both on the server and the client:

```
setsebool -P allow_ypbind=0 authlogin_nsswitch_use_ldap=0
```

where **allow_ypbind** is required for LDAP-based authentication, and **authlogin_nsswitch_use_ldap** may be needed by some applications.

2) Enable and start the service:

```
systemctl enable slapd.service
systemctl start slapd.service
```

Keep in mind that you can also disable, restart, or stop the service with systemctl as well:

```
systemctl disable slapd.service
systemctl restart slapd.service
systemctl stop slapd.service
```

3) Since the slapd service runs as the ldap user (which you can verify with ps -e -o pid,uname,comm | grep slapd), such user should own the /var/lib/ldap directory in order for the server to be able to modify entries created by administrative tools that can only be run as root (more on this in a minute). Before changing the ownership of this directory recursively, copy the sample database configuration file for slapd into it:

```
cp /usr/share/openldap-servers/DB_CONFIG.example /var/lib/ldap/DB_CONFIG
chown -R ldap:ldap /var/lib/ldap
```

4) Set up an OpenLDAP administrative user and assign a password (see Fig. 2):

```
slappasswd
```

as shown in the next image:

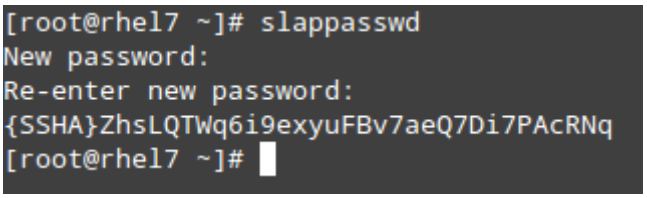


Figure 2: Assigning a password for a LDAP administrative user

and create an LDIF file (ldaprootpasswd.ldif) with the following contents:

```
dn: olcDatabase={0}config,cn=config
changetype: modify
add: olcRootPW
olcRootPW: {SSHA}PASSWORD
```

where:

- **PASSWORD** is the hashed string obtained earlier.
- **cn=config** indicates global config options.
- **olcDatabase** indicates a specific database instance name and can be typically found inside /etc/openldap/slapd.d/cn=config.

Referring to the theoretical background provided earlier, the ldaprootpasswd.ldif file will add an entry to the LDAP directory. In that entry, each line represents an attribute: value pair (where dn, changetype, add, and olcRootPW are the attributes and the strings to the right of each colon are their corresponding values). You may want to keep this in mind as we proceed further, and please note that we are using the same Common Names (**cn=**) throughout the rest of this article, where each step depends on the previous one.

5) Now, add the corresponding LDAP entry by specifying the URI referring to the ldap server, where only the protocol/host/port fields are allowed.

```
ldapadd -H ldapi:/// -f ldaprootpasswd.ldif
```

The output should be similar to Figure 3:

```
[root@rhel7 ~]# ldapadd -H ldapi:/// -f ldaprootpasswd.ldif
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
modifying entry "olcDatabase={0}config,cn=config"

[root@rhel7 ~]#
```

Figure 3: Adding a LDAP entry

and import some basic LDAP definitions from the /etc/openldap/schema directory (refer to Fig. 4):

```
for def in cosine.ldif nis.ldif inetorgperson.ldif; do ldapadd -H
ldapi:/// -f /etc/openldap/schema/$def; done
```

```
[root@rhel7 ~]# for def in cosine.ldif nis.ldif inetorgperson.ldif; do ldapadd -H ldapi:/// -f /etc/openldap/schema/$def; done
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
adding new entry "cn=cosine,cn=schema,cn=config"

SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
adding new entry "cn=nis,cn=schema,cn=config"

SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
adding new entry "cn=inetorgperson,cn=schema,cn=config"

[root@rhel7 ~]#
```

Figure 4: Importing LDAP definitions

6) Have LDAP use your domain in its database

Create another LDIF file, which we will call ldapdomain.ldif, with the following contents, replacing your domain (in the Domain Component **dc=**) and password as appropriate:

```
dn: olcDatabase={1}monitor,cn=config

changetype: modify

replace: olcAccess

olcAccess: {0}to * by
dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth"
```

```
  read by dn.base="cn=Manager,dc=mydomain,dc=com" read by * none

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=mydomain,dc=com

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcRootDN
olcRootDN: cn=Manager,dc=mydomain,dc=com

dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcRootPW
olcRootPW: {SSHA}PASSWORD

dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange by
  dn="cn=Manager,dc=mydomain,dc=com" write by anonymous auth by self write
by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to * by dn="cn=Manager,dc=mydomain,dc=com" write by * read
```

Then load it as follows (see Fig. 5):

```
ldapmodify -H ldapi:/// -f ldapdomain.ldif
```

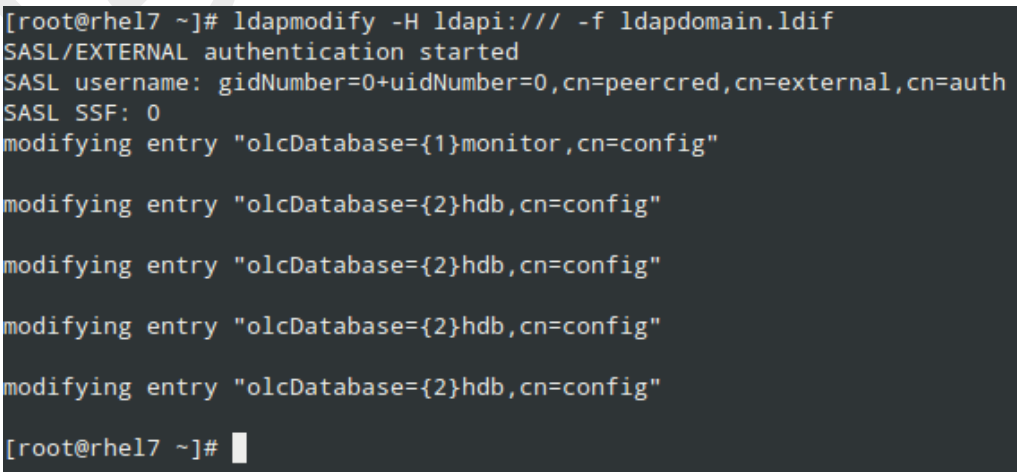


Figure 5: Populating the LDAP database

7) Now it's time to add some entries to our LDAP directory. Attributes and values are separated by a colon (:) in the following file, which we'll name baseldapdomain.ldif:

```
dn: dc=mydomain,dc=com
objectClass: top
objectClass: dcObject
objectclass: organization
o: mydomain com
dc: mydomain

dn: cn=Manager,dc=mydomain,dc=com
objectClass: organizationalRole
cn: Manager
description: Directory Manager

dn: ou=People,dc=mydomain,dc=com
objectClass: organizationalUnit
ou: People

dn: ou=Group,dc=mydomain,dc=com
objectClass: organizationalUnit
ou: Group
```

Add the entries to the LDAP directory (see Fig. 6):

```
ldapadd -x -D cn=Manager,dc=mydomain,dc=com -W -f baseldapdomain.ldif
```

```
[root@rhel7 ~]# ldapadd -x -D cn=Manager,dc=mydomain,dc=com -W -f baseldapdomain.ldif
Enter LDAP Password:
adding new entry "dc=mydomain,dc=com"

adding new entry "cn=Manager,dc=mydomain,dc=com"

adding new entry "ou=People,dc=mydomain,dc=com"

adding new entry "ou=Group,dc=mydomain,dc=com"

[root@rhel7 ~]#
```

Figure 6: Populating the LDAP directory

8) Create a LDAP user called ldapuser (**adduser ldapuser**), then create the definitions for a LDAP group in ldapgroup.ldif (where gidNumber is the GID in /etc/group for ldapuser) and load it:

```
dn: cn=Manager,ou=Group,dc=mydomain,dc=com
objectClass: top
objectClass: posixGroup
gidNumber: 1004
ldapadd -x -W -D "cn=Manager,dc=mydomain,dc=com" -f ldapgroup.ldif
```

9) Add a LDIF file with the definitions for user ldapuser (ldapuser.ldif):

```
dn: uid=ldapuser,ou=People,dc=mydomain,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: ldapuser
uid: ldapuser
uidNumber: 1004
gidNumber: 1004
homeDirectory: /home/ldapuser
userPassword: {SSHA}fiN0YqzbDuDI0Fpqq9UudWmjZQY28S3M
loginShell: /bin/bash
gecos: ldapuser
shadowLastChange: 0
shadowMax: 0
shadowWarning: 0
```

and load it (see Fig. 7):

```
ldapadd -x -D cn=Manager,dc=mydomain,dc=com -W -f ldapuser.ldif
```

```
[root@rhel7 ~]# ldapadd -x -D cn=Manager,dc=mydomain,dc=com -W -f ldapuser.ldif
Enter LDAP Password:
adding new entry "uid=ldapuser,ou=People,dc=mydomain,dc=com"

[root@rhel7 ~]#
```

Figure 7: Loading the definitions for the LDAP user

Likewise, you can delete the user entry you just created:

```
ldapdelete -x -W -D cn=Manager,dc=mydomain,dc=com
"uid=ldapuser,ou=People,dc=mydomain,dc=com"
```

10) Allow communication through the firewall

```
firewall-cmd --add-service=ldap
```

11) Last, but not least, enable the client to authenticate using LDAP

To help us in this final step, we will use the authconfig utility (an interface for configuring system authentication resources).

Using the following command, the home directory for the requested user is created if it doesn't exist after the authentication against the LDAP server succeeds (see Fig. 8):

```
authconfig --enableldap --enableldapauth --ldapserver=rhel7.mydomain.com -
-ldapbasedn="dc=mydomain,dc=com" --enablemkhomedir --update
```

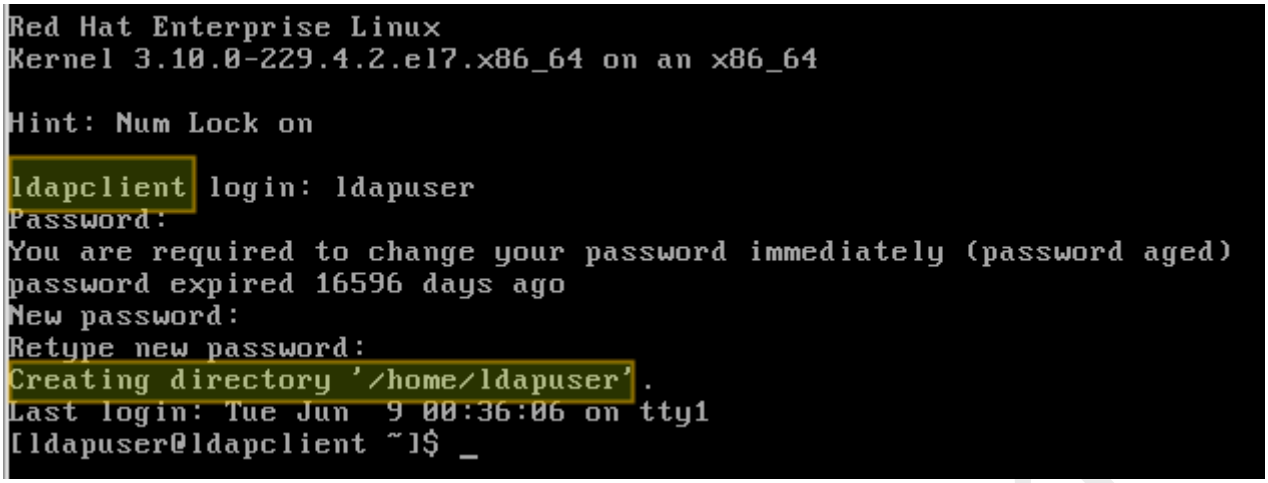


Figure 8: Logging in as the LDAP user

Summary

In this article we have explained how to set up basic authentication against a LDAP server. To further configure the setup described in the present guide, please refer to Chapter 13 in the RHEL 7 System administrator's guide, paying special attention to the security settings using TLS.

Chapter 15: KVM Virtualization

If you look up the word virtualize in a dictionary, you will find that it means “to create a virtual (rather than actual) version of something”. In computing, the term virtualization refers to the possibility of running multiple operating systems simultaneously and isolated one from another, on top of the same physical (hardware) system, known in the virtualization schema as **host**.

Through the use of the virtual machine monitor (also known as **hypervisor**), virtual machines (referred to as **guests**) are provided virtual resources (i.e. CPU, RAM, storage, network interfaces, to name a few) from the underlying hardware.

With that in mind, it is plain to see that one of the main advantages of virtualization is cost savings (in equipment and network infrastructure and in terms of maintenance effort) and a substantial reduction in the physical space required to accommodate all the necessary hardware.

Since this brief how-to cannot cover all virtualization methods, I encourage you to refer to the documentation listed in the summary for further details on the subject. Please keep in mind that the present article is intended to be a starting point to learn the basics of virtualization in RHEL 7 using KVM (Kernel-based Virtual Machine) with command-line utilities, and not an in-depth discussion of the topic.

Verifying hardware requirements and installing packages

In order to set up virtualization, your CPU must support it. You can verify whether your system meets the requirements with the following command:

```
grep -E 'svm|vmx' /proc/cpuinfo
```

In the following screenshot we can see that the current system (with an AMD microprocessor) supports virtualization, as indicated by **svm**. If we had an Intel-based processor, we would see **vmx** instead in the results of the above command (see Fig. 1).

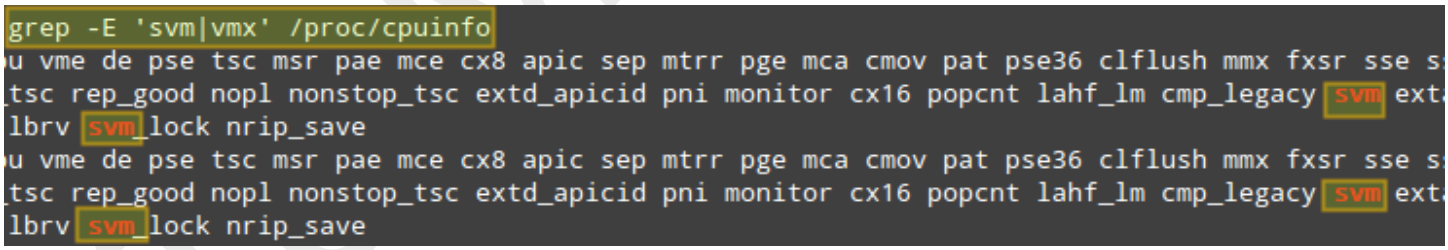


Figure 1: Checking whether your system supports virtualization

In addition, you will need to have virtualization capabilities enabled in the firmware of your host (BIOS or UEFI)

Now install the necessary packages:

- **qemu-kvm** is an open source virtualizer that provides hardware emulation for the KVM hypervisor whereas **qemu-img** provides a command line tool for manipulating disk images.
- **libvirt** includes the tools to interact with the virtualization capabilities of the operating system.
- **libvirt-python** contains a module that permits applications written in Python to use the interface supplied by libvirt.
- **libguestfs-tools**: miscellaneous system administrator command line tools for virtual machines.
- **virt-install**: other command-line utilities for virtual machine administration.

yum update && yum install qemu-kvm qemu-img libvirt libvirt-python libguestfs-tools virt-install

Once the installation completes, make sure you start and enable the libvirtd service:

```
systemctl start libvirtd.service
systemctl enable libvirtd.service
```

By default, each virtual machine will only be able to communicate with the rest in the same physical server and with the host itself. To allow the guests to reach other machines inside our LAN and also the Internet, we need to set up a bridge interface in our host (say br0, for example) by

1) adding the following line to our main NIC configuration (most likely /etc/sysconfig/network-scripts/ifcfg-enp0s3):

```
BRIDGE=br0
```

2) creating the configuration file for br0 (/etc/sysconfig/network-scripts/ifcfg-br0) with these contents (note that you may have to change the IP address, gateway address, and DNS information):

```
DEVICE=br0
TYPE=Bridge
BOOTPROTO=static
IPADDR=192.168.0.18
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
NM_CONTROLLED=no
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=br0
ONBOOT=yes
DNS1=8.8.8.8
DNS2=8.8.4.4
```

Finally,

3) enabling packet forwarding by making, in /etc/sysctl.conf,

```
net.ipv4.ip_forward = 1
```

and loading the changes to the current kernel configuration:

```
sysctl -p
```

Note that you may also need to tell firewalld that this kind of traffic should be allowed. Remember that you can refer to the Chapter 11 in this same book if you need help to do that.

Creating VM images

By default, VM images will be created to /var/lib/libvirt/images and you are strongly advised to not change this unless you really need to, know what you're doing, and want to handle SELinux settings yourself. This means that you need to make sure that you have allocated the necessary space in that filesystem to accommodate your virtual machines.

The following command will create a virtual machine named tecmint-virt01 with 1 virtual CPU, 1 GB (=1024 MB) of RAM, and 20 GB of disk space (represented by /var/lib/libvirt/images/tecmint-virt01.img) using the rhel-server-7.0-x86_64-dvd.iso image located inside /home/gacanepa/ISOs as installation media and the br0 as network bridge:

```
virt-install \
--network bridge=br0
--name tecmint-virt01 \
--ram=1024 \
--vcpus=1 \
--disk path=/var/lib/libvirt/images/tecmint-virt01.img,size=20 \
--graphics none \
--cdrom /home/gacanepa/ISOs/rhel-server-7.0-x86_64-dvd.iso
--extra-args="console=tty0 console=ttyS0,115200"
```

If the installation file was located in a HTTP server instead of an image stored in your disk, you will have to replace the **--cdrom** flag with **--location** and indicate the address of the online repository.

As for the **--graphics none** option, it tells the installer to perform the installation in text-mode exclusively. You can omit that flag if you are using a GUI interface and a VNC window to access the main VM console. Finally, with **--extra-args** we are passing kernel boot parameters to the installer that set up a serial VM console.

The installation should now proceed as a regular (real) server now. If not, please review the steps listed above.

Managing virtual machines

These are some typical administration tasks that you, as a system administrator, will need to perform on your virtual machines. Note that all of the following commands need to be run from your host:

1) List all VMs:

```
virsh list --all
```

From the output of the above command you will have to note the Id for the virtual machine (although it will also return its name and current status) because you will need it for most administration tasks related to a particular VM.

2) Display information about a guest:

```
virsh dominfo [VM Id]
```

3) Start, restart, or stop a guest operating system:

```
virsh start | reboot | shutdown [VM Id]
```

4) Access a VM's serial console if networking is not available and no X server is running on the host:

```
virsh console [VM Id]
```

Note that this will require that you add the serial console configuration information to the /etc/grub.conf file (refer to the argument passed to the **--extra-args** option when the VM was created).

5) Modify assigned memory or virtual CPUs:

First, shutdown the guest:

```
virsh shutdown [VM Id]
```

Edit the VM configuration:

For RAM:

```
virsh edit [VM Id]
```

Then modify

```
<memory>[Memory size here without brackets]</memory>
```

Restart the VM with the new settings:

```
virsh create /etc/libvirt/qemu/tecmint-virt01.xml
```

Finally, change the memory dynamically:

```
virsh setmem [VM Id] [Memory size here without brackets]
```

For CPU:

```
virsh edit [VM Id]
```

Then modify

```
<cpu>[Number of CPUs here without brackets]</cpu>
```

For further commands and details, please refer to table 26.1 in Chapter 26 of the RHEL 5 Virtualization guide (that guide, though a bit old, includes an exhaustive list of **virsh** commands used for guest administration).

Summary

In this article we have covered some basic aspects of virtualization with KVM in RHEL 7, which is both a vast and a fascinating topic, and I hope it will be helpful as a starting guide for you to later explore more advanced subjects found in the official RHEL virtualization getting started and deployment / administration guides.

Chapter 16: Static routing

In this article and the next, we will present basic, yet typical, cases where the principles of static routing, packet filtering, and network address translation come into play. Please note that we will not cover them in depth, but rather organize these contents in such a way that will be helpful to take the first steps and build from there.

Static routing in Red Hat Enterprise Linux 7

One of the wonders of modern networking is the vast availability of devices that can connect groups of computers, whether in relatively small numbers and confined to a single room or several machines in the same building, city, country, or across continents.

However, in order to effectively accomplish this in any situation, network packets need to be routed, or in other words, the path they follow from source to destination must be ruled somehow.

Static routing is the process of specifying a route for network packets other than the default, which is provided by a network device known as the default gateway. Unless specified otherwise through static routing, network packets are directed to the default gateway; with static routing, other paths are defined based on predefined criteria, such as the packet destination.

Let us define the following scenario for this tutorial. We have a Red Hat Enterprise Linux 7 box connecting to router #1 [192.168.0.1] to access the Internet and machines in 192.168.0.0/24. A second router (router #2) has two network interface cards: enp0s3 is also connected to router #1 to access the Internet and to communicate with the RHEL 7 box and other machines in the same network, whereas the other (enp0s8) is used to grant access to the 10.0.0.0/24 network where internal services reside, such as a web and / or database server.

This scenario is illustrated in the diagram below (Fig. 1):

Figure 1: Network diagram for this chapter

In this article we will focus exclusively on setting up the routing table on our RHEL 7 box to make sure that it can both access the Internet through router #1 and the internal network via router #2.

In RHEL 7, you will use the **ip** command to configure and show devices and routing using the command line. These changes can take effect immediately on a running system but since they are not persistent across reboots, we will use **ifcfg-enp0sX** and **route-enp0sX** files inside **/etc/sysconfig/network-scripts** to save our configuration permanently.

To begin, let's print our current routing table (see Fig. 2):

```
ip route show
```

```
[root@rhel7 ~]# ip route show
default via 192.168.0.1 dev enp0s3
169.254.0.0/16 dev enp0s3  scope link  metric 1002
192.168.0.0/24 dev enp0s3  proto kernel  scope link  src 192.168.0.18
[root@rhel7 ~]#
```

Figure 2: Viewing the current routing table

From the output above, we can see the following facts:

- The default gateway's IP address is 192.168.0.1 and can be accessed via the enp0s3 NIC.
- When the system booted up, it enabled the zeroconf route to 169.254.0.0/16 (just in case). In few words, if a machine is set to obtain an IP address through DHCP but fails to do so for some reason, it is automatically assigned an address in this network. Bottom line is, this route will allow us to communicate, also via enp0s3, with other machines who have failed to obtain an IP address from a DHCP server.
- Last, but not least, we can communicate with other boxes inside the 192.168.0.0/24 network through enp0s3, whose IP address is 192.168.0.18.

These are the typical tasks that you would have to perform in such a setting. Unless specified otherwise, the following tasks should be performed in router #2:

Make sure all NICs have been properly installed:

```
ip link show
```

If one of them is down, bring it up:

```
ip link set dev enp0s8 up
```

and assign an IP address in the 10.0.0.0/24 network to it:

```
ip addr add 10.0.0.17 dev enp0s8
```

Oops! We made a mistake in the IP address. We will have to remove the one we assigned earlier and then add the right one (10.0.0.18):

```
ip addr del 10.0.0.17 dev enp0s8
ip addr add 10.0.0.18 dev enp0s8
```

Now, please note that you can only add a route to a destination network through a gateway that is itself already reachable. For that reason, we need to assign an IP address within the 192.168.0.0/24 range to enp0s3 so that our RHEL 7 box can communicate with it:

```
ip addr add 192.168.0.19 dev enp0s3
```

Finally, we will need to enable packet forwarding:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

and stop / disable (just for the time being – until we cover packet filtering in the next article) the firewall:

```
systemctl stop firewalld
systemctl disable firewalld
```

Back in our RHEL 7 box (192.168.0.18), let's configure a route to 10.0.0.0/24 through 192.168.0.19 (enp0s3 in router #2):

```
ip route add 10.0.0.0/24 via 192.168.0.19
```

After that, the routing table looks as follows (see Fig. 3):

```
[root@rhel7 ~]# ip route show
default via 192.168.0.1 dev enp0s3
10.0.0.0/24 via 192.168.0.19 dev enp0s3
169.254.0.0/16 dev enp0s3  scope link  metric 1002
192.168.0.0/24 dev enp0s3  proto kernel  scope link  src 192.168.0.18
[root@rhel7 ~]#
```

Figure 3: Routing table after making changes

Likewise, add the corresponding route in the machine(s) you're trying to reach in 10.0.0.0/24:

```
ip route add 192.168.0.0/24 via 10.0.0.18
```

You can test for basic connectivity using ping:

In the RHEL 7 box, run

```
ping -c 4 10.0.0.20
```

where 10.0.0.20 is the IP address of a web server in the 10.0.0.0/24 network.

In the web server (10.0.0.20), run

```
ping -c 192.168.0.18
```

where 192.168.0.18 is, as you will recall, the IP address of our RHEL 7 machine.

Alternatively, we can use **tcpdump** (you may need to install it with `yum install tcpdump`) to check the 2-way communication over TCP between our RHEL 7 box and the web server at 10.0.0.20. To do so, let's start the logging in the first machine with

```
tcpdump -qnnvvv -i enp0s3 host 10.0.0.20
```

and from another terminal in the same system let's telnet to port 80 in the web server (assuming Apache is listening on that port; otherwise, indicate the right port in the following command):

```
telnet 10.0.0.20 80
```

The **tcpdump** log should look as follows (see Fig. 4):

```
[root@rhel7 ~]# tcpdump -qnnvvv -i enp0s3 host 10.0.0.20
tcpdump: listening on enp0s3, link-type EN10MB (Ethernet), capture size 65535 bytes
23:12:22.682531 IP (tos 0x10, ttl 64, id 61663, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.0.18.36676 > 10.0.0.20.80: tcp 0
23:12:22.683875 IP (tos 0x0, ttl 63, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    10.0.0.20.80 > 192.168.0.18.36676: tcp 0
23:12:22.683965 IP (tos 0x10, ttl 64, id 61664, offset 0, flags [DF], proto TCP (6), length 52)
```

Figure 4: Tcpdump log

where the connection has been properly initialized, as we can tell by looking at the 2-way communication between our RHEL 7 box (192.168.0.18) and the web server (10.0.0.20).

Please remember that these changes will go away when you restart the system. If you want to make them persistent, you will need to edit (or create, if they don't already exist) the following files, in the same systems where we performed the above commands.

Though not strictly necessary for our test case, you should know that /etc/sysconfig/network contains system-wide network parameters. A typical /etc/sysconfig/network looks as follows:

```
# Enable networking on this system?
NETWORKING=yes
# Hostname. Should match the value in /etc/hostname
HOSTNAME=yourhostnamehere
# Default gateway
GATEWAY=XXX.XXX.XXX.XXX
# Device used to connect to default gateway. Replace X with the
appropriate number.
GATEWAYDEV=enp0sX
```

When it comes to setting specific variables and values for each NIC (as we did for router #2), you will have to edit **/etc/sysconfig/network-scripts/ifcfg-enp0s3** and **/etc/sysconfig/network-scripts/ifcfg-enp0s8**.

Following our case,

```
TYPE=Ethernet
```

```
BOOTPROTO=static
IPADDR=192.168.0.19
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
NAME=enp0s3
ONBOOT=yes
```

and

```
TYPE=Ethernet
BOOTPROTO=static
IPADDR=10.0.0.18
NETMASK=255.255.255.0
GATEWAY=10.0.0.1
NAME=enp0s8
ONBOOT=yes
```

for enp0s3 and enp0s8, respectively.

As for routing in our client machine (192.168.0.18), we will need to edit **/etc/sysconfig/network-scripts/route-enp0s3**:

```
10.0.0.0/24 via 192.168.0.19 dev enp0s3
```

Now reboot your system and you should see that route in your table.

Summary

In this article we have covered the essentials of static routing in Red Hat Enterprise Linux 7. Although scenarios may vary, the case presented here illustrates the required principles and the procedures to perform this task. Before wrapping up, I would like to suggest you to take a look at Chapter 4 of the Securing and Optimizing Linux section in The Linux Documentation Project site for further details on the topics covered here.

In the next article we will talk about packet filtering and network address translation to sum up the networking basic skills needed for the RHCE certification.

Chapter 17: Packet filtering and NAT

As promised in the last chapter, in this article we will begin by introducing the principles of packet filtering and network address translation (NAT) in Red Hat Enterprise Linux 7, before diving into setting runtime kernel parameters to modify the behavior of a running kernel if certain conditions change or needs arise.

Packet filtering in RHEL 7

When we talk about packet filtering, we refer to a process performed by a firewall in which it reads the header of each data packet that attempts to pass through it. Then, it filters the packet by taking the required action based on rules that have been previously defined by the system administrator.

As you probably know, beginning with RHEL 7, the default service that manages firewall rules is **firewalld**. Like **iptables**, it talks to the **netfilter** module in the Linux kernel in order to examine and manipulate network packets. Unlike iptables, updates can take effect immediately without interrupting active connections -you don't even have to restart the service.

Another advantage of firewalld is that it allows us to define rules based on preconfigured service names (more on that in a minute).

In the last article, we used the following scenario (see Fig. 1):

Figure 1: Network diagram from last chapter

However, you will recall that we disabled the firewall on router #2 to simplify the example since we had not covered packet filtering yet. Let's see now how we can enable incoming packets destined for a specific service or port in the destination.

First, let's add a permanent rule to allow inbound traffic in enp0s3 (192.168.0.19) to enp0s8 (10.0.0.18):

```
firewall-cmd --permanent --direct --add-rule ipv4 filter FORWARD 0 -i enp0s3 -o enp0s8 -j ACCEPT
```

The above command will save the rule to /etc/firewalld/direct.xml (see Fig. 2):

```
[root@router2 ~]# cat /etc/firewalld/direct.xml
<?xml version="1.0" encoding="utf-8"?>
<direct>
  <rule priority="0" table="filter" ipv="ipv4" chain="FORWARD">-i enp0s3 -o enp0s8 -j ACCEPT</rule>
</direct>
[root@router2 ~]#
```

Figure 2: Allowing traffic from one network interface to another

Then enable the rule for it to take effect immediately:

```
firewall-cmd --direct --add-rule ipv4 filter FORWARD 0 -i enp0s3 -o enp0s8 -j ACCEPT
```

Now you can telnet to the web server from the RHEL 7 box and run tcpdump again to monitor the TCP traffic between the two machines, this time with the firewall in router #2 enabled.

```
telnet 10.0.0.20 80
```

```
tcpdump -qnnvvv -i enp0s3 host 10.0.0.20
```

What if you want to only allow incoming connections to the web server (port 80) from 192.168.0.18 and block connections from other sources in the 192.168.0.0/24 network?

In the web server's firewall, add the following rules:

```
firewall-cmd --add-rich-rule 'rule family="ipv4" source
address="192.168.0.18/24" service name="http" accept'

firewall-cmd --add-rich-rule 'rule family="ipv4" source
address="192.168.0.18/24" service name="http" accept' --permanent

firewall-cmd --add-rich-rule 'rule family="ipv4" source
address="192.168.0.0/24" service name="http" drop'

firewall-cmd --add-rich-rule 'rule family="ipv4" source
address="192.168.0.0/24" service name="http" drop' --permanent
```

Now you can make HTTP requests to the web server, from 192.168.0.18 and from some other machine in 192.168.0.0/24. In the first case the connection should complete successfully, whereas in the second it will eventually timeout.

To do so, any of the following commands will do the trick:

```
telnet 10.0.0.20 80

wget 10.0.0.20
```

I strongly advise you to check out the Firewalld Rich Language documentation in the Fedora Project Wiki for further details on rich rules.

Network Address Translation in RHEL 7

Network Address Translation (NAT) is the process where a group of computers (it can also be just one of them) in a private network are assigned an unique public IP address. As result, they are still uniquely identified by their own private IP address inside the network but to the outside they all "seem" the same. In addition, NAT makes it possible that computers inside a network sends requests to outside resources (like the Internet) and have the corresponding responses be sent back to the source system only.

Let's now consider the following scenario (see Fig. 3):

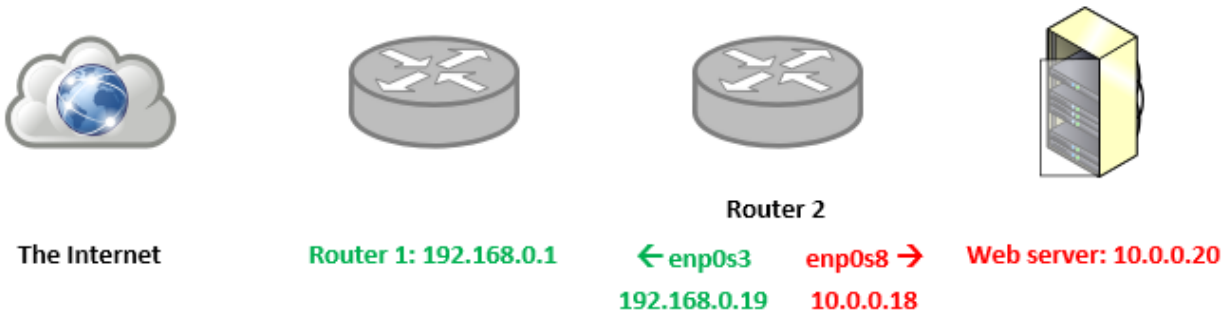


Figure 3: NAT scenario

In router #2, we will move the enp0s3 interface to the external zone, and enp0s8 to the internal zone, where masquerading, or NAT, is enabled by default:

```
firewall-cmd --list-all --zone=external

firewall-cmd --change-interface=enp0s3 --zone=external

firewall-cmd --change-interface=enp0s3 --zone=external --permanent
```

```
firewall-cmd --change-interface=enp0s8 --zone=internal
firewall-cmd --change-interface=enp0s8 --zone=internal --permanent
```

For our current setup, the internal zone -along with everything that is enabled in it- will be the default zone:

```
firewall-cmd --set-default-zone=internal
```

Next, let's reload firewall rules and keep state information:

```
firewall-cmd --reload
```

Finally, let's add router #2 as default gateway in the web server:

```
ip route add default via 10.0.0.18
```

You can now verify that you can ping router #1 and an external site (**tecmint.com**, for example) from the web server (see Fig. 4):

```
[root@webserver ~]# ping -c 2 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=63 time=1.91 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=63 time=1.12 ms

--- 192.168.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.121/1.516/1.912/0.397 ms
[root@webserver ~]# ping -c 2 tecmint.com
PING tecmint.com (212.71.234.61) 56(84) bytes of data.
64 bytes from mail.tecmint.com (212.71.234.61): icmp_seq=1 ttl=51 time=323 ms
64 bytes from mail.tecmint.com (212.71.234.61): icmp_seq=2 ttl=51 time=257 ms

--- tecmint.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1259ms
rtt min/avg/max/mdev = 257.981/290.800/323.619/32.819 ms
[root@webserver ~]# _
```

Figure 4: Verifying NAT

Setting kernel runtime parameters in RHEL 7

In Linux, you are allowed to change, enable, and disable the kernel runtime parameters, and RHEL is no exception. The /proc/sys interface (**sysctl**) lets you set runtime parameters on-the-fly to modify the system's behavior without much hassle when operating conditions change.

To do so, the echo shell built-in is used to write to files inside **/proc/sys/<category>**, where **<category>** is most likely one of the following directories:

- **dev**: parameters for specific devices connected to the machine.
- **fs**: filesystem configuration (quotas and inodes, for example)
- **kernel**: kernel-specific configuration.
- **net**: network configuration.
- **vm**: use of the kernel's virtual memory.

To display the list of all the currently available values, run

```
sysctl -a | less
```

In Part 1, we changed the value of the **net.ipv4.ip_forward** parameter by doing

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

in order to allow a Linux machine to act as router.

Another runtime parameter that you may want to set is kernel.sysrq, which enables the Sysrq key in your keyboard to instruct the system to perform gracefully some low-level functions, such as rebooting the system if it has frozen for some reason:

```
echo 1 > /proc/sys/kernel/sysrq
```

To display the value of a specific parameter, use sysctl as follows:

```
sysctl <parameter.name>
```

For example,

```
sysctl net.ipv4.ip_forward
sysctl kernel.sysrq
```

Some parameters, such as the ones mentioned above, require only one value, whereas others (for example, **fs.inode-state**) require multiple values (see Fig. 5):

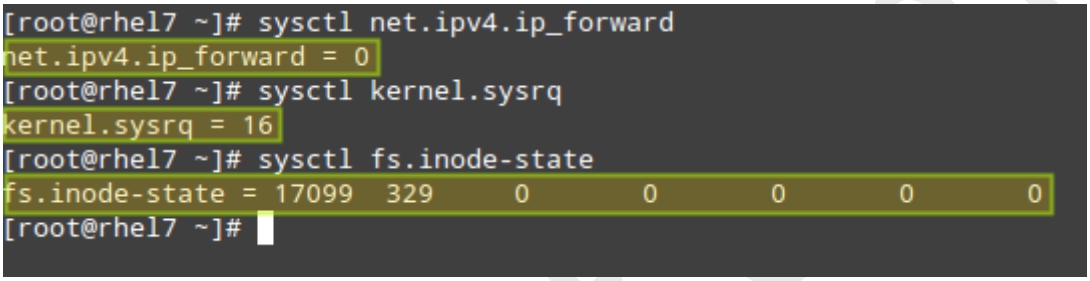


Figure 5: Viewing kernel runtime parameters

In either case, you need to read the kernel's documentation before making any changes.

Please note that these settings will go away when the system is rebooted. To make these changes permanent, we will need to add .conf files inside the /etc/sysctl.d as follows:

```
echo "net.ipv4.ip_forward = 1" > /etc/sysctl.d/10-forward.conf
```

(where the number **10** indicates the order of processing relative to other files in the same directory).

and enable the changes with

```
sysctl -p /etc/sysctl.d/10-forward.conf
```

Summary

In this tutorial we have explained the basics of packet filtering, network address translation, and setting kernel runtime parameters on a running system and persistently across reboots. I hope you have found this information useful, and as always, we look forward to hearing from you!

Chapter 18: Producing and delivering system activity reports

As a system engineer, you will often need to produce reports that show the utilization of your system's resources in order to make sure that 1) they are being utilized optimally, 2) prevent bottlenecks, and 3) ensure scalability, among other reasons.

Besides the well-known native Linux tools that are used to check disk, memory, and CPU usage –to name a few examples-, Red Hat Enterprise Linux 7 provides two additional toolsets to enhance the data you can collect for your reports: **sysstat** and **dstat**. In this article we will describe both, but let's first start by reviewing the usage of the classic tools.

Native Linux tools

With **df**, you will be able to report disk space and inode usage of by filesystem. You need to monitor both because a lack of space will prevent you from being able to save further files (and may even cause the system to crash), just like running out of inodes will mean you can't link further files with their corresponding data structures, thus producing the same effect: you won't be able to save those files to disk (see Figs. 1 and 2).

```
df -h # Display output in human-readable form
df -h --total # Produce a grand total
```

```
[root@rhel7 ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root    28G  8.5G   20G  31% /
devtmpfs                488M     0  488M   0% /dev
tmpfs                   497M     0  497M   0% /dev/shm
tmpfs                   497M  6.6M  491M   2% /run
tmpfs                   497M     0  497M   0% /sys/fs/cgroup
/dev/sda1               497M  191M  307M  39% /boot
[root@rhel7 ~]# df -h --total
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root    28G  8.5G   20G  31% /
devtmpfs                488M     0  488M   0% /dev
tmpfs                   497M     0  497M   0% /dev/shm
tmpfs                   497M  6.6M  491M   2% /run
tmpfs                   497M     0  497M   0% /sys/fs/cgroup
/dev/sda1               497M  191M  307M  39% /boot
total                    30G  8.7G   22G  29% -
[root@rhel7 ~]#
```

Figure 1: Using df to report available disk space

```
df -i # Show inode count by filesystem
df -i --total # Produce a grand total
```

```
[root@rhel7 ~]# df -i
Filesystem              Inodes IUsed    IFree IUse% Mounted on
/dev/mapper/rhel-root 28811264 45596 28765668    1% /
devtmpfs                124888   377   124511    1% /dev
tmpfs                   127185     1   127184    1% /dev/shm
tmpfs                   127185   446   126739    1% /run
tmpfs                   127185    13   127172    1% /sys/fs/cgroup
/dev/sda1               512000   342   511658    1% /boot
[root@rhel7 ~]# df -i --total
Filesystem              Inodes IUsed    IFree IUse% Mounted on
/dev/mapper/rhel-root 28811264 45596 28765668    1% /
devtmpfs                124888   377   124511    1% /dev
tmpfs                   127185     1   127184    1% /dev/shm
tmpfs                   127185   446   126739    1% /run
tmpfs                   127185    13   127172    1% /sys/fs/cgroup
/dev/sda1               512000   342   511658    1% /boot
total                 29829707 46775 29782932    1% -
[root@rhel7 ~]#
```

Figure 2: Using df to report inode usage count

With **du**, you can estimate file space usage by either file, directory, or filesystem.

For example, let's see how much space is used by the /home directory, which includes all of the user's personal files. The first command will return the overall space currently used by the entire **/home** directory, whereas the second will also display a disaggregated list by subdirectory as well (see Fig. 3):

```
du -sch /home
du -sch /home/*
```

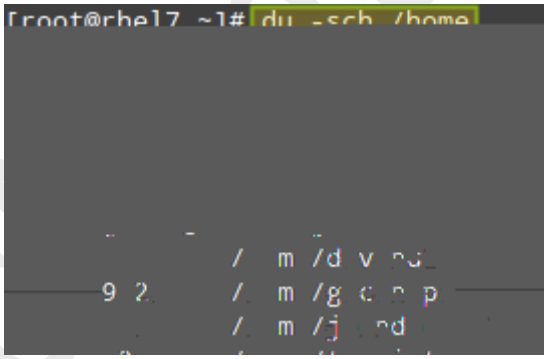


Figure 3: Using du to report disk usage

Another utility that can't be missing from your toolset is **vmstat**. It will allow you to see at a quick glance information about processes, CPU and memory usage, disk activity, and more.

If run without arguments, **vmstat** will return averages since the last reboot. While you may use this form of the command once in a while, it will be more helpful to take a certain amount of system utilization samples, one after another, with a defined time separation between samples.

For example,

```
vmstat 5 10
```

will return 10 samples taken every 5 seconds (see Fig. 4):

```
[root@rhel7 ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 562540    764 347968    0    0    62    23   31   47  0  1 99  0  0
[root@rhel7 ~]# vmstat 5 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 562544    764 347968    0    0    62    23   31   47  0  1 99  0  0
 0  0      0 562548    764 347968    0    0     0     0    8   10  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    8   11  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    9   10  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    8   10  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    8   10  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    7   11  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    8   10  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    9   13  0  0 100  0  0
 0  0      0 562548    764 347968    0    0     0     0    9   11  0  0 100  0  0
[root@rhel7 ~]#
```

Figure 4: Using vmstat to view system stats

As you can see in the above picture, the output of vmstat is divided by columns: **procs** (processes), **memory**, **swap**, **io**, **system**, and **cpu**. The meaning of each field can be found in the **FIELD DESCRIPTION** sections in the man page of **vmstat**.

Where can vmstat come in handy? Let's examine the behavior of the system before and during a **yum update** (see Fig. 5).

```
vmstat -a 1 5
```

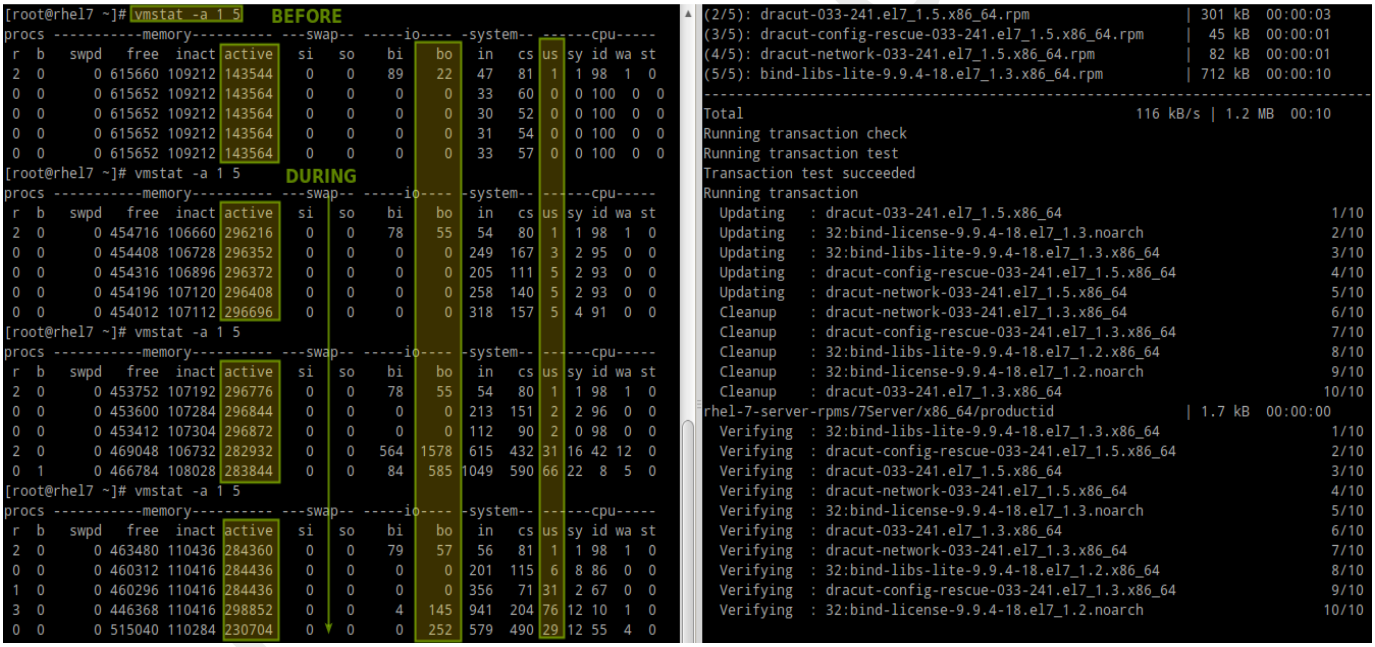


Figure 5: Reporting system resource utilization during an update

Please note that as files are being modified on disk, the amount of **active** memory increases and so does the number of blocks written to disk (**bo**) and the CPU time that is dedicated to user processes (**us**).

Or during the saving process of a large file directly to disk (caused by **dsync**), as shown in Fig. 6:

```
vmstat -a 1 5

dd if=/dev/zero of=dummy.out bs=1M count=1000 oflag=dsync
```

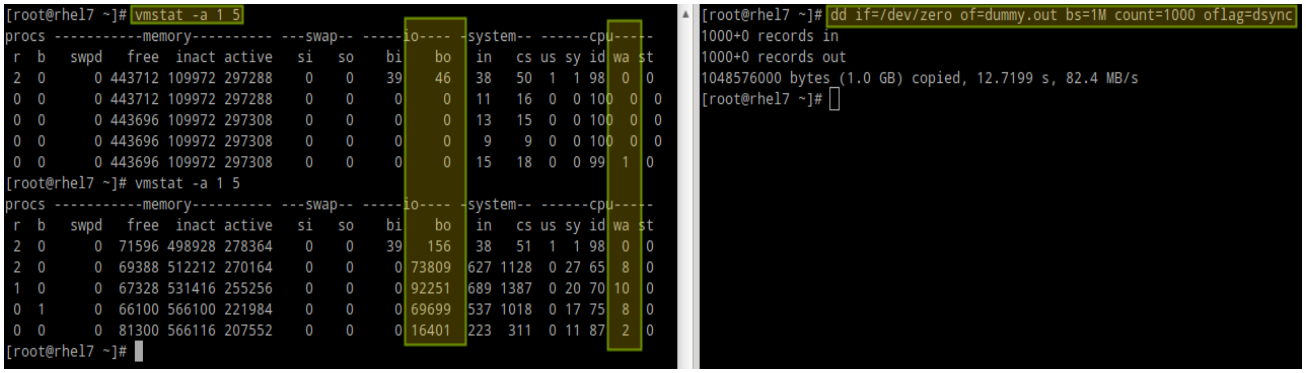


Figure 6: Reporting system resource utilization during a write operation

In this case, we can see a yet larger number of blocks being written to disk (**bo**), which was to be expected, but also an increase of the amount of CPU time that it has to wait for I/O operations to complete before processing tasks (**wa**).

Other tools

As mentioned in the introduction of this chapter, there are other tools that you can use to check the system status and utilization (they are not only provided by Red Hat but also by other major distributions from their oficially supported repositories).

The **sysstat** package contains the following utilities: **sar** (collect, report, or save system activity information), **sadf** (display data collected by **sar** in multiple formats), **mpstat** (report processors related statistics), **iostat** (report CPU statistics and I/O statistics for devices and partitions), **pidstat** (report statistics for Linux tasks), **nfsiostat** (report input/output statistics for NFS), **cifsiostat** (report CIFS statistics) and **sa1/2** (sa1: Collect and store binary data in the system activity daily data file, sa2: write a daily report in the /var/log/sa directory) tools, whereas **dstat** adds some extra features to the functionality provided by those tools, along with more counters and flexibility. You can find an overall description of each tool by running **yum info sysstat** or **yum info dstat**, respectively, or checking the individual man pages after installation.

To install both packages:

```
yum update && yum install sysstat dstat
```

The main configuration file for sysstat is /etc/sysconfig/sysstat. You will find the following parameters in that file:

```
# How long to keep log files (in days).
# If value is greater than 28, then log files are kept in
# multiple directories, one for each month.
HISTORY=28
# Compress (using gzip or bzip2) sa and sar files older than (in days):
COMPRESSAFTER=31
# Parameters for the system activity data collector (see sadc manual page)
# which are used for the generation of log files.
SADC_OPTIONS="-S DISK"
# Compression program to use.
ZIP="bzip2"
```

When **sysstat** is installed, two cron jobs are added and enabled in **/etc/cron.d/sysstat**. The first job runs the system activity accounting tool every 10 minutes and stores the reports in **/var/log/sa/saXX** where XX is the day of the month. Thus, **/var/log/sa/sa05** will contain all the system activity reports from the 5th of the month. This assumes that we are using the default value in the HISTORY variable in the configuration file above:

```
*/10 * * * * root /usr/lib64/sa/sa1 1 1
```

The second job generates a daily summary of process accounting at 11:53 pm every day and stores it in /var/log/sa/sarXX files, where XX has the same meaning as in the previous example:

```
53 23 * * * root /usr/lib64/sa/sa2 -A
```

For example, you may want to output system statistics from 9:30 am through 5:30 pm of the sixth of the month to a .csv file that can easily be viewed using LibreOffice Calc or Microsoft Excel (this approach will also allow you to create charts or graphs):

```
sadf -s 09:30:00 -e 17:30:00 -dh /var/log/sa/sa06 -- | sed 's/;/,/g' > system_stats20150806.csv
```

You could altenatively use the **–j** flag instead of **–d** in the **sadf** command above to output the system stats in JSON format, which could be useful if you need to consume the data in a web application, for example (see Fig. 7).

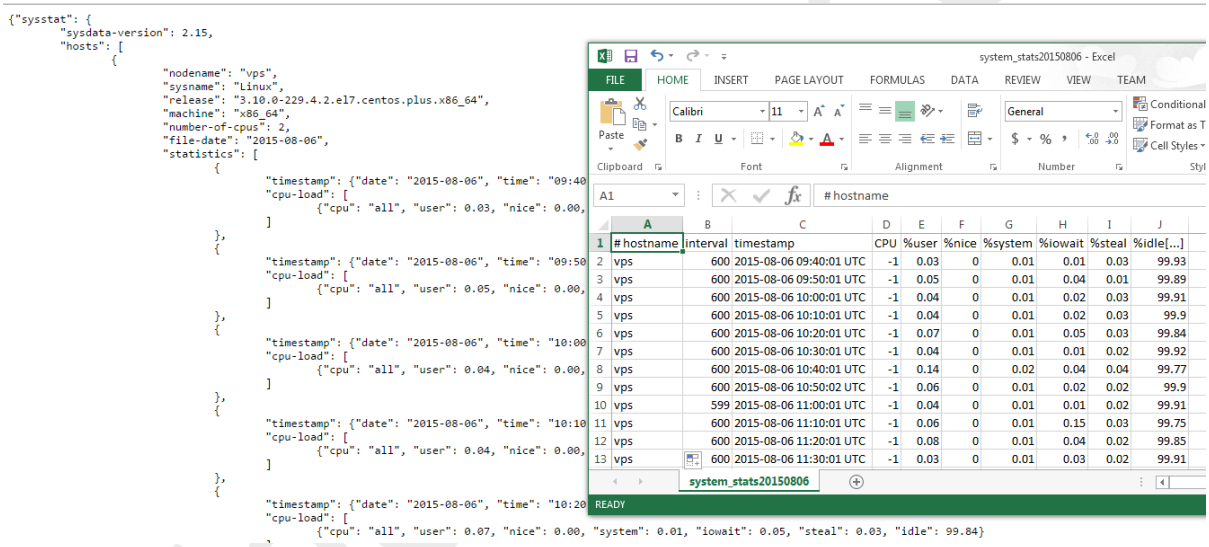


Figure 7: Displaying system stats in JSON format

Finally, let's see what **dstat** has to offer. Please note that if run without arguments, dstat assumes **–cdngy** by default (short for **C**PU, **d**isk, **n**etwork, memory pa**g**es, and s**y**stem stats, respectively), and adds one line every second (execution can be interrupted anytime with Ctrl + C), as seen in Fig. 8:

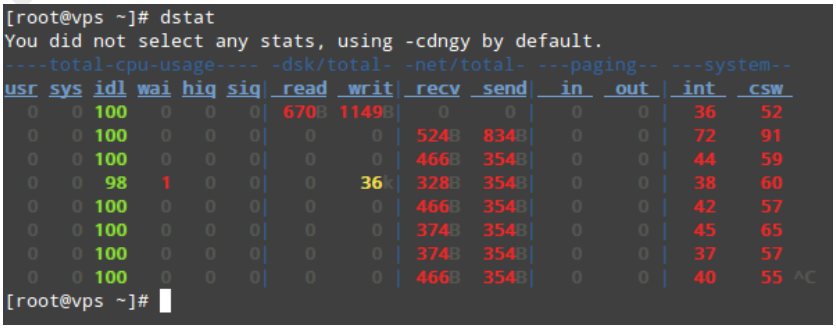


Figure 8: Viewing system stats with dstat

To output the stats to a .csv file, use the **--output** flag followed by a file name. Let's see how this looks on LibreOffice Calc (see Fig. 9):

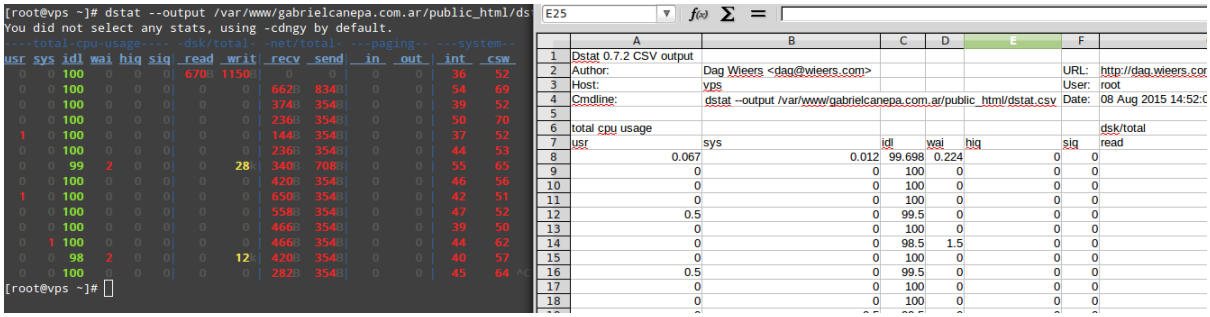


Figure 9: Exporting system stats to LibreOffice Calc

I strongly advise you to check out the man pages of **dstat** and **sysstat**. You will find several other options that will help you create custom and detailed system activity reports.

Summary

In this guide we have explained how to use both native Linux tools and specific utilities provided with RHEL 7 in order to produce reports on system utilization. At one point or another, you will come to rely on these reports as best friends.

Chapter 19: Automating system monitoring and maintenance using shell scripts

Some time ago I read that one of the distinguishing characteristics of an effective system administrator / engineer is ***laziness***. It seemed a little contradictory at first but the author then proceeded to explain why: if a sysadmin spends most of his time solving issues and doing repetitive tasks, you can suspect he or she is not doing things quite right. In other words, an effective system administrator / engineer should develop a plan to perform repetitive tasks with as less action on his / her part as possible, and should foresee problems by using, for example, the tools reviewed in Chapter 18. Thus, although he or she may not seem to be doing much, it's because most of his / her responsibilities have been taken care of with the help of shell scripting, which is what we're going to talk about in this tutorial.

What is a shell script?

In few words, a shell script is nothing more and nothing less than a program that is executed step by step by a shell, which is another program that provides an interface layer between the Linux kernel and the end user.

By default, the shell used for user accounts in RHEL 7 is bash (/bin/bash). If you want a detailed description and some historical background, you can refer to this Wikipedia article. To find out more about the enormous set of features provided by this shell, you may want to check out its man page, which is attached in this article in PDF format. Other than that, it is assumed that you are familiar with Linux commands (if not, I strongly advise you to go through this other article in Tecmint.com before proceeding). Now let's get started.

Writing a script to display system information

For our convenience, let's create a directory to store our shell scripts:

```
mkdir scripts
cd scripts
```

And open a new text file named **system_info.sh** with your preferred text editor. We will begin by inserting a few comments at the top and some commands afterwards:

```
#!/bin/bash

# Sample script written for Part 4 of the RHCE series
# This script will return the following set of system information:
# -Hostname information:
echo -e "\e[31;43m***** HOSTNAME INFORMATION *****\e[0m"
hostnamectl
echo ""
# -File system disk space usage:
echo -e "\e[31;43m***** FILE SYSTEM DISK SPACE USAGE *****\e[0m"
df -h
echo ""
# -Free and used memory in the system:
echo -e "\e[31;43m ***** FREE AND USED MEMORY *****\e[0m"
free
echo ""
# -System uptime and load:
echo -e "\e[31;43m***** SYSTEM UPTIME AND LOAD *****\e[0m"
uptime
echo ""
```

```
# -Logged-in users:
echo -e "\e[31;43m***** CURRENTLY LOGGED-IN USERS *****\e[0m"
who
echo ""
# -Top 5 processes as far as memory usage is concerned
echo -e "\e[31;43m***** TOP 5 MEMORY-CONSUMING PROCESSES *****\e[0m"
ps -eo %mem,%cpu,comm --sort=-%mem | head -n 6
echo ""
echo -e "\e[1;32mDone.\e[0m"
```

Next, give the script execute permissions:

```
chmod +x system_info.sh
```

and run it:

```
./system_info.sh
```

Note that the headers of each section are shown in color for better visualization, as can be seen in Fig. 1:

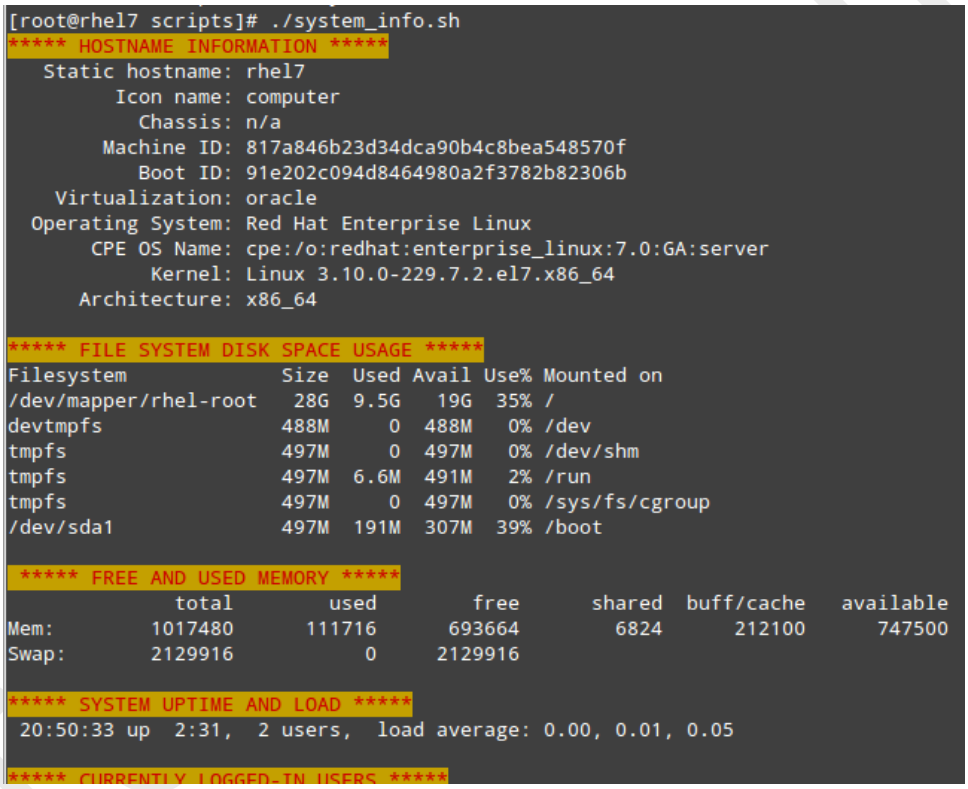


Figure 1: A simple system report via shell scripting

That functionality is provided by this command:

```
echo -e "\e[COLOR1;COLOR2m<YOUR TEXT HERE>\e[0m"
```

Where **COLOR1** and **COLOR2** are the foreground and background colors, respectively (more info and options are explained in this entry from the Arch Linux Wiki) and **<YOUR TEXT HERE>** is the string that you want to show in color.

Automating tasks

The tasks that you may need to automate may vary from case to case. Thus, we cannot possibly cover all of the possible scenarios in a single article, but we will present three classic tasks that can be automated using

shell scripting: 1) update the local file database, 2) find (and alternatively delete) files with 777 permissions, and 3) alert when filesystem usage surpasses a defined limit.

Let's create a file named **auto_tasks.sh** in our scripts directory with the following content:

```
#!/bin/bash

# Sample script to automate tasks:
# -Update local file database:
echo -e "\e[4;32mUPDATING LOCAL FILE DATABASE\e[0m"
updatedb
if [ $? == 0 ]; then
        echo "The local file database was updated correctly."
else
        echo "The local file database was not updated correctly."
fi
echo ""

# -Find and / or delete files with 777 permissions.
echo -e "\e[4;32mLOOKING FOR FILES WITH 777 PERMISSIONS\e[0m"
# Enable either option (comment out the other line), but not both.
# Option 1: Delete files without prompting for confirmation. Assumes GNU
version of find.
#find -type f -perm 0777 -delete
# Option 2: Ask for confirmation before deleting files. More portable
across systems.
find -type f -perm 0777 -exec rm -i {} +;
echo ""
# -Alert when file system usage surpasses a defined limit
echo -e "\e[4;32mCHECKING FILE SYSTEM USAGE\e[0m"
THRESHOLD=30
while read line; do
        # This variable stores the file system path as a string
        FILESYSTEM=$(echo $line | awk '{print $1}')
        # This variable stores the use percentage (XX%)
        PERCENTAGE=$(echo $line | awk '{print $5}')
        # Use percentage without the % sign.
        USAGE=${PERCENTAGE%?}
        if [ $USAGE -gt $THRESHOLD ]; then
                echo "The remaining available space in $FILESYSTEM is
critically low. Used: $PERCENTAGE"
        fi
done < <(df -h --total | grep -vi filesystem)
```

Please note that there is a space between the two **<** signs in the last line of the script. The result is shown in Fig. 2:

```
[root@rhel7 scripts]# ./auto_tasks.sh
UPDATING LOCAL FILE DATABASE
The local file database was updated correctly.

LOOKING FOR FILES WITH 777 PERMISSIONS
rm: remove regular empty file ‘./file.txt’? y

CHECKING FILE SYSTEM USAGE
The remaining available space in /dev/mapper/rhel-root is critically low. Used: 35%
The remaining available space in /dev/sda1 is critically low. Used: 39%
The remaining available space in total is critically low. Used: 33%
[root@rhel7 scripts]#
```

Figure 2: Automating tasks using shell scripts

Using cron

To take efficiency one step further, you will not want to sit in front of your computer and run those scripts manually. Rather, you will use cron to schedule those tasks to run on a periodic basis and sends the results to a predefined list of recipients via email or save them to a file that can be viewed using a web browser.

The following script (filesystem_usage.sh) will run the well-known **df –h** command, format the output into a HTML table and save it in the report.html file:

```
#!/bin/bash
# Sample script to demonstrate the creation of an HTML report using shell scripting
# Web directory
WEB_DIR=/var/www/html
# A little CSS and table layout to make the report look a little nicer
echo "<HTML>
<HEAD>
<style>
.titulo{font-size: 1em; color: white; background:#0863CE; padding: 0.1em 0.2em;}
table
{
border-collapse:collapse;
}
table, td, th
{
border:1px solid black;
}
</style>
<meta http-equiv='Content-Type' content='text/html; charset=UTF-8' />
</HEAD>
<BODY>" > $WEB_DIR/report.html
# View hostname and insert it at the top of the html body
HOST=$(hostname)
echo "Filesystem usage for host <strong>$HOST</strong><br>
Last updated: <strong>$(date)</strong><br><br>
<table border='1'>
<tr><th class='titulo'>Filesystem</td>
<th class='titulo'>Size</td>
<th class='titulo'>Use %</td>
</tr>" >> $WEB_DIR/report.html
```

```
# Read the output of df -h line by line
while read line; do
echo "<tr><td align='center'>" >> $WEB_DIR/report.html
echo $line | awk '{print $1}' >> $WEB_DIR/report.html
echo "</td><td align='center'>" >> $WEB_DIR/report.html
echo $line | awk '{print $2}' >> $WEB_DIR/report.html
echo "</td><td align='center'>" >> $WEB_DIR/report.html
echo $line | awk '{print $5}' >> $WEB_DIR/report.html
echo "</td></tr>" >> $WEB_DIR/report.html
done < <(df -h | grep -vi filesystem)
echo "</table></BODY></HTML>" >> $WEB_DIR/report.html
```

In our RHEL 7 server (192.168.0.18), this looks as follows (see Fig. 3):

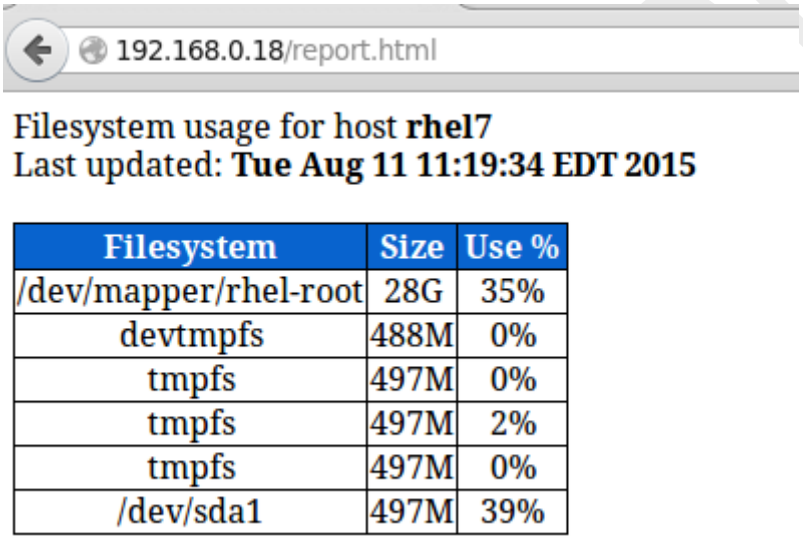


Figure 3: Viewing a basic available disk space report in HTML format

You can add to that report as much information as you want.

To run the script every day at 1:30 pm, add the following crontab entry:

```
30 13 * * * /root/scripts/filesystem_usage.sh
```

Summary

You will most likely think of several other tasks that you want or need to automate; as you can see, using shell scripting will greatly simplify this effort.

Chapter 20: Managing system logs

In order to keep your RHEL 7 systems secure, you need to know how to monitor all of the activities that take place on such systems by examining log files. Thus, you will be able to detect any unusual or potentially malicious activity and perform system troubleshooting or take another appropriate action.

In RHEL 7, the rsyslogd daemon is responsible for system logging and reads its configuration from /etc/rsyslog.conf (this file specifies the default location for all system logs) and from files inside /etc/rsyslog.d, if any.

The main configuration file

A quick inspection of the rsyslog.conf will be helpful to start. This file is divided into 3 main sections: **Modules** (since rsyslog follows a modular design), **Global directives** (used to set global properties of the rsyslogd daemon), and **Rules**. As you will probably guess, this last section indicates ***what*** gets logged or shown (also known as the selector) and ***where***, and will be our focus throughout this article.

A typical line in rsyslog.conf is as follows (see Fig. 1):

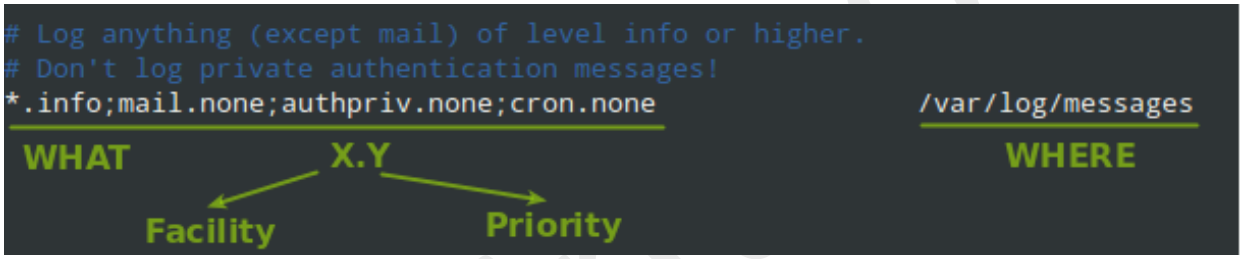


Figure 1: The rsyslogd configuration file

In the image above, we can see that a selector consists of one or more pairs **Facility:Priority** separated by semicolons, where **Facility** describes the type of message (refer to section 4.1.1 in RFC 3164 to see the complete list of facilities available for rsyslog) and **Priority** indicates its severity, which can be one of the following self-explanatory words:

1. debug
2. info
3. notice
4. warning
5. err
6. crit
7. alert
8. emerg

Though not a priority itself, the keyword **none** means no priority –at all- of the given facility.

Note that a given priority indicates that all messages of such priority and above should be logged. Thus, the line in the example above instructs the rsyslogd daemon to log all messages of priority **info** or higher (regardless of the facility) except those belonging to mail, authpriv, and cron services (no messages coming from this facilities will be taken into account) to /var/log/messages.

You can also group multiple facilities using the colon sign to apply the same priority to all of them. Thus, the line

```
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
```

Could be rewritten as

```
*.info;mail,authpriv,cron.none                /var/log/messages
```

In other words, the facilities mail, authpriv, and cron are grouped and the keyword none is applied to the three of them.

To log all daemon messages to /var/log/tecmint.log, we need to add the following line either in rsyslog.conf or in a separate file (easier to manage) inside /etc/rsyslog.d:

```
daemon.*    /var/log/tecmint.log
```

Let's restart the daemon (note that the service name does not end with a d):

```
systemctl restart rsyslog
```

And check the contents of our custom log before and after restarting two random daemons (see Fig. 2):

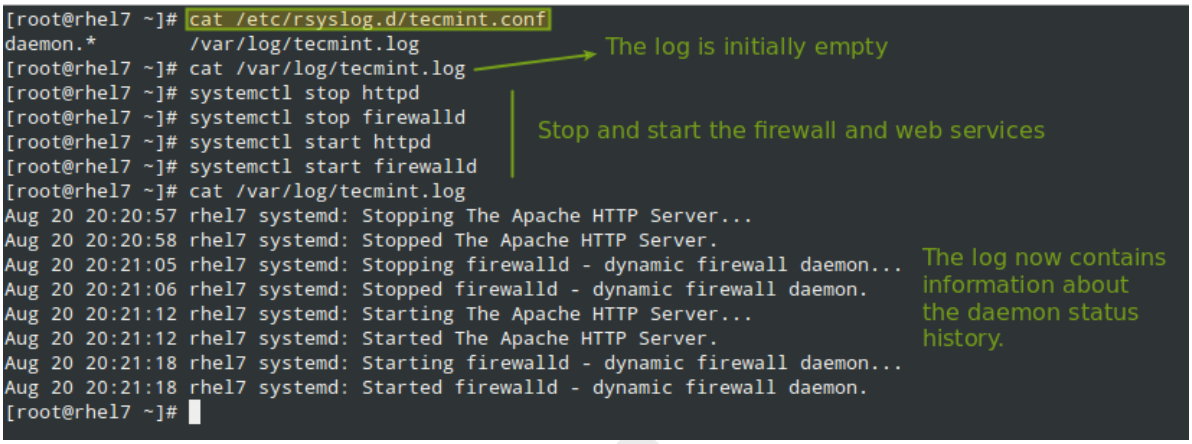


Figure 2: Inspecting logs

As a self-study exercise, I would recommend you play around with the facilities and priorities and either log additional messages to existing log files or create new ones as in the previous example.

Rotating logs

To prevent log files from growing endlessly, the logrotate utility is used to rotate, compress, remove, and alternatively mail logs, thus easing the administration of systems that generate large numbers of log files.

Logrotate runs daily as a cron job (/etc/cron.daily/logrotate) and reads its configuration from /etc/logrotate.conf and from files located in /etc/logrotate.d, if any. As with the case of rsyslog, even when you can include settings for specific services in the main file, creating separate configuration files for each one will help organize your settings better.

Let's take a look at a typical logrotate.conf:

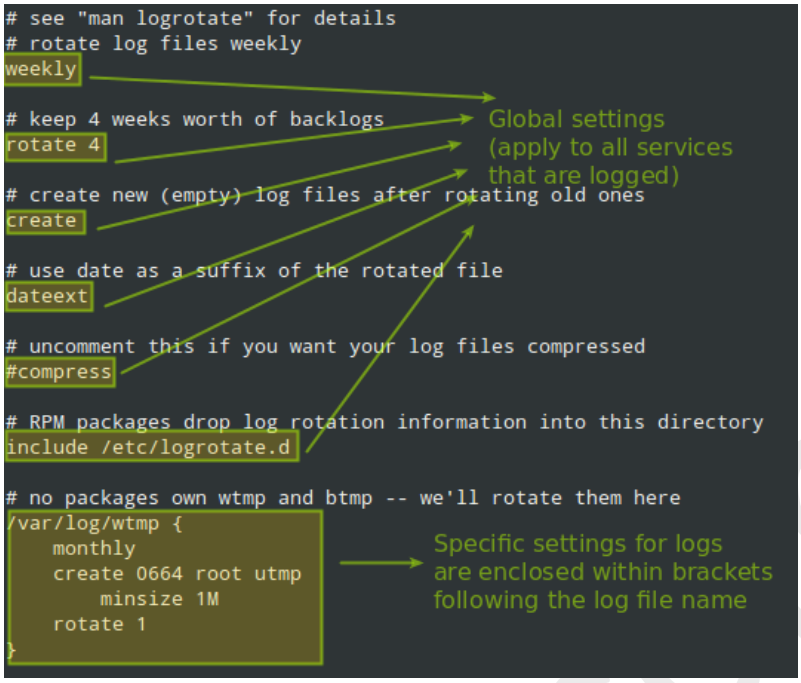


Figure 3: Logrotate configuration file

In the example above, logrotate will perform the following actions for /var/loh/wtmp: attempt to rotate only once a month, but only if the file is at least 1 MB in size, then create a brand new log file with permissions set to 0664 and ownership given to user root and group utmp. Next, only keep one archived log, as specified by the **rotate** directive (see Fig. 3):

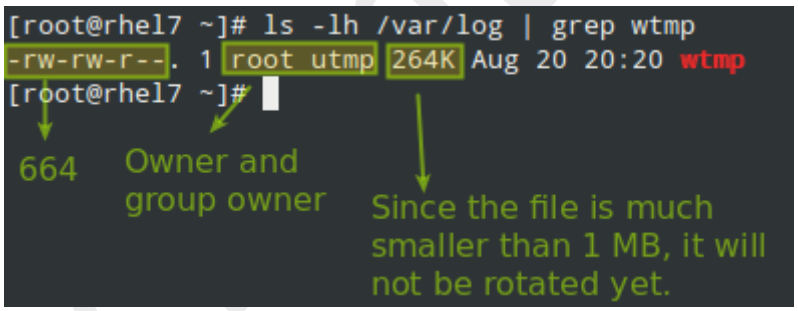


Figure 3: Log rotation and details

Let's now consider another example as found in /etc/logrotate.d/httpd (see Fig. 4):

Figure 4: Logs are rotated as per the configuration of logrotate

You can read more about the settings for logrotate in its man pages (man logrotate and man logrotate.conf). Both files are provided along with this article in PDF format for your reading convenience.

As a system engineer, it will be pretty much up to you to decide for how long logs will be stored and in what format, depending on whether you have /var in a separate partition / logical volume. Otherwise, you really want to consider removing old logs to save storage space. On the other hand, you may be forced to keep several logs for future security auditing according to your company's or client's internal policies.

Saving logs to a database

Of course examining logs (even with the help of tools such as grep and regular expressions) can become a rather tedious task. For that reason, rsyslog allows us to export them into a database (OTB supported RDBMS include MySQL, MariaDB, PostgreSQL, and Oracle.

This section of the tutorial assumes that you have already installed the MariaDB server and client in the same RHEL 7 box where the logs are being managed:

```
yum update && yum install mariadb mariadb-server mariadb-client rsyslog-mysql
systemctl enable mariadb && systemctl start mariadb
```

Then use the **mysql_secure_installation** utility to set the password for the root user and other security considerations (see Fig. 5):

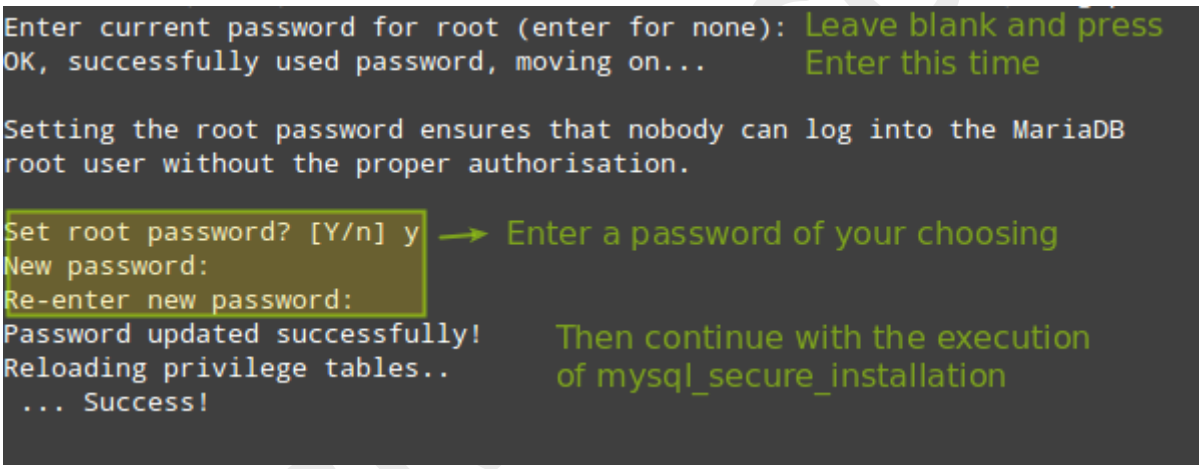


Figure 5: Securing MariaDB

Note: if you don't want to use the MariaDB root user to insert log messages to the database, you can configure another user account to do so. Explaining how to do that is out of the scope of this tutorial but is explained in detail in this entry of the MariaDB knowledge base. In this tutorial we will use the root account for simplicity.

Next, download the createDB.sql script from GitHub and import it into your database server:

```
mysql -u root -p < createDB.sql
```

```
[root@rhel7 ~]# mysql -u root -p < createDB.sql
Enter password:
[root@rhel7 ~]# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 13
Server version: 5.5.41-MariaDB MariaDB Server

Copyright (c) 2000, 2014, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE Syslog;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [Syslog]> SHOW TABLES;
+------------------------+
| Tables_in_Syslog       |
+------------------------+
| SystemEvents           |
| SystemEventsProperties |
+------------------------+
2 rows in set (0.00 sec)

MariaDB [Syslog]>
```

Figure 6: Importing the createDB.sql script to the database server

Finally, add the following lines to /etc/rsyslog.conf:

```
$ModLoad ommysql
$ActionOmmysqlServerPort 3306
*.* :ommysql:localhost,Syslog,root,YourPasswordHere
```

Restart rsyslog and the database server:

```
systemctl restart rsyslog && systemctl restart mariadb
```

Now let's perform some tasks that will modify the logs (like stopping and starting services, for example), then log to your DB server and use standard SQL commands to display and search in the logs (see Fig. 7):

```
USE Syslog;
SELECT ReceivedAt, Message FROM SystemEvents;
```

```
MariaDB [Syslog]> SELECT ReceivedAt, Message FROM SystemEvents;
+---------------------+-------------------------------------------------------------------
| ReceivedAt          | Message
+---------------------+-------------------------------------------------------------------
| 2015-08-21 10:15:20 |  [origin software="rsyslogd" swVersion="7.4.7" x-pid="4010" x-i
| 2015-08-21 10:15:20 | Stopping System Logging Service...
| 2015-08-21 10:15:20 | Starting System Logging Service...
| 2015-08-21 10:15:20 | Started System Logging Service.
| 2015-08-21 10:15:38 | Stopping MariaDB database server...
| 2015-08-21 10:20:01 | Created slice user-0.slice.
| 2015-08-21 10:20:01 | Starting Session 29 of user root.
| 2015-08-21 10:20:01 | Started Session 29 of user root.
| 2015-08-21 10:20:01 | (root) CMD (/usr/lib64/sa/sa1 1 1)
+---------------------+-------------------------------------------------------------------
9 rows in set (0.00 sec)
```

Figure 7: Retrieving system events from the database server

Summary

In this article we have explained how to set up system logging, how to rotate logs, and how to redirect the messages to a database for easier search. We hope that these skills will be helpful as you prepare for the RHCE exam and in your daily responsibilities as well.

Chapter 21: Samba

Since computers seldom work as isolated systems, it is to be expected that as a system administrator or engineer, you know how to set up and maintain a network with multiple types of servers.

In this article and in the next of this series we will go through the essentials of setting up Samba and NFS servers with Windows/Linux and Linux clients, respectively. This article will definitely come in handy if you're called upon to set up file servers in corporate or enterprise environments where you are likely to find different operating systems and types of devices.

Since you can read about the background and the technical aspects of both Samba and NFS all over the Internet, in this article and the next we will cut right to the chase with the topic at hand.

Installing packages

Our current testing environment consists of two RHEL 7 boxes and one Windows 8 machine, in that order:

1) Samba / NFS server [**box1 (RHEL 7)**: **192.168.0.18**],

2) Samba client #1 [box2 (RHEL 7): 192.168.0.20]

3) Samba client #2 [Windows 8 machine: 192.168.0.106]

On box1, install the following packages:

```
yum update && yum install samba samba-client samba-common
```

On box2:

```
yum update && yum install samba samba-client samba-common cifs-utils
```

Once the installation is complete, we're ready to configure our share.

Setting up file sharing through Samba

One of the reason why Samba is so relevant is because it provides file and print services to SMB/CIFS clients, which causes those clients to see the server as if it was a Windows system (I must admit I tend to get a little emotional while writing about this topic as it was my first setup as a new Linux system administrator some years ago).

Adding system users and setting up permissions and ownership

To allow for group collaboration, we will create a group named **finance** with two users (**user1** and **user2**) and a directory /finance in box1. We will also change the group owner of this directory to **finance** and set its permissions to **0770** (read, write, and execution permissions for the owner and the group owner):

```
groupadd finance
useradd user1
useradd user2
usermod -a -G finance user1
usermod -a -G finance user2
mkdir /finance
chmod 0770 /finance
chgrp finance /finance
```

Configuring SELinux and firewalld

In preparation to configure /finance as a Samba share, we will need to either disable SELinux or set the proper boolean and security context values as follows (otherwise, SELinux will prevent clients from accessing the share):

```
setsebool -P samba_export_all_ro=1 samba_export_all_rw=1
getsebool -a | grep samba_export
semanage fcontext -at samba_share_t "/finance(/.*)?"
restorecon /finance
```

In addition, we must ensure that Samba traffic is allowed by the firewalld

```
firewall-cmd --permanent --add-service=samba
firewall-cmd --reload
```

The Samba configuration file: /etc/samba/smb.conf

Now it's time to dive into the configuration file and add the section for our share: we want the members of the finance group to be able to browse the contents of /finance, and save / create files or subdirectories in it (which by default will have their permission bits set to 0770 and finance will be their group owner):

```
[finance]
comment=Directory for collaboration of the company's finance team
browsable=yes
path=/finance
public=no
valid users=@finance
write list=@finance
writeable=yes
create mask=0770
Force create mode=0770
force group=finance
```

Save the file and then test it with the **testparm** utility. If there are any errors, the output of the following command will indicate what you need to fix. Otherwise, it will display a review of your Samba server configuration (see Fig. 1):

```
[root@box1 ~]# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[homes]"
Processing section "[printers]"
Processing section "[finance]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions
```

Figure 1: Testing the smb.conf file

Should you want to add another share that is open to the public (meaning without any authentication whatsoever), create another section in /etc/samba/smb.conf and under the new share's name copy the section above, only changing **public=no** to **public=yes** and not including the **valid users** and **write lis**t directives.

Adding Samba users

Next, you will need to add user1 and user2 as Samba users. To do so, you will use the smbpasswd command, which interacts with Samba's internal database. You will be prompted to enter a password that you will later use to connect to the share:

```
smbpasswd -a user1
smbpasswd -a user2
```

Finally, restart Samba, enable the service to start on boot, and make sure the share is actually available to network clients (see Fig. 2):

```
systemctl start smb
systemctl enable smb
smbclient -L localhost -U user1
smbclient -L localhost -U user2
```

```
[root@box1 ~]# smbclient -L localhost -U user1
Enter user1's password:
Domain=[MYGROUP] OS=[Unix] Server=[Samba 4.1.12]

        Sharename       Type      Comment
        ---------       ----      -------
        finance         Disk      Directory for collaboration of the company's finance team
        IPC$            IPC       IPC Service (Samba Server Version 4.1.12)
        user1           Disk      Home Directories
Domain=[MYGROUP] OS=[Unix] Server=[Samba 4.1.12]

        Server               Comment
        ---------            -------

        Workgroup            Master
        ---------            -------
[root@box1 ~]# smbclient -L localhost -U user2
Enter user2's password:
Domain=[MYGROUP] OS=[Unix] Server=[Samba 4.1.12]

        Sharename       Type      Comment
        ---------       ----      -------
        finance         Disk      Directory for collaboration of the company's finance team
        IPC$            IPC       IPC Service (Samba Server Version 4.1.12)
        user2           Disk      Home Directories
Domain=[MYGROUP] OS=[Unix] Server=[Samba 4.1.12]

        Server               Comment
        ---------            -------

        Workgroup            Master
        ---------            -------
[root@box1 ~]#
```

Figure 2: Testing availability of the Samba shares

At this point, the Samba file server has been properly installed and configured. Now it's time to test this setup on our RHEL 7 and Windows 8 clients.

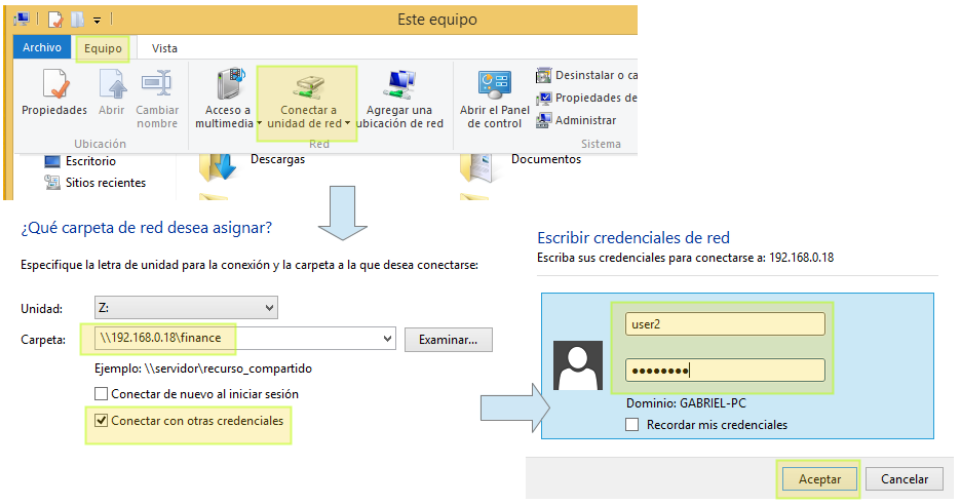


Figure 5: Mounting the Samba share in Microsoft Windows as user2

Finally, let's create a file and check the permissions and ownership (see Fig. 6):

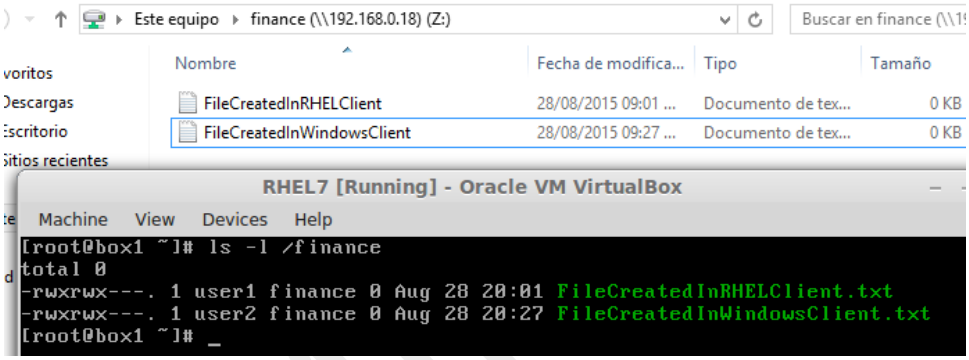


Figure 6: File permissions and ownership of file created using a Windows client

This time the file belongs to user2 since that's the account we used to connect from the Windows client.

Summary

In this article we have explained not only how to set up a Samba server and two clients using different operating systems, but also how to configure the firewalld and SELinux on the server to allow the desired group collaboration capabilities.

Last, but not least, let me recommend the reading of the online man page of smb.conf to explore other configuration directives that may be more suitable for your case than the scenario described in this article.

Chapter 22: NFS and Kerberos

In the last chapter we reviewed how to set up a Samba share over a network that may consist of multiple types of operating systems. Now, if you need to set up file sharing for a group of Unix-like clients you will automatically think of the Network File System, or NFS for short. In this article we will walk you through the process of using Kerberos-based authentication for NFS shares.

It is assumed that you already have set up a NFS server and a client. If not, please refer to this article -which will list the necessary packages that need to be installed and explain how to perform initial configurations on the server- before proceeding further.

In addition, you will want to configure both SELinux and firewalld to allow for file sharing through NFS. The following example assumes that your NFS share is located in /nfs in box2:

```
semanage fcontext -a -t public_content_rw_t "/nfs(/.*)?"
restorecon -R /nfs

setsebool -P nfs_export_all_rw on
setsebool -P nfs_export_all_ro on
```

(where the **–P** flag indicates persistence across reboots)

Finally, don't forget to:

1) Create a group called **nfs** and add the **nfsnobody** user to it, then change the permissions of the /nfs directory to 0770 and its group owner to nfs. Thus, nfsnobody (which is mapped to the client requests) will have write permissions on the share) and you won't need to use **no_root_squash** in the /etc/exports file.

```
groupadd nfs

usermod –a –G nfs nfsnobody

chmod 0770 /nfs

chgrp nfs /nfs
```

2) Modify the exports file (/etc/exports) as follows to only allow access from box1 using Kerberos security (**sec=krb5**). Note that the value of anongid has been set to the GID of the nfs group that we created previously:

```
/nfs box1(rw,sec=krb5,anongid=1004)
```

2) Re-export (**-r**) all (**-a**) the NFS shares. Adding verbosity to the output (**-v**) is a good idea since it will provide helpful information to troubleshoot the server if something goes wrong:

```
exportfs -arv
```

3) restart and enable the NFS server and related services. Note that you don't have to enable **nfs-lock** and **nfs-idmapd** because they will be automatically started by the other services on boot:

```
systemctl restart rpcbind nfs-server nfs-lock nfs-idmap
systemctl enable rpcbind nfs-server
```

Testing environment and other prerequisites

In this guide we will use the following test environment:

- Client machine [**box1**: **192.168.0.18**]
- NFS / Kerberos server [**box2**: **192.168.0.20**] (also known as ***Key Distribution Center***, or ***KDC*** for short). Note that this service is crucial to the authentication scheme.

As you can see, the NFS server and the KDC are hosted in the same machine for simplicity, although you can set them up in separate machines if you have more available. Both machines are members of the **mydomain.com** domain.

Last but not least, Kerberos requires at least a basic schema of name resolution and the Network Time Protocol service to be present in both client and server since the security of Kerberos authentication is in part based upon the timestamps of tickets.

To set up name resolution, we will use the /etc/hosts file in both client and server:

```
192.168.0.18    box1.mydomain.com    box1
192.168.0.20    box2.mydomain.com    box2
```

In RHEL 7, **chrony** is the default software that is used for NTP synchronization:

```
yum install chrony
systemctl start chronyd
systemctl enable chronyd
```

To make sure chrony is actually synchronizing your system's time with time servers you may want to issue the following command two or three times and make sure the offset is getting nearer to zero (see Fig. 1):

```
chronyc tracking
```

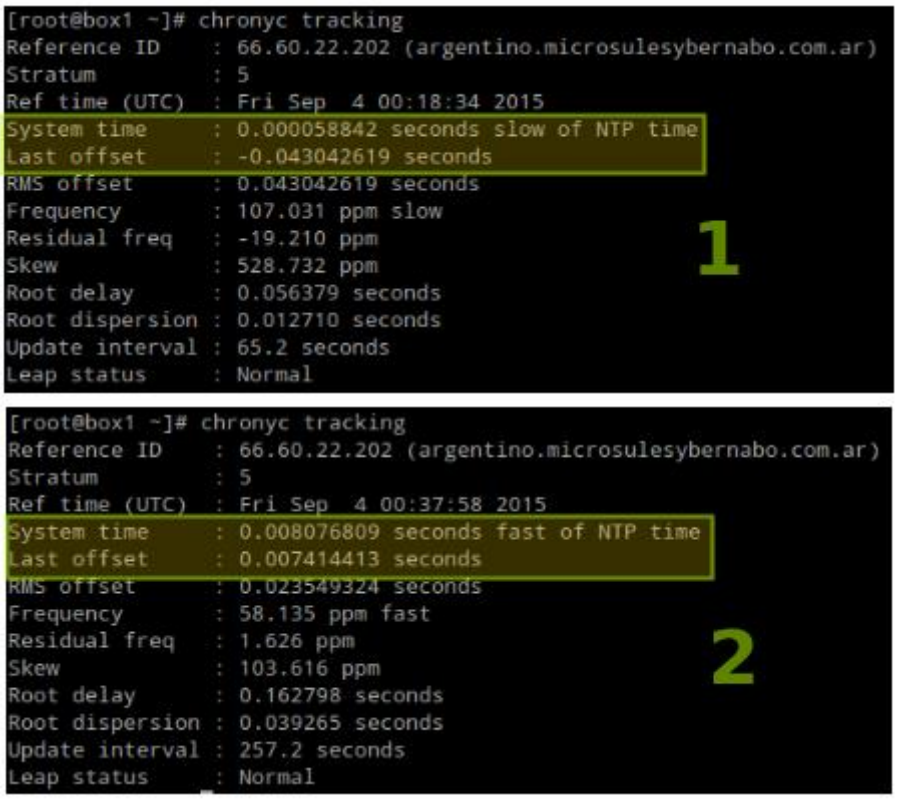


Figure 1: Checking time synchronization with chrony

Setting up Kerberos

To set up the KDC, install the following packages on both server and client (omit the server package in the client):

```
yum update && yum install krb5-server krb5-workstation pam_krb5
```

Once it is installed, edit the configuration files (/etc/krb5.conf and **/var/kerberos/krb5kdc/kadm5.acl**) and replace all instances of example.com (lowercase and uppercase) with mydomain.com as follows.

Next, enable Kerberos through the firewall and start / enable the related services. Important: nfs-secure must be started and enabled on the client as well:

```
firewall-cmd --permanent --add-service=kerberos
systemctl start krb5kdc kadmin nfs-secure
systemctl enable krb5kdc kadmin nfs-secure
```

Now create the Kerberos database (please note that this may take a while as it requires a some level of entropy in your system. To speed things up, I opened another terminal and ran `ping -f localhost` for 30-45 seconds). See Fig. 2 for a detail of what you can expect.

```
kdb5_util create -s
```

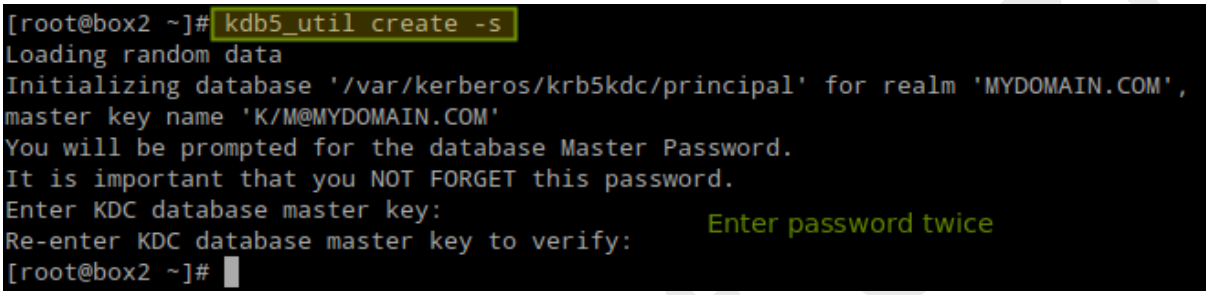


Figure 2: Creating the kerberos database

Next, using the **kadmin.local** tool, create an admin principal for root:

```
kadmin.local
addprinc root/admin
```

And add the Kerberos server to the database:

```
addprinc -randkey host/box2.mydomain.com
```

Same with the NFS service for both client (box1) and server (box2). Please note that in the screenshot below I forgot to do it for box1 before quitting:

```
addprinc -randkey nfs/box2.mydomain.com
addprinc -randkey nfs/box1.mydomain.com
```

And exit by typing **quit** and pressing Enter (see Fig. 2):

```
[root@box2 ~]# kdb5_util create -s
Loading random data
Initializing database '/var/kerberos/krb5kdc/principal' for realm 'MYDOMAIN.COM',
master key name 'K/M@MYDOMAIN.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:                    Enter password twice
Re-enter KDC database master key to verify:
[root@box2 ~]# kadmin.local
Authenticating as principal root/admin@MYDOMAIN.COM with password.
kadmin.local:  addprinc root/admin
WARNING: no policy specified for root/admin@MYDOMAIN.COM; defaulting to no policy
Enter password for principal "root/admin@MYDOMAIN.COM":  Enter password twice
Re-enter password for principal "root/admin@MYDOMAIN.COM":
Principal "root/admin@MYDOMAIN.COM" created.
kadmin.local:  addprinc -randkey host/box2.mydomain.com
WARNING: no policy specified for host/box2.mydomain.com@MYDOMAIN.COM; defaulting t
o no policy
Principal "host/box2.mydomain.com@MYDOMAIN.COM" created.
kadmin.local:  addprinc -randkey nfs/box2.mydomain.com
WARNING: no policy specified for nfs/box2.mydomain.com@MYDOMAIN.COM; defaulting to
 no policy
Principal "nfs/box2.mydomain.com@MYDOMAIN.COM" created.
kadmin.local:  quit
[root@box2 ~]#
```

Figure 2: Setting up Kerberos

Then obtain and cache Kerberos ticket-granting ticket for root/admin (see Fig. 3):

```
kinit root/admin
klist
```

```
[root@box2 ~]# kinit root/admin
Password for root/admin@MYDOMAIN.COM:
[root@box2 ~]# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: root/admin@MYDOMAIN.COM

Valid starting       Expires              Service principal
09/05/2015 14:08:42  09/06/2015 14:08:42  krbtgt/MYDOMAIN.COM@MYDOMAIN.COM
[root@box2 ~]#
```

Figure 3: Obtaining and caching a Kerberos ticket

The last step before actually using Kerberos is storing into a keytab file (in the server) the principals that are authorized to use Kerberos authentication:

```
kdadmin.local
ktadd host/box2.mydomain.com
ktadd nfs/box2.mydomain.com
ktadd nfs/box1.mydomain.com
```

Finally, mount the share and perform a write test (see Fig. 4):

```
mount -t nfs4 -o sec=krb5 box2:/nfs /mnt
echo "Hello from Tecmint.com" > /mnt/greeting.txt
```

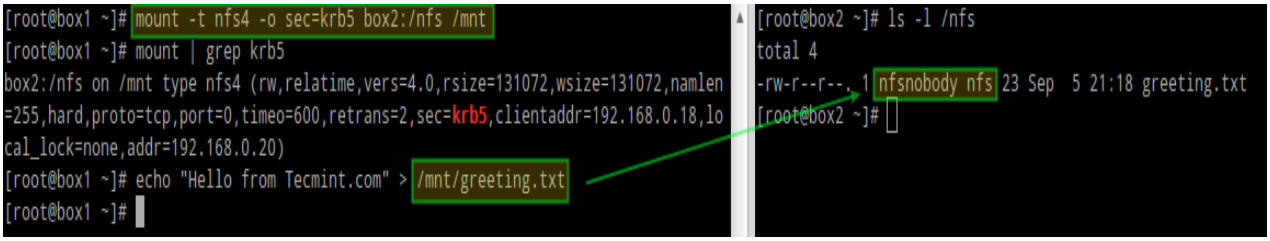


Figure 4: Mounting the NFS share

Let's now unmount the share, rename the keytab file in the client (to simulate it's not present) and try to mount the share again (see Fig. 5):

```
umount /mnt
mv /etc/krb5.keytab /etc/krb5.keytab.orig
```

Figure 5: The keytab file is necessary to mount the share

Now you can use the NFS share with Kerberos-based authentication.

Summary

In this article we have explained how to set up NFS with Kerberos authentication. Since there is much more to the topic than we can cover in a single guide, feel free to check the online Kerberos documentation and since Kerberos is a bit tricky –to say the least- don't hesitate to drop us a note using the form below if you run into any issue or need help with your testing or implementation.

Chapter 23: Securing the Apache web server with TLS

If you are a system administrator who is in charge of maintaining and securing a web server, you can't afford to not devote your very best efforts to ensure that data served by or going through your server is protected at all times.

In order to provide more secure communications between web clients and servers, the HTTPS protocol was born as a combination of HTTP and SSL (Secure Sockets Layer) or more recently, TLS (Transport Layer Security). Due to some serious security breaches, SSL has been deprecated in favor of the more robust TLS. For that reason, in this article we will explain how to secure connections between your web server and clients using TLS.

This tutorial assumes that you have already installed and configured your Apache web server. If not, please refer to this article in this site before proceeding further.

Prerequisites and installation

First off, make sure that Apache is running and that both http and https are allowed through the firewall:

```
systemctl start http
systemctl enable http
firewall-cmd --permanent --add-service=http
firewall-cmd --permanent --add-service=https
```

Then install the necessary packages:

```
yum update && yum install openssl mod_nss crypto-utils
```

Please note that you can replace **mod_nss** with **mod_ssl** in the command above if you want to use OpenSSL libraries instead of NSS (Network Security Service) to implement TLS (which one to use is left entirely up to you, but we will use NSS in this article as it is more robust; for example, it supports recent cryptography standards such as PKCS #11).

Finally, uninstall mod_ssl if you chose to use mod_nss, or viceversa.

```
yum remove mod_ssl
```

Configuring NSS

After mod_nss is installed, its default configuration file is created as **/etc/httpd/conf.d/nss.conf**. You should then make sure that all of the **Listen** and **VirtualHost** directives point to port 443 (default port for HTTPS):

```
Listen 443
VirtualHost _default_:443
```

Then restart Apache and check whether the mod_nss module has been loaded (see Fig. 1):

```
apachectl restart
httpd -M | grep nss
```

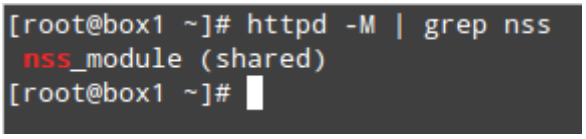


Figure 1: Checking that the mod_nss module has been loaded

Next, the following edits should be made in **/etc/httpd/conf.d/nss.conf**:

1) Indicate NSS database directory. You can use the default directory or create a new one. In this tutorial we will use the default:

```
NSSCertificateDatabase /etc/httpd/alias
```

2) Avoid manual passphrase entry on each system start by saving the password to the database directory in /etc/httpd/nss-db-password.conf:

```
NSSPassPhraseDialog file:/etc/httpd/nss-db-password.conf
```

where /etc/httpd/nss-db-password.conf contains ONLY the following line and **mypassword** is the password that you will set later for the NSS database:

```
internal:mypassword
```

In addition, its permissions and ownership should be set to **0640** and **root:apache**, respectively:

```
chmod 640 /etc/httpd/nss-db-password.conf
chgrp apache /etc/httpd/nss-db-password.conf
```

3) Red Hat recommends disabling SSL and all versions of TLS previous to TLSv1.0 due to the POODLE SSLv3 vulnerability (more information here). Make sure that every instance of the NSSProtocol directive reads as follows (you are likely to find only one if you are not hosting other virtual hosts):

```
NSSProtocol TLSv1.0,TLSv1.1
```

4) Apache will refuse to restart as this is a self-signed certificate and will not recognize the issuer as valid. For this reason, in this particular case you will have to add

```
NSSEnforceValidCerts off
```

5) Though not strictly required, it is important to set a password for the NSS database (see Fig. 2):

```
certutil -W -d /etc/httpd/alias
```

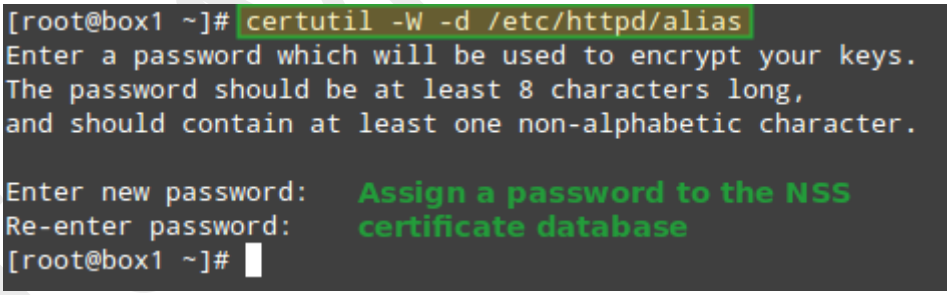


Figure 2: Setting a password to the NSS certificate database

Creating a self-signed certificate

Next, we will create a self-signed certificate that will identify the server to our clients (please note that this method is not the best option for production environments; for such use you may want to consider buying a certificate verified by a 3rd trusted certificate authority, such as DigiCert).

To create a new NSS-compliant certificate for box1 which will be valid for 365 days, we will use the genkey command. When this process completes:

```
genkey --nss --days 365 box1
```

Choose **Next** (see Fig. 3):

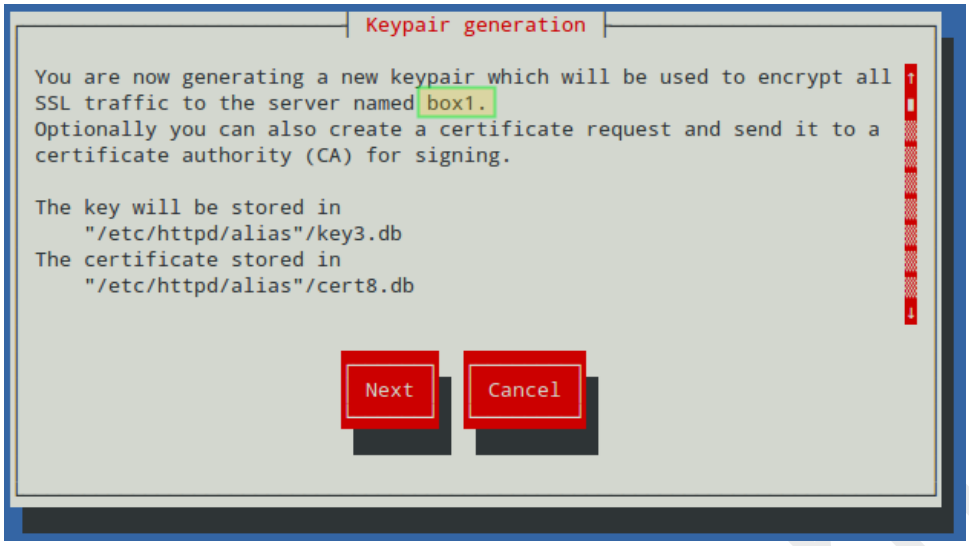


Figure 3: Creating the self-signed certificate, step 1

You can leave the default choice for the key size (2048), then choose **Next** again (see Fig. 4):

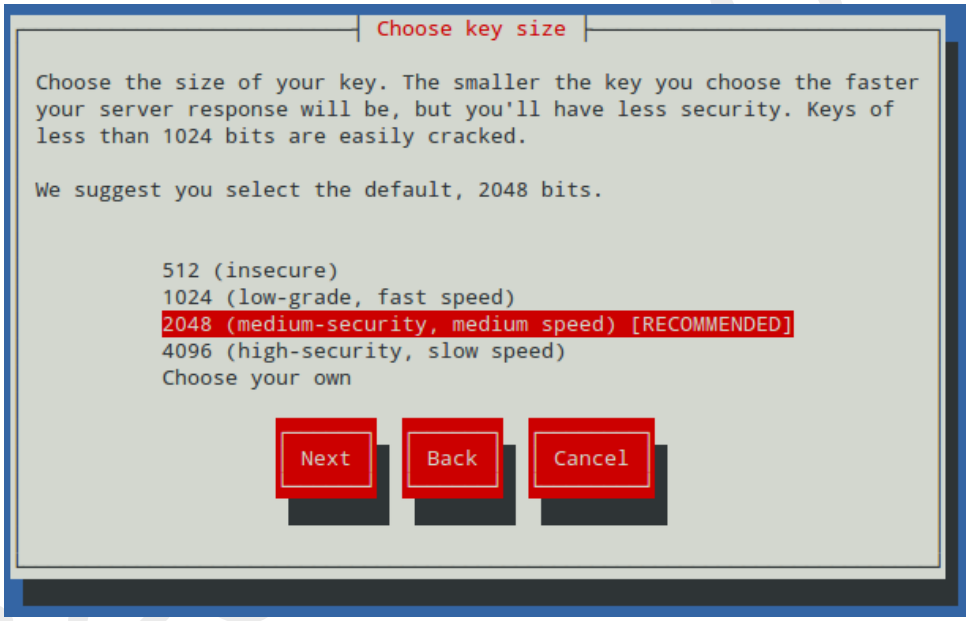


Figure 4: Creating a self-signed certificate, step 2

Wait while the system generates random bits (see Fig. 5):

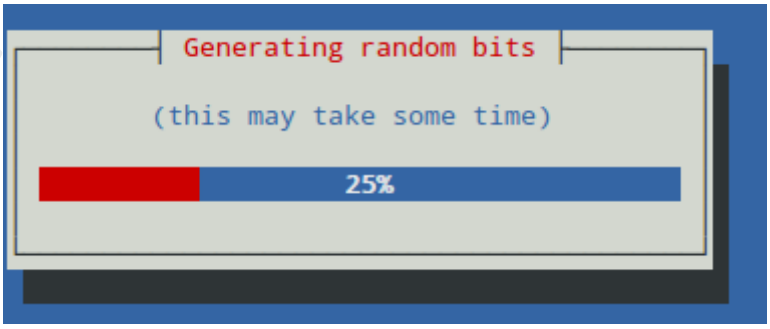


Figure 4: Creating a self-signed certificate, step 3

To speed up the process, you will be prompted to enter random text in your console, as shown in the following screencast. Please note how the progress bar stops when no input from the keyboard is received. Then, you will be asked to:

1) whether to send the Certificate Sign Request (CSR) to a Certificate Authority (CA): Choose **No**, as this is a self-signed certificate.

2) to enter the information for the certificate.

Finally, you will be prompted to enter the password to the NSS certificate that you set ealier (see Fig. 5):

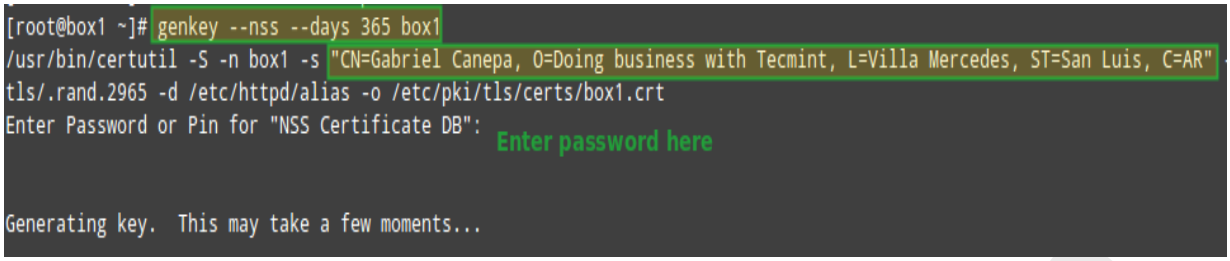


Figure 5: Creating a self-signed certificate, step 4

At anytime, you can list the existing certificates with

```
certutil -L -d /etc/httpd/alias
```

as can be seen in Fig. 6:

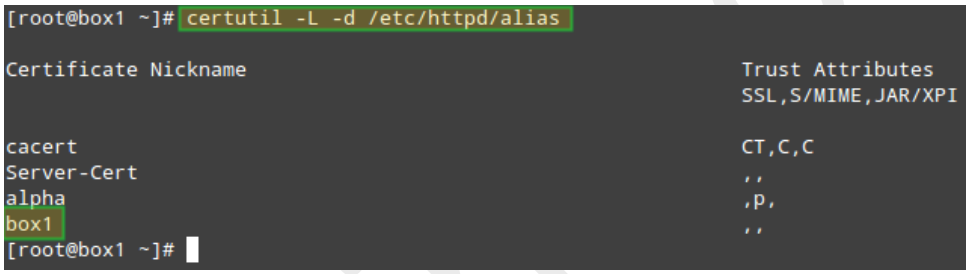


Figure 6: Listing certificates

And delete them by name (only if strictly required, replacing **box1** by your own certificate name) with

```
certutil -d /etc/httpd/alias -D -n "box1"
```

if you need to.

Testing connections

Finally, it's time to test the secure connection to our web server. When you point your browser to https://<web server IP or hostname>, you will get the well-known message "This connection is untrusted" (see Fig. 7):

Figure 7: Accepting the self-signed certificate as a security exception

In the above situation, you can click on **Add Exception** and then **Confirm Security Exception** – but don't do it yet. Let's first examine the certificate to see if its details match the information that we entered earlier (as shown in the screencast). To do so, click on **View...** --> **Details** tab above and you should see this when you select **Issuer** from the list (see Fig. 8):

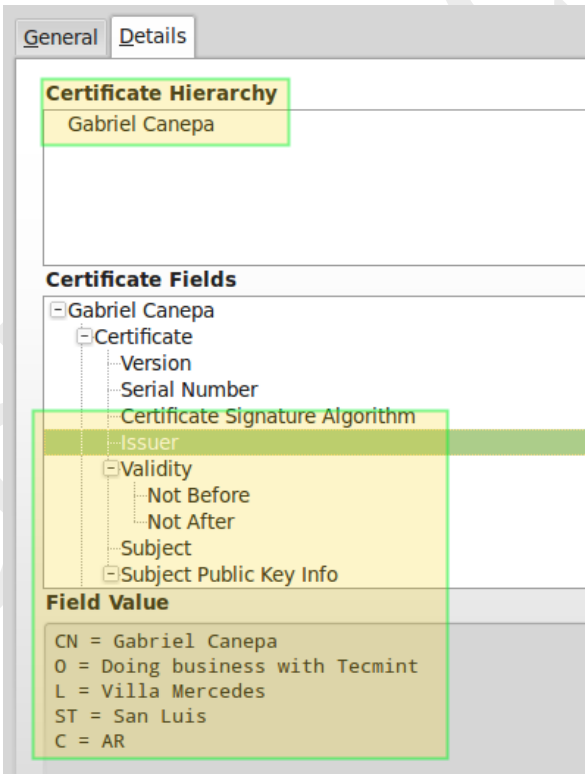


Figure 8: Viewing certificate details

Now you can go ahead, confirm the exception (either for this time or permanently) and you will be taken to your web server's DocumentRoot directory via https, where you can inspect the connection details using your browser's bultin developer tools (In Firefox you can launch it by right clicking on the screen, and choosing **Inspect Element** from the context menu), specifically through the **Network** tab (see Fig. 9):

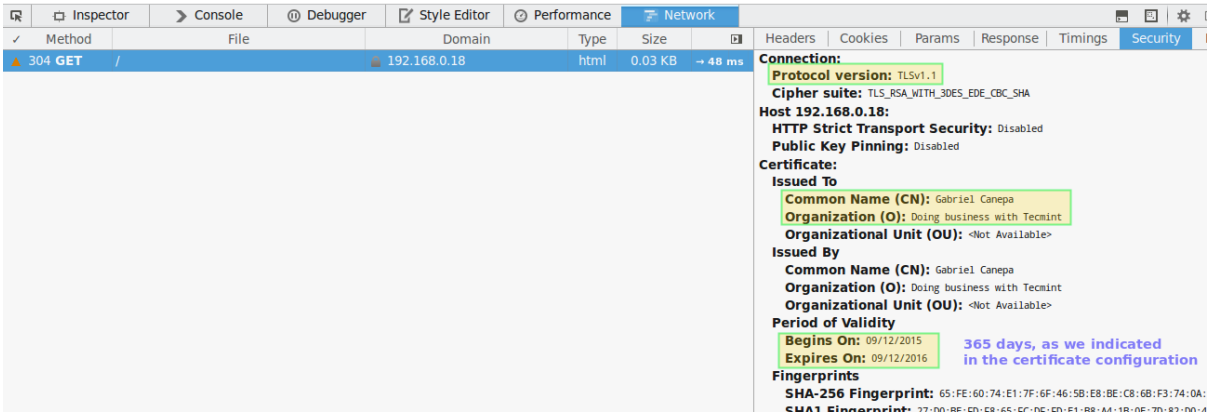


Figure 9: Inspecting connection details

Please note that this is the same information as displayed before, which was entered during the certificate previously.

There's also a way to test the connection using command line tolos (see Fig. 10):

On the left (testing SSLv3):

```
openssl s_client -connect localhost:443 -ssl3
```

On the right (testing TLS):

```
openssl s_client -connect localhost:443 -tls1
```

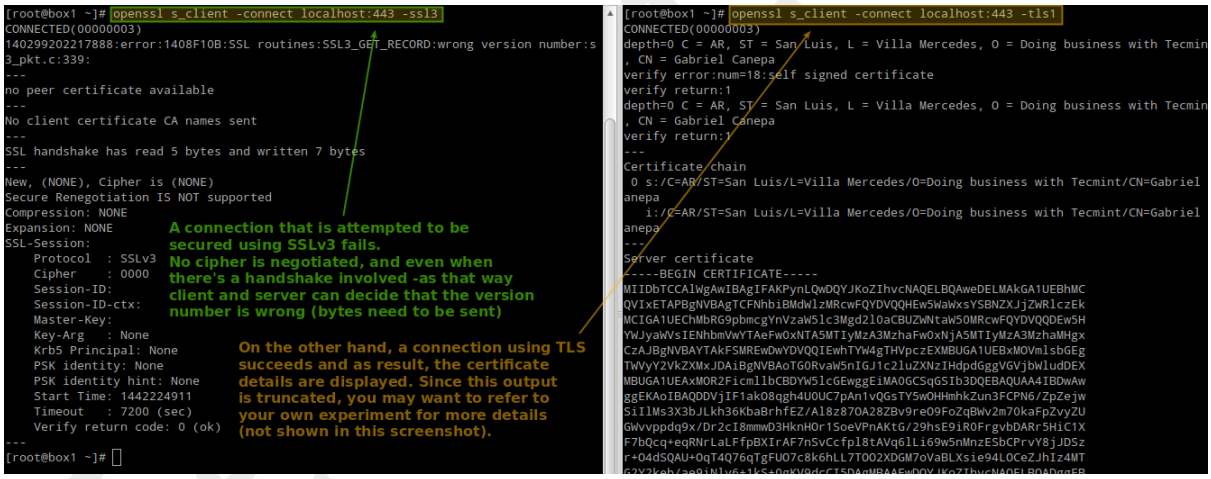


Figure 10: Testing connection using command line tools

Refer to the screenshot above for more details.

Summary

As I'm sure you already know, the presence of HTTPS inspires trust in visitors who may have to enter personal information in your site (from user names and passwords all the way to financial / bank account information). In that case, you will want to get a certificate signed by a trusted Certificate Authority as we explained earlier (the steps to set it up are identical with the exception that you will need to send the CSR to a CA, and you will get the signed certificate back); otherwise, a self-signed certificate as the one used in this tutorial will do.

For more details on the use of NSS, please refer to the online help here.

Chapter 24: Setting up a mail server

Regardless of the many online communication methods that are available today, email remains a practical way to deliver messages from one end of the world to another, or to a person sitting in the office next to ours.

Fig. 1 illustrates the process of email transport starting with the sender until the message reaches the recipient's inbox:

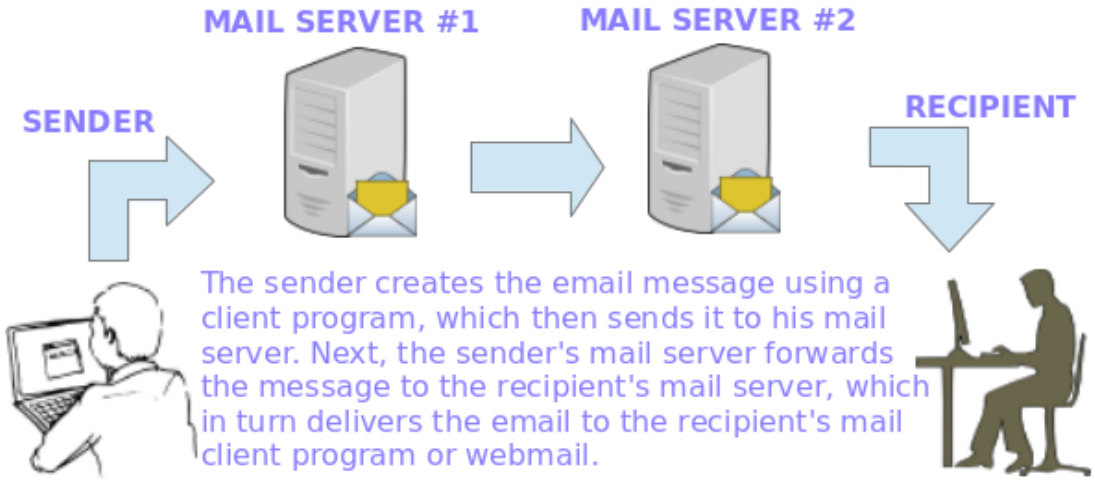


Figure 1: The process of email transport from sender to recipient

To make this possible, several things happen behind the scenes. In order for an email message to be delivered from a client application (such as Thunderbird, Outlook, or webmail services such as Gmail or Yahoo! Mail) to a mail server, and from there to the destination server and finally to its intended recipient, a SMTP (Simple Mail Transfer Protocol) service must be in place in each server. That is the reason why in this article we will explain how to set up a SMTP server in RHEL 7 where emails sent by local users (even to other local users) are forwarded to a central mail server for easier access. In the exam's requirements this is called a null-client setup.

Our test environment will consist of an originating mail server (hostname: box1.mydomain.com / IP: 192.168.0.18) and a central mail server or relayhost (hostname: mail.mydomain.com / IP: 192.168.0.20). For name resolution we will use the well-known /etc/hosts file on both boxes:

```
192.168.0.18    box1.mydomain.com       box1
192.168.0.20    mail.mydomain.com       mail
```

Installing Postfix and firewall / SELinux considerations

To begin, we will need to (in both servers)

1) Install Postfix:

```
yum update && yum install postfix
```

2) Start the service and enable it to run on future reboots:

```
systemctl start postfix

systemctl enable postfix
```

3) Allow mail traffic through the firewall (see Fig. 2):

```
firewall-cmd --permanent --add-service=smtp
```

```
firewall-cmd --add-service=smtp
```

```
[root@box1 ~]# firewall-cmd --list-services
dhcpv6-client ftp http https samba ssh
[root@box1 ~]# firewall-cmd --permanent --add-service=smtp
success
[root@box1 ~]# firewall-cmd --add-service=smtp
success
[root@box1 ~]# firewall-cmd --list-services
dhcpv6-client ftp http https samba smtp ssh
[root@box1 ~]#
```

Figure 2: Allowing mail traffic through the firewall

4) Configure Postfix on box1.mydomain.com

Postfix's main configuration file is located in /etc/postfix/main.cf. This file itself is a great documentation source as the included comments explain the purpose of the program's settings. For brevity, let's display only the lines that need to be edited (yes, you need to leave **mydestination** blank in the originating server; otherwise the emails will be stored locally as opposed to in a central mail server which is what we actually want):

```
myhostname = box1.mydomain.com
mydomain = mydomain.com
myorigin = $mydomain
inet_interfaces = loopback-only
mydestination =
relayhost = 192.168.0.20
```

5) Configure Postfix on mail.mydomain.com

```
myhostname = mail.mydomain.com
mydomain = mydomain.com
myorigin = $mydomain
inet_interfaces = all
mydestination = $myhostname, localhost.$mydomain, localhost, $mydomain
mynetworks = 192.168.0.0/24, 127.0.0.0/8
```

And set the related SELinux boolean to true permanently if not already done (see Fig. 3):

```
setsebool -P allow_postfix_local_write_mail_spool on
```

```
[root@mail ~]# getsebool -a | grep postfix
postfix_local_write_mail_spool --> off
[root@mail ~]# setsebool -P allow_postfix_local_write_mail_spool on
[root@mail ~]# getsebool -a | grep postfix
postfix_local_write_mail_spool --> on
[root@mail ~]#
```

Figure 3: Allowing Postfix in SELinux

The above SELinux boolean will allow Postfix to write to the mail spool in the central server.

5) Restart the service on both servers for the changes to take effect:

```
systemctl restart postfix
```

If Postfix does not start correctly, you can use

```
systemctl -l status postfix

journalctl -xn

postconf -n
```

to troubleshoot.

Testing the mail servers

To test the mail servers, you can use any **M**ail **U**ser **A**gent (most commonly known as **MUA** for short) such as mail or mutt. Since mutt is a personal favorite, I will use it in box1 to send an email to user **tecmint** using an existing file (**mailbody.txt**) as message body (see Fig. 4):

```
mutt -s "Part 9-RHCE series" tecmint@mydomain.com < mailbody.txt
```

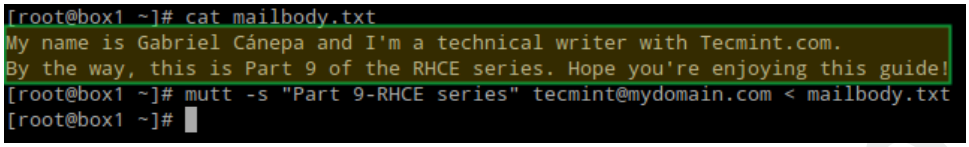


Figure 4: Sending a test message

Now go to the central mail server (mail.mydomain.com), log on as user **tecmint**, and check whether the email was received (see Fig. 5):

```
su - tecmint
mail
```

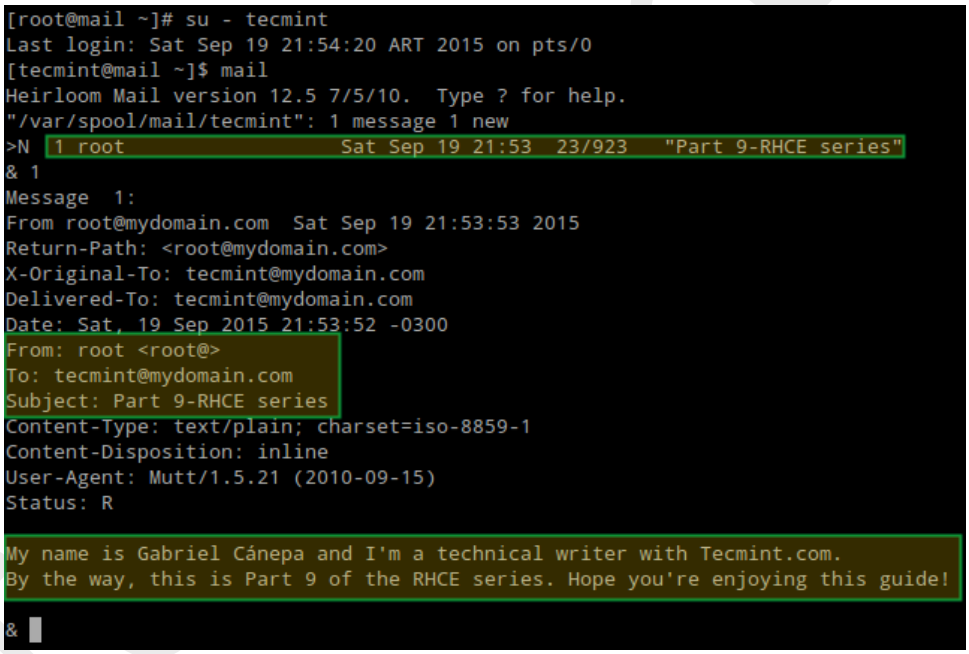


Figure 5: Checking for messages in the recipient's email inbox

If the email was not received, check root's mail spool for a warning or error notification. You may also want to make sure that the SMTP service is running on both servers and that port 25 is open in the central mail server (see Fig. 6):

```
nmap -PN 192.168.0.20
```

```
gacanepa@Mint13 ~ $ nmap -PN 192.168.0.20

Starting Nmap 5.21 ( http://nmap.org ) at 2015-09-19 21:25 ART
Nmap scan report for 192.168.0.20
Host is up (0.88s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
88/tcp   open  kerberos-sec
2049/tcp open  nfs

Nmap done: 1 IP address (1 host up) scanned in 50.00 seconds
gacanepa@Mint13 ~ $
```

Figure 6: Checking that port 25 is open on the mail server

Summary

Setting up a mail server and a relay host as shown in this article is an essential skill that every system administrator must have, and represents the foundation to understand and install a more complex scenario such as a mail server hosting a live domain for several (even hundreds or thousands) of email accounts. (Please note that this kind of setup requires a DNS server, which is out of the scope of this guide).

Finally, I highly recommend you become familiar with Postfix's configuration file (main.cf) and the program's man page. If in doubt, don't hesitate to drop us a line using the form below or using our forum, Linuxsay.com, where you will get almost immediate help from Linux experts from all around the world.

Chapter 25: Setting up a cache-only DNS server

There are several types of DNS servers: master, slave, forwarding and cache, to name a few examples, with cache-only DNS being the one that is easier to setup. Since DNS uses the UDP protocol, it improves the query time because it does not require an acknowledgement.

The cache-only DNS server is also known as resolver. It will query DNS records, get all DNS information from other servers, and store each query request in its cache for later use so that when we perform the same request in the future, it will serve from its cache, thus reducing the response time even more.

Testing Environment

DNS server : dns.tecmintlocal.com (Red Hat Enterprise Linux 7.1)

Server IP Address : 192.168.0.18

Client : node1.tecmintlocal.com (CentOS 7.1)

Client IP Address : 192.168.0.29

Step 1: Installing Cache-Only DNS

1. The Cache-Only DNS server can be installed through the **bind** package. Let's do a small search for the package name if we don't remember the full package name using the command below (see Fig. 1).

```
yum search bind
```

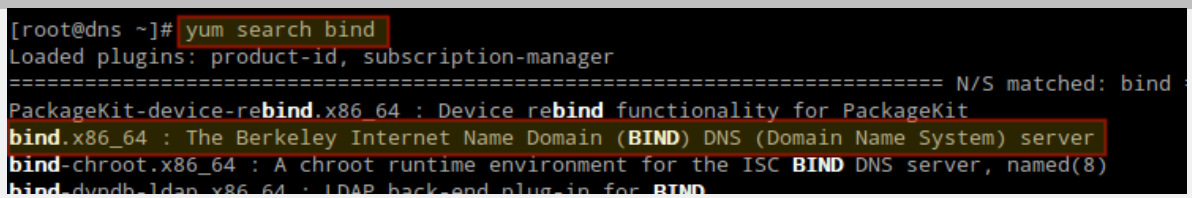


Figure 1: Search Bind Package

2. In the above result, you will see several packages. From those, we need to choose the **bind** and **bind-utils** packages. Let's install them using following **yum** (see Fig. 2):

```
yum install bind bind-utils -y
```

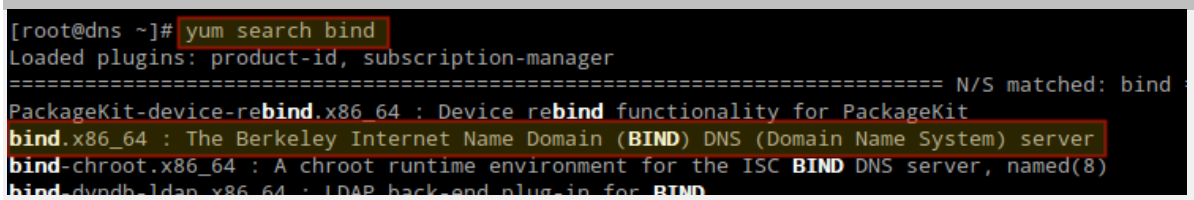


Figure 2: Install DNS utilities

Step 2: Configure Cache-Only DNS

3. Once DNS packages are installed we can go ahead and configure DNS. Open and edit /etc/named.conf using your preferred text editor. Make the changes suggested below (or you can use your settings as per your requirements). Refer to Fig. 3 for more details:

```
listen-on port 53 { 127.0.0.1; any; };
allow-query     { localhost; any; };
```

```
allow-query-cache       { localhost; any; };
```

```
// named.conf
//
// Provided by Red Hat bind package to configure the ISC BIND named
// server as a caching only nameserver (as a localhost DNS resolver
//
// See /usr/share/doc/bind*/sample/ for example named configuration
//

options {
        listen-on port 53 { 127.0.0.1; any; };
        listen-on-v6 port 53 { ::1; };
        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     { localhost; any; };
        allow-query-cache       { localhost; any; };
        /*
         - If you are building an AUTHORITATIVE DNS server, do NOT
         - If you are building a RECURSIVE (caching) DNS server, yo
           recursion.
```

Figure 3: Configure Cache Only DNS

These directives instruct the DNS server to listen on UDP port 53, and to allow queries and caches responses from localhost and any other machine that reaches the server. It is important to note that the ownership of this file must be set to **root:named**.

4. If SELinux is enabled, after editing the configuration file we need to make sure that its context is set to **named_conf_t** as shown in Fig. 4 (same thing for the auxiliary file /etc/named.rfc1912.zones):

```
ls -lZ /etc/named.conf
ls -lZ /etc/named.rfc1912.zones
```

Otherwise, configure the SELinux context before proceeding:

```
semanage fcontext -a -t named_conf_t /etc/named.conf
semanage fcontext -a -t named_conf_t /etc/named.rfc1912.zones
```

Additionally, we need to test the DNS configuration now for some syntax error before starting the bind service:

```
named-checkconf /etc/named.conf
```

After the syntax verification results seems perfect, restart the service to take effect for above changes, and make the service to run persistently across boots, and then check its status, as shown in Fig. 4:

```
systemctl restart named
systemctl enable named
systemctl status named
```

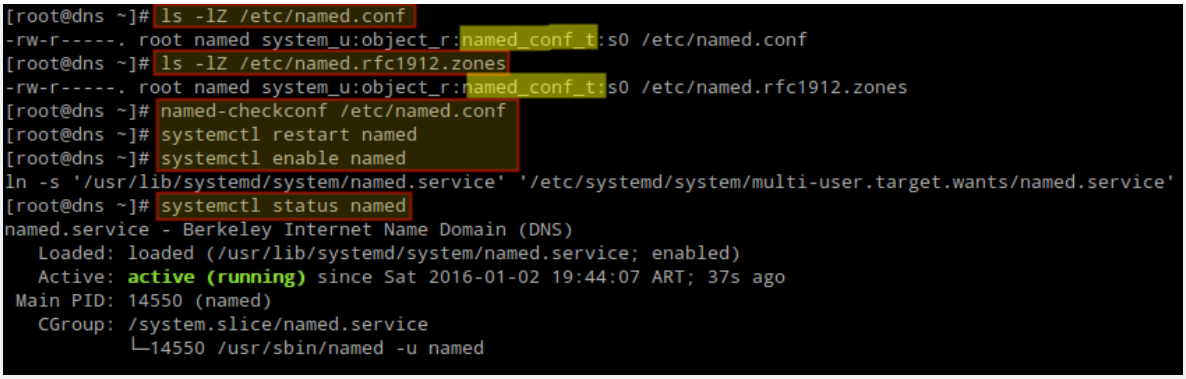


Figure 4: Configure and Start DNS

5. Next, open the port 53 on the firewall (see Fig. 5):

```
firewall-cmd --add-port=53/udp
firewall-cmd --add-port=53/udp --permanent
```

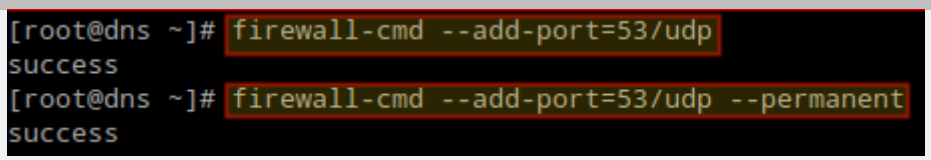


Figure 5: Opening UDP port 53 in firewalld

Step 4: Chroot Cache-Only DNS

6. If you want to run the DNS Cache-server under chroot environment, you need to install the chroot package only. This will not require any further configuration, as it is by default hard-link to chroot.

```
yum install bind-chroot -y
```

Once the chroot package has been installed, you can restart named to take the new changes:

```
systemctl restart named
```

7. Next, create a symbolic link (also named /etc/named.conf) inside /var/named/chroot/etc/:

```
ln -s /etc/named.conf /var/named/chroot/etc/named.conf
```

Step 5: Client Side DNS Setup

8. Add the DNS Cache servers IP 192.168.0.18 as resolver to the client machine. Edit **/etc/sysconfig/network-scripts/ifcfg-enp0s3** as shown in Fig. 6:

DNS=192.168.0.18

```
HWADDR=08:00:27:CB:39:E5
TYPE=Ethernet
BOOTPROTO=static
IPADDR=192.168.0.29
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
NM_CONTROLLED=no
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=enp0s3
UUID=8483ad3e-de0a-4cca-b8fc-d2cf9a1e169f
ONBOOT=yes
DNS=192.168.0.18
```

Figure 6: Editing the client's network configuration

And /etc/resolv.conf as follows:

```
nameserver 192.168.0.18
```

9. Finally it's time to check our cache server. To do this, you can use **dig** or **nslookup**. Choose any website and query it twice (we will use facebook.com as an example). Note that with **dig** the second time the query is completed much faster because it is being served from the cache (see Fig. 7):

```
dig facebook.com
```

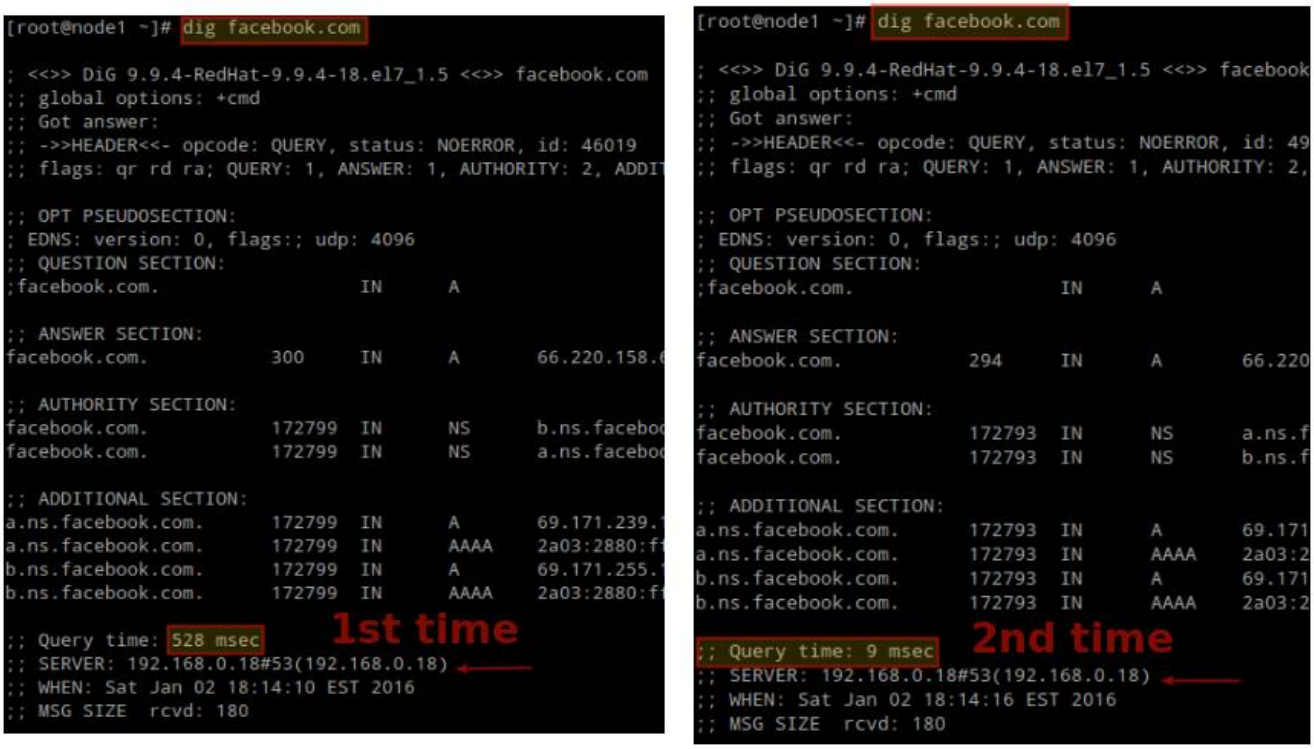


Figure 7: Cached queries are returned faster

You can also use nslookup to verify that the DNS server is working as expected (see Fig. 8):

```
nslookup facebook.com
```

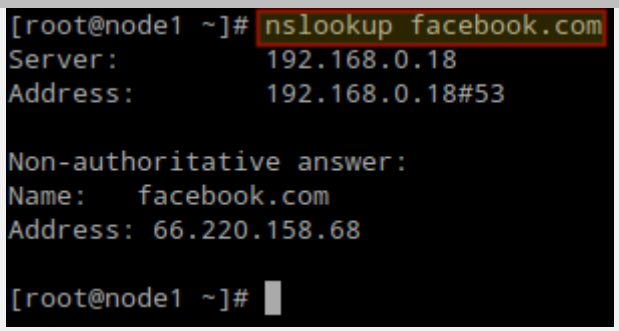


Figure 8: Checking the DNS server

Summary

In this article we have explained how to set up a DNS Cache-only server in Red Hat Enterprise Linux 7, and tested it in a client machine. Feel free to let us know if you have any questions or suggestions.

Chapter 26: Network bonding

When a system administrator wants to increase the bandwidth available and provide redundancy and load balancing for data transfers, a kernel feature known as network bonding allows to get the job done in a cost-effective way.

In simple words, bonding means aggregating two or more physical network interfaces (called *slaves*) into a single, logical one (called *master*). If a specific NIC (Network Interface Card) experiences a problem, communications are not affected significantly as long as the other(s) remain active.

Enabling and configuring bonding

By default, the bonding kernel module is not enabled. Thus, we will need to load it and ensure it is persistent across boots. When used with the --first-time option, modprobe will alert us if loading the module fails:

```
modprobe --first-time bonding
```

The above command will load the bonding module for the current session. In order to ensure persistency, create a .conf file inside /etc/modules-load.d with a descriptive name, such as /etc/modules-load.d/bonding.conf:

```
echo "# Load the bonding kernel module at boot" > /etc/modules-load.d/bonding.conf
echo "bonding" >> /etc/modules-load.d/bonding.conf
```

Now reboot your server and once it restarts, make sure the bonding module is loaded automatically, as seen in Fig. 1:

```
lsmod | grep bonding
```

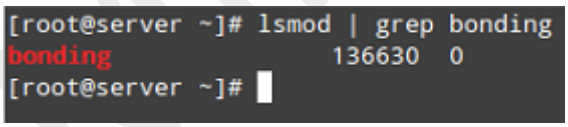


Figure 1: Verifying the status of the bonding module

In this article we will use 3 interfaces (**enp0s3**, **enp0s8**, and **enp0s9**) to create a bond, named conveniently **bond0**.

To create bond0, we can either use nmtui, the text interface for controlling NetworkManager. When invoked without arguments from the command line, nmtui brings up a text interface that allows you to edit an existing connection, activate a connection, or set the system hostname. Choose *Edit connection* → *Add* → *Bond* as illustrated in Fig. 2:

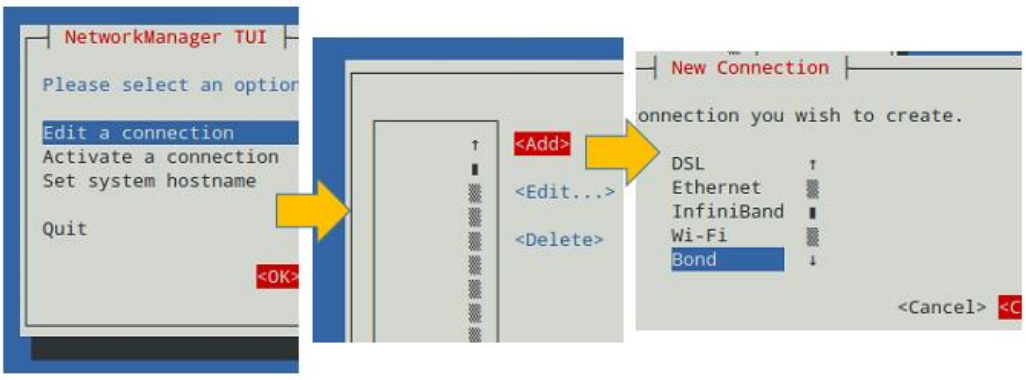


Figure 2: First steps in creating bond0

In the *Edit Connection* screen, add the slave interfaces (enp0s3, enp0s8, and enp0s9 in our case) and give them a descriptive (Profile) name (for example, NIC #1, NIC #2, and NIC #3, respectively).

In addition, you will need to set a name and device for the bond (*TecmintBond* and *bond0* in Fig. 3, respectively) and an IP address for bond0, enter a gateway address, and the IPs of DNS servers. Note that you do not need to enter the MAC address of each interface since nmtui will do that for you. You can leave all other settings as default. See Fig. 3 for more details.

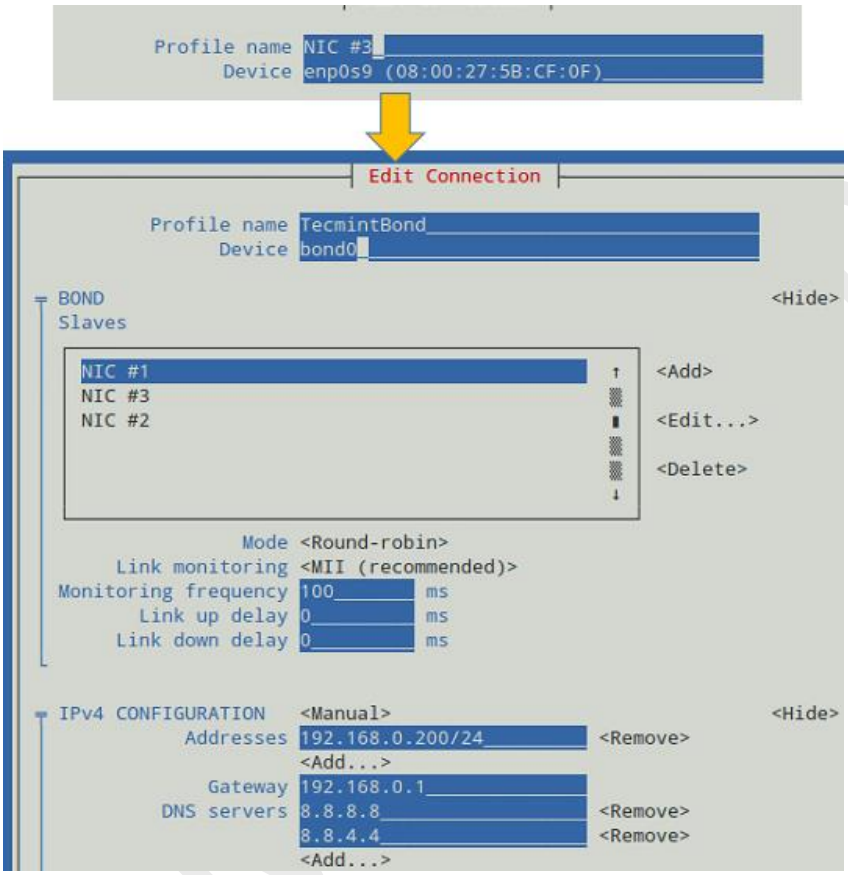


Figure 3: Configuring bond0

When you're done, go to the bottom of the screen and choose OK (see Fig. 4):

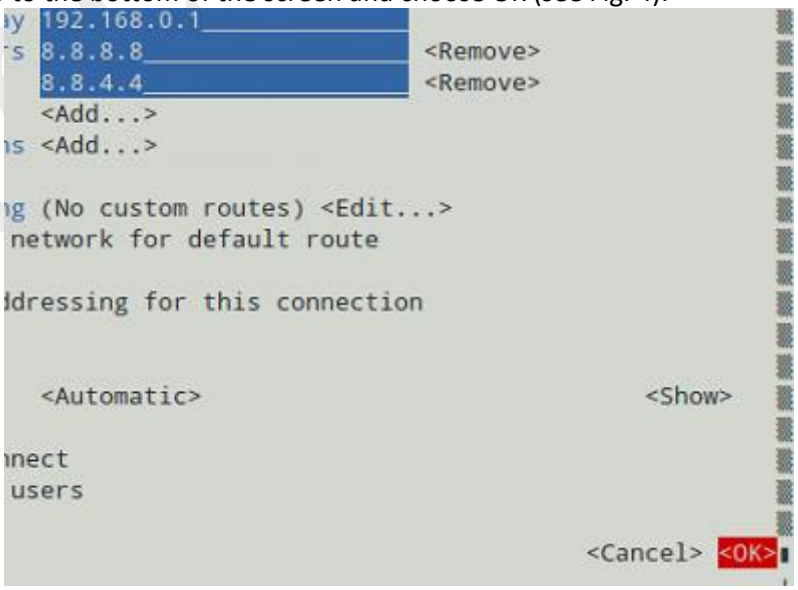


Figure 4: Finishing configuration of bond0

And you're done. Now you can exit the text interface and return to the command line, where you will enable the newly created interface:

```
ip link set dev bond0 up
```

After that, you can see that bond0 is UP and is assigned 192.168.0.200, as seen in Fig. 5:

```
ip addr show bond0
```

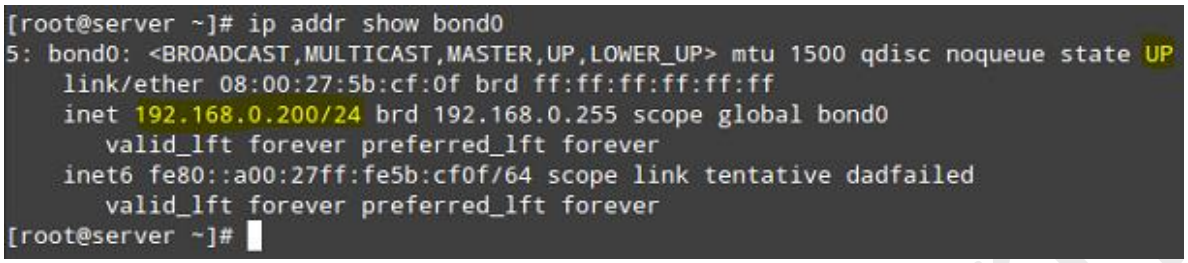


Figure 5: Viewing the status and address of bond0

Testing bonding

To verify that bond0 actually works, you can either ping its IP address from another machine, or what's even better, watch the kernel interface table in real time (well, the refresh time in seconds is given by the **-n** option) to see how network traffic is distributed between the three network interfaces, as shown in Fig. 6. The **-d** option is used to highlight changes when they occur:

```
watch -d -n1 netstat -i
```

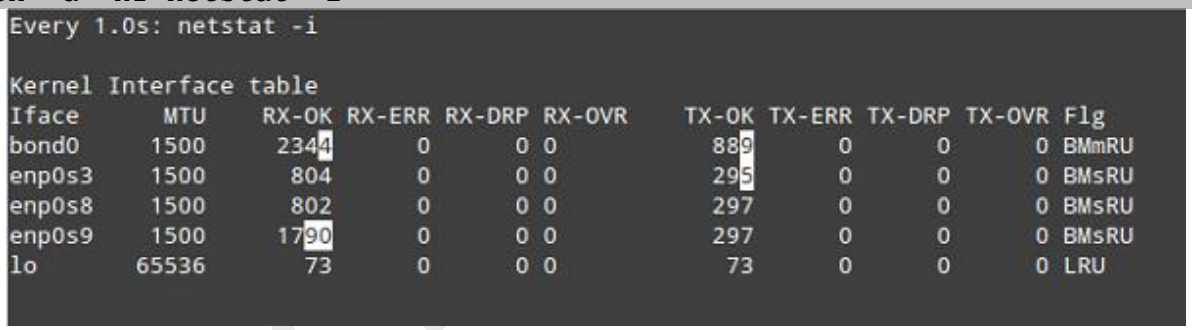


Figure 6: Viewing the kernel interface table

It is important to note that there are several bonding modes, each with its distinguishing characteristics. They are documented in section 4.5 of the Red Hat Enterprise Linux 7 Network Administration guide. Depending on your needs, you will choose one or the other.

In our current setup, we chose the Round-robin mode (see Fig. 3), which ensures packets are transmitted beginning with the first slave in sequential order, ending with the last slave, and starting with the first again. The Round-robin alternative is also called mode 0, and provides load balancing and fault tolerance.

To change the bonding mode, you can use **nmtui** as explained before (see also Fig. 7):

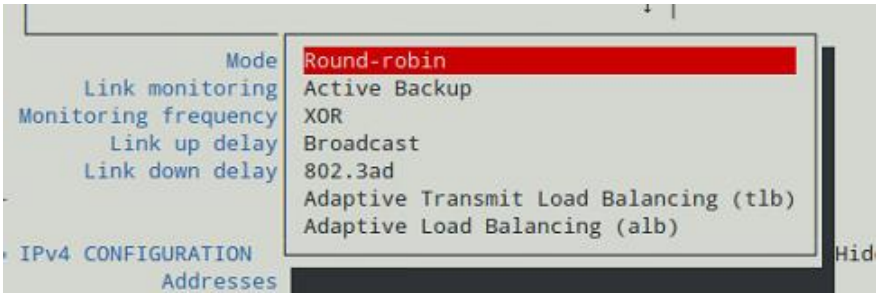


Figure 7: Changing bonding mode using nmtui

If we change it to Active Backup, we will be prompted to choose a slave that will the only one active interface at a given time. If such card fails, one of the remaining slaves will take its place and becomes active.

Let's choose enp0s3 to be the primary slave, bring bond0 down and up again, restart the network, and display the kernel interface table (see Fig. 8). Note how data transfers (**TX-OK** and **RX-OK**) are now being made over enp0s3 only:

```
ip link set dev bond0 down
ip link set dev bond0 up
systemctl restart network
```

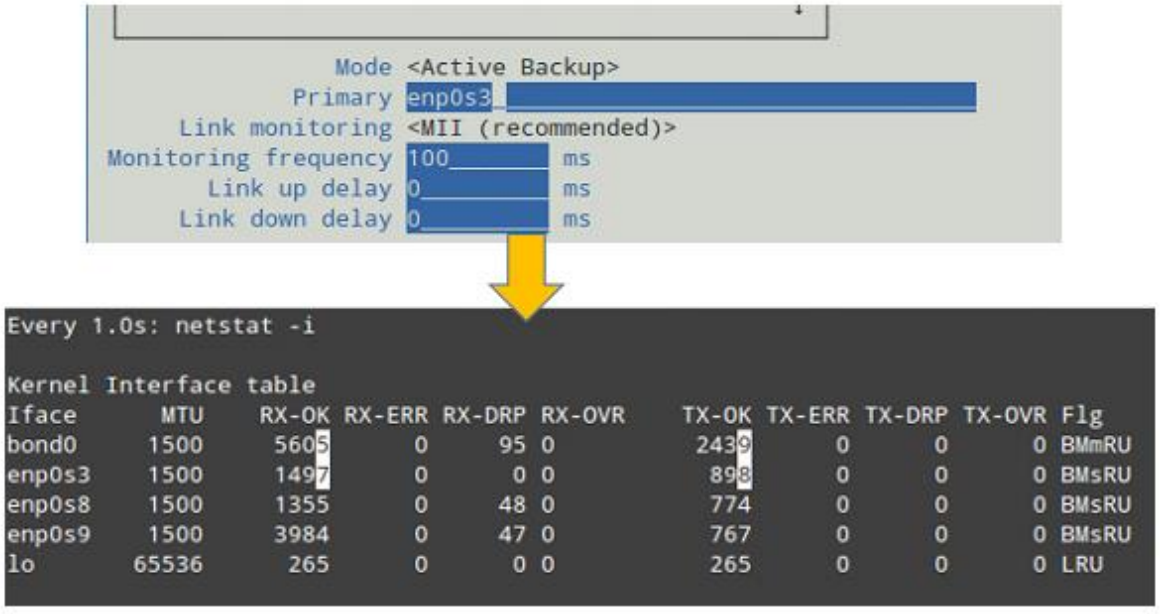


Figure 8: A bond acting in Active Backup mode

Alternatively, you can view the bond as the kernel sees it (see Fig. 9):

```
cat /proc/net/bonding/bond0
```

```
[root@server ~]# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: enp0s3 (primary_reselect always)
Currently Active Slave: enp0s3
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: enp0s3
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:4e:59:37
Slave queue ID: 0

Slave Interface: enp0s8
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:17:b6:c1
Slave queue ID: 0

Slave Interface: enp0s9
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:5b:cf:0f
Slave queue ID: 0
[root@server ~]#
```

Figure 9: Viewing the bond as the kernel sees it

Summary

In this chapter we have discussed how to set up and configure bonding in Red Hat Enterprise Linux 7 in order to increase bandwidth along with load balancing and redundancy for data transfers. As you take the time to explore other bonding modes, you will come to master the concepts and practice related with this topic of the certification. Best of luck!

Chapter 27: Create Centralized Secure Storage using iSCSI Target / Initiator

iSCSI is a block level Protocol for managing storage devices over TCP/IP Networks, specially over long distances. iSCSI target is a remote hard disk presented from an remote iSCSI server (or) target. On the other hand, the iSCSI client is called the Initiator, and will access the storage that is shared in the Target machine.

The following machines have been used in this article:

1) Server (Target):

- Operating System – Red Hat Enterprise Linux 7
- iSCSI Target IP – 192.168.0.29
- Ports Used : TCP 860, 3260

2) Client (Initiator):

- Operating System – Red Hat Enterprise Linux 7
- iSCSI Target IP – 192.168.0.30
- Ports Used : TCP 3260

To install the packages needed for the target (we will deal with the client later), do:

```
yum install targetcli -y
```

When the installation completes, we will start and enable the service as follows:

```
systemctl start target
systemctl enable target
```

Finally, we need to allow the service in firewalld:

```
firewall-cmd --add-service=iscsi-target
firewall-cmd --add-service=iscsi-target --permanent
```

And last but not least, we must not forget to allow the iSCSI target discovery:

```
firewall-cmd --add-port=860/tcp
firewall-cmd --add-port=860/tcp --permanent
firewall-cmd --reload
```

Defining LUNs in Target Server

Before proceeding to defining LUNs in the Target, we need to create two logical volumes as explained in Part 6 (“Configuring system storage”). This time we will name them **vol_projects** and **vol_backups** and place them inside a volume group called **vg00**, as shown in Fig. 1. Feel free to choose the space allocated to each LV:

```
[root@server ~]# lvdisplay vg00/vol_backups vg00/vol_projects
  --- Logical volume ---
  LV Path                /dev/vg00/vol_projects
  LV Name                vol_projects
  VG Name                vg00
  LV UUID                03os1b-encP-XVDX-wAHY-j1LK-iAdf-l8WD1y
  LV Write Access        read/write
  LV Creation host, time centos, 2016-02-20 19:27:39 -0500
  LV Status              available
  # open                 1
  LV Size                7.50 GiB
  Current LE             1920
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           253:2

  --- Logical volume ---
  LV Path                /dev/vg00/vol_backups
  LV Name                vol_backups
  VG Name                vg00
  LV UUID                E5xsjt-lKZ1-nP5d-qhNX-KzQi-trhv-67Nd9c
  LV Write Access        read/write
  LV Creation host, time centos, 2016-02-20 19:27:47 -0500
  LV Status              available
  # open                 1
  LV Size                8.49 GiB
  Current LE             2174
  Segments               3
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           253:3

[root@server ~]#
```

Figure 1: The Logical Volumes

After creating the LVs, we are ready to define the LUNs in the Target in order to make them available for the client machine. As shown in Fig. 2, we will open a targetcli shell and issue the following commands, which will create two block backstores (local storage resources that represent the LUN the initiator will actually use) and an Iscsi Qualified Name (IQN), a method of addressing the target server. Please refer to Page 32 of RFC 3720 for more details on the structure of the IQN. In particular, the text after the colon character (:**tgt1**) specifies the name of the target, while the text before (**server**:) indicates the hostname of the target inside the domain.

```
targetcli
cd backstores
cd block
create server.backups /dev/vg00/vol_backups
create server.projects /dev/vg00/vol_projects
cd /iscsi
create iqn.2016-02.com.tecmint.server:tgt1
```

```
[root@server ~]# targetcli
targetcli shell version 2.1.fb41
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/> cd backstores
/backstores> cd block
/backstores/block> create server.backups /dev/vg00/vol_backups
Created block storage object server.backups using /dev/vg00/vol_backups.
/backstores/block> create server.projects /dev/vg00/vol_projects
Created block storage object server.projects using /dev/vg00/vol_projects.
/backstores/block> cd /iscsi
/iscsi> create iqn.2016-02.com.tecmint.server:tgt1
Created target iqn.2016-02.com.tecmint.server:tgt1.
Created TPG 1.
Global pref auto_add_default_portal=true
Created default portal listening on all IPs (0.0.0.0), port 3260.
/iscsi>
```

Figure 2: Creating backstores

With the above step, a new TPG (Target Portal Group) was created along with the default portal (a pair consisting of an IP address and a port which is the way initiators can reach the target) listening on port 3260 of all IP addresses. If you want to bind your portal to a specific IP (the Target's main IP, for example), delete the default portal and create a new one as follows (otherwise, skip the following targetcli commands. Note that for simplicity we have skipped them as well):

```
cd /iscsi/iqn.2016-02.com.tecmint.server:tgt1/tpg1/portals
delete 0.0.0.0 3260
create 192.168.0.29 3260
```

Now we are ready to proceed with the creation of LUNs. Note that we are using the backstores we previously created (server.backups and server.projects). This process is illustrated in Fig. 3:

```
cd iqn.2016-02.com.tecmint.server:tgt1/tpg1/luns
create /backstores/block/server.backups
create /backstores/block/server.projects
```

```
/iscsi> cd iqn.2016-02.com.tecmint.server:tgt1/tpg1/luns
/iscsi/iqn.20...gt1/tpg1/luns> create /backstores/block/server.backups
Created LUN 0.
/iscsi/iqn.20...gt1/tpg1/luns> create /backstores/block/server.projects
Created LUN 1.
/iscsi/iqn.20...gt1/tpg1/luns>
```

Figure 3: Creating LUNs

The last part in the Target configuration consists of creating an Access Control List to restrict access on a per-initiator basis. Since our client machine is named "client", we will append that text to the IQN. Refer to Fig. 4 for details:

```
cd ../acls
create iqn.2016-02.com.tecmint.server:client
```

```
/iscsi/iqn.20...gt1/tpg1/luns> cd ../acls
/iscsi/iqn.20...gt1/tpg1/acls> create iqn.2016-02.com.tecmint.server:client
Created Node ACL for iqn.2016-02.com.tecmint.server:client
Created mapped LUN 1.
Created mapped LUN 0.
```

Figure 4: Setting an ACL for the client

At this point we can the targetcli shell to show all configured resources, as we can see in Fig. 5:

```
targetcli
cd /
ls
```

```
[root@server ~]# targetcli
targetcli shell version 2.1.fb41
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/backstores/b...erver.backups> cd /
/> ls
o- / ..... [...]
  o- backstores ..... [...]
  | o- block ..... [Storage Objects: 2]
  | | o- server.backups ..... [/dev/vg00/vol_backups (8.5GiB) write-thru activated]
  | | o- server.projects ..... [/dev/vg00/vol_projects (7.5GiB) write-thru activated]
  | o- fileio ..... [Storage Objects: 0]
  | o- pscsi ..... [Storage Objects: 0]
  | o- ramdisk ..... [Storage Objects: 0]
  o- iscsi ..... [Targets: 1]
  | o- iqn.2016-02.com.tecmint.server:tgt1 ..... [TPGs: 1]
  |   o- tpg1 ..... [no-gen-acls, no-auth]
  |     o- acls ..... [ACLs: 1]
  |     | o- iqn.2016-02.com.tecmint.server:client ..... [Mapped LUNs: 2]
  |     |   o- mapped_lun0 ..... [lun0 block/server.backups (rw)]
  |     |   o- mapped_lun1 ..... [lun1 block/server.projects (rw)]
  |     o- luns ..... [LUNs: 2]
  |     | o- lun0 ..... [block/server.backups (/dev/vg00/vol_backups)]
  |     | o- lun1 ..... [block/server.projects (/dev/vg00/vol_projects)]
  |     o- portals ..... [Portals: 1]
  |       o- 0.0.0.0:3260 ..... [OK]
  o- loopback ..... [Targets: 0]
/>
```

Figure 5: Listing configured resources

To quit the targetcli shell, simply type exit and press Enter. The configuration will be saved automatically to /etc/target/saveconfig.json.

As you can see in Fig. 5 above, we have a portal listening on port 3260 of all IP addresses as expected. We can verify that using netstat (see Fig. 6):

```
netstat -npltu | grep 3260
```

```
[root@server ~]# netstat -tulnp | grep 3260
tcp        0      0 0.0.0.0:3260            0.0.0.0:*               LISTEN      -
[root@server ~]#
```

Figure 6: Using netstat to verify the portal status

This concludes the Target configuration. Feel free to restart the system and verify that all settings survive a reboot. If not, make sure to open the necessary ports in the firewall configuration and to start the target service on boot. We are now ready to set up the Initiator and to connect to the client.

Setting up the Initiator

In the client we will need to install the iscsi-initiator-utils package, which provides the server daemon for the iSCSI protocol (iscsid) as well as iscsiadm, the administration utility:

```
yum update && yum install iscsi-initiator-utils
```

Once the installation completes, open /etc/iscsi/initiatorname.iscsi and replace the default initiator name (commented in Fig. 7) with the name that was previously set in the ACL on the server (iqn.2016-02.com.tecmint.server:client). Then save the file and run iscsiadm in discovery mode pointing to the target. If successful, this command will return the target information as shown in Fig. 7:

```
iscsiadm -m discovery -t st -p 192.168.0.29
```

Figure 7: Discovering the target

The next step consists in restarting and enabling the iscsid service:

```
systemctl start iscsid
systemctl enable iscsid
```

and contacting the target in node mode. This should result in kernel-level messages, which -when captured through dmesg- show the device identification that the remote LUNs have been given in the local system (sde and sdf as seen in Fig. 8):

```
iscsiadm -m node -T iqn.2016-02.com.tecmint.server:tgt1 -p 192.168.0.29 -l
```

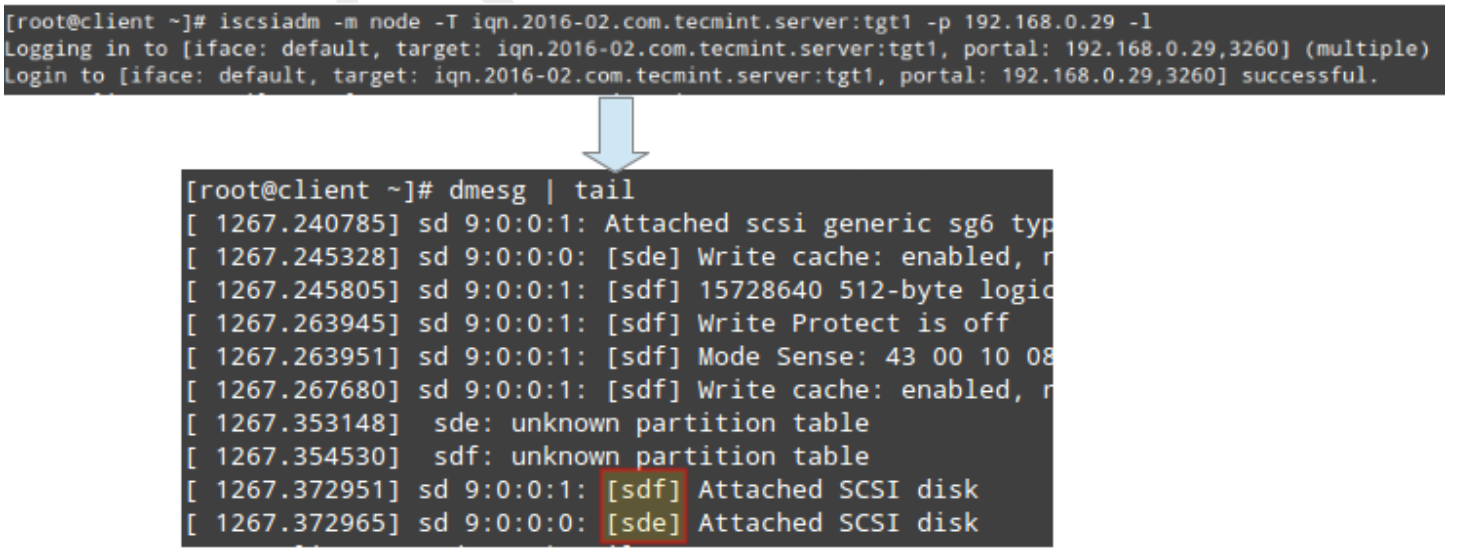


Figure 8: Attaching the iSCSI devices to the client

From this point on, you can create partitions, or even LVs (and filesystems on top of them) as you would do with any other storage device. For simplicity, we will create a primary partition on each disk that will occupy its entire available space, and format it with ext4.

Finally, let's mount /dev/sde1 and /dev/sdf1 on /projects and /backups, respectively (note that these directories must be created first):

```
mount /dev/sde1 /projects
mount /dev/sdf1 /backups
```

Additionally, you can add two entries in /etc/fstab in order for both filesystems to be mounted automatically at boot using each filesystem's UUID as returned by blkid. Note that the **_netdev** mount option must be used in order to defer the mounting of these filesystems until the network service has been started. Refer to Fig. 9 for more details:

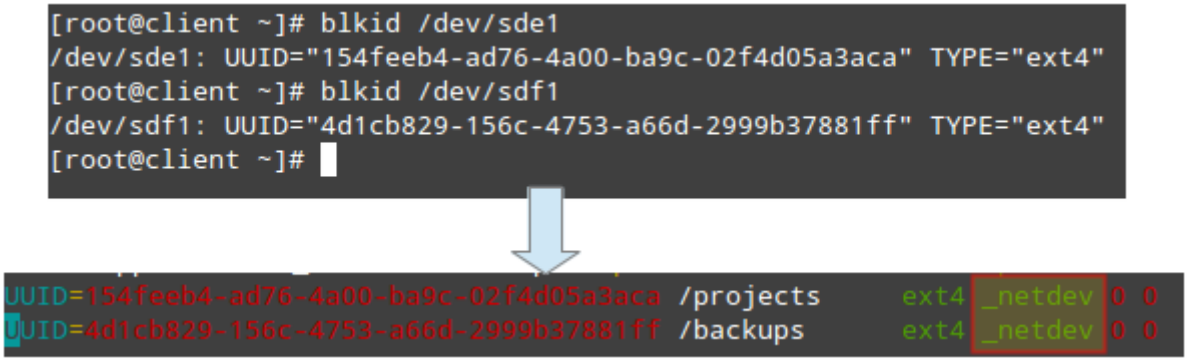


Figure 9: Adding the iSCSI devices to /etc/fstab

You can now use these devices as you would with any other storage media.

Summary

In this article we have covered how to set up and configure an iSCSI Target and an Initiator. Although the first task is not part of the required competencies of the EX300 (RHCE) exam, it is needed in order to implement the second topic.

Chapter 28: Database server

A database server is a critical component of the network infrastructure necessary for today's applications. Without the ability to store, retrieve, update, and delete data (when needed), the usefulness and scope of web and desktop apps becomes very limited. In addition, knowing how to install, manage, and configure a database server (so that it operates as expected) is an essential skill that every system administrator must have.

In this article we will briefly review how to install and secure a MariaDB database server and then we will explain how to configure it.

Installing and securing a MariaDB server

In CentOS 7.x, MariaDB replaced MySQL, which still can be found in the Ubuntu (along with MariaDB). The same is true for openSUSE. For brevity, we will only use MariaDB in this tutorial, but please note that -besides having different names and development philosophies-, both Relational DataBase Management Systems (RDBMSs for short) are almost identical. This means that the client-side commands are the same on both MySQL and MariaDB, and the configuration files are named and located in the same places.

To install MariaDB, do:

```
yum update && yum install mariadb mariadb-server
```

Once the above packages have been installed, make sure the database service is running and has been activated to start on boot:

```
systemctl start mariadb && systemctl enable mariadb
```

Then run the mysql_secure_installation script. This process will allow you to 1) set / reset the password for the RDBMS root user, 2) remove anonymous logins (thus enabling only users with a valid account to log in to the RDBMS), 3) disable root access for machines other than localhost, 4) remove the test database (which anyone can access), and 5) activate the changes associated with 1 through 4. For a more detailed description of this process, you can refer to the Post installation section in Install MariaDB 5.5 Database in RHEL/CentOS/Fedora and Debian/Ubuntu.

Configuring the database server

The default configuration options are read from the following files in the given order: /etc/mysql/my.cnf, /etc/my.cnf, and ~/.my.cnf. Most often, only /etc/my.cnf exists. It is on this file that we will set the server-wide settings (which can be overridden with the same settings in ~/.my.cnf for each user).

The first thing that we need to note about my.cnf is that settings are organized into categories (or groups) where each category name is enclosed with square brackets. Server system configurations are given in the [mysqld] section, where typically you will find only the first two settings in the table below. The rest are other frequently used options (where indicated, we will change the default value with a custom one of our choosing):

Setting and description	Default value
datadir is the directory where the data files are stored.	`datadir=/var/lib/mysql`
socket indicates the name and location of the socket file that is used for local client connections. Keep in mind that a socket file is a resource that is utilized to pass information between applications.	`socket=/var/lib/mysql/mysql.sock`
bind_address is the address where the database server will listen on for TCP/IP connections. If you need your server to listen on more than one IP address, leave out this setting (0.0.0.0 which means it will listen on all IP addresses assigned to this specific host). We will change this to instruct the service to listen only on its main address (192.168.0.13): `bind-address=192.168.0.13`	`bind_address=0.0.0.0`
port represents the port where the database server will be listening. We will replace the default value(3306) with 20500 (but we need to make sure nothing else is using that port): `port=20500` While some people will argue that security through obscurity is not good practice, changing the default application ports for higher ones is a rudimentary -yet effective- method to discourage port scans.	`port=3306`
innodb_buffer_pool_size is the buffer pool (in bytes) of memory that is allocated for data and indexes that are accessed frequently when using	`innodb_buffer_pool_size=134217728`

Innodb (which is the default in MariaDB) or XtraDB as storage engine. We will replace the default value with 256 MB: `innodb_buffer_pool_size=256M`	
skip_name_resolve indicates whether hostnames will be resolved or not on incoming connections. If set to 1, as we will do in this guide, only IP addresses. Unless you require hostnames to determine permissions, it is advisable to disable this variable (in order to speed up connections and queries) by setting its value to 1: `skip_name_resolve=1`	`skip_name_resolve=0`
query_cache_size represents the size (in bytes) available to the query cache in disk, where the results of SELECT queries are stored for future use when an identical query (to the same database and using the same protocol and same character set) is performed. You should choose a query cache size that matches your needs based on 1) the number of repetitive queries, and 2) the approximate number of records those repetitive queries are expected to return. We will set this value to 100 MB for the time being: `query_cache_size=100M`	`query_cache_size=0` (which means it is disabled by default)
max_connections is the maximum number of simultaneous client connections to the server. We will set this value to 30: `max_connections=30` Each connection will use a thread, and thus will consume memory. Take this fact into account while setting	`max_connections=151`

max_connections.	
thread_cache_size indicates the numbers of threads that the server allocates for reuse after a client disconnects and frees thread(s) previously in use. In this situation, it is cheaper (performance-wise) to reuse a thread than instantiating a new one. Again, this depends on the number of connections you are expecting. We can safely set this value to half the number of max_connections: `thread_cache_size=15`	`thread_cache_size=0` (disabled by default)

In CentOS, we will need to tell SELinux to allow MariaDB to listen on a non-standard port (20500) before restarting the service:

```
yum install policycoreutils-python
semanage port -a -t mysqld_port_t -p tcp 20500
```

Then restart the service.

Checking the configuration

To assist us in checking and tuning the configuration as per our specific needs, we can install mysqltuner (a script that will provide suggestions to improve the performance of our database server and increase its stability):

```
wget https://github.com/major/MySQLTuner-perl/tarball/master
tar xzf master
```

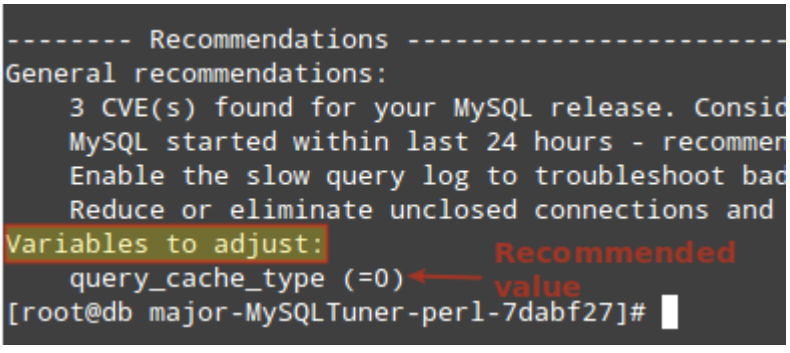
Then change directory into the folder extracted from the tarball (the exact version may differ in your case):

```
cd major-MySQLTuner-perl-7dabf27
```

and run it (you will be prompted to enter the credentials of your administrative MariaDB account)

```
./mysqltuner.pl
```

The output of the script is in itself very interesting, but let's skip to the bottom where the variables to adjust are listed with the recommended value:



The query_cache_type setting indicates whether the query cache is disabled (0) or enabled (1). In this case, mysqltuner is advising us to disable it. So why are we advised to deactivate it now? The reason is that the query cache is useful mostly in high-read / low-write scenarios (which is not our case, since we just installed the database server).

WARNING: Before making changes to the configuration of a production server, you are highly encouraged to consult an expert database administrator to ensure that a recommendation given by **mysqltuner** will not impact negatively on an existing setting.

Creating a database / schema

Of course, having a MariaDB server without an actual database that we can use is pointless. In this section we will learn how to create a database and tables to store information persistently, which we can later query and use.

Please note that you are expected to know the fundamentals of DML (Data Manipulation Language) statements in standard SQL. If you don't feel confident with your current skills,

To access the MariaDB prompt, use the following command. You will be prompted to enter the root password you chose when you ran the **mysql_secure_installation script**.

```
mysql -u root -p
```

To see the current list of databases, do

```
SHOW DATABASES;
```

Also, you can use

```
SHOW SCHEMAS;
```

Although in other RDBMS (such as SQL Server and Oracle) a database and a schema are two somewhat different things, in MariaDB both terms can be used interchangeably.

Then if you want to explore one of them, type

```
USE database_name_here;
```

and

```
SHOW TABLES;
```

where **database_name_here** is the name of the database you want to examine.

Similarly to the Linux command line, you can clear the screen from the MariaDB prompt with

```
!\ clear
```

To create our first database, we will use the following command within the MariaDB prompt. Before doing so, let's agree on a naming convention. For databases we will append **_db** to whatever name we have chosen for our database, whereas with will use **_tbl** for tables. Although this is not strictly required, it will help us avoid mistakes when we use these objects later.

Let's begin by creating a database named Library_db:

```
CREATE DATABASE Library_db;
```

We can now create two tables inside **Library_db** named **Authors_tbl** and **Books_tbl**. To do so, we need to indicate that we want to use our newly-created database first:

```
USE Library_db;
```

Before creating the tables, we need to decide which type of information (or more properly speaking, data types) we want to store in them. This will suffice for now:

- **Authors_tbl**: **AuthorID** (integer), **AuthorName** (string)
- **Books_tbl**: **BookID** (integer), **BookName** (string), **AuthorID** (integer), **PublishedDate** (date)

where AuthorID and BookID will uniquely identify each author and book in the related tables. This is called the primary key of each table. In addition, AuthorID in Books_tbl will reference an author in Authors_tbl and is thus called a foreign key. This is a constraint that will ensure that when we insert a record in Books_tbl, the corresponding author exists in Authors_tbl.

Without further ado, let's now create the tables:

```
CREATE TABLE Authors_tbl(
   AuthorID INT NOT NULL AUTO_INCREMENT,
   AuthorName VARCHAR(50) NOT NULL,
   PRIMARY KEY (AuthorID)
);

CREATE TABLE Books_tbl(
   BookID INT NOT NULL AUTO_INCREMENT,
   BookName VARCHAR(100) NOT NULL,
   AuthorID INT NOT NULL,
   PublishedDate DATE NOT NULL,
   PRIMARY KEY (BookID),
   FOREIGN KEY(AuthorID) REFERENCES Authors_tbl(AuthorID)
);
```

In the above statements:

- NOT NULL means this field does not accept NULL values. In other words, this is a required field.
- AUTO_INCREMENT means we don't have to specify a value for the primary key. It will be inserted automatically for us starting at 1 and incrementing automatically by 1.

The output should be identical to the following image:

```
MariaDB [(none)]> CREATE DATABASE Library_db;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> USE Library_db;
Database changed
MariaDB [Library_db]> CREATE TABLE Authors_tbl(
    ->    AuthorID INT NOT NULL AUTO_INCREMENT,
    ->    AuthorName VARCHAR(50) NOT NULL,
    ->    PRIMARY KEY (AuthorID)
    -> );
Query OK, 0 rows affected (0.06 sec)

MariaDB [Library_db]> CREATE TABLE Books_tbl(
    ->    BookID INT NOT NULL AUTO_INCREMENT,
    ->    BookName VARCHAR(100) NOT NULL,
    ->    AuthorID INT NOT NULL,
    ->    PublishedDate DATE NOT NULL,
    ->    PRIMARY KEY (BookID),
    ->    FOREIGN KEY(AuthorID) REFERENCES Authors_tbl(AuthorID));
Query OK, 0 rows affected (0.06 sec)

MariaDB [Library_db]>
```

Up to this point we have been using the MariaDB root user. It is a good idea to create a regular username we can use to insert data into the tables we just created, and to query such information later.

To create a MariaDB user account named **gabriel** with password **mysecretpassword** (don't worry, it will NOT be stored in plain text, but MariaDB will hash it and store it in the **user** table inside the **mysql** database) and grant all permissions on all objects of Library_db, do:

```
CREATE USER 'gabriel'@'localhost' IDENTIFIED BY 'mysecretpassword';
GRANT ALL ON Library_db.* TO 'gabriel'@'localhost';
```

You can now exit the MariaDB prompt by typing exit and try to connect using your new credentials (enter the password when you're prompted to do so):

```
mysql -u gabriel -p
```

Now we can see that the current account does not have access to all databases (compare to the previous image):

```
[gacanepa@server ~]$ mysql -u gabriel -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.1.14-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| Library_db         |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

In MariaDB and MySQL, information_schema is a database that contains metadata, information about the server itself, access privileges, etc.

Let's insert some data in both tables:

```
USE Library_db;
INSERT INTO Authors_tbl (AuthorName) VALUES ('J. K. Rowling');
INSERT INTO Authors_tbl (AuthorName) VALUES ('J. R. R. Tolkien');
INSERT INTO Authors_tbl (AuthorName) VALUES ('C. S. Lewis');
```

If we query the Authors_tbl now, the result should be as follows:

```
SELECT AuthorID, AuthorName FROM Authors_tbl;
```

```
MariaDB [Library_db]> SELECT AuthorID, AuthorName FROM Authors_tbl;
+----------+------------------+
| AuthorID | AuthorName       |
+----------+------------------+
|        1 | J. K. Rowling    |
|        2 | J. R. R. Tolkien |
|        3 | C. S. Lewis      |
+----------+------------------+
3 rows in set (0.00 sec)
```

Please note that the value of AuthorID for J. R. R. Tolkien is 2. We need this value to insert records in Books_tbl corresponding to books authored by him:

```
INSERT INTO Books_tbl (BookName,AuthorID,PublishedDate) VALUES ('The Fellowship
of the Ring', 2, '1954-07-29');
INSERT INTO Books_tbl (BookName,AuthorID,PublishedDate) VALUES ('The Two Towers',
2, '1954-11-11');
INSERT INTO Books_tbl (BookName,AuthorID,PublishedDate) VALUES (' The Return of
the King', 2, '1955-10-20');
```

We can combine the records from both tables with a JOIN to view the information that we need. For example, let's say we want to show AuthorName, BookName, and PublishedDate of The Two Towers:

```
SELECT A.AuthorName AS AUTHOR, B.BookName AS BOOK, B.PublishedDate AS DATE
FROM Authors_tbl A JOIN Books_tbl B ON A.AuthorID=B.AuthorID
WHERE B.BookName='The Two Towers';
```

Note how the **AS** keyword allow us to create an alias to replace the actual name of a given field with a more friendly description:

```
MariaDB [Library_db]> SELECT A.AuthorName AS AUTHOR, B.BookName AS BOOK, B.PublishedDate AS DATE
    -> FROM Authors_tbl A JOIN Books_tbl B ON A.AuthorID=B.AuthorID
    -> WHERE B.BookName='The Two Towers';
+------------------+---------------+------------+
| AUTHOR           | BOOK          | DATE       |
+------------------+---------------+------------+
| J. R. R. Tolkien | The Two Towers | 1954-11-11 |
+------------------+---------------+------------+
1 row in set (0.00 sec)

MariaDB [Library_db]>
```

Backing up and restoring a database

Database records are kept in files which also need to be backed up on a periodic basis. It is important to learn how to properly back up a database and how to restore the data. This, of course, also includes the database structure - not only the data found therein!

To back up Library_db into a file named **Library_db-$(date +%F).sql**, do:

```
mysqldump -u gabriel -p Library_db > Library_db-$(date +%F).sql
```

You can now delete the tables from the MariaDB prompt (do this as root):

```
DROP DATABASE Library_db;
```

Then recreate it:

```
CREATE DATABASE Library_db;
```

At this point, Library_db will be empty. Exit the MariaDB prompt and do:

```
mysql -u gabriel -p Library_db < Library_db-$(date +%F).sql
```

The database should have now been restored and you can use it as usual.

Summary

In this article we have explained how to configure a MariaDB database server after we have installed and secured it. The configuration variables listed in the table above are only a few settings that you may want to consider while preparing the server for use or when tuning it later. Always refer to the official MariaDB documentation before making changes.

Thank you for choosing Tecmint.com and buying this ebook.