Intro to Linux and CLI

Knowing Surroundings

- `pwd` - List present working directory
- `ls [path]` - Lists files in current directory or in given path. Common flag to pass is -l, this gives more information on files. Another flag is -a, list hidden files.
- `cat <file>` - Show contents of file
- `set` - Will list variables set, -o will list options set
- `cd` - Will change directory

There is a special file in your home directory called .profile, put anything in here that you want to run when you first login.

Understanding Paths

- `.` Refers to current directory
- `..` Refers to one directory up
- `~` Refers to home directory of current user
- `/` Refers to root directory
- `*` Wildcard, can be mixed in pathnames or filenames

Note: . in front of file is a "hidden file"

There are multiple ways to refer to a file, for example let's say in our home directory we had a file called test.c, we could list out the contents several of the following ways:

`cat test.c` (assuming we are in the home directory)
`cat ../krupp/test.c`
`cat ~/test.c`
`cat /home/krupp/test.c`

Other Common Commands

- `grep <search> [filename]` - Used to search contents within file or from output
- `less [filename]` - Used to scroll through file
- `man command` - Will pull up manual for a particular command (use it!)

Piping Output to Input to Process

You can take the output of one command and send it as input to another command. This is known as piping.

Example:

`ps -ef | grep 212` <— Will look for a specific PID

`cat test.c | grep // | grep -v “Brian”` <— Pulls up any comments in c source that do not have Brian in it

Redirecting Input/Output

You can also redirect input and output. In most cases, you will redirect output.

`ps -ef > process_log` Takes the process listing and saves it in a file called process_log

`ps -ef >> process_log` Takes the process listing and appends it to the file called process_log

Sending Process to Background

We can send a process to the background by appending an ampersand to the end:

`ps -ef &`

However, just because the process is in the background doesn’t mean that it won’t print to our console. We can redirect that output to a file:

`ps -ef > out &`

However, let say we send a process to the background that runs into an error, redirecting output(STDOUT) is not good enough on its own. We must also redirect STDERR, which is the any error output:

`ps -ef > out 2>&1 &` 2 = STDERR, is being appended to STDOUT(1)

To list any processes we sent in the background type in jobs:

`jobs`

More Ways of Looking at Files

You can look at the bottom of a file or the top of a file with tail and head respectively.

If you want to monitor any new input on a file, you will want to do a tail-f on the file:
`tail -f logfile`

File Permissions

To list file permissions, use the following command: `ls -l`

In general there are 3 permissions with Unix/Linux files, read, write execute. These are represented using 3 bits: rwx. These permissions are classified for the user that owns the file, the group that owns the file, and everyone else. As an example below, root is the file owner, operator is the group owner.

```
-rw-r--r-- 1 root   operator 43   Sep    2 15:26 out
drwxr-xr-x 2 root   operator 128  Sep    2 15:16 somedir
-rwxr-xr-x 1 root   operator 5305 Sep    2 15:01 test
```

The first field is what type of file we are working with. The two most common types are - for normal file, and d for directory.

The next 3 letters are the permissions for the owner, so for file test, root has read, write, and execute permissions. Execute meaning they can run that file as code. Group has just read and execute permissions, so they cannot modify the file, and everyone else has the same permissions as the group for this case.

If you want to set permissions you use the chmod command, here are two ways of doing it. One is absolute where you specify the exact permissions you want set by specifying the number for each permission area, User, Group, Everyone else. Each number corresponds to the appropriate bit, rwx, and are computed as a binary number. Read = 4, Write = 2, Execute = 1. So if I want rwx for the owner and no permissions for everyone else, I would do the following on a file:

`chmod 700 test`

If you want rwx for owner, just read and write for group, and nothing for everyone else, you would do the following:

`chmod 760`

7 = 4 (read) + 2 (write) + 1(execute)

6 = 4 (read) + 2 (write)

0 = nothing

You can also change permissions relative to what the file currently has set. For example, you can set or add permissions:

`chmod u+x test` Adds execute permission for the user

`chmod +x test` Adds execute permission for everyone

`chmod g+rw` Adds read/write for group

`chmod o-x` Removes execute for other (everyone that is not the owner or the group

Modifying Files

We can move files around, rename them, create directories, etc

`mv <file> <dest>` Moves a file to a directory or renames it

`rm <file>` Removes a file

`mkdir <dirname>` Makes a directory

`rmdir <dirname>` Removes a directory

`rm -fr directory_name` Removes a directory and everything in it (be careful!)

`cp <source file> <destination>` Copies a file or directory

As an example using wildcards, if you wanted to remove all programs that end with .c in a file, you would do the following: `rm *.c`

exit

Important Environment Variables

- PATH - Where we look for a particular command
- PS1 - What you see in shell

To set environment variable:

`PS1="readytorun: "`

To print out environment variable

`echo $PS1`

Other Useful Commands

- `uname` List information on OS you are on
- `ip addr` List information about network interface and IP
- `host <name>` Retrieves IP for a given name
- `hostname <name>` Set the hostname or get the hostname of your OS
- `cat /etc/os-release` Shows information about the current OS
    - `locate <name>` Pass in the name of a file to help locate it