Commands General

The singular of which is command general

The formatting on this one was a real headache, so I hope you guys appreciate it

Command Structure

Office of the President
Secretary of Defense
Secretary of the Navy
Area 11 Manager
Principal of Troy HS
Senior Naval Science Instructor
Naval Science Instructor
Naval Science Instructor

Commands, options, arguments

E.g.:

```
grep -rl --max-count=1 -- “Cat Caro” /home/neville/
```

Don’t worry about this just yet

```
<cmd> <-p<={}> or--param<={}>> <--> <inputs>
```

`cmd`- the command itself

- Could be `ls`, `cd`, `pwd`, `nano`, `sudo`, `sublime_text`, etc.
- Refers to the actual command run
- Can also be a filepath, e.g. `/usr/bin/ls`

`<cmd> <-p<={}> or--param<={}>> <--> <inputs>`

`-p` and `--param`: parameters passed into the command

- e.g. `-r`, `-l`, `--ignore-case`, `--num-lines=4`, etc.
- "`-`" can combine multiple params (e.g. `-qv` is equivalent to `-q -v`)
- Modify the way the command runs
  - Might make it run recursively or silently or similar
- `=` allows an argument to the parameter, e.g. `--num-lines=4`
- `--` is for long form parameters: e.g. `--extract` instead of `-x`
  - These can not be merged like the short form parameters can

```
<cmd> <-p<={}> or--param<={}>> <--> <inputs>
```

`--` - Separates the parameters from the input

- That’s it, that’s the slide

```
<cmd> <-p<={}> or--param<={}>> <--> <inputs>
```

inputs- the input to the actual command

- This is what the command actually uses
- Input can be a file path, string, number, etc.
- These are the same types that apply to an argument

`<cmd> <-p<={}> or--param<={}>> <--> <inputs>`

- Command is obviously necessary
- Some commands might not use parameters at all, some require them
    - `ls` can run without parameters, `tar` doesn't do much without them
- Not all parameters require arguments
    - `ls`'s `-h` doesn't need an option, `head`'s `-n` does
- `--` is optional

`<cmd> <-p<={}> or--param<={}>> <--> <inputs>`

- Parameters that exist for one command might not exist for another or might have a different meaning
  - `ls --recursive` will look through each directory recursively
  - `tar --recursive` does not exist
  - `ls -R` is equivalent to `ls --recursive`
  - `tar -R` exists but is something else
- Arguments don’t require the `=`
  - `head -n 16` is equivalent to `head -n=16`

`<cmd> <-p<={}> or--param<={}>> <--> <inputs>`

- Input types vary from command to command
  - `pwd` runs without using inputs
  - `ls` can run without an input, but will assume the input is the current directory
  - `printf` requires a textual input

What/where is/are the...

Command?

Parameters?

Arguments?

Inputs?

```
grep -rl --max-count=1 -- “Cat Caro”
/home/neville/
```

What/where is/are the...

Command?

Parameters?

Arguments?

Inputs?

```
grep -rl --max-count=1 -- “Cat Caro”
/home/neville/
```

Why the `--`?

(This title was unintentional btw)

- Parameters can be put anywhere in the command
- `--` helps clarify the command

```
grep --max-count=1 “Cat Caro” -r /home/neville/ -l
```

```
grep -rl --max-count=1 -- “Cat Caro” /home/neville/
```

Which would you rather parse?

Other notes

- Linux commands, parameters, arguments, and inputs are all CASE SENSITIVE
- You can precede a command with as many spaces or tab characters as you want
- You can end commands with semicolons`;`
- Tab completion exists

Some specific commands

Lotta notes incoming!

- `ls` (there's still more to mention)
- `sudo` and `su`
- `touch` and `mkdir`
- `mv`, `cp`, `rm`, and `rmdir`
- `cat`
- `nano` and `gedit`
- `man`
- some others if we have time

`ls <file or directory>`

“list”

- Input can be a file or a directory or blank
    - If input is blank, will infer you mean current directory
- `-a` or `--all`- lists all files, including hidden files
- `-l`- lists files in a long listed format
- `-R` or `--recursive`- lists recursively

`ls -al`?

`find <directory>`

- Lists all files in the directory recursively
  - Goes through each directory inside
- `-type=`(`f` or `d`)- only lists certain types of files
  - `f` is for files, `d` is for directories
- Same as `ls -R` but faster and with worse graphics

`sudo <a full-on command>` and `su`

- `sudo`: provides higher privileges for a user
  - Super User DO
  - Allows them to run commands as the omnipotent root user
  - Really useful when configuring files, as they may have pretty secure file permissions
  - `-i`- equivalent to `su` but asks for your password and not root’s
- `su`: you are root now
  - Switch user or super user
  - Everything you run will be run as root
  - Can also pass another username as input to switch to a different user
- Try not to run these 100% of the time
  - Can be dangerous if you screw stuff up
  - So only use it if you need to
  - Also a vulnerability if used by others

`touch <file path>` and `mkdir <directory>`

- Commands for making files
    - `touch` makes files
    - `mkdir` makes directories
- None of the files or directories made will have content
    - You’ll have to fill that in if you want stuff to be in there

`mv`, `cp`, `rm` and `rmdir`

```
mv <file> <destination file or directory>
```

- move
- Moves files from one location to another
- If moving to a directory, will keep the same filename in new directory

```
cp <file> <destination file or directory>
```

- copy
- Behaves the same as mv but copies the file instead

`mv`, `cp`, `rm` and `rmdir` (cont.)

```
rm <file path>
```

- remove
- Removes this file

```
rmdir <directory>
```

- remove directory
- Removes this EMPTY directory
  - Directories with files in them require a different command...

`rm -rf <directory>`

- Deletes directories with files in them
- Very strong, try to avoid if you can
- Don’t do `rm -rf /`, I will be very disappointed in you

`cat <file>`, `head <file>`, and `tail <file>`

- Throws up contents of file onto the terminal as output
- `head` prints the first 10 lines
- `tail` prints the last 10 lines
- `cat` prints the whole file
- `head` and `tail` both have the `-n <N>` parameters to print the first or last `N` lines instead of the first or last 10 lines

`nano <file>` and `gedit <file>`

- Most-used text editors by Linux users
- `gedit` is graphical, `nano` stays inside terminal
- Type up stuff up in the files here
- You will edit config files through here (unless you like other editors ~~(SUBLIME TEXT)~~)

`man <command>`

- Manual pages/man pages
- Tell you everything you need to know about a command
  - It’s called documentation kids
- Lists out what the command does along with its options
- May documents bugs and fixes, but don’t worry about that for now

Some others if we have time

`clear`- clears all the junk in your terminal so it looks clean :)

`exit`- exits the terminal and closes it

`shutdown`- turns off the computer*

`reboot`- restarts the computer*

*- Requires `sudo` or `su`

`less <file path>`

- A n o t h e r t e x t e d i t o r
- Pipe output to this make it scrollable
- Mostly used for viewing but can be used to edit

`grep “search term” [filename or directory]`

- Searches for terms in a file
    - Returns lines in the file with the search term
- `-r`- read through directories for a file with the search term
- `-i`- ignore case (upper/lower case doesn’t matter anymore!)
- `-v`- invert match
    - Basically outputs lines that DO NOT have the search term

Piping

- Takes output of one command turn into input of another command
- Very, very useful if you don't want to run extra commands
- Use the `|` character (under the backspace button) to signal shell to use this technique
- Ex: `cat hello.txt | grep hello`
- Ex: `find / | less`