Operating Systems

Practical 2

Learning the Linux Command-Line

Dr. Kevin Farrell
February 2025

This page intentionally left blank

Table of Contents

Legal

1 http://creativecommons.org/
2 http://creativecommons.org/licenses/by-nc-nd/3.0/

1. Aims and Objectives

The aims and objectives of this practical are as follows:

1. To revise/learn basic use of the Linux command line, including:
   a) Reading about command structure in Linux/UNIX
   b) Listing files and directories
   c) Navigating around the Linux/UNIX filesystem
   d) Writing a simple shell script
   e) Manipulating files (moving, copying, looking at them)
   f) Looking up the manual page
2. Please use a paper notebook and pen to record your work.
3. To answer a quiz as a Reflective Activity

2. Commands in Linux/UNIX

2.1. Before you start, please make sure you have a (paper) notebook and pen to record your work. You will be permitted to use your ***handwritten*** material to assist you with the “Reflective” MCQ Quiz at the end of the practical. However, you will ***not*** be permitted to use the Internet or any electronic devices.

2.2. Boot your Mageia Linux Virtual Machine (VM), and ensure that you have Internet access on it. Since the default network settings use NAT on your VM, there is no need to configure a Wi-Fi connection – the VM simply uses the Internet Connection of your host (Windows OS). Ask for assistance if necessary!

2.3. Now run Firefox from *within* your VM to access BrightSpace, and open this handout on Linux instead.

2.4. Use Right-CTRL-F to enter full-screen mode. Right-CTRL is the Control key on the right-hand side of the keyboard. It is referred to as the *host* key. If you don’t have a Right-CTRL key, you can set a host key on VirtualBox by going to:

> **File → Preferences → Input → Virtual Machine (tab) → Host Key Combination → Select Right Ctrl →** then tap for example the **Alt Gr** key on your keyboard instead, which will make it the host key.

2.5. As you are probably aware, a command in Linux/UNIX is a sequence of characters, which you type at a “command prompt” in a “terminal window” often called the **console.** When you hit the <RETURN> key, the command then performs some action.

2.6. In Mageia 8 Linux, we use the Plasma Desktop Environment, sometimes referred to as **KDE Plasma**. KDE used to stand for **Kool Desktop Environment**. Because of this, many applications written for this Desktop Environment begin with the letter ‘K’. So, the console *application* in your Mageia 8 Linux installation is referred to as **`Konsole`**.

2.7. When you run the **`Konsole`** application, it opens a terminal window, and presents a prompt. What you're actually running is a Linux **shell** program. The shell interprets the commands you type; hence, the term **'Command Interpreter'**, which you may have encountered before.

2.8. In Mageia 8 Linux, the default shell is called **`bash`**. This stands for **Bourne Again Shell**, and is a pun on the original UNIX shell, from many years ago, called the **Bourne Shell**. The **`bash`** shell is a re-write and improved version of the Bourne Shell, **`sh`**. There are actually many different shells; some examples are: **`csh`**, **`tcsh`**, **`zsh`**, **`ash`**, **`ksh`**. The main difference between one shell and another is in the syntax used for writing scripts, and for a small number of specialised commands. But for a beginner user, they essentially look the same.

2.9. When you run the **`Konsole`** application, the shell places you in *your* home directory. Note: A **directory** in Linux is the same as a **folder** in MS Windows/MacOS.

2.10. Your home directory has a **path** or location: **`/home/username`**, where **`username`** is your user account name; for e.g.: **`/home/kfarrell`** is user kfarrell's **home** directory

2.11. The structure of a Linux/UNIX command is as follows:

```
$ command [options] [arguments]
```

2.12. In relation to the above, note some things:

1. The "**$**" symbol preceding the command is simply the command prompt. A "**$**" indicates that you should type the command as an **ordinary** user. But, you don't need to type the "**$**" symbol itself. In documentation, the command prompt symbol is used to tell the reader whether they should type the command as an ordinary user or as the System Administrator, also known as the root user, super user, or simply, **root**. You will see the phrase "log in as root" quite a lot in documentation.

2. The command name, the options and arguments are each separated by spaces. That means that a space has a *special meaning*.
3. Options are, by definition, optional. That is, you don't necessarily have to use them, but using options increase a command's functionality.
4. Options, which consist of a single letter, must be preceded by a single dash: "`-`". There is no space between the dash and the option. For e.g.: **`ls -a`**
5. Some options consist of words. These generally must be preceded by a double dash: "`--`". There is no space between the double dashes and the option. For e.g.: **`ls --help`**
6. Several single letter options can be grouped together. These must also preceded by a *single* dash: "`-`". For e.g.: **`ls -lt`**
7. Arguments are parameters without any preceding dashes. They are often **file** names or **directory** names. In Linux/UNIX, these are case-sensitive. That means that you must type the file or directory name ***exactly*** as written – in lowercase or uppercase letters or a mix of these if it is listed as such.
8. Commands in Linux/UNIX are also case-sensitive. In an examination, you ***must*** write the command in the correct case – if you write a command in uppercase letters, when it is supposed to be lowercase, you may lose marks.

2.13. Use the Internet to find out what is the symbol for the **root** prompt in Linux. Write your answer in your notebook.

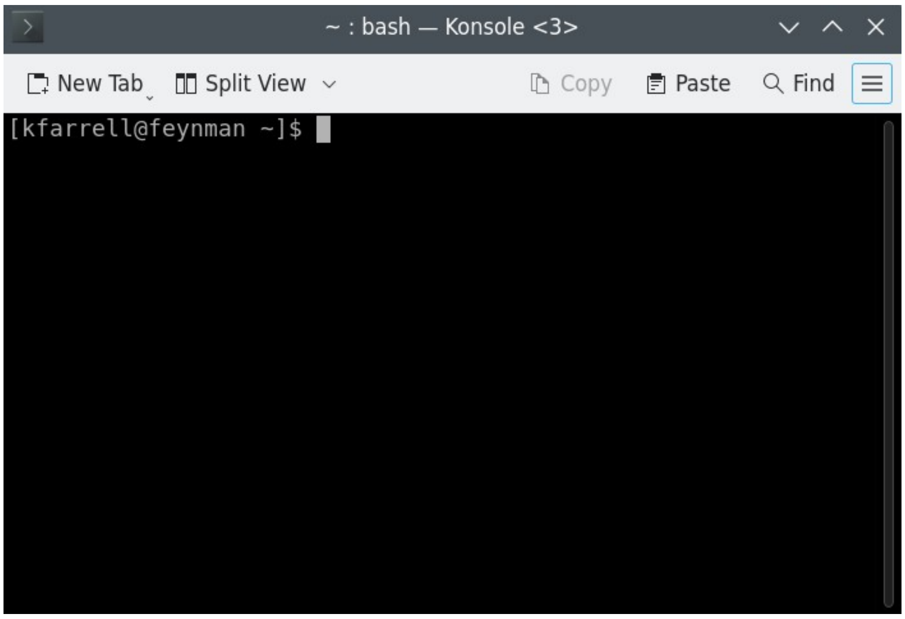
3. Basic Linux command line usage for working with Files and Directories

3.1. Open a terminal window (console) by running the **`Konsole`** program. If you didn't pin it to the bottom Panel in Practical 1, you can find it by clicking on the Mageia Start Button:

→ **`Tools`** → **`Konsole`**

3.2. Right-clicking it will allow you to add it to the Panel – do so now if you didn't complete this in Practical 1. Left-clicking it, will run it.

3.3. You should now see a **`Konsole`** window like mine:



```
[kfarrell@feynman ~]$
```

3.4. Notice the prompt: **[kfarrell@feynman ~]$**, where the elements of it have the following meaning:

1. **`kfarrell`** is the user
2. **`feynman`** is the hostname (PC name)
3. The "**`~`**" (tilda character) indicates that I am in *my* home directory: **`/home/kfarrell`** in this case.
4. The "**`$`**" indicates that I am an **ordinary** user.

3.5. Verify the directory that you are currently in, which we refer to as the **Current Working Directory (CWD)**, by typing the following command:

```
$ pwd
```

3.6. The command **`pwd`** stands for "print working directory". It prints the **absolute path** to your CWD. Write in your notebook, the above command and the output you see.

3.7. Look up on the Internet what an **absolute path** is. Note: an absolute path always begins with a "**/**"

3.8. Now list the contents of your home directory:

```
$ ls
```

3.9. The **`ls`** command stands for "list directory contents".

3.10. Note that "file names" which are coloured blue and followed by a "/" are actually directories. However, note that the "/" is *not* part of the directory name – it is simply a visual "flag" to the user, to tell them that this is a directory. On black and white terminals, where you can't see the colours, this visual flag is useful.

3.11. Write in in your notebook, the above command in 3.8. and the output that you see.

3.12. Create some files in our home directory by typing the following commands:

```
$ touch testfile1

$ touch testfile2
```

3.13. The **`touch`** command does one of two things: if the file doesn't exist, it creates a new file of zero Bytes in size. If the file does exist, it changes the **date stamp** on the file to the current time.

3.14. **Interesting Fact**: time in Linux/UNIX is counted as the number of seconds since 1 January 1970. This is why there was no Y2K bug for Linux and UNIX systems!

3.15. List the contents of the directory again. Notice that ordinary file names are white in colour, and do not have any visual "flag" at the end of them.

3.16. The **`ls`** command has some useful options. Type the following:

```
    $ ls -a
```

3.17. Describe in your notebook, the *differences* between the output from 3.8. and 3.16.

3.18. Is there anything that you can conclude about the files which are *not* visible with the **`ls`** command, but which *are* visible with the **`ls -a`** command? Write your conclusions in your notebook.

3.19. The "**`-a`**" option above stands for "all"; that is, list *all* files.

3.20. We are now going to work with directories. Create some directories in your home directory:

```
    $ mkdir programs

    $ mkdir testdir1

    $ mkdir testdir2
```

3.21. Change into the **`testdir1`** directory, by typing:

```
    $ cd testdir1
```

3.22. The command **`cd`** stands for "change directory".

3.23. Now list the contents using the **`ls`** command with the "**`-a`**" option, as you did earlier.

3.24. You should notice that there are two directories in the listing, one called "**`.`**" and the other "**`..`**"

3.25. These have special meaning, which we describe below. But first, print your working directory again, and write in your notebook the command you typed, and the path that is outputted to the console. Hint: Earlier, you used the command to print working directory. Check your notes and/or this handout.

3.26. You should see the following path outputted:

```
    /home/username/testdir1
```

3.27. If you got different output, or if you're lost, ask your Lecturer for assistance.

3.28. The "`.`" directory here is equivalent to **`/home/username/testdir1`** More on this below!

3.29. Change into the "`.`" directory, by typing the following (note the "`.`" and the space between it and **`cd`**):

```
    $ cd .
```

3.30. Now print your working directory again. Notice how it is the same as before – you haven't changed into a different directory! This is to be expected since the "`.`" directory is just a shorthand way of referring to your Current Working Directory; i.e. the directory that you are already in.

3.31. Change into the "`..`" directory, by typing the following:

```
    $ cd ..
```

3.32. Now print your working directory again. What do you notice? Write in your notebook the effect of the above command.

3.33. Hopefully, it is clear to you that the "`..`" directory is the *parent* of the Current Working Directory (CWD). This is how we navigate, on the command-line, up and down the directory tree.

3.34. In your notebook, draw a diagram in tree-format—a **directory-tree**—representing your home directory and its subdirectories, including the directories which you created. Include the files you also created. If you're unsure what a directory tree should look like, go to **Lecture 01 – Fundamental Concepts** on BrightSpace and look at the slides on the **Filesystem Layout** in **Part II: Overview of Linux**.

4. Writing a simple Shell Script

4.1. In the steps below, you are now going to write a small **bash** script (shell script) that prints “Hello World!” to the console. A shell script is a file that contains a sequence of Linux/UNIX commands that can be run like a program.

4.2. Open the KWrite program: → **Tools** → **KWrite**

4.3. Type the following into KWrite and save it in the **programs** directory as the file **sayhello**:

```
#!/bin/bash

echo “Hello World!”
```

4.4. On the command-line, change into your **programs** directory, and list the contents. If necessary, review the earlier material in this handout and your notebook to jog our memory as to how to do this! Write in your notebook, the commands you just typed and the output you see.

4.5. In order to run your newly-created **sayhello** script, you need to set the permissions on it so that it is executable for the owner of the file (i.e. for you!):

```
$ chmod u+x sayhello
```

4.6. Now list the contents of the directory again. What do you notice? Describe in your notebook, the change in the output that you see. Note that the “*****” following the filename is *not* part of the directory name – it is simply a visual “flag” to the user, to tell them that this is an executable file. On black and white terminals, where you can’t see the colours, this visual flag is useful.

4.7. In 4.5. above, you set the permissions on the **`sayhello`** script file so that it would be executable by you. Access the **Lecture 01 – Fundamentals** folder on the Operating Systems BrightSpace page, and read the document **"UNIX and Linux File Permissions Model"** in the **Supplementary** folder. Also, have a read of the explanations at the website link that I provide in this folder.

4.8. Having read the document and the information at the link, try to work out the **three-digit *octal*** number to represent the same permissions of "**`u+x`**" on the script file in 4.5. above. Write in your notebook, the command you would type to set the *same* permissions as in 4.5. above but using this three-digit octal number instead of "**`u+x`**".

4.9. List the contents of the directory again, but this time, use the **`ls`** command with the "**`-l`**" option. This is the "long" format option. This shows permissions and other information about the files in the directory:

```
$ ls -l
```

4.10. Write in your notebook, the output displayed.

4.11. To run the script, you *could* type:

```
$ /home/username/programs/sayhello
```

4.12. Do this, and write in your notebook, the command you typed and the output you see.

4.13. Clearly, the above command is a bit time-consuming to type every time you want to run the script. There is an easier way! Recall that the "**`.`**" directory is the same as the Current Working Directory. Since you are in your **`programs`** directory, the "**`.`**" directory is the same as **`/home/username/programs`**. So, substituting the "**`.`**" for **`/home/username/programs`** into 4.11. above, gives us the following equivalent command (note the "**`.`**"):

```
    $ ./sayhello
```

4.14. Did you get the same output as in 4.11. ? If not, ask your Lecturer for assistance.

5. Moving, copying and looking at the contents of Files and Directories

5.1. In the following tasks, we list some more useful commands that are commonly used for manipulating files and directories, such as moving them or copying them.

5.2. Copy the **`sayhello`** script file from the programs directory to the **`testdir1`** directory. From the above tasks, your CWD should already be the **`programs`** directory. So, type the following:

```
    $ cp sayhello ../testdir1
```

5.3. The path **`../testdir1`** is known as a **relative path**, because to copy the **`sayhello`** file from the CWD to the **`testdir1`**, we followed a path up the directory tree and down into **`testdir1`**. The path followed is *relative* to our *current* location.

5.4. Change into the **`testdir1`** directory. You should be able to use a relative path to do so! List the contents to check that the copy worked.

5.5. Move the **`sayhello`** script file from the **`testdir1`** directory to the **`testdir2`** directory. From the above task, your CWD should already be the **`testdir1`** directory. So, type the following:

```
    $ mv sayhello ../testdir2
```

5.6. Change into the **`testdir2`** directory, and list the contents to check that the move worked.

5.7. View the contents of the **`sayhello`** script file by typing:

```
$ less sayhello
```

5.8. Hit “q” to quit viewing it, and return to the command prompt.

6. The Linux/UNIX Manual Pages

6.1. Finally, to look up the manual page of a Linux command, simply type:

```
$ man command
```

6.2. For e.g.: To look up the manual page for the `ls` command, type:

```
$ man ls
```

6.3. You can navigate up and down a manual page using the arrow keys or the page-up and page-down keys. You can also search for a keyword using:

```
/keyword
```

6.4. When searching, use “**n**” to look for the next instance of the keyword, and “**N**” to look for the previous instance.

6.5. You might have formed the impression that Linux/UNIX manual pages are quite complex and perhaps difficult to understand. This is a normal view when you are learning Linux/UNIX at first. Don’t worry, you will not be expected to explain a manual page!

6.6. When you are finished looking at the manual page, hit “**q**” to quit viewing it and return to the command prompt.

6.7. That concludes our revision of/basic introduction to Linux commands.