Dal CSS Geek Labs

Introduction to the Linux Command Line Interface

(Presented Oct. 18, 2016)

Contact information: Corrie Watt (CR535115@dal.ca)

> "Graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible."
>
> *Original source unknown, quoted in Shotts, W. E. (2012),* ***The Linux Command Line****.*

The shell is a *program* in your Linux distribution which takes keyboard commands and passes them to the operating system to perform. In the good old days , it was the only user interface available.

By learning the text-mode tools, you'll be able to use Linux's true power, enabling you to to do simple or complex things very quickly. It will also be a useful thing to know if your Linux or OS-X GUI system ever fails. And, if you ever have to do something with *system* files, there is no better friend.

Resources:

1. https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal *...Part 1 of a very good series of tutorials for new Linux users.*
2. http://vic.gedris.org/Manual-ShellIntro/1.2/ShellIntro.pdf *...BASH introduction and cheat sheet; download and read (it lists DOS equivalents to Linux commands).*
3. *https://www.digitalocean.com/community/tutorials/basic-linux-navigation-and-file-management*
4. http://www.howtogeek.com/117435/htg-explains-the-linux-directory-structure-explained *...This is an excellent look at the Unix/Linux file system; gives comparison with Windows file system.*
5. http://www.howtogeek.com/125157/8-deadly-commands-you-should-never-run-on-linux/ *...No explanation needed; power is wonderful, and Linux gives you absolute power!*
6. https://www.digitalocean.com/community/tutorials/top-10-linux-easter-eggs *...If you have spare time and access to a Linux system: Enjoy!*

About shell commands:

A shell command can be...

1. An *executable program* (like the files in `/usr/bin`): It can be a shellscript, a compiled program (in C or C++), or a script in Perl, Python, Ruby, etc.

2. A *shell function* (miniature shell scripts that are part of the shell environment)

3. A *command* built into the shell (ls, cd, cp, mkdir ...)

4. An *alias* – a command we can define in a special file, which we will not cover here, but feel free to check it out.

Exercise 1. Starting a shell

A Linux shell is *a program* (we will be using the BASH shell, available in Linux and OS-X).

- Start the terminal emulator (this program opens a window to let you interact with the shell). A shell prompt will appear:

  me@linuxbox: ~$ *...this is your* ***command line***

- The format of a CLI command: **command -option filename**
  Type in some stuff, press *Enter*:

  me@linuxbox: ~$ WTF

  bash: WTF: command not found

  me@linuxbox: ~$ *...A new command prompt is returned to you*
  Your shell could not find the command `WTF`, *because it does not exist.*

  Examples:

  - Display current time and date or this month's calendar:
    me@linuxbox: ~$ date
    me@linuxbox: ~$ cal
  - See the current amount of free space on your disk drives:
    me@linuxbox: ~$ df
  - See amount of free memory:
    me@linuxbox: ~$ free
  - Ending a terminal sesssion:
    me@linuxbox ~$ exit ...*Or type in:* 'Control'+D

Exercise 2. Getting around on the CLI

- For file and directory names, type in the start of a name, then press ***Tab*** to ***auto-complete***. (*If you hear a beep, press TAB again: a list of possible completions is returned. Type in more characters of your filename untll it becomes 'the only choice' left. Press TAB again.*)
- Use the ***up and down arrow keys*** to scroll through previous commands, stop on the command you want (edit it, if you like), press 'Enter' to run it.
- Use the ***arrow keys*** to move the cursor (to go forward or backward in a line of text).
- ***Copy-Paste*** with your mouse usually works (do not copy a blank space at the end of your text string—the 'Enter' key will then be assumed to have been pressed).
- Pressing ***Enter*** will run the command you just typed in.

Exercise 3. Getting around in a Unix-style file system

Linux stores programs in several locations, including **/bin , /usr/bin , and /usr/local/bin**. (The programs used mainly by *root* appear in /sbin , /usr/sbin and /usr/local/sbin as well).
If an *executable program* appears in one of these directories (which make up the path), you can run it by typing its name.

- ***Files*** in Unix/Linux are hierarchical (i.e. in an upside-down tree structure)
- A file may contain data in some format (with or without a file extension label) or it may contain other files, making it a ***directory***.
- Directories are files that have files in them: sub-directories and data files (eg. Downloads, home, Video, Documents, books, ...).
- Mounted media (such as a USB stick or an external hard drive) are treated as normal files.
- You can use the PDF from Resource #1 to help you in this section.

1. We need to know what directory we are currently in, so that we can change our working directory to where we want to be:

> **me@linuxbox: ~$ pwd** *...Print my current working directory*
> **/home/me** *...This is your current working directory*
> **me@linuxbox: ~$** *...followed by a new command prompt when done*

2. Make three new directories in your home directory, using the **mkdir** command.

Example:

> **$ mkdir my-directory1 my-directory2 my-superdirectory**
> *...Make several new directories at once*

3. Change location into one of the new directories, using the **cd** command, and make two or more new files in the current directory using the **touch** command; give them valid Linux filenames.

> *What makes a Linux file name valid?*
> *Why does the validity of the file name matter?*

Example :

> **$ cd my-directory1**
> **$ touch myfile1**

4. Check if these new files now exist; use the **ls** command to do this.

4-a. Next, use **ls** with *options*: what else can you find out about the files this way? (See Resource #2 above.)

5. Copy *one* of the files to a new file; use the **cp** command. Use **ls** to check if the new file exists.

Example:

> **$ cp file1 file2**

What would happen if you did a **$ mv file1 file2** *instead of* **cp***? **Try it** (use* **ls** *to see the result).*

6. Copy a file from the current directory into one of your other directories. You will need to tell the shell what file to move from where *to* where (i.e., provide a **file path**).

7. Add content to a file using the **cat >** (catenate into) command.

Example: **$ cat > my_file_1.txt** *...The* **>** *tells the machine to open the named file, and put what you are going to type below into the file.*

a. Type some text.
b. End with a new line (by pressing the Enter key).
c. Press Control-C to close the file.
d. Execute the command **$ cat my_file_1.txt.** *What is the result?*
e. What does the **cat >>** command do?

8. Deleting files using the **rm, rm -i, or rm -r** commands

What does the **-i** do? The **-i** means you will be prompted for consent before executing the command for each of the files in turn. Use it when deleting multiple files, wild-carded files (see `rm -i *.jpg),` or if you want to be careful!

Examples:

> **$ rm thisfile** *...Deletes* `'thisfile'` *from the current directory.*
> **$ rm -r directory** *...What does the* **-r** *do?*
> **$ rm -i *.jpg** *...Lets you decide whether or not to delete this image file*

(.jpg) as the shell gets to it; without the **-i**, *all your images in the current directory will be deleted (as in: gone, all totally gone!).*

Exercise 4. Odds and Ends

1. What *type* of command am I using?

What does ***command type*** mean? Where can you look it up?

Use this shell built-in, ***type****, to find out what type your command is.*

Examples:

```
me@linuxbox: ~  type type
type is a shell builtin

me@linuxbox: ~  type cp
cp is /bin/cp

me@linuxbox: ~ type ls
ls is aliased to 'ls –color=tty'
```

2. Wildcard character (*) as used in the shell:

me@linuxbox: ~$ echo * ... the wildcard character * means "match any characters in a file name". *The shell expands the * into something else before the echo command is executed. Below we see a display of the contents of our current working directory.*

*Example: (*This is only an example to show ***wildcard expansion*** *(*);* for actually doing this sort of listing, do not use **echo**, use **ls**!)

```
me@linuxbox: ~$ echo *
Documents  Downloads  medusa1.php  output.txt  myBooks
```

...lists the files (in this case, we have only directories) in this home directory (the tilde, **~**, *stands for my* home directory*)*

```
me@linuxbox: ~$ cd Documents/
me@linuxbox:~/Documents$ echo *
ASH.pdf  Assignment4.txt  bash_notes.odt  BNK_chapterss.odt
ButtonGrid.java Daily-Weekly-Review-Checklist.doc  Enc1.odt
fallSchedule.png feb_reviewNotes.odt
```

...lists the files (files and directories!) in this directory in alphabetical order

3. The Help Files

-Bash has built-in help for each shell **builtin**. Type in '**help**' plus the command.

Example:

```
me@linuxbox: ~$ help cd
```

Result:

```
cd: cd [-L|[-P [-e]]] [dir]
Change the shell working directory.
Change the current directory to DIR. The default DIR
is the value of
the HOME shell variable.
```

...and it goes on and on, with a list of command options and path information. Optional items (options) are in square brackets; a vertical bar indicates mutually exclusive options.

-**Help** for **executables**:

Executable programs that are usable from the command line, such as **`mkdir`** or **`chmod`**, usually have a manual ('man page') available. The program **`man`** will display the man page, which is broken into sections 1-8, where section 1 contains the user commands. It lets you scroll through the whole length of it. (Usually, **man** uses **less** to display the 'man' pages).

Try it:

Type in **`man mkdir`** ...type **h** for Help or **q** to quit (get out of the man page, back to your command prompt)--**h** will open a help file with a nice summary of all the **`less`** commands, which you can use to move around in man page!

4. Command Line History: *Use it to work faster, retrieve a complex command, etc.*

See the included Cheat Sheet.

The Bash history function runs behind the scenes, adding each command you run into the equivalent of a linear database. You can then scroll through it using the up and down arrow keys, or display it in its entirety by typing in **`history`** at the command prompt.

By default it saves your last 500 commands.
It is great for finding a complex command that you would rather not retype!

Example:

`me@linuxbox: ~$ history` *...and now a long list of commands will appear on the screen!*

To look for a *specific command* in your history, use ***search mode***:

`$ history | grep ssh` *...This will look for instances of the* **`ssh`** *command in your history.* Instead of `ssh`, of course, you type in the name of the command you want to find.

OR *use* ***reverse-search mode****:*

Press **Ctrl+R,** and type **`ssh`** (or your command). Ctrl+R will start search from the most recent command to the next older one (reverse-search). If you have more than one command which starts with ssh, press Ctrl+R again, and again, until you find the match.

Example:
Press **Ctrl+R**, and the first match shows up, as below. Press Ctrl+R again, and your previous use of the command is printed to the screen. Do this until you find the instance you are looking for.
`(reverse-i-search)`ssh': ssh freds_machine`

5. **Less** is More

The **`less`** program was created to replace the older *paging* program called **`more`**.

`me@linuxbox: ~$ less` *filename*

The **`less`** command launches a program to view *text files*. Many of the files that contain system settings (*configuration files*) are stored in text format. Many of the programs the system uses (*scripts*) are also stored in this format. Being able to read them gives us a look at how the system works.

Example:

`me@linuxbox: ~$ less /etc/passwd`

Once the **less** program starts, we can look at the contents of the file; if the file is longer than one page, we can scroll up and down. To exit **`less`**, press the **Q** key (for quit). *For the* **`less`** *control commands, see:* http://ss64.com/bash/less.html.

Cheat Sheet: Bash editing and command history features

Keystroke	***Effect***
Up arrow	Retrieves the previous entry from the command history.
Down arrow	Retrieves an earlier entry bypassed when using the up arrow.
Left arrow	Moves the cursor left one character.
Right arrow	Moves the cursor right one character.
Ctrl+A	Moves the cursor to the start of the line.
Ctrl+E	Moves the cursor to the end of the line.
Delete key	Deletes the character under the cursor.
Backspace key	Deletes the character to the left of the cursor.
Ctrl+T	Swaps the character under the cursor with the one to the left of the cursor.
Ctrl+X and then Ctrl+E	Launches a full-fledged editor on the current command line.
Ctrl+R	Searches for a command. Type a few characters, and the shell will locate the latest command to include those characters. You can search for the next-most-recent command to include those characters by pressing Ctrl+R again.

Part 2: **Linux Files and Directories: Authorization and Access Control**

Resources:

1. https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-permissions
2. https://www.digitalocean.com/community/tutorials/how-to-understand-the-filesystem-layout-in-a-linux-vps *...Getting around in the Linux filesystem; an excellent review for if you want to feel more at home in Linux.*

About User files and System files:

Linux is a multiuser OS. This means a single computer can host thousands of users. Such a computer might be a mainframe at Dal or at a business, or a cloud-computing server. Most of this computer's users will be unfamiliar with the details of Linux system administration—they just want to use their word processors, email clients, and other applications. These users have no need to deal with system configuration files. In fact, giving them access to such files—especially write access—could have very unpleasant consequences. Some system files, such as the `/etc/shadow` file, should not even allow read access to regular users. *Why should regular users be kept out of that file?*

On a computer with many users, each of whom can change the system's configuration, somebody will make a change (accidentally or not) that will bring down the computer! Linux guards against this sort of catastrophe by enforcing strict file and directory permissions, which allow only those authorized to do so to view or alter a file.

Ordinary users are unable to write to most system files; this protects these files from being played with! Only a superuser, **root,** has ALL the powers (to become root , use the command **su**—or **sudo**–with a root password). Because `root` can read and write to any file, you must acquire root privileges to perform most *system maintenance* tasks.

Your user files are also protected: unless you give specific permissions, only you (not the other users on the computer) can access and change your personal files. You can delete only your own files—or, more accurately, you can delete files only if you have *write permission* to the *directories* that contain them.

File ownership

Files have owners. You can change a file's ownership by using a GUI file manager or a text-mode shell (e.g. Bash). Running programs also have owners. Most programs you might run are tied to the account (by the account's user-ID number) that you use to launch them: this fact, together with the file's ownership and permissions, determines if the program may modify, or even read, a particular file.

File permissions can be set independently for the file's owner, the file's group, and the 'other' computer users. File owners can change the permissions and the group (but only to a group the owner already belongs to). Directories (files that contain files) also have owners and groups.

Fun fact: As *root*, you can change the owner and group of any file you want.

To see who owns a file, and what permissions are associated with it, do an **`ls -l`** :

Example:

```
$ ls -l secret_file
-rwxr-xr-x   1   corrie users     169681   Jan  7 14:02  secret_file
permissions  links  owner  group    file size   time stamp      file name
```

Setting file ownership in a shell (Bash)

To change a file's ownership, as *root* (**`su`** or **`sudo`** to become root), use the command **`chown.`** The command, in the basic form, needs a username and the name of a file:

```
$  chown superman secret_file
```

The file will now be owned by superman. (Check the result by using **`ls -l`**)

About file permissions

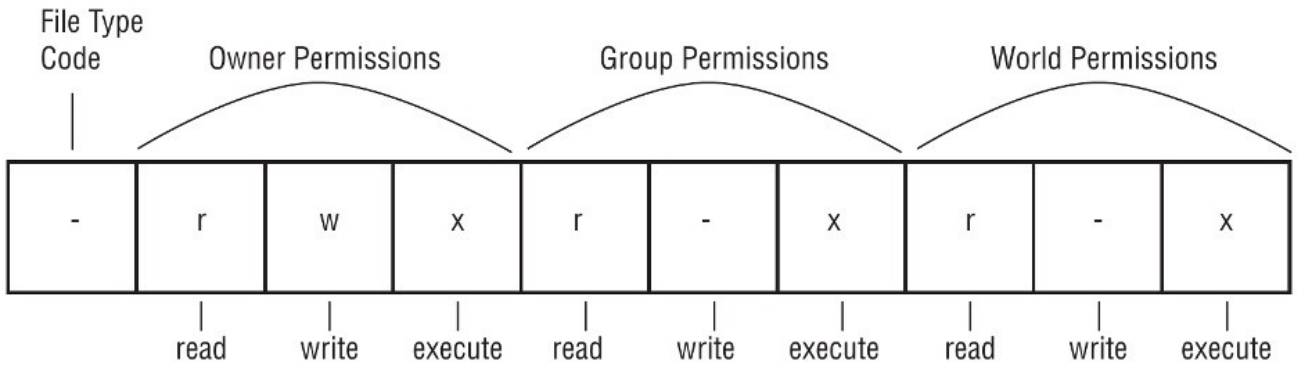

Image source: Bresnahan, C., Blum, R. (2015). *Linux Essentials, Second Edition*. Sybex.

Owner permissions*:* These tell us what the file's owner can do with the file.

Group permissions*:* These determine what members of the file's group (who are not the file owner) can do with the file.

Other ("the world") permissions: These determine what others, who are not the file's owner or in the file's group, can do with the file.

-**Read and write** permissions are simple to understand.
-**Execute** permission means the file can be run as a program, or, in the case of a directory, can be searched (looked inside of).
-Permissions can be written in two forms: as an octal number or in a symbolic form (rwxr- -r- -).

Directories, even though they are files, are treated in a special way: if a user can *write* to a directory, she can create, delete or rename files inside the directory, even if she is not the owner and does not have permission to write to those files.

Look up how permisssions can be specified in octal code. How would you write the following as an octal representation: **`rwxr-xr-x`**? What permissions does octal code 750 represent?

Setting file permissions

File permissions can be set through some GUI interfaces, but here we will use the Bash shell program to do this. We will use the command chmod to change permissions.

Example:

```
 -rw-r--r-- 1 corrie corrie 0 Jan 11 12:47 fakefile.txt
$ chmod g+w fakefile.txt   ...change the permissions—add group write permission
$ ls -l                    ...list the file information to see what changed
-rw-rw-r-- 1 corrie corrie 0 Jan 11 12:47 fakefile.txt   ...It worked!
```

How to specify permissions:

1. Indicate the permission set that you want to modify—
u for the user (that is, the owner), **g** for the group, **o** for other users, and **a** for all permissions.

2. Include a symbol indicating whether you want to add (**+**), delete (**-**), or set the mode equal to (**=**) the stated value.

3. Add code specifying what the permission should be, such as the **r** , **w** , or **x** symbols.

Example:

Permissions before chmod: **`r--------`** *...user can only read the file*
`chmod ug=rw report.txt` *... set the user + group permissions to: read, write*

Result: After the chmod we have **`rw-rw----`**

Practice Exercises

Exercise 1.

a) Create a file with **touch** (or some other program) and then practice copying it with **cp**, rename it with **mv**, move it to another directory with **mv**, and delete it with **rm**.

b) Use **ls -l** to determine the default permissions your system sets on a file you create.

Exercise 2.

Create a directory with **mkdir,** and then practice using **cp**, **mv**, and **rm** on it, just as with files.
Try copying files into it and then try deleting the directory with both **rmdir** and **rm**.
Do both commands work?
How can you delete a directory and its contained files?

Exercise 3.

As root , copy a file that you created as an ordinary user, placing the copy in your ordinary user home directory.
Using your normal account, try to edit the file with a text editor and save your changes. What happens?
Try to delete that file with the **rm** command. What happens?

Exercise 4.

Create a test file as an ordinary user.
As root , use **chown** and **chmod** to experiment with different types of ownership and permissions, to discover when you can read and write the file by using your normal login account (i.e. not as root or superuser).

Exercise 5.

Use the **ls -l** command to view the ownership and permissions of files in your home directory, in **/usr/bin** (where many program files reside), and in **/etc** (where most system configuration files reside).
What are the implications of the different ownership and permissions you see for who can read, write, and execute these files?