LINUX BASH PROGRAMMING COOKBOOK

Hot Recipes for BASH Development

SYSTEM CODE GEEKS
SYSADMINS RESOURCE CENTER

Linux BASH Programming Cookbook

Contents

Preface

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. First released in 1989, it has been distributed widely as it is a default shell on the major Linux distributions and OS X.

Bash is a command processor that typically runs in a text window, where the user types commands that cause actions. Bash can also read commands from a file, called a script. Like all Unix shells, it supports filename globbing (wildcard matching), piping, here documents, command substitution, variables and control structures for condition-testing and iteration. The keywords, syntax and other basic features of the language were all copied from sh. Other features, e.g., history, were copied from csh and ksh. Bash is a POSIX shell, but with a number of extensions. (https://bit.ly/2bBmYap)

In this ebook, we provide a compilation of BASH programming examples that will help you kick-start your own projects. We cover a wide range of topics, from user management and permissions setting, to specific commands like sed, tar, etc. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

SCGs (System Code Geeks) is an independent online community focused on creating the ultimate Operating System developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

SCGs serve the OS developer, OS engineer and DevOps communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at https://www.systemcodegeeks.com/

Chapter 1

Linux Find Command Tutorial

Linux `find` command is a powerful and flexible search tool that not only every system administrator must master, but also any kind of Linux user. This tutorial will show how to use it to perform almost any type search with this useful command.

For this tutorial, Linux Mint 17.3 and `find` 4.4.2 have been used.

1.1 Find by file name

The syntax for find command is the following

`find [path] [expression]`

1.1.1 By literal name

To find, for example, `README.txt` files in `home` directory, we would type the following:

```
find /home -name README.txt
```

The `-name` expression is case sensitive. The following command will find every `README.txt` file in the whole disk, not taking into account the case sensitivity:

```
find / -iname README.txt
```

1.1.2 By regular expression

Using regular expressions is useful mostly when we are looking for certain type of files.

For example, to find every `.txt` file in the home directory, the regular expression would be:

```
find /home -regextype posix-extended -regex ".*\.txt"
```

Let's see it carefully:

- With `-regextype` we are specifying a regular expression type, `posix-extended`, which is more widely implemented in other systems than `find` 's default one.
- And the regular expression itself:
  - The `.` is a wild card for searching any character.

- The `*` is for searching zero or more repetitions. Combined with `.`, will look for every file in the specified directory, because every file will have at least a character, repeated zero or more times.
- Once that we have looked for every file, we have to filter the results. In this case, we are finding for every string ending in `.txt`. Note that we have escaped the dot character, with `.`, since it's a special character for regex.

Regular expressions can be as flexible as we want. We can find every .txt and .log files with the following command:

```
find /home -regextype posix-extended -regex ".*(\.txt|\.log)"
```

What we are doing is to tell find to look for every file ending with the .txt substring **or** with the .log substring. The | character is an or operand. And everything inside round braces, is for matching that exact substring.

We have seen how to match exact substrings. Now, let's see how to match characters inside a range. For that, we will find files that have numbers in their name:

```
find /home -regextype posix-extended -regex ".*[0-9]"
```

The square brackets are for matching specific characters (not strings). And we can define ranges with the hyphen.

To end with this section, let's see how we would find files that are composed by a date (supposing it in UK format):

```
find /home -regextype posix-extended -regex ".*[0-9]{2}-[0-9]{2}-[0-9]{4}.*"
```

The curly braces are used for repetitions. So, we are looking for something with `*XX-XX-XXXX*` format.

Note: for finding files by their extension, there's a shorter way using `-name` :

```
find /home -name "*.txt"
```

Which is a more appropriate option when we want to make a search specifying an unique file extension.

1.2 Find by file properties

Apart from the name, we can also find for some specific file properties. In this section we will see for which properties we can make searches, and how.

1.2.1 Through time

1.2.1.1 In minutes

There are many cases in which we can be interested in finding by the time they have been modified (as same as when we sort by modification time when using a GUI).

We can look for files that have been modified, for example, in the **last 30 minutes**:

```
find /home -mmin -30
```

Note that we are using a hyphen when specifying the minutes. Without it, `find` would look for files that have been modificated **exactly 30 minutes ago**:

```
find /home -mmin 30
```

For finding files by **access time**, is almost the same than for modification time, but with `-amin` expression instead of `-mmin` :

```
find /home -amin -30
```

There's another property for finding through the time, that is the **change time**. Don't confuse it with modification time. The change time is a status change (for example, permission or owner change).

As you probably have guessed, for finding by status change time, we have to use the `-cmin` expression:

```
find /home -cmin -30
```

1.2.2 In days

We have seen how find files through time, specifying the time in minutes. But we can also make searches specifying the time in days. For that, we have to change the `*min` suffix of the previous expressions, with `*time` suffix.

Take into account that the searches made with `*time` are always truncated. So, if we want to find modified files between the current moment and the previous day, we have to type:

```
find /home -mtime 0
```

1.2.3 By permissions

Finding files by permissions is so simple. We can look for files belonging to a certain user or group:

```
find /home -user julen
find /home -group development
```

And also by the permissions of the files:

```
find / -perm 777
```

The above command would find the files with exactly `777` permissions, which can be useful to find files with permissions that should be fixed.

There is the dash prefix available, `/`, for finding files that match at least one of the specified permission bits. It's easier to understand seen in an example:

```
find / -perm /755
```

The above command would find files that are readable, writable or executable by the owner; readable or executable by the owner group; or readable or executable by other users. Finds

Of course, we can also use the symbolic notation to find files by permissions:

```
find / -perm a=rwx
```

Which would be the equivalent to `777`.

1.2.4 By file type

You may noticed that `find` also returns folders, for example. By default, find looks for every file that matches with the criteria, regardless its type.

If we are only looking for a certain type of files, we have to specify it with `-type`:

```
find / -perm a=rwx -type f
```

That command would only find the regular files.

`find` accepts the following values for this option:

- **d**: directory
- **f**: regular file
- **l**: symbolic link
- **b**: buffered block
- **c**: unbuffered character
- **p**: named pipe
- **s**: socket

1.2.5 By size

Finding by size is specially useful when we can to free some space in the disk. And is pretty simple:

```
find /var/log -size +20M
```

The above command will find files bigger than 20 megabytes in `/var/log` directory, a typical directory whose size has to be continuously watched by systems administrators.

We have several units to choose:

- **b**: 512-byte blocks (the one used by default, if no other unit is specified)
- **c**: bytes
- **k**: kilobytes
- **M**: megabytes
- **G**: gigabytes

1.3 Using conditions

`find` allows to use `and`, `not` and `or` boolean conditions.

Let's see it with an example for each one:

```
find / -size +10M -and -name "*.txt"
find / -perm 777 -not -user root
find / -user developer -or -group development
```

Which, respectively, mean: "find every file bigger than 10MB **and** with `.txt` extension", "find every file with `777` permissions that **does not** belong to `root` user", and "find every file that belongs to `developer` user **or** to `development` group".

Actually, the `-and` expression is not necessary to combine expressions with `and` logic.

```
find / -size +10M -name "*.txt"
```

The command shown above would work as same as the previous one with `-and`.

1.4 Executing actions with exec

This is the most powerful feature of `find`. The `-exec` option allows us to execute commands for every found file.

A very typical use of `-exec` is for showing file information as we would do with `ls` command:

```
find / -size +500M -exec ls -l {} \;
```

Apart from executing the command after `-exec` option, we have to add `{}`, where the file returned by `find` will be placed, like passing a parameter to the specified command. And the **;** is to set the command end. We can chain multiple `-exec` commands after the command end.

1.5 Useful and recurrent commands

To end with the tutorial, we will see some of the most common `find` commands.

Delete old and big log files

```
find / -iname "*.log" -mtime +1 -size +10M -exec rm {} \;
```

Fix 777 permissions

```
find / -perm 777 -exec chmod 755 {} \;
```

Delete empty folders

```
find / -type d -empty -exec rm --dir {} \;
```

Find 10 biggest files

```
find / -type f -exec ls -s {} \; | sort -n -r | head -10
```

Delete broken symbolic links

```
find / -xtype l -exec rm {} \;
```

1.6 Conclusion

In this tutorial we have seen how to deal with Linux `find` command, a very powerful tool for file search, from simple searches, like by name or extension; to more advanced ones, using complex regular expressions or filtering by permissions, and also being able to execute any commands for every search we make.

Chapter 2

Linux Screen Command Tutorial

Screen is a window manager that allows to have several virtual terminals, several sessions and programs in text mode executing simultaneously in the same console. In this tutorial we will see how to use it.

For this tutorial, Linux Mint 18 has been used.

2.1 Installation

If you don't have installed Screen, you can install it easily with `apt-get`:

```
sudo apt-get update
sudo apt-get install screen
```

2.2 Creating a screen

To create a screen, we just have to execute the `screen` command:

```
screen
```

We will see an informative screen about the software itself. Just hit return to exit.

At this moment, a new screen will be generated inside the terminal window. You may have noticed it, since the buffer has been cleared.

And that's it!

This has created a pipe file for this session in `/var/run/screen/S-<username>/` directory. The username I'm working with is named `julen`, so, the directory is `/var/run/screen/S-julen/`.

In my case, the name for the session is `3360.pts-0.julen-VAIO`. `screen` generates the sessions with the following format:

```
<pid>.<tty>.<host>
```

You may be wondering how to exit the screen. You can obviously close the terminal, but that would leave the session alive. If you want to kill it, you can execute the following command inside the session you want to kill:

```
exit
```

You will see that you return to the "original" shell, and that `screen` has told that, effectively, you have exited its screen:

```
[screen is terminating]
```

If you check the directory where `screen` saves the sessions, you will notice that the previous session has disappeared.

2.2.1 Creating windows inside screens

We have seen that executing `screen` in the host computer, creates a new session, which is saved in `/var/run/screen/S-<username>/` directory. If we would execute `screen` n times in the host, we would have n sessions.

Inside each session, we can have other several screens, named windows. The windows are created, again, executing the `screen` command. So, the following:

```
screen
# We are know inside a screen session
screen
```

Would only create a session in `/var/run/screen/S-<username>/`:

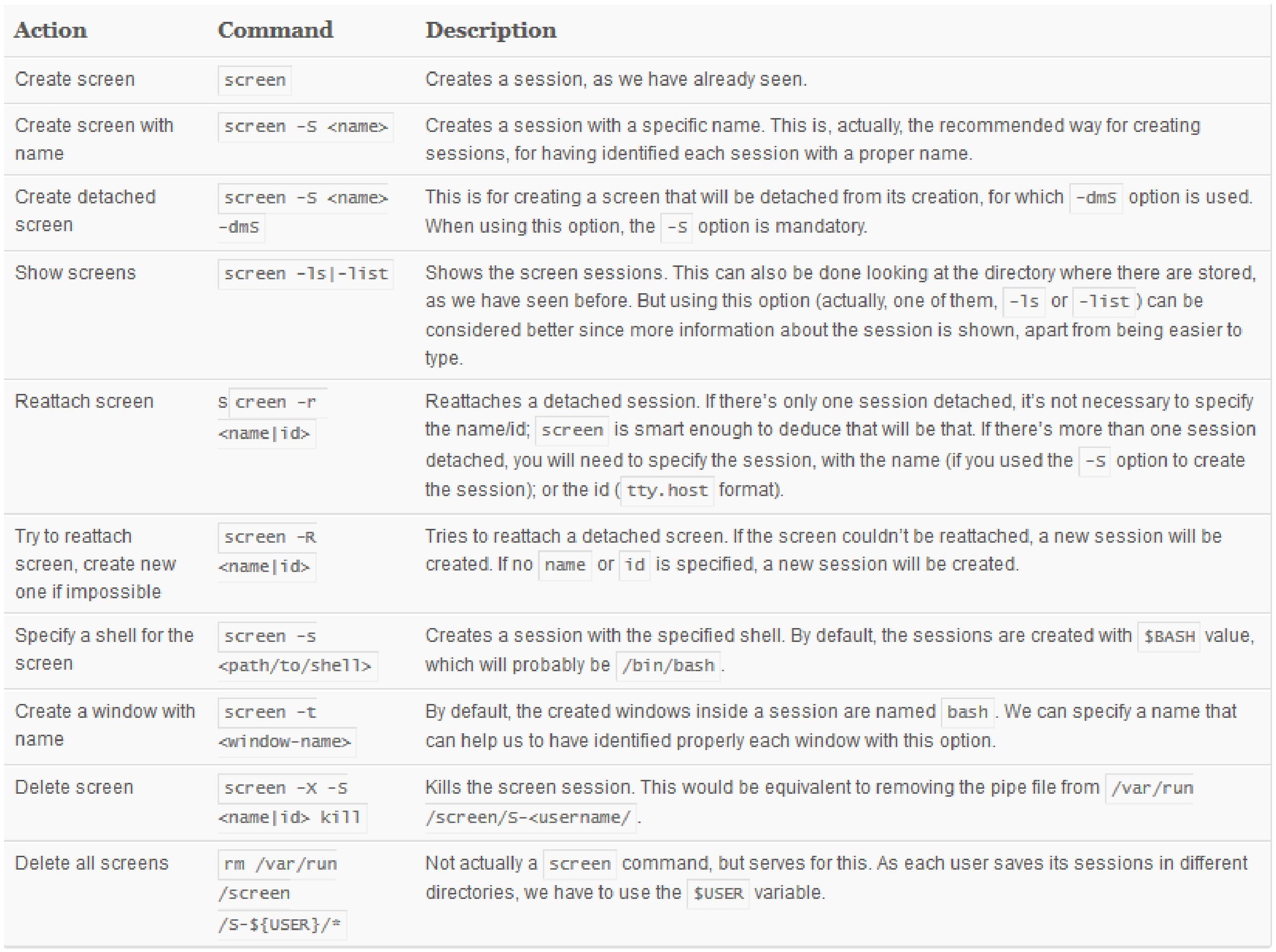
```
11714.pts-0.julen-VAIO
```

This is because, as we have said, executing `screen` inside a screen session will create a window inside that screen, not another session.

In the next sections we will see the commands available for these screens.

2.3 Screen commands options

`screen` command has many inline options. The following table shows and explains them.



Action	Command	Description
Create screen	`screen`	Creates a session, as we have already seen.
Create screen with name	`screen -S <name>`	Creates a session with a specific name. This is, actually, the recommended way for creating sessions, for having identified each session with a proper name.
Create detached screen	`screen -S <name> -dmS`	This is for creating a screen that will be detached from its creation, for which `-dmS` option is used. When using this option, the `-S` option is mandatory.
Show screens	`screen -ls|-list`	Shows the screen sessions. This can also be done looking at the directory where there are stored, as we have seen before. But using this option (actually, one of them, `-ls` or `-list`) can be considered better since more information about the session is shown, apart from being easier to type.
Reattach screen	s `creen -r <name|id>`	Reattaches a detached session. If there's only one session detached, it's not necessary to specify the name/id; `screen` is smart enough to deduce that will be that. If there's more than one session detached, you will need to specify the session, with the name (if you used the `-S` option to create the session); or the id (`tty.host` format).
Try to reattach screen, create new one if impossible	`screen -R <name|id>`	Tries to reattach a detached screen. If the screen couldn't be reattached, a new session will be created. If no `name` or `id` is specified, a new session will be created.
Specify a shell for the screen	`screen -s <path/to/shell>`	Creates a session with the specified shell. By default, the sessions are created with `$BASH` value, which will probably be `/bin/bash` .
Create a window with name	`screen -t <window-name>`	By default, the created windows inside a session are named `bash` . We can specify a name that can help us to have identified properly each window with this option.
Delete screen	`screen -X -S <name|id> kill`	Kills the screen session. This would be equivalent to removing the pipe file from `/var/run /screen/S-<username>/` .
Delete all screens	`rm /var/run /screen /S-${USER}/*`	Not actually a `screen` command, but serves for this. As each user saves its sessions in different directories, we have to use the `$USER` variable.

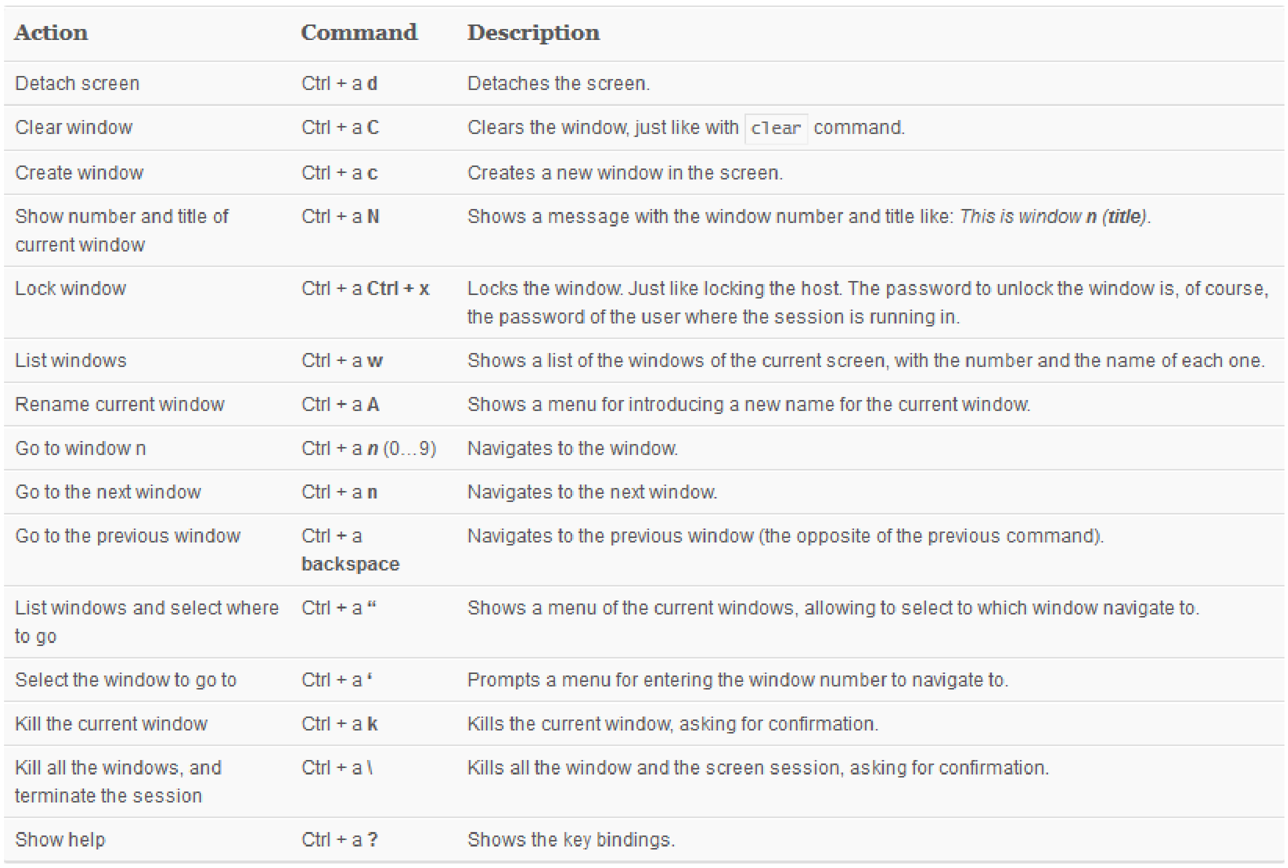
Figure 2.1: Most popular key bindings

2.4 Key bindings

When we are inside a screen and its windows, we have available some special key combinations to perform some actions, e.g., navigating through different windows.

These key bindings consist on pressing Ctrl + a, and then pressing the key(s) in question. Not at the same time: first, Ctrl + a; then, release them; and finally, press the key in question.

The following table shows the most used key bindings.



Action	Command	Description
Detach screen	Ctrl + a **d**	Detaches the screen.
Clear window	Ctrl + a **C**	Clears the window, just like with `clear` command.
Create window	Ctrl + a **c**	Creates a new window in the screen.
Show number and title of current window	Ctrl + a **N**	Shows a message with the window number and title like: *This is window **n** (**title**)*.
Lock window	Ctrl + a **Ctrl + x**	Locks the window. Just like locking the host. The password to unlock the window is, of course, the password of the user where the session is running in.
List windows	Ctrl + a **w**	Shows a list of the windows of the current screen, with the number and the name of each one.
Rename current window	Ctrl + a **A**	Shows a menu for introducing a new name for the current window.
Go to window n	Ctrl + a ***n*** (0…9)	Navigates to the window.
Go to the next window	Ctrl + a **n**	Navigates to the next window.
Go to the previous window	Ctrl + a **backspace**	Navigates to the previous window (the opposite of the previous command).
List windows and select where to go	Ctrl + a **“**	Shows a menu of the current windows, allowing to select to which window navigate to.
Select the window to go to	Ctrl + a **‘**	Prompts a menu for entering the window number to navigate to.
Kill the current window	Ctrl + a **k**	Kills the current window, asking for confirmation.
Kill all the windows, and terminate the session	Ctrl + a ****	Kills all the window and the screen session, asking for confirmation.
Show help	Ctrl + a **?**	Shows the key bindings.

Figure 2.2: Most popular key bindings

2.5 Customizing Screen

`screen` allows us to configure it as we want. It looks for the configuration in a file named `.screenrc` in the user's home directory; or system wide, in `/etc/screenrc`.

2.5.1 Disabling startup message

You may already be tired of the startup message when you create a screen, that one about the version, copyright, etc. This can be disabled with the following line in the configuration file:

```
startup_message off
```

2.5.2 Setting default windows for each session

One of the most interesting possibilities is defining a set of default windows for every `screen` session. For that, we just have to write the commands we would execute, in the configuration file.

For example, let's say that for every `screen` session we want, apart from a shell, a Python and PHP console. We could write the following in the configuration file:

```
screen -t Shell  /bin/bash
screen -t Python /usr/bin/python3ve
screen -t PHP    /usr/bin/php -a # Interactive mode.

select 0 # After creating the windows, go to the first one.
```

Now, for the configuration shown above, you would have, for each `screen` session, those three windows.

2.5.3 Key bindings

The remaining customization that is worth mentioning is the configuration of key bindings, for having a better experience.

A usual practice is to use the function keys (F1 - F12). For example, we could use them for accessing windows:

```
bindkey "^[[[A"  select 1  # F1 -> window 1
bindkey "^[OQ"   select 2  # F2 -> window 2
bindkey "^[OR"   select 3  # F3 -> window 3
bindkey "^[OS"   select 4  # F4 -> window 4
bindkey "^[[15~" select 5  # F5 -> window 5
bindkey "^[[17~" select 6  # F6 -> window 6
bindkey "^[[18~" select 7  # F7 -> window 7
bindkey "^[[19~" select 8  # F8 -> window 8
bindkey "^[[20~" select 9  # F9 -> window 9
bindkey "^[[21~" select 10 # F10 -> window 10
bindkey "^[[23~" prev      # F11 -> previous window
bindkey "^[[24~" next      # F12 -> next window
```

Note: if you want to know which key code corresponds to each keystroke, you can use `showkey`:

```
showkey -a
```

And then press a key to know its code.

2.6 Summary

In this tutorial we have seen how to use the Screen utility, for managing several virtual windows in the same terminal. We have seen the command's inline options, and also the special key combinations for when we are inside the virtual windows. To end up with the tutorial, we have shown how we can customize Screen with the configuration file.

Chapter 3

Linux chown Example

3.1 Introduction

In this example, we will see how to use the Unix-like system command `chown`

The `chown` command changes the owner and owning group of files.

3.1.1 Linux Machine

If you are on Windows OS, before we get into the details of this command, I would suggest you to have a Linux machine for better understanding of the usage of this command.

Not necessarily you need to physically have a separate machine. You can very much have a virtual Linux machine on a virtual box like Oracle VirtualBox.

3.2 Syntax

```
chown [_OPTION_]...[_OWNER_][:[_GROUP_]] _FILE_... chown [_OPTION_]...--reference=_RF
ILE_ _FILE_...
```

3.3 Description

chown changes the user and/or group ownership of each given file.

If only an owner is given, that user is made the owner of each given file, and the files' group is not changed.

If the owner is followed by a colon and a group name, with no spaces between them, the group ownership of the files is changed as well.

If a colon but no group name follows the user name, that user is made the owner of the files and the group of the files is changed to that user's login group.

If the colon and group are given, but the owner is omitted, only the group of the files is changed; in this case, **chown** performs the same function as **chgrp**. If only a colon is given, or if the entire operand is empty, neither the owner nor the group is changed.

3.4 Options

Here is the list of options available to use with `chown` command

-c, --changes	like verbose but report only when a change is made.
-f, --silent, --quiet	suppress most error messages.
-v, --verbose	output a diagnostic for every file processed.
--dereference	affect the referenced file of each symbolic link rather than the symbolic link itself. This is the default.
-h, --no-dereference	affect symbolic links instead of any referenced file. This is useful only on systems that can change the ownership of a symlink.
--from=CURRENT_OWNER:CURRENT_GROUP	change the owner and/or group of each file only if its current owner and/or group match those specified here. Either may be omitted, in which case a match is not required for the omitted attribute.
--no-preserve-root	do not treat */* (the root directory) in any special way. This is the default.
--preserve-root	Do not operate recursively on */*.
--reference=RFILE	use RFILE's owner and group rather than specifying OWNER:GROUP values
-R, --recursive	operate on files and directories recursively.

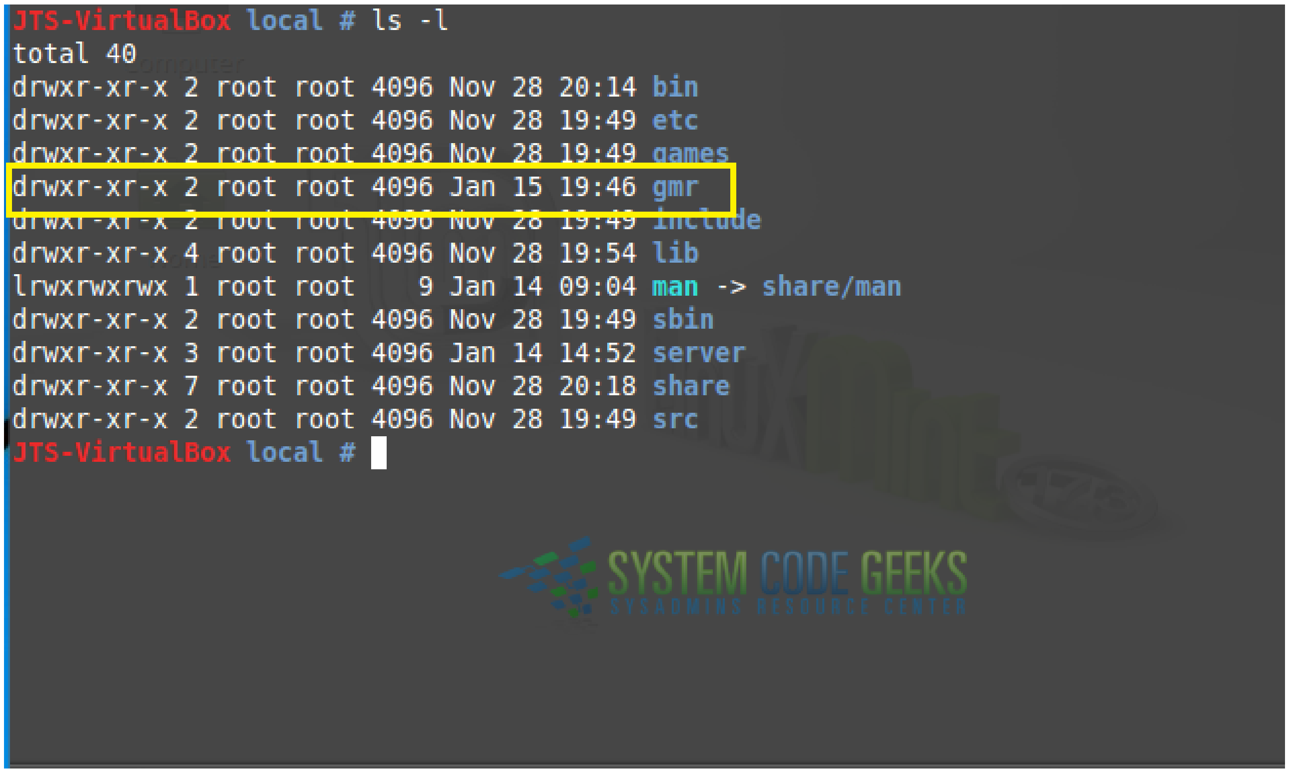
The following options modify how a hierarchy is traversed when the **-R** option is also specified. If more than one is specified, only the final one takes effect.

-H	if a command line argument is a symbolic link to a directory, traverse it.
-L	traverse every symbolic link to a directory encountered
-P	do not traverse any symbolic links. This is the default.
--help	display this help and exit.
--version	output version information and exit.

Owner is unchanged if unspecified. Group is unchanged if unspecified, or changed to the login group if implied by a *:* following a symbolic **OWNER**. **OWNER** and **GROUP** may be numeric as well as symbolic.

3.5 Examples

Please refer to the image shown below.



```
JTS-VirtualBox local # ls -l
total 40
drwxr-xr-x 2 root root 4096 Nov 28 20:14 bin
drwxr-xr-x 2 root root 4096 Nov 28 19:49 etc
drwxr-xr-x 2 root root 4096 Nov 28 19:49 games
drwxr-xr-x 2 root root 4096 Jan 15 19:46 gmr
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local #
```

Figure 3.1: List Command Details

We will see the details of the information given above with yellow rectangle mark.

- **1st Character - File Type:** First character specifies the type of the file.

In the picture above the *d* in the 1st character indicates that this is a directory.

Following are the possible file type options in the 1st character of the ls -l output.

- **Field Explanation**
- - normal file
- d directory
- s socket file
- l link file

Field 1 - File Permissions:

The remaining 9 characters, in order, refer to the read/write/execute(rwx) permission for the user owner, the read/write/execute(rwx) permission for the group owner, and then the read/write/execute(rwx) permission for everyone else.

In this example, `rwxr-xr-x` indicates read-write-execute permission for user, read-execute permission for group, and read-execute permission for others.

Field 2 - Number of links: Second field specifies the number of links for that file. In this example, 2 indicates 2 links to this file.

Field 3 - Owner: Third field specifies owner of the file. In this example, this file is owned by username *root*.

Field 4 - Group: Fourth field specifies the group of the file. In this example, this file belongs to "root" group.

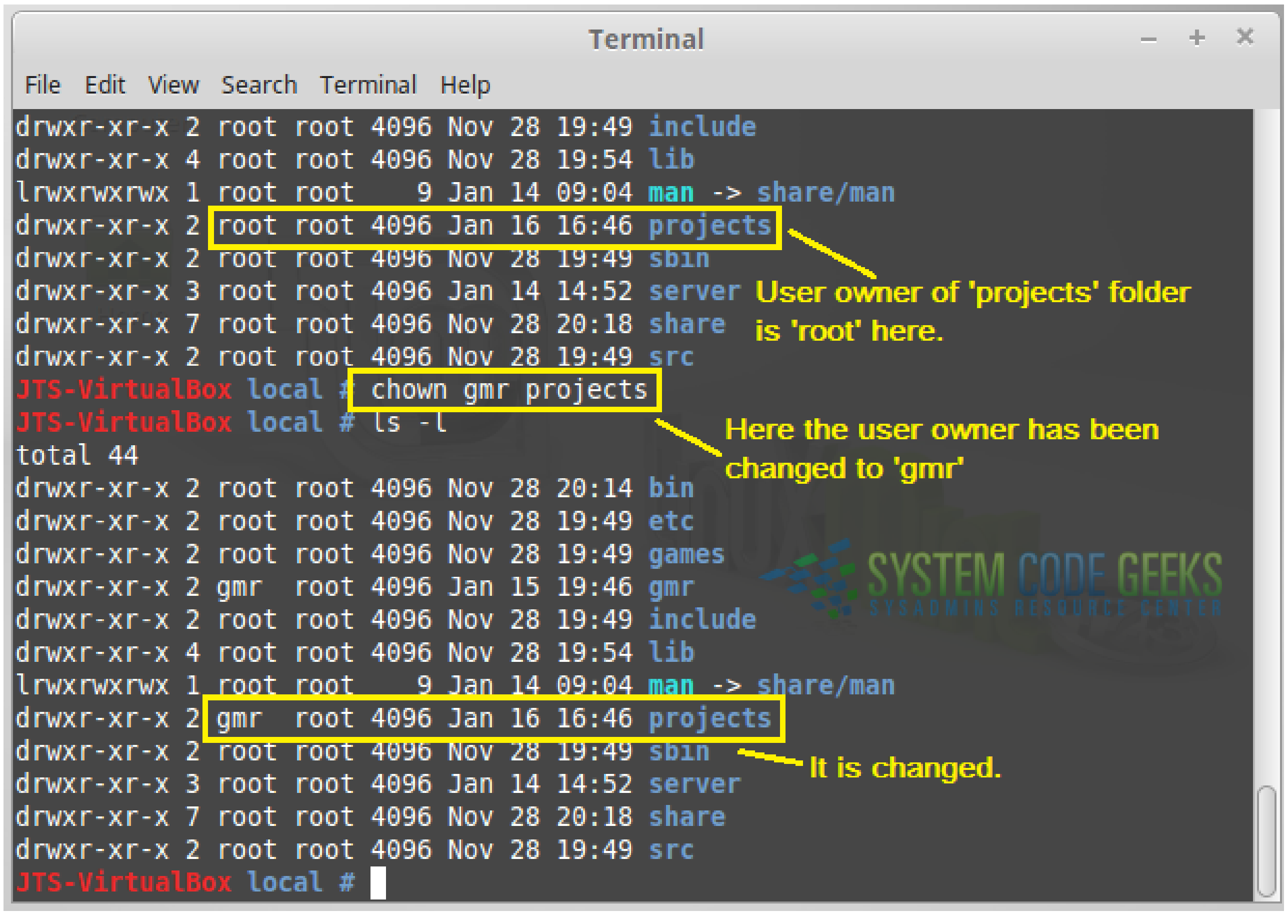
Field 5 - Size: Fifth field specifies the size of file. In this example, ’4096’ indicates the file size.

Field 6 - Last modified date & time: Sixth field specifies the date and time of the last modification of the file. In this example, *Jan 15 19:46* specifies the last modification time of the file.

Field 7 - File name: The last field is the name of the file/directory. In this example, the file name is *gmr*.

3.5.1 Change User Owner

Please refer the image shown below. In this the user owner of a folder has been changed to a new user.


Terminal
File Edit View Search Terminal Help
User owner of 'projects' folder is 'root' here.
Here the user owner has been changed to 'gmr'
It is changed.
SYSTEM CODE GEEKS
SYSADMINS RESOURCE CENTER

```
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 root root 4096 Jan 16 16:46 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local # chown gmr projects
JTS-VirtualBox local # ls -l
total 44
drwxr-xr-x 2 root root 4096 Nov 28 20:14 bin
drwxr-xr-x 2 root root 4096 Nov 28 19:49 etc
drwxr-xr-x 2 root root 4096 Nov 28 19:49 games
drwxr-xr-x 2 gmr  root 4096 Jan 15 19:46 gmr
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 gmr  root 4096 Jan 16 16:46 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local #
```

Figure 3.2: Change Owner

But, user owner has been changed only for the directory *projects*. Ownership of the inner files or directories will remain the same, unchanged.

If the flag *-R* is used, it will change the owner for inner files / directories also.

Recursively grant ownership of the directory **projects**, and all files and sub-directories, to user **gmr**.

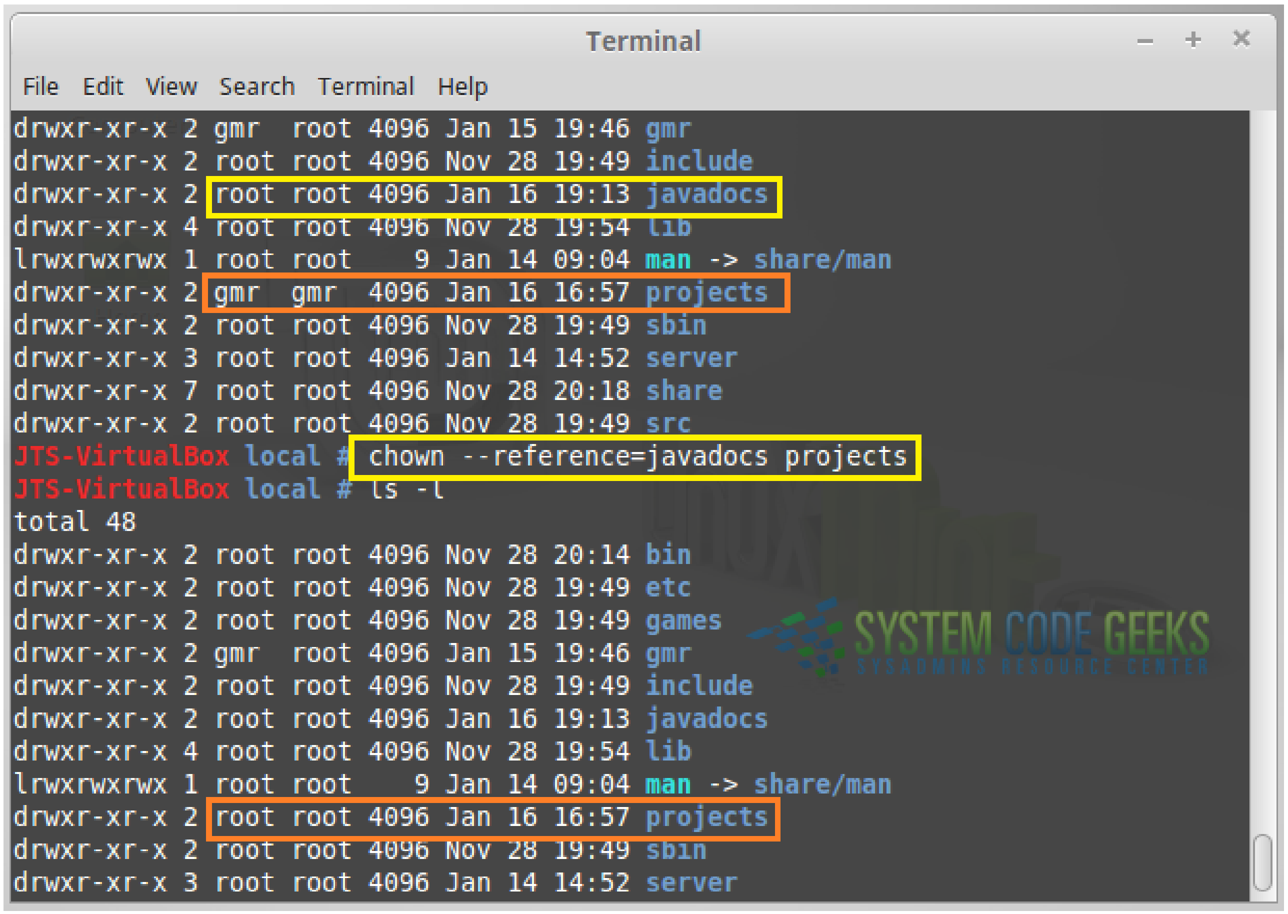
```
chown -R gmr projects
```

3.5.2 Copy ownership from one file to another

This can be done using the ‘-reference’ flag.

```
# chown --reference=file tmpfile
```

As shown below, ownership of *javadocs* folder has been copied to *projects* folder.

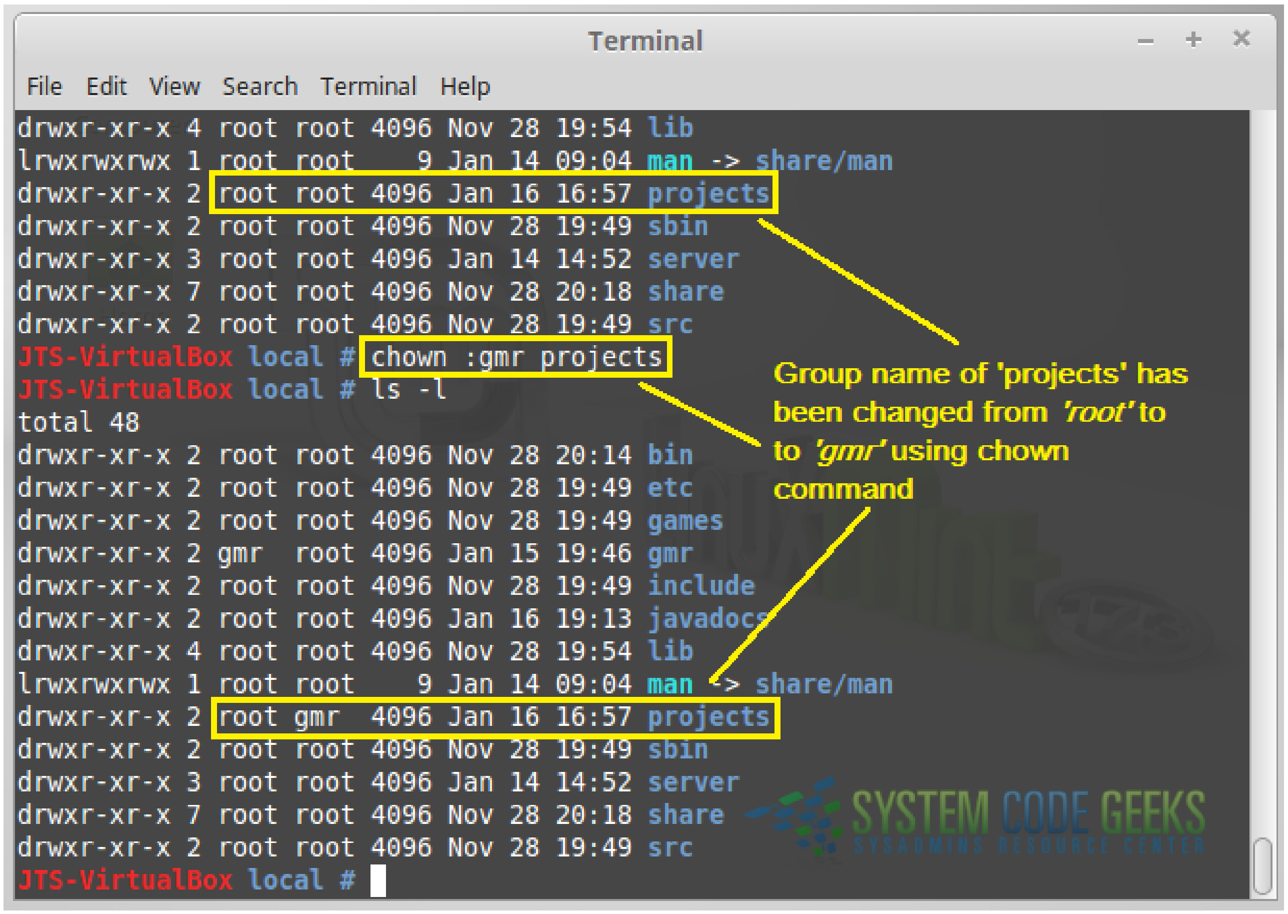


```
drwxr-xr-x 2 gmr  root 4096 Jan 15 19:46 gmr
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 2 root root 4096 Jan 16 19:13 javadocs
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 gmr  gmr  4096 Jan 16 16:57 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local # chown --reference=javadocs projects
JTS-VirtualBox local # ls -l
total 48
drwxr-xr-x 2 root root 4096 Nov 28 20:14 bin
drwxr-xr-x 2 root root 4096 Nov 28 19:49 etc
drwxr-xr-x 2 root root 4096 Nov 28 19:49 games
drwxr-xr-x 2 gmr  root 4096 Jan 15 19:46 gmr
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 2 root root 4096 Jan 16 19:13 javadocs
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 root root 4096 Jan 16 16:57 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
```

Figure 3.3: Copy ownership

3.5.3 Change the group of a file

Using `chown` command, the group (that a file belongs to) can also be changed.


Terminal
File Edit View Search Terminal Help
Group name of 'projects' has been changed from 'root' to to 'gmr' using chown command

```
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 root root 4096 Jan 16 16:57 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local # chown :gmr projects
JTS-VirtualBox local # ls -l
total 48
drwxr-xr-x 2 root root 4096 Nov 28 20:14 bin
drwxr-xr-x 2 root root 4096 Nov 28 19:49 etc
drwxr-xr-x 2 root root 4096 Nov 28 19:49 games
drwxr-xr-x 2 gmr  root 4096 Jan 15 19:46 gmr
drwxr-xr-x 2 root root 4096 Nov 28 19:49 include
drwxr-xr-x 2 root root 4096 Jan 16 19:13 javadocs
drwxr-xr-x 4 root root 4096 Nov 28 19:54 lib
lrwxrwxrwx 1 root root    9 Jan 14 09:04 man -> share/man
drwxr-xr-x 2 root gmr  4096 Jan 16 16:57 projects
drwxr-xr-x 2 root root 4096 Nov 28 19:49 sbin
drwxr-xr-x 3 root root 4096 Jan 14 14:52 server
drwxr-xr-x 7 root root 4096 Nov 28 20:18 share
drwxr-xr-x 2 root root 4096 Nov 28 19:49 src
JTS-VirtualBox local #
```

Figure 3.4: Change group name

3.6 Conclusion

In this example, we have seen how to use Linux command `chown` with different options.

Chapter 4

Linux chmod Example

One of the most critical jobs a system administrator has to continuously be dealing with is the permission administration. The most small carelessness with the permissions can lead to a security hole in the system.

This example will show how are changed the permissions, a task for which `chmod` command is used

For this example, Linux Mint 17.3 has been used.

4.1 Linux permission system

What makes the Linux permission system so great is its simplicity, specially when it's compared to others, such us Windows'. In any case, we will see briefly how it works.

There are three main things that have to be understood: the **elements** the permissions are defined for, the **actions** that can be performed, and **who** can perform them.

The elements are two:

- Files.
- Directories.

The actions are three:

- Read
- Write.
- Execute. Apart from for executing scripts and binaries, also **corresponds to folders**: to create files and other folders inside it.

And who can perform them, other three:

- The user that owns the file.
- The group that the user owning the file belongs to.
- Any other user that is not the owner and does not belong to the group the owner does.

To see how the permissions are organized, we can list the files in the terminal:

```
ls -l
```

And we will see something similar to the following:

```
-rw-r--r-- 1 julen julen      0 Jun 26 14:20 file.txt
drwxr-xr-x 2 julen julen   4096 Jun 26 14:22 folder
```

The first section, 10 characters, are which correspond to permissions. Let's examine it:

- The first character is for file type. `-` means that the file is a regular file, and `d` means that is a directory.
- The following nine characters are for read (`r`), write (`w`) and execute (`x`) permissions for the owner, the group of the owner, and others, respectively.

4.2 Changing permissions

To change the permissions, the `chmod` command (contraction of change and mode) is used. The syntax is the following:

```
chmod permissions file [file 2] [file n]
```

The specification of the files seems quite obvious, but, how are the permissions specified?

4.2.1 Octal representation

One of the options is the octal notation.

For example, for setting read and write permissions for the owner, read permissions for its group, and no permission for others, to a `file.txt` file, we would have to execute:

```
sudo chmod 640 file.txt
```

We can try to read the file with a user that is not the owner and does not belong the the owner's group, to see what happens:

```
sudo -u other-user more file.txt
```

And we would receive an error message:

```
file.txt: Permission denied
```

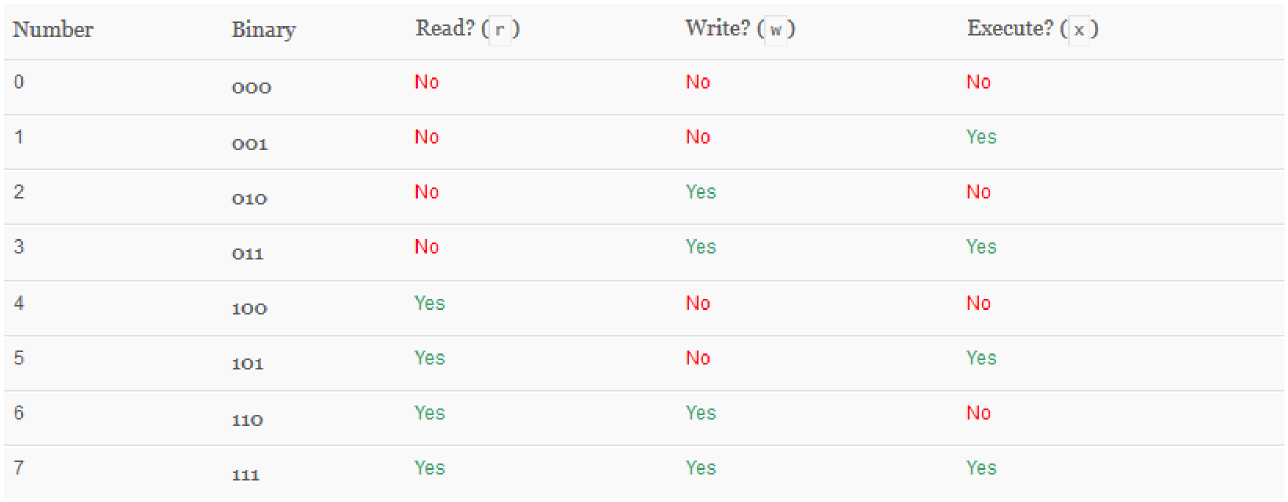
But where comes that `640` from?

Each digit of that number represents which permissions will have the owner, its group, and others. And each digit is the octal representation of the permission combination we want to assign. So:

- `0` for no permission.
- `1` is for execution permission.
- `2` is for write permission.
- `4` is for read permission.

So, for assigning read and write permissions, what we have done is `2 + 4 =6`.

The following table shows the permissions that gives each digit:



Number	Binary	Read? (r)	Write? (w)	Execute? (x)
0	000	No	No	No
1	001	No	No	Yes
2	010	No	Yes	No
3	011	No	Yes	Yes
4	100	Yes	No	No
5	101	Yes	No	Yes
6	110	Yes	Yes	No
7	111	Yes	Yes	Yes

Figure 4.1: Most popular key bindings

4.2.2 Symbolic representation

Apart from octal representation, `chmod` also allows symbolic representation.

In the symbolic representation letters and arithmetic operators are used:

For the **users**, the symbols are the following:

- `u`: file owner.
- `g`: file owner's group.
- `o`: other users.
- `a`: all users.

The **actions** are represented with the same symbols that we have seen before:

- `r`: read.
- `w`: write.
- `x`: execute.

And the operators to set the permissions are:

- `=`: sets the permissions as specified, overwriting any other previous permissions.
- `+`: adds permissions.
- `-`: removes permissions.

The syntax for the symbolic representation is the following:

```
sudo chmod who operator action(s)[,who operator action(s),...]
```

For example, the previous example (owner: read and write; group: read; others: none) with the symbolic notation, would be the following:

```
sudo chmod u=rw,g=r,o-rwx file.txt
```

As you can see, for this example, this notation requires much more verbosity.

But for other cases is more suitable. For example, if we want to maintain the current permissions, but allowing now everybody to write, we would have to type:

```
sudo chmod a+w file.txt
```

And the file would pass from these permissions:

```
-rw-r----- 1 julen julen 0 Jun 26 17:38 file.txt
```

To these ones:

```
-rw-rw--w- 1 julen julen 0 Jun 26 17:39 file.txt
```

4.3 Special permissions

Apart from those permissions we have seen, there are three more special permissions in Linux: the `setuid` (user id), the `setgid` (group id), and the `sticky`.

4.3.1 setuid

The `setuid` bit (set user ID) can be assigned to executable files, which is for, when a file is executed, allow the process acquire that file's owner's permissions. **This is generally used to allow normal users (those without superuser privileges) to obtain root privileges for some executables**.

This can be seen, for example, in common binaries, such us `/bin/ping`. If we check its permissions, we will see that it has the `setuid` bit bit assigned:

```
-rwsr-xr-x 1 root root 44168 May  7  2014 /bin/ping*
```

Which is expressed with a `s`, in the place for the execution bit for the owner.

If we remove the `setuid` bit from that executable:

```
sudo chmod u-s /bin/ping
```

A normal user won't be able to use `ping`, and if it would try to use it:

```
ping localhost
```

The following message will be shown:

```
ping: icmp open socket: Operation not permitted
```

Of course, an user with `sudo` permissions could execute the command with root privileges, but, as said before, **this bit is though for users that don't have superuser privileges**.

To add the `setuid` bit again:

```
sudo chmod u+s /bin/ping
```

4.3.2 setgid

If the `setuid` bit allows the user acquire the permissions of the owner, **the `setgid` (set group id) allows to acquire the permissions of the group**.

The bit can be set as follows with the octal notation:

```
sudo chmod 2777 script.sh
```

Which is set with the `2`.

And with symbolic notation:

```
sudo chmod g+s script.sh
```

If the `setuid` bit was placed in the owner's execution bit place, in this case, is placed in the group's execution's place:

```
-rwxrwsrwx  1 julen julen     14 Jun 27 19:22 script.sh*
```

4.3.3 sticky

The sticky bit is used in directories, when we want **a file, or folder writable for several users, but where files and folders inside it can only be deleted by the owner**. This bit is used, for example, in `/tmp` and `/var/tmp` directories.

The sticky bit can be assigned the following way:

```
sudo chmod 1777 sticky/
```

Where the sticky bit is set with the leading `1`, and then, the wanted permissions (which usually are `777` for this cases).

And, with symbolic notation:

```
sudo chmod +t sticky/
```

The sticky bit is expressed with a `t` in the place for the execution bit for others:

```
drwxrwxrwt  2 julen julen   4096 Jun 27 19:22 sticky/
```

We can see how it works, we can create a file in our `sticky/` directory, try to modify it with the owner and other user, and also to delete it:

```
touch file.txt
echo "written with owner" > file.txt
sudo -u other-user echo "written with other user" > file.txt
```

Which would work. But if we try to delete it with the user that is not the owner. . .

```
sudo -u other-user rm file.txt
```

And we would see:

```
rm: remove write-protected regular file 'file.txt'?
```

We can type `y` to confirm, but. . .

```
rm: cannot remove 'file.txt': Operation not permitted
```

But the owner of the file could delete it with no problem.

To remove the sticky bit:

```
sudo chmod -t sticky/
```

4.4 Summary

In this example, we have first seen very briefly how the Linux permission, identifying the main concepts that are involved when dealing with permissions. Then, we have seen how the permissions are changed for the different notations, and, to end up with the example, we have taken a look for the special permissions bits that are available in Linux.

Chapter 5

Linux Create User Example

This example will show you how to create users in Linux, looking at the two different available commands: the default utility, `useradd`; and a script that acts as more friendly front-end for the default utility, which is named `adduser`.

For this example, Linux Mint 17.03 has been used.

5.1 How users are organized

The existing users of the system are registered in the file `/etc/passwd`. **This file defines who has legit access to the system**. This is an example of a line of the file:

```
julen:x:1000:1000:Julen Pardo:/home/julen:/bin/bash
```

Which follows the following format:

```
username:password:uid:gid:real_name:home_directory:command_shell
```

- The `username` is the account name for the login.
- The `password` field is actually not used in modern systems. The users credentials are stored in `/etc/shadow` file.
- The `uid` (user id) and `gid` (group id) are the unique identifiers of the user and the group it belongs to, respectively.
- The `real_name` is that, the user's real name.
- The `home_directory` is the working directory of each user, usually `/home/<username>`.
- Finally, the `command_shell` is the program that is ran at login. Usually, this is the path to a shell. If not set, `/bin/sh` is used.

It's better not to touch manually this file to add (or modify/remove) **users**. To add users, we should use the methods that we will see in this tutorial.

5.2 Using native binary: useradd

`useradd` is the native, low level, binary of Linux systems. Its use is very simple:

```
sudo useradd [options] username # superuser privileges are needed.
```

So, we could create a user named `john_doe`:

```
sudo useradd john_doe
```

Now, a new user named `john_doe` has been created in the users database. We can check it in the `/etc/passwd` file:

```
grep "john_doe" /etc/passwd
```

Which will show:

```
john_doe:x:1002:1005::/home/john_doe:
```

5.2.1 Setting a password

We have created a user without a password! We can check it in the `/etc/shadow` file:

```
grep "john_doe" /etc/shadow
```

Returning:

```
john_doe:!:17018:0:99999:7:::
```

That exclamation mark `!` means that no password is set for the user.

Setting a password for each user is not an advice, but mandatory. For this, we have two options: create the user and then set the password (with `passwd` command), or specify it at creation time with `-p` (`--password`) option. The recommended option is the first one, since the second one has two obvious downsides:

- The password is visible in the command line.
- We are not asked for confirmation, so we won't notice if we make a miss typing the password.

Use always the `passwd` command to set the passwords. We only have to run it specifying the user, as in the following example:

```
sudo passwd john_doe
```

And we will be asked to set the password (with confirmation).

5.2.2 Creating the home directory

Now that we have this new user, we can try to login in the system with it:

```
sudo -u john_doe -i # Login with user john_doe.
```

But we will get an error:

```
sudo:unable to change directory to /home/john_doe:No such file or directory
```

This is because `useradd` **sets the home directory for new users, but it does not create it by default**. We can fix it by creating manually the directory, but is better to create the home directory at user creation time. This is achieved passing the `-m` (`--create-home`) option to `useradd`:

```
sudo userdel john_doe # To delete it.
sudo useradd john_doe -m
```

This will create a directory for the new user. The default behavior for this option is to create the directory with the same name as the created user, in the `/home` directory.

5.2.3 Setting a different home directory

For some reason, we might want to set the home directory in a different place from `/home`. This is allowed using the `-b` (`--base-dir`) option. For example:

```
sudo useradd john_doe -b /tmp
```

Will create the following entry in `/etc/passwd`:

```
john_doe:x:1002:1005::/tmp/john_doe:
```

Note that we only have specified **the directory where the home directory will be placed, not the home directory name itself**.

When we use this option, we also have to tell `userrad` to create the home directory, as in the example of the previous section:

```
sudo useradd john_doe -b /tmp -m
```

5.2.4 Setting the shell

You may have noticed that the in the line for our user, the value for the shell is not set. Usually, we would want to use `/bin/bash` instead of the default `/bin/sh`. To specify the shell, we have to use the `-s` (`--shell`) option:

```
sudo useradd john_doe -m -s /bin/bash
```

5.2.5 Other options

Let's see other common options for `useradd` command.

5.2.5.1 Specifying the primary group

The default behavior when creating a user is to create a group for it, with the same name, and set it as primary. But we have the option to avoid this and specify a group name (or `gid`) to be the primary of the creating user. For this, `-g` (`--gid`) option is used, as in the following example:

```
sudo useradd john_doe -g developers
```

And `john_doe` will be created with `developers` as primary group. We can check it with the `groups` command:

```
groups john_doe
```

5.2.5.2 Setting secondary groups

Similarly to the primary group, we may want to set secondary group(s) for a user at creation time. This time, `-G` (`-groups`) option has to be used, specifying the list of groups separated by commas, without whitespaces, e.g.:

```
sudo useradd john_doe -G developers,another_secondary
```

5.2.5.3 Setting an expiration date

This option is useful when we have to create accounts for users that we know beforehand have to have access to the system only until a certain date. For this, we have to use the `-e` (`--expiredate`) option, specifying the date in `YYYY-MM-DD` format. Let's see it with an example:

```
sudo useradd john_doe -e 2017-01-01
```

5.2.6 Setting personal information

Actually, we can set any type of additional comments, but this option is usually used to specify personal information, such as real name. We have to use the `-c` (`--comment`) option, specifying the information between quotes (single or double, doesn't matter) if the comment contains whitespaces, e.g.:

```
sudo useradd john_doe -c 'John Doe'
```

Will generate the following entry:

```
john_doe:x:1002:1005:John Doe:/home/john_doe:
```

5.3 Using a user-friendly wrapper for useradd: adduser

With `useradd`, we have seen that creating users is not actually difficult, but, by default, it doesn't perform some actions that can be supposed as essential, like creating the home directory. We can even create a user without a password, and do not notice it.

To make user creation easier and in a more comfortable way, `adduser` was created. This is just a Perl script for an interactive use of `useradd`.

If we try to create a user with `adduser`, e.g.:

```
sudo adduser john_doe
```

And we will see that, only typing that, `adduser` does many things for us:

```
Adding user 'john_doe' ...
Adding new group 'john_doe' (1001) ...
Adding new user 'john_doe' (1002) with group 'john_doe' ...
Creating home directory '/home/john_doe' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for john_doe
Enter the new value, or press ENTER for the default
Full Name []: John Doe
Room Number []: 1
Work Phone []: 111-111-111
Home Phone []: 222-222-222
Other []: 333-333-333
Is the information correct? [Y/n] Y
```

(In italic the values specified by hand).

That is, apart from creating the home directory and setting the password with `passwd`, also allows to set personal information about the user. And also sets `/bin/bash` for the shell. This is the line that has been added in `/etc/passwd` for the user we have just created:

```
john_doe:x:1002:1001:John Doe,1,111-111-111,222-222-222,333-333-333:/home/john_doe:/bin/ ↩
    bash
```

5.3.1 Changing the options

Even if `adduser` does makes more comfortable the user creation, we can change the options. Let's see the equivalents for `adduser` that we have seen for `useradd`.

The format is the same as with `useradd`:

```
sudo adduser <username> [option1] <value1>...[optionN] <valueN>
```

- Changing the home directory: `--home`
- Changing the shell: `--shell`
- Specifying the primary group: `--ingroup`

The `adduser` utility does not provide options for setting secondary groups and and an expiration date.

5.4 Summary

This example has shown how to create users in Linux systems, with two different commands: `useradd` and `adduser`. As we have seen, `adduser` can be considered a better (in terms of usability) option, since it performs two essential actions that `useradd` does not perform by default: create a home directory, and set a password. Even if a user creation can require more options, those two are always fundamental.

Chapter 6

Linux Add User to Group Example

In this tutorial we will see how to add users to groups in Linux, looking at the different possibilities (existing or non-existing users), and taking into account also the different types of groups users can belong to (primary and secondary groups).

For this example, Linux Mint 17.03 has been used.

6.1 How groups are organized

Linux defines the groups in the file `/etc/group`. If we open it, we will see many rows, with the following format:

```
adm:x:4:syslog,julen
```

Which follows this format:

```
group_name:password:gid:user1,user2,...,userN
```

- The `group_name` is the name we give to the group when we create it with `groupadd`.
- The `password` is optional. Really, this is almost never used.
- The `gid` (group identifier) is the numerical identifier that each group has.
- Finally, the members of the group are listed.

It is possible to modify this file manually, but also dangerous, since it can become corrupt. In any case, there is available a tool for checking the integrity of this file, called `grpck`. To use it, just execute it with `sudo` permissions. If the file is correct, no message will be shown. **In any case, is not recommendable to modify it manually**.

It is necessary to know that Linux distinguishes two types of groups for a user: the primary, and secondaries. The primary group is the one used when the user creates files and directories. Let's suppose that we have a user named `john_doe`, whose primary group is `developers`, but that is enrolled also in a group called `testers`. Every file created by him:

```
touch foo
```

Will be created with with `developers` as the owner group:

```
-rw-r--r-- 1 john_doe developers 0 jul 27 12:03 foo
```

To see to which groups a user belongs to, we can use the `groups` command, specifying the user name:

```
groups <username>
```

Which will return an output with the following format:

```
<username> : <primary_group>[<secondary_group1>,...,<secondary_groupN>]
```

That is, the primary group will be the first of the list (or the unique, if the user does not belong to more groups).

6.2 Non-existing users

When we are going to create a new user, with `useradd`, we can specify its group(s), so we can create users and assign groups to it with one command.

6.2.1 Primary group

The primary group is configured with `-g` (`--gid`) option. For example, to create a `john_doe` user with the `developers` group as primary, we would have to type:

```
sudo useradd john_doe -g developers
```

(Remember to always assign a password to each new user, with `passwd` command.)

We can check that it has been created as expected, using `groups` command:

```
groups john_doe
```

Which would return:

```
john_doe : developers
```

6.2.1.1 Changing default configuration of primary group assignment

If no primary group is specified, the assignation of the primary group will depend on the configuration defined in `/etc/login.defs`. If the variable `USERGROUP_ENAB` is set to `yes`, the primary group of the user will be a new group with the same name as the username. If the variable is set to `no`, the primary group of the user will be the one specified in `/etc/default/useradd`, in the `GROUP` variable.

So, if we assume that every user created in the future has to have a specific group as primary, e.g., `developers`, we first have to edit the `/etc/login.defs` file:

```
USERGROUP_ENAB no
```

The second and last step is to specify the group in `/etc/default/useradd` file:

```
GROUP=developers
```

6.2.2 Secondary groups

The option for assigning secondary groups to the user that is going to be created is `-G` (`--groups`), specifying the list of groups separated by commas, without whitespaces. For example:

```
sudo useradd john_doe -G developers,testers
```

Would create the `john_doe` user, with `developers` and `testers` groups as secondary.

```
john_doe : john_doe developers testers
```

6.3 Existing users

The `usermod` command, as its name suggests, is for modifying users, in all its facets, including their groups.

For the modification of groups, it works exactly as with `useradd`: `-g` for modifying the primary group, and `-G` for the secondary ones.

6.3.1 Primary group

Changing the primary group of an existing user is pretty simple, we just have to use the -g option for the useradd, as told before:

```
sudo usermod john_doe -g developers # Now primary group of 'john_doe' is 'developers'.
```

The manual of useradd warns about changing the primary group of a user:

```
Any file from the user's home directory owned by the previous primary group of the user ↩
   will be owned by this new group.

The group ownership of files outside of the user's home directory must be fixed manually.
```

This manual fixing for the whole disk can be easily done with find. Let's suppose that we have changed john_doe user's primary group from john_doe to developers, and that we want to change the owner of every file to this one. We could execute the following:

```
sudo find / -group john_doe -exec chgrp developers {} \;
```

Finding every file in / and subdirectories (i.e., all the disk) that has john_doe as group owner, executing for each result a chgrp to change to group owner to developers.

6.3.2 Secondary groups

Let's suppose that we have a john_doe user with the following output for groups:

```
john_doe : john_doe
```

Now, we want to assign some groups, developers and testers, to the existing john_doe user. We would just have to execute the usermod command with the -G option, specifying the groups:

```
sudo usermod john_doe -G developers,testers
```

If we now check the groups with groups, we will see:

```
john_doe : john_doe developers testers
```

According to the manual of usermod, this is what happens when using -G option:

```
[...]  If the user is currently a member of a group which is not listed, the user will
be removed from the group.This behavior can be changed via the -a option, which append
s the user to the current supplementary group list.
```

In the previous case, we can see that the group that john_doe was already belonging to (the primary), has not disappeared, but this is **just because it was the primary group**. Note that, with the current groups for john_doe, the following:

```
sudo usermod john_doe -G another_group
```

john_doe would be removed from developers and testers groups:

```
john_doe : john_doe another_group
```

To append groups, use the -a (--append) option, as the manual suggests.

6.4 Giving users sudo permissions

Users are given sudo permissions by just being added to sudo group. So, the only thing we have to do is to add the users to sudo group, as same as we have been seeing in this example:

```
sudo useradd john_doe -G sudo    # At creation time.
sudo usermod john_doe -G sudo -a # For existing user.
```

6.5 Summary

This example has shown how to add users to groups, for both non-existing, and already existing users, taking into account also the differences between the primary group and the secondary ones, considering also the side effects of a primary group change, and proposing an easy fix to them.

Chapter 7

Linux tar Examples

I am a real fan of learning Linux commands by examples and from my personal experience it will really help to see some Linux tar command examples. But first, a brief bit of background information. The name "tar" stands for "tape archive".

As that name implies, originally it was a command that Unix administrators used to deal with tape drives. Where we now use the Linux tar command to create a tar file, we used to tell it to write the tar archive to a device file (e.g in /dev).

Nowadays the Linux tar command is more often used to create compressed archives that can easily be moved around, from one disk to anther disk or machine to machine, in other words we can call it as backup utility and its been used in most of the Unix flavors. One user may archive a large collection of files, and another user may extract those files, with both of them using the tar command.

7.1 Create a tar archive of a directory

In this example we will come to know basic tar command using option *cvf* to create a tar archive. Here we are creating a tar file *my-archive.tar* for a directory */home/my-directory* in current working directory:

```
[root@chasiota /]# tar cvf my-archive.tar /home/my-directory
```

c - Creates a new .tar archive file v - Verbosely list files which are processed f - File name type of the archive file

7.2 Create a zipped archive

In order to make tar ball as zipped archive , we need to use the option "z". In below example the command will make "my-archive.tar.gz" for a directory "/home/my-directory" in current working directory.

```
[root@chasiota /]# tar cvzf my-archive.tar.gz /home/my-directory
```

7.3 List the contents of a tar archive

To list the contents of an uncompressed tar archive, just replace the cflag with the tflag, like this:

```
[root@chasiota /]# tar tvf my-archive.tar
```

This lists all the files in the archive, but does not extract them. To list all the files in a compressed archive, add the zflag like before:

```
[root@chasiota /]# tar tzvf my-archive.tgz
```

or for a tar.gz archive:

```
[root@chasiota /]# tar tzvf my-archive.tar.gz
```

7.4 Extract tar archive contents

To extract the contents of a Linux tar archive, now just replace the tflag with the x("extract") flag. For uncompressed archives the extract command looks like this:

```
[root@chasiota /]# tar xvf my-archive.tar
```

For compressed archives the tar extract command looks like this:

```
[root@chasiota /]# tar xzvf my-archive.tar.gz
```

or this for *.tgz:

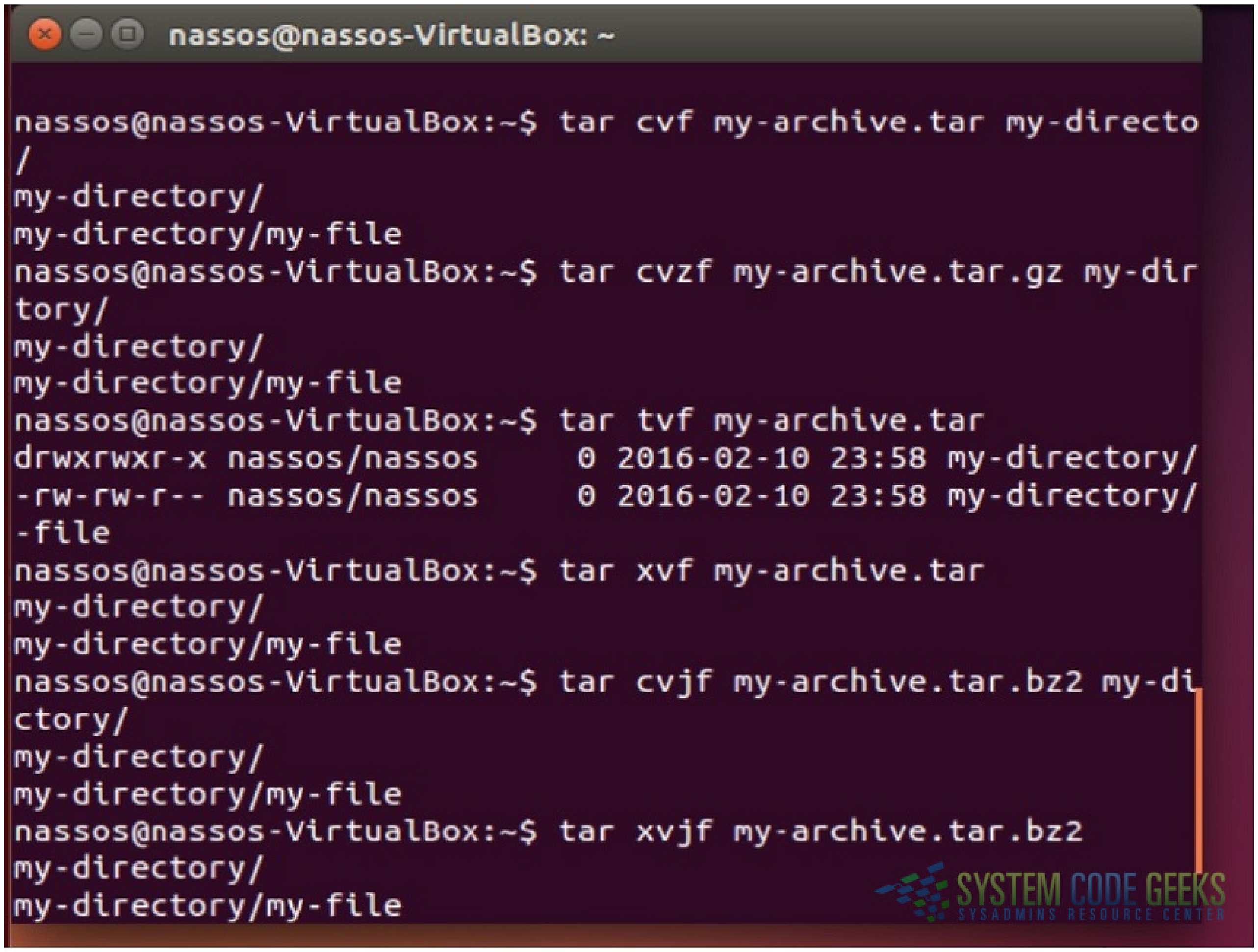
```
[root@chasiota /]# tar xzvf my-archive.tgz
```

7.5 Extract tar.bz2 archive contents

In order to extract the contents of a *.tar.bz2 file the options are "xj".

```
[root@chasiota /]# tar xvjf my-archive.tar.bz2
```

The below image illustrates the commands seen so far:



```
nassos@nassos-VirtualBox:~$ tar cvf my-archive.tar my-directo
/
my-directory/
my-directory/my-file
nassos@nassos-VirtualBox:~$ tar cvzf my-archive.tar.gz my-dir
tory/
my-directory/
my-directory/my-file
nassos@nassos-VirtualBox:~$ tar tvf my-archive.tar
drwxrwxr-x nassos/nassos     0 2016-02-10 23:58 my-directory/
-rw-rw-r-- nassos/nassos     0 2016-02-10 23:58 my-directory/
-file
nassos@nassos-VirtualBox:~$ tar xvf my-archive.tar
my-directory/
my-directory/my-file
nassos@nassos-VirtualBox:~$ tar cvjf my-archive.tar.bz2 my-di
ctory/
my-directory/
my-directory/my-file
nassos@nassos-VirtualBox:~$ tar xvjf my-archive.tar.bz2
my-directory/
my-directory/my-file
```

Figure 7.1: Tar Commands (1)

7.6 Extract a single file from tar archive

To extract a specific file from a tar archive, specify the file name at the end of the tar xvf command as shown below. The following command extracts only a specific file (my-file.sh) in the current directory from a large tar file.

```
[root@chasiota /]# tar xvf my-archive.tar my-file.sh
```

7.7 Create a compressed archive of the current directory

Many times when using the Linux tar command you will want to create an archive of all files in the current directory, including all subdirectories. You can easily create this archive like this:

```
[root@chasiota /]# tar czvf my-directory.tgz .
```

In the above example, the . at the end of the command is how you refer to the current directory.

7.8 Create an archive in a different directory

You may also want to create a new tar archive like that previous example in a different directory, like this:

```
[root@chasiota /]# tar czvf /tmp/my-directory.tgz .
```

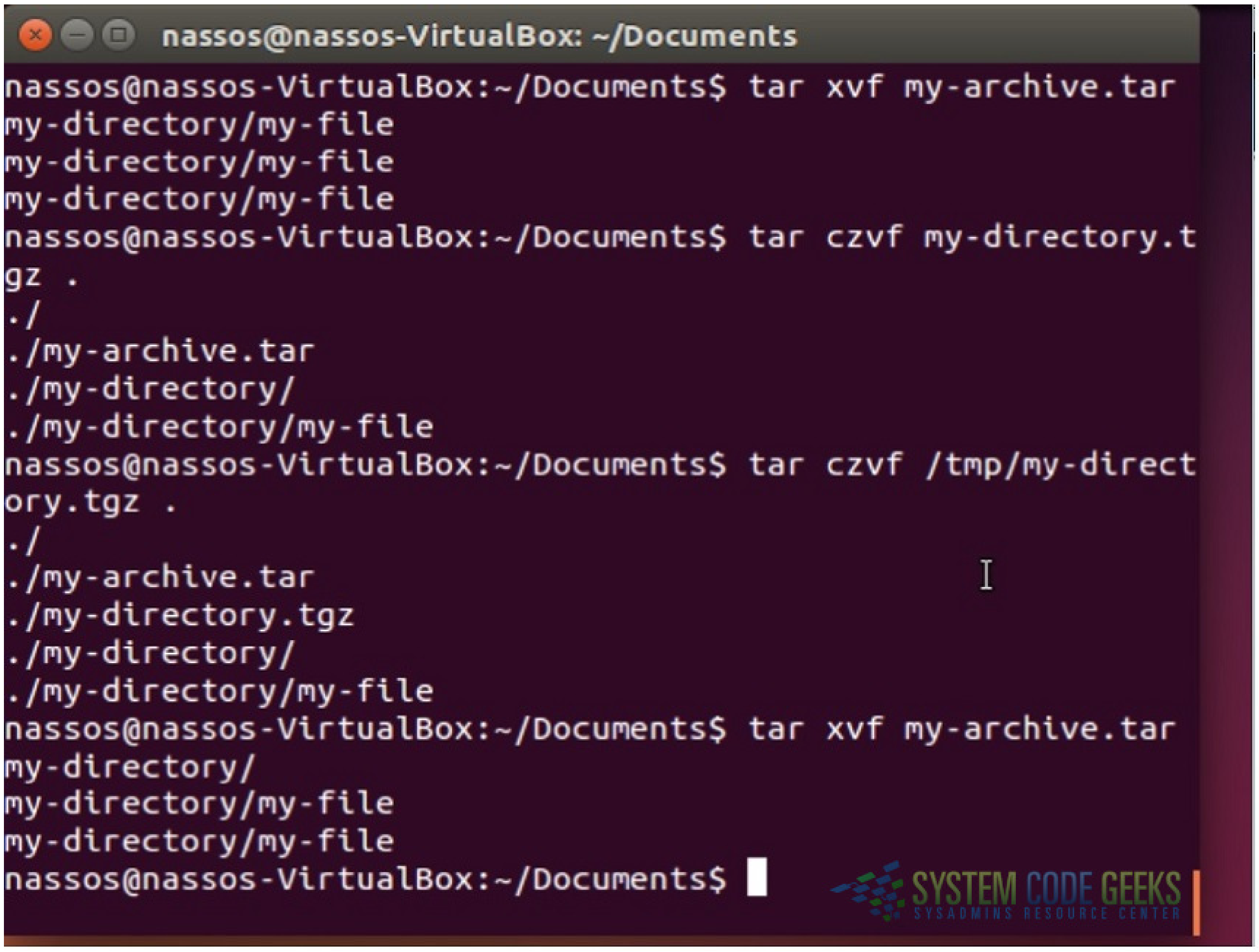
As you can see, you just add a path before the name of your tar archive to specify what directory the archive should be created in.

7.9 Extract a single directory from tar archive

To extract a single directory (along with its subdirectory and files) from a tar archive, specify the directory name at the end of the tar xvf command as shown below. The following extracts only a specific directory from a large tar file:

```
[root@chasiota /]# tar xvf my-archive.tar home/my-directory/
```

Check below image for examples 6-9:


nassos@nassos-VirtualBox: ~/Documents
SYSTEM CODE GEEKS
SYSADMINS RESOURCE CENTER

```
nassos@nassos-VirtualBox:~/Documents$ tar xvf my-archive.tar
my-directory/my-file
my-directory/my-file
my-directory/my-file
nassos@nassos-VirtualBox:~/Documents$ tar czvf my-directory.t
gz .
./
./my-archive.tar
./my-directory/
./my-directory/my-file
nassos@nassos-VirtualBox:~/Documents$ tar czvf /tmp/my-direct
ory.tgz .
./
./my-archive.tar
./my-directory.tgz
./my-directory/
./my-directory/my-file
nassos@nassos-VirtualBox:~/Documents$ tar xvf my-archive.tar
my-directory/
my-directory/my-file
my-directory/my-file
nassos@nassos-VirtualBox:~/Documents$
```

Figure 7.2: Tar Commands (2)

7.10 Extract a single directory from tar.gz archive

We just need to add “z” to the above extract command “xvf”

```
[root@chasiota /]# tar xvzf my-archive.tar.gz home/my-directory/
```

7.11 Check the size of the tar, tar.gz and tar.bz2 Archive File

For any tar, tar.gz and tar.bz2 archive file, the below commands will display the size of archive file in Kilobytes (KB):

```
[root@chasiota /]# tar -cf - my-archive.tar | wc -l
```

```
[root@chasiota /]# tar -czf - my-archive.tar.gz | wc -l
```

```
[root@chasiota /]# tar -cjf  - my-archive.tar.bz2 | wc -l
```

7.12 Verify integrity of tar file

As part of creating a tar file, you can verify the integrity of the archive file that got created using the option “W” as shown below:

```
[root@chasiota /]# tar tvfW my-archive.tar
```

If an output line starts with Verify, and there is no differs line then the file/directory is OK. If not, you should investigate the issue.

Note: for a compressed archive file (*.tar.gz, *.tar.bz2) you cannot do the verification.

7.13 Find the difference between an archive and file system

Finding the difference between an archive and file system can be done even for a compressed archive. It also shows the same output as above excluding the lines with Verify.

Finding the difference between gzip archive file and file system:

```
[root@chasiota /]# tar dfz my-archive.tgz
```

Finding the difference between bzip2 archive file and file system:

```
[root@chasiota /]# tar dfj my-archive.tar.bz2
```

7.14 Delete a file from tar ball

You can use the following syntax to delete a file from a tar ball:

```
[root@chasiota /]# tar --delete -f my-archive.tar home/my-file
```

7.15 Add a file to an existing archive

You can add additional files to an existing tar archive with “r” option:

```
[root@chasiota /]# tar rvf my-archive.tar my-file
```

7.16 Add a directory to an existing archive

Adding a directory is also the same. We need to mention the directory name instead of the file name:

```
[root@chasiota /]# tar rvf my-archive.tar my-dir/
```

7.17 Extract group of files from tar, tar.gz, tar.bz2 archives using regular expression

You can specify a regular expressions , to extract files matching a specified pattern. For example, the following tar command extracts all the files whose file ends with .java:

```
[root@chasiota /]# tar xvf my-archive.tar  --wildcards '*.java'
```

7.18 Untar multiple files from tar, tar.gz and tar.bz2 File

To extract or untar multiple files from the tar, tar.gz and tar.bz2 archive file. For example the below command will extract “my-file-1” “my-file-2” and "my-file-3" from the archive files:

```
[root@chasiota /]# tar -xvf my-archive.tar "my-file-1" "my-file-2" "my-file-3"
```

```
[root@chasiota /]# tar -zxvf my-archive.tar.gz "my-file-1" "my-file-2" "my-file-3"
```

```
[root@chasiota /]# tar -jxvf my-archive.tar.bz2 "my-file-1" "my-file-2" "my-file-3"
```

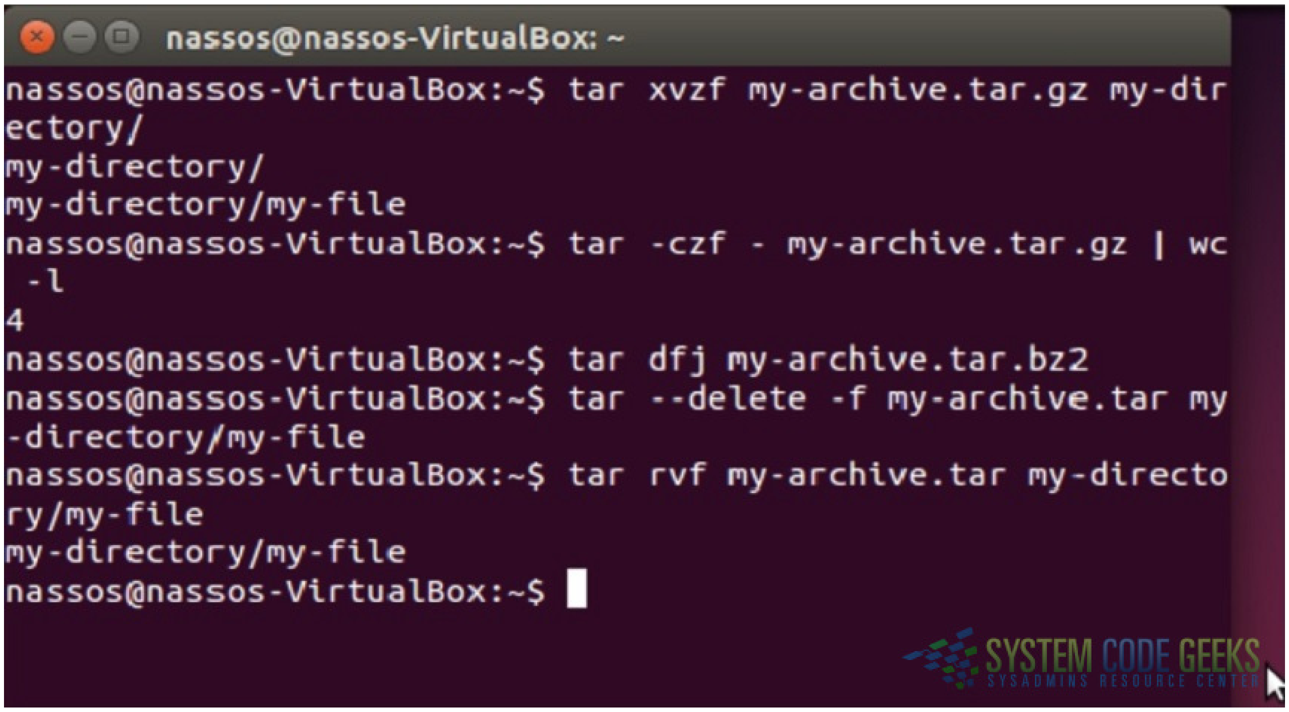
7.19 Restore files with tar

More important than performing regular backups is having them available when we need to recover important files. The following command will restore all files from the full-backup-Day-Month-Year.tar archive, which is an example backup of our home directory:

```
[root@chasiota /]# tar xpf /dev/st0/full-backup-Day-Month-Year.tar
```

The p option preserves permissions; file protection information will be remembered.

Some of the above examples are depicted in the below image:



```
nassos@nassos-VirtualBox:~$ tar xvzf my-archive.tar.gz my-dir
ectory/
my-directory/
my-directory/my-file
nassos@nassos-VirtualBox:~$ tar -czf - my-archive.tar.gz | wc
 -l
4
nassos@nassos-VirtualBox:~$ tar dfj my-archive.tar.bz2
nassos@nassos-VirtualBox:~$ tar --delete -f my-archive.tar my
-directory/my-file
nassos@nassos-VirtualBox:~$ tar rvf my-archive.tar my-directo
ry/my-file
my-directory/my-file
nassos@nassos-VirtualBox:~$
```

Figure 7.3: Tar Commands (3)

7.20 Check the manual page for tar

You can always refer to the manual page for all available tar commands:

```
[root@chasiota /]# man tar
```

Chapter 8

Linux sed Examples

Sed is basically a stream editor used for modifying files in unix or linux. It provides a nifty way to perform operations on files which can be passed around through pipes. Most people never learn its real power, they just simply use sed to replace text. You can do many things apart from replacing text with sed.

As mentioned earlier, sed is an editor that allows us to use pattern matching to search and replace within a file, or an input stream. This works by using Regular Expressions. By default, the results of any edits we make to a source file, or input stream, will be sent to STDOUT, the standard output. The original file will be unaffected by the edits.

Also the sed command can be incredibly useful when bootstrapping a new server, or machine, as it can easily be scripted. A common use case for sed is to script the editing of configuration files on a new server instance to facilitate the further setup of the needed environment for that machine.

In this article I will describe the capabilities of sed with examples. Consider the below file as input to our examples:

```
>cat example.txt
I want to learn java. Learn java. Learn java
java is the best
java forever
```

8.1 Replacing string

Sed command is mostly used to replace the text in a file. The below sed command replaces the word "java" with "guava" in the file only for the first occurrence in each line:

```
>sed 's/java/guava/' example.txt
```

Here the "s" specifies the substitution operation. The "/" are delimiters. The "java" is the search pattern and the "guava" is the replacement string.

By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace next occurences.

8.2 Replacing the nth occurrence of a pattern in a line

Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word "java" with "guava" in a line:

```
>sed 's/java/guava/2' example.txt
```

8.3 Replacing all the occurrences of a pattern in a line

The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line:

```
>sed 's/java/guava/g' example.txt
```

8.4 Replacing from nth occurrence to all occurrences in a line

Use the combination of /1, /2, /n and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces from the second occurrence until the nth of the word "java" with the word "guava" in a line:

```
>sed 's/java/guava/2g' example.txt
```

8.5 Duplicating the replaced line with /p flag

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

```
>sed 's/java/guava/p' example.txt
```

The below image illustrates the execution of the first 5 sed commands:

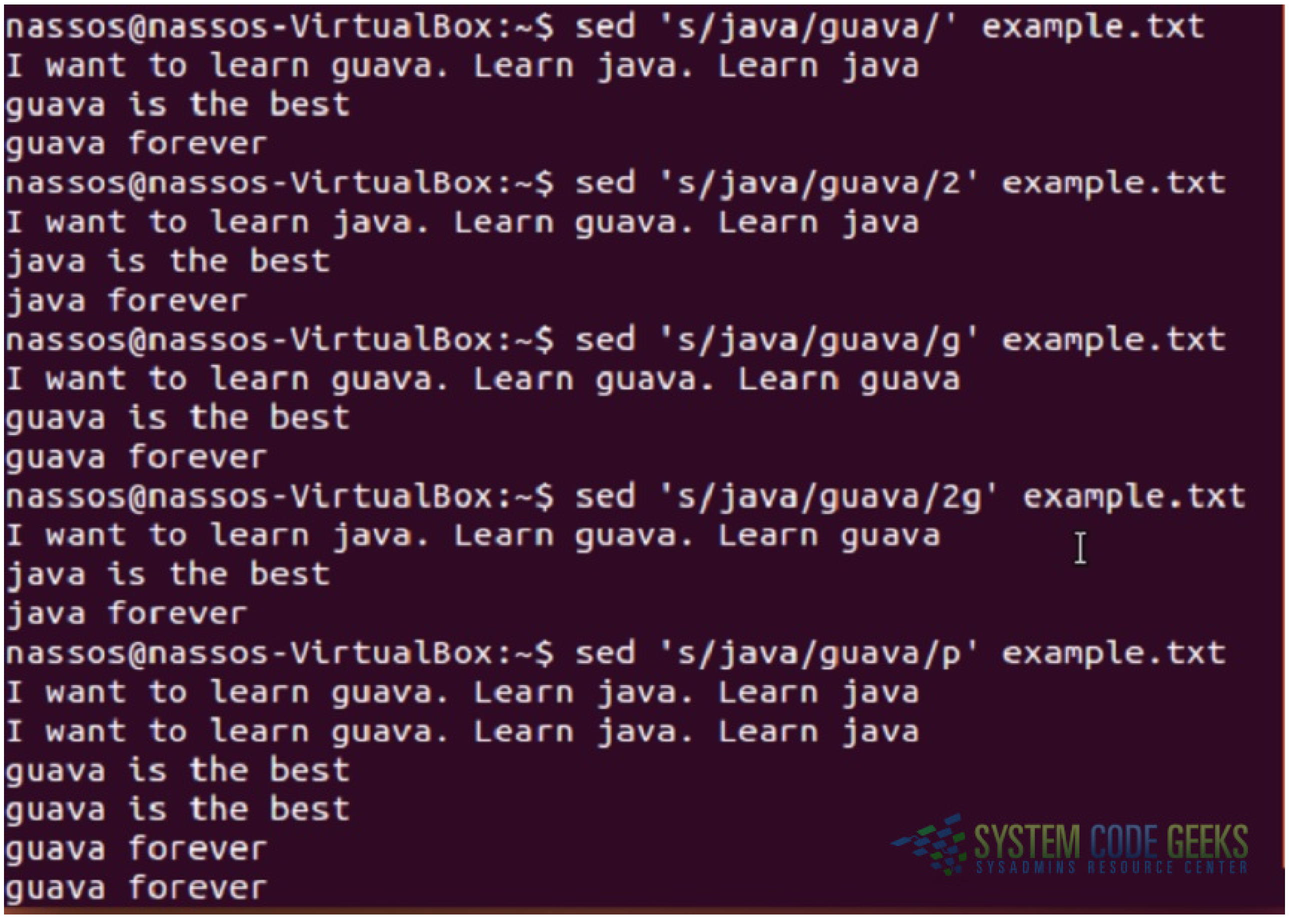

```
nassos@nassos-VirtualBox:~$ sed 's/java/guava/' example.txt
I want to learn guava. Learn java. Learn java
guava is the best
guava forever
nassos@nassos-VirtualBox:~$ sed 's/java/guava/2' example.txt
I want to learn java. Learn guava. Learn java
java is the best
java forever
nassos@nassos-VirtualBox:~$ sed 's/java/guava/g' example.txt
I want to learn guava. Learn guava. Learn guava
guava is the best
guava forever
nassos@nassos-VirtualBox:~$ sed 's/java/guava/2g' example.txt
I want to learn java. Learn guava. Learn guava
java is the best
java forever
nassos@nassos-VirtualBox:~$ sed 's/java/guava/p' example.txt
I want to learn guava. Learn java. Learn java
I want to learn guava. Learn java. Learn java
guava is the best
guava is the best
guava forever
guava forever
```

Figure 8.1: Sed Examples 1-5

8.6 Replacing string on a specific line number

You can restrict the sed command to replace the string on a specific line number. The below sed command replaces the string "java" only on the second line:

```
>sed '2 s/java/guava/' example.txt
```

8.7 Replacing string on a range of lines

You can specify a range of line numbers to the sed command for replacing a string. Here the sed command replaces the lines with range from 1 to 3. You may use the $ operator to indicate the last line in the file:

```
>sed '1,$ s/java/guava/' example.txt
```

8.8 Replacing on a line which matches a pattern

You can specify a pattern to the sed command to match in a line. If the pattern match occurs, then the sed command looks only for the string to be replaced and if it finds it, then it replaces the string. Here the sed command first looks for the lines which have the pattern "java" and then replaces the word "java" with "guava".

```
>sed '/java/ s/java/guava/' example.txt
```

8.9 Deleting lines

You can delete the lines of a file by specifying the line number or a range of numbers:

```
>sed '2 d' example.txt
```

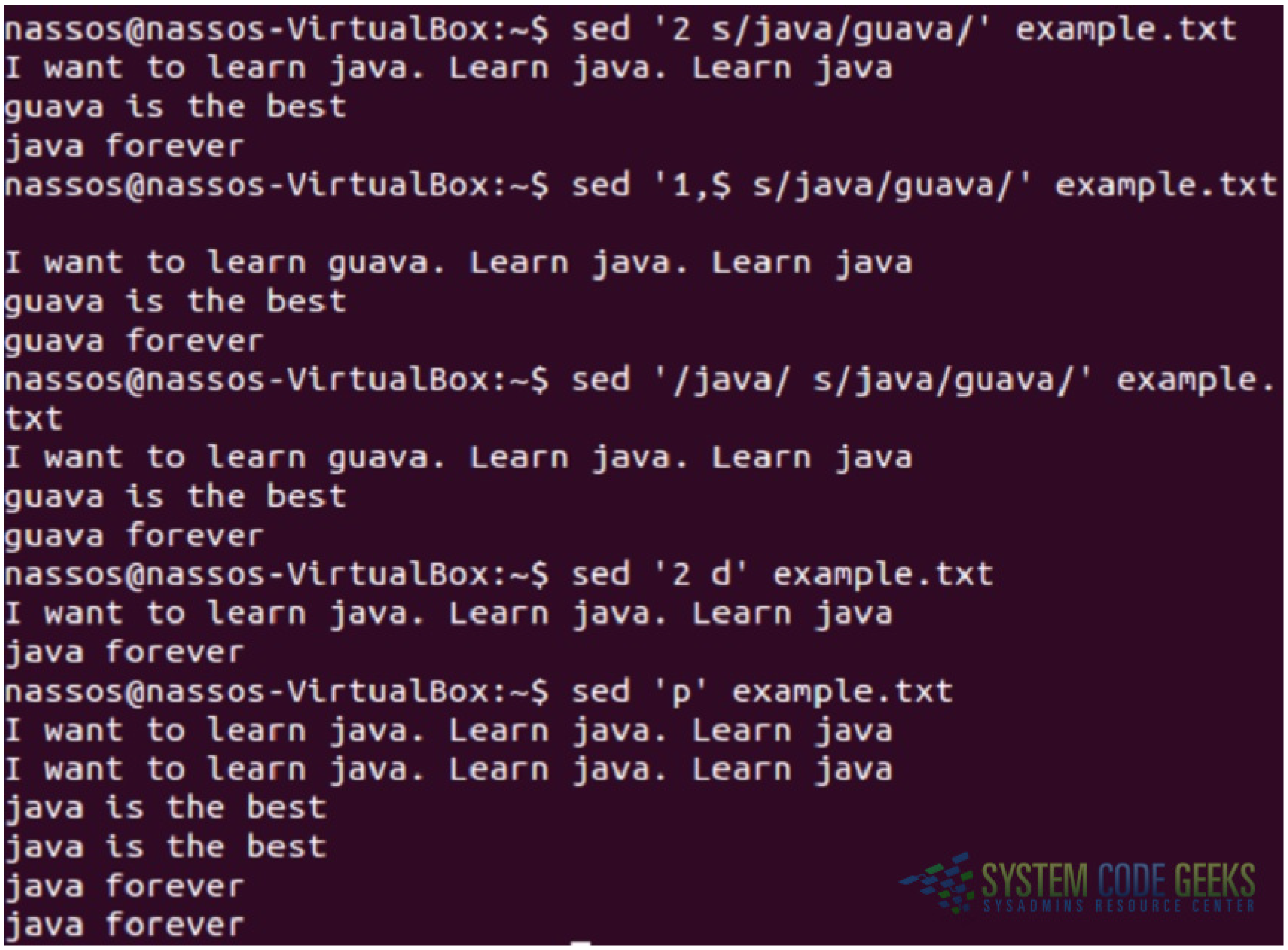
```
>sed '1,$ d' example.txt
```

8.10 Duplicating lines

You can use the sed command to print each line of a file two times:

```
>sed 'p' example.txt
```

The below image illustrates the execution of the previous 5 sed commands:



```
nassos@nassos-VirtualBox:~$ sed '2 s/java/guava/' example.txt
I want to learn java. Learn java. Learn java
guava is the best
java forever
nassos@nassos-VirtualBox:~$ sed '1,$ s/java/guava/' example.txt

I want to learn guava. Learn java. Learn java
guava is the best
guava forever
nassos@nassos-VirtualBox:~$ sed '/java/ s/java/guava/' example.
txt
I want to learn guava. Learn java. Learn java
guava is the best
guava forever
nassos@nassos-VirtualBox:~$ sed '2 d' example.txt
I want to learn java. Learn java. Learn java
java forever
nassos@nassos-VirtualBox:~$ sed 'p' example.txt
I want to learn java. Learn java. Learn java
I want to learn java. Learn java. Learn java
java is the best
java is the best
java forever
java forever
```

Figure 8.2: Sed Examples 6-10

8.11 Changing the slash (/) delimiter

You can use any delimiter other than the slash. As an example if you want to change the web url to another url, using too many backslashes makes the sed command look awkward. In this case we can change the delimiter to another character as shown in the below example:

```
>sed 's_https://_www_' example.txt
```

8.12 Using & as the matched string

There might be cases where you want to search for the pattern and replace that pattern by adding some extra characters to it. In such cases & comes in handy. The & represents the matched string:

```
>sed 's/java/{&}/' example.txt
```

8.13 Using 1,2 and so on to 9

The first pair of parenthesis specified in the pattern represents the 1, the second represents the 2 and so on. The 1,2 can be used in the replacement string to make changes to the source string. As an example, if you want to replace the word "java" in a line with twice as the word like "javajava" use the sed command as below:

```
>sed 's/\(java\)/\1\1/' example.txt
```

8.14 Running multiple sed commands

You can run multiple sed commands by piping the output of one sed command as input to another sed command. Sed provides also an -e option to run multiple sed commands in a single sed command:

```
>sed 's/java/guava/' example.txt | sed 's/guava/python/'
```

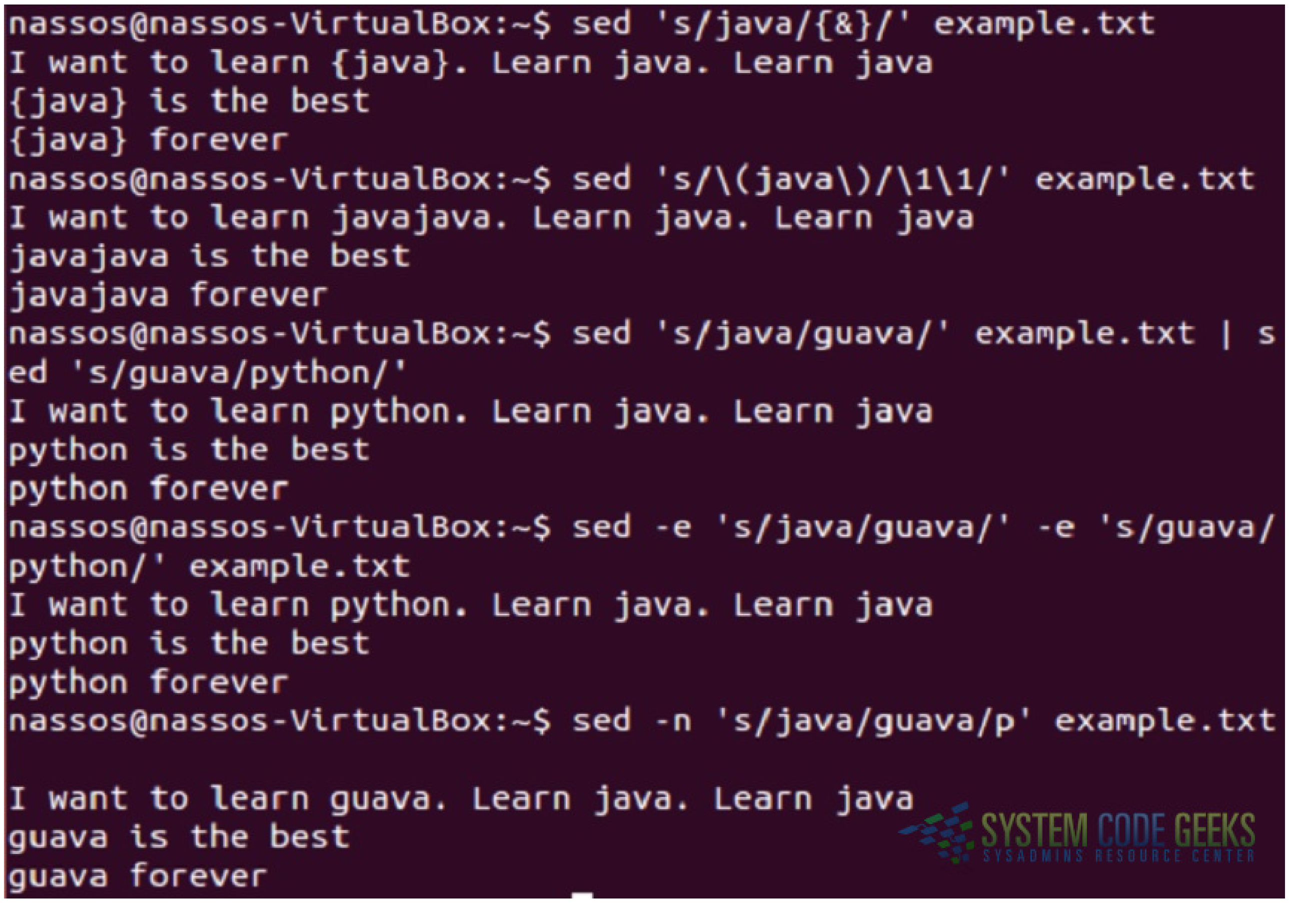
```
>sed -e 's/java/guava/' -e 's/guava/python/' example.txt
```

8.15 Printing only the replaced lines

Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time:

```
>sed -n 's/java/guava/p' example.txt
```

The below image illustrates the execution of the previous 5 sed commands:



```
nassos@nassos-VirtualBox:~$ sed 's/java/{&}/' example.txt
I want to learn {java}. Learn java. Learn java
{java} is the best
{java} forever
nassos@nassos-VirtualBox:~$ sed 's/\(java\)/\1\1/' example.txt
I want to learn javajava. Learn java. Learn java
javajava is the best
javajava forever
nassos@nassos-VirtualBox:~$ sed 's/java/guava/' example.txt | s
ed 's/guava/python/'
I want to learn python. Learn java. Learn java
python is the best
python forever
nassos@nassos-VirtualBox:~$ sed -e 's/java/guava/' -e 's/guava/
python/' example.txt
I want to learn python. Learn java. Learn java
python is the best
python forever
nassos@nassos-VirtualBox:~$ sed -n 's/java/guava/p' example.txt

I want to learn guava. Learn java. Learn java
guava is the best
guava forever
```

Figure 8.3: Sed Examples 11-15

8.16 Using sed as grep

You can make sed command to work as similar to grep command. Here the sed command looks for the pattern "java" in each line of a file and prints those lines that have the pattern:

```
>grep 'java' example.txt
```

```
>sed -n '/java/ p' example.txt
```

8.17 Adding a line after a match is found.

The sed command can add a new line after a pattern match is found. The "a" command to sed tells it to add a new line after a match is found:

```
>sed '/java/ a "Add a new line"' example.txt
```

8.18 Adding a line before a match

The sed command can add a new line before a pattern match is found. The "i" command to sed tells it to add a new line before a match is found:

```
>sed '/java/ i "New line"' example.txt
```

8.19 Changing a line

The sed command can be used to replace an entire line with a new line if a match is found. The "c" command to sed tells it to change the line.

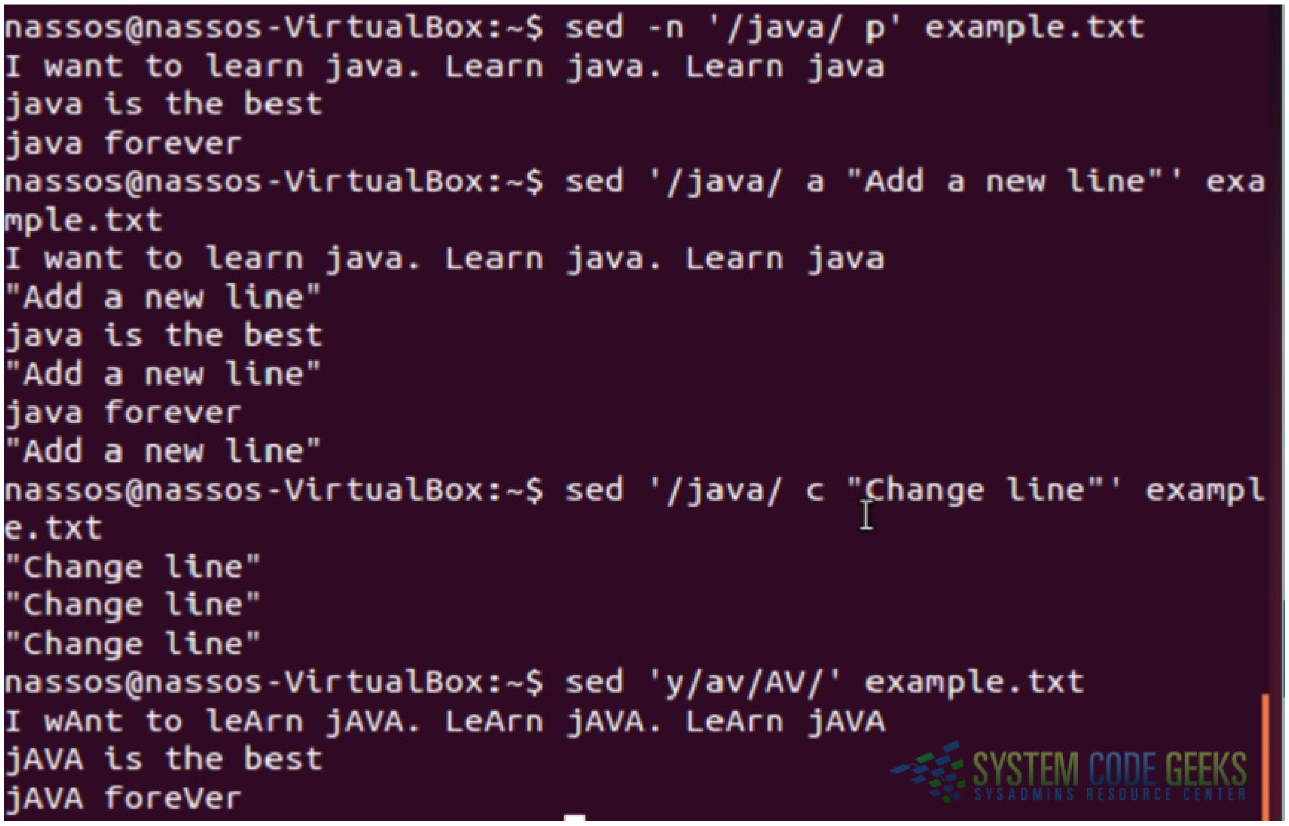
```
>sed '/java/ c "Change line"' example.txt
```

8.20 Transforming like tr command

The sed command can be used to convert the lower case letters to upper case letters by using the transform "y" option. In the below example the sed command transforms the alphabets "av" into their uppercase format "AV"

```
>sed 'y/av/AV/' example.txt
```

The below image illustrates the execution of the last 5 sed commands:



```
nassos@nassos-VirtualBox:~$ sed -n '/java/ p' example.txt
I want to learn java. Learn java. Learn java
java is the best
java forever
nassos@nassos-VirtualBox:~$ sed '/java/ a "Add a new line"' exa
mple.txt
I want to learn java. Learn java. Learn java
"Add a new line"
java is the best
"Add a new line"
java forever
"Add a new line"
nassos@nassos-VirtualBox:~$ sed '/java/ c "Change line"' exampl
e.txt
"Change line"
"Change line"
"Change line"
nassos@nassos-VirtualBox:~$ sed 'y/av/AV/' example.txt
I wAnt to leArn jAVA. LeArn jAVA. LeArn jAVA
jAVA is the best
jAVA foreVer
```

Figure 8.4: Sed Examples 16-20

8.21 Man sed

We tried to present some basic examples of using the sed command. Of course you may always use the man command to find the full functionality of the sed command:

```
>man sed
```

Chapter 9

Linux cut Examples

Cut is a Unix command line utility which is used to extract sections from each line of input, usually from a file. It is currently part of the GNU coreutils package and the BSD Base System. It first appeared in AT&T System III UNIX in 1982. Extraction of line segments can typically be done by bytes (-b), characters (-c), or fields (-f) separated by a delimiter (-d - the tab character by default).

A range must be provided in each case which consists of one of N, N-M, N- (N to the end of the line), or -M (beginning of the line to M), where N and M are counted from 1 (there is no zeroth value). Basically the cut command slices a line and extracts the text.

In this article, we will start with a few basic examples and then explore all the advanced options that *cut* provides. First consider the below text file as an example:

```
>cat example.txt
I want to learn linux, learn linux, learn linux.
Linux is the best
Linux forever
```

9.1 Printing characters by position

The cut command can be used to print characters in a line by specifying the position of the characters. To print the characters in a line, use the -c option in cut command. The below cut command prints the fourth character in each line of the file. You can also print more than one character at a time by specifying the character positions in a comma separated list.

```
>cut -c4 example.txt
```

9.2 Printing characters by range

You can print a range of characters in a line by specifying the start and end position of the characters. The below cut command prints the characters from fourth position to the seventh position in each line:

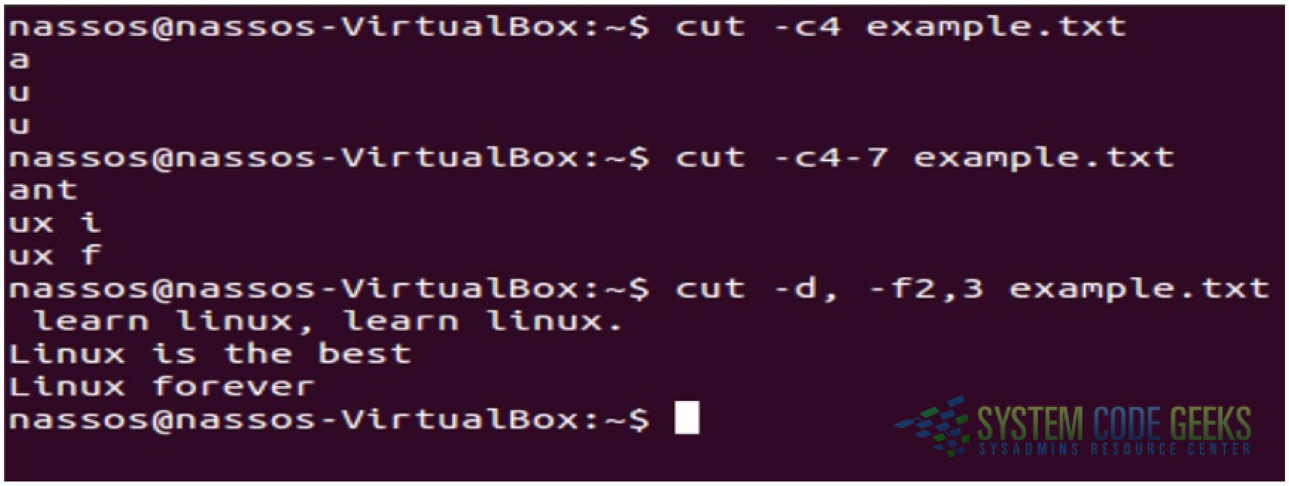
```
>cut -c4-7 example.txt
```

9.3 Printing fields using *comma* delimiter

You can use the cut command just as awk command to extract the fields in a file using a delimiter. The -d option in cut command can be used to specify the delimiter and -f option is used to specify the field position. The below cut command uses comma as a delimiter and prints the second and third field in each line:

```
>cut -d, -f2,3 example.txt
```

The below image illustrates the execution of the first 3 cut commands:



```
nassos@nassos-VirtualBox:~$ cut -c4 example.txt
a
u
u
nassos@nassos-VirtualBox:~$ cut -c4-7 example.txt
ant
ux i
ux f
nassos@nassos-VirtualBox:~$ cut -d, -f2,3 example.txt
 learn linux, learn linux.
Linux is the best
Linux forever
nassos@nassos-VirtualBox:~$
```

Figure 9.1: Cut Examples 1-3

9.4 Printing fields using space delimiter

The below cut command prints the second field in each line by treating the space as delimiter. You can also print more than one fields by specifying the position of the fields in a comma delimited list.

```
>cut -d' ' -f2 example.txt
```

9.5 Displaying range of fields

You can print a range of fields by specifying the start and end position. The below cut command prints the first, second and third fields:

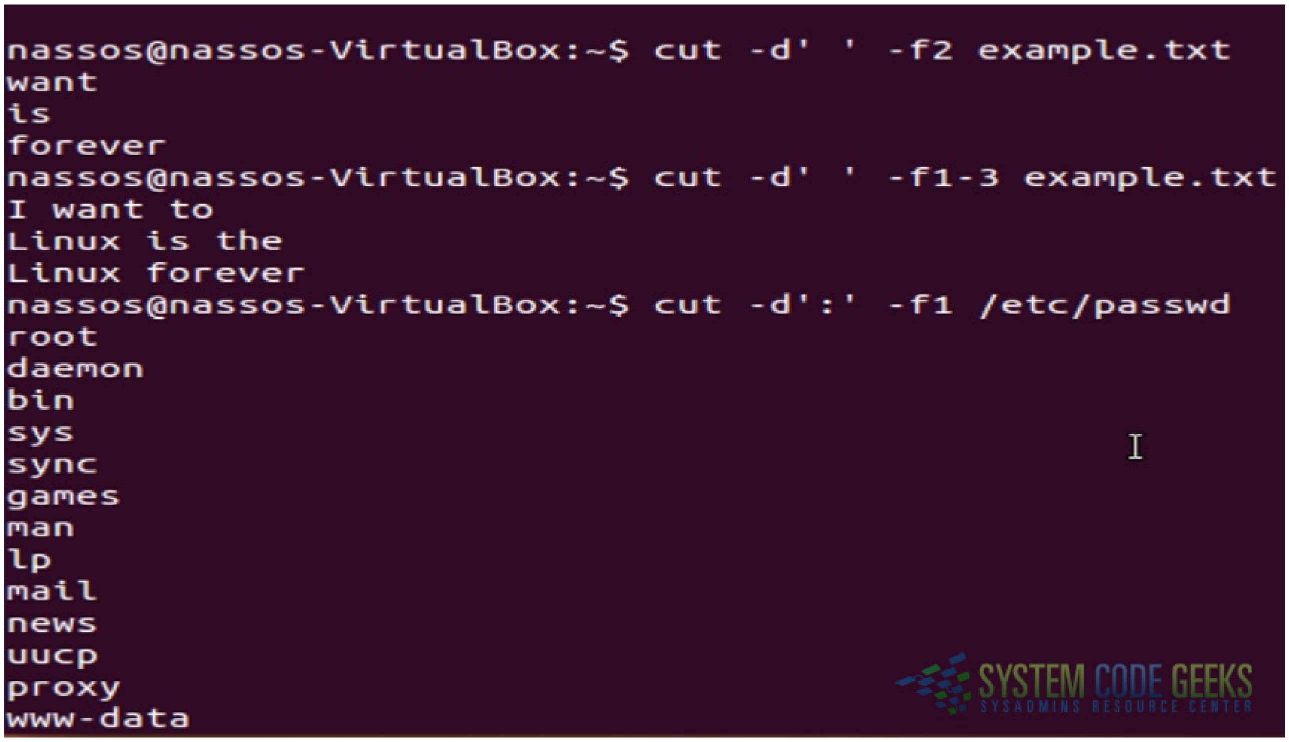
```
>cut -d' ' -f1-3 example.txt
```

9.6 Displaying first field from file

The /etc/passwd is a delimited file and the delimiter is a colon (:). The cut command to display the first field in /etc/passwd file is the following:

```
>cut -d':' -f1 /etc/passwd
```

The below image illustrates the execution of the previous 3 cut commands:



```
nassos@nassos-VirtualBox:~$ cut -d' ' -f2 example.txt
want
is
forever
nassos@nassos-VirtualBox:~$ cut -d' ' -f1-3 example.txt
I want to
Linux is the
Linux forever
nassos@nassos-VirtualBox:~$ cut -d':' -f1 /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
```

Figure 9.2: Cut Examples 4-6

9.7 Displaying fields from 1st to nth

As we already shown in 5th example, a range of fields can be printed by specifying the start and end position. There is also a possibility to omit the start position and declare only the last. The below cut command prints all fields from first to 3rd using space as a delimiter:

```
>cut -d' ' -f-3 example.txt
```

9.8 Displaying fields from nth to last

You can also print all fields by omitting the last position and declaring only the start. The below cut command prints from second field until the last one using space as delimiter:

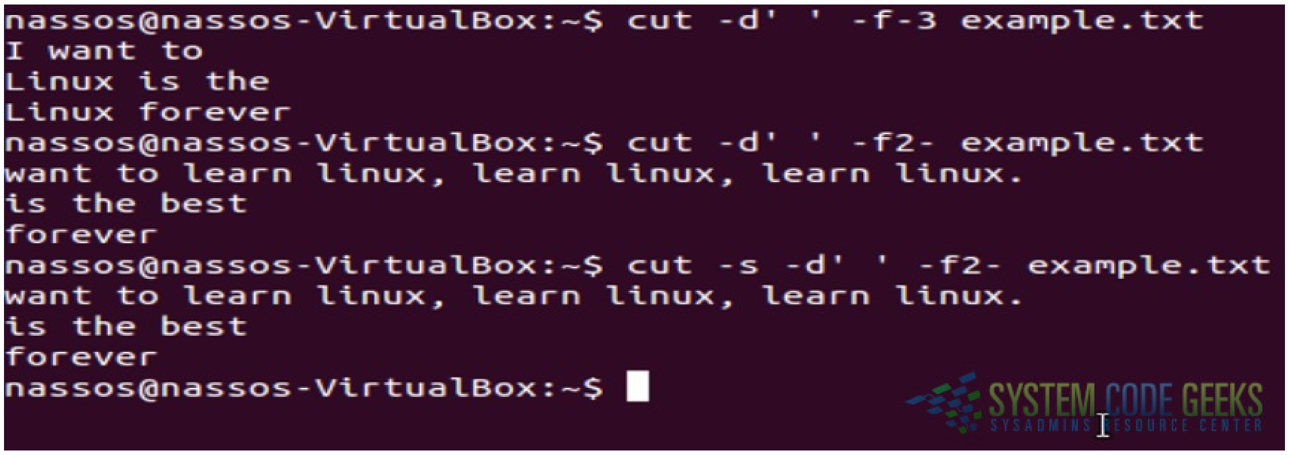
```
>cut -d' ' -f2- example.txt
```

9.9 Ignoring lines that do not contain delimiter

The cut command can be also be used to print only the lines that do not contain a certain delimiter by using the -s option. The below cut command prints only the lines which contain the comma delimiter and fields from 2nd to last:

```
>cut -s -d' ' -f2- example.txt
```

The below image illustrates the execution of the previous 3 cut commands:



```
nassos@nassos-VirtualBox:~$ cut -d' ' -f-3 example.txt
I want to
Linux is the
Linux forever
nassos@nassos-VirtualBox:~$ cut -d' ' -f2- example.txt
want to learn linux, learn linux, learn linux.
is the best
forever
nassos@nassos-VirtualBox:~$ cut -s -d' ' -f2- example.txt
want to learn linux, learn linux, learn linux.
is the best
forever
nassos@nassos-VirtualBox:~$
```

Figure 9.3: Cut Examples 7-9

9.10 Inverting field selection

The --complement option can be used with the cut command in order to print the inverted fields declared with the -f option. The below cut command prints all except for the 2nd field from each line using space as a delimiter:

```
>cut --complement -d' ' -f2 example.txt
```

9.11 Specifying delimiter to be used in output

The previous cut commands which were used to print fields using a delimiter can be modified to have a different delimiter as output, for example use space as delimiter to find fields but print the fields using comma as a delimiter. The below cut command uses space as a delimiter and prints the second and third fields from each line but the output contains comma separated fields:

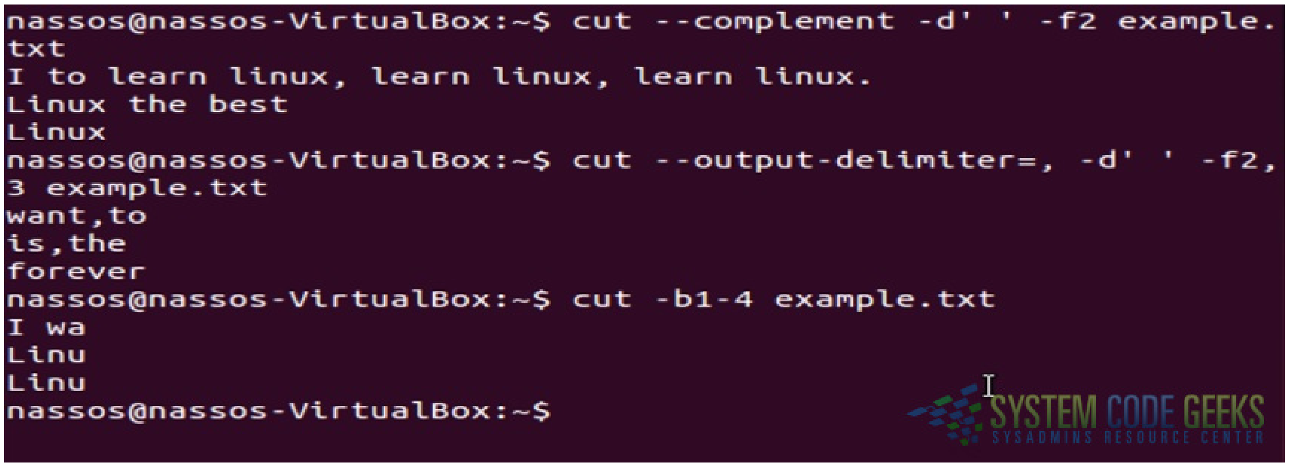
```
>cut --output-delimiter=, -d' ' -f2,3 example.txt
```

9.12 Printing bytes

Similarly as printing characters using the cut command, you can also print bytes using the -b option. The below cut command prints from first to fourth byte from each line:

```
>cut -b1-4 example.txt
```

The below image illustrates the execution of the last 3 cut commands:



```
nassos@nassos-VirtualBox:~$ cut --complement -d' ' -f2 example.
txt
I to learn linux, learn linux, learn linux.
Linux the best
Linux
nassos@nassos-VirtualBox:~$ cut --output-delimiter=, -d' ' -f2,
3 example.txt
want,to
is,the
forever
nassos@nassos-VirtualBox:~$ cut -b1-4 example.txt
I wa
Linu
Linu
nassos@nassos-VirtualBox:~$
```

Figure 9.4: Cut Examples 10-12

9.13 man cut

We tried to present some basic examples of using the cut command. Of course you may always use the man command to find the full functionality of the linux cut command:

```
>man cut
```

Chapter 10

Linux curl Examples

cURL is an incredibly flexible and powerful tool data transferring tool, that supports a wide variety of protocols, being the HTTP (and HTTPS) the most used. cURL library is available for many programming languages, but in this example we will see how to use it from the command line.

For this tutorial, Mint 17.3 has been used.

10.1 Installation

cURL can be installed easily via `apt-get`:

```
sudo apt-get update
sudo apt-get install curl
```

10.2 Basic usage

The simplest way to use cURL is just executing it, and specifying a site:

```
curl <site>
```

For example

```
curl google.com
```

And the terminal would show:

```
<HTML>
<HEAD>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <TITLE>301 Moved</TITLE>
</HEAD>
<BODY>
    <H1>301 Moved</H1>
    The document has moved
    <A HREF="https://www.google.es/">here</A>.
</BODY>
</HTML>
```

Which is not what we were expecting. This is because, **by default, cURL doesn't follow 3XX redirections**. To make cURL follow redirections, we have to pass the `-L` option:

```
curl -L google.com
```

To save the HTTP response returned by cURL, we can always make a shell redirect with >, or use the `-o` option:

```
curl -o google.html -L google.com
```

When used, data about the receiving response is shown:

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   219  100   219    0     0   2259      0 --:--:-- --:--:-- --:--:--  2281
  0     0    0 11675    0     0  50630      0 --:--:-- --:--:-- --:--:-- 10.5M
```

To see more information, including the HTTP headers, we have available the `-v` (`--verbose`) option:

```
curl -v google.com
```

Which will show something similar to the following before the HTTP response:

```
* Rebuilt URL to: google.com/
* Hostname was NOT found in DNS cache
*   Trying 130.206.193.53...
* Connected to google.com (130.206.193.53) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.35.0
> Host: google.com
> Accept: */*
>
< HTTP/1.1 302 Found
< Cache-Control: private
< Content-Type: text/html; charset=UTF-8
< Location: https://www.google.es/?gfe_rd=cr&ei=OulzV836N9Op8weYzoxY
< Content-Length: 256
< Date: Wed, 29 Jun 2016 15:28:58 GMT
<
# HTML
```

10.3 Requests

10.3.1 GET

Making GET request with cURL is so simple. Actually, is as same as making it in the browser. For example:

```
curl -o systemcodegeeks.html https://www.systemcodegeeks.com?s=linux
```

We just have to add the query string in `field=value&` format after `?`.

10.3.2 POST

To make a POST request, the usage is slightly different. The syntax is the following:

```
curl -d "field1=value1&field2=value2&fieldN=valueN" <site>
```

So, for making a search as the one done above, but supposing that the form method is POST, would be:

```
curl -d "s=linux" systemcodegeeks.com -o systemcodegeeks.html
```

Of course, we can make a POST request for uploading files:

```
curl -F "field=@/path/to/file.txt" awebsite.com/uploadfile
```

In this case, we have to specify the -F option for the file. And, the leading @ before the path to the file is necessary.

For making a POST request that combines both files and normal data, the data has to bet set also with -F, and not with -d. This is because the forms for file uploads sets the data encoding to -multipart/form-data, and this is what -F option does; whereas -d option sets the encoding to application/x-www-form-urlencoded. And all the data in a form has to be encoded under the same encoding. So, we have to specify also the data with -F option:

```
curl -F "field=value" -F "file=@/path/to/file.txt" awebsite.com/uploadfile
```

10.3.3 Custom requests: PUT and DELETE

PUT and DELETE are considered custom-methods. These requests are mostly used in CRUDs (Create, Read, Update, Delete), in RESTful services. To specify these commands, we have to use -X option.

For example:

```
curl -X PUT -d "data=stuff" https://myserver.com/rows/1
curl -X DELETE https://myserver.com/rows/1
```

10.3.4 Adding extra headers

For adding extra headers to a request, we just have to use the -H option (--header), specifying the header, and the value:

```
curl --header "X-ExtraHeader: myheader" https://myserver.com
```

10.4 FTP

We can also manage FTP server with cURL:

```
curl -u username:password ftp://myftp.com
# Or
curl fpt://username:password@myftp.com
```

We must take into account that, unless SSL is being used, the data will travel in plain text. If SSL is available, ftps should be use, instead of ftp.

We can upload files pretty simply, just specifying the local file with -T option:

```
curl -T file.txt ftps://username:password@myftp.com/path
```

Several files can be uploaded at once:

```
curl -T {file1.txt, file2.txt} ftps://username:password@myftp.com/path
```

Downloading files from FTP server is the same as downloading a simple HTML:

```
curl ftps://username:password@myftp.com/path/file.txt -o file.txt
```

10.5 Other interesting options

Apart from the common uses of cURL, let's see other features that may result useful in some occasions.

10.5.1 HTTP authentication

With cURL is also possible to make HTTP authentication (that login prompts shown by the browser, not the authentication forms). For that, we have to use the `-u` (`--user`) option and specify the credentials:

```
curl -u username:password https://mylogin.com
```

10.5.2 Ignoring SSL errors

You will probably ever have encountered a site for which the browser shows a SSL certificate error, that it can be caused because it's expired, because the certificate authority is not trusted by the browser, or any other reason. If we want to ignore the warning and access the site, we have to give explicit consent to the browser. By default, cURL does not accept the connections with SSL errors, but we can make it ignore them with `-k` (`--insecure`) option:

```
curl -k https://insecure-site.com
```

10.5.3 Cookies

Setting custom cookies with cURL is as easy as specifying them in `name=value` format, with `-b` (`--cookie`) option:

```
curl -b "mycookie=value" systemcodegeeks.com
```

We can even specify the cookies from a file!

```
curl -b cookies.txt systemcodegeeks.com
```

And, of course, we can save the cookies to a file, with `-c` (`--cookie-jar`) option:

```
curl -c cookies.txt systemcodegeeks.com
```

10.5.4 Making requests through proxy

Yes, cURL also allows to make requests through proxies. We just have to use the `-x` option, specifying the proxy:

```
curl -x https://aproxy.com systemcodegeeks.com
```

If the proxy server requires authentication, we have to specify as same as with HTTP authentication, but with `--proxy-user` option:

```
curl -x https://aproxy.com --proxy-user username:password systemcodegeeks.com
```

10.5.5 Download depending on the modification time

This is nice. We can make cURL download a file only if it has suffered modifications after the specified time, with `-z` (`--time-cond`) option:

```
curl -z 25 -Jun-16 systemcodegeeks.com
```

This feature can be specially useful for automating the task of having up-to-date documentation or software, for example.

10.6 Summary

This example has shown how to use cURL, from its simplest uses, which are downloading pages or making simple requests, to more advanced topics (which are actually easy with cURL), such us FTP connection, setting and saving cookies, or using proxies.