**What is UNIX?**

1. An operating system (un)like Windows; created in late 1960’s at AT&T Bell Labs

2. Designed to be a programmer’s operating system

3. Turned out to be a portable, multi-user, multi-tasking operating system - a first!

4. There are many different versions of Unix:
   - 4a. Apple’s OS X / macOS (and iOS!) is a user-friendly desktop
   - 4b. Linux is a clone of Unix which offers extremely good performance and is free
   - 4c. Therefore, Linux is the de facto standard for High Performance Computing (HPC) clusters

Ken Thompson (seated) and Dennis Ritchie, the creators of UNIX
All versions of Unix provide some sort of GUI, but...

1. Users interact with the cluster via a shell

2. Yes, the command line! It’s more lightweight, efficient, better suited for remote access, etc.

3. There are many different shells:
   3a. bash - most common; the default in OS X, Ubuntu Linux, and ACCRE
   3b. tcsh and zsh are examples of other shells some people prefer
The format of Unix commands is: `command [options][arguments]`

1. `ls` is a command
2. `-l` is an option to the `ls` command
3. `example1` is an argument to the `ls` command

```bash
$ ls example1 example2
$ ls -l
total 8
-rw-r----- 1 ken accretraining 69 Oct 13 2008 example1
-rw-r----- 1 ken accretraining 61 Oct 20 2008 example2
$ ls -l example1
-rw-r----- 1 ken accretraining 69 Oct 13 2008 example1
$ 
```
The Most Important Command of All

1. The **man** command displays manual pages; example at left is output of **man ls**

2. Displays a synopsis of how to use the command and a description of each option / argument

3. Long options are preceded by two dashes

4. You cannot assume that an option does the same thing with different commands

4a. Example: **-v** means “verbose” with many commands, but it means “doesn’t match” with **grep**

5. **command --help** displays similar information
The shell maintains a history of the commands you have previously executed (1,000 on ACCRE)

Up and down arrow keys scroll thru your history; left and right arrow keys move thru a command

Edits can be made by inserting or deleting text; pressing enter executes the command
The path separator is a forward slash

- There is no “C:” or “F:” drive - it’s all one big filesystem anchored at the root directory (“/”)

- File extensions, if they exist, have no meaning to the operating system

- File names are case sensitive
**Absolute Versus Relative Paths**

1. Assuming my current directory is `/home/ken/src`

2. The absolute path to `grantproposal.txt` is: `/home/ken/doc/grantproposal.txt`

3. The relative path to `grantproposal.txt` is: `../doc/grantproposal.txt`

4. The absolute path to `SlurmJob.out` is: `/scratch/ken/Project1/SlurmJob.out`

5. The relative path to `SlurmJob.out` is: `../../../scratch/ken/Project1/SlurmJob.out`

6. You should use whichever one is shorter ... or easier for you to remember!
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td><em>pwd</em> prints your present working directory</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td><em>ls</em> lists directories and files</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td><em>cd</em> changes directories</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td><em>mkdir</em> makes a directory</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td><em>rmdir</em> removes a (empty) directory</td>
</tr>
<tr>
<td><strong>5a</strong></td>
<td><em>rm</em> -r recursively deletes a directory tree</td>
</tr>
<tr>
<td><strong>5b</strong></td>
<td>Be very careful using it!!!</td>
</tr>
</tbody>
</table>

![Example output](image)
**COMMANDS FOR WORKING WITH FILES**

1. cat, more, or less display the contents of a file

2. **cp** copies files

3. **mv** moves (renames) files

4. **rm** removes files

5. The -i option makes **cp**, **mv**, and **rm** “interactive”
**AUTOCOMPLETING FILENAMES WITH THE TAB KEY**

1. You only have to type enough of a filename to ensure uniqueness and then you can **<TAB>**

2. If you haven’t typed enough to uniquely identify the file, press **<TAB>** twice for a list

3. Type enough additional to uniquely identify the file and then press **<TAB>** to complete!
There are 3 editors commonly available: **emacs**, **nano**, and **vim**

**emacs** is very popular with programmers

**vim** has the steepest learning curve, but is the fastest of the three

**nano** is the easiest to learn and is good for basic editing - **nano .bashrc**

The bottom two lines of the screen are reserved for **nano**

The arrow keys let you move around in the file, as does Control-Y and Control-V

You can easily insert and delete text

**Control-O** outputs (saves) a file; **Control-X** exits **nano**
**File Permissions**

1. All files have one owner (user) and one group associated with them.

2. Only the user may change the user, group, or permissions.

3. Permissions are read, write, and execute; they apply to the user, group, and others.

User ken has read, write, and execute permission.

Group members have read and execute, but not write permission.

Others have no permission.
<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Write</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>You can look at the file contents</td>
<td>You can modify the file contents</td>
<td>You can run the program</td>
</tr>
<tr>
<td>Directories</td>
<td>You can ls the directory</td>
<td>You can create new files, rename existing files, and delete files</td>
<td>You can cd to the directory</td>
</tr>
</tbody>
</table>
**CHANGING PERMISSIONS**

1. Use the `chmod` (change mode) command

2. Alphabetic method - add or take away (r)ead, (w)rite, e(x)ecute from (u)ser, (g)roup, (o)ther

3. Numeric method - read = 4, write = 2, execute = 1; total up for user, group, and other

```
$ ls -l example*
-rw-r--r--. 1 ken accretraining 69 Oct 13 2008 example1
-rw-r--r--. 1 ken accretraining 61 Oct 20 2008 example2
$ chmod ug+x,o-r example1
$ chmod 750 example2
$ ls -l example*
-rwxr-x---. 1 ken accretraining 69 Oct 13 2008 example1
-rwxr-x---. 1 ken accretraining 61 Oct 20 2008 example2
```
**Pattern Matching**

- `*` Matches zero or more instances of any character
- `?` Matches one instance of any character
- `[abc]` Matches any one character within the brackets
- `[0-9]` Matches any one character within the range defined in the brackets
- `[A-z]` Matches all letters, plus most punctuation characters, because this is an ASCII range

```
$ ls
example example1 example12 example1a example2 example3 exampleA
$ ls example*
example example1 example12 example1a example2 example3 exampleA
$ ls example?
example example1 example2
$ ls example[0-9]
example example1 example2 example3
$ ls example[0-9]?
example example1 example2 example3
$ ls example[A-Za-z]
example exampleA
```

**Finding Files and Folders with Find**

1. Syntax is: `find “where to start looking” -“criteria” “what to look for” -“what to do with it”`

2. Find all files starting at the current directory whose names are example followed by another character

3. Same as the 1st example, but instead of printing their filenames, ls them

4. Find all files starting at the current directory whose modification time is less than 3 days ago

5. Find all files starting at /scratch/ken whose name ends in “.err” and print them

6. Same as the previous example, but instead of ls’ing them, rm them!

7. Two criteria used: 1) file name, 2) file size (all files larger than 100 characters in this example)
1. `alias name="some value"

2. Now when you type `rm`, the shell will automatically replace it with `rm -i`

3. `alias ll="ls -la"

4. `alias mroe=more

5. `alias login="ssh ken@login.accre.vanderbilt.edu"`
**Shell Variables**

1. Many variables are set for you; `env` lists them
2. System variables are in all CAPITAL letters
3. `export my_var=some_value` - creates a variable
4. The `echo` command can be used to display the value of a variable
5. When referencing a variable, precede the name with a `$` sign

```
$ export claim2fame="Inventor of UNIX"
$ echo claim2fame
claim2fame
$ echo $claim2fame
Inventor of UNIX
$ echo $PATH
/usr/scheduler/slurm/sbin:/usr/scheduler/slurm/bin:/usr/lpp/mmfs/bin:/usr/local/bin:/usr/local/common/bin:/usr/bin:/usr/scheduler/slurm/bin:/usr/lpp/mmfs/bin:/usr/local/bin:/usr/local/common/bin:/usr/bin:/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/sbin:/usr/sbin:/bin:/usr/sbin:/bin:/usr/sbin:/bin:
```

Any aliases or variables you define on the command line are in effect only until you logout.

To make them permanent, simply add them to your .bashrc file in your home directory.

```bash
# Source global definitions
if [ -f /etc/bashrc ]; then
  . /etc/bashrc
fi

# User specific aliases and variables
alias ll="ls -la"
alias mroe=more

export PS1="$ "   # Redefine shell prompt
export PATH=$PATH:/usr/lpp/mmfs/bin
export claim2fame="Inventor of UNIX"
```
COMMAND SUBSTITUTION

1. Any command enclosed in grave accents is executed first and its’ output substituted in.

1a. $(command) does the same thing as `command`.

2. This can be used with other commands or to assign a value to a variable.

```bash
$ echo "Today's date and time is `date`"
Today's date and time is Tue Aug 23 15:09:11 CDT 2016
$ export right_now=`date`
$ echo $right_now
Tue Aug 23 15:09:53 CDT 2016
```

⚠️ Not single quotes!
Any shell has 3 filehandles open by default

0: stdin - standard input, defaults to keyboard, file descriptor 0

1: stdout - standard output, defaults to screen, file descriptor 1

2: stderr - standard error, defaults to screen, file descriptor 2
Input / Output Redirection

1. Input redirection - e-mail yourself a file: `mailx ken.thompson@att.com < example1`

2. Output redirection - `myprogram > output.log`

3. Error redirection - `myprogram 2> error.log`

4. Output and error redirection to different files - `myprogram > output.log 2> error.log`

5. Output and error redirection to the same file - `myprogram > combined.log 2>&1`

6. But don’t forget the Unix philosophy!
Pipes take the output of one command and make it the input to another command

Analogous to plumbing pipes

Filters are commands which can accept input from another command and also produce output

Syntax is `command | filter_command`

Multiple pipes and filters can be strung together:

```
cat *.out | grep ERROR | wc -l
```
**Some Useful Filter Commands**

1. `wc` - word count
2. `grep` - get a regular expression and print it
3. `sort` - very powerful sort utility
4. `uniq` - filter duplicate lines
5. `cut` - cuts specific fields or columns
6. `sed` - stream editor, does search and replace
Or are you ready to go forth and compute?!