IT Scripting and Automation

Review:
OSF
Part II (Linux)

Lecturer: Art Ó Coileáin

Unix Shell

- Modern Linux distributions have a wide range of graphical user interfaces, but an administrator will always need to know how to work with the command line, or shell as it is called.
- The shell is a program that enables text-based communication between the operating system and the user. It is usually a text mode program that reads the user's input and interprets it as commands to the system.
- A shell and consist of both and interactive command language as well as a scripting language.

Unix Shell

- In Unix there are two major types of shells:
  - The Bourne Shell – default prompt is $ character.
  - The C Shell – default prompt is the % character.

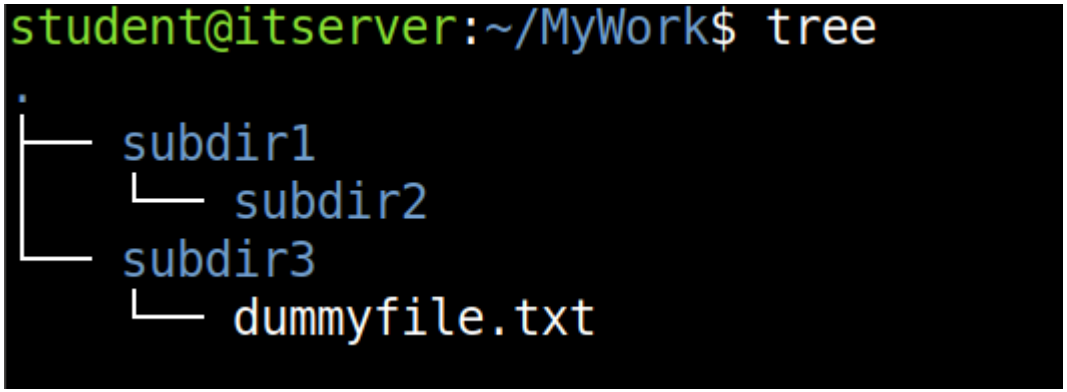
- Subcategories of Bourne Shell:
  - Bourne Shell (sh)
  - Korn Shell (ksh)
  - Bourne Again Shell (bash)
  - POSIX Shell (sh)

- Subcategories of C-type shell:
  - C shell (csh)
  - TENEX/TOPS C Shell (tcsh)

The Filesystem

- The Linux filesystem is similar to other operating system's filesystems in that it contains files and directories.
- The file system is responsible for representing and organising the system's storage resources.

- The file system is presented as a single unified hierarchy that starts at the directory /and continue downward through an arbitrary number of subdirectories.
- / is also called the root directory.



File and Directories

- File and directory names in Linux can contain lower case and upper-case letters, numbers, spaces and special characters. However, since many special characters have a special meaning in the Linux shell, it is good practice to not use spaces or special characters when naming files or directories.

Changing Current Directory

- Navigation in Linux is primarily done with the **cd** command. This changes directory.

```
student@itserver:~$ pwd
/home/student
student@itserver:~$ cd itsa/
student@itserver:~/itsa$ pwd
/home/student/itsa
student@itserver:~/itsa$ cd /
student@itserver:/$ pwd
/
student@itserver:/$
```

Pathnames

- The list of directories that must traversed to locate a particular file plus the file's filename form the pathname.

- Pathnames can be:
  - **absolute** such as '/tmp/foo' **or**
  - **relative** such as 'bookN1/topic'
  - **relative e.g.**: `cat ../../somefolder/dummy.txt`

- Relative pathnames are interpreted **starting** at the current directory.

- Usually the terms: pathname and path are used interchangeably.

File and Directories (2)

The Special Relative Path for Home

- When you start a new terminal session in Linux, you see a command prompt similar to this:

```
student@itserver:~$
```

- The tilde (~) here represents our home directory.

Hidden Files and Directories

- In the past you might have used the **-a** option for the **ls** command and you might have notice two special relative paths: **.** and **..** The -a option will list all files and directories, including hidden files and directories.
- Hidden files and directories will always begin with a (**.**)

Frequently used Linux Commands

Exercise:

- The following are common Linux commands you encountered in OS Fundamentals, list and explain the purpose and correct usage of each:

  - man
  - ls
  - pwd
  - mkdir
  - touch
  - cp
  - mv
  - rm
  - ps
  - date
  - df
  - chmod
  - cat

Hint: To get the manual of a particular command use “man” e.g.: man ls

Meta and WildCards

- Metacharacters are characters that the shell interprets as having a special meaning, examples:
  - <> | ; ! ? * [] $ \ “ ‘ ~() {}

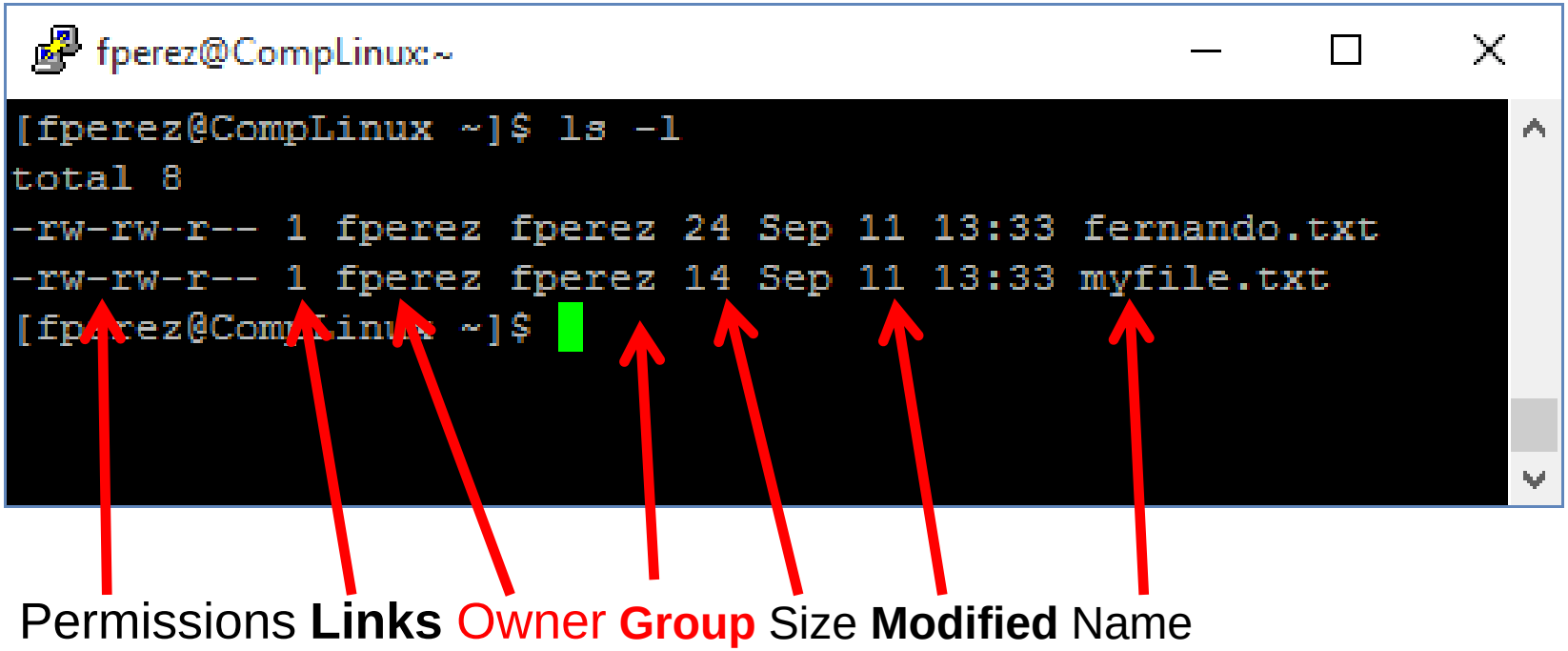
- Wildcards are a subset of metacharacters that are used to search for and match file patterns, examples:
  - ? * [] [-]
- Examples:
- **ls na?** List all files beginning *na* and ending with one further letter. This letter is not known in advance. Possible results include files such as nas, nan
- ls ?[2-4] List all files beginning with a character, and ending with a number 2,3 or 4.

Most common used wildcards

Wildcard	Matches
*	zero or more characters
?	exactly one character
[abcde]	exactly one character listed
[a-e]	exactly one character in the given range
[!abcde]	any character that is not listed
[!a-e]	any character that is not in the given range
{linux, debian}	exactly one entire word in the options given
[1-6]	any number in the given range

Basic File System Security

- Permissions – To control read, write, and execute access to
  - a file for the file's owner,
  - group and
  - other users.



Basic File System Security

- Permissions Bits

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x,

Example: `-rw-r--r--` or `drwxr-xr-x`

If **d** is present, it will be at the left hand end of the string, and indicates a directory.

otherwise **-** will be the starting symbol of the string, which indicates a file.

```
[fperez@CompLinux ~]$ ls -l
total 8
-rw-rw-r-- 1 fperez fperez 24 Sep 11 13:33 fernando.txt
-rwxrw-rw- 1 fperez fperez 14 Sep 11 13:33 myfile.txt
```

The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

– **Owner**: The **left** group of 3 gives the file permissions for the user that owns the file (or directory)

– **Group**: The **middle** group gives the permissions for the group of people to whom the file (or directory) belongs

– "**World**": The **rightmost** group gives the permissions for all others ("world").

The symbols **r**, **w**, **x** have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access Rights on Directories and Files

- So, in order to read a file, you must have execute permission on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.
- To change access permissions on a file we need to change the file access rights. We do this using the **chmod** command for Changing File Permissions
- The format of chmod is **chmod [mode] filename**
- Create **mode** by concatenating characters from ***who***, ***opcode*** and ***permission***

Symbol	Meaning
u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

Redirects

- The reassignment of a channel's file descriptor in the shell environment is called a redirect.
- A redirect is defined by a special character within the command line. E.G.: to redirect the standard output of a process to a file, the greater than symbol **>** is positioned at the end of the command and followed by the path to the file that will receive the redirected output:

```
student@itserver:~$ cat /proc/cpuinfo >cpuinfo.txt
```

- By default, only the content coming to **stdout** is redirected.
- The numerical value of the file descriptor should be specified just before the greater than symbol, however when it is not specified Bash redirects the standard output.

  I.E.: Using **>** is equivalent to use **1>** (*the value of stdout's file descriptor is **1***).

Error Redirection

- Default Standard Error
  - cat filename.txt : If filename does not exist we get a message like…
    - cat: filename.txt: No such file or directory
- Redirecting error output to a file:
  - cat filename.txt 2> errorfile.txt
  - cat errorfile.txt
  - cat: filename.txt: No such file or directory
- Redirect and append error output to a file:
  - cat filename.txt 2>> errorfile.txt

Linux Environment

Summary:

- STDIN (Standard Input)
- STDOUT (Standard Output)
- STDERR (Standard Error)

File Descriptor	Name	Abbreviation	Default	Symbol
0	Standard In	STDIN	Keyboard	<
1	Standard Out	STDOUT	Terminal	>
2	Standard Error	STDERR	Terminal	2>

Processes and Jobs (Linux)

- A process is an executing program identified by a unique PID (process identifier).
- **ps** Command display information about the processes.
- To background a process, type an **&** at the end of the command line.
- The command **sleep** waits a given number of seconds before continuing
  - **sleep 10**
- The command **jobs** lists suspended and background processes
- To restart (foreground) a suspended processes, type **fg *%jobnumber***

Processes and Jobs (Linux)

Process Termination:

- To Kill a process (for example, when an executing program is in an infinite loop). To kill a job running in the foreground, type **^C** (control + c).
  - **sleep 100 ^C**

- To kill a suspended or background process type
- **kill *%jobnumber*** or
- ***kill -9 PIDnumber***
  - **sleep 100 &**
  - **jobs**
  - If it is job number 4, type **kill %4**

Environment Variables & other cmds

- **Environment Variables**
- To view the global environment variables, use the **env** (or **printenv** command)
  - **$ printenv**
- To define a variable as follows: $ X="hello"
- To view the global and local environment variables, use the **set** command.

- **Other useful commands**:
- The wc command (*short for word count*): print newline, word and byte counts for each file.
  - **wc –l myfile.txt**
- The **grep** command: print lines matching a pattern.
  - **grep 'word' filename**
  - **grep 'word' file1 file2 file3**
  - **cat otherfile | grep 'something'**
  - **grep --color 'word' fileName**

Quotes

Double Quotes:

- When using quotation marks in Bash scripting, everything between “ and ” is taken literally, ***with the exception of some special characters*** (see next slide) . Therefore, if you have the following line in your script:
- **echo “printing X to the screen”**
- you will see the following: **printing X to the screen**

Single Quotes:

- With single quotes, everything between ‘ and ’ is taken literally, except for another ’. *Single quotation marks* will always gives you literally what's inside the quotation marks--any characters that might otherwise have special meaning to the shell (like the dollar sign or the backslash) are treated literally (i.e. not seen as being special).

Use *double quotation marks* when you want to assign a string that contains special characters the shell should act on.

Quotes and command substitution

- The exceptions to this are the following characters, which keep their special meaning:
- $ variable substitution will occur
- ` command substitution will occur when using the backtick
  - The key beside 1 on your keyboard.
  - This will sometimes enter 2 single quotes so delete 1.
- \ The backslash is used to escape (treat literally) a single character (such as $ or *) that might otherwise be treated as a special character by the shell, i.e. \ the character following a \ is taken literally.

Quotes & Backslash

- **Examples:**
- **$ X=”hello world”**
- **$ echo “printing $X to screen”**
- *printing hello world to screen*
-
- **$ echo \$X**
- $X
-
- **echo ‘Single quotes “protect” double quotes ’**
- *Single quotes “protect” double quotes*
-
- **echo “Well, isn’t that \”special\”?”**
- *Well, isn’t that “special”?*
-
- **echo “You have the following files in `pwd`”**
- *You have the following files in /home/student_number/LinusLab3*
-
- **x=100**
- **echo “The value of \$x is $x”**
- *The value of $x is 100*
-

Shell Scripts

- The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a hash, #, describing the steps.

- To create a Shell script, we will use VIM text editor
  - Enter the script commands into a file with extension .sh
  - Save and Exit
  - Make the file executable using “chmod”
  - Run the script file

Example:

- MyScript.sh

```
#!/bin/bash
MESSAGE="Hello World"
echo $MESSAGE
```