Unit 2: Shell Commands

Reading: *PCfB* Ch 4 (“Command-Line Operations”) and Ch 5 (“Handling Text in the Shell”)

- History: terminals, Unix, shells
- Simple commands
- Paths, directories, working directory
- Moving, copying, renaming, deleting files
- Options
- Command line editing
- Configuration file

Project	Milestone Exam
Shell Commands (10 points)	Exam 2 (20 points)

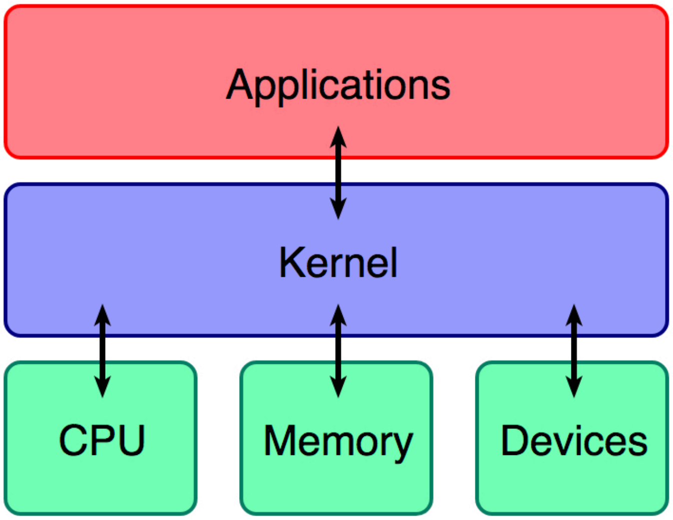
You should be able to complete the project and take the exam in lab in Week 2 (Apr 13).

Background: Unix

Unix is an **operating system**

- software that controls a computer's hardware
- makes it possible for application programmers to write programs that run on different hardware

A picture from Wikipedia:



The green layer represents **hardware**

- each machine is unique (different CPU chip, amount of RAM, ...)

The blue layer is the **kernel** (inner layer) of software

- written by programmers at Apple, Dell, Acer, etc to work with their company's machines

Origins

The name comes from an operating system develop at AT&T Bell Labs in the early 1970s

- a common (and expensive) OS at the time was named MULTICS ("multiuser ...")
- people who developed Unix wanted a simpler, streamlined OS for programmers to use

Commercial computer companies wrote versions for their own systems

- HP-UX
- Irix
- many other names ending with "x"

These were not free: the companies paid license fees to AT&T

Jargon: the name `*nix` refers to a generic Unix system

Open Source

A project named GNU (“Gnu’s Not Unix”) had the goal of creating a brand new system

- same basic organization with kernel and applications
- all new code so no license fees required

A founding principle of the GNU project: software should be free, to encourage creativity

“Open Source” means anyone can obtain the program, modify it, give it away to others, sell it, …

Linux

The GNU project focussed mainly on applications

In 1991 Linus Torvalds released an open source implementation of a full Unix kernel

Shells

The idea of a **command line interface** dates to the 1960s

- typical commands worked with files: “create a file named X”, “move X to a folder named Y”, “delete the file named Z”, …
- other commands launched applications: “run the accounting software to create this month’s report”, …
- users typed commands on a typewriter (later on “video display terminals”)
- any output generated by a command was printed on the terminal

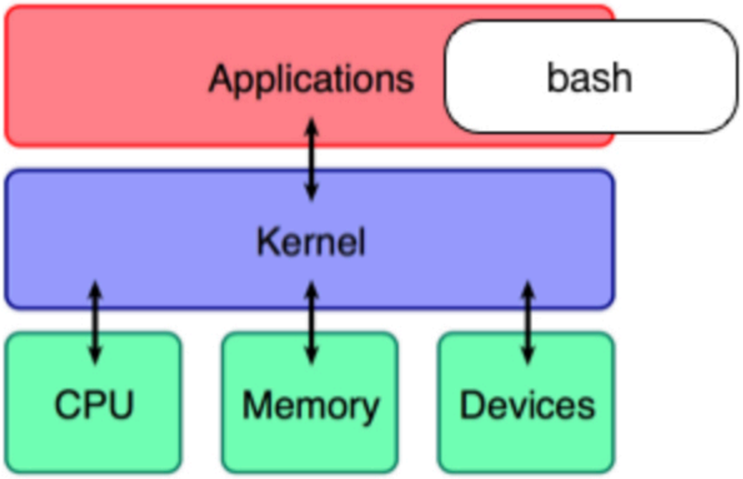
Shells are Applications

The program that read commands and printed results is itself an application called a **shell**

- the original program was named **sh**
- many others were developed later, adding new features: **`csh`**, **`bash`**, **`ksh`**, …

`bash` is now a de facto standard

- the default shell in macOS, most Linux implementations
- now available to download for Windows 10



Terminal Emulator

To use a command line interface on a modern OS run an application called a **terminal emulator**

- as the name implies, this is an application that mimics an old-fashioned video display terminal

macOS

The application is named Terminal, you'll find it in a folder named Utilities inside your Applications folder

Windows 10

When you start bash it will run a program named Command Prompt

Note: do not start Command Prompt from the Start menu (if you do the program expects you to type MSDOS commands, not Unix commands)

Linux

The application is named Terminal, it's probably on your desktop or in your top level application menu

REPL

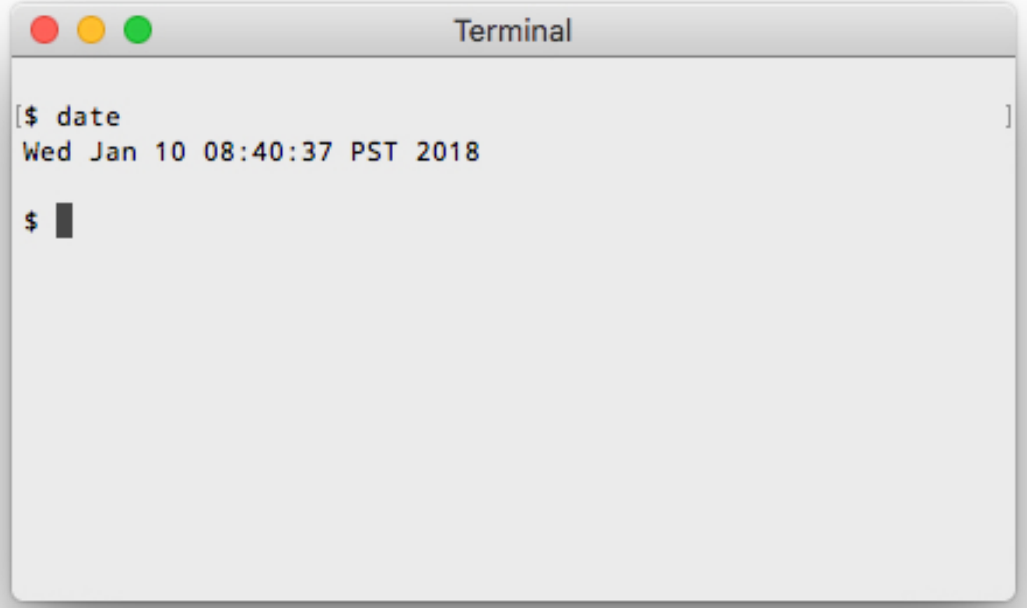
The shell running in the terminal emulator uses a **read-eval-print loop** (REPL):

- it prints a prompt character to let you know it is ready for you to type a command
- when you hit the enter/return key the shell executes the command
- if the command generates any output it is displayed in the window

Example

This is what my terminal window looked like after I typed the `date` command (which asks the OS to print the current time and date).

The dollar sign is the prompt character used by `bash`

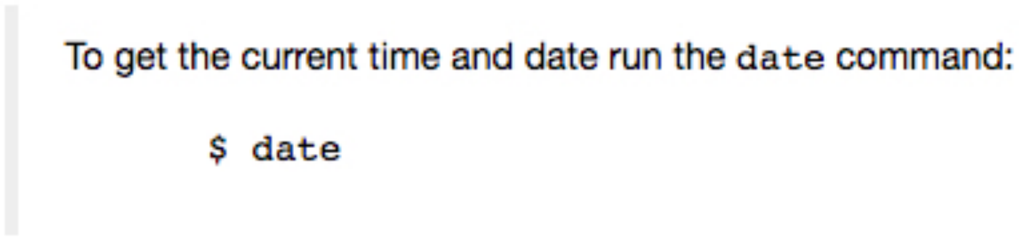


```
$ date
Wed Jan 10 08:40:37 PST 2018

$ 
```

Typographic Conventions

When you're reading about shell commands in books, documentation, blog posts, etc, you will often see something like this:



To get the current time and date run the `date` command:

```
$ date
```

The dollar sign at the front of the line is the prompt.

the dollar sign is not part of the command

Do not type the dollar sign -- just type (or sometimes, when you see a long command, copy and paste) the characters that follow the prompt.

Note: other common prompt characters are a percent sign (csh) and a greater-than sign (MSDOS).

Warning: *shell commands are case-sensitive*

Command names, file names, etc must be spelled exactly

Working Directory

In macOS and Windows files can be collected and organized in folders

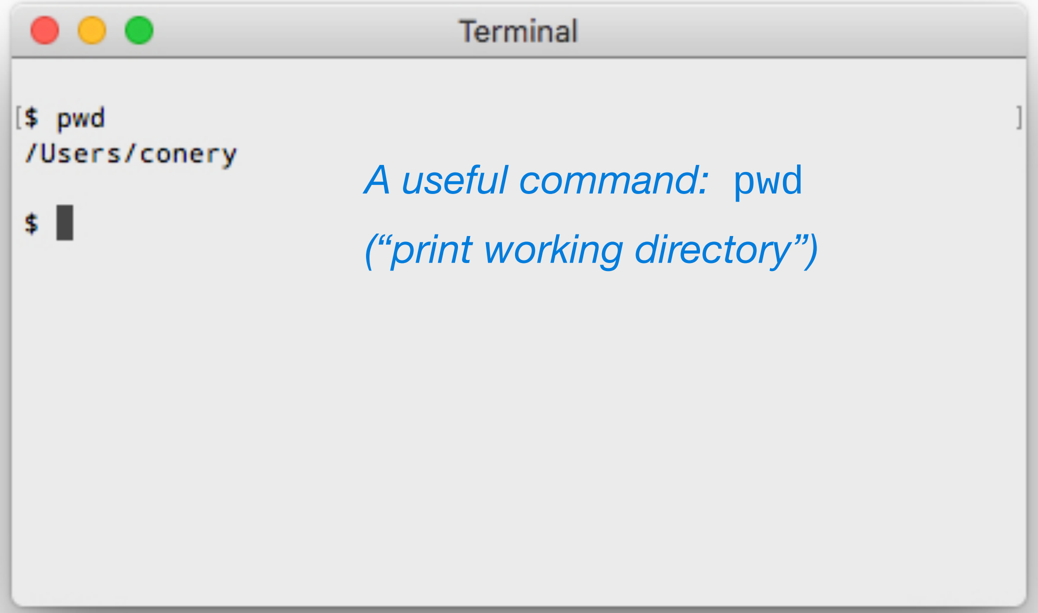
In Unix a folder is called a **directory**

An important concept is the **working directory**

- if a command needs data from a file the shell looks for the file in the current working directory
- if the output of a command is saved in a file the new file will be created in the current working directory

When a shell first starts the working directory is your **home directory**

- this is the same folder/directory you see when you open a new Finder (macOS) or File Browser (Windows)



```
[$ pwd
/Users/conery

$ 
```

A useful command: pwd

("print working directory")

Paths

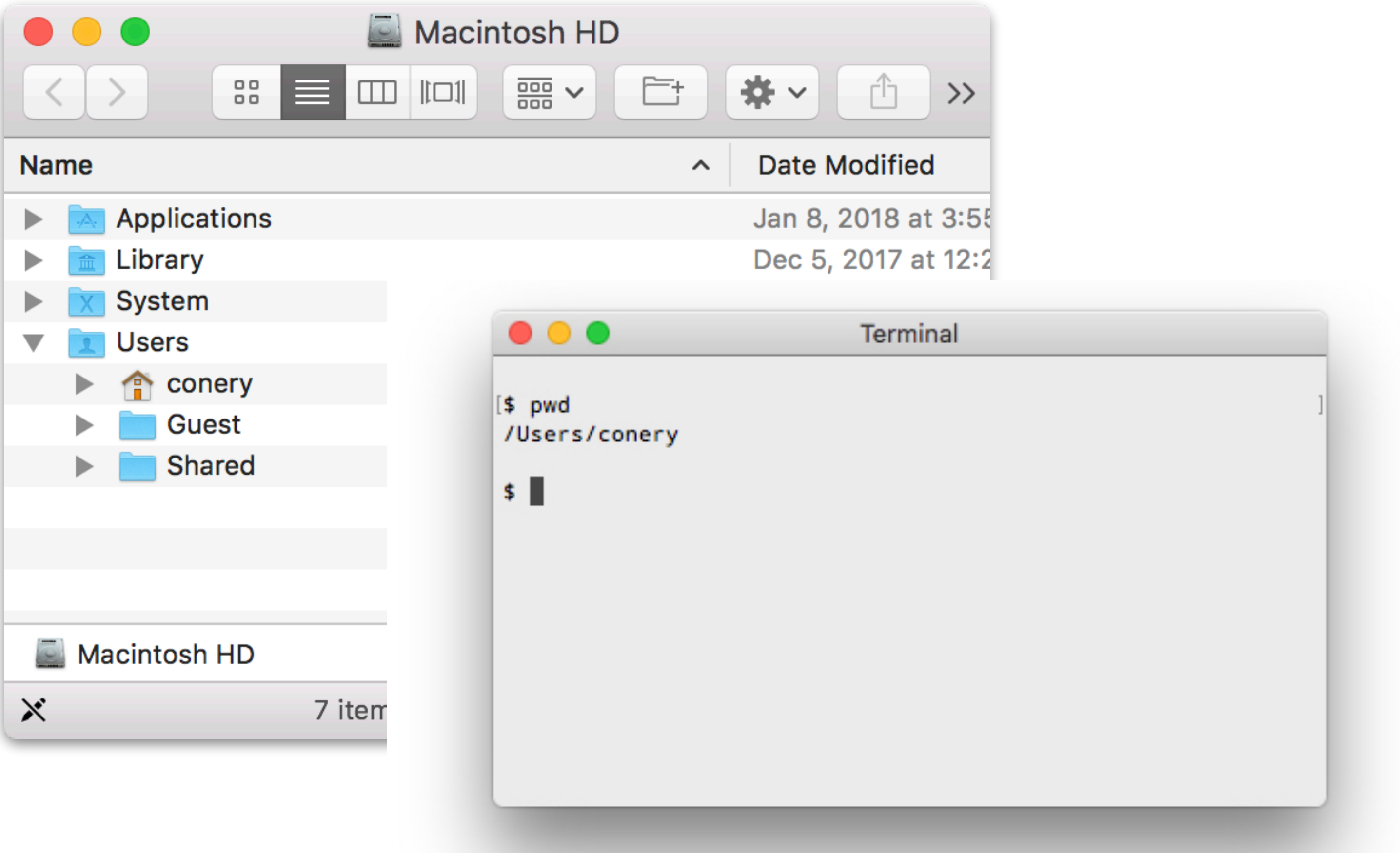
The output from the pwd command shown on the previous page is an example of a **path**

- a path is a series of folder names separated by slash characters

If the path starts with a slash it is an **absolute path**

- the slash at the front refers to the root directory

Otherwise the path is a **relative path**



```
[$ pwd
/Users/conery

$ 
```

List Directory Contents: `ls`

The `ls` command is good way to experiment with path names

Here are some examples from my laptop

```
$ pwd
/Users/conery
```

The current directory is my home folder

```
$ ls
Admin
Applications
Backups
Books
Classes
Desktop
…
```

With no argument the ls command prints the names of files and folders in the current directory

```
$ ls Classes
410
Archives
MAPS
Sandbox
```

Display the contents of the folder named Classes

ls Examples (cont'd)

```
$ ls Classes/410
documents
downloads
units
...
```

The argument can be a path — this means "list the contents of the 410 folder inside the Classes folder"

```
$ ls Books/Textbooks
A First Course in Mathematical Statistics.pdf
Biological Sequence Analysis.pdf
Introduction to Bioinformatics Algorithms.pdf
```

```
$ ls /System/Library
AWD
Accessibility
...
```

Note this example has an absolute path name

```
$ ls /Users/conery/Classes
410
Archives
...
```

Another way to show the contents of my Classes folder

Be Careful with Spaces in Names

The example above showed that macOS (and Windows) allow spaces in file names

However, you cannot have a space in a name in a path in a shell command

Example: suppose you make a folder named My Data using the Finder. If you try to look at the contents with a shell command you'll get an error message:

```
$ ls My Data
ls: Data: No such file or directory
ls: My: No such file or directory
```

There are workarounds:

(1) enclose the file name in single quotes

```
$ ls 'My Data'
...
```

(2) put a **backslash** before the space:

```
$ ls My\ Data
...
```

Other characters to avoid: parentheses, brackets, asterisk, …

Best to stick with letters and digits and if possible avoid spaces.

Change the Working Directory: cd

If you find yourself doing a lot of work in a directory you can save some typing by making it your working directory

```
$ ls Classes/410
documents
downloads
units
...
```

List the names in the 410 folder

```
$ ls Classes/410/units
0_intro
1_text
2_shell
...
```

```
$ cd Classes/410
410
Archives
...
```

Make 410 the current directory

```
$ ls units
0_intro
...
```

Now I don't have to type `Classes/410` *at the start of each path*

Terminology

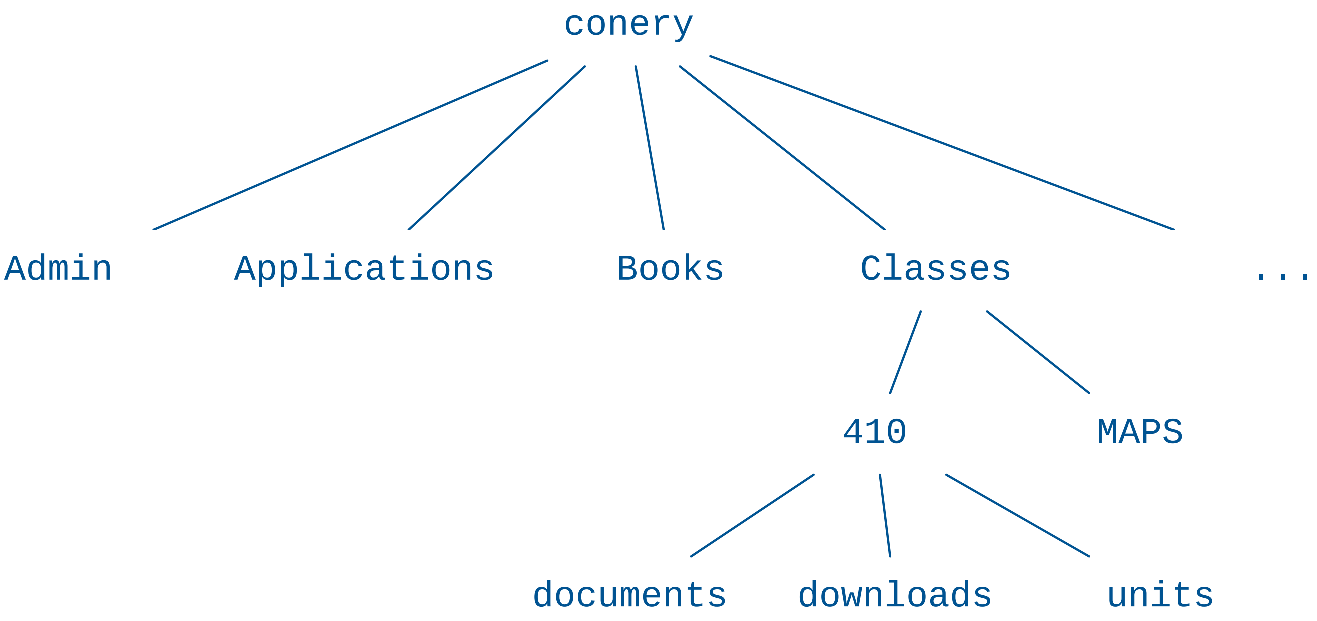
When we change the working directory we often say “we’re moving to…” or “we’re going to…” the new directory

- you’ll see this terminology a lot when you read instructions online for how to use or install a program

cd is a verb: “cd to the project directory…”

We also talk about moving “up” or “down” in a file hierarchy

- directories are often drawn as trees (in the CS sense of the word)
- the root of the tree is at the top



Special Characters in Path Names

Several characters have special meanings when they are used in paths

Tilde

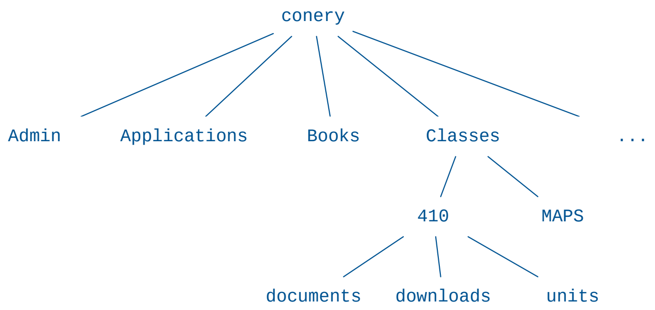
~ means “home directory”

Find it in the upper left of an American keyboard — you’ll have to hold the shift key

```
$ pwd
/Users/conery/Classes/410

$ cd ~/Books

$ pwd
/Users/conery/Books
```



Dot-Dot

Two periods (..) mean “the parent directory”, *i.e.* one step up in the tree

```
$ pwd
/Users/conery/Classes/410

$ cd ../MAPS

$ pwd
/Users/conery/Classes/MAPS
```

Working with Text Files: `wc, head, tail, cat`

The `wc` ("word count") command displays information about a file. It prints the number of lines, "words", and characters in the file.

Demo

Let's see how to use wc to answer the questions on Project 1:

- use the cd command to go to the folder where the file was saved (I have a copy in my Bi 410 folder):

  ```
  $ cd Classes/410/sandbox
  ```

- check to see if the file is there:

  ```
  $ ls
  ```

- now we can use wc to get some information about the contents of the file:

  ```
  $ wc ThalassocalyceData.txt
         7       62       577
  ```

  This file has 7 lines, 62 words, 577 characters

  Note: a "word" is any string of characters surrounded by whitespace…

Aside: Filename Completion

Before we continue with shell commands here is a “power user” skill that is very useful

It’s called “completion”, and it’s best described using an example

- the filename “ThalassocalyceData.txt” is easy to misspell
- but all I need to do is type “Tha” and then **hit the tab key**
- since there is only one file in this directory that starts with the letters “Tha” the shell types the rest of the name for me!

The details:

- if more than one file matches the letters you typed so far you’ll hear a “beep”
- type two tabs to see the list of all files that match what you typed
- you can use completion to create long path names, using the tab key to complete each part of the path if you want

You can learn this skill at your own pace

I’m including it here at the beginning of these notes because you’ll see me doing it during lecture (I’m not that good at typing!)

Working with Text Files (cont'd)

There are three commands that will display all or part of a text file

- `head` prints the first few lines in a file
- `tail` prints the last lines
- `cat` prints the entire file

In each case the argument is the path to the file you want to print

```
$ cd Classes/410/downloads/pcfb

$ ls
examples
sandbox
scripts
```

This folder has data and other files described in the textbook

You can download the file from Canvas

Print the entire *Thalassocalyce* data file:

```
$ cat examples/ThalassocalyceData.txt
ConceptName    Depth    Latitude Longitude...
Thalassocalyce 348.7    36.71804 -122.0574
...
Thalassocalyce 1509.6   36.58464 -122.52111
```

Working with Text Files (cont’d)

Specify the number of lines you want `head` or `tail` to print

- type a dash followed by a number
- the default is 10

```
$ head -1 examples/ThalassocalyceData.txt
ConceptName     Depth    Latitude Longitude    ...

$ tail -2 examples/ThalassocalyceData.txt
Thalassocalyce 100.85   36.726974    -122.04878   ...
Thalassocalyce 1509.6   36.584644    -122.52111   ...
```

Aside: Command Line Editing

Another very useful "power user" skill is **command line editing**

- type `^P` or `^N` to move to the previous/next shell command
- `^B` or `^F` moves the cursor backward/forward within the current line
- use the DELETE key to erase characters
- `^A` and `^E` move the cursor to the start/end of the current line

Standard notation: *a caret before an uppercase letter means "hold the control key when typing this letter"*

You can also use the arrow keys to move up/down/left/right

Here's an example: after typing the `head` command shown on the previous page, suppose I now want to use `tail` to look at the last line

I don't have to type the entire command again — I just need to edit the previous command to change "head" to "tail"

- hit the up arrow to display the previous command
- type `^A` to move the cursor to the front
- hold down the right arrow until it moves past "head"
- delete "head", type "tail", hit return (I don't need to put the cursor at the end of the line)

Another skill to learn at your own pace

Also learn how to move forward/backward one word at a time (ESC-F or ESC-B) or delete entire words with one keystroke

Copying, Moving, Deleting, Renaming Files ⚠️

I'm describing these operations in a single group because they all have one important thing in common

💣😟😱 **Warning!** 😭😡🍸

If you use a shell command that deletes or replaces a file the old file is **immediately erased**

- the file is not moved to your trash can
- you cannot recover the file from the trash
- you will never see that file again
- your program / data is lost
  [unless you know someone who works for the NSA]

*(but see the section on "configuration files"
below to see how to set up your shell so these
commands always print a warning first)*

Copy a File: cp

The `cp` command has two arguments

- the first is the name of the file you want to copy
- the second is usually the name of a directory where you want to put the copy

This example copies the Thalassocalyce data to the 410 folder inside my Classes directory:

```
$ pwd
/Users/conery/Classes/410/downloads/pcfb/examples

$ cp ThalassocalyceData.txt ~/Classes/410
```

If the second argument is a file name (not a directory) name the copy will be given that name:

```
$ cp ThalassocalyceData.txt ~/Classes/410/example.txt
```

⚠ if there is already a file named `example.txt` in that location it is deleted without warning!

Another Special Character

We saw above how .. (two dots in a row) means “the parent directory”

A single dot means “the current directory”

One place it’s useful is when making a backup copy of a file, *e.g.*

```
$ cp ThalassocalyceData.txt ./original_data.txt
```

Move a File: `mv`

The mv command also has two arguments, and they mean the same thing as the arguments for cp

- the first is the name of a file to move
- the second is usually a directory and is the location where the file will be moved

⚠️ if there is already a file with the same name in that location…

```
$ mv documents/syllabus/syllabus.pdf ~
```

```
$ mv documents/syllabus/syllabus.pdf ~/Bi410.syl.pdf
```

Note the first argument can be a directory — it's possible to move a complete folder in a single command:

```
$ mv documents/syllabus downloads
```

Rename a File with `mv`

Unix does not have a rename command

Simply use `mv`, specifying the old name and the new name:

```
$ mv syllabus.pdf Bi410_syllabus.pdf
```

Delete a File: `rm`

To delete a file use the rm command

- specify one or more file names, separated by spaces

⚠ the files will be deleted immediately without warning

In this example I’m removing two temporary files made by my text formatting application:

```
$ rm syllabus.log syllabus.synctex.gz
```

Delete a Folder: rmdir

You will get an error message if you try to remove a folder with `rm`

Use `rmdir` instead:

```
$ rmdir sim
```

Note: the directory must be empty

Wildcards

Here are two more special characters to use in shell commands

- a question mark in a name means “any letter/digit can appear here"
- an asterisk in a name means “zero or more letters/digits can appear here”

Examples:

`$ ls scripts`	*list all files in* `scripts`
`$ ls scripts/dna*`	*list files that start with “dna”*
`$ ls scripts/dnacalc?.py`	*matches* `dnacalc1.py` *and* `dnacalc2.py`
`$ ls scripts/*.py`	*print the names of all the Python scripts*
`$ mv examples/FEC* .`	*moves all files that start with “FEC” to the current directory*

> *Advice: before you type a command that does anything drastic (e.g. moving or deleting a set of files) use the pattern in an* `ls` *command, then use command line editing to change* `ls` *to* `mv` *or* `rm`*…*

Options

Almost every shell command allows us to specify options

Examples:

- the `ls` and `date` commands have several different output formats
- we can tell `cp` and `mv` to warn us before they delete a file

Old Style Options

Options are single letters. To specify an option, type a dash followed by the letter (no space in between). Options can (usually) be entered in any order.

Example: print a directory listing with the `-a` (list all files) and `-l` (long form) options:

```
$ ls -a -l downloads/pcfb/
drwxr-xr-x  conery  staff   192 Jan  4 15:22 .
drwxr-xr-x  conery  staff   192 Jan  4 15:22 ..
drwxr-xr-x  conery  staff  1632 Jan  8 10:02 examples
drwxr-xr-x  conery  staff    64 Jan  4  2016 sandbox
drwxr-xr-x  conery  staff   992 Jan  4  2016 scripts
```

Single-letter options can be combined into a single “token”:

```
$ ls -al downloads/pcfb/
...
```

Options (cont’d)

Useful Options for `ls`

I like to use `-F` (print a slash after directory names).

Useful Options for `cp`, `rm`, and `mv`

The `-i` option tells the program to print a warning before deleting or overwriting a file.

New Style Options

A newer format for options uses complete words instead of single letters.

Specify these options with a double dash.

Example (from `usearch`, a bioinformatics application used by our pipeline):

```
$ usearch --global --strand plus --id 0.97 ...
```

Random Useful Commands

Some commands you might find useful…

`mkdir`

Use this command to create a new empty directory. The argument is the name of the directory.

```
$ mkdir foo
$ mkdir ~/Classes/399
```

`more`

This command is useful for printing a large file. It prints a “screenful” and then waits for you to hit the space bar to continue with the next page.

```
$ more examples/shaver_etal.csv
```

`man`

“man” is short for “manual”. Type “man x” to see a help page for a command named `x`. For example, to learn more about the `ls` command:

```
$ man ls
```

Configuration File

The lab session for Week 2 (Apr 13) will be devoted to setting up the configuration file on your system — don't miss it!

When the shell first starts it looks for a file named `.bashrc` in your home directory

- note the period at the front of the name!

This file can contain a series of shell commands, one per line. These commands are executed first, before the shell prints its first prompt.

Some settings from my configuration file that you might find useful are:

- include the machine name and current directory name in the prompt
- change the prompt color after logging in to a remote machine
- tell `ls` to print / after directory names (use `-F`)
- tell `rm`, `cp`, and `mv` to use `-i` (warn when deleting or overwriting a file)
- define a few other useful commands

If you want to add these settings to your own configuration:

- download a file named `bashrc` (without the period) from Canvas
- move it to your home directory
- type this command to append my settings to your current configuration:

```
$ cat bashrc >> .bashrc
```

Be careful when you type that command! Note how "bashrc" has a period in one place and not the other!