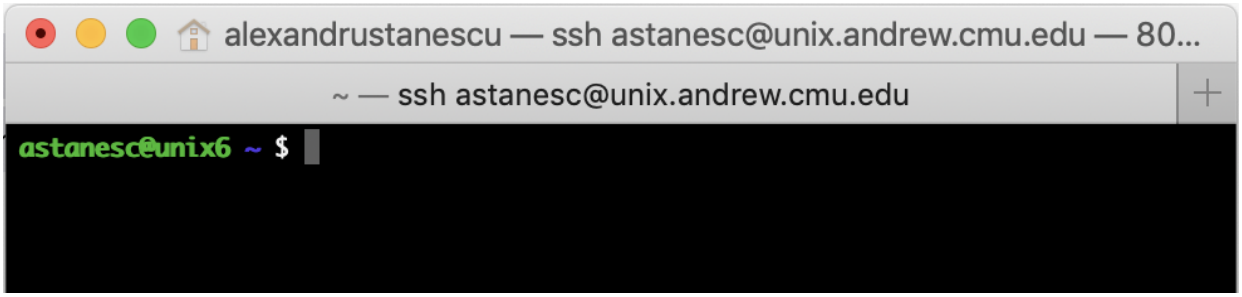
How to Use the Linux Terminal

1 What is Linux/The Terminal?

Linux is an operating system much like Windows, MacOS, Android, etc. The terminal is a text-based program that allows one to interact with the operating system at a deeper level than what the graphical user interface allows. The terminal looks something like this:



Important: Notice the **$** — this is called the *prompt*. When we show a command in this guide or in general, we will preface the command either with a prompt. Traditionally **%** is used instead of **$** as the prompt, so we will use that in this guide as well. **You should not type the prompt out when executing commands.**

2 Anatomy of a Command

In order to use terminal, one types out and then runs commands. Thus, before we start exploring useful terminal commands, we will start by discussing the anatomy of a command. A command has 3 main parts:

- **The name of the command**. This is generally a short 2 or 3 letter abbreviation of what the command does, such as `ls` for "**L**i**S**t files" (i.e. showing all the files in a folder).
- **Optional flags**. These are effectively "settings" for the command, in that they slightly change what the command does. Generally speaking flags start with a dash and are one letter. As an example, the `-a` flag for `ls` tells `ls` to list **a**ll files (including hidden files).
- **Positional arguments**. These are zero or more arguments that come after all of the flags and tell the command what to act on. For instance, `ls path/to/folder` lists the files in the folder that is given by the first positional argument (in this case, `path/to/folder`).

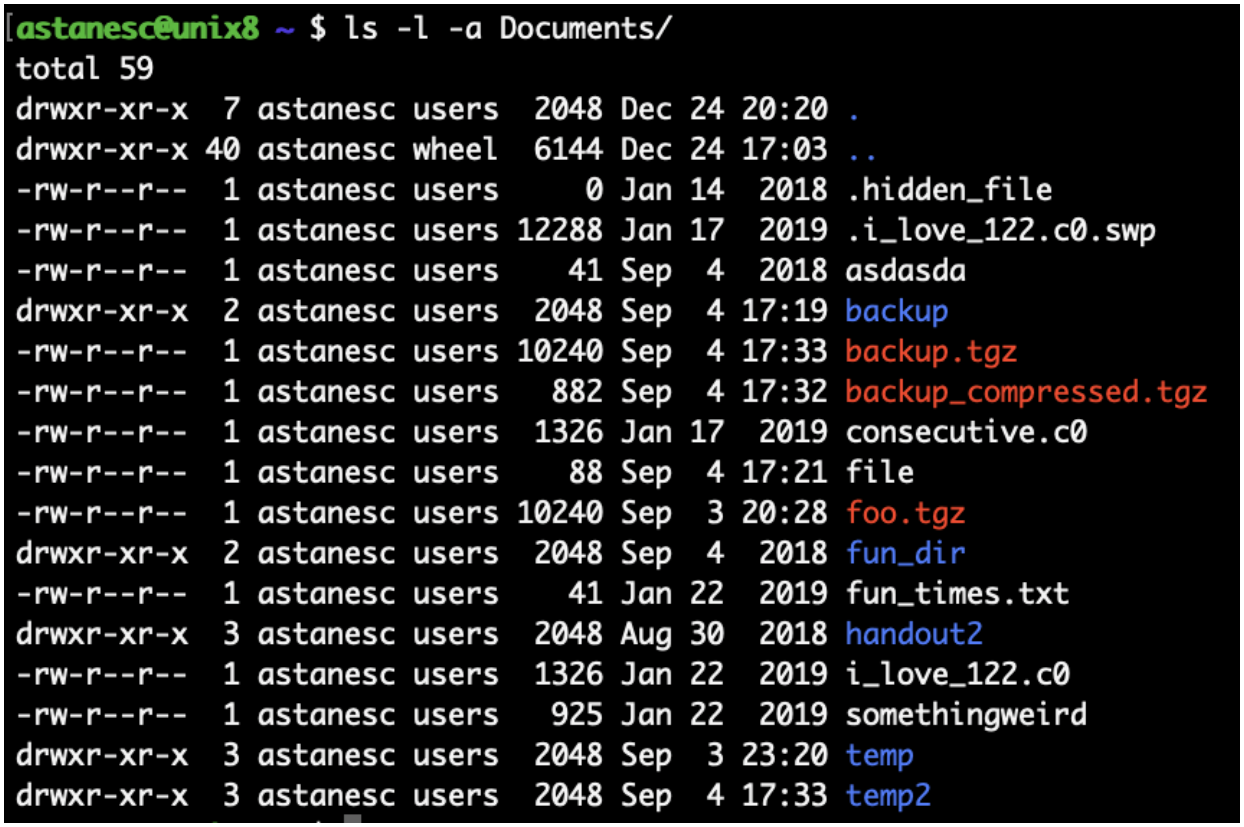
Note: Arguments, flags and commands are all separated by spaces. If you want to pass in a multi-word argument (such as `Random File`) you can either "escape" the space like in `Random\ File` or you can throw the whole argument inside quotations like in `"Random File"`.

Important: You can go through previously run commands by pressing the up arrow (to go further back in your history) or the down arrow (to go forward in your history). This is extremely useful to avoid having to retype commands!

To explore the anatomy of a command further, let use take the following command and break it down:

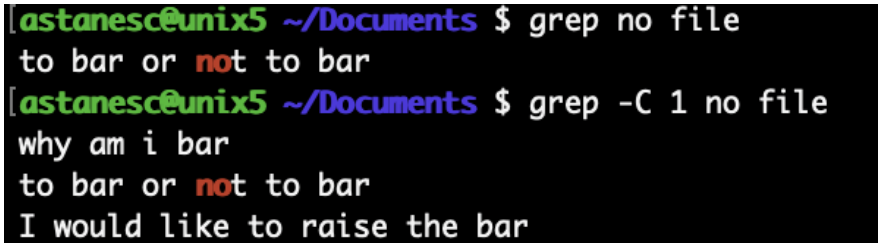
```
ls -l -a Documents/
```

Here `ls` is the name of the command. As mentioned before, `ls` lists files. Then, we have two flags: `-a` and `-l`. `-a` tells `ls` to display all files, and `-l` tells `ls` to display extra information about each file. Finally, we have the positional argument `Documents/`. This tells `ls` to list the files in the folder `Documents`. The final product might look like the following picture:



Note: It is possible to combine all of the flags into one argument. In other words, `ls -l -a Documents/` means the same thing as `ls -la Documents/`.

It is also possible for a flag to take in an argument. For instance, the `grep` command (which searches a file for a pattern) has the `-C` flag, which lets `grep` know how many lines surrounding any matches to print. In the below example, we showcase `grep` searching for "no" in a file named "file", which contains some text.



Notice how in the second run of grep, because we passed in `-C 1`; the `grep` has shown us not only the line containing the match, but also the line before and after.

Note: Most argument flags allow you to omit the space between the flag and the argument (as in `-C1`) and some allow you to replace the space with an "=" (as in `-C=1`).

Although most flags are one letter, some flags can be whole words. To differentiate between the two, flags that are whole words are generally preceded with two dashes. As an example, `grep` has the `--count` flag, which simply returns the number of matches of the pattern rather than the locations of the matches. In the previous example, `grep --count no file` would return `1`.

3 Navigating Linux

In your previous experiences working in Windows/MacOS, you probably have used a program like Window's **Explorer**, or MacOS's **Finder**, where you have a window which shows the insides of a particular folder. Within that window, you can interact with files by clicking on them (allowing you to do things like rename them, delete them or open them in an editor); or you can enter an internal folder to see its insides.

The terminal in some sense is merely a text-based version of these graphical applications — in that you are inside of a folder, and are able to view the contents of said folder, interact with files within that folder, enter internal folders, etc.

Important: In Linux and when navigating via the terminal, folders are referred to as "directories". Thus, we will be using the term "directory" from here on out rather than the term "folder"

Since the terminal does not have a graphical interface to let you "click" into folders, there are three important commands that we use to navigate through the terminal

- `pwd` — **P**rint **W**orking **D**irectory — This command prints out the full path of the directory that you are currently in. This is useful when you are unsure which directory you are in.
- `cd` — **C**hange **D**irectory — This command changes your working directory to the directory specified by the first argument.

  As an example, if the output of `pwd` is `astanesc/` and I ran `cd Documents/`, it would cause the current working directory to switch to the **Documents** directory within the **astanesc** directory. This means that the output of `pwd` would now be `astanesc/Documents/`.
- `ls` — **L**i**S**t files — This command lists all the files in a directory. If you give it no arguments, it defaults to listing all of the files in the current working directory.

Note: You can also chain directories together. For instance, in the `cd` example above, if I had wanted to go to the `15122` directory within the `Documents` directory, I can either execute `cd Documents` followed by `cd 15122`, or I can execute `cd Documents/15122`.

Important: Files and directories can be auto-completed by pressing tab. If there are multiple possible auto-completions, a list of the possible auto-completions can be obtained by pressing tab twice.

In addition to the directories within your current directory, there are four special directories that you can refer to in file paths. These are:

- `/` — This is the root directory. It is the only directory that is not enclosed by another directory. An example of a path containing this directory is: `/afs`. You will likely rarely use this directory directly.
- `~` — This is your home directory. Whenever you connect to a new terminal your working directory will by default always be the home directory. On AFS the full path of the home directory is something like `/afs/andrew.cmu.edu/usr23/acarnegie`, so `/afs/andrew.cmu.edu/usr23/acarnegie/Documents` and `~/Documents` are the same thing.
- `.` — The current directory. This can be useful in order to make certain commands more explicit.
- `..` — The parent directory. If your current directory is something like `~/Documents/15122/code/` then your parent directory refers to `~/Documents/15122/`.

4 Affecting Files

There are 6 main commands used to affect files in various ways. These are listed below:

4.1 Creating/Editing Files

To create or edit a file, you must use a file editor. Popular choices for a file editor are `vim` and `emacs`. We have written a guide on how to use vim, which you can read here: vim. If you wish to create or edit a file named `file` at location `path/to/`, you should run one of the following commands:

```
% vim path/to/file
% emacs path/to/file
```

4.2 Removing Files

To **rem**ove files, we use the `rm` command. To remove a file named `file` at location `path/to/`, simply run the command

```
% rm path/to/file
```

If you attempt to remove an entire directory using this method, you will get an error saying:

```
    rm: cannot remove 'path/to/dir': Is a directory
```

to silence this error, you must pass in the `-r` flag. Since when you remove a directory it is possible for that directory to have files you didn't expect (such as files within directories inside the directory you are removing), **it is recommended** you also pass in the `-i` flag, which will ask before deleting any individual file to make sure that you really want to delete that file.

Important: When you delete a file using this command, that file is permanently deleted. With the possible exception of the nightly backup (see Section Recovering Files on AFS) it is **IMPOSSIBLE** to recover a file after removing it with this command. Use this command at your own risk

4.3 Copy Files

To **cop**y files, we use the `cp` command. To copy a file named `file` from the directory `original/path/to/` to the directory `new/path/to/`, simply run the command

```
% cp original/path/to/file new/path/to/file
```

If you wish to rename the file (say, to `newname`) you simply change the command to be

```
% cp original/path/to/file new/path/to/newname
```

Note: If you do not wish to rename the file, it is not necessary to include the name of the file again — the following commands are identical

```
% cp original/path/to/file new/path/to/file
% cp original/path/to/file new/path/to/
```

This is a good use of the `.` directory — if you wish to copy a file into your current directory, you can simply execute

```
% cp original/path/to/file .
```

Important: When you copy a file into a new location, if a file already existed at that location, then the old file that existed at that location will be deleted and will be impossible to recover. Please be careful! Similarly to `rm`, **it is recommended** you also pass in the `-i` flag, which will ask before overwriting any individual file to make sure that you really want to overwrite that file.

4.4 Moving Files

To **mov**e files, we use the `mv` command. To move a file named `file` from the directory `original/path/to/` to the directory `new/path/to/`, simply run the command

```
% mv original/path/to/file new/path/to/file
```

If you wish to rename the file (say, to `newname`) you simply change the command to be

```
% mv original/path/to/file new/path/to/newname
```

Note: If you do not wish to rename the file, it is not necessary to include the name of the file again — the following commands are identical

```
% mv original/path/to/file new/path/to/file
% mv original/path/to/file new/path/to/
```

Important: Much like `cp`, when you copy a file into a new location, if a file already existed at that location, then the old file that existed at that location will be deleted and will be impossible to recover. Please be careful! Similarly to `rm`, **it is recommended** you also pass in the `-i` flag, which will ask before overwriting any individual file to make sure that you really want to overwrite that file.

4.5 Creating Directories

To **ma**k**e** a **dir**ectory, we use the `mkdir` command. To create a new directory named `newdir` inside the directory `path/to/` we simply execute:

```
% mkdir path/to/newdir
```

4.6 Searching Files

In order to search a file for a piece of text, we use `grep`. To search for "some text" in a file called `file` inside of the directory `path/to`, we simply execute:

```
% grep "some text" path/to/file
```

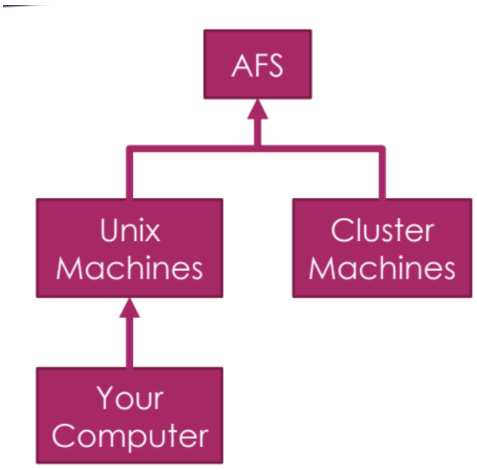
In Section Learning a New Command we further explore some interesting flags that can be passed into `grep` in order to showcase man pages.

5 Using the Andrew File System

In this class, you likely will be spending a lot of time working on the AFS Unix machines, as these have the C0 Compiler and other tools pre-installed.

The Andrew File System (or AFS) is a centralized database and associated server that allows you to use many different computers as if they all had the exact same files and file structure. The AFS Unix servers are servers that use AFS as their file system and that you are able to connect to from your laptop in order to work on coursework. Furthermore, the GHC cluster machines (and many other university computers) also use AFS as their file system!

If one therefore were to map all of these systems, it would likely look something like this:



In order to connect to the unix machines, we use `ssh` — **s**ecure **sh**ell, which allows us to run commands and have them execute as if they were executed in the unix machines. In fact, what the SSH connection does on a high level is transmit each keystroke you make to the unix machines and then wait for the unix machines to respond with what to display. As a result of this behaviour, if your internet is slow, you may notice that it takes some time between you typing a key and the key appearing on your screen!

To connect to the unix servers, please follow the instructions in the Ed post entitled "Laptop Setup for <OS>" where <OS> is the OS you are running on your laptop.

5.1 Transferring Files

At times you may find yourself needing to transfer a file from AFS to your computer, or vice versa. The easiest way to do this is by using a command called **S**ecure **C**o**p**y (or `scp`). `scp`, much like `cp` takes in two arguments — the original location of the file and the new location of the file. If you need to transfer a file from AFS to your computer, the following command will do the trick, replacing ANDREWID with your own andrew id:

```
% scp ANDREWID@unix.andrew.cmu.edu:path/to/file ./
```

This (much like `ssh`) will first ask for your andrew password. The path to the file is assumed to start at your home directory. Thus, if you wanted the file at `~/private/15122/importantfile.txt`, you would need to write `ANDREWID@unix.andrew.cmu.edu:private/15122/importantfile.txt`. Furthermore, much like with `cp`, if you want to rename the file along with copying it, then simply specify the new name of the file:

```
% scp ANDREWID@unix.andrew.cmu.edu:path/to/file ./new_file_name
```

Lastly, if you wish to transfer from your computer to AFS, then you simply swap the order of the arguments, as in:

```
% scp path/to/file ANDREWID@unix.andrew.cmu.edu:new/path/to/
```

Important: In order to be able to run the `scp` command, you must be in a terminal which is not already connected to the unix machines. On Mac/Linux, this can be accomplished by simply not

running the `ssh` command when opening up a new terminal instance. On Windows, you can open a new MobaXTerm tab and clock "Open new Local Terminal".

5.2 Recovering Files on AFS

On AFS, there is a nightly backup made "sometime between 6pm and 4am" of your entire AFS content to the `OldFiles` directory. If you accidentally delete or inadvertently modify a file, it is possible to recover the version of the file from the nightly backup by copying it over from the `OldFiles` directory.

There is no other way to recover a file in Linux.

6 Compressing and Uncompressing Files

On Windows, you may be familiar with the `.zip` extension for compressed archives (where an archive is just a collection of files with some associated metadata — effectively just a fancy folder). In Linux, there are two separate file extensions for these two concepts:

- `.tar` — This represents only the archive part of a compressed archive. Thus, a tar file is simply a collection of files with a bit of extra metadata.
- `.gz` — A g-zipped file. This is a file that has been compressed. Note that this is similar to a `.zip` file, but it is only a single file.
- If one takes an archive and then g-zips it, one forms a `.tar.gz` file, or more commonly abbreviated as `.tgz`. This is the most similar file type to the `.zip` file in Windows.

6.1 Compressing

To convert a set of files into a compressed archive, we use the `tar` command. If we wish to compress the files `file1`, `file2`, `file3` and `file4` into an archive called `result.tgz`, we use the following command:

```
% tar -czvf result.tgz file1 file2 file3 file4
```

The four flags have the following meanings:

- `-c` — Compress the files into the archive (as opposed to uncompressing!)
- `-z` — Compress the archive in addition to just creating it. If you do not wish to compress the archive, leave this flag off.
- `-v` — "Be Verbose": while creating the archive provide additional info about what is going on.
- `-f result.tgz` — Name the resulting archive `result.tgz`

6.2 Uncompressing

To uncompress and expand a compressed archive into its constituent files, we simply replace the `-c` flag with a `-x` flag! In other words, to expand the archive `result.tgz`, we run

```
% tar -xzvf result.tgz
```

Important: Some browsers uncompress `tgz` files when you download them, and as a result you may encounter the following error:

```
gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now
```

To suppress the error, simply omit the `-z` flag when uncompressing files downloaded by that browser.

7 Learning a New Command

There will come a time when youâĂŹre not sure how to do something on a terminal. In those cases, youâĂŹll want to get help on how to do it. Thankfully, there are several useful resources to provide you with help. One of the best such resources is the manual (or man) page for the command.

The man page for a command `cmd` can be accessed by executing `% man cmd`. The man page for a command is split up into at least 4 sections (some have more). Throughout this section we will be using the man page for `grep` as an example.

7.1 Section: Name

This section gives a list of equivalent names that one can call this command by as well as a short description of the command. For `grep`, the name section looks as follows:

```
grep, egrep, fgrep --- print lines matching a pattern
```

7.2 Section: Synopsis

This section provides a short description of how one should call the command. If the command has multiple distinct ways to call the command, then these will be displayed one per line. For grep, this section looks as follows:

```
grep [OPTIONS] PATTERN [FILE...]
grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]
```

To understand exactly what this means, let's break down the first way to call grep. Before we get into the first way to call grep, we must first go through a few conventions:

- If a word appears in all caps (e.g. `PATTERN`), it is understood to mean a concept rather than typing exactly those letters. Conversely, if a word is not in all caps (e.g. `grep`), it is understood to mean "Type exactly this".
- If a word appears within brackets (e.g. `[OPTIONS]`) then the word typed (or the concept it represents) is optional, unless:
- If a word appears within brackets and with a pipe (|) inside, then it is understood to mean "either this or that". In other words, `[XXX | YYY]` means "either XXX or YYY", **not** "optionally type XXX | YYY"

- If an elipsis (...) appears after a word, then that word (or concept) can be entered as one or more arguments. In other words, `FILE...` means "one or more files"

Thus, the first line (`grep [OPTIONS] PATTERN [FILE...]`) is telling you "`grep` can be called by first typing `grep`, then potentially typing some other options (e.g. flags), then typing a pattern, and then potentially also typing one or more files". Try to figure out on your own what the second way to call `grep` is saying!

Note: The conventions discussed above apply nearly everywhere that command-related things are described!

7.3 Section: Description

This section gives a longer description of the command as well as explaining any arguments that appear in the synopsis. It will also make a note of any idiosyncrasies of the arguments — for instance, if you pass in "-" as a file (or do not pass in a file), `grep` will search through whatever you type in next to find the pattern rather searching for a file named "-".

7.4 Section: Options

Here, the man page describes every single flag available to you as well as whether it takes in an argument or not. Some flags have multiple ways to be invoked — these will be listed separated by commas. As an example, it is possible to suppress error messages about nonexistent or unreadable files by using either `-s` or `--no-messages`, so in the man page, these are listed as:

```
-s, --no-messages
    Suppress error messages about nonexistent or unreadable files.
```

7.5 Other Tools

Another useful tool for understanding a command is the help flag. Most (not all) commands have a `--help` or `-h` option that will print out a message about how to use it. This message is generally shorter and easier to read through than the man page.

Lastly, you can also use Google to look for answers to questions that are not as easily answered by man. StackOverflow tends to be a very good resource to answer questions. ItâĂŹs important to be careful with Googling, since some answers are wrong or overly complicated.