

Unix

Desktop Tasks + Programming Power

What's Unix?

- Category of operating system
- Open source
- Many, many variants
 - Including Linux and all *its* variants
- Used often for servers, also Android
- Can also refer to the terminal/shell/text interface
 - That's what we'll be looking at

Cygwin

- Unix environment that runs in Windows
- Lets you run the unix terminal on Windows files
- Window's also has a text terminal (the command line) but we're ignoring that
- Search for Cygwin from the start bar and run it

Navigation and Basic Info

`ls` - list files in the current directory/folder

`pwd` - print working directory, aka tell me what directory I'm in now

`whoami` - print your username

`cd path` - change directory to the provided path

Paths

Paths are maps in the text terminal, they tell the computer how to get to a specific folder or file

Absolute paths start with a “/” and begin from the root folder on the computer

Relative paths *don't* start with a slash and begin from the terminal's current directory

Keywords:

“.” refers to the current directory

“..” refers to the directory above/containing the current directory

“~” refers to the user's home directory

Try Paths

Where do these go?

`cd temp`

`cd .`

`cd ..`

`cd ~/Documents`

`cd /temp`

`cd ~/../../blah/example.txt`

Get Some Files to Play With

- Download the class repo again (like last time)
- Figure out where it downloaded to
- Move the shell to the `unix_files` folder in the class files
- Move directly to the folder in one ``cd`` command

File Manipulation

`cp path1 path2` - Creates a copy of the file in path1 at path2

`mv path1 path2` - Moves the file in path1 from its current location to path2

`rm path` - Removes the file/folder at the end of the path (careful with this one!)

`touch path` - Updates the edited/modified time of a file and creates it if it wasn't there

Try Some Manipulation

```
touch test.txt
```

```
cp test.txt test2.txt
```

```
rm test2.txt
```

```
mv test.txt ../parent_test.txt
```

```
mv ../parent_test.txt .
```

Good Shortcuts

- Up and down arrows cycle through previous commands you have typed into the terminal
- Pressing tab while typing a path will make the terminal try to guess what you were typing
- Double-tab to see all tab completion guesses
- Copy+Paste still work

Documentation and Options

`man command` - displays info on that command

`command -option` - runs command with one or more options (like parameters)

Options can be

- puttogether

- one –at –a –time

Some options require another parameter, like
`touch –t` (check it out with `man touch`)

Examining Files

`cat path` - displays the full text contents of the file

`less path` - displays file contents one page at a time

`head path` - displays the first few lines of a file

`tail path` - displays the last few lines of a file

`grep text path` - displays each line of the file that contains the provided text (super powerful tool!)

`wc path` - displays the number of words in the file (can count other things too)

Moving Data Around

`> path` sends the output of a command to a file (and creates it if need be, otherwise overwrites)

Try `ls > myfiles.txt`

`>> path` sends the output of a command to a file (creates it if need be) and *appends it* to the current contents

Try `ls .. > myfiles.txt`

Moving Data Around

`command` < text` - Uses the text as input for that command

Try ``wc < myfiles.txt`` (what does this do?)

`command1` | command2` - Uses the output of command1 as the input for command2

Try ``ls | wc`` (no need for a text file)

Wildcards

`*` Can stand in for any number of other characters in a filename (not a path though)

Try `cat *.txt` (what does this do?)

`?` Also works but just for single characters

E.g. `head version?.png`

What Do These Do?

```
$ whoami | wc -m
```

```
$ mv *.jpg ~/Pictures/
```

```
$ ls -R | grep *.txt | wc -l
```

```
$ man grep | grep grep
```


Executing Programs

The stuff we've been doing are all built-in commands, but the shell can run any executable file

`./executable_file`

Dumb as it is, you need to include the “./” if the executable is in the same directory as the shell

Make and Run Code!

```
$ echo "print 'Hello World'" > hw.py
```

```
$ python hw.py
```

```
$ javac HelloWorld.java
```

```
$ java Hello World
```

Why Does ``$ python`` Work?

We had to install it so it's not a built-in thing like `ls` or `grep`, so why do we not need `./python` or need to know where the python executable is?

\$PATH Variable

Try ``echo $PATH``

- \$PATH is a special variable that keeps a list of folders
- Any time a command is run, the shell looks through all the folders in the \$PATH to see if the executable from the command is in any of them
- There should be at least one reference to Python in the \$PATH (if you were able to run it before)

Other Cool Commands

``ps`` - list running processes...

``kill`` - ...and kill them!

``curl`` - makes a web request

(try ``curl www.google.com``)

``wget`` - downloads online files

``ssh`` - access the terminal of a different computer
(can still do all the same stuff there!)

``chmod`` - Change file/folder permissions
(assuming you're allowed to!)