Workshop: Software Writing Skills for Young Researchers (23.-25. September 2015)

% Document created by Malvika Sharan

Novice Unix Shell:

Introduction

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or kernel. It interacts with hardware and most of the tasks like memory management, tash scheduling and file management. UNIX is a computer operating systems (OS) originally invented by Ken Thompson in 1969 and reimplemented in C during 1972-1974, making it the first source-portable OS. Unix is a multi-user and multi-tasking operating system. You can read more about Unix in Wikipedia, a free encyclopdia, at: http://www.wikipedia.org/wiki/Unix.

Shell provides a command-line interpreters for interactive sessions (via terminal) to execute commands and programs in Unix. We will use shell to work with some useful Unix-shell commands.

The first thing you will notice in the shell terminal is a dollar sign. This is command prompt, issued by shell that suggests that the commands can be typed. A command followed by Enter key (or return key) allows shell to read and execute the command. We refer this as running a command (command + Enter/return key).

Run command date, to get the date and time and press Enter key:

```
$ date
```

It displays an output like this: Wed Sep 23 10:27:19 CEST 2015

A new prompt $ is printed after a command is executed that tells us that it's ready for more commands.

Run command whoami to get ID of the current user:

```
$ whoami
```

Files and directories:

The part of OS responsible for managing files and directories (folders) is called file system. A filesystem is a logical subdivision of hard disk space that organizes our data into files and folders in

hierarchical structure. All data in UNIX is organized into files and all files are organized into directories. These directories are organized into a tree-like structure called the filesystem. The topmost node of tree is called root which encloses all the other files.

To identify where we are in the system, run pwd command, which stands for “print working directory”:

```
$ pwd
```

This displays the current working directory (path of the current location), which would look something like this:

```
$ /Users/username…
```

By default you are in the folder 'software_writing_skills'. Check what's in the current working directory by running ls (stands for listing):

```
$ ls
```

This displays all the files and folders in the current working directory, which are examples, shell_introduction.txt and shell_material.txt. By using the flag -l, we can get a more comprehensible output of the command ls.

```
$ ls -l
```

There are several flags that can be used with ls. For example:

```
$ ls -r (r stands for reverse, lists file and folders in reverse order)

$ ls -a (a stand for all, lists everything including the special files)

$ ls -s (s stands for sort, lists files and folders in alphabetical order)

$ ls -R (R stands for recursive, lists the contents of directories recursively)

$ ls -t (t stands for time, lists things by time of last change)
```

Additionally, multiple flags can be used in the same command:

```
$ ls -R -s
```

A file or folder can be addressed with an absolute or a relative path. One can access the files in the

current folder directly by using a relative path, which refers to the folder relative to the current path. To access files in different folder the absolute path should be provided which refers to the whole path (starts with /). For example, using pwd command the absolute path of the current working directory is displayed.

By giving the name of the folder (in the current working directory or the full path of a folder somewhere else) one can see the contents of the folder:

Try this and hit tab.

```
$ ls /root/Downloads/
```

This is where you downloaded the example folders.

We can use cd followed by a directory name to change our working directory. cd stands for “change directory”, which is a bit misleading: the command doesn't change the directory; it changes the shell's idea of what directory we are in. We can change our current directory by moving to /root/Downloads.

```
$ cd /root/Downloads
```

Since all our examples are in the following specified directory, let's move there by cd and check what do you have there.

```
$ software_writing_skills_potsdam-master/software_writing_skill/examples

$ ls examples
```

We can check the changed working directory by pwd and we can run ls -l to see the files in the current path.

To go down to the directory enclosing the current files and folders we can use cd .., where .. refers to a special directory, meaning “the directory containing this one”, or more concisely, the parent of the current directory. If we run pwd after running cd .. we're back in the last location.

The default cd changes the working directory to the users home directory.

```
$ cd
```

To go back to the last location of working directory we can use cd followed by the absolute path or simply type in cd -.

```
$ cd -
```

Creating and editing files and folders

To create a new directory in the current path run mkdir followed by directory name.

```
$ mkdir first_folder
```

We can change our current working folder to the newly created folder by cd first_folder. We can further create folder(s) into this folder by using mkdir followed by foldername.

```
$ mkdir second_folder
```

To create an empty file in the current path, run touch command followed by filename.

```
$ touch testfile1.txt
```

Or we can create a file while writing into it by using text editors like nano, Vim, Emacs etc. For example here we will use nano editor to write something into the file:

```
$ nano testfile2.txt
```

Using Control-O (+Enter key) we can write our data to disk. Once the file is saved we can use Control-X to quit the editor and return to the shell. More options are displayed at the bottom of the editor. Alternatively, we can use graphical editors like Gedit and Komodo or Notepad++ in windows for writing into the file.

Run ls to see the created file/files.

```
$ ls (this will show testfile1.txt, testfile2.txt and second_folder that you
created in the current path)
```

An asterisk * is a wild card, which in shell stands for all. * can be used in the commands to execute an action on all, in this context files and folders. We can use ls *.txt to list all the files that comprise of .txt file extension.

The file can be copied from one path to another by using cp command.

```
$ cp testfile2.txt second_folder
```

All the .txt files can be copied into the second_folder by using *.

```
$ cp *.txt second_folder
```

The file can be renamed while copying to another location or folder:

```
$ cp testfile2.txt second_folder/testfile3.txt
```

The file can re-named and copied into the same folder in the same manner except that the path of the folder is not required.

```
$ cp testfile2.txt testfile4.txt
```

The file can be moved to another location by using the command mv. The files can be renamed while moving in similar manner as mentioned above.

```
$ mv testfile4.txt second_folder/
```

The contents in the folder are removed by using rm followed by the name of the folder in the current path or the absolute path of the folder.

```
$ rm testfile1.txt

$ rm second_folder/testfile1.txt
```

You can check is we really deleted the file by using ls command.

To delete an entire folder -r flag is used which stands for recursive.

```
$ rm -r second_folder/
```

rm * will remove all the files in the current working directory. There is no trash bin in Unix shell, which means if you delete a file or a folder, it is deleted forever and cannot be restored.

These were the basic Unix shell commands. In the beginning you will have many questions about what does a command do, what are flags that can be used with a command etc. An important command man (stands for manual) can be used understand each commands in detail. For example, to get the documentation of ls command:

```
$ ls man
```

To close the manual use q. Additionally many tools offer help via the parameter -h, -help or -help.

```
$ ls -help
```

Few keyboard shortcuts that will come in handy:

```
Tab: to extend command or file/folder names
```

Up and down key: to call the previously used commands

```
Ctrl+l: clear the screen (optionally type clear to do the same)

Ctrl+A: to move to the beginning of the file

Ctrl+e: to move to the end of the file

Ctrl+c: stop the command that is currently running

Ctrl+z: to send the currently running command to the background. The fg command is
used to bring the background commands to foreground.
```

Handling contents of the files and folders

Earlier we learnt the use of editors to create and edit a file. In this next part we will see what more can be done for handling a file. For this, we will work on some example files to get a better understanding. I have saved some mock files in the folder 'software_writing_skills/examples'. if you are in the folder first_folder, go the folder software_writing_skills by typing:

```
$ cd ..

$ cd software_writing_skills/examples
```

less and more commands are used to view the file content without editing. Use q to quit the file viewing by less.

```
$ less permafrost_database.txt

$ more permafrost_database.txt
```

Using less the file can be viewed by scrolling up and down, but by using more the first few lines of the files can be seen on the terminal.

To see the first or last few lines of the file, use head and tail commands respectively.

```
$ head permafrost_database.txt
```

```
$ tail permafrost_database.txt
```

Flag -n with the commands head and tail is used to specify the number of lines that you want to view. For example, to view first 10 lines of the file either of the following commands can be used:

```
$ head -n 10 permafrost_database.txt

$ head -10 permafrost_database.txt
```

cut command is used to extract certain fields of the file. Unlike less, more, head and tail where the file can extracted horizontally, cut helps in extract the vertical file contents.

Show character at position 2 in each line:

```
$ cut -c2 permafrost_database.txt
```

Show characters from position 2 to position 5 in each line:

```
$ cut -c2-5 permafrost_database.txt
```

Show characters at position 2 and 5:

```
$ cut -c2,5 permafrost_database.txt
```

Show characters starting at position 2 till the end:

```
$ cut -c2- permafrost_database.txt
```

Show characters from the beginning till the position 5:

```
$ cut -c-5 permafrost_database.txt
```

The flag -d specifies the field delimiter used in the input file (comma, tab) and flag -fà specifies the field (divided by the delimiter) we want to see:

```
$ cut -d',' -f1 permafrost_database.txt
```

These are different options of cut commands that can be used for reading files. However, the practical use of this command is in reading a file that contains table. A table comprise of values that are separated by special characters like comma , or tab \t that can be used to visualize certain

column (also known as field). In the current folder we have a table geochemistry_germany_rivers.tab containing several columns that are separated by tab. By using less you can check the contents of the file. We can extract the values in the column 3:

```
$ cut -d$'\t' -f3 geochemistry_germany_rivers.tab
$ cut -f3 geochemistry_germany_rivers.tab
```

Regular expressions (for e.g.: specific strings) are searched in files by using command grep:

```
$ grep permafrost permafrost_database.txt
```

Count the occurrence of the string by using the flag -c:

```
$ grep -c permafrost permafrost_database.txt
```

To make the search case insensitive -i flag can be used:

```
$ grep -c -i permafrost permafrost_database.txt
```

We can extract the entire content of the file on the terminal by using cat.

```
$ cat geochemistry_germany_rivers.tab
```

The wide use ofthe command cat is to collect the entire contents of the file for further analysis like for writing the content from one file to another (in relative or absolute path).

```
$ cat geochemistry_germany_rivers.tab > copy_germany_rivers.tab

$ cat temperature_course.txt permafrost_database.txt > abstracts.txt
```

Please check the contents of abstract.txt by less.

To print something on the terminal, echo command is used.

```
$ echo “print this!”
```

echo can also be used to append something to a file:

```
$ echo “Combining all the abstract files” > abstract.txt
```

The > symbol directs the given content to the destination file, however this overwrites the contents if the destination file already exists. This you can realize by checking the abstract.txt file now, which contains only one line that you wrote by using echo command. To avoid the overwriting, >> is used:

```
$ echo “Combining all the abstract files” > abstract.txt

$ echo “Abstract-1” >> abstract.txt

$ cat temperature_course.txt >> abstract.txt

$ echo “Abstract-2” >> abstract.txt

$ head -20 permafrost_database.txt >> abstract.txt
```

In another example we will collect few columns (column 3 and 5) from the file geochemistry_germany_rivers.tab

```
$ cut -f3,5 geochemistry_germany_rivers.tab > water_depth.txt
```

To remove empty lines from the file use following:

```
$ grep -v '^$' water_depth.txt > new_water_depth.txt
```

To count the total number of words wc and total lines wc -l commands are used. For example, this command can be used for counting total number of lines in file:

```
$ wc water_depth.txt
$ wc -l water_depth.txt
$ wc -l new_water_depth.txt
```

In order to sort the content of a file the command sort is used. By default the sorting takes placed alphabetically but several flags can be used to specify the type of sorting, for example, sort -n numerical sorting, sort -r reverse sorting etc.

```
$ sort new_water_depth.txt

$ sort -n new_water_depth.txt

$ sort -r new_water_depth.txt

$ sort -n new_water_depth.txt > sorted_water_depth.txt
```

To view non-redundant contents of a file uniq command is used:

```
$ uniq sorted_water_depth.txt
```

By using the -c flag with uniq, the number of occurrence of each entry can be seen as the first field on the terminal.

```
$ uniq -c sorted_water_depth.txt
```

Loops:

Wildcards * and keyboard shortcuts are two ways to reduce typing (and typing mistakes). Another is to tell the shell to do something over and over again when we intend to carry out same action on 1000 files, which is impractical and tiring. Instead, we can use a loop to do some operation once for each thing in a list. Here's a simple example that displays the first three lines of each file in turn (write each of the following lines one by one):

```
$ for filename in permafrost_database.txt temperature_course.txt
> do
>    head -3 $filename
> done
```

As you noticed that in the 'for-loop', each file is taken one by one as a variable filename. This variable will be referred with a $ in the 'for-loop' ($filename). > in the command above shows that more commands can be written as the action is not finished.

Combining commands:

In Unix we can connect several commands by pipe | to run several actions on standard input and standard output in a stepwise manner.

```
$ cat water_depth.txt | head -5 | tail -5 | sort
```

Shell Scripts

Shell is a powerful programming environment because it allows us to take the commands we repeat frequently and save them in files. The file extension for Shell script is '.sh' and using this file we can re-run all the operations again by using a single command. For historical reasons, a bunch of commands saved in a file is usually called a shell script, but make no mistake: these are actually small programs. The shell scripts are run by using sh (stands for Shell) or bash (Bourne again Shell).

```
$ bash script.sh
```

Useful resources

- Tutorial Point
- UNIX Tutorial for Beginners