STAT 605
Data Science Computing

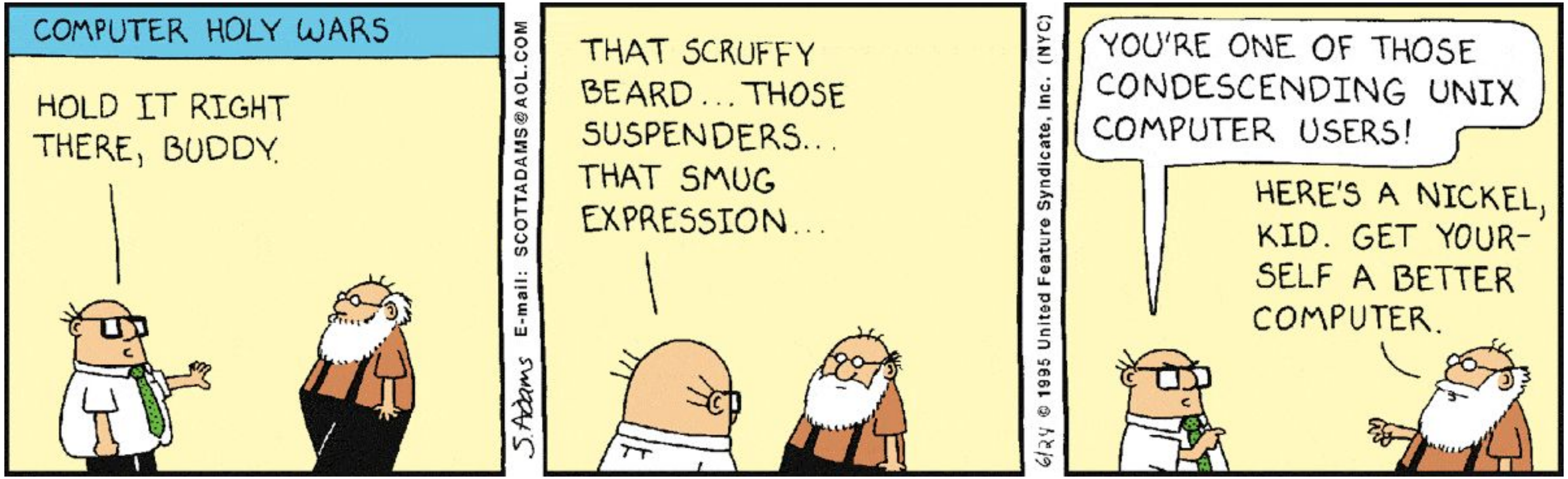
Introduction to the UNIX/Linux command line

Why UNIX/Linux?

As a data scientist, you will spend **most** of your time dealing with data

- Data sets never arrive "ready to analyze"
- Cleaning data, fixing formatting, etc is 80% of the process

These "data wrangling" tasks are (often) best done on the command line


COMPUTER HOLY WARS
HOLD IT RIGHT THERE, BUDDY.
E-mail: SCOTTADAMS@AOL.COM
S. Adams
THAT SCRUFFY BEARD... THOSE SUSPENDERS... THAT SMUG EXPRESSION...
6/24 © 1995 United Feature Syndicate, Inc. (NYC)
YOU'RE ONE OF THOSE CONDESCENDING UNIX COMPUTER USERS!
HERE'S A NICKEL, KID. GET YOURSELF A BETTER COMPUTER.

UNIX/Linux: a (very) brief history

1960s: Multics (Bell Labs, MIT, GE), a time-sharing operating system

1970s: UNIX developed at Bell Labs

1980s: the UNIX wars https://en.wikipedia.org/wiki/Unix_wars

1990s: GNU/Linux emerges

2000s: MacOS developed based on UNIX

Bell labs film about UNIX from 1982:
http://techchannel.att.com/play-video.cfm/2012/2/22/AT&T-Archives-The-UNIX-System

The Unix philosophy: do one thing well

1. Write programs that do one thing and do it well.
2. Write programs to work together.
3. Write programs to handle text streams, because that is a universal interface.

The Unix philosophy: do one thing well

1. Write programs that do one thing and do it well.

2. Write programs to work together.

3. Write programs to handle text streams, because that is a universal interface.

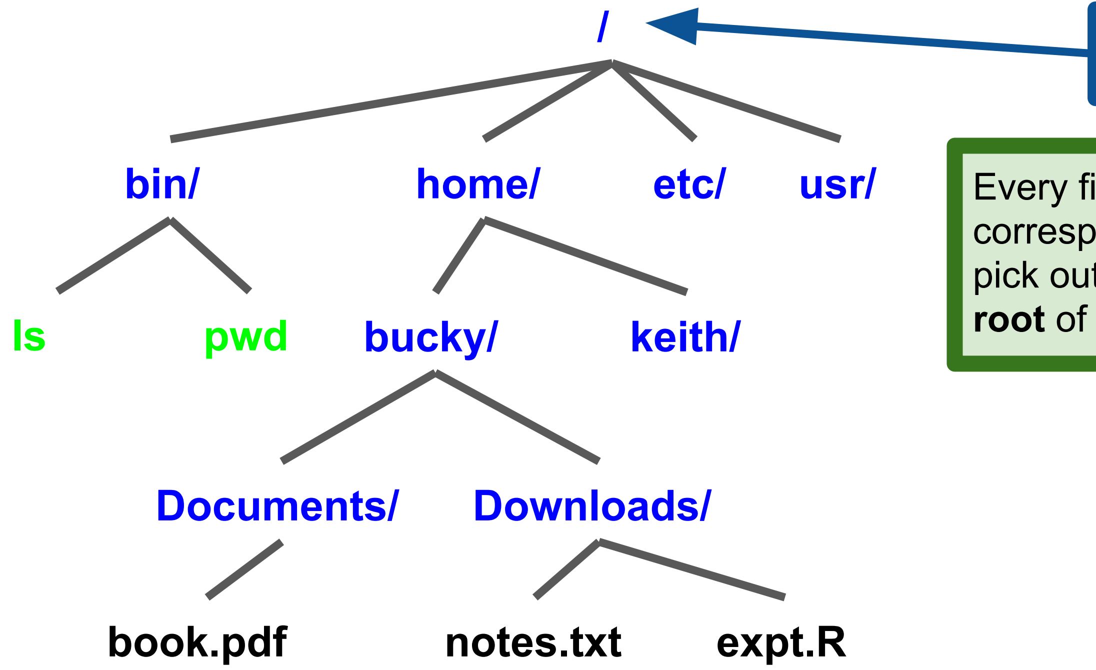
These three design principles, articulated in the concise form above long after Unix was written, go a long way toward explaining how to approach the command line. For nearly any task you wish to accomplish, there almost certainly exists a way to do it (reasonably) easily by stringing together several different programs. **More information:** https://en.wikipedia.org/wiki/Unix_philosophy

Exercises: Part 1

1) In your VM, open a terminal.

Interacting with the File System

Files on your computer are organized in a tree structure

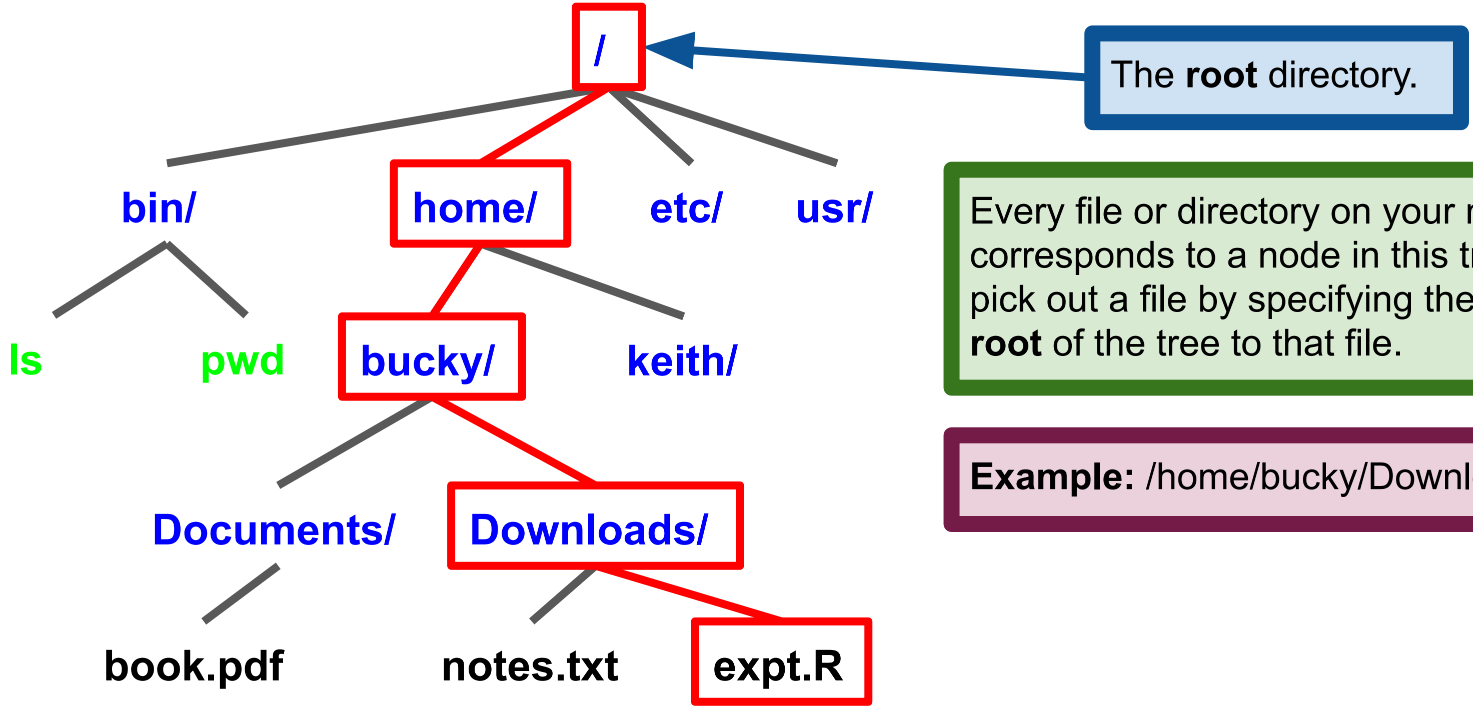


The **root** directory.

Every file or directory on your machine corresponds to a node in this tree. We can pick out a file by specifying the path from the **root** of the tree to that file.

Interacting with the File System

Files on your computer are organized in a tree structure



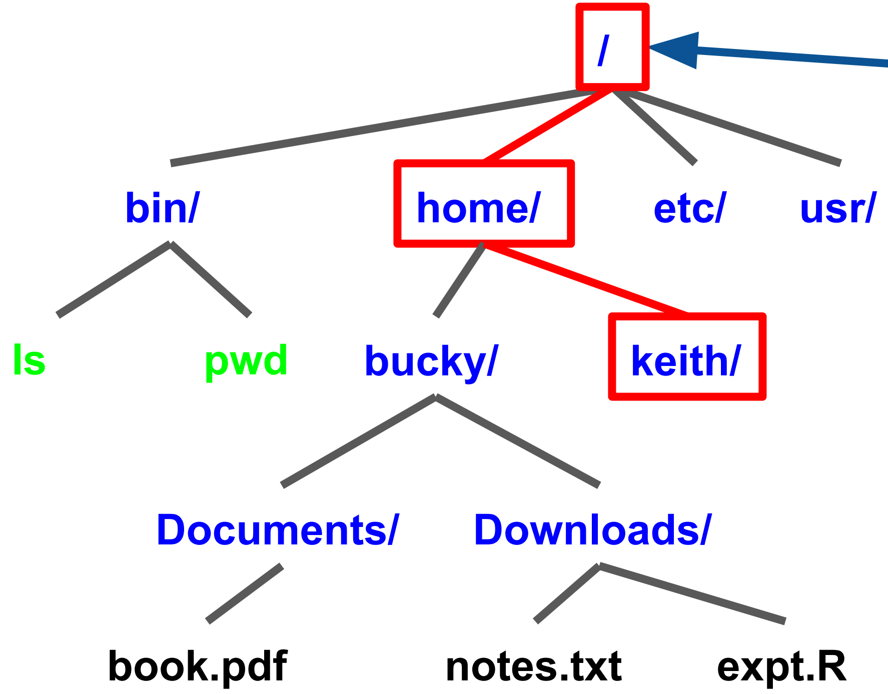
The **root** directory.

Every file or directory on your machine corresponds to a node in this tree. We can pick out a file by specifying the path from the **root** of the tree to that file.

Example: /home/bucky/Downloads/expt.R

Interacting with the File System

Files on your computer are organized in a tree structure



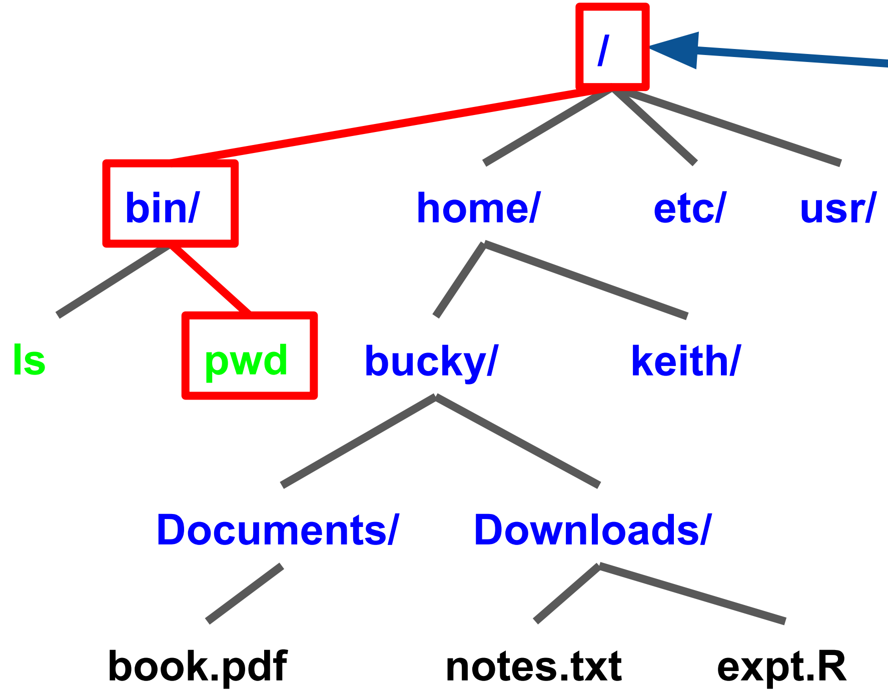
The **root** directory.

Every file or directory on your machine corresponds to a node in this tree. We can pick out a file by specifying the path from the **root** of the tree to that file.

Example: /home/keith/

Interacting with the File System

Files on your computer are organized in a tree structure



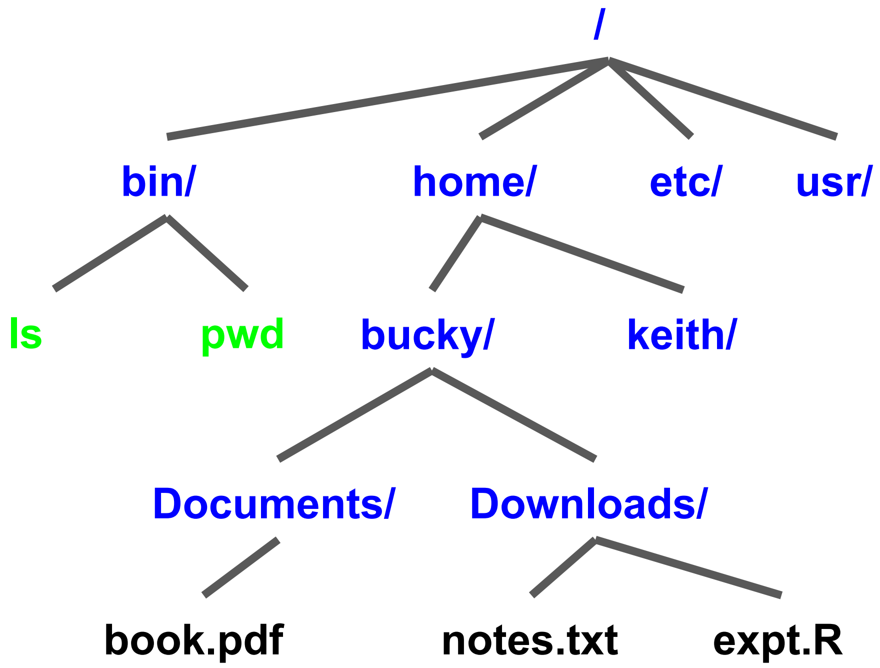
The **root** directory.

Every file or directory on your machine corresponds to a node in this tree. We can pick out a file by specifying the path from the **root** of the tree to that file.

Example: /home/bin/pwd

Interacting with the File System

Files on your computer are organized in a tree structure

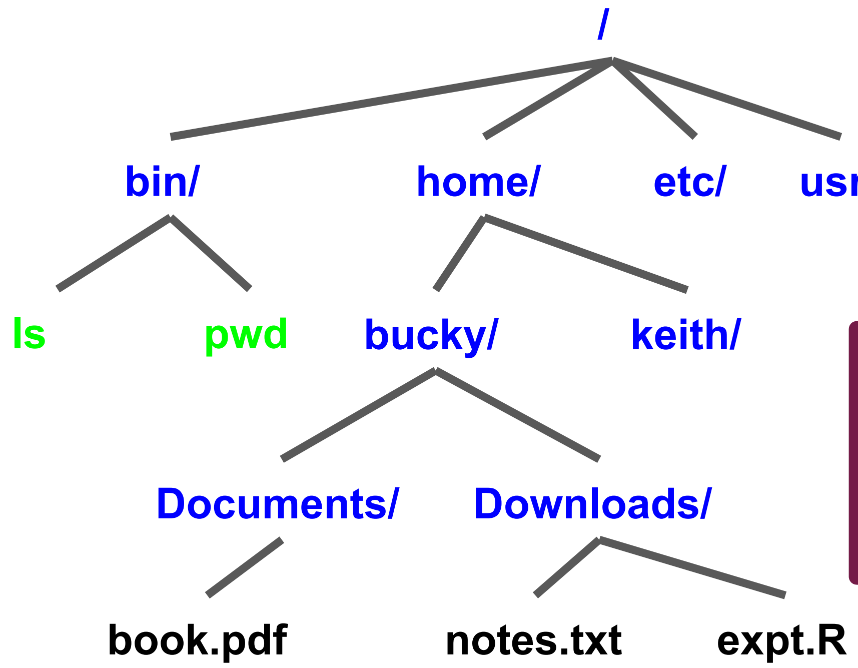


At any time, the shell has a **working directory**, which is a directory (i.e., a folder) somewhere in the file system tree.

We can find out the working directory using the command `pwd` (print working directory) and change it using `cd` (change directory).

Interacting with the File System

Files on your computer are organized in a tree structure

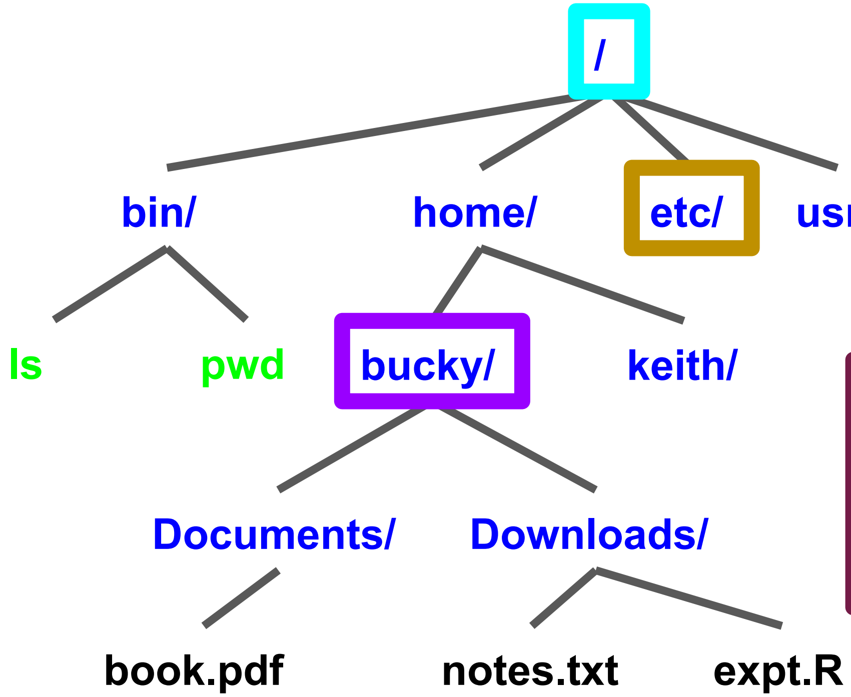


At any time, the shell has a **working directory**, which is a directory (i.e., a folder) somewhere in the file system tree.

Some special directory symbols:

- ~ : your home directory, e.g., `/home/bucky`
- . : the current directory
- . . : the directory above the current directory

Interacting with the File System



So, if I am logged in as `bucky`, and the current directory is `/etc/`, then

~ refers to `/home/bucky/`

. refers to `/etc/`

and . . refers to `/`

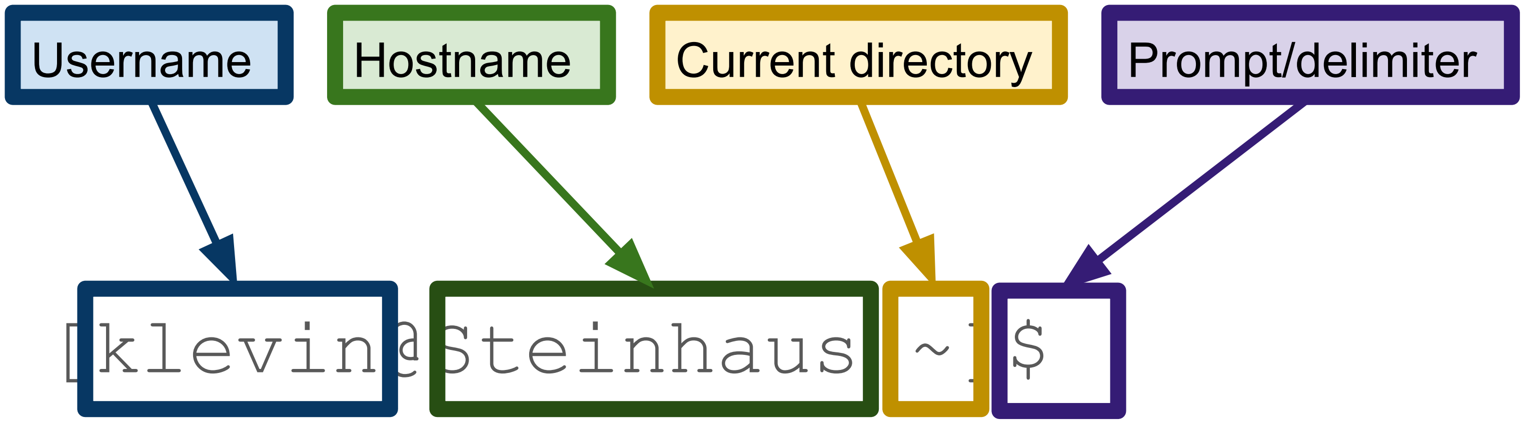
Some special directory symbols:

~ : your home directory, e.g., `/home/bucky`

. : the current directory

. . : the directory above the current directory

Parts of the command line prompt



```
[klevin@Steinhaus ~]$
```

Note: details of this will vary from one computer to the next (and it can be customized by the user), but this is the default on many clusters. For information on customizing the command line prompt, see https://linuxconfig.org/bash-prompt-basics

Basic commands for navigating

`pwd` : "print/present working directory". Print the directory that you are currently in.

`ls` : list the contents of the current directory.

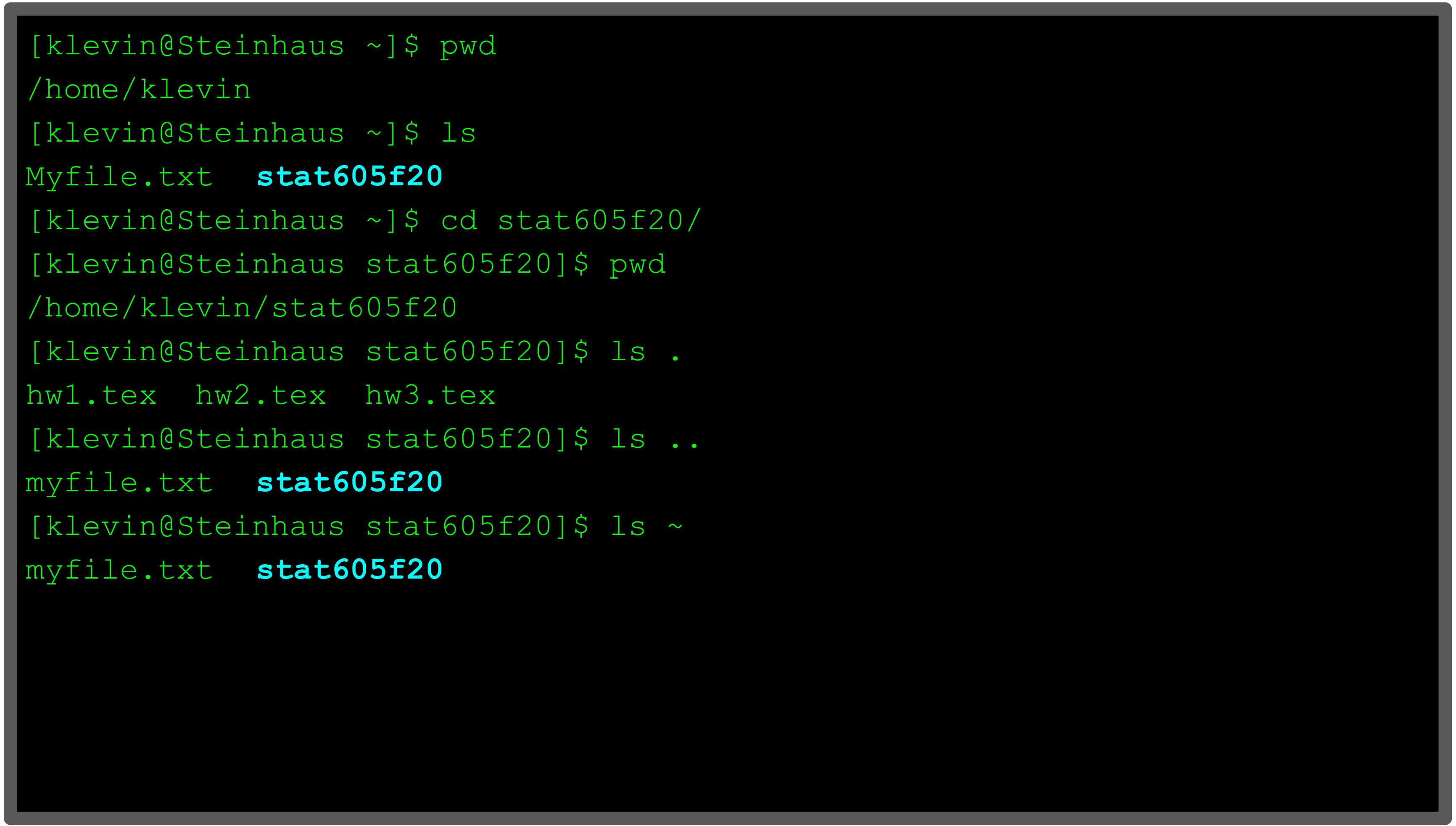
Try this. Type `pwd` or `ls` in your shell (either on your VM or on your local machine).

`cd dirname` : change the working directory to `dirname`.

Some special directory symbols:

- ~ : your home directory. `cd ~` will take you back to your home.
- . : the current directory. `cd .` will take you to where you are right now.
- `..` : the directory above the current directory.
  - If you're in `/home/klevin/stats`, then `cd ..` will take you to `/home/klevin`.

Example: `pwd`, `ls` and `cd`



```
[klevin@Steinhaus ~]$ pwd
/home/klevin
[klevin@Steinhaus ~]$ ls
Myfile.txt  stat605f20
[klevin@Steinhaus ~]$ cd stat605f20/
[klevin@Steinhaus stat605f20]$ pwd
/home/klevin/stat605f20
[klevin@Steinhaus stat605f20]$ ls .
hw1.tex  hw2.tex  hw3.tex
[klevin@Steinhaus stat605f20]$ ls ..
myfile.txt  stat605f20
[klevin@Steinhaus stat605f20]$ ls ~
myfile.txt  stat605f20
```

Exercises: Part 2

1) Examine the prompt in your terminal. Does it match the one from the lecture?
2) In the terminal, use `cd`, `pwd` and `ls` to explore the file system a little bit

Getting help: man pages

When in doubt, the shell has built-in documentation, and it tends to be good!

`man cmdname` : brings up documentation about the command `cmdname`

This help page is called a man (short for manual) page. These have a reputation for being terse, but once you get used to reading them, they are extremely useful!

Some shells also have a command `apropos:`

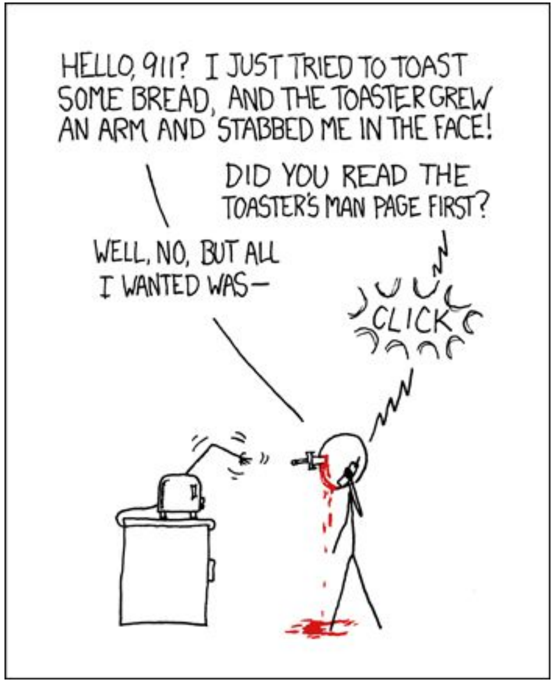
`apropos topic` : lists all commands that might be relevant to `topic.`

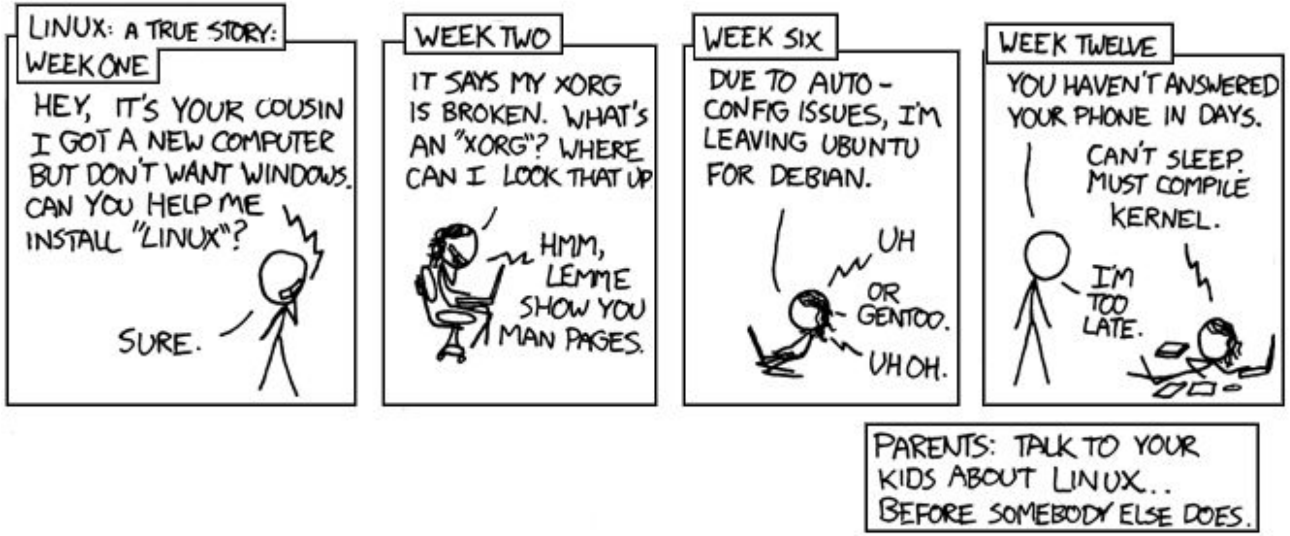
Let's read some of the `ls` man page and see if we can make sense of it.

Exercises: Part 3

1) Read (some of) the man page for `ls`. Don’t worry if you don’t understand everything; just read enough to get a feel for the style of writing.
2) Choose a topic, and try using `apropos` to find a relevant command. Read the `man` page for that command (again, don’t worry if you don’t understand everything).

Relevant xkcds


HELLO, 911? I JUST TRIED TO TOAST SOME BREAD, AND THE TOASTER GREW AN ARM AND STABBED ME IN THE FACE!
DID YOU READ THE TOASTER'S MAN PAGE FIRST?
WELL, NO, BUT ALL I WANTED WAS—
CLICK


LINUX: A TRUE STORY: WEEK ONE
HEY, IT'S YOUR COUSIN I GOT A NEW COMPUTER BUT DON'T WANT WINDOWS. CAN YOU HELP ME INSTALL "LINUX"?
SURE.
WEEK TWO
IT SAYS MY XORG IS BROKEN. WHAT'S AN "XORG"? WHERE CAN I LOOK THAT UP
HMM, LEMME SHOW YOU MAN PAGES.
WEEK SIX
DUE TO AUTO-CONFIG ISSUES, I'M LEAVING UBUNTU FOR DEBIAN.
UH
OR GENTOO.
UH OH.
WEEK TWELVE
YOU HAVEN'T ANSWERED YOUR PHONE IN DAYS.
CAN'T SLEEP. MUST COMPILE KERNEL.
I'M TOO LATE.
PARENTS: TALK TO YOUR KIDS ABOUT LINUX... BEFORE SOMEBODY ELSE DOES.

Special file handles: `stdin`, `stdout`, `stderr`

File handles are pointers to files

Familiar if you've programmed in C/C++

Similar: object returned by python `open()`

By default, most command line programs

- take input from `stdin`
- Write output to `stdout`
- Write errors and status information to `stderr`

Basic commands: actually doing things

In the next few slides, we’ll look at some commands that actually let you do things like creating files and directories, reading files, and moving them around.

Follow along with the examples in your terminal, if you like (highly recommended).

Basic commands: `echo`

`echo string` : prints string to the shell.

```
keith@Steinhaus:~$ echo "hello world."
hello world.
keith@Steinhaus:~$ echo "hello world!"
-bash: !": event not found
keith@Steinhaus:~$ echo "hello world\!"
hello world\!
keith@Steinhaus:~$ echo 'hello world!'
hello world!
keith@Steinhaus:~$ echo "hello\tworld."
hello\tworld.
keith@Steinhaus:~$ echo -e "hello\tworld."
hello   world.
```

The shell tries to interpret the exclamation point as referencing a previous command rather than as text. Escaping doesn't do the trick here. Instead, use single-quotes to tell the shell not to try and process the string. Note that this error will occur in MacOSX but not Ubuntu.

To print special characters (tabs, newlines, etc), use the flag `-e`, without which `echo` just prints what it's given.

Note: different shells will have slightly different behavior here, due to differences in parsers.

Aside: redirections using >

What if I want to send output someplace other than the shell?

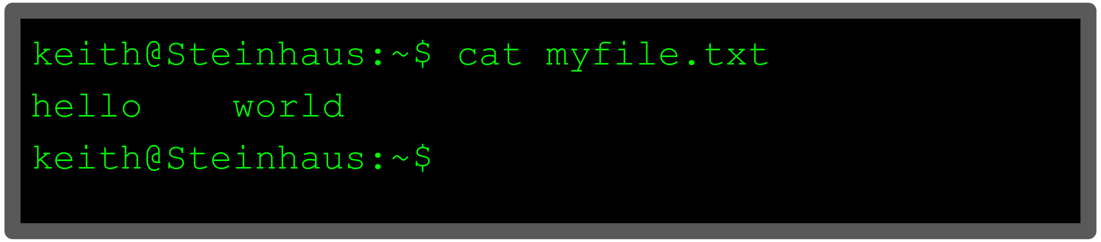
```
keith@Steinhaus:~$ echo -e  "hello\tworld." > myfile.txt
keith@Steinhaus:~$
```

Note: the other redirect, <, has a somewhat similar function, but is beyond our purposes here (stay tuned for command-line workshop at end of semester, perhaps?)

Redirect tells the shell to send the output of the program on the “greater than” side to the file on the “lesser than” side. **This creates the file on the RHS, an overwrites the old file, if it already exists!**

Basic commands: `cat`

`cat filename` : prints the contents of the file filename.



```
keith@Steinhaus:~$ cat myfile.txt
hello    world
keith@Steinhaus:~$
```

So `cat` is like `echo` but it takes a filename as argument instead of a string.

Basic commands: `head`

`head filename` : prints the first 10 lines of filename.

`head -n X filename` : prints the first X lines of filename.

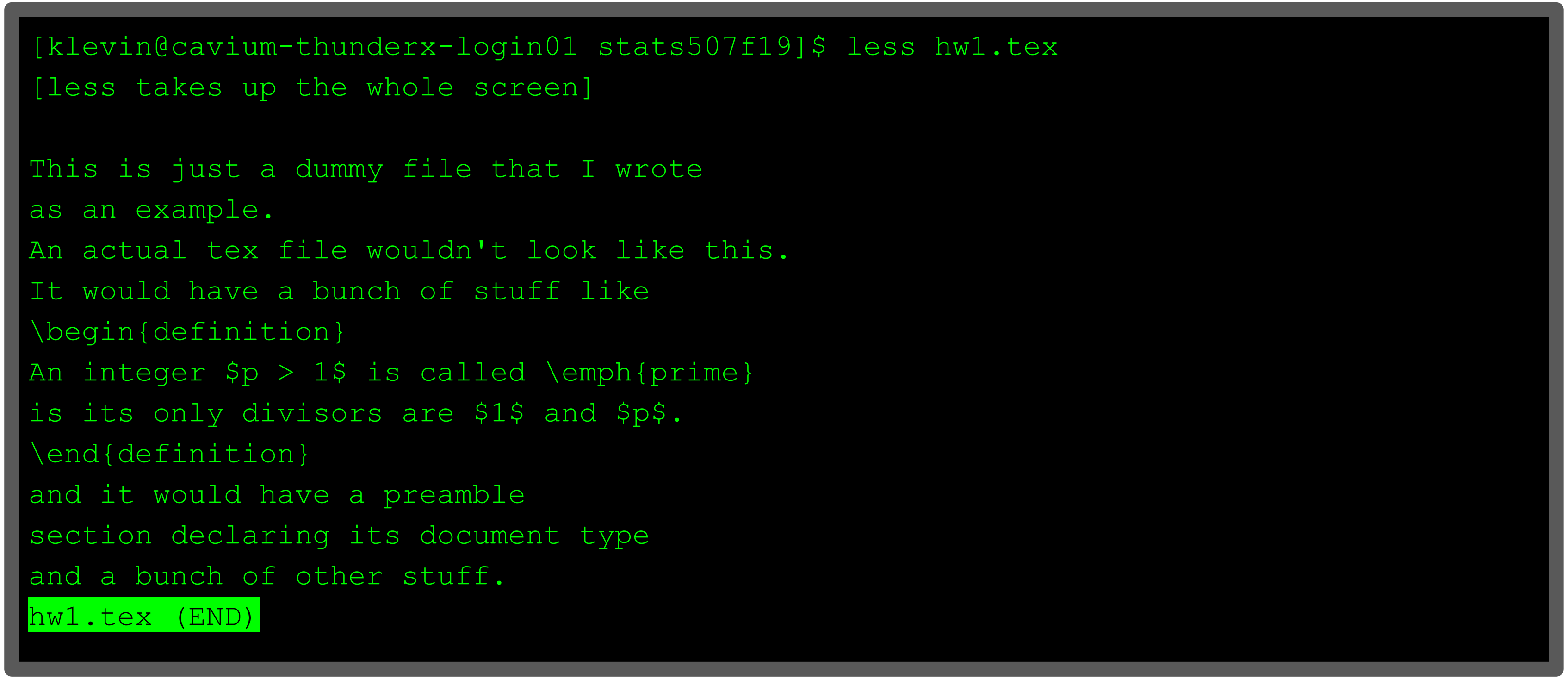
```
keith@Steinhaus:~$ head ~/Teaching/Homeworks/HW1/homework1.tex
\documentclass[11pt]{article}

\usepackage{enumerate}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{hyperref}

\oddsidemargin 0mm
\evensidemargin 5mm
\topmargin -20mm
keith@Steinhaus:~$
```

Basic commands: `more`/`less`

`more` and `less` are two (very similar) programs for reading ASCII files.



```
[klevin@cavium-thunderx-login01 stats507f19]$ less hw1.tex
[less takes up the whole screen]

This is just a dummy file that I wrote
as an example.
An actual tex file wouldn't look like this.
It would have a bunch of stuff like
\begin{definition}
An integer $p > 1$ is called \emph{prime}
is its only divisors are $1$ and $p$.
\end{definition}
and it would have a preamble
section declaring its document type
and a bunch of other stuff.
hw1.tex (END)
```

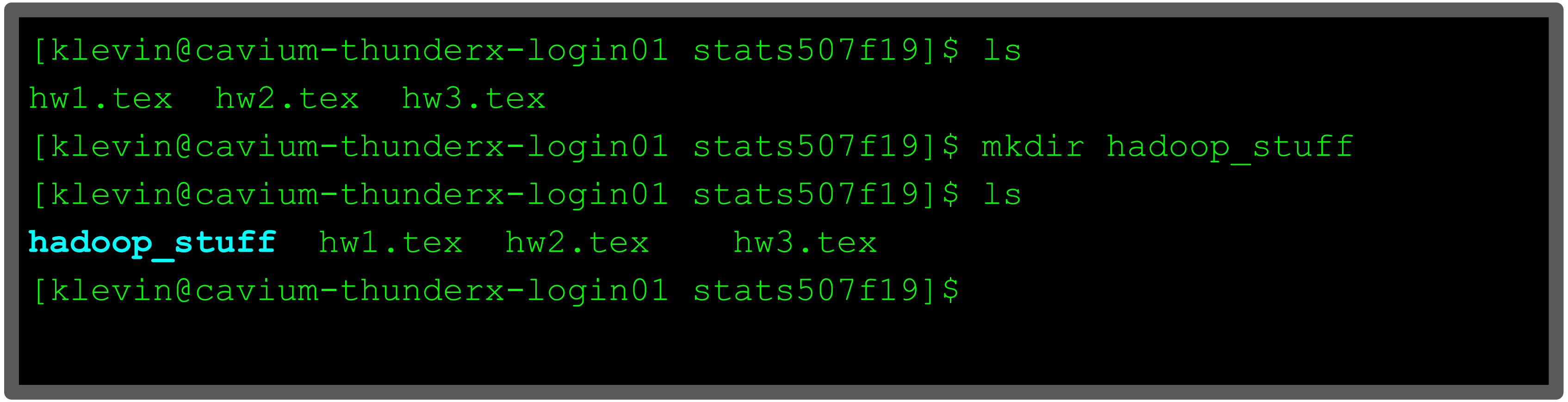
Note: press “q” to **quit** `less`/`more` and return to the command line.

Exercises: Part 4

1) Use `echo` and a redirect to create a file called `quick.txt`, containing the text “`The quick brown fox jumped over the lazy dog.`”
2) Use `cat` to print the contents of `quick.txt` to the terminal.
3) Download jabberwocky.txt from the course webpage. Combine head and tail in a clever way to print the first stanza of the poem (the first stanza spans the fourth through seventh lines of the file)
4) Use less to page through `jabberwocky.txt`. Practice scrolling with the arrow keys, `j` and `k`, and paging (`spacebar` and `b`)

Basic commands: `mkdir`

`mkdir dirname` : creates a new directory called dirname, if it doesn’t exist

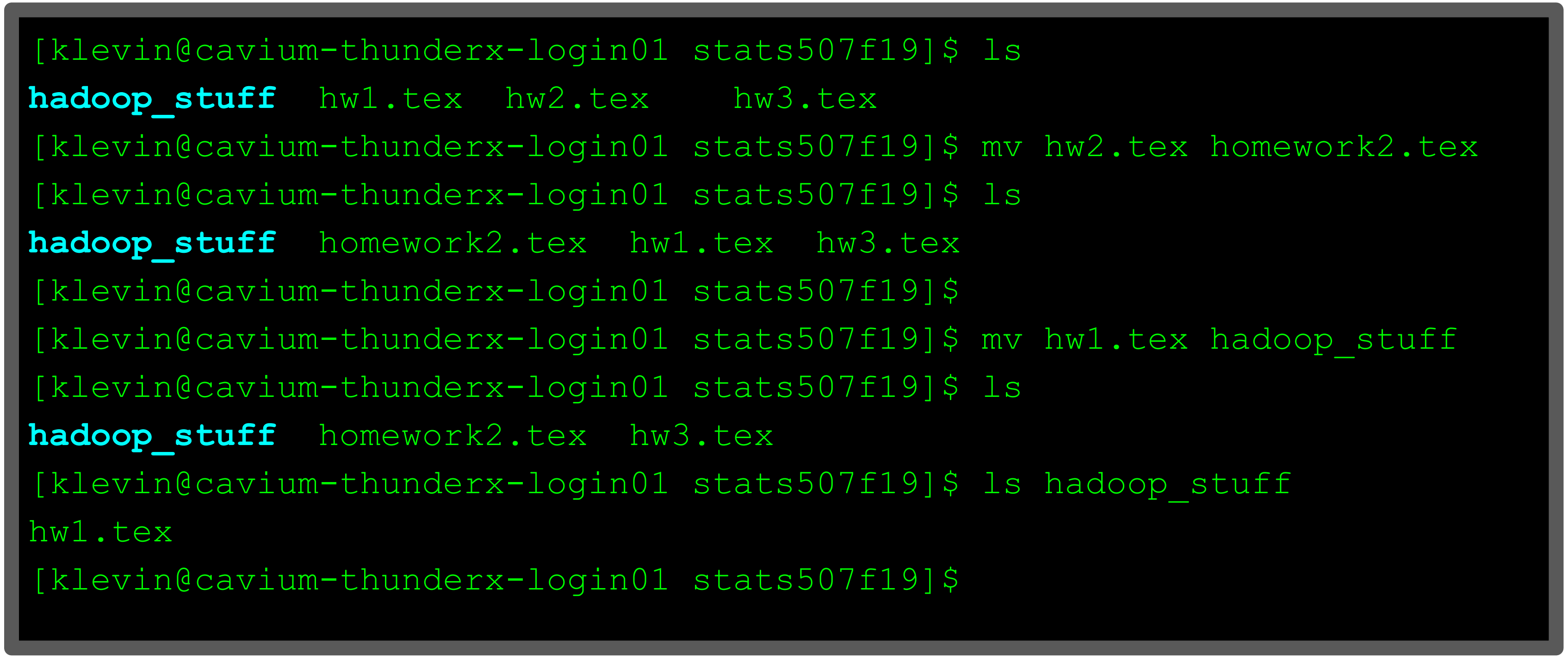


```
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hw1.tex  hw2.tex  hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$ mkdir hadoop_stuff
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  hw1.tex  hw2.tex    hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$
```

Basic commands: `mv`

`mv file1 file2` : “moves” `file1` to `file2`, overwriting `file2`.

If `file2` is a directory, this places `file1` inside that directory, again replacing any existing file with the same **basename** as `file1`. /path/to/file/basename.txt



```
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  hw1.tex  hw2.tex    hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$ mv hw2.tex homework2.tex
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  homework2.tex  hw1.tex  hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$
[klevin@cavium-thunderx-login01 stats507f19]$ mv hw1.tex hadoop_stuff
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  homework2.tex  hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$ ls hadoop_stuff
hw1.tex
[klevin@cavium-thunderx-login01 stats507f19]$
```

Basic commands: `cp`

`cp file1 file2` : similar to `mv`, but creates a copy of `file1` with name `file2`

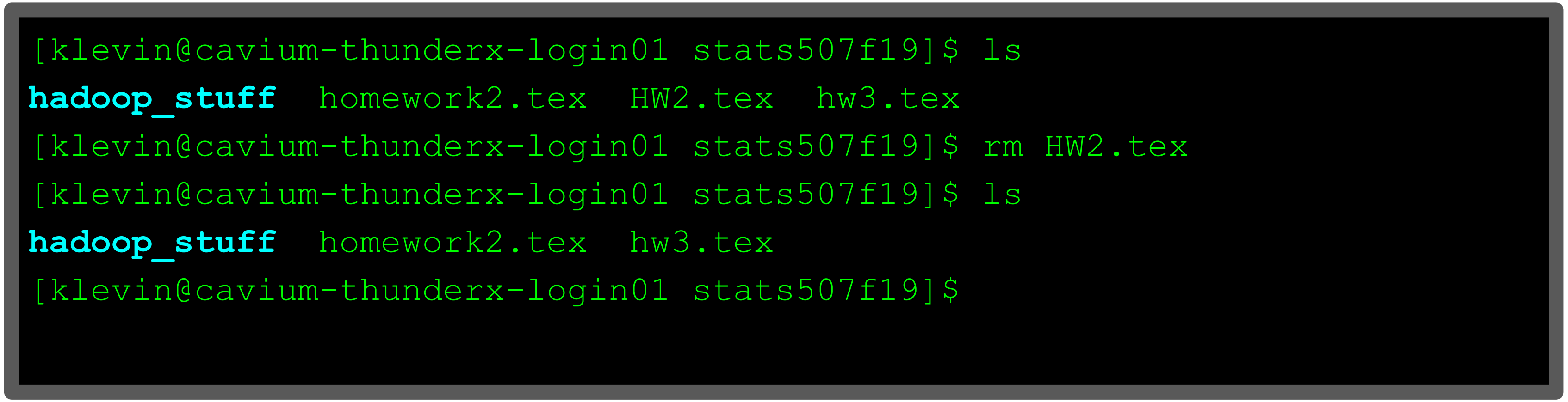
So `cp` is like `mv` but `file1` is copied instead of being renamed

```
[klevin@cavium-thunderx-login01 stats507f19]$ cat homework2.tex
This is the second homework!
[klevin@cavium-thunderx-login01 stats507f19]$ cp homework2.tex HW2.tex
[klevin@cavium-thunderx-login01 stats507f19]$ cat homework2.tex
This is the second homework!
[klevin@cavium-thunderx-login01 stats507f19]$ cat HW2.tex
This is the second homework!
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  homework2.tex  HW2.tex  hw3.tex
```

Note: to copy a directory, you must include the -r flag to cp: `cp -r dirname otherdirname`

Basic commands: `rm`

`rm filename` : deletes the file `filename`. **Be very very careful with this!**



```
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  homework2.tex  HW2.tex  hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$ rm HW2.tex
[klevin@cavium-thunderx-login01 stats507f19]$ ls
hadoop_stuff  homework2.tex  hw3.tex
[klevin@cavium-thunderx-login01 stats507f19]$
```

Exercises: Part 5

1) Use echo and a redirect to create a file called `my_file.txt` containing the string `This is my file`.
2) Create a directory called “`directory_exercise`” and `cd` into it
3) Use `mv` to move `my_file.txt` into the directory (hint: use `..`)
4) `cd` back up a directory.
5) Use `cp` to create a copy of `directory_exercise` called `copy_of_dir`
6) Use `cat` to check that `my_file.txt` is the same in both directories
7) Use `rm` to delete both `directory_exercise` and `copy_of_dir`