BASH: The Kind of Introduction I'd Have Liked

Steven E. Bopp, Materials Science and Engineering

September-October 2016, February 2018

Contents

> "There are not many persons who know what wonders are opened to them in the stories and visions of their youth; for when as children we listen and dream, we think but half-formed thoughts, and when as men we try to remember, we are dulled and prosaic with the poison of life. But some of us awake in the night with strange phantasms of enchanted hills and gardens, of fountains that sing in the sun, of golden cliffs overhanging murmuring seas, of plains that stretch down to sleeping cities of bronze and stone, and of shadowy companies of heroes that ride caparisoned white horses along the edges of thick forests; and then we know that we have looked back through the ivory gates into that world of wonder which was ours before we were wise and unhappy."
>
> — H. P. Lovecraft, Celephaïs

1 Introduction

In general terms, as it relates to computing, a shell is a user interface for an operating system's services. In order to view this document on a computer, you are probably using the explorer.exe shell if you are on Windows or the unity shell on Ubuntu Linux. These shells are graphical user interfaces and, chances are, you already know how to use them. This document will attempt to teach the reader how to use a terminal shell, like Bash in the stead of, or in addition to a GUI shell.

The primary focus of this document will be an introduction to command line use and scripting in the Bash shell for Linux and UNIX (specifically Darwin) machines. This guide is by no means meant to be exhaustive and absolutely no warranty is offered with this document.

As it stands, Bash is one of, if not the most widely used shell on Linux and UNIX machines. Using the shell from a terminal offers a user much greater access to the machine and its contents, especially when run as an administrator or a root user (the user on the computer which has full access to everything on the machine).

In addition to the obvious benefits of controlling totally the processes on a machine, uses for the shell include automating system tasks, running an incredible variety of programs, downloading them from trusted sources and compiling them all through one terminal window.

1.1 Motivation for This Text

Somewhere around my sophomore year in college, the mechanical hard drive of my very first (and about six year old at that point) laptop died spectacularly, taking my files and wildly generic super, super, super awful operating system which shall not be named, along to the clicky, crunchy, hardware failure grave. Intending to never again spend a penny on the operating system that shall not be named, I bought a new disk and hopped on the linux train.

2 Shell

2.1 The Bourne-Again Shell (Bash)

2.2 Accessing the Shell

There are many shells, worthy of mention, beside Bash, are the following: sh, csh, zsh and fish (the former being one of this author's favorites especially for Darwin machines). Shells can be accessed through a command window called the terminal (sometimes called konsole or similar on other operating systems). In Ubuntu Linux and Darwin UNIX, Terminal is the default command interpreter for the shell. Ubuntu users can easily access the terminal with the keyboard shortcut Ctrl+Shift+T or from the start menu; Darwin users can access

the terminal from the spotlight search or the Utilities directory in their Applications directory.

Whatever way you access the terminal, the end result will be similar. To list what shell you are using in the terminal, type the following:

```
echo $0
```

As explained, this will return the shell you are using. In order to switch shells (assuming that they are installed) a user need only enter into the terminal the name of the shell he or she wants to run. For example, if one wanted to run the c-shell, he or she would need only type the following into the terminal:

```
csh
```

To exit this shell, back to the default shell, one only needs to enter the following:

```
exit
```

2.3 TTYs

2.4 Beyond Bash

3 Shell Commands

Running commands in the shell is made to be extremely simple, if you've been following this guide, then you've already successfully run several commands...no biggie eh?

One of the most helpful of all the commands is the manual command, as its name implies, it will open the user manual for a specific command. For example, run the following:

```
man bash
```

As is visible in the terminal window, you have been pulled into a program which shows the user available information for the Bash shell. Every program which has been installed on the computer and is accessible through the terminal should have a manual page. This author cannot stress enough how helpful this will be for a user in his or her future. Enter q to exit this manual page.

3.1 Basics of the Bash Shell

Herein the next sections will be supplied to the reader a list of several necessary tools for successfully using the Bash shell (or most other shells for that matter). Do not be daunted by the several pages that these sections will occupy, all of the tutorials contained in section 3.2 are simple and should take less than 30 minutes to complete, start to finish.

After completing this section a user should be able to do all of the following: navigate their system in the shell; create, open and delete files and directories (as well as archives); read, write and destroy files; set and change the ownership

or the mode of a file; and run programs as the root user or the fake root user (a command whose level of usefulness is devious).

At any time, it is helpful for the user to know the following: first, the command:

```
!!
```

will repeat the last input command to the terminal; second, using the up and down arrows in the command processor mode of the terminal will navigate a user through the history of recently issued commands. Pressing the up arrow once and then enter will have the same effect as running the '!!' command. Additionally, using the tab key will tell the shell to attempt an auto completion of what you are typing; for example, if your directory contains a file called steven.txt and one were to type 'st' and then press tab, the shell would attempt to insert the remainder of the file name, assuming that there are no other files starting with 'st.'

3.1.1 Navigation in the Shell

In order to navigate your way around in the shell, you need to use text commands, just like almost everything else in the shell. There are, however workarounds for the navigation explained herein, these will be explained at the end of this section.

Enter into the terminal the following:

```
pwd
```

This program is called print working directory and it will do just that. Bash will output the location of the directory in which you are currently located. This will be a /home/user directory for an Ubuntu user or a /Users/macuser type directory for a Darwin machine.

In order to navigate one directory closer to the root directory, enter into the terminal the following:

```
cd .. ; pwd
```

You have just run two commands one after the other: cd .. navigated you 'up' one directory; pwd lists for you your new location. Now, enter the following into the terminal:

```
cd /
```

You are now in the root directory, this can be evidenced with pwd at any time. Using the cd / command is similar to the bare cd command which sends you to your home directory. Let's say that you're a curious user and you want to see what files and folders are in your root directory. Enter the following into the terminal:

```
ls -a
```

What have you done? All together, you have listed all of the directories and files in the root directory. Now try running the command above again but

without the '-a' tag. You'll notice that a variety of items which were prepended with a '.' are now missing. These are called hidden files and hidden directories, the '-a' tag has allowed you to list all of these items, regardless of their having been prepended with a '.' or not.

How do you know what a certain file or folder is, especially if your terminal does not highlight types with different colors? Fear not, the following command will never fail to help:

```
file bin/
```

You have just executed the 'file' command on the bin/ directory. The Bash output should be something like: "bin/: directory." In order to navigate into one of the directories which you can see, it is as simple as this:

```
cd /bin
```

List out the files and you can see that you are in a directory which contains programs that can be run with the shell. Do you see our friends ls, pwd and Bash? With the cd command, navigate back to the home directory and enter again into the command window the following:

```
cd
cd /bin
```

With pwd you can see that you are again in the /bin directory, in the same way, you can access any directory in your computer (assuming that you have the correct permissions) with the cd command.

There are easier ways to navigate and explore files in the terminal, one such shining example is the program called Ranger. Ranger is a file explorer written in a language called ncurses which allows for extremely efficient exploration of files and directories on your system. To run Ranger, you need to install the program, if you are an Ubuntu (or similar OS) user, this can be accomplished easily with the following command:

```
sudo apt-get install ranger
```

(Sudo is a program called supervisory user do, it is extremely powerful and can damage your computer's system if used incorrectly, don't let this deter you though; we'll talk more about it later) If you are a Darwin user, then you will need to enter the following:

```
brew install ranger
```

Brew is part of the homebrew is a package manager for Darwin which is similar to apt-get; to install it, search the Internet for the Homebrew Package Manager and follow its instructions. Other systems, such as distributions based on Fedora Linux and Red Hat will need you to substitute the 'apt' command for the 'yum' command.

The use of Ranger and other programs will be explained in the next section.

3.1.2 Create, Rename, Copy, Move and Delete Files and Directories

Using the shell, it is facile to create a file (let's call the file file.txt); this can be done with the following command:

```
touch file.txt
```

Upon checking the contents of the current directory, one will find that the file called file.txt now exists. Now try running the following command:

```
mv file.txt .file.txt
```

Listing again the files in the directory will not show the file, adding the '-a' tag to the 'ls' command will reveal the file—you have modified the file by renaming it; additionally, you have turned it into a hidden file. The command can be reversed to rename the file to what it was previously. This is one of the two uses for the move (mv) command in the shell. Renaming the file back to file.txt will be left as an exercise for the reader.

Issue the following command to the terminal:

```
mkdir text_files
```

List the contents of your current directory and you will see a new directory called text_files (the character is used because spaces are more difficult to call in the shell, one would need to subpend each separate word in the file name with another backslash).

Now issue the following command:

```
cp file.txt copy.txt
```

Listing the files in the directory, a user will find that there is a new file called copy.txt; this is the power of the cp command, to make copies of files and name them in the same line of text.

Enter the following command:

```
mv file.txt text_files
```

Again list the items in your current directory and you will see that the file.txt file is no longer present. You have in fact moved the file into the previously created directory. This is the second use of the move (mv) command. In the command, the target directory can be any which you want (assuming that you have the proper permissions).

Navigate into your newly created directory and issue the following command:

```
rm file.txt
```

Upon listing again the items in the directory, one will find that there are no contents. What you have done is use the remove (rm) command to remove the file. It is worthy of note that the 'rm' command does not overwrite or destroy the file, it only erases the link between the file and the shell so that it is difficult (but not impossible) to access. File destruction techniques will be explained in section 3.1.4.

Navigate back to the parent directory (try using 'cd ..') and list its contents, you should again see the text_files directory. Now issue the following command:

```
rm -r text_files
```

List again the items in your current directory and you will see that the text_files directory is missing. What you have done is erase the text_files directory. You used the '-r' tag because the vanilla 'rm' command will not erase files and directories recursively.

3.1.3 Create and Unzip Archives

Here we will see how to store and extract files from a .tar archive as well as use the programs called gzip and gunzip. The discussion here will be limited to the .tar (tape archive) files and the .gz and .bz2 compression formats. Here, the reader is encouraged to explore the manual page for the program called tar, this can be easily reached with the following command:

```
man tar
```

To pack files into a .tar package, it is as simple as issuing the following command (omit the above if you wish, substituting files for what ever is necessary):

```
touch file1.sh file2.sh file3.sh
```

tar -cf archive.tar file1.sh file2.sh file3.sh

If you list out the contents of your directory, you will see that there are four new files, one of which being appended with the .tar format; this file is the package which you have just made. To zip the file, one must only issue the following command:

```
gzip archive.tar
```

Listing the contents again will show that the file has again been appended with a new format, this time its name will be archive.tar.gz; the .gz format is that of the gzip program.

To un-do all of the commands which you have just done, issue the following commands:

```
rm file1.sh file2.sh file3.sh ; gunzip archive.tar.gz ; tar -xvf
    archive.tar
```

What you have done is the following: first, you deleted the three original files; next, you unzipped the .gz file with the program called gunzip; finally, you unpacked the .tar archive (adding the -v tag will print the operations to the terminal) and got back all of the three files which you made in the beginning. Listing the contents of the directory will confirm this.

If, as you will in the future of this document, you are to run into a file appended with the format .tar.bz2, you can use the same program (tar) to unpack the files, you must just add the -j tag to the list of options you choose; an example is included here for a theoretical file called archive.tar.bz2:

```
tar -xvjf archive.tar.bz2
```

3.1.4 Reading, Writing and Destroying Files

When a user spends time in the shell, he or she will find it convenient to view and edit text in files, as well as securely deleting them (shredding or destroying). There is a program called 'sed' which allows a user to edit text without another program to aid them, 'sed' is beyond the scope of this section.

There are two primary ways to view text, only the first of which will allow you to edit. The first of these is the use of text editors (like textedit on Darwin systems or gedit on Ubuntu and similar systems), the second is the use of operations which read text from a file and print that text to the terminal.

Find for yourself, if you will, a text containing file whose length is considerable, maybe more than 40 lines and move or copy it into your current directory (let's call the file text.txt).

Let's try to edit the file called text.txt; to do this, enter into the terminal the following command:

```
nano text.txt
```

In your terminal, you should see the entire body of text contained in the file called text.txt, additionally you will see a list of possible commands at the bottom of the screen and the label 'GNU nano' at the top left.

Currently, you are in the terminal-based text editor called nano. You can read, write to and delete from the document. Enter ctrl+o to save changes and ctrl+x to exit the nano program (in the shell, the control key is abbreviated as '^' so, ctrl+o would be abbreviated as '^o').

There are many other programs which are made to edit text, such as pico (closed source), vi and vim and emacs. These programs are beyond the scope of this section.

Alternative to nano, there are many ways of viewing text without entering another program or editing the text—helpful when one wants to just view the contents of a file. Enter into the terminal the following commands:

```
cat text.txt ; head text.txt ; tail text.txt
```

In your terminal, you should have seen first the entire text of the file printed out, then the top ten lines of the file and then the bottom ten lines of the file; these are, respectively, the functions of the cat, head and tail commands, all of which are extremely helpful.

Finally, it is necessary, or the purpose of security, to be able to securely delete files, not just remove the link which allows the shell to access the and mark the to be overwritten by new files. There are several programs which can do this effectively, but there are none that this author trusts more than the Linux shred command.

Let's again ise the example of a file called text.txt. Make sure that such a file is indeed in your current directory and run the following command:

```
cp text.txt delete.txt ; rm –Pv file.txt
```

Above, you copied the file text.txt, naming the copy delete.txt. Next, you ran the rm command with the 'P' tag (overwrite files before unlinking) and the

'v' tag (verbose). It is unknown to this author how effective the 'P' option is relative to the two commands which will follow (although it is assumed to be less effective).

On Linux machines, chances are good that the program called 'shred' will be either installed or easy to install from the terminal. On Darwin machines, the program called 'srm' should come pre-installed with the machine.

Make of the text.txt file with the same command as above.

Now, if you are running a Linux operating system, run the first of the two following commands; if you are running a Darwin machine, run the second of these two commands:

```
shred -uvz delete.txt
srm -mvz delete.txt
```

In both cases, the file called delete.txt will have been overwritten, written completely with zeros and then unlinked. In both cases, the 'v' option was added so that the shell would print out what it was doing and the 'z' option was added to write the file completely with zeros before truncation.

It should be known that, with sufficient time and computing power, these files can be reconstructed, the time and power necessary for such a reconstruction being directly proportional to the number of overwrite steps and time spent overwriting. It is beneficial for the reader to explore the manual pages of these programs before their use.

3.1.5 The Wildcard Character

The wildcard character * is a way for the user to define the argument of a shell command more generally. That is to say the wildcard adds an element of 'whatever' into a command. If it pleases the reader, run the ls command and then run the following:

```
ls *
```

The reader will find that the common ls command did its job and showed the unhidden contents of the home directory. Of more interest is waht's happened when the wildcard character was added to the command: contents of the current directory are listed as well as the first order contents of all the directories in the working direcotry.

Next let's try making several files with different formats. Consider using the following (it's going to be helpful to run this command in a new, empty directory, for simplicity let's call it 'directory'):

```
touch {a..c}.txt {a..c}.dat {a..c}.sh ; ls
```

You've created nine new files of three different extensions using the touch command and then listed the contents of the working directory. But what if there were an arbitrary number of items in the directory and only the ones starting with a certain letter or ending with a certain extension are relevant? At the reader's leisure, run the following:

```
ls a*
```

```
ls *.sh
```

```
ls *.*
```

The terminal will print first a list of all the files starting with the letter a (lower case exclusively) regardless of any information following the first character. Next, the terminal will print a list of all the files that are .sh extended regardless of any information before the .sh. Finally, the terminal will list all of the files in the current directory. A discerning reader can tell that the wildcard operation is excceptionally handy when performing operations on multiple objects in the same command. Moving, removing, editing, and much more are accessable with the wildcard. Delete it all (be careful to perform this operation only in a dirctory that has no files that the reader would miss if they were deleted), try the following:

```
ls ; rm * ; ls
```

The reader will find, after looking at the obvious differences between the lists given before and after the 'rm *' operation that there are no longer any objects in the working directory. If the reader had exceuted mv * .. instead, then all of the files in the working directory would be moved one directory up in the tree (assuming that such a directory exists).

3.1.6 Ownership and Change Mode

If, say, someone were to want to set ownership parameters for a certain file, let's call it file.txt, then he or she would need to use the change ownership function which is a built-in for the Bash shell. Enter the following into the terminal:

```
touch file.txt ; sudo chown steven file.txt
```

What we have done is the following: first, we created a file called file.txt; next, we modified the ownership parameter (with chown) of the file such that the user called 'steven' is set as the owner of the file.

Ownership is important on a system for several reasons, one of the most important is that many system files are set to be owned by the root user (discussed in section 3.1.6), by default a current user will not find him or herself logged into the root account unless several steps have been taken. Modifying root-owned files and directories is generally impossible unless one is logged into the root account or one uses the sudo (to be discussed in section 3.1.6) command. Modifying the ownership of such a file can make modifying it simpler.

Change mode is another program which the reader will find to be of enormous value in the coming sections (mainly in executing their own or downloaded scripts. Change mode is the program called chmod and it is used to change the file mode or the access control list. That is to say, chmod allows a user to define what users can do what with a file. Run the following command in the terminal:

```
touch scrip.sh ; chmod 755 script.sh ; ./script.sh
```

Here is what you have done: the first section, as you should know, created a file called script.sh; the next section changed the mode of the file to 755, this means that the owner can execute the file, read the file and write to the file but other users can only read and write to the file; finally, you executed the file called script.sh (using the ./ command will execute a file).

Congratulations, you have just made your very first shell script; while it may not have done anything interesting, it is the first step to shell scripting—an extremely powerful tool. The same could have be accomplished if the reader had replaced the 755 with +x where x is the executable tag. To demonstrate the smilairty between 755 and +x, the reader can compare the result of the previous command to that of the following.

```
touch scrip2.sh ; chmod +x script2.sh ; ./script2.sh
```

3.1.7 Su, Sudo and Fakeroot

Often, when running programs or commands in the shell, one may find that only users with administrator (or root) permissions on the system will be able to perform certain commands or access certain files. For the purposes of security, it is ill-advised for a user to be signed permanently into the root account, so the question becomes: how do I run administrator commands?

One will find that the command to operate as the root user is called 'sudo,' an acronym for supervisory user do (some people pronounce it like 'sue-do,' which is the technically correct way but fun people say it like 'pseudo'). Chances are that, if you are running a Linux system, you have already used the sudo command for installations. Try entering into the terminal the following:

```
sudo -i
```

The terminal will require the administrator password; when it is entered, the user will find that they have been logged into a new account called root@user where user can be replaced by the hostname of the user. This is, however, not usually recommended because of security issues and possibilities of damage to the system by a less experienced user. To log out of the root account, enter the following into the terminal:

```
exit
```

Running commands with root privileges without being signed into the root account is very simple and can be accesses with the sudo command in the following way:

```
sudo echo "You're running this as the root user!"
```

For those intending to run interactively as root, that is to be free of the relative burden of entering the sudo command every time that something needed to be run as the root user, one can use the 'su' command. It's really just as easy as keying in the following command and giving your password, however the su command is generally considered much riskier to run that the sudo command. There are a lot of benefits of using sudo over su but for the general user, its

mainly a concern of the extra time it takes to type sudo every time you run a command which can make changes which may be difficult or impractical to reverse giving the user more chance to contemplate whether they're doing exactly what they want to do. The exit command will again remove the user from the root mode.

```
su
```

A similar command to sudo is called 'fakeroot.' A user will find that the fakeroot command does not come pre-installed on must systems, use the appropriate command to install it now. Run the following:

```
fakeroot
```

A clever user will find that they have been logged into an account which is analogous to the root account but has no actual root permissions. The devious beauty of fakeroot is that, for many operations, it is enough for the system to just look like the user is the root user instead of actually being signed into the root account.

Exploration of the manual page for the fakeroot command will show its myriad uses which will not be enumerated here, suffice it to say however that there's a lot of fun to be had with fakeroot.

3.2 Running Shell Programs

Running programs in the shell is simple, if you have indeed installed the Ranger file explorer into your shell as shown in the previous section, then enter the following into your terminal to execute the program:

```
ranger
```

Use the arrow keys to navigate between directories and files without the use of the cd command. This program is extremely helpful and more helpful, in many cases, with the exploration of files than a typical graphical user interface shell.

Ranger is by no means the only program which can be installed on Linux and UNIX systems. Some of this author's favorite programs will be discussed in the remainder of this section.

3.2.1 top, htop and atop

Top is a process viewer which will, more likely than not, be installed on your system currently. Htop is, as explained by the developers, "An interactive process viewer for Unix [and Linux]." Atop is another process viewer/monitor that is more useful for servers and clusters than are htop or top. Install and run htop with the following command (users on Darwin or Fedora-like systems will have to make substitutions for the 'apt-get' command with 'brew' or 'yum' respectively):

```
sudo apt-get install htop ; htop
```

Use the arrow keys to navigate in htop or enter 'q' to quit htop. The program has its own instructions which will not be repeated here, I invite you to use the manual page for htop in order to see its full capabilities.

Top and atop have differing features but, as this author is concerned, htop is the most helpful, reconfigurable and visually appealing.

One excellent use for all of these programs is their ability to kill existing processes running on your computer while giving you a real-time update of such programs system load, memory and CPU percentages and their parent processes.

3.2.2 whoami, uname and hostname

About whoami, uname and hostname, there is little to be said. Simply, whoami will print the user ID of the account which a user in currently logged into.

Uname is slightly more interesting than whoami, upon exploring the manual page, the reader will find that uname will, depending on the option he or she selects, uname will supply a user with information about the computer and the current user. For instance, uname can tell you the kernel which you are running, the operating system and other pertinent information. The most helpful of all these commands is the following:

```
uname -a
```

A user will find that all of the pertinent information about the system, in brief terms, has been printed to the terminal.

Hostname has interesting utility inasmuch as it will tell a user his or her IP address on the sub network to which that person is attached. Consider the following commands:

```
hostname ; hostname -I
```

The first of these two commands will return the name of the user on the system, the second of these commands will return all of the IPv4 addresses attached to the host. The latter is very helpful for connecting two computers on the same network via SSH.

3.2.3 bmon, nmap, arp and ping

There exist many network tools for the Linux and UNIX shell however, three of the best are bmon, nmap and arp.

Bmon is a bandwidth monitor which gives the user information about receive and transmit bits on a per-interface basis as well as printing to the terminal a semi-GUI of the network traffic in real time (many features not listed).

Nmap, standing for Network Map, is an excellent tool which can be used to determine who is on a network (e.g. the network map), what device they are using and what operating system that device is using.

Arp, standing for Address Resolution Protocol, will print the sub-net addresses of all the devices running on the network to which you are attached.

Installing these programs is simple and follows the installation rules as above, listed again here for convenience:

```
sudo apt-get install program
```

brew install program

sudo yum install program

Specifics of these tools can be found in their respective manual pages but some interesting features will be listed here. We will start with bmon, the simplest of all the programs in this subsection. Run in the terminal the following:

```
bmon
```

Press on your keyboard the letters 'd' and 'i,' these will bring up detailed network statistics as well as a real-time pseudo-GUI of the network receive and transmit load as a function of time.

Nmap is a simple program to run and the reader is strongly encouraged to read the manual page for nmap and, for that matter, any program that you run. After installing nmap in the normal way, run the following command:

```
nmap -sn 192.168.0.1/24
```

This program will list the used IPv4 addresses being used on the network to which the user is attached, the address in the command can be substituted for any valid IPv4 address. Nmap has excellent utility in the field of network exploration and port scanning.

Arp is the address resolution protocol and is excellent for determining information about the media access controls of devices attached to a network; additionally, arp can manipulate the IPv4 network cache inasmuch as that it can add, manipulate or delete entries from the network neighbour cache. A simple way to start using arp is to issue the following command (this will hardly scratch the surface of the program's capability):

```
arp -a
```

Ping is a helpful command which issues packets to a known gateway that elicit a response from that gateway, the response will be printed with relevant statistics onto your terminal. To exit this command the use of ctrl+c will be helpful. Consider running the command:

```
ping 8.8.8.8
```

This command will send packets to the 8.8.8.8 Google DNS server and receive back a response. Among its many applications, Ping is particularly helpful when configuring clusters for checking whether nodes are online, and for generally determining whether a connection to a particular server or address is accessible.

3.2.4 Cryptography

Generally speaking, cryptography is beyond the scope of this document. There are however several helpful, convenient, and fun examples of cryptography in the shell.

3.2.5 No Hangups and Forks

As can be seen in the previous sections, it is infinitely useful to run programs in the terminal. Running programs in the terminal can have drawbacks however; one such case is that the terminal must remain open while the program is running if special actions are not taken. These actions are called no hangups and forks. Let's take, for example the case of running PyRoom (one of this author's favorites).

Run the program with the following (assuming that it is installed on the system):

```
pyroom
```

Now close the terminal window and watch PyRoom close along with it. This is a two-edged sword: if a program which has a high propensity for crashing or other catastrophic failures, it can be helpful to close the terminal in order to close the program; other times, when a program is running in the terminal and you want to continue using that terminal, a user might find him or herself our of luck.

The workarounds for these problems are called no hangups and the fork. No hangups is a program which ignores problems (like closing the terminal when a program is running a.k.a hangups) when a program is running inside of it, no hangups will not give your terminal back to you before the program is closed. The fork is different. Forking a program means that you will run the program you want and get your terminal back immediately (that is you can issue new commands in the terminal which is running an old command); you will see, however that, when you close the terminal, any programs running in it will be closed as well.

Now let's try running PrRoom with no hangups, the fork and a combination of each. Enter into the terminal the following:

```
nohup pyroom
```

Now try closing the terminal in which you have run the program, you will find that PyRoom will stay open even though the terminal is closed: this is the power of nohup (the command to run with no hangups). Close PyRoom and enter into the terminal the following:

```
pyroom &
```

Try entering a new command into the terminal, you will see that the terminal is available to process new commands; if, however, you closed the terminal, you will see that PyRoom will close with it. This is the power of the fork (the fork is entered into the terminal after a command as the & option).

An excellent usage of both these programs simultaneously is to call them on the same program, this can be done in the following way:

```
nohup pyroom &
```

Running the above command will run PyRoom without hangups and fork the program, immediately giving you back access to the terminal.

3.2.6 Find

Find allows the reader, as is said in its man page, to "...Search for files in a directory heirarchy." Find is a particularly useful program in many instances where analogous operations with the ls program would be cumbersome. A good instance of this is to list all of the files in the working directory which are executable. Refer to the Ownership and Change Mode section of this document and create an executable file, or multiple if that's more interesting. Now run the following:

```
find . -maxdepth 1 -perm -a+x -type f
```

The reader will find that the newly created executable(s) as well as any pre-existing in the working directory have been printed to the terminal. Elements of this command are the -maxdepth option whcih specifies how many subdirectories will be be searched for the argument (1 searches only the working directory), the perm option accepts an argument of what the permissions the files that find is looking for should have (the a+x can be replaced by 111 whcih is just to say that only a special type of user can execute the file), and the -type f is a cost-based optimizer looking for regular files (can be replaced to search for other file types, see manual).

3.2.7 Service (in Linux)

3.2.8 wget

Downloading large numbers of files from online repositories manually can quickly become excessively tedious. Wget is a "GNU utility for downloading network data," as per the program's manual. Wget is one of the most popular programs for downloading infomration off of FTP servers but it works excellently on HTTP and HTTPS protocols as well. Non-interactive and link-following operation make the program especially helpful so that download processes can be run in the background or while you're away at lunch. Database collectors rejoice, we're going to learn today.

Let's try an example to download an online database of crystal structure information on the Strukturbericht designations. The site is hosted by Duke university and has a wealth of useful data files for use in materials science. However, navigating to each page and individually downloading the files would be laborious and I certainly don't want to spend all of that time to get the files that I want. Run the following code in your terminal, if wget is not installed then install it the usual way. Wget is smart enough to make its own directories and sub directories, if all goes well you should be able to browse your favorite crystal structure and pseudopotential databases offline (disk space beware of the latter).

```
wget -r http://aflowlib.duke.edu/users/egossett/lattice/struk/index
    .html
```

Wget should have run and printed a large number of operations to the terminal consisting of what file it is downloading, where it is being downloaded from,

where it is being placed on your disk, and a nice status bar of the download progress. The -r tag specifies a recursive download e.g. grab the files from the directory you specify and those related to it.

4 Redirects and Pipes

Redirects and pipes underpin the incredible modularity and versatility of the shell, what I consider to be its most valuable attribute. Redirects are commands instructing shell programs to output into a separate (either extant or created at the time of the redirect) file; redirects can also work in reverse, e.g. reading a file to a program. Pipes are slightly different and are intended to transform command or program outputs into inputs for other commands or programs. Daisy chaining redirects and pipes lends itself well to modular programming and shell scripting as we'll explore in further sections.

4.1 Redirects

4.2 Pipes

5 Variables and Structures

6 Shell Programs

I generally separate shell commands from shell programs on a case by case basis, others may disagree, but I know it when I see it. Programs, I think, have a broader utility than a command, its maybe analogous to clicking your application launcher and actually opening an application: the former I'd consider to be a command and the latter a program. Some shell programs have already been covered in the commands section because I felt that they would be more fittingly placed where they are. In the following subsections, we'll cover some shell programs that I think have great value and personally use regularly.

6.1 GNUPlot

6.2 Lynx

6.3 Htop

7 Shell Scripting

Shell scripting is wildly useful in numerous situations. Mainly, as we've seen previously in this text, the shell is used to reduce the total effort required to perform what might be otherwise excessively tedious operations if performed with a graphical user interface (GUI), mouse, and keyboard. We've already seen the utility of shell in perfroming such tedious operations in the wget section. In

several computational physical sciences, a very common implementation of shell scripting is calculating properties of materials with molecular dynamics (MD), density functional theory (DFT), (I shutter to say it but) GW, and all sorts of quantum chemistry, theoretical spectroscopy, and others I'm undoubtedly forgetting.

In the next few subsections, we'll explore some shell scripts that I think are worthwhile for a variety of reasons. The first will be the construction of a text-based user interface to demonstrate how commands can be quickly stored locally and recalled to circumvent the doldrums of typing the same string every time wants to re-perform a common command. The second will be a DFT calculation script (don't freak out, DFT won't be the main point here) for the popular code Quantum-Espresso where we'll learn to implement a series of commands that create their own input files, call external programs, and return a time stamp after each operation (or at arbitrary places in the script) for measuring the elapsed time it takes to run a script.

7.1 Make a Text-Based User Interface (Think curses)

User interfaces (UIs) are clearly ubiquitous but design of a graphical UI can be tedious and unnecessary for many applications. Consider the older-style BIOS text-based user interface that many of us are accustomed to, it can be navigated quickly and clearly understood without having nearly any footprint on system resources. Curses, and, this writer's preference ncurses, are text-based user interface (TUI) libraries that are very useful for programs which need some user instructions but where a GUI might be overkill. There's clearly also some strange sort of style points (at least in this author's opinion) to be awarded for the use of ncurses.

Let's consider how to make a text-based user interface with the shell using a function, some variables, and a whole bunch of echo statements.

```
#!/bin/bash

# menu.sh
# Written by Steven Bopp on 18 May 2016

function selection () {
echo -e "\n" # Add a new line with \n
echo -e "Enter your selection \c" # Surpress a new line with \c
}

function menu () {

clear # Erase the previous input each time this block is run
        # Display menu text
        echo "=============================================="
        echo "====== Programs and Tools Launch Menu ======"
        echo "=============================================="
        echo "Enter 1 to launch programs"
        echo "Enter 2 to access system tools"
        echo "Enter 3 to manage linux packages"
        echo "Enter q to exit this menu"
```

```
        selection

    read answer_zero
    case $answer_zero in # Start primary switch case block

1) # First case: this block launches the program launch menu
        clear
        echo "=================================================="
        echo "========== Program Launch Menu ==========="
        echo "=================================================="
        echo "Enter 1 to launch VESTA"
        echo "Enter 2 to run nano on an existing file"
        echo "Enter 3 to run the Ranger file explorer"
        echo "Enter q to exit the menu"
        selection

    read answer_one
    case $answer_one in
        1) # This block executes VESTA in Ubuntu.
            cd /home/steven/Documents/Programs/VESTA/VESTA-x86_64
            ./VESTA ;;
        2) # This block will execute nano on a user input file name
            echo -e "Enter the name of the file \c"
            read textfile # Read user input and store as $textfile
            echo "You are now editing $textfile"
            nano $textfile ;; # Execute nano on $textfile
  3) # Execute Ranger
      ranger ;;
        q) # This block executes the exit command from this menu
            exit ;;

    esac # End program launch sub-menu switch case block
    read input_one
            ;; # End program launcher

2) # Second case: this block launches the system tools sub-menu
        clear
        echo "=================================================="
        echo "======== System Tools Launch Menu ========"
        echo "=================================================="
        echo "Enter 1 to launch htop"
        echo "Enter 3 to run Nmap on a specific IPv4 address"
        echo "Enter 4 to run Address Resolution Protocol"
        echo "Enter 5 to restart the network-manager service"
        echo "Enter q to exit this menu"
        selection

    read answer_two
    case $answer_two in
        1) # This block launches htop
            htop ;;
        3) # Execute Nmap on a user input IPv4 address
            echo -e "Enter the IPv4 address \c"
            read nmap
            echo "Nmap will now scan the given IPv4 address: $nmap"
            Nmap $nmap ;;
        4) # Run local network IPv4 Address Resolution Protocol
```

```
            echo "Host address:"
            hostname -I
            echo "Network addresses:"
            arp -a ;;
        5) # This block will restart the network-manager service
            sudo service network-manager restart ;;
        q) # This block executes the exit command from this menu
            exit ;;

    esac # End of system tools launch sub-menu
    read input_two
            ;; # End system tools sub-menu

3) # Third case: this block manages packages on linux
        clear
        echo "================================================"
        echo "========== Manage Linux Packages ============"
        echo "================================================"
        echo "Enter 1 to see install policy of a program"
        echo "Enter 2 to search for installed printer packages"
        echo "Enter 3 to delete configuration and/or data files + dependencies of a package in ubuntu"
        echo "Enter q to exit this menu"
        selection

    read answer_three
    case $answer_three in
    1) # Run sudo apt-cache policy on a program
        echo -e "Enter the name of the program \c"
        read program    # Read user input and store as $program
        echo "You are now editing $program"
        apt-cache policy $program ;; # run on $program

    2) aptitude search printer | grep ^i ;;
    3) echo -e "Enter the name of the program \c"
        read programa   # Read user input and store as $programa
        echo "You are now purging $programa"
            sudo apt-get purge --auto-remove $programa ;;
    q) exit ;;

    esac
    read input_three
     ;; # End of package manager

q) exit # Fourth case: execute the exit command on the menu
            ;; # End menu exit command command

    esac # End primary switch case block
    echo -e "Enter return to continue \c"
    read input_zero # New variable called input from the case statement which is displayed

}


while true
do
```

```
    menu
done

```

7.2 Implement Execution Options with Getopts

```
#!/bin/bash

# options.sh
# Written by Steven Bopp on March 28, 2019


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-Banner Variables:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-Thanks to kammerl.de/ascii/AsciiSignature.php-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:standard font for ASCII art:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

WELCOME="
__        __   _
\ \      / /  | |
 \ \ /\  / /__| | ___ ___  _ __ ___   ___
  \ \/  \/ / _ \ |/ __/ _ \| '_ ' _ \ / _ \
   \  /\  /  __/ | (_| (_) | | | | | |  __/
    \/  \/ \___|_|\___\___/|_| |_| |_|\___|
          Continue to menu? y/n
"

HELPMENU="
   __  __                        _
  |  \/  | ___ _ __  _   _      ___| |__
  | |\/| |/ _ \ '_ \| | | |    / __| '_ \
  | |  | |  __/ | | | |_| |  _ \__ \ | | |
  |_|  |_|\___|_| |_|\__,_| (_) |___/_| |_|
 _   _      _         __  __
| | | | ___| |_ __   |  \/  | ___ _ __  _   _
| |_| |/ _ \ | '_ \  | |\/| |/ _ \ '_ \| | | |
|  _  |  __/ | |_) | | |  | |  __/ | | | |_| |
|_| |_|\___|_| .__/  |_|  |_|\___|_| |_|\__,_|
              |_|

       Access the menu with ./menu.sh

    Simple args are -m -i -h
   For info, run ./menu.sh -i
   For menu, run ./menu.sh -m
   For help, run ./menu.sh -h

   More specific args with -x
  For single-use menu, run ./menu.sh -x menu
       For help, run ./menu.sh -x help
       For info, run ./menu.sh -x help
"

INFO="
```

```
 __   __                              _
|  \/  | ___ _ __  _   _        ___| |__
| |\/| |/ _ \ '_ \| | | |      / __| '_ \
| |  | |  __/ | | | |_| |  _  \__ \ | | |
|_|  |_|\___|_| |_|\__,_| (_) |___/_| |_|
      ___         __
     |_ _|_ __  / _| ___
      | || '_ \| |_ / _ \
      | || | | |  _| (_) |
     |___|_| |_|_|  \___/

      Menu.sh version 2.0
Created by Steven E. Bopp on March 17, 2019:
For a curses-like text-based user interface
       Materials Science & Engineering
"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-begin getopts-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:Thanks to the Run.sh program in /tests of the:-:-:-:-:-:
#-:-:-:-:-:-:-:-:WanT code for teaching me getopts:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

ALLOWED_ACTION="help info menu" # -x * options allowed to be called
ACTION=
LIST=

# Begin getopts primary while loop
while getopts :hmwix: OPT # Define possible arguments; Careful, *: 
    requires and argument
do
  case $OPT in
  (x) ACTION="$OPTARG" ; shift 2 ;;
  (h) echo "$HELPMENU" ; exit 0 ;;
  (m) while true
  do
    menu
  done ;;
  (i) echo "$INFO" ; exit 0 ;;
  (w) clear; echo "$WELCOME" ;
          read welcome
          case $welcome in
                  y) # This allows for navingation in the filesystem.
        while true
        do
          menu
        done
          ;;
    n) # This navigtes the user to the desktop
          exit ;;
                  q) # This block executes the exit command from this 
     menu application
                      exit ;;
          esac # End program launch sub-menu switch case block
          read welcome
              ;; # End Welcome (-w)
```

```
106   (:) echo "error: $OPTARG requires an argument" ; exit 1 ;; #
        Tells if arrument is unknown
107   (?) echo "error: unkwown option $OPTARG" ; exit 1 ;; # Tells if
        option is unknown
108   esac
109 done
110
111 # Runs menu function in the event that no options are chosen
112 while [ -z "$LIST" -a -z "$ACTION" ]
113 do
114   menu
115 done
116
117 # Deploy -x * options from ALLOWED_ACTIONS variable help
118 if [ "$ACTION" = "help" ] ; then
119    echo "$HELPMENU"
120    exit 0
121 fi
122 # Deploy -x * options from ALLOWED_ACTIONS variable info
123 if [ "$ACTION" = "info" ] ; then
124    echo "$INFO"
125    exit 0
126 fi
127 # Deploy -x * options from ALLOWED_ACTIONS variable menu (one time
      run since if..fi)
128 if [ "$ACTION" = "menu" ] ; then
129    menu
130    exit 0
131 fi
132
133 FOUND=
134 for allowed in $ALLOWED_ACTION
135 do
136     if [ "$ACTION" = "$allowed" ] ; then FOUND="yes" ; fi
137 done
138
139 if [ -z "$FOUND" ] ; then
140    echo "error: unkwown action = $ACTION"
141    exit 2
142 fi
143
```

7.3 Make a Timer in the Shell for Use Elsewhere

7.4 Walk Through of a Quantum Espresso shell script

8 Topics Just for Fun

8.1 A Polyglot in the Shell

8.1.1 Polyglots Defined

8.1.2 This File is a .PDF document and a .ZIP of Its Own Source

8.2 Fork Bomb