Essential Linux Shell Commands

- Special Characters
- Quoting and Escaping
- Change Directory
- Show Current Directory
- List Directory Contents
- Working with Files
- Working with Directories

Special Characters

- There are some characters that have special meaning in Unix
- They should **never** be used in file or directory names
- They are

```
& : | * ? ' " [ ] ( ) $ < > { } # / \ ! ~
```

- Three other characters, called **whitespace**, are also special
- The whitespace characters are
  - Space
  - Tab
  - Newline

Special Characters

- The Space and Tab characters are used to separate words on the command line
- They separate the command from its options and its arguments
- The **newline** character is what you get when you press Enter on a PC or Return on a Mac
- The newline character tells the **shell** you are done typing a command and the shell should run the command you just entered
- When the shell sees a newline it executes whatever is on the command line

Quoting and Escaping

- If you need to use a special character on the command line you must do two one of two things
  - Use quotes
  - Escape the special character
- **Escaping** means putting a backslash, `\` in front of a special character to turn off its special meaning
- You can put backslash character, \ just before the character
- The backslash must come **immediately** in front of the special character

Quoting and Escaping

- The backslash turns off the special meaning of the newline, which is *normally* to run the command you have typed

```
$ cat foo.txt
foo
bar
bletch

$ cat > \
> foo.txt
foo
bar
bletch
```

- The backslash turned off the special meaning of the Enter key
- The shell did not try to execute the command line
- Instead, it waited for you to finish the command on the next line
- The second greater than symbol, > is the shell saying it is waiting for more of the command

Quoting and Escaping

- What if you have *many* characters to escape?
- You could put a backslash before each one
- But, it is probably better to use quotes

```
$ echo >>
-bash: syntax error near unexpected token `newline'
$ echo \>\>
>>
$ echo '>>'
>>
```

- There are two types of quotes
  - **Single quotes -** ' '
  - **Double quotes -** " "
- They have slightly different meanings, but you don't need to worry about the difference for now

cd - Change Directory

- *cd* (**c**hange **d**irectory) changes your current directory
- You use *cd* to move from one directory to another

```
cd  DIRECTORY_NAME
```

- If you use *cd* with the name of a directory, it moves you to that directory
- If you use *cd* without an argument, it takes you to your **home directory**
- The home directory is the directory you are in when you first log in to Unix
- To go up one directory, use `..` as the argument for *cd*

```
cd ..
```

pwd - Show Your Current Directory

- *pwd* (**p**rint **w**orking **d**irectory) displays your current directory

```
pwd
```

- In the beginning, use *pwd* every time you use *cd*
- This will keep you from getting lost

ls - List the Contents of a Directory

- *ls* (**ls**t) is one of the most basic Unix commands
- It shows you the files and directories inside a directory
- The command line is **not** a GUI
- It is easy to lose track of where you are
- When this happens, certain things will not work as you would expect

ls - List the Contents of a Directory

- When *ls* is used without an argument, it lists the contents of your current directory
- When you use *ls* followed by the name of a directory, it lists the contents of that directory

```
ls  DIRECTORY
```

- When you use *ls* followed by the name of a file, it simply displays the file name

ls - List the Contents of a Directory

- When used with the -a (for **a**ll) option, *ls* will list **all** files -- including those whose names begin with a .
- Any file whose name begins with a . will not show up when you use *ls*, unless you use the -a option
- These "invisible" files are configuration files
- Under normal circumstances, you pay no attention to them
- We'll discuss these files later in the course

ls - List the Contents of a Directory

- Another useful option to *ls* is **`-l`** which displays a "long" listing

```
$ ls -l
total 20
-rw-rw-r--   1 ghoffmn        103 Sep 11 14:34 basic.css
-rw-r--r--   1 ghoffmn       3560 Aug 29 13:30
emacs_cheat_sheet.html
-rw-r--r--   1 ghoffmn        701 Aug 29 13:30 index.html
drwxr-xr-x   6 ghoffmn        512 Sep 15 14:11 it_244
-rw-r--r--   1 ghoffmn       6831 Aug 29 13:30 tips.html
-rw-r--r--   1 ghoffmn       6052 Aug 29 13:30
unix_cheat_sheet.html
```

- (NOTE: This is the lowercase letter **l**, not the number 1. Do not get them confused!)
- We'll talk about this more when we discuss permissions

cat - Print the Contents of a File

- *cat* (con**cat**enate) displays the contents of a file

```
$ cat foo.txt
foo
bar
blecth
```

cat - Print the Contents of a File

- When used with the -n option *cat* displays line numbers

```
$ cat -n lines.txt
     1  line 1
     2  line 2
     3  line 3
     4  line 4
     5  line 5
     6  line 6
     7  line 7
     8  line 8
     9  line 9
    10  line 10
```

cat - Print the Contents of a File

- If you run *cat* on a long file, the contents of the file will pass by too quickly to read
- To view long files, you should use a **paging program**
- Paging programs, like *more* and *less*, show the content of a file one screenful at a time
- You can navigate using arrows, the space bar, and other keys on your keyboard
- I'll discuss them more in a future class

rm - Delete a File

- *rm* (**re**m**o**ve) deletes a file

```
rm   FILENAME
```

- *rm* does not ask you if you are sure before deleting the file
- **There is no undelete feature in Unix**
- Deleted files cannot be recovered, if not backed up
- Don't delete a file, unless you are ***sure*** you won't need it

rm **- Delete a File**

- To remove **all** the files in a directory use

```
rm  *
```

- Be careful when using * with *rm*
- It will delete everything, and there is no way to get back what was deleted

rm - Delete a File

- If you run *rm* with the -i option, it will ask you before deleting each file

```
rm –i foo.txt
rm: remove regular file `foo.txt'? y [Enter]
```

- It's good idea to use this option when running *rm* *
- *rm* cannot be used on a directory -- unless you use the -f or "force" option

Directories

- ***mkdir* - Create a Directory**
  - *mkdir* (**ma**k**e dir**ectory) creates a directory

    ```
    mkdir  DIRECTORY_NAME
    ```

  - The directory will be created in the current directory, unless you specify another location
- ***rmdir* - Delete a Directory**
  - *rmdir* (**rem**ove **dir**ectory) deletes a directory

    ```
    rmdir  DIRECTORY_NAME
    ```

  - *rmdir* will not work on a directory, unless the directory is empty

cp - Copy Files

- *cp* (**co**p**y**) makes a copy of a file or a directory
- *cp* takes two arguments
  - The first argument is the source - the file or directory to be copied
  - The second argument is either the new filename, if you are making a copy in the same directory or the directory into which the copy will go

```
cp  FILENAME  NEW_FILENAME_OR_DIRECTORY
```

- *cp* can copy an entire directory -- when used with the -r (for **r**ecursive) option

mv - Move a File or Directory

- *mv* (**m**o**v**e) is a command that does two different things
- It can change the ***location*** of a file or directory

```
mv   FILENAME_OR_DIRECTORY_NAME   NEW_DIRECTORY
```

- It can also change the ***name*** of a file or directory

```
mv   FILE_OR_DIR_NAME   NEW_FILE_OR_DIR_NAME
```

- In either case, *mv* takes **two** arguments

mv - Move a File or Directory

- When used to move something, the first argument is thing to be moved, and the second argument is the new location

```
$ ls
foo.txt  it244  work

$ ls work

$ mv foo.txt work

$ ls
it244  work

$ ls work
foo.txt
```

mv - Move a File or Directory

- When changing the name of a file or directory, the first argument is the old name, and the second is the new name

```
$ ls
foo.txt  hold  it244  work

$ mv foo.txt bar.txt

$ ls
bar.txt  hold  it244  work
```