コース: ALX Software Engineering

Bash Notebook

作家: Anrich Tait

Abstract

My notes on the bash shell, this will also include some useful information regarding Unix systems. Most of my learning in this regard was in relation to my software engineering studies.

Contents

Chapter 1

Shell Basics

1.1 Objectives

1. What is the shell?
2. Navigation.
3. Looking around
4. A guided tour
5. Manipulating files
6. Working with commands
7. Reading man pages
8. Keyboard shortcuts
9. LTS
10. Shebang

1.2 Resources

1. What is the shell?
2. Navigation

3. Looking around
4. A guided tour
5. Manipulating files
6. Working with commands
7. Reading man pages
8. Keyboard shortcuts
9. LTS
10. Shebang

Man Pages

cd, ls, pwd, less, file, ln, cp, mv, rm, mkdir, type, which, help, man

1.3 Notes

1.3.1 What is the shell?

The shell is a command line interface (CLI) that takes commands and passes them to this operating system.

Bash is the most common example of a shell program, others include: ksh, tcsh and zsh.

Most interactions with the shell are done through a terminal like gnome, alacritty or kitty.

NOTE: Make sure that the last symbol of your shell prompt is not #. If it is this means that you are in sudo (super user) mode and this can be dangerous.

1.3.2 Navigation

Nothing much here for me to learn, maybe important to note thoguh:
Important facts about file names:

1. File names that begin with a period character are hidden. This only means that ls will not list them unless we say ls -a. When your account was created, several hidden files were placed in your home directory to configure things for your account. Later on we will take a closer look at some of these files to see how you can customize our environment. In addition, some applications will place their configuration and settings files in your home directory as hidden files.

2. File names in Linux, like Unix, are case sensitive. The file names "File1" and "file1" refer to different files.

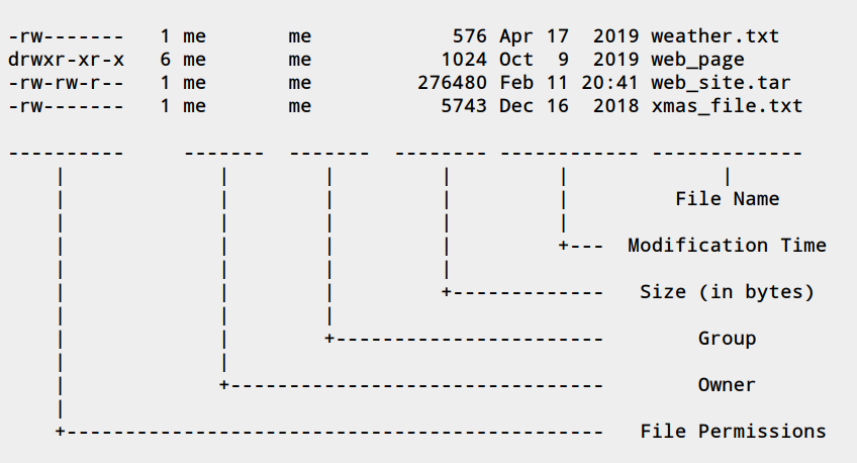
3. Linux has no concept of a "file extension" like Windows systems. You may name files any way you like. However, while Linux itself does not care about file extensions, many application programs do.

4. Though Linux supports long file names which may contain embedded spaces and punctuation characters, limit the punctuation characters to period, dash, and underscore. Most importantly, do not embed spaces in file names. If

you want to represent spaces between words in a file name, use underscore characters. You will thank yourself later.

1.3.3 Looking around

A closer look at the long format: ls -l



- File name: Name of file or directory
- Modification time: Last time the file was modifified
- Size: size of the file in bytes
- Group: Group that has file permissions other than the file owner
- Owner: The user that owns the file.
- File Permissions: A representaion of the file's access permissions. The first character is the file type, "-" indicates a regular or ordinary file. A "d" indicates a directory. The second character represents the read, write and execution rights of the file owner. The third represents the rights of the file's group. The last represents the permissions granted to everyone else.

Directory	Description
/	The root directory where the file system begins.
/boot	Linux kernel and bootloader directory. The kernel is a file called vmlinuz.
/etc	Configuration files, important locations include: /etc/passwd - user info, /etc/fstab - mounted devices(disk drives), /etc/hosts - Network host names and IP adresses, / - system service scripts run at boot time.
/bin, /usr/bin	Most of the system programs.
/usr	folders/files that support user applications
/usr/local	Used for installation of software (user). Most commononly in /usr/local/bin
/var	Files that change as the system runs, including logs and spools.
/lib	The shared libraries
home	user files

The list goes on and on, do research to find specific files.

1.3.4 Manipulating Files

Commands to know/understand:

1. cp: copy files and directories
2. mv: move or rename files and directories
3. rm: remove files and directories
4. mkdir: make directories

Wildcards: Wildcards allow the user to specify groups of filenames. This makes mass file manipulation much easier.

Wildcard	Meaning
*	Matches any characters
?	Matches any single character
characters	Matches any character that is part of the set characters.
!characters	Matches any character that is not a member of the set characters

Here are some examples of patterns and what they match.

Pattern	Matches
*	All filenames
g*	All filenames that begin with the character ”g”
b*.txt	All filenames that begin with ”b” and end with ”.txt”
Data???	Any filename that begins with the characters ”data” followed by any other characters.

1.4 Working with commands

Commands to know/understand

1. type: Display information about command type
2. which: Locate a command
3. help: Display reference page for shell builtin
4. man: Display an on-line command reference

What are ”Commands”

Commands fall into 4 categories:

1. Executable programs: programs that can be executed
2. A command built into the shell: or ”shell builtins. Like, ”cd”
3. A shell function: Miniature shell scripts.
4. An alias: Commands that the user defines, built from other commands.

Chapter 2

Shell permissions

2.1 Resources

1. Permissions

man or help pages:

1. chmod - modify file access rights
2. sudo - enter super user mode or execute a command as such
3. su - temporarily enter sudo mode
4. chown - change file ownership
5. chgrp - change a file's group ownership
6. id - print effective user and group IDs
7. groups - display current group names
8. whoami - print effective username
9. adduser - ?
10. useradd - create a new user or update default new user info
11. addgroup - ?

2.2 Permissions Notes

Each file/directory is assigned access rights for the owner of the file, members of a group of related users and everybody outside of the afore-mentioned groups.

Rights can be assigned to read, write and execute a file.

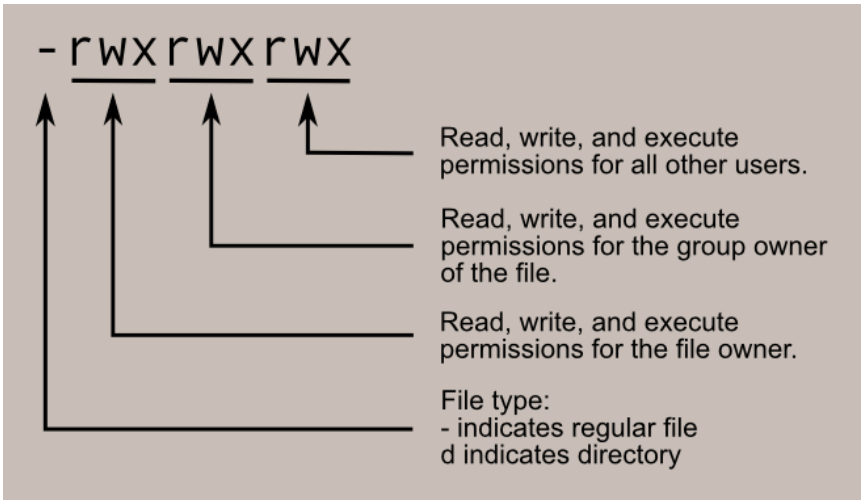
As an example the ls command will be used to look at the 'bash' program located in the /bin directory:

```
~ ✗  ls -l /bin/bash
-rwxr-xr-x 1 root root 1071664 Feb  2 08:38 /bin/bash
```

Here we can see:

1. The file "/bin/bash" is owned by user 'root'
2. The superuser has the right to read, write and execute
3. The file is owned by the group "root"
4. Members of the group 'root' can also read and execute
5. Everybody else can read and execute the file

Below is graphic showing what each portion of the first listing represents:



2.3 Look at all the pretty commands

2.3.1 chmod

Used to change the permissions of a file or directory. To use it specify the desired permission settings and the file or files that are to be modified.
There are two ways to do this, the following method uses the octal notation method.

Think of the permission settings as a series of bits (like a computer):

```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000

and so on...

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

No we can represent each of the three sets of permissions, (owner, group and other) as a single digit to create a convenient way of expressions the permission settings.

For example to set the permissions of a file to have read and write permission for the owner, but wanted to keep the file private from others we would use:

```
~ ❯  chmod 600 some_file
```

File Permissions:
Below is a table of common settings for files, the ones starting with '7' are used with programs since they enable execution. The rest are other kinds of files:

1. 777 - (rwxrwxrwx) No restrictions on permissions, All groups can do everythin. (BE WARNED)
2. 755 - (rwxr-xr-x) File owner can read, write and exec, All others may read and exec.
3. 700 - (rwx——) File woner can read, write and exec. Nobody else can do anything.
4. 666 - (rw-rw-rw-) All users may read and write the file.
5. 644 - (rw-r–r–) File owner may read and write, others can only read.
6. 600 - (rw——-) Owner can read and write a file. All other have no rights.

Directory Permissions:
The chmod command can also be used to change directory permissions.
In this case octal notation is also used but the r, w and x meanings are different:

1. r - Allows the contents of the directory to be listed if the x attribute is also set
2. w - Allows files within the directory to be created, deleted or renamed if the x attribute is also set
3. x - Allows a directory to be entered (cd dir)

Here are some common settings for directories:

1. 777 - (rwxrwxrwx) No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting.
2. 755 - (rwxr-xr-x) The directory owner has full access. All others may list the directory, but cannot create files nor delete them.
3. 700 - (rwx——) The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must keep private.

2.3.2 chown

Change file ownership.
Example:
Change the owner of some_file from ”me” to ”you”:

2.3.3 chgrp

Change the group ownership of a file or directory.
Example: