Preface
Quick Note

- Many flavors of Unix, many for the PC platform, including many distributions Linux.
  - Collectively, they will be referred to as *nix

- Where there’s a difference, these notes discuss Linux, and many of the utilities from the gnome toolkit

- So, on some other *nix platforms, you might notice slightly different behavior, maybe some missing options, some other small differences
  - E.g., emacs is the default Linux editor, rather than vim
  - Linux pushes info pages (but still has man)
Flavors of Unix

There are many flavors of Unix used by many people. This is *not* a complete listing:

- SysV (from AT&T)
- BSD (from Berkeley)
- Solaris (Sun)
- IRIX (SGI)
- AIX (IBM)
- OSF1 (DEC)
- Linux (free software)
  - Thank Linus Torvalds
Most flavors of *nix provide the same set of applications and services (commands, shells)

Although these programs are not directly part of the OS, they are standardised enough that learning your way around one flavor of *nix is sufficient

Unix got its start in the early 70s

Was used (and grown) by engineering and science types
Since OS X, Mac runs on BSD Unix

You can get many of the gnome command-line utilities discussed here just by installing XTools

Or, Homebrew, a package manager for MacOS, provides access to coreutils, and other gnu utilities

  Installed separately, can be made default

You can simply open a terminal window, ssh to the department machines, and work there
Notes for Windows Users

- Cygwin – A *nix-like subsystem, runs on top of Windows
  - Try MobaXTerm. Very nice front-end to cygwin, with an X-Server
- Linux Bash Shell on Windows 10 – User space and bash shell, running natively on Windows\(^1\)
- You can install some flavor of Linux on a partition of your disk
- Or, run Linux inside a Virtual Machine

----

\(^1\) I’m not in love w/it
**Objective**: Introduce students to the features of *nix, and the Unix Philosophy (a collection of combinable tools and environments that support their use)

- Basic commands
- File system
- Shell
- Filters (*more*, *grep*, *sort*, *wc*)
- Pipes, file redirection
An *Operating System* controls (manages) hardware and software

- Provides support for peripherals such as keyboard, mouse, screen, disk drives, etc.
- The OS typically manages (starts, stops, schedules, etc.) applications
- Software applications use the OS to communicate with peripherals (screen, networking, etc), and with other applications
Kernel (OS)

- Interacts directly with the hardware through device drivers
- Provides sets of services to programs, insulating these programs from the underlying hardware
- Manages memory, controls access, maintains filesystem, handles interrupts, allocates resources of the computer
The Shell

- Helps manage user applications
- An *interactive shell* is the user interface
  - Responds to user commands
- A *desktop* is a GUI shell
- A shell *is* just another program
Structure of the *nix System

There are many standard applications:
- Filesystem commands
- Text editors
- Compilers
- Text processing
Logging In
Logging In

You can:

- Sit at the *console* (the computer itself)\(^1\)
- Connect from any remote computer connected by a network, via SSH, e.g.
- Remember, usernames and passwords are case sensitive!

\(^1\)Note, if you sit at any of the department Linux machines, your home directory will be mounted there. You shouldn’t notice a difference.
Incorrect Login

- You will receive the “Password:” prompt even if you type an incorrect or nonexistent login name.
- Nothing will happen while you type your password. It’s fine.

Can you guess why?
Connecting Remotely

- From a *nix machine, or a Mac, just open terminal, use `ssh`
- Windows doesn’t have SSH built in
  - Any SSH client would do
  - I recommend PuTTY
  - Windows 10 is supposed to have the SSH stack, but I’ve not seen it yet
- To avoid always typing your password, search the Web for `ssh-keygen`
- Keep your passwords and keys safe!
CS Dept. Machines

- See http://www.cs.drexel.edu/~kschmidt/Ref/csLogin.html
- All CS machines are running Linux
  - tux.cs.drexel.edu – a farm you may connect to from anywhere on the ’Net.
  - Lab machines – any of the desktop machines in the labs
- Your files are backed up daily (nightly)
Usernames

Typically (or, on tux, anyway):

- A sequence of alphanumerical characters (there might be some others)
- Length no more than 8
- The primary identifying attribute of your account
- Unique (so, typically how I know and refer to you)
- Used as your email address
- The name of your home directory is related
  - On the CS machines, if your ID is abc123, then your home directory is /home/abc123
Passwords

- A secret string, not even the system knows
  - System hashes (encrypts) the password, compares it to the stored hash
- Should have at least 6 characters
- Should contain upper- and lower-case letters, numbers, and even other characters
- Don’t use anything that appears in any dictionary
- Don’t use anything that can be gleaned from your past, or your current likes
- Consider a line in a song, or poem. Use the first letter of each word
Files & Permissions
The user’s personal directory

- All home (users’) directories on tux are in /home: E.g., /home/kschmidt

- Where all your files go (hopefully organised into subdirectories)

- Mounted from a file server – available on any department machine you log into
Home Directory

- **current working directory** (CWD) when you log in
- `cd` (without an argument) takes you home
- Location of many startup and customisation files:
  - `.bashrc` `.vimrc` `.forward` `.plan`
Files and File Names

- A file is a basic unit of storage (e.g., the disk)
- Every file has name
- Filenames are case sensitive
- Unix filenames can contain any character except the slash ( / ) and the null character
  - Some characters, like shell metacharacters, make it more difficult to refer to the file
Every file has at least one name
  - See `ln`, inodes
Each file *in the same directory* must have a unique name
Files in different directories can have identical names
Files that start with a `. are, by default, hidden by `ls`, and other utilities
Directories

- Sometimes called a *folder*
- A *directory* is a special sort of file
  - holds information about other files
- Container for other files (including directories)
- Other file types include *symbolic links* (just like shortcuts), *named pipes*, *block special files* (disks, USB drives)
Unix Filesystem

- A hierarchical system of organising files and directories
- The top level in the hierarchy is called the *root*
  - Holds *all* files and directories in the filesystem
  - Its name is `/`
- Filesystem may span many disks, even across a network
Unix – Introduction
Kurt Schmidt
Preface
Intro
Logging In
Filesystem
Commands
Files & Permissions
Permissions
Editors
The *pathname* of a file includes the name of the file, the directory that holds the file, the directory that holds *that* directory... up to the root

The pathname of every file in a given filesystem is unique

Absolute pathnames start at the root, drill down through successive subdirectories

The forward slash, `/`, separates path components

- So, can’t be used in a filename
- The only other character is `\0`, the null-terminating character
Pathnames – eg.

- `/usr/bin/w3m`
- `/home/kschmidt/Public/README`
Absolute Pathnames

- The pathnames, above, are *absolute* pathnames
- Start at the root
- Uniquely identify files
- There are 2 absolute paths that don’t, apparently, start at the root:
  - ~kschmidt/ ⇔ /home/kschmidt (to refer to any user’s home directory)
  - ~/ – **Your** home directory. So, relative to login, $USER
Relative Pathnames

- Prefixed with the current directory, $PWD
- So, relative to the current directory

```
$ cd /home/abc12
$ ls public_html/
index.html schedule.html
$ ls Public
ls: cannot access 'Public': No such file or directory
$ cd /home/kschmidt
$ ls public_html/
index.html
$ ls Public
README
```
Special Relative Paths

- . – the *current* directory
- .. – the *parent* directory

```
$ cd ~abc12
$ pwd
/home/abc12
$ ls -F ../kschmidt/
public_html/ Public/
$ cp ../kschmidt/Public/README . # copy that file here
```
The hierarchy can actually span parts of many disk drives (partitions)

Even partitions on other computers
Bash is the default shell, and the one we’ll discuss here

- Tokens are separated by whitespace
- Shell expects the first token to be a command\(^1\)
- All subsequent tokens are arguments
- Arguments that start with a dash, -, or two dashes, are called *options* (generally, Posixly)
  - Used to modify the behavior of the command
  - Note, not all utilities are Posix compliant (e.g, `tar`)
- Non-option arguments are data passed to the command

---
\(^1\) Commands may be preceded by a sequence of variable assignments.
Command Syntax

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>utility, to list contents of a directory</td>
</tr>
<tr>
<td><code>-a</code></td>
<td>option, to include hidden files (all)</td>
</tr>
<tr>
<td><code>-l</code></td>
<td>option, spit out long listing</td>
</tr>
<tr>
<td><code>Labs/Unix</code></td>
<td>argument, directory to list</td>
</tr>
<tr>
<td><code>Lectures</code></td>
<td>argument, another directory to list</td>
</tr>
</tbody>
</table>

Note, short options which don’t require arguments (optargs) can generally be stacked:

```
s -al Labs/Unix Lectures
```
Options, Optargs

Options come in 2 flavors:

- **Toggles**, or flags. On or off
- Options which, in turn, need information

```
tail -f -n30 error.log
```

- `-f` – A toggle, tells `tail` to update (follow)
- `-n 30` – Tells `tail` to display 30 lines
Traversing the Filesystem

- `ls` – lists file or contents of a directory (current directory by default)
  - `-a` – show hidden files (all)
  - `-o`, `-l` – long (and longer) listing
  - `-d` – directory (don’t list out the contents)
  - `-F` – Decorate names depending on filetype

- `pwd` – print the working (current) directory

- `cd` – change directory
  - By default, takes you home

1 Also see Bash’s `pushd` and `popd`
Getting Help – man, info

Man Pages:

```
man ls
```

- Get information on any properly installed utility (ls, grep, etc.)
- Can do a keyword search: `man -k music`
- Split into sections (note them)
- Flat, unexciting, but very useful

Info pages are often provided

- Hierarchical; not flat
- Navigation uses emacs-like bindings
- If no info page, it’ll display the man page
Viewing Text Files

- **cat** – *concatenate*, send to stdout. View text files
- **less** **more** – paging utility. **h** for help, **q** to quit
- **od** – *octal dump*. For viewing raw data in octal, hex, control chars, etc. Useful for looking for non-printing characters in your code.
Copying, Removing, Linking

- **rm** – *remove* file
  - `-r` – *recursive*. Careful, here
  - `-f` – *force*. Ignore nonexistent files

- **cp** – *copy*
  - `-i` – *interactive*. Ask before overwriting destination file (if it exists)

- **mv** – *move*. Also, rename, you can give the file a different name as you move it
  - `-i` – *interactive*. Ask before overwriting destination file (if it exists)
Directories

- `mkdir` – *make directory*
- `rmdir` – *remove directory*
  - Safe; it won’t remove non-empty directories
  - Compare to `rm -rf` (and be careful)
- Directories can be moved/renamed using `mv`
- Entire directories can be copied using `cp -r`
  - See `rsync`
Archiving

- **tar** – *tape archive*
  - makes one large file from many smaller files
- **gzip**, **gunzip** – One (of many) compression utilities
  - **bzip2**, **compress**, **xz**, **zcat**, **zip**
- **tar** on Linux does gzip compression (and others) using the *z* option:

  ```
  tar czf 571back.tgz CS571
  tar xzf assn1.tgz # or .tar.gz
  ```
Filters

Programs that read some input, perform some transformation, write out the results

- `head`, `tail` – Displays first (last) n lines of input
- `grep` – Search input using *regular expressions*
- `sort` – Sorts input by lines (lexically, or numerically)
- `uniq` – *Unique*, removes identical, adjacent lines
- `wc` – *Word count* (line count, character count)
- `cut` – Select fields of a line
- `tr` – *Translate*
Some Other Utilities

- **date** – Print current date and time
- **time** – Does *not* show you the current time
- **who** – Print who is currently logged in
- **finger** *user* – more information about *user*
- **du** `-sh` – Disk usage summary, human readable
Files & Permissions
Every file has some attributes stored by the filesystem

- Times of creation, last change, last modify, last access
- Size
- Owner and group
- Permissions
- ACLs
Time Attributes

- `stat file` shows all of these attributes
- `ls -o` shows the last modification time
- `ls -ot` sorts by modification time
- See `find's -ctime -mtime -atime`
ls -l

-rw-rw-r-- 1 kschmidt 265-inst 20749 Oct 30 11:37 unix.tex

- File type and mode bits
- Number of hard links
- Owner
- Group
- Size (see -h)
- Modification time
- Filename
File Permissions

- Each file has a set of permissions that controls who can do what to the file
  - Note, ACLs are newer, ride on top of these permissions
- There are three types of permissions
  - r – read
  - w – write
  - x – execute
- Permissions are set for these entities
  - user (the file’s owner)
  - group (members of the file’s group)
  - other (world; everybody else)
Type & Permission Bits

- Plain file
- Directory
- Symbolic link
- Named pipe
- Character file (keyboard, mouse, etc.)
- Block (disk drives, USB, etc.)

-user’s permissions
-group’s permissions
-others’ permissions
Files:
- r – allowed to read
- w – allowed to write
- x – allowed to execute

Directories:
- r – Can list out the directory (view contents)
- w – allowed to create and remove files
- x – allowed to "enter" the directory, change to subdirectories, edit files
Changing Permissions – `chmod`

`chmod mode(s) file(s)`

- `chmod` command changes permissions on a file or directory
- Modes can be expressed symbolically, or as octal values

```bash
chmod 755 Public # Typical perms for a public directory or executable
chmod 644 README # Typical perms for a public file
chmod a+x script # Add execute permissions for everybody
```

- `-R` – `chmod` goes recursive
- See `+X`
Consider each set of permission bits as a 3-digit binary number:

- \( r - 4 \)
- \( w - 2 \)
- \( x - 1 \)

A permission (mode) for all three sets is a 3-digit octal number:

- 755 – rwxr-xr-x
- 640 – rw-r-----
- 711 – rwx--x--x
$ chmod 700 CS571
$ ls -o Personal
  drwx------ 10 kschmidt 4096 Dec 19 2004 CS571/
$ chmod 755 public_html
$ chmod 644 public_html/index.html
$ ls -ao public_html # $
  drwxr-xr-x 16 kschmidt 4096 Jan 8 10:15 .
  drwx--x--x 92 kschmidt 8192 Jan 8 13:36 ..
  -rw-r--r-- 5 kschmidt 151 Nov 16 19:18 index.html
$ chmod 644 .plan
$ ls -o .plan
  -rw-r--r-- 5 kschmidt 151 Nov 16 19:18 .plan
Can modify (add or remove) permissions, or set permissions absolutely

\[ [ugo\text{a}] [+-=] [rwx] \]

- u – user
- g – group
- o – other
- a – all

+ – add permission(s)
- – remove permission(s)
= – set permission(s)
chmod – Examples

```bash
$ ls -al foo
-rw-rwx--x 1 hollingd grads foo
$ chmod g-wx foo
$ ls -al foo
-rwxr----x 1 hollingd grads foo
$ chmod u-r .
$ ls
ls: .: Permission denied
```
Editors
In this course you will use either `emacs` or `vim`.

It is well worth learning a good, richly-featured editor:
- Syntax highlighting
- Regular expression search and replace
- Keyboard navigation
- Extensible through macros
- Much more

GUI versions of `emacs` and `vim` exist.

Take the time to learn navigation, w/out the mouse and the arrows:
- You won’t always have a GUI running
- After a bit of practice, the mouse simply slows you down
emacs VS. vim

I, for good or ill, am a VI guy, so, I’ll better be able to answer those questions

- **vim** – Vi IMproved
  - Was the standard Unix editor
  - Built on `ed`
  - Shares some syntax with `sed`, and many other utilities, including, amusingly, `mutt` and `cmus`, my mp3 player

- **emacs** is written in (Emacs) LISP
  - A bit more powerful than `vim` (you can run a shell inside, or, play Tetris)
  - The default editor for Linux

Both are excellent text editors. Both have a steep-ish initial learning curve. Put in the time, learn one!
Shell Readline Library

- Bash (and other interactive programs, such as Maple, Python, etc.)\(^1\) allow you to edit the command-line as you might edit a text file
- By default, you can use a subset of Emacs key bindings\(^2\)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-p</td>
<td>Up (back) a line</td>
<td>C-n</td>
<td>Down a line</td>
</tr>
<tr>
<td>C-b</td>
<td>Left a character</td>
<td>C-f</td>
<td>Right a character</td>
</tr>
<tr>
<td>M-b</td>
<td>Left a word</td>
<td>M-f</td>
<td>Right a word</td>
</tr>
<tr>
<td>C-a</td>
<td>Beginning of line</td>
<td>C-e</td>
<td>End of line</td>
</tr>
<tr>
<td>M-&lt;del&gt;</td>
<td>Kill word before cursor</td>
<td>C-k</td>
<td>Kill to end of line</td>
</tr>
<tr>
<td>M-d</td>
<td>Kill word after cursor</td>
<td>C-r</td>
<td>Regex reverse search</td>
</tr>
</tbody>
</table>

\(^1\) see `~/.inputrc`  
\(^2\) This is only a partial list
You can change to Vim key bindings:
\[
\text{set -o vi}
\]

Some of the common, helpful bindings (Remember to hit Esc)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>Up (back) a line</td>
</tr>
<tr>
<td>h</td>
<td>Left a character</td>
</tr>
<tr>
<td>b</td>
<td>Left a word</td>
</tr>
<tr>
<td>0</td>
<td>Beginning of line</td>
</tr>
<tr>
<td>db</td>
<td>Delete word back</td>
</tr>
<tr>
<td>dw</td>
<td>Delete word forward</td>
</tr>
<tr>
<td>v</td>
<td>Open current line in vim editor</td>
</tr>
<tr>
<td>j</td>
<td>Down a line</td>
</tr>
<tr>
<td>l</td>
<td>Right a character</td>
</tr>
<tr>
<td>w</td>
<td>Right a word</td>
</tr>
<tr>
<td>$</td>
<td>End of line</td>
</tr>
<tr>
<td>d$</td>
<td>Kill to end of line</td>
</tr>
<tr>
<td>/</td>
<td>Search back</td>
</tr>
</tbody>
</table>

---

\(^1\text{Add (uncomment) this line to your \texttt{~/.bashrc} file.}\)