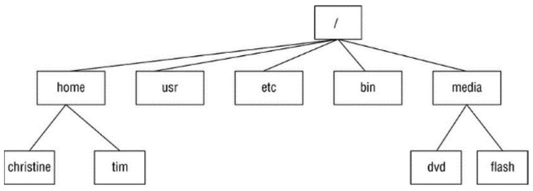
CHAPTER 2 MANAGING DIRECTORY, FILE, USER

Linux uses a unified directory tree, which means that all files can be located relative to a single root directory, which is often referred to using the slash (/) character. If your disk contains multiple partitions, one of these devices becomes the root filesystem, and others are mounted at some location within the overall directory tree. The same thing happens when you mount a USB flash drive, DVD, or other removable disk device.



Home Directory References The tilde (~) character refers to the user's home directory.

MANIPULATING DIRECTORIES

You are probably familiar with the concept of directories, although you may think of them as "folders".

CREATING DIRECTORIES

You can use the mkdir command to create a directory. Ordinarily, you'll use this command by typing the name of one or more directories following the command:

```
$ mkdir newdir
```

```
$ mkdir dirone newdir/dirtwo
```

The first example creates just one new directory, newdir, which will then reside in the current directory. The second example creates two new directories: dirone and newdir/dirtwo. In this example, mkdir creates dirtwo inside the newdir directory, which was created with the preceding command.

DELETING DIRECTORIES

The rmdir command is the opposite of mkdir; it destroys a directory.

To use it, you normally type the command followed by the names of one or more directories that you want to delete:

```
$ rmdir dirone
```

```
$ rmdir newdir/dirtwo newdir
```

These examples delete the three directories created by the mkdir commands shown earlier.

Normally, if a directory contains files or other directories, rmdir doesn't delete it and returns an error message. Of course, in real life you're likely to want to delete directory trees that hold files. In such cases, you can use the rm command,described earlier in "Deleting Files," along with its -r.

```
$ rm -r newdir
```

This command deletes newdir and any files or subdirectories that it might contain. This fact makes rm and its -r option potentially dangerous, so you should be particularly cautious when using it.

CREATING FILES

Normally, you create files using the programs that manipulate them.

You can type this program's name followed by the name of a file that you want to create, such as **touch newfile.txt** to create an empty file called newfile.txt.

If you pass touch the name of a file that already exists, touch updates that file's access and modification time stamps to the current date and time.

COPYING FILES

The cp command copies a file. (Its name is short for copy.) To use it, you can pass cp a source filename and a destination filename, a destination directory name, or both.

cp orig.txt new.txt   : Copies orig.txt to new.txt in the current directory
cp orig.txt /otherdir : Copies orig.txt to the /otherdir directory. The copy will be called orig.txt.

cp orig.txt /otherdir/new.txt   : Copies orig.txt to the /otherdir directory. The copy will be called new.txt.

MOVING AND RENAMING FILES

Its use is similar to that of cp.

If you specify a filename with the destination, the file will be renamed as it's moved.
If you specify a filename and the destination directory is the same as the source directory, the file will be renamed but not moved.

USING LINKS

Sometimes it's handy to refer to a single file by multiple names. Rather than create several copies of the file, you can create multiple links to one file. Linux supports two types of links, both of which are created with the **ln command:**

Hard Link
A hard link is a duplicate directory entry. Both entries point to the same file. In a hard link scenario, neither filename holds any sort of priority over the other; both tie directly to the file's data structures and data.
Type ln origname linkname, where origname is the original name and linkname is the new link's name, to create a hard link.

Symbolic Link A
symbolic link (aka soft link) is a file that refers to another file by name. That is, the symbolic link is a file that holds another file's name, and when you tell a program to read to or write from a symbolic link file, Linux redirects the access to the original file.

Type ln -s origname linkname to create a symbolic link.

Symbolic links are similar to shortcuts on the Windows desktop.

You can identify links in long directory listings (using the -l option to ls).
The following example illustrates this:

```
$ touch report.odt

$ ln report.odt hardlink.odt
$ ln -s report.odt softlink.odt
```

```
$ ls -l
```

The two commands created two links, a hard link (hardlink.odt) and a symbolic link (softlink.odt). Typing ls -l shows all three files. The original file and the hard link can be identified as links by the presence of the value 2 in the second column of the ls -l output; this column identifies the number of filename entries that point to the file, so a value higher than 1 indicates that a hard link exists. The symbolic link is denoted by an l (a lowercase L, not a digit 1) in the first character of the softlink.odt file's permissions string (lrwxrwxrwx). Furthermore, the symbolic link's filename specification includes an explicit pointer to the linked-to file.

If you use symbolic links, deleting the original file makes the file completely inaccessible; the symbolic links remain but point to a nonexistent file. If you use hard links, by contrast, you must delete all copies of the file to delete the file itself. This is because hard links are duplicate directory entries that point to the same file, whereas symbolic links are separate files that refer to the original file by name.

Deleting Files

The rm command deletes files in a shell. As you might expect, you pass the names of one or more files to this command:

```
$ rm outline.pdf outline.txt
```

This example deletes two files, outline.pdf and outline.txt. If you want to delete an entire directory tree, you can pass rm the -r, -R, or –recursive option along with a directory name:

```
$ rm -r oldstuff/
```

The -i option causes rm to prompt before deleting each file. This is a useful safety measure.

You should be careful when using rm and even more careful when using it with its -r.

CREATING USERS AND GROUPS

Linux is a multi-user system, which means that more than one person can interact with the same system at the same time. As a system administrator, you have the responsibility to manage the system's users and groups by creating and removing users and assign them to different groups .

useradd Command

The general syntax for the useradd command is as follows:

```
useradd [OPTIONS] USERNAME
```

Only root or users with sudo privileges can use the useradd command to create new user accounts.

To create a new user account, invoke the useradd command followed by the name of the user.
For example to create a new user named username you would run:

```
$ sudo useradd username
```

Or

```
# useradd username
```

When executed without any option, useradd creates a new user account using the default settings.

The command adds an entry to the:
/etc/passwd,
/etc/shadow,
/etc/group
and /etc/gshadow files.

To be able to log in as the newly created user, you need to set the user password. To do that run the passwd command followed by the username:

```
sudo passwd username
```

You will be prompted to enter and confirm the password.
Make sure you use a strong password.

HOW TO ADD A NEW USER AND CREATE HOME DIRECTORY

On most Linux distributions, when creating a new user account with useradd, the user's home directory is not created.

Use the -m (--create-home) option to create the user home directory as /home/username:

```
sudo useradd -m username
```

The command above creates the new user's home directory.
If you list the files in the /home/username directory, you will see the initialization files:

```
ls -la /home/username/
```

Within the home directory, the user can write, edit and delete files and directories.

CREATING A USER WITH SPECIFIC GROUP ID

Linux groups are organization units that are used to organize and administer user accounts in Linux. The primary purpose of groups is to define a set of privileges such as reading, writing, or executing permission for a given resource that can be shared among the users within the group.

When creating a new user, the default behavior of the useradd command is to create a group with the same name as the username, and same GID as UID.

The -g (--gid) option allows you to create a user with a specific initial login group.
You can specify either the group name or the GID number. The group name or GID must already exist.

The following example shows how to create a new user named username and set the login group to users type:

```
sudo useradd -g users username
```

To verify the user's GID, use the id command:

```
id -gn username
```

CREATING A USER WITH SPECIFIC LOGIN SHELL
In some distributions the default shell is set to /bin/sh while in others it is set to /bin/bash.

The -s (--shell) option allows you to specify the new user's login shell.

For example, to create a new user named username with /usr/bin/zsh as a login shell type:

```
sudo useradd -s /usr/bin/zsh username
```

CREATING A USER WITH AN EXPIRY DATE

To define a time at which the new user accounts will expire, use the -e (--expiredate) option. This is useful for creating temporary accounts.

The date must be specified using the YYYY-MM-DD format.

For example to create a new user account named username with an expiry time set to January 22 2022 you would run:

```
sudo useradd -e 2022-01-22 username
```

Use the chage command to verify the user account expiry date:

```
sudo chage -l username
```

DELETING ACCOUNTS

The userdel command deletes accounts from a text-mode shell. In its simplest form, using super user privileges you pass it a username and nothing more:

```
$ sudo userdel hwash
```

The program doesn’t prompt you for confirmation; it just deletes the account.

It does not, however, delete the user’s home directory by default. To have it do so, pass it the --remove (-r) option.

If the user is currently logged in, userdel notifies you of that fact and does nothing. You can pass it the --force (-f) option to delete the account even though it’s in use. To both force account deletion and remove the user’s files, you can pass both options:

```
$ sudo userdel -rf zwash
```

MANAGING GROUPS

You can create groups from the shell by using the groupadd command.

Using the groupadd command, including setting an option, looks like this:

```
# groupadd -g 1001 consultants
```

This example creates a group with a GID of 1001 and a group name of consultants. Notice in the preceding example that the # prompt is shown, indicating that the root account was used to obtain the super user privileges required to complete this command successfully.

After the group is added, new members can be added to the group. This requires the use of the usermod command and super user privileges, as follows:

```
# usermod -aG consultants rich

groups rich
```

The -aG options were used together to add the account, rich, to the new consultants group. If the -a option was not used, the rich account on some distributions would be removed from its current group memberships and belong only to the consultants group. Thus, it is wise to use -aG by default.

Good habit is to check if the modification was successful by using the groups command. The groups command shows all the group memberships for the specified user account.

Deleting groups from the shell involves use of the groupdel command, which takes a group name as a single option, as in groupdel consultants to delete the consultant group.

DOING MATH IN BASH WITH INTEGER

command let

```
myvar=6 ; echo $myvar

let myvar+=1 ; echo $myvar

let myvar+1 ; echo $myvar

let myvar2=myvar+1 ; echo $myvar2
```

USING THE BASH ARITHMETIC EXPANSION

The recommended way to evaluate arithmetic expressions with integers in Bash is to use the Arithmetic Expansion capability of the shell. The builtin shell expansion allows you to use the parentheses ((...)) to do math calculations.

The format for the Bash arithmetic expansion is $((arithmetic expression)). The shell expansion will return the result of the latest expression given.

The $((...)) notation is what is called the Arithmetic Expansion while the ((...))

```
x=3
 echo $x

echo $((x+2))

x=$((x+3))

echo $x

((x+=3))

echo $x
```

CREATING SCRIPTS

A script is a program written in an interpreted language, typically associated with a shell or other program whose primary purpose is something other than as an interpreted language. In Linux, many scripts are shell scripts, which are associated with Bash or another shell.

A shell script begins with a line that identifies the shell that’s used to run it, such as the following:

```
#!/bin/bash
```

The first two characters are a special code that tells the Linux kernel that this is a script and to use the rest of the line as a pathname to the program that interprets the script. (This line is sometimes

called the shebang.)
Shell scripting languages use a hash mark (#) as a comment character, so the script utility ignores this line, although the kernel doesn't.

For instance, to make a file called my-script executable, you issue the following command:

```
$ chmod a+x my-script
```

You'll then be able to execute the script by typing its name, possibly preceded by ./ to tell Linux to run the script in the current directory rather than searching the current path.

USING ARGUMENTS

Variables can help you expand the utility of scripts. A variable is a placeholder in a script for a value that will be determined when the script runs. Variables values can be passed as parameters to a script.

Variables

You can use variables as in any programming languages. There are no data types. A variable in bash can contain a number, a character, a string of characters.

You have no need to declare a variable, just assigning a value to its reference will create it.

```
#!/bin/bash

        STR="Hello World!"

        echo $STR
```

Line 2 creates a variable called STR and assigns the string "Hello World!" to it. Then the VALUE of this variable is retrieved by putting the '$' in at the beginning. Please notice that if you don't use the '$' sign, the output of the program will be different, and probably not what you want it to be.
Variables that are passed to the script are frequently called parameters or arguments. They're represented in the script by a dollar sign ($) followed by a number from 0 up; $0 stands for the name of the script, $1 is the first parameter to the script, $2 is the second parameter, and so on.

Example:

When you run the script, you must provide the user account as a parameter on the command line. The script retrieves that value using the $1 variable and creates an account based on the value with the useradd command. It then changes the account's password using the passwd command (the script prompts you to enter the password when you run the script).

```
#!/bin/bash
useradd -m $1
passwd $1
```