EPBI 414

Unit 5
Computing Fundamentals & Unix / Linux

The Recap - Unit 4

- Optimizing databases using indexes
- ACID in database transactions
- Distributed data systems and CAP theorem
- Denormalized data storage
- Introduction to NoSQL

Unit 5 Overview

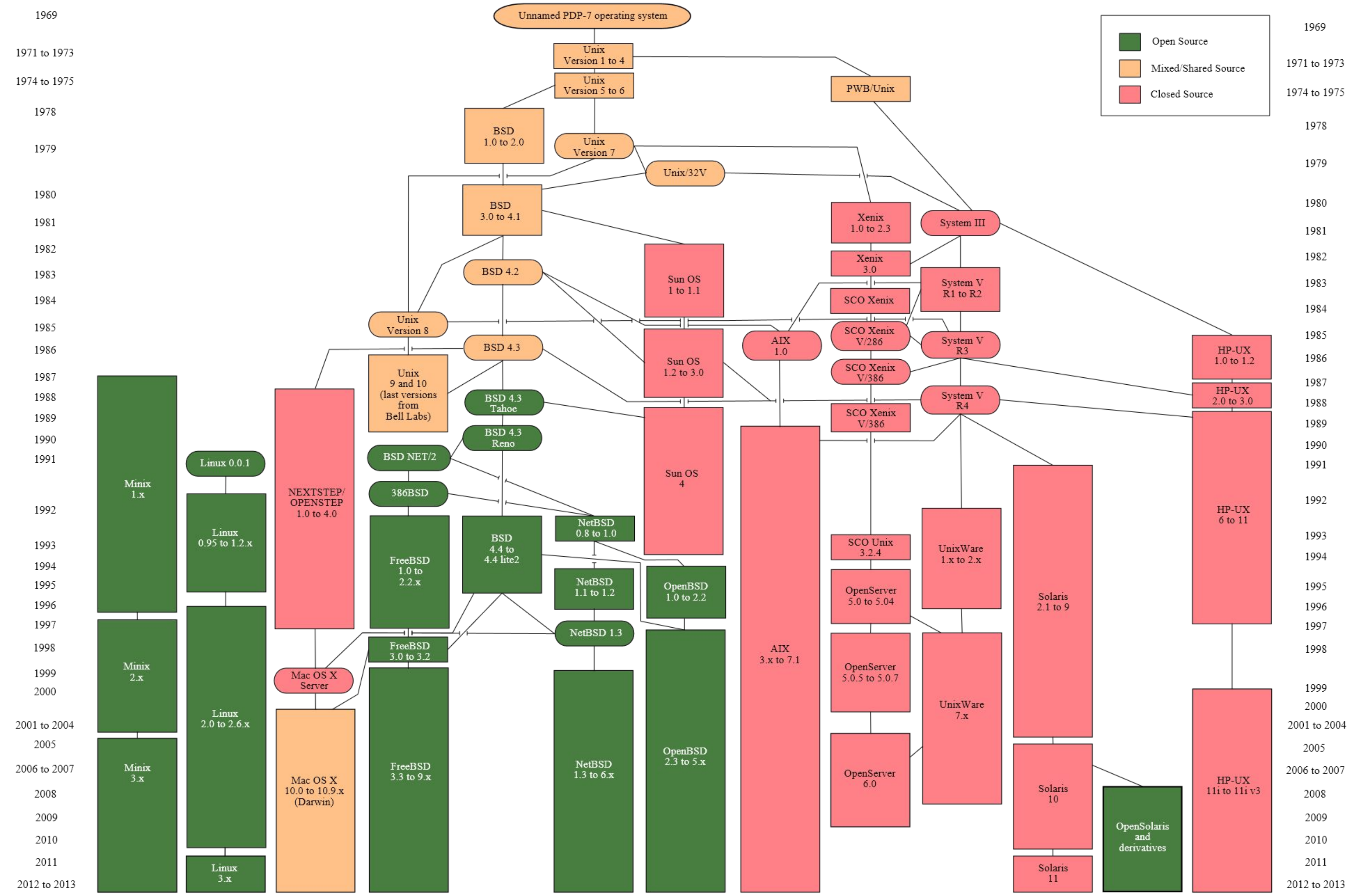
- What is Unix / Linux?
  - The Linux kernel and OS

- Introducing the command line
  - Accessing servers over SSH
  - Understanding file permissions, users, and groups
  - Core *nix commands (e.g. `ls`, `pwd`, `mkdir`, `rm`, etc)
  - Processes, pipes, and redirects
- Basic scripting for file manipulation

What is Unix?

- 1970s-era operating system (OS), first built by AT&T

- Designed around core principles:
    - Simplicity - each program is focused on doing one thing well
    - Modularity - programs are self-contained units
    - Synergy - the system is useful because it is more than the sum of the programs



A brief history of Unix[1]

What is Linux?

- Contrary to popular belief, it is not a descendent of Unix
    - That's FreeBSD (aka Mac OS X)
- It is a Unix-like (designed to be similar)
    - POSIX compliant (OS standard)
- Has spawned one of the biggest open-source movements ever: GNU/Linux

Linus Torvalds, father of the Linux kernel[2]

The Linux kernel

- To be clear: Torvalds did not create all of Linux

- His main contribution is the "Linux kernel"

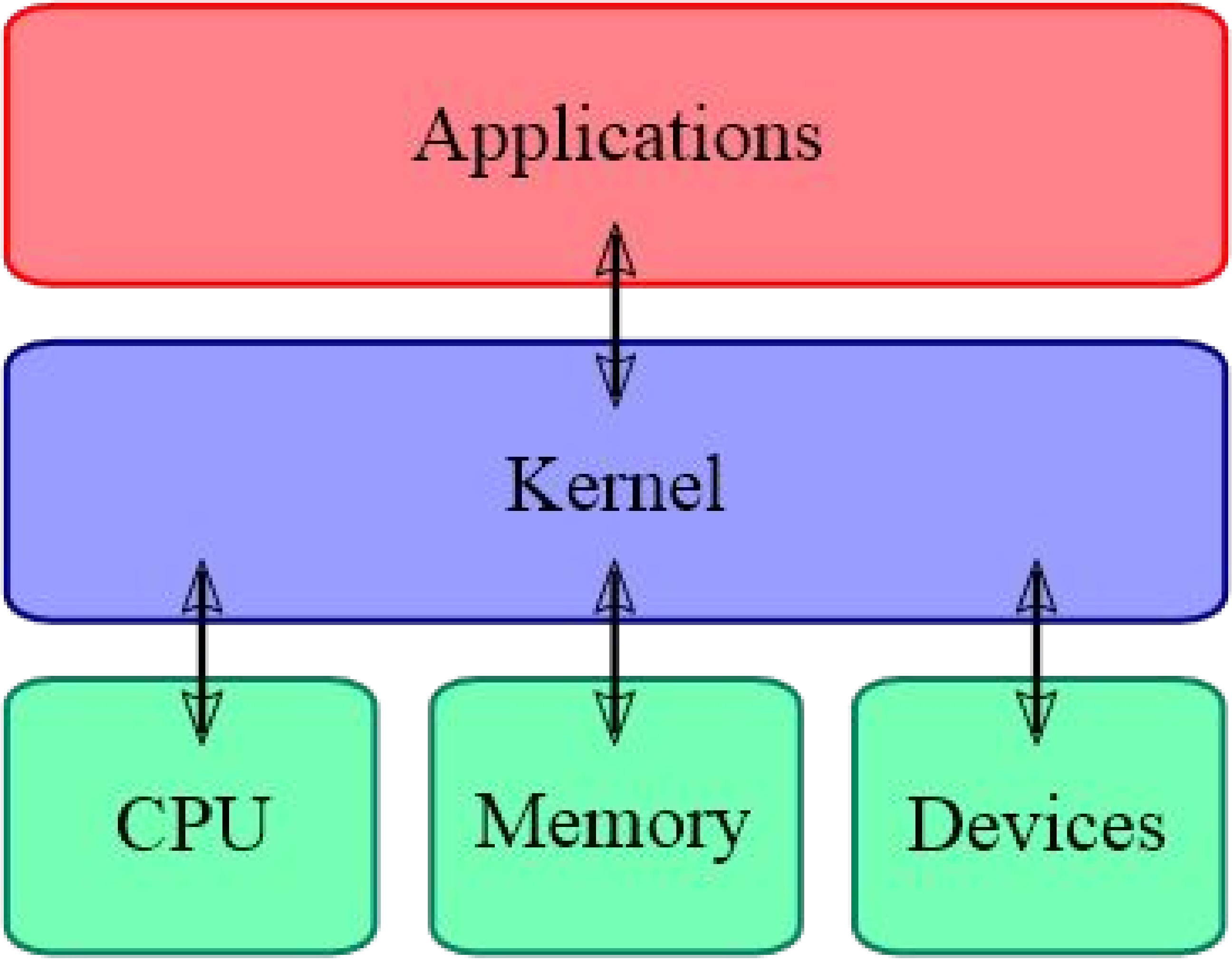
- The kernel sits between the software and the hardware
  - Controls access to hardware resources
  - "Closest to the metal"

Kernel as abstraction

- The kernel is the core of a computer operating system, and controls access to and usage of hardware resources

- It is a key example of computing abstraction
    - Takes over managing low-level details of the computer - user operates at the higher level

    - Not the same as data abstraction, but same idea: reduce complexity to human understanding



The kernel as low-level abstraction layer[3]

Free as in freedom

- GNU/Linux is extremely prominent example of **free and open-source software** (FOSS)

- Torvalds did not write the kernel alone
  - To date, there have been thousands of authors

- Often both free as in beer (i.e. no money) and free as in freedom (you can copy and reproduce it)

Richard Stallman (rms), Free Software Advocate[4]

The GNU Frontier

- rms is a huge advocate for free software
- Created the GNU General Public License, or GPL
  - GPL requires that the source code be open
  - Also requires anything you build from it to be open
  - Hence, the forking family tree
- He is also a prominent author of GNU

Advantages of Unix / Linux

- Stability
- Low-level access to hardware
- For Linux: cost ($$$)
- Also, freedom (philosophically)
- Case in point: R is FOSS

The downside

"Windows makes 80% of what needs done easy, and 20% of what needs done impossible.

Linux, in contrast, makes everything generally medium-hard in difficulty."

To learn more about FOSS

- Read "The Cathedral and the Bazaar"
  - Seminal piece about the open-source development philosophy
- Note that open-source isn't the same as freedom, in some ways
- Different reasons to make each argument
  - All of it is relevant to this ecosystem

Your new home

Your new computing home for this week:

`hal.epbi.cwru.edu`

How to get home

- SSH (**S**ecure **Sh**ell) is the primary way you communicate with "headless" servers
  - Outdated terminology, but it sounds gnarly
- SSH encrypts your communications
  - Older way was **telnet** - effectively deprecated now (NOT SECURE)
- You need to connect to **`hal`** over SSH to work

SSH client choices

- You have four major ones for the class
  - SSH Secure Shell
  - Cmder
  - Cygwin
  - PuTTy (cross-platform)
- Mac OS X users can use Terminal, as can any Linux users

Getting Connected

- Your username is your Case ID

- Connect like this:
  - From a terminal:

```
ssh tar9@hal.epbi.cwru.edu
```

  - Otherwise, set the server as `hal.epbi.cwru.edu` and your user name as `xyz123`

A note on security

You will see something like this:

```
The authenticity of host 'hal.epbi.cwru.edu (129.22.208.44)' can't be
established.
RSA key fingerprint is 76:6e:b4:79:94:06:fd:ad:25:b3:3d:a2:39:47:ae:72.
Are you sure you want to continue connecting (yes/no)?
```

This is okay in this case - but understand it

Man in the middle

- Modern encryption is very strong
  - In practice, hard to decrypt
- A better way of attacking: trick people into logging into a machine that is not the one they use
- This is called a "man in the middle" or MITM attack

MITM, continued

- The first time, your computer doesn't know if Hal is genuine.
    - You can say "Yes" if you trust that computer or its key

- If the key offered by Hal changes later, your computer will be very suspicious - because someone may be ***pretending*** to be Hal

A possible MITM attack

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
76:6e:b4:79:94:06:fd:ad:25:b3:3d:a2:39:47:ae:72.
Please contact your system administrator.
Add correct host key in /home/Tom/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/Tom/.ssh/known_hosts:18
RSA host key for hal.epbi.cwru.edu has changed and you have requested strict checking.
Host key verification failed.
```

The shell

- A shell is another term for a command prompt

- Unlike Windows, there are a lot of shells you can choose from in *nix
  - We will predominantly use Bash (the **B**ourne **A**gain **Sh**ell)

  - Other choices: Csh, Zsh, Korn, Tcsh...

The magic shell

- Lets you interact with the OS
- Interprets and executes your commands
- Starts and controls programs for you
- Lets you write **scripts** to automate things

Useful shell features

- Use Tab to complete things
- Use Up & Down to see command history
- Use characters to navigate commands
  - ~ = home directory
  - . = current directory
  - .. = one level up

Users and groups

- Unix-like systems have file-level permissions and ownership tied to users and groups

- Your account is a user (xyz123)

- Your account can be a member of one or more groups
    - Often, default is for every user to have a group of the same name (i.e. tar9 is in tar9)

Storing users and groups

- Common method: the `passwd` and `group` file

- You can see the users and groups on the server using:

```
    cat /etc/passwd
    cat /etc/group
```

File permissions

- Managed using ***octal*** format

- File permissions are displayed in the following manner:

```
rwxrwxrwx
```

- Gives owner (rwx), group (rwx), all (rwx)

Example permissions

Owner read only	`r--------`
Owner full control	`rwx------`
Owner read/write, group read only	`rw-r-----`
Everyone read/write	`rw-rw-rw-`

Translating to octal

- Octal representation ranges from 0 to 7 (eight levels)

- 4 is used to denote read, 2 to denote write, and 1 to denote execute

- Add them together to get combinations
  - 6 = 4 (read) plus 2 (write)
  - 5 = 4 (read) plus 1 (execute)

Writing octal permissions

- Octal permissions let us reduce a set of permissions to a single number

- A file's permissions for owner, group, and everyone can be set with three digits

- Full access for owner, read and execute for group, read-only for everyone else:
    - (4+2+1) (4+1) (4) = 754

Some octal permissions

Displayed Permission	Octal Code	Owner Permissions	Group Permissions	Global Permissions
rw-rw----	660	Read, Write	Read, Write	None
rwx--xrw-	716	Read, Write, Execute	Execute	Read, Write
rwxrwxrwx	777	Read, Write, Execute	Read, Write, Execute	Read, Write, Execute
rw-rw-rw-	666	Read, Write	Read, Write	Read, Write

Visualizing permissions

- Your second Unix command: `ls`

- This command lists files (similar to `dir` in a Windows computer)

- The generic version just shows files
  - To get more - you need a switch! (aka an option)

Command options

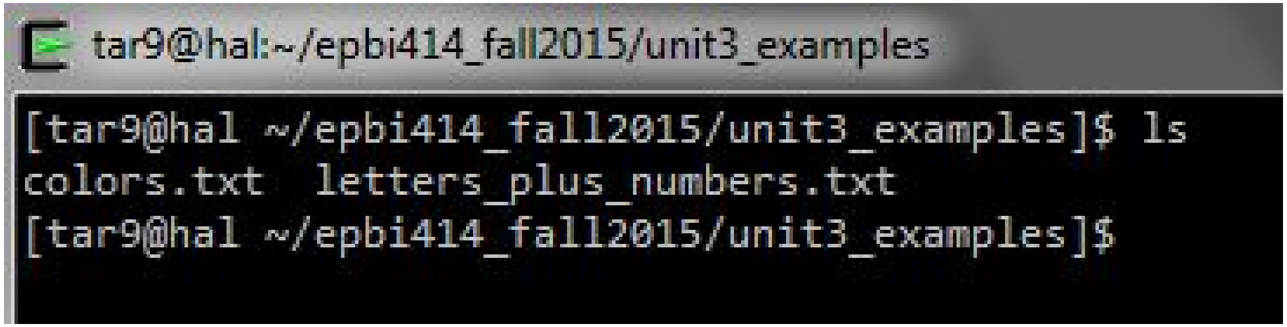
- A command option (colloquially a switch) modifies the behavior of a command

- Different from an ***argument***

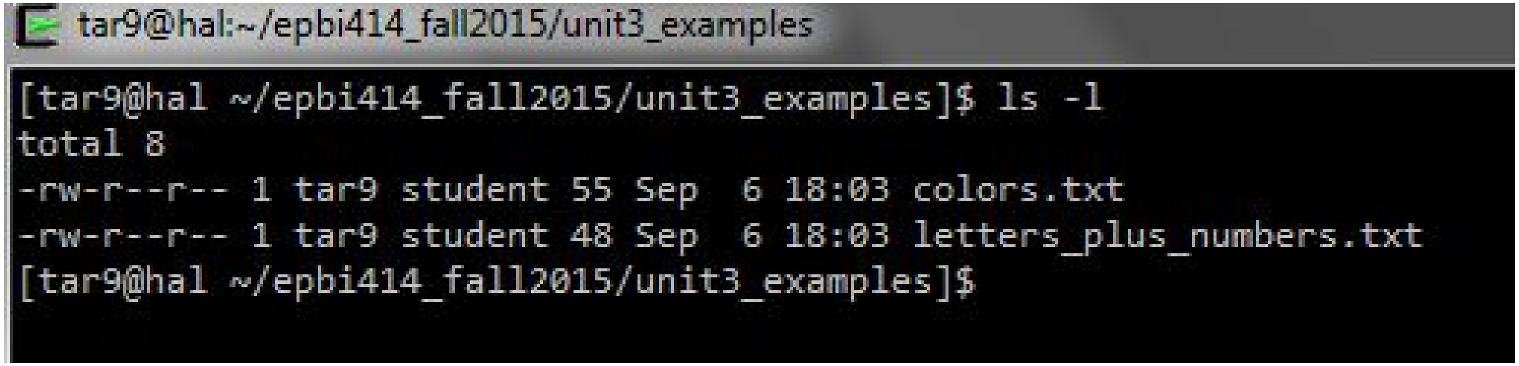
- First example: the long format of `ls`
  - `ls -l`

- This shows files and attributes

`ls` - with & without switch



```
[tar9@hal ~/epbi414_fall2015/unit3_examples]$ ls
colors.txt  letters_plus_numbers.txt
[tar9@hal ~/epbi414_fall2015/unit3_examples]$
```



```
[tar9@hal ~/epbi414_fall2015/unit3_examples]$ ls -l
total 8
-rw-r--r-- 1 tar9 student 55 Sep  6 18:03 colors.txt
-rw-r--r-- 1 tar9 student 48 Sep  6 18:03 letters_plus_numbers.txt
[tar9@hal ~/epbi414_fall2015/unit3_examples]$
```

Setting permissions

- Manipulating permissions is done with the `chmod` command

- Syntax:

```
chmod <octal code> <file name>

chmod 640 yourfile.txt
```

Recursion

- Recursion is applying the same command to subdirectories
- Using the `-R` flag on `chmod` can let you apply permissions to all elements of a directory
- Be careful - this may not be what you want

Setting ownership

- Similar to setting permissions
- Uses the `chown` command
- Syntax:

```
chown <owner:group> <file name>

chown tar9:faculty yourfile.txt
```

Other core commands

- `cd` - **c**hange **d**irectory
  - Changes the directory you are in
  - `cd /path/to/directory`
- `pwd` - **p**rint **w**orking **d**irectory
  - Shows you the current directory you are in
- `cp` - **c**opy and **p**aste
  - Copies file to file (use recursive to do directories)
  - `cp file.txt new_file.txt`

Core commands, 2

- `mv` - **mo**ve file
  - Moves a file or directory (also used to rename)
  - `mv file.txt betterfile.txt`
- `rm` - **re**mo**ve**
  - A dangerous command - Unix has no Recycling Bin
  - Removes files or directories (use recursive for directories)
  - `rm badfile.txt`

Core commands, 3

- `mkdir` - **make directory**
  - Creates a new directory
  - `mkdir a_new_directory`
- `scp` - **secure copy paste**
  - Allows you to copy files between machines
  - Can also be done using SFTP Client
  - `scp this.txt tar9@hal.epbi.cwru.edu:~`

Core commands, 4

- `sort` - **sorts** lines of text input
  - `sort a_txt_file.txt`
- `cat` - con**cat**enates and prints files
  - Can be used to print a file
  - `cat a_txt_file.txt`
- `head` - shows the first rows of a file
  - `head a_txt_file.txt`

Core commands, 5

- `tail` - shows the last rows of a file
    - `tail a_txt_file.txt`
- `less` - lets you page through large files
    - Can also be used to page through output from files
    - `less a_big_file.txt`
- `echo` - print a string to the terminal
    - `echo "123"`

Core commands, 6

- `grep` - searches text files using regular expressions (regex)
    - `grep A letters_plus_numbers.txt`

- Ancient wisdom: "Some people, when confronted with a problem, think: 'I know, I'll use regular expressions.' Now they have two problems."[5]

There are many more

- Can't possibly cover all of the functions in Unix

- Some other useful ones are:
    - `wc, cut, spell, more, who, ps, sed, awk, which...`

- You'll need good Google-Fu to master Unix

A very useful command

`man`

- Used to access the "manual" for a command
- Example: `man ssh`
- When in doubt, read the directions
    - Origin of RTFM ("Read the F[rea]king Manual!")

Wildcards and globbing

- Often called wildcard matching, a ***glob*** is a pattern that is used to match multiple items

- Most common example: *.exe
  - This would match all files that ended in .exe

- Globbing is supported on the command line
  - Allows you to select multiple files at once

- The more advanced version: regexs

Running executable files

- To run an executable file, use `./filename`
  - This is really what happens when you type `ls`
  - Gets into your PATH variable, not relevant right now
- The file "extension" is not relevant in Unix-like systems (can really be anything)
- Scripts contain a line indicating what program is used to execute them

Processes

- All computers have ***processes*** underlying things

- Unix-like systems assign numeric ***process IDs*** to processes

- Everything is a process, including your commands (`ls`, `cp`, etc)

Viewing processes

- `top` - a process viewer for Unix
- `top` is a way of viewing the processes that are running in the system
- Can quit using Ctrl-C, or q key

Sending processes away

- Something that may take awhile to run can be sent to the background

- This lets you start a process and then continue doing something else

- Use an ampersand (&) to send a program to the background
  - `./program1.sh &`

STDIN and STDOUT

- STDIN and STDOUT are ***streams***
  - Generally, STDIN = keyboard
  - STDOUT = monitor / terminal
- A typical program takes input from STDIN and prints output to STDOUT
- The shell offers pipes and redirects that can change this!

Redirecting STDIN

- You can use the < character to redirect STDIN

- Lets you read something from a file, rather than typing it into the keyboard

- Mostly useful when you have a script or program that takes arguments from the keyboard

Redirecting STDOUT

- You can use the character > to redirect the output of programs into a file

- By default, it goes to the terminal - sometimes, we want to save it somewhere else

- > overwrites a file that exists; >> ***appends*** to an existing file

Piping and plumbing

- Using > and < lets you read from and write to files

- Sometimes, we want to send output from one command into another command

- For this, we use a **pipe**, denoted by the pipe symbol (|)

Rules of piping

- A pipe connects the STDOUT stream from one process to the STDIN stream of another process
- Goes in order from left to right across the command
- Example:
  - `sort colors.txt | grep Blue`

The value of piping

- Pipes allow us to connect together various commands, which enables "filters"

- We can rapidly search large amounts of text files to find strings, for instance

- Or list all the files, search for a specific one, and write the contents of it to another file

Stopping the flow

- You can connect a lot of pipes together, but once you get to a redirect (a >), the flow ends

- Sometimes, more than one command is easier than fifteen pipes

Break Time

Text editors and scripts

- Many text editors in Unix and Unix-like systems
- Ask 10 programmers what the best text editor is: get 12, all of which are the best
- You will eventually find the one that works best for you

Some big ones

- vim, or vi Improved (my first)
- emacs
- nano
- joe (a personal favorite)
  - Simple learning curve
  - What I learned in EPBI 414!

JOE - An Introduction

- JOE - Joe's Own Editor
  - Also referred to in all lowercase, i.e. `joe`
- A relatively simple, lightweight editor
  - Often ***not*** installed by default - you may need to fall back to vim (usually the standard)
- There are two ways to start `joe`
  - `joe`
  - `joe <filename>`

Quick Start to JOE

- Use Ctrl-K-<letter> for many commands

- Built-in help: Ctrl-K-H
  - ^KH

- JOE saves the file you are editing in a backup with a ~ at the end
  - e.g. I am editing `file.txt`, directory will contain `file.txt~`

A few JOE commands

- ^KD = save the file
- ^KF = find text in the file
- ^KX = save and exit joe
- Ctrl-C = exit without saving (will prompt)
- ^_ = Undo (usually Shift plus Minus key)

Tips for JOE and Unix

- **On the command line, the mouse is your enemy. Do not use it.**

- Save your work regularly

- Make backups - copy your file and work on the copy if you are writing a lot of new stuff

- Consider version control (Git)

Shell scripting

- A script is generally any program written in a ***scripting language***

- Here, we mean a ***shell script***, which is a "program" written in the "language" of the shell

- Lets you automate repetitive tasks you would perform at the shell

The first line

- The first line of every script needs to tell the OS what program to use when running the script

- This is accomplished in something like this:

```
#!/bin/bash
#!/usr/bin/R
```

The first line, continued

- Starts with a hash-bang (#!)

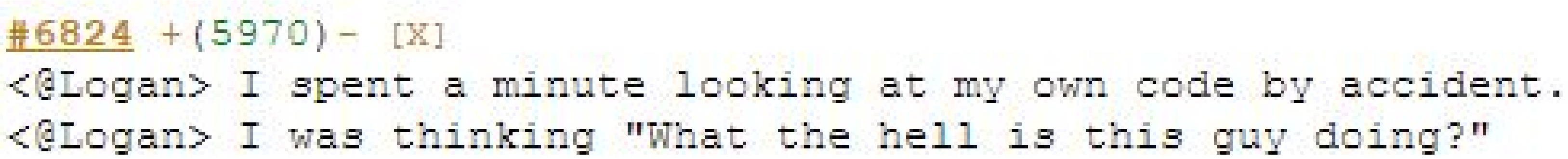
- Then, the ***path*** to the executable file that will execute the script
  - You can find this using `which`

- With this, the shell knows what to use in executing the file

Comments

- For bash scripts, the hash (#) lets you put comments into your programs

- **Comment. Your. Code.**
    - It is good practice.
    - In this class, it will affect your grade.
    - It helps you understand what you were doing.
    - It helps others understand what you were doing.



```
#6824 +(5970)- [X]
<@Logan> I spent a minute looking at my own code by accident.
<@Logan> I was thinking "What the hell is this guy doing?"
```

Why you should comment your code[6]

My first script

```
#!/bin/bash

# A script to print a message and then make a directory

# First, this prints a message
echo "This is a message."

# Next, we make a new directory
mkdir new_directory
```

Command-line arguments

- Sometimes, it is useful to pass an argument to a program

- One way is to pass a "command line argument"
    - Like this: `./myscript.sh myarg`

- These can make your scripts useful by letting you target them

My first command-line arg

```
#!/bin/bash

# A script that shows how command-line arguments work
echo "The command line argument is $1"
```

Expectations

- You cannot learn scripting overnight
  - bash is a really weird language, too
- Trial and error
  - Start simple
  - Use online resources
  - Get better over time

When to script

- When you find yourself writing the same series of commands again and again

- When you need to do the same thing to fifty (or a hundred...or whatever) directories

- When you need to control the computer and what it is doing

When not to script

- When you need to analyze data
  - Use an analysis language
- When you only need to do something once
  - Consider your time tradeoff
- When you are doing dangerous things
  - Scripts will let you do anything the shell will let you do - so be careful

Attributions, 1

1. Image obtained from:
   https://commons.wikimedia.org/wiki/File:Unix_history-simple.svg

   By Eraserhead1, Infinity0, Sav_vas [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)] via Wikimedia Commons

2. Image obtained from:
   https://commons.wikimedia.org/wiki/File:LinuxCon_Europe_Linus_Torvalds_05.jpg

   By Krd (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)] via Wikimedia Commons

Attributions, 2

3. Image obtained from:
   https://commons.wikimedia.org/wiki/File:Kernel_Layout.svg

   By Bobbo (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)] via Wikimedia Commons

4. Image obtained from:
   https://commons.wikimedia.org/wiki/File:Richard_M_Stallman_Swathanthra_2014_kerala.jpg

   By Ranjithsiji (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)] via Wikimedia Commons

5. Jeffrey Friedl (http://regex.info/blog/2006-09-15/247)

Attributions, 3

6. Screenshot taken by the presenter at this site: http://www.bash.org/?6824

   License unclear.