1 Unix Shell 1: Introduction

Lab Objective: *Explore the basics of the Unix Shell. Understand how to navigate and manipulate file directories. Introduce the Vim text editor for easy writing and editing of text or other similar documents.*

Unix was first developed by AT&T Bell Labs in the 1970s. In the 1990s, Unix became the foundation of Linux and MacOSX. The Unix shell is an interface for executing commands to the operating system. The majority of servers are Linux based, so having a knowledge of Unix shell commands allows us to interact with these servers.

As you get into Unix, you will find it is easy to learn but difficult to master. We will build a foundation of simple file system management and a basic introduction to the Vim text editor. We will address some of the basics in detail and also include lists of commands that interested learners are encouraged to research further.

> **Note**
>
> Windows is not built off of Unix, but it does come with a command line tool. We will not cover the equivalent commands in Windows command line, but you could download a Unix-based shell such as Git Bash or Cygwin to complete this lab (you will still lose out on certain commands).

File System

> **Achtung!**
>
> In this lab you will work with files on your computer. Be careful as you go through each problem and as you experiment on your own. Be sure you are in the right directories and subfolders before you start creating and deleting files; some actions are irreversible.

Navigation

Typically you have probably navigated your comptuer by clicking on icons to open directories and programs. In the terminal, instead of point and click we use typed commands to move from directory to directory.

Begin by opening the Terminal. The text you see in the upper left of the Terminal is called the *prompt*. As you navigate through the file system you will want to know *where* you are so that you know you aren't creating or deleting files in the wrong locations.

To see what directory you are currently working in, type `pwd` into the prompt. This command stands for **p**rint **w**orking **d**irectory, and as the name suggests it prints out the string of your current location.

Once you know where you are, you'll want to know where you can move. The `ls`, or **l**ist **s**egments, command will list all the files and directories in your current folder location. Try typing it in.

When you know what's around you, you'll want to navigate directories. The `cd`, or **c**hange **d**irectory, command allows you to move through directories. To change to a new directory, type the `cd` command followed by the name of the directory to which you want to move (if you cd into a file, you will get an error). You can move up one directory by typing `cd ..`.

Two important directories are the root directory and the home directory. You can navigate to the home directory by typing `cd ~` or just `cd`. You can navigate to root by typing `cd /`.

Problem 1. Using these commands, navigate to the `Shell1/` directory provided with this lab. We will use this directory for the remainder of the lab. Use the `ls` command to list the contents of this directory. NOTE: You will find a directory within this directory called `Test/` that is availabe for you to experiment with the concepts and commands found in this lab. The other files and directories are necessary for the exercises we will be doing, so take care not to modify them.

Getting Help

As you go through this lab, you will come across many commands with functionality beyond what is taught here. The Terminal has two nice commands to help you with these commands. The first is `man <command>`, which opens the manual page for the command following `man`. Try typing in `man ls`; you will see a list of the name and description of the `ls` command, among other things. If you forget how to use a command the manual page is the first place you should check to remember.

The `apropos <keyword>` command will list all Unix commands that have `<keyword>` contained somewhere in their manual page names and descriptions. For example, if you forget how to copy files, you can type in `apropos copy` and you'll get a list of all commands that have `copy` in their description.

Flags

When you typed in `man ls` up above, you may have noticed several options listed in the description, such as `-a`, `-A`, `--author`. These are called flags and change the functionality of commands. Most commands will have flags that change their behavior. Table 1.1 contains some of the most common flags for the `ls` command.

Flags	Description
`-a`	Do not ignore hidden files and folders
`-l`	List files and folders in long format
`-r`	Reverse order while sorting
`-R`	Print files and subdirectories recursively
`-s`	Print item name and size
`-S`	Sort by size
`-t`	Sort output by date modified

Table 1.1: Common flags of the `ls` command.

Multiple flags can be combined as one flag. For example, if we wanted to list all the files in a directory in long format sorted by date modified, we would use `ls -a -l -t` or `ls -alt`.

Manipulating Files and Directories

In this section we will learn how to create, copy, move, and delete files and folders. Before you begin, `cd` into the `Test/` directory in `Shell1/`.

To create a text file, use `touch <filename>`. To create a new directory, use `mkdir <dir_name>`.

To copy a file into a directory, use `cp <filename> <dir_name>`. When making a copy of a directory, the command is similar but must use the `-r` flag. This flag stands for recursively copying files in subdirectories. If you try to copy a file without the `-r` the command will return an error.

Moving files and directories follows a similar format, except no `-r` flag is used when moving one directory into another. The command `mv <filename> <dir_name>` will move a file to a folder and `mv <dir1> <dir2>` will move the first directory into the second. If you want to rename a file, use `mv <file_old> <file_new>`; the same goes for directories.

When deleting files, use `rm <filename>`, or `rm -r <dir_name>` when deleting a directory. Again, the `-r` flag tells the Terminal to recursively remove all the files and subfolders within the targeted directory.

If you want to make sure your command is doing what you intend, the `-v` flag tells `rm`, `cp`, or `mkdir` to have the Terminal print strings of what it is doing. When your Terminal gets too cluttered, use `clear` to clean it up.

Below is an example of all these commands in action.

```
$ cd Test
$ touch data.txt                # create new empty file data.txt
$ mkdir New                     # create directory New
$ ls                            # list items in test directory
New     data.txt
$ cp data.txt New/              # copy data.txt to New directory
$ cd New/                       # enter the New directory
$ ls                            # list items in New directory
data.txt
$ mv data.txt new_data.txt      # rename data.txt new_data.txt
$ ls                            # list items in New directory
new_data.txt
$ cd ..                         # Return to test directory
$ rm -rv New/                   # Remove New directory and its contents
```

Commands	Description
`clear`	Clear the terminal screen
`cp file1 dir1`	Create a copy of `file1` and move it to `dir1/`
`cp file1 file2`	Create a copy of `file1` and name it `file2`
`cp -r dir1 dir2`	Create a copy of `dir1/` and all its contents into `dir2/`
`mkdir dir1`	Create a new directory named `dir1/`
`mkdir -p path/to/new/dir1`	Create `dir1/` and all intermediate directories
`mv file1 dir1`	Move `file1` to `dir1/`
`mv file1 file2`	Rename `file1` as `file2`
`rm file1`	Delete `file1` [`-i`, `-v`]
`rm -r dir1`	Delete `dir1/` and all items within `dir1/` [`-i`, `-v`]
`touch file1`	Create an empty file named `file1`

Table 1.2: The commands discussed in this section.

```
removed 'New/data.txt'
removed directory: 'New/'
$ clear                          # Clear terminal screen
```

Table 1.2 contains all the commands we have discussed so far. Notice the common flags are contained in square brackets; use `man` to see what these mean.

Problem 2. Inside the `Shell1/` directory, delete the `Audio/` folder along with all its contents. Create `Documents/`, `Photos/`, and `Python/` directories.

Wildcards

As we are working in the file system, there will be times that we want to perform the same command to a group of similar files. For example, if you needed to move all text files within a directory to a new directory. Rather than copy each file one at a time, we can apply one command to several files using *wildcards*. We will use the `*` and `?` wildcards. The `*` wildcard represents any string and the `?` wildcard represents any single character. Though these wildcards can be used in almost every Unix command, they are particularly useful when dealing with files.

```
$ ls
File1.txt   File2.txt   File3.jpg   text_files
$ mv -v *.txt text_files/
File1.txt -> text_files/File1.txt
File2.txt -> text_files/File2.txt
$ ls
File3.jpg   text_files
```

See Table 1.3 for examples of common wildcard usuage.

Command	Description
`*.txt`	All files that end with `.txt`.
`image*`	All files that have `image` as the first 5 characters.
`*py*`	All files that contain `py` in the name.
`doc*.txt`	All files of the form `doc1.txt`, `doc2.txt`, `docA.txt`, etc.

Table 1.3: Common uses for wildcards.

Command	Description
`cat`	Print the contents of a file in its entirety
`more`	Print the contents of a file one page at a time
`less`	Like more, but you can navigate forward and backward
`head`	Print the first 10 lines of a file
`head -nK`	Print the first `K` lines of a file
`tail`	Print just the last 10 lines of a file
`tail -nK`	Print the last `K` lines of a file

Table 1.4: Commands for printing contents of a file

Problem 3. Within the `Shell1/` directory, there are many files. We will organize these files into directories. Using wildcards, move all the `.jpg` files to the `Photos/` directory, all the `.txt` files to the `Documents/` directory, and all the `.py` files to the `Python/` directory. You will see a few other folders in the `Shell1/` directory. Do not move any of the files within these folders at this point.

Displaying File Contents

When using the file system, you may be interested in checking file content to be sure you're looking at the right file. Several commands are made available for ease in reading file content.

The `cat` command, followed by the filename will display all the contents of a file on the screen. If you are dealing with a large file, you may only want to view a certain number of lines at a time. Use `less <filename>` to restrict the number of lines that show up at a time. Use the arrow keys to navigate up and down. Press **q** to exit.

For other similar commands, look at table 1.4.

Searching the File System

There are two commands we use for searching through our directories. The `find` command is used to find files or directories in a directory hierarchy. The `grep` command is used to find lines matching a string. More specifically, we can use `grep` to find words inside files. We will provide a basic template in Table 1.5 for using these two commands and leave it to you to explore the uses of the other flags. The `man` command can help you learn about them.

Problem 4. In addition to the `.jpg` files you have already moved into the `Photot/` folder, there are a few other `.jpg` files in a few other folders within the `Shell1/` directory. Find where

Command	Description
`find dir1 -type f -name "word"`	Find all files in `dir1/` (and its subdirectories) called `word` (`-type f` is for files; `-type d` is for directories)
`grep "word" filename`	Find all occurences of `word` within `filename`
`grep -nr "word" dir1`	Find all occurences of `word` within the files inside `dir1/` (`-n` lists the line number; `-r` performs a recursive search)

Table 1.5: Commands using `find` and `grep`.

these files are using the `find` command and move them to the `Photos/` folder.

Pipes and Redirects

Terminal commands can be combined using *pipes*. When combined, or *piped*, the output of one command is passed to the another. Two commands are piped together using the `|` operator. To demonstrate pipes we will first introduce commands that allow us to view the contents of a file in Table 1.4.

In the first example below, the `cat` command output is piped to `wc -l`. The `wc` command stands for **w**ord **c**ount. This command can be used to count words or lines. The `-l` flag tells the `wc` command to count lines. Therefore, this first example counts the number of lines in `assignments.txt`. In the second example below, the command lists the files in the current directory sorted by size in descending order. For details on what the flags in this command do, consult `man sort`.

```
$ cd Shell1/Files/Feb
$ cat assignments.txt | wc -l
9

$ ls -s | sort -nr
12 project3.py
12 project2.py
12 assignments.txt
 4 pics
total 40
```

In the previous example, we pipe the contents of `assignments.txt` to `wc -l` using `cat`. When working with files specifically, you can also use *redirects*. The `<` operator gives a file to a Terminal command. The same output from the first example above can be achieved by running the following command:

```
$ wc -l < assignments.txt
9
```

If you are wanting to save the resulting output of a command to a file, use `>` or `>>`. The `>` operator will overwrite anything that may exist in the output file whereas `>>` will append the output to the end of the output file. For example, if we want to append the number of lines in `assignments.txt` to `word_count.txt`, we would run the following commmand:

```
$ wc -l < assignements.txt >> word_count.txt
```

Since `grep` is used to print lines matching a pattern, it is also very useful to use in conjunction with piping. For example, `ls -l | grep root` prints all files associated with the root user.

Problem 5. The `words.txt` file in the `Documents/` directory contains a list of words that are not in alphabetical order. Write the number of words in `words.txt` and an alphabetically sorted list of words to `sortedwords.txt` using pipes and redirects. Save this file in the `Documents/` directory. Try to accomplish this with a total of two commands or fewer.

Archiving and Compression

In file management, the terms archiving and compressing are commonly used interchangeably. However, these are quite different. To archive is to combine a certain number of files into one file. The resulting file will be the same size as the group of files that were archived. To compress is to take a file or group of files and shrink the file size as much as possible. The resulting compressed file will need to be extracted before being used.

The `ZIP` file format is the most popular for archiving and compressing files. If the `zip` Unix command is not installed on your system, you can download it by running `sudo apt-get install zip`. Note that you will need to have administrative rights to download this package. To unzip a file, use `unzip`.

```
$ cd Shell1/Documents
$ zip zipfile.zip doc?.txt
 adding: doc1.txt (deflated 87%)
 adding: doc2.txt (deflated 90%)
 adding: doc3.txt (deflated 85%)
 adding: doc4.txt (deflated 97%)

# use -l to view contents of zip file
$ unzip -l zipfile.zip
Archive:  zipfile.zip
  Length     Date     Time    Name
---------  ---------- -----   ----
     5234  2015-08-26 21:21   doc1.txt
     7213  2015-08-26 21:21   doc2.txt
     3634  2015-08-26 21:21   doc3.txt
     4516  2015-08-26 21:21   doc4.txt
---------                     -------
    16081                     3 files

$ unzip zipfile.zip
  inflating: doc1.txt
  inflating: doc2.txt
  inflating: doc3.txt
  inflating: doc4.txt
```

While the zip file format is more popular on the Windows platform, the `tar` utility is more common in the Unix environment. The following commands use `tar` to archive the files and `gzip` to compress the archive.

Notice that all the commands below have the `-z`, `-v`, and `-f` flags. The `-z` flag calls for the `gzip` compression tool, the `-v` flag calls for a verbose output, and `-f` indicates the next parameter will be the name of the archive file.

```
$ ls
doc1.txt    doc2.txt    doc3.txt    doc4.txt

# use -c to create a new archive
$ tar -zcvf docs.tar.gz doc?.txt
doc1.txt
doc2.txt
doc3.txt
doc4.txt

$ ls
docs.tar.gz

# use -t to view contents
$ tar -ztvf <archive>
-rw-rw-r-- username/groupname 5119 2015-08-26 16:50 doc1.txt
-rw-rw-r-- username/groupname 7253 2015-08-26 16:50 doc2.txt
-rw-rw-r-- username/groupname 3524 2015-08-26 16:50 doc3.txt
-rw-rw-r-- username/groupname 4516 2015-08-26 16:50 doc4.txt

# use -x to extract
$ tar -zxvf <archive>
doc1.txt
doc2.txt
doc3.txt
doc4.txt
```

Problem 6. Archive and compress the files in the `Photos/` directory using `tar` and `gzip`. Name the arhive `pics.tar.gz` and save it inside the `Photos/` directory. Use `ls -l` to see how much the files were compressed in the process.

Vim: A Terminal Text Editor

Today many have become accustomed to having GUIs (Graphic User Interfaces) for all their applications. Before modern text editors (i.e. Microsoft Word, Pages for Mac, Google Docs) there were terminal text editors. Vim is one of the most popular terminal text editors. While vim may

be intimidating at first, as you become familiar with vim it may become one of your preferred text editors for writing code.

One of the major philosophies of vim is to be able to keep your fingers on the keyboard at all times. Thus, vim has many keyboard shortcuts that allow you to navigate the file and execute commands without relying on a mouse, toolbars, or arrow keys.

In this section, we will go over the basics of navigation and a few of the most common commands. We will also provide a list of commands that interested readers are encouraged to research.

It has been said that at no point does somebody finish learning Vim. You will find that you will constantly be able to add something new to your arsenal.

Getting Started

Start Vim with the following command:

```
$ vim my_file.txt
```

When executing this command, if `my_file.txt` already exists, vim will open the file and we may begin editing the existing file. If `my_file.txt` does not exist, it will be created and we may begin editing the file.

You may notice if you start typing the characters may or may not appear on your screen. This is because vim has multiple modes. When vim starts, we are placed in *command mode*. We want to be in *insert mode* to begin entering text. To enter insert mode from command mode, hit the `i` key. You should see `-- INSERT --` at the bottom of your terminal window. In insert mode vim act like a typical word processor. Letters will appear in the document as you type them. If you ever need to leave insert mode and return to command mode, hit the `Esc` key.

Saving/Quitting Vim

To save or quit the current document, first enter last line mode by pressing the **:** key. To just save, type **w** and hit enter. To save and quit, type **wq**. To quit without saving, run **q!**

Problem 7. Using vim, create a new file in the `Documents/` directory named `first_vim.txt`. Write least multiple lines to this file. Save and exit the file you have created.

Navigation

We are accustomed to navigating GUI text editors using a mouse and arrow keys. In vim, we navigate using keyboard shortcuts while in command mode.

Problem 8. Become accustomed to navigating in command mode using the following keys:

Command	Description
`a`	**a**ppend text after cursor
`A`	**A**ppend text to end of line
`o`	Begin a new line below the cursor
`O`	Begin a new line above the cursor
`s`	Substitute characters under cursor

Table 1.6: Commands for entering insert mode

Command	Description
`k`	up
`j`	down
`h`	left
`l`	right
`w`	beginning of next **w**ord
`e`	**e**nd of next word
`b`	**b**eginning of previous word
`0`	(zero) beginning of line
`$`	end of line
`gg`	beginning of file
`#gg`	go to line #
`G`	end of file

Alternative Ways to Enter Insert Mode

Hitting the `i` key is not the only way to enter insert mode. Alternative methods are described in Table 1.6.

Visual Mode

Visual mode allows you to select multiple characters. Among other things, we can use this to replace words with the `s` command, and we can select text to cut or copy.

Problem 9. Open the document you created in the previous problem. While in command mode, enter visual mode by pressing the `v` key. Using the navigation keys discussed earlier, move the cursor to select a few words. Copy this text using the `y` key (stands for **y**ank). Return to command mode by pressing `Esc`. Move the cursor to where you would like to paste the text and press the `p` key to paste. Similarly, select text in visual mode and hit `d` to **d**elete the text and paste it somewhere else with the `p` key.

Deleting Text in Command Mode

Insert mode should only be used for inserting text. Try to get in the habit of leaving insert mode as soon as you are done adding the text you want to add. Deleting text is much more efficient and versatile in command mode. The `x` and `X` commands are used to delete single characters. The `d` command is always accompanied by another navigational command. See Table 1.7 for a few examples.

Command	Description
`x`	delete letter after cursor
`X`	delete letter before cursor
`dd`	delete line
`dl`	**d**elete **l**etter
`d#l`	**d**elete # **l**etters
`dw`	**d**elete **w**ord
`d#w`	**d**elete # **w**ords

Table 1.7: Commands for deleting in command mode

Command	Description
`:map`	customize
`:help`	view vim docs
`cw`	**c**hange **w**ord
`u`	undo
`Ctrl-R`	redo
`.`	Repeat the previous command
`*`	find next occurrence of word under cursor
`#`	find previous occurrence of word under cursor
`/str`	find `str` in file
`n`	find next match
`N`	find previous match

Table 1.8: Commands for entering insert mode

A Few Closing Remarks

In the next lab, we will introduce how to access another machine through the terminal. Vim will be essential in this situation since GUIs will not be an option.

If you are interested in continuing to use vim, you may be interested in checking out *gvim*. Gvim is a GUI that uses vim commands in a more traditional text editor window.

Also, in Table 1.8, we have listed a few more commands that are worth exploring. If you are interested in any of these features of vim, we encourage you to research these features further on the internet. Additionally, many people have published their `vimrc` file on the internet so other vim users can learn what options are worth exploring. It is also worth noting that we can use vim navigation commands in many other places in the shell. For example, try using the navigation commands when viewing the `man vim` page.