Introduction

- Show New User slides
Linux
What is Linux?

- Linux is an Operating System (OS)
  - Software that enables computer devices and applications to perform tasks
- Linux is an open OS (unlike Windows & Mac)
  - Created in 1991 as a variant of Unix for PCs
    - Thousands of developers contribute
    - www.linuxfoundation.org
  - Most web-services today are run under Linux
GUIs and CMIs

- **GUI**: Graphical User Interfaces
  - Linux has these, too
  - But HPC does not support

- **CMI**: Command Line Interfaces/Interpreter
  - Called a ‘shell’ in Unix/Linux
  - The only way to communicate with HPC nodes
  - PCs & Macs also have CMIs
    - Accessed via terminals/xterms
    - *on PCs (DOS) & Macs (MacOSX)*
What is a Unix/Linux Shell?

- A command interpreter and processor
- HPC supports two shells
  - bash and tcsh ("t" shell)
    - Bash is default shell on HPC nodes, considered more functional
    - Bash = Bourne-again shell (sh), developed by Bourne in 1977

To verify and change your shell

$ echo $0 # show shell
-bash
$ tcsh # change shell to
tcsh
$ echo $0
tcsh
$ bash

## Comparison of Shells

<table>
<thead>
<tr>
<th>Function</th>
<th>sh</th>
<th>csh</th>
<th>ksh</th>
<th>bash</th>
<th>tcsh</th>
<th>zsh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job control</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Aliases</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Shell functions</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>I/O redirection</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Directory stack</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Command history</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Command line editing</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Vi Command line editing</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Emacs Cmd line editing</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Rebind Cmd line editing</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>User name look up</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Login/Logout watching</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Filename completion</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Username completion</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Hostname completion</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>History completion</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Full Completion</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mh Mailbox completion</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Co Processes</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Built-in arithmetic eval</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can follow sym links</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Periodic cmd execution</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Custom Prompt (easily)</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Sun Keyboard Hack</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Spelling Correction</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Process Substitution</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Underlying Syntax</td>
<td>sh</td>
<td>csh</td>
<td>sh</td>
<td>sh</td>
<td>csh</td>
<td>sh</td>
</tr>
<tr>
<td>Freely Available</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Checks Mailbox</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Tty Sanity Checking</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can cope w large arg lists</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Has non-interac. startup file</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Has non-login startup file</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can avoid user startup files</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Can specify startup file</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Low level cmd redefinition</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Has anonymous functions</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>List Variables</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Full signal trap handling</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>File no clobber ability</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Local variables</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Lexically scoped variables</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Exceptions</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Environment Variables

- Variables – a name and a value – defined within the Linux environment
  - Set and used by shells, users, scripts, programs
  - Names uppercase by convention, referenced with a “$”

- Display all environment variables
  
  ```
  $ env
  HOSTNAME=indigo.usc.edu
  TERM=xterm
  SHELL=/bin/bash
  USER=avalonjo
  MAIL=/var/spool/mail/avalonjo
  PWD=/home/avalonjo/workshop
  ```
Environment Variables

- **Example:** $PATH holds a user’s search path
  
  ```
  $ echo $PATH
  /usr/bin:/usr/local/bin:/bin:/sbin
  
  - When you type a command, bash looks searches for it in the directories listed: First in /usr/bin, then in /usr/local/bin, etc.
  ```

- **Set environment variables**
  
  ```
  $ JAVA_HOME='/usr/usc/java/default'
  
  - Many programs set (or require that you set) a ‘home’ variable
  
  $ export /usc/usc/matlab/default:${PATH}
  
  - Add MATLAB directory to path (gets searched before other directories)
  
  $ echo $PATH
A Case Study: System vs Software Repo

- Sometimes software and libraries (e.g. gcc, python, fftw) come with OS
- Although command name is the same, the system software and repo software are often different (versions, libraries, developers). Make sure that you use what you want to use
- which command shows the absolute path of a command

```
$ which python
/usr/bin/python
$ source /usr/usc/python/enthought/default/setup.sh
$ which python
/usr/usc/python/enthought/default/bin/python
```
Linux – Basic Commands
Basic Linux Commands

ls       List file and/or directory names
pwd      Print working (current) directory

$ touch a.a
$ touch b.b
$ ls
a.a      b.b
$ ls -lt   #list -long form, -sort by time
          total 0
-rw-rw-r--  1 avalonjo  avalonjo  0 Mar 24 13:44  b.b
-rw-rw-r--  1 avalonjo  avalonjo  0 Mar 24 13:44  a.a
$ pwd
/home/avalonjo/workshop
Basic Linux Commands

**cd**  
Change directory

**mkdir/rmdir**  
Print working (current) directory

```
$ ls  $ cd ..
a.a  b.b
$a.a b.b dir1
$ ls  $ ls dir1
    a.a b.b
$ cd dir1
$ pwd
/home/avalonjo/workshop/dir1
```
### Other Special Characters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Command</th>
<th>Current Directory</th>
<th>Parent Directory</th>
<th>Any Number of Any Character (Wild Card)</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>home directory</td>
<td><code>$ cd ~</code></td>
<td><code>/home/avalonjo</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>current directory</td>
<td><code>$ pwd</code></td>
<td><code>/home/avalonjo</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..</td>
<td>parent directory</td>
<td><code>$ cd ..</code></td>
<td><code>/home/avalonjo</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>any number of any character (wild card)</td>
<td><code>$ cd .</code></td>
<td><code>/home/avalonjo</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>$ pwd</code></td>
<td><code>/home/avalonjo</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
<td><code>$ ls .bash*</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `$ ls .bash*` shows bash dot files

---

USC ITS
Information Technology Services

University of Southern California
Basic Linux Commands

**cp/mv/rm**  
Copy/move/remove file or directory

**alias**  
Create an alias for a command

```
$ ls
a.a  b.b

$ cp b.b c.c
$ ls
a.a  b.b  c.c

$ mv c.c c.new
$ ls
a.a  b.b  c.new
```
Basic Linux Commands

*cp/mv/rm*  
Copy/move/remove file or directory

*alias*  
Create an alias for a command

```
$ ls
a.a  b.b  c.new
$ rm a.a
$ ls
b.b  c.new

$ alias rm 'rm -i'
$ rm b.b
rm: remove regular empty file `b.b'? n
```

```
$ mkdir dir2
$ ls
b.b  c.new  dir2
$ ls
b.b  c.new  dir2/
```
Basic Linux Commands

cat/more/less*  Display file contents

>  Redirect output from command line to file

$ ls
b.b  c.new dir2/

$ ls > ls.out
$ cat ls.out
b.b  c.new  ls.out  dir2/

$ less ls.out

*Less is a program similar to more (1), but which allows backward movement in the file as well as forward movement. Also, less does not have to read the entire input file before starting, so with large input files it starts up faster...
## Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>List file and/or directory names</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Print working (current) directory</td>
</tr>
<tr>
<td><code>mkdir/rmdir</code></td>
<td>Create/remove directory</td>
</tr>
<tr>
<td><code>cd cp/mv/rm</code></td>
<td>Change directory</td>
</tr>
<tr>
<td><code>cat/more/less</code></td>
<td>Copy/move/remove file or directory</td>
</tr>
<tr>
<td><code>man</code></td>
<td>Display file contents</td>
</tr>
<tr>
<td><code>chmod/chown</code></td>
<td>Change permission/ownership</td>
</tr>
<tr>
<td></td>
<td>Display online manual</td>
</tr>
</tbody>
</table>
Command History***

`history` Display a history* of your commands

↑ ↓ Previous command, next command, in order

! # Rerun a command

`$ history`

```
... 32 mkdir dir2
    33 ls
    34 alias ls 'ls -CF'
    35 ls
```

`$!35 #re-run the 35th command`

*Shell keeps a log of your commands -- it’s always a good idea to review this if you forget what you typed.
Command-line Completion***

<tab> Auto-complete the cmd/file/dir name
<tab><tab> Show candidates for the cmd/file/dir name

It can substantially reduce the number of keystrokes

$ ls /usr/usc/mat <tab> to autocomplete (fails)
<tab> to show candidates

$ ls /usr/usc/matlab <tab> to autocomplete (succeeds)
<tab><tab> to show candidates

$ ls /usr/usc/matlab/2 <tab> etc.
Linux
– Editing & Scripting
- File permissions and access
Editors

- There are several editors installed on HPC
  - “vi” is native to Unix/Linux operating systems
  - “vim”, “emacs”, and “nano” are also installed on HPC
- We teach nano because it is easiest to use
Editors

- The most useful commands are displayed along bottom
  - ^o indicates press Ctrl-o to write out (save) text
  - ^k/^u = cut/uncut, ^a/^e = move to beg/end of line
  - ^v/^y = page forward/backward, ^w = search, ^x = exit
Shell Scripting

- Now that you can edit, let’s create a script!
- Create a bash script named hello.sh
  - Open nano (nano hello.sh)
  - Type in the lines =============>
  - Save the file (^o), enter/return,
    Then exit (^x)
- Run the script!
  - Type hello.sh
  - Hmmm…. what’s wrong?
Shell Scripting

$ hello.sh
-bash: hello.sh: command not found

Problem: hello.sh is not in your path
  • Your current directory is not in your path by default
  • Solution: Use an absolute path

$ ./hello.sh
Shell Scripting

$ ./hello.sh
-bash: ./hello.sh: Permission denied

Problem: hello.sh is a text file

• It is not an executable program
• Solution: Change mode of file to executable

You need “permission” to execute, just like you need permission to read and write files.

$ chmod +x hello.sh
Shell Scripting

$ ./hello.sh

hello world
File Ownership and Permissions

- Permissions are associated w/ every file & directory
  - $ls –lg      # list files (-l long) and groups (-g)
    # This will display permissions

- Permissions – three types
  - read, write, execute
  - r=read, w=write, x=execute

- Ownership – three types
  - owner, group, others
  - u=user (owner), g=group, o=others (also a=all)
File Ownership and Permissions

chmod file  Change file permission
chmod a+w file  Add write permissions for all users
chmod 750 file  Set rwx permission for user, r-x permission for group and no permission for others

rwx can each be set to 0 or 1,
so range is 000-111 (binary) and 0-7 (decimal)
i.e., r=100 (4), w=010 (2), x=001 (1)

111 101 000 => rwx  r--x  ----- 
7 5 0 => u  g  o
File Ownership and Permissions

groups

Displays the group ID of all groups you belong to. The first ID is your primary group ID. By default, all files you create will have this group ID.

chgrp <grp> mydir

Change group ID of this directory (or file)

chmod g+s mydir

Set group ID (also setgid) of mydir so that all new files and subdirectories within mydir will inherit the group ID of the directory, rather than the primary group ID of the user who create the file.

(The exception is if you mv a file to mydir.)
Bash Script #2

- Create clock.sh
  - $ nano clock.sh
  - Type in the lines
    ===========>
  - Save the file and exit:
    ^o + <enter/return> + ^x

- Run the script!
  - Add +x permission
  - Specify current directory
Linux – Shell configuration files
bash Shell Configuration Files

- Configuration files are used to set up user environments (also called *config*, *init*, and *dot* files)
  - Minimal shell configuration files come in each user’s home directory by default (use “ls -a” to view)

- For bash, these are `.bash_profile` & `.bashrc`
  - `.bash_profile` is read by `bash`, hpc’s default shell, when you login
  - `.bash_profile` sets up your hpc environment and then runs `.bashrc`

- Place your custom run commands, prompts, paths, aliases, and environment variables in `.bashrc`
tcsh Shell Configuration Files

- For **tcsh**, the config files are **.cshrc** & **.login**
  - **.cshrc** is read by **tcsh**, if you are using this shell, when you login
  - **.cshrc** sets up your hpc environment and then runs **.login**
- If you use tcsh, place your custom run commands, prompts, paths, aliases, and environment variables in **.cshrc**
.bashrc editing

- Make a backup
  - `cp .bashrc .bashrc_ori`
  - `nano .bashrc`
  - Type in the lines `====>`
  - Save the file and exit

- Source the script!
  - `$source ./bashrc`
  - Show your aliases ($aliases)
  - Now test your aliases
Linux – Process management
Process Management

- **A process is a task executed by Linux**
  - When you execute a command, at least one process, or job, will be created
  - Each process is assigned a unique integer called a **Process ID (PID)**
- **Type $ top to view running jobs**

```
Processes: 200 total, 2 running, 6 stuck, 192 sleeping, 922 threads
Load Avg: 1.34, 1.21, 1.03 CPU usage: 1.93% user, 4.52% sys, 93.55% idle
MemRegions: 93261 total, 2125M resident, 72M private, 657M shared.
PhysMem: 5749M used (1364M wired), 595M unused.
VM: 489G vsizer, 1066M framework vsizer, 174811v(0) swaps, 355954(0) swapouts.
```

- `97999` **syspolicyd** 0.0 00:00:06 2 1 23 40 2824K 2640K 0B 920K 53M 2412M 97999 1 sleeping 0
- `97926` Google Chrom 0.0 00:23:02 4 0 58 132 8356K 6752K 0B 6034K 106M 824M 97923 97923 sleeping 501
- `97923` Google Chrom 0.0 00:57:03 34 1 375+ 442+ 33M+ 39M+ 0B 11M 322M+ 1030M+ 97923 177 sleeping 501
- `97477` Apple1DAuthA 0.0 00:00:01 3 2 38 48 404K 240K 0B 396K 46M 2412M 97477 177 sleeping 501
- `97238` Microsoft Po 0.3 11:10:39 11 2 255+ 4211+ 170M+ 121M+ 56M 41M 215M+ 1527M+ 97238 177 sleeping 501
- `92796` dbfsentst 0.0 00:02:79 1 0 7 27 32K 12K 0B 80K 20K 591M 248 89632 sleeping 501
- `92788` Dropbox 0.1 08:40:39 44 1 256+ 855+ 58M+ 50M+ 12K 47M 299M+ 1000M+ 248 1 sleeping 501
- `92740` adb_sos 0.0 00:00:47 5 0 83 62 468K 340K 0B 732K 40M 618M 92739 1 sleeping 501
- `92553` Console 0.0 00:07:48 3 0 158 221 416K 250K 0B 33M 33M 250M 92553 177 sleeping 501
- `89792` com.apple.We 0.0 05:43:72 8 3 372 1057 54M 34M 0B 20M 67M 355M 89792 1 sleeping 501
- `89788` Safari 0.0 16:47:59 13 1 2530 2811 134M 82M 812K 48M 585M 4237M 89788 177 sleeping 501
- `89632` dbfsentst 0.0 00:06:00 1 0 7 27 4172K 414K 0B 96K 414K 591M 248 89631 sleeping 0
- `89631` dbfsentst 0.0 00:02:02 1 0 14 26 40K 20K 0B 136K 528K 583M 248 1 sleeping 0
- `88379` com.apple.s 0.0 00:02:02 2 0 49 42 544K 368K 0B 384K 45M 2433M 88379 177 sleeping 501
- `83267` helpd 0.0 00:01:07 2 0 47 46 472K 384K 0B 696K 45M 2434M 83267 177 sleeping 501
- `82077` usmxd 0.0 00:01:10 3 0 44 45 384K 256K 0B 820K 55M 2423M 82077 1 sleeping 213
Process Management

You can put a process as **background** with `&` (ampersand) after a command. A background job will keep running until it finishes. This allows users to work on different tasks while the background job running. Don’t forget your background jobs are consuming resources (CPU, Memory, File I/O etc).

- **sleep** delay for a specified amount of time
- **&** run a process as a background job
- **Ctrl-z/fg** send a foreground job to background and vice versa

```
$ sleep 2
$ sleep 10 &
[1] 18506
$ fg
sleep 10
[1]+ Stopped sleep 10

$ bg
[1]+ sleep 10 &
```

$ bg

University of Southern California
Process Management

**ps**
- display currently running jobs

**kill/killall**
- terminate a process (not for PBS job)

```
$ sleep 10 &
[1] 27629
$ ps
PID TTY          TIME CMD
27362 pts/27   00:00:00 bash
27629 pts/27   00:00:00 sleep
27791 pts/27   00:00:00 ps
$ kill 27629
$ ps
PID TTY          TIME CMD
27362 pts/27   00:00:00 bash
28248 pts/27   00:00:00 ps
[1]+ Terminated    sleep 10
```
Redirect and Pipe

A special character > redirects output from commands into different channels. Two types of outputs are commonly used, standard output and standard error.

```
[~]$ env
MANPATH=/usr/share/man:
HOSTNAME=hpc-login2
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
...
[~]$ env > env.log

[~]$ cat env.log
MANPATH=/usr/share/man:
HOSTNAME=hpc-login2
TERM=xterm-256color
SHELL=/bin/bash
```
Redirect and Pipe (cont.)

A special character | pass output from one command to another command, called pipe. Many command can be daisy-chained by pipe.

grep | print lines matching a pattern
head/tail | show first/last several lines
sort | sort text alphabetically/numerically

Example: Print top 5 users who are consuming CPU except myself:

```bash
[~]$ ps axuw | grep -v ${USER} | sort -r -n -k 3 | head -n 5
```
And many more!

There are many useful, even fun, unix commands and utility programs that are beyond the scope of an introductory tutorial.

- grep matches things
- find finds things
- sort sort things
- sed string substitution
- awk ?
Thank you for attending!

Questions?

A special thanks to:
Avalon Johnson
and Dr. Ken-Ichi Nomura Ph.D.
for slides upon which this presentation is based