EECS2031

Lab 1
Summer 2016

Lab Objectives

In this lab, you will be introduced to the Linux operating system. The basic commands will be presented in this lab.

What is a shell?

A shell is the interface between you and the computer. It accepts commands typed by the user and then uses the operating system in order to execute these commands. For example the command could be `ls` followed by enter

```
sh-3.00$ ls
emp.data  f2   file1     one_liners.log one_liners.pdf p1   p3
f1        f3   file2     one_liners.odt one_liners.ps  p2
sh-3.00$
```

Before you enter `ls`, the shell displayed the prompt (in this case sh-3.00$) and sits patiently waiting for your input. After typing `ls` and pressing enter, it takes the command (ls – listing all the files in the current directory) and runs it (using the OS) producing the output shown above.

Where the shell can find the program `ls` to run? There is a variable called PATH (We will discuss how to set PATH variable later). This variable lets the shell look for any command that you type in specific places, if the command is not in one of the places defined by PATH, then the shell displays a message on the form (The exact form depends on what shell you are running on your system)

```
sh-3.00$ Command not found
```

`ls` is one of many basic commands that you can use to manipulate your files. The best way to know how to use a command is to use the `man` command (short for manual). For example type `man ls` at your shell prompt and you get a description of the `ls` command and the options you can use with it.

What are the different kinds of shells?

There are many shells that you can use on any Unix/Linux system. We will briefly introduce these shells and compare between them. There is a shell that starts when you login, that is the default shell and is set by your system administrator. You can start other shells, or you can change your default shell later.

The Bourne Shell

Designed by Stephen Bourne of AT&T Bell Laboratories and released in 1977. The Bourne shell, or sh as it is widely known is considered to be the original shell (although it was actually a replacement for Thompson Shell). It is considered to be a standard on any UNIX system. The Bourne shell is considered the least common denominator for shells. For truly portable scripts, use the Bourne Shell and it will run on many shells.

The C Shell

Designed by Bill Joy at the University of California at Berkeley for the BSD UNIX. Its syntax resembles that of C, hence the name. It introduced new features to the shell like the history substitution (!! to repeat the last command) and the ~ expansion for home directory lookup.

There are some syntax difference between csh and sh, for example in sh we set a variable as "a=b" while in csh "set a=b"

The Korn Shell

Developed by David Korn of AT&T Bell Laboratories for its System V Unix as a response to the csh. It has many of the features of csh but is backward compatible with sh. The Korn shell has been standardized as a part of POSIX

The Bourne Again Shell – Bash

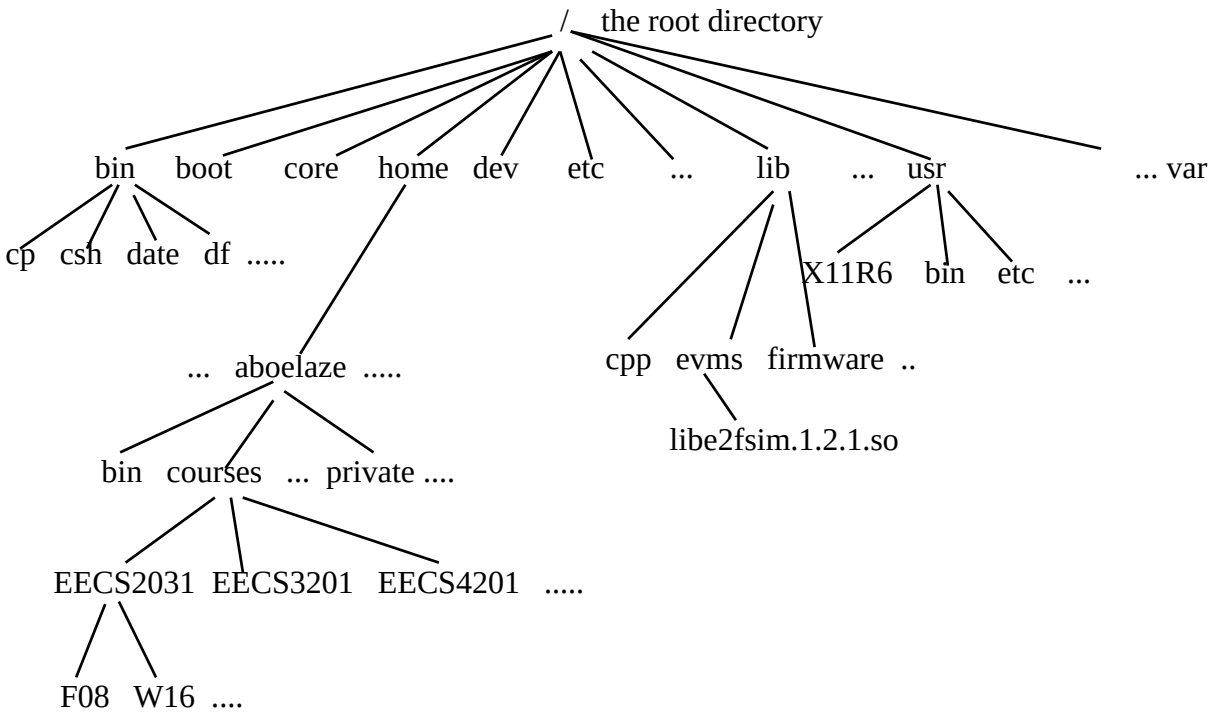
The Bourne Again Shell or bash has been developed for the GNU project and is mainly a response to the Korn shell. It has many features in the Korn shell, but is backward compatible with the original Bourne Shell.

The Extended C shell tcsh

The tcsh extends the original c shell by adding file name completion and command line editing. It is also backward compatible with csh.

Unix File System

The Unix/Linux file system is organized on the form of a tree. The root directory is defined as "/" without the quotation marks. You can define files or directories relative to the directory you are in, or as an absolute starting from the / root directory. An example of a UNIX directory tree is shown below.



For example, when I refer to the directory "courses" I can refer to it as ~/courses (~ means home directory) or the full path /home/aboelaze/courses. The command pwd (print working directory) displays the name of the directory you are currently in.

File Protection

In Unix/Linux each file has access rights that are determined by the access bits. For example, when you type `ls -l filename`

```
tigger 127 % ls -l abs.txt
-rwx------  1 aboelaze faculty 1066 Nov 15  2005 abs.txt*
tigger 128 %
```

The access bits in this case is – rwx------
The first dash states that the file is a regular file and not a directory (it would be d if it were a directory). The following 9 characters are divided into three groups with 3 character per group. The groups determine the access rights for user (owner),

group and others. In each group the rights are rwx (for read, write, and execute). For the above example, the owner has the right to read, write (modify), or execute the file. The group and others do not have access rights at all to the file.

Access bits on the form rwxr-xr-- means the owner have the right to read, write, and execute the file. People in the same group as the owner have the rights to read and execute the file (no write) stated as r-x, while others have the rights to read the file but not write it or execute it.

Basic Unix Commands

Login and open an xterm. First to know your login shell, echo the login shell variable

```
tigger 207 % echo $SHELL
/cs/local/bin/tcsh
tigger 208 %
```

That means that I am using a tcsh shell. To explain what the commands mean, echo echoes what follows it verbatim if you say echo bla bla bla, then "bla bla bla" will be displayed on the screen. However the dollar sign before SHELL tells the shell do not just echo the phrase SHELL but the variable named SHELL which is a shell variable holds the name of the login shell.

Unix/Linux commands are of the form
`commands arguments`
or
`commands options arguments`

For example, when we did ls command before, that is an example of a command without any arguments. We can do

ls file lists the named file if it exists
or
ls directory lists the contents of the directory if exists

There are a lot of options that could be used with ls (-a -s -l -f) do man ls to check all the options.

Some other commands

`rm`	removes (deletes) a file or a directory
`mv`	moves a file or director
`cp`	copies a file
`ps`	lists the running processes
`kill`	kills a process (kill -9 for maximum effects)

For example, try to man uname (another Linux command), here is part of the man pages for uname.

After you man uname, try it for different options as given below.

```
tigger 119 % man uname
UNAME(1)             User Commands                    UNAME(1)


NAME
       uname - print system information

SYNOPSIS
       uname [OPTION]...

DESCRIPTION
       Print certain system information.  With no OPTION, same as -s.

       -a, --all
              print all information, in the following order:

       -s, --kernel-name
              print the kernel name

       -n, --nodename
              print the network node hostname

       -r, --kernel-release
              print the kernel release

       -v, --kernel-version
              print the kernel version

       -m, --machine
              print the machine hardware name

       -p, --processor
              print the processor type

       -i, --hardware-platform
              print the hardware platform
```

grep (**g**lobally search a **r**egular **e**xpression and **p**rint)
The command grep, as its name indicates, greps a specific pattern from a file and display the lines containing that line (by default). It has many options check man grep for complete list of options.

```
grep book file1
```

display all the lines in file1 that include the string book.

Another useful command is wc (short for word count). wc counts the number of characters, words, and lines in a file.

```
tigger 157 % wc temp
  3  32 169 temp
tigger 158 %
```

That means the file temp has 3 lines, 32 words, and 169 characters.

Redirection and Pipes

By default, Unix/Linux uses three input/output channels

- 0=stdin (standard input, by default the keyboard)
- 1=stdout (standard output, by default the monitor)
- 2=stderr (standard error, by default the monitor)

For example when I type cat filename the file is displayed on the monitor. If there is an error of any type (for example, the file does not exist, the error message is displayed on the monitor too).

```
sh-3.00$ cat hgfhgfjhf
cat: hgfhgfjhf: No such file or directory
sh-3.00$
```

we can redirect inputs (<) and outputs (>) for example

```
cat filename >out.txt >err.txt
```

cats the file filename and stores it in a file called out.txt, if there is any error message it will be stored in a file err.txt instead of being displayed on the monitor.
`>>` means to append the output file (> deletes the contents of the file before sending the output to it).
`<< string` means the string is considered to be the end of input and on a line by itself (default CTRL-D)

```
tigger 113 % cat <<XX
? first line
? second line
```

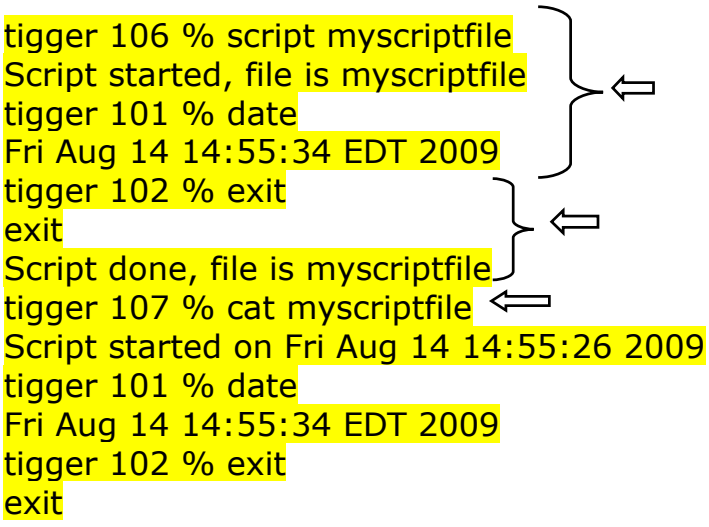
```
? last line XX
? XX
first line
second line
last line XX
tigger 114 %
```

Pipes are used to send the output of a program to the input of another program. For example, the command

```
ls | less
```

`ls` by itself displays the directory on the monitor. The "|" pipe the output of the `ls` to the input of another program called `less` (`less` displays its input on the screen). So what is the difference between the above command and simple `ls`. One thing about less is that it displays the contents a screen by screen, so a screen-full of file names is shown, and it waits for any input character

Finally, the command `script`. Script makes a copy of your terminal session (do man script for more details).
The way `script` works is as follows. When you type `script`, a file is created that will store every command you type and everything that is displayed on the screen. So, for example, if you type `ls` then the ls command together with the displayed results will be stored in the script file. You can specify the name of the file with the script command. If you did not specify the file, the default file is `typescript`. Do not forget to exit the scripting session using `exit` command.
An example is shown below.



Starting the script, then you executed the date command, the result is shown on the screen

Exit to end the scripting session

Now, we are out of the scripting session, the contents is stored in a file called myscriptfile. Cat this file to see the contents of the session

Practice

1. Start a script session to a file called myscript0. Type and execute a Linux command to do the following

    - Count the number of files in your current directory
    - Count the number of files in your current directory that includes the phrase "HW"
    - Repeat the previous command but consider both capital and small letters (i.e. the files that contains HW, Hw, hW, and hw).
    - A command to count the number of lines that you input from the standard input that contains the string book. The number should be displayed after you finish entering text.

2. Write a C program named countpos.c The program reads a sequence of positive numbers and displays their sum. It stops after the first negative number (the negative number is not added), and prints the result. The result should be printed in 10 spaces left justified followed by a newline character.