Workshop 12: The Linux Command Line

This workshop teaches the very basics of using the Linux Command Line Interface. (CLI)

Difficulty: Introductory workshop

Contents

0 Introduction

On a Linux computer the desktop only scratches the surface of the programs and functionality available. Linux users often access these additional features through a terminal window, using the Command Line Interface (**CLI**), a non-graphical user interface.

For many tasks, creative use of command-line commands is much faster than their GUI counterparts, especially in bulk tasks. We'll only be able to take a look at the very basics today, but Linux geeks swear by the fact that each new trick they learn in the CLI allows them to save a little more time when using their computer.

This is an introductory workshop. Once you're familiar with using your Pi on the desktop, you should be ready to attempt this workshop.

How to use these booklets

The aim of these booklets is to help you attempt these workshops at home, and to explain concepts in more detail than at the workshop. You don't need to refer to use these booklets during the workshop, but you can if you'd like to.

When you need to type a command in the command line interface, they will be listed like the following example. Copy everything *after* the dollar sign. Lines without dollar signs are example outputs, and do not need to be copied.

```
1 $ echo Hello, World!
2 Hello, World!
```

Occasionally, something will be explained in greater detail in asides, like the one below. You can read these as you wish, but they're not required to complete the workshop.

The Shell

The command line interface isn't really a built-in part of the Linux operating system. It's a type of program called a 'shell', that runs on top of the operating system.

Raspbian uses 'bash' (**B**ourne-**A**gain **SH**ell), which is the most typical shell in Linux distributions, but you could choose to install an alternative shell.

What you'll need

For this workshop, you'll just need a working Raspberry Pi running a Linux operating system like Raspbian.

Everything else

These booklets were created using LaTeX, an advanced typesetting system used for several sorts of books, academic reports and letters.

To allow modification and redistribution of these booklets, they are distributed under the CC BY-SA 4.0 License. LaTeX source documents are available at github.com/McrRaspJam/booklet-wo

If you get stuck, find errors or have feedback about these booklets, please email me at: jam@jackjkelly.com

1 Starting at the Deep End

The Linux command line is usually accessed through one of two ways.

1. By opening a terminal window from the desktop.
2. By running the operating system in the command line mode.

When we use the latter method, the desktop environment won't start at all. We'll set our Pi's into this mode now, so we won't feel tempted to stray outside of the command line for this workshop.

Why use the Command Line?

Other than as a fun challenge, running a machine without a desktop environment has some benefits.

Imagine a server in a large server-bank. These machines run automatically, and are only accessed by humans for maintenance purposes. So most of the time, running a graphical desktop would waste resources.

1.1 Enable boot to Console

Open a terminal window and type the following

```
$ sudo raspi-config
```

a bright blue menu should appear. Navigate to the following options using the arrow keys and enter.

- 3 Boot Options
  then
- B1 Desktop/CLI
  then
- B1 Console

Then navigate to Finish. You will be then asked if you wish to restart. Select Yes.

1.2 Login

If you joined the Raspberry Pi community recently, you'll have always had the Raspberry Pi boot to desktop automatically.

We've now disabled this behaviour, and when we reboot our Pi, you'll be greeted with a login screen. The default Raspbian login credentials are as follows:

username: pi
password: raspberry

Take care when typing your password, it won't appear on screen as you type it, not even as asterisks or dots.

The hidden password

This is a security feature to hide the length of your password from onlookers, as well as the password itself.

Knowing the length of a password would make guessing or brute-forcing a passworded lock considerably faster.

2 Directories

2.1 Where am I?

Once you've logged in, you should see a line that looks like this.

```
pi@raspberrypi:~$
```

This is your command prompt. Let's break up that line.

- The first part is the current user. We logged in as **pi**, so that appears here.
- Following the @ is the hostname, or the 'machine name'. **raspberrypi** is the default hostname in Raspbian.
- Following the colon, your current director is shown. The tilde character ~ is a shortcut for your home folder.
- **$** is the current prompt. When this is the last thing printed, the terminal is ready to receive an instruction.

Let's confirm that the tilde is pointing at our home folder. To find the full address to our current directory, we can use the **pwd** command, which stands for 'print working directory'.

```
pi@raspberrypi:~$ pwd
/home/pi
```

2.2 Moving around

So, we're currently in our home directory, and there are two levels above us. to move to the root folder, we can use the **cd** command, which stands for 'change directory'.

```
pi@raspberrypi:~$ cd ..
pi@raspberrypi:/home $ cd ..
pi@raspberrypi:/ $ pwd
/
```

.. causes us to move up a directory. to move into a directory, we can type its name.

```
pi@raspberrypi:/ $ cd usr
pi@raspberrypi:/usr $ pwd
/usr
```

We can type an absolute directory starting with '/', the root folder.

```
pi@raspberrypi:/usr $ cd /media
pi@raspberrypi:/media $ pwd
/media
```

Using the cd command without a directory or .. takes us to the home folder. We can string together directory changes using forward slashes.

```
pi@raspberrypi:/media $ cd
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ cd ../../usr
pi@raspberrypi:/usr $ pwd
/usr
```

Now have a go at finding at finding the location of the Python 3 binary executable. You can use the **whereis** command to find the location of a program.

```
$ whereis python3
```

The first part of the output will be the directory of the binary file.

3 Files

Once you've picked a directory, you can use the **ls** command, which is short for the word list.

```
 $ cd
~$ ls
Desktop Downloads Pictures python_games
```

Once we've found the file we're looking for, it's straightforward to edit it. For example, a text file could be opened using

```
~$ nano textfile.txt
```

If a file doesn't exist, most editor programs will create a new file with that name. Save this new file by pressing **Ctrl+O**, then quitting with **Ctrl+X**. Run ls again, and your file should now be present.

3.1 Manipulating Files

Let's blitz through a few file operation commands.

To copy a file:

```
~$ cp textfile.txt anothertextfile.txt
```

To move a file:

```
~$ cp textfile.txt originaltextfile.txt
```

You'll note that both of these commands use the same parameters, the original file followed by the resulting file.

To remove a file:

```
~$ rm anothertextfile.txt
```

To make a directory:

```
~$ mkdir textfile
```

We should be left with originaltextfile.txt and a directory. Try *moving* the text file to within this new directory.

you can remove an empty directory with rmdir, but because ours has a file in it, we will delete it using the following command.

```
~$ rm -r textfile
```

-r is flag that tells rm to run recursively. There are lots of flags for most commands, for example try **ls -l**. you can view the flags of a program by viewing it's manual using the **man** command.

```
~$ man ls
```

4 Using apt-get

<Written explanation to be added after the workshop>

5 The Ultimate Command of Destruction

The Linux command line places a lot of trust in you. As far as it's aware, you're somebody who knows what your doing. If you don't know what you're doing, be careful.

We're going to have a look at some useful features of the command line, and see what happens when you combine them incorrectly.

-f

-f stands for force, and on a command like rm skips any warning messages that would usually be created.

if we were trying to delete 1000 files which were set as read-only, we would usually have to answer a yes/no check on every single file. -f will stop that from happening, so the command would finish instantly.

sudo

sudo stands for superuser do. Any time we want to complete a command that requires a high level of authority, such as modifying the read-only files mentioned above, we run that command as a superuser using sudo.

This command effectively means "I know what I'm doing", so make sure you do!

Wildcards

In the CLI, the asterisk character * is used as a wildcard. If we wanted to delete all the .txt files from a directory filled with thousands of different files, we could find each one and manually type in

```
$ rm file23of9000.txt
$ rm file72of9000.txt
$ rm file73of9000.txt
$ rm file79of9000.txt
...
```

or, using a wildcard, we can search for a partial name. Entering:

```
$ rm *.txt
```

would delete all of the files ending in '.txt', just like we wanted. Likewise, to delete all the files beginning with January, we could do:

```
$ rm january*
```

But what if you tried a command such as 'rm *'? By default it would stop within the current folder, and possibly the sub folders, but what if we accidentally told it not to?

5.1 Making the wrong combination

The truly scary thing about the ultimate command of destruction is how inconspicuous it looks.

```
$ sudo rm -rf /
```

Let's break the command down.

rm is our standard remove file function. To it, we have supplied the flags **-rf**. -f means we know what we're doing, and -r means it will start at one point and work through every subdirectory it finds.

This is fine if we're trying to delete a Java Project folder, or a collection of cat pictures, but what directory have we supplied?

/ is the root directory, the directory under which every single file on our computer is contained.

The Operating system would have stopped you, but we've ran it under **sudo**, so the computer's convinced we done this intentionally.

And now (if you're happy to wipe whatever SD card you're running on) you can try it on your Raspberry Pi.[1] I'd recommend adding the flag -v for dramatic effect, which will display the files as they're being deleted.

```
$ sudo rm -rfv /
```

6 The End

Whilst it certainly is for your Raspberry Pi (until you reinstall NOOBS, that is), I thought I'd leave you with one story about accidental misuse of rm.

This story about a quite famous film company is available as a video at https://youtu.be/8dhp_20j0Ys, and a text article is available at http://bit.ly/2jgR3nC.

Try not to be too intimidated by what you've just accomplished Getting you to destroy your Pi in a single command wasn't meant to scare you away from using the command line, just to show you the power that it gifts you.

[1] A feature of modern Linux distributions, at this point the program throws one last warning at you to try and stop you.
If you're really *really* happy to lose all of your files, just add the flag it tells you to, and it *will* finally run your command.