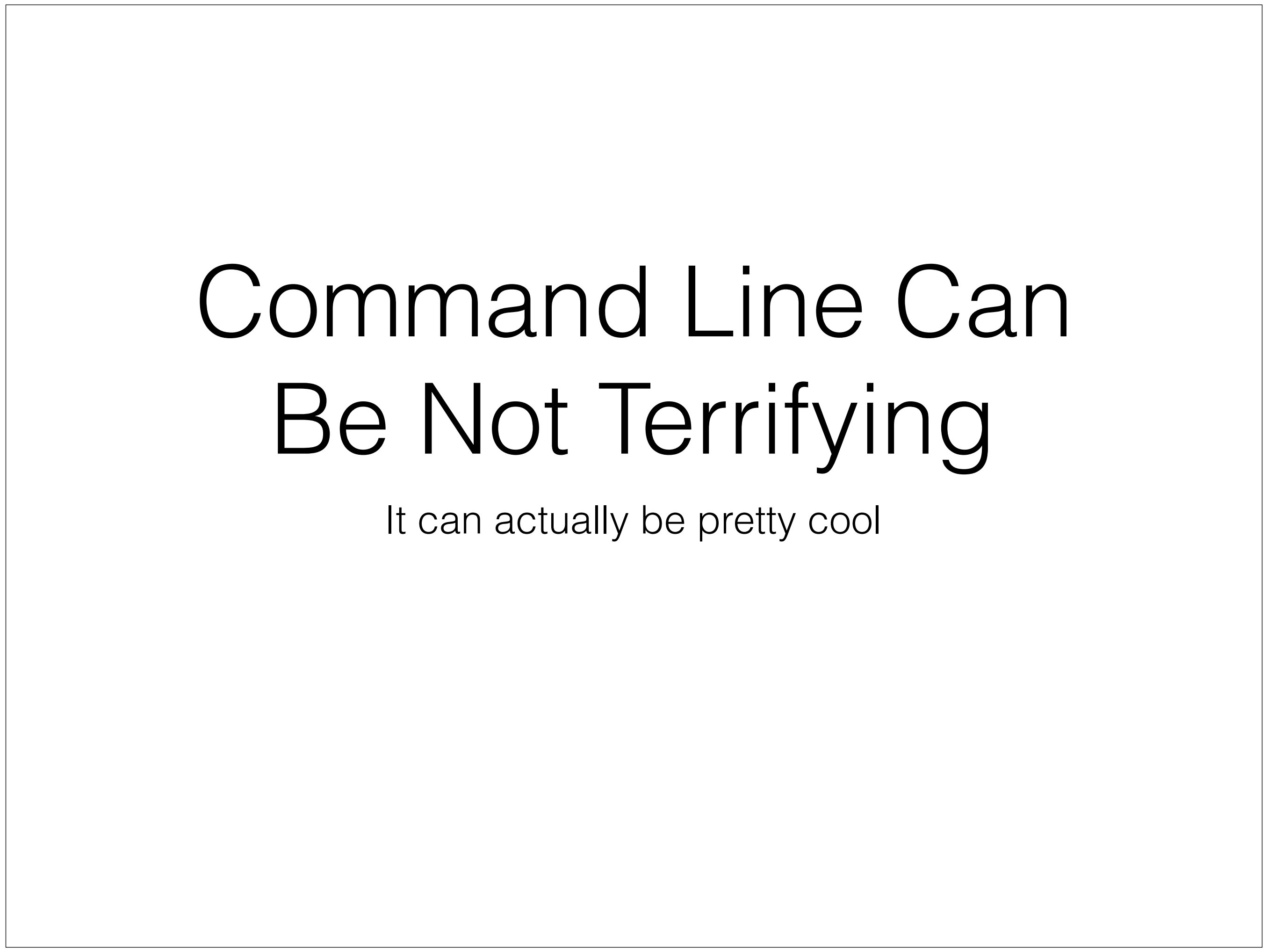


Command Line Can Be Not Terrifying

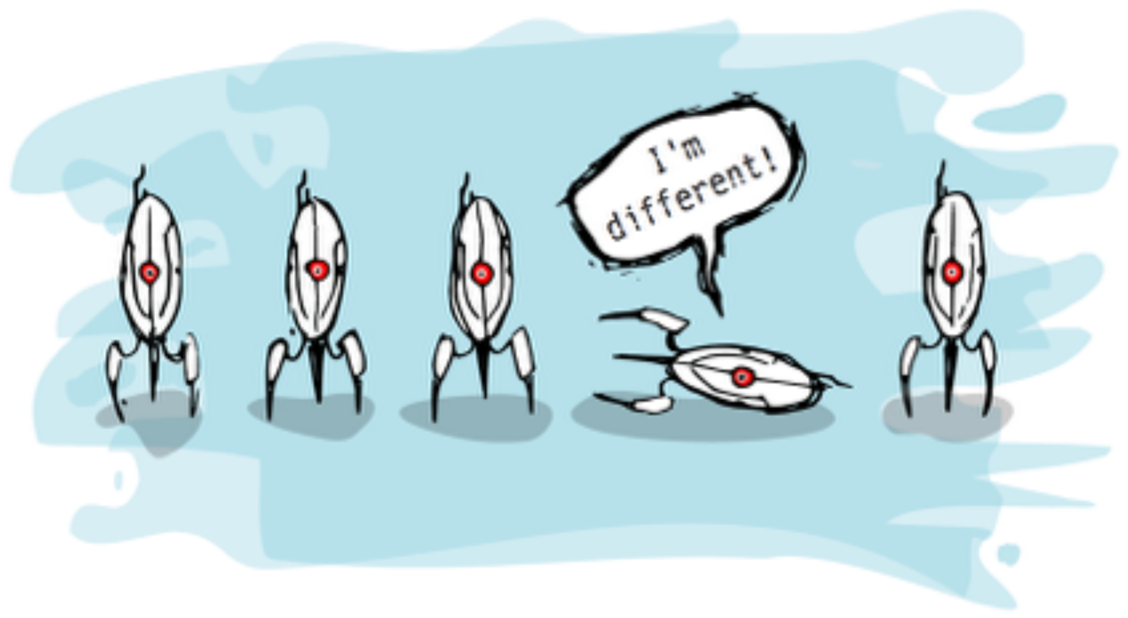
It can actually be pretty cool

so I’m gonna give you an intro to command line

I want to talk about source control

why? b/c I want to talk about compilers and dependencies and build scripts and package managers

and that’s hard w/out command line



(via Laggy Creations)

but this is not like most command line tutorials

I wanna talk about “what can you do” rather than “how you can do it”

b/c

a) it’s more interesting
b) it’s really easy to find the answers on the internet
c) and I’ll probably be wrong for a lot of things.

“Stack Overflow copy files UNIX”

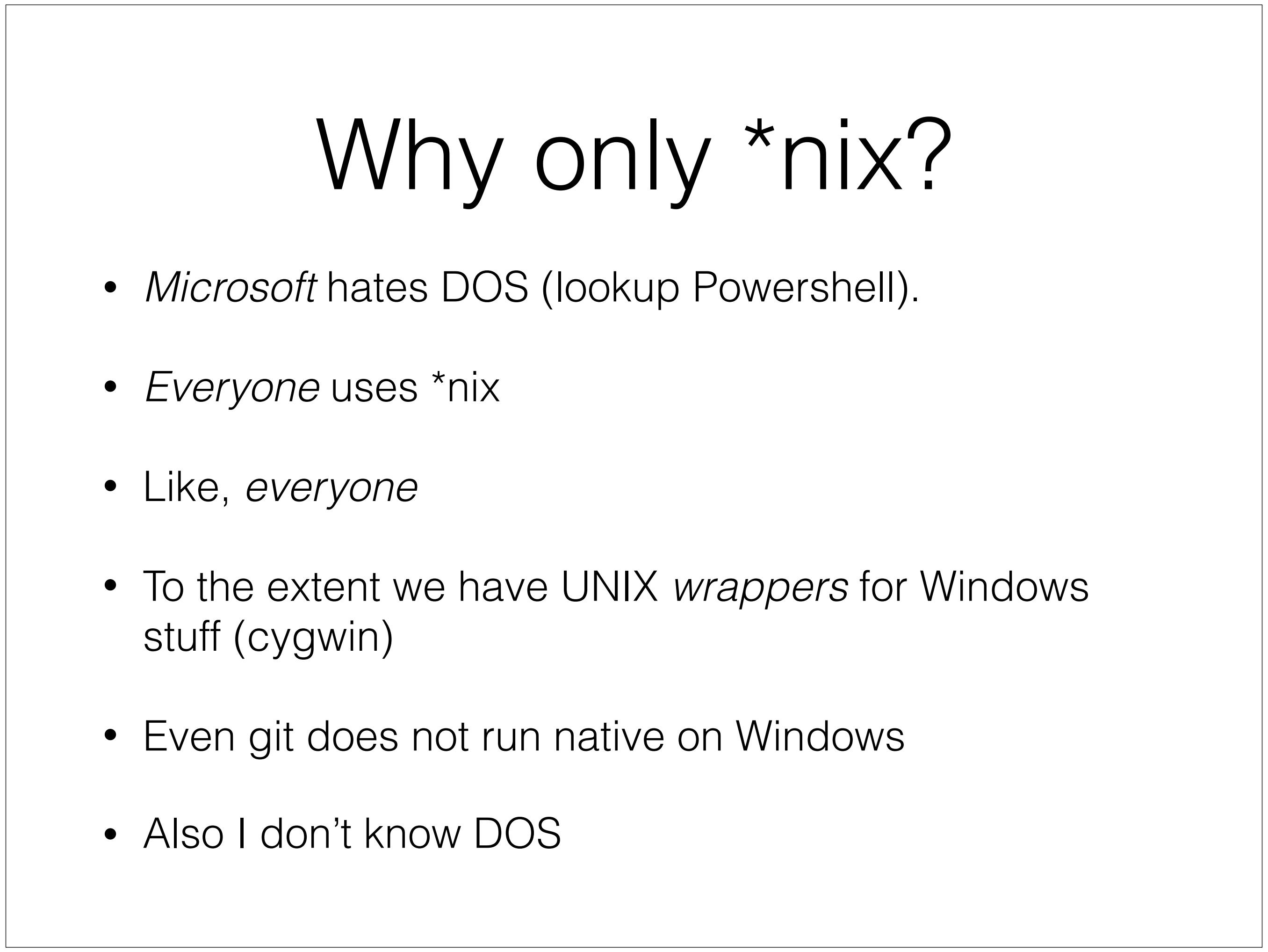
honestly, just google this if you wanna learn how to copy files “stakc coyp fiels linux”

this will probably work, too.

Unix/Linux/*nix/un*x

we’re gonna talk about linux
or unix
or asterisk-nix
or un-asterisk-x

Unix is a trademark
Linux is a Unix clone
They’re very similar



Why only *nix?

- *Microsoft* hates DOS (lookup Powershell).
- *Everyone* uses *nix
- Like, *everyone*
- To the extent we have UNIX *wrappers* for Windows stuff (cygwin)
- Even git does not run native on Windows
- Also I don't know DOS

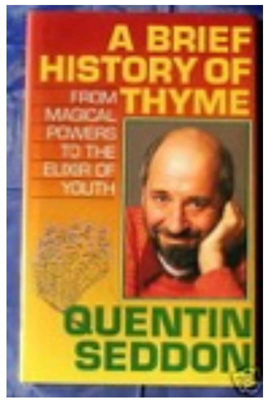
we will only talk about UNIX in this class

for some of these reasons

a lot of the concepts carry over. Some don't, and that's because UNIX is better.

I've been asked why I use a mac.
=> UNIX

All jokes aside, I would teach DOS if it were useful, or even go over it briefly. But it's really not. Shell scripts are rarely used on Windows, and all but the weirdest of servers run Linux. It's another language to learn, and it's worth it if you, for some reason, need to use it. But you generally won't unless you're an IT person.



(via ebay.com)

let’s start with some history

(via wikipedia)

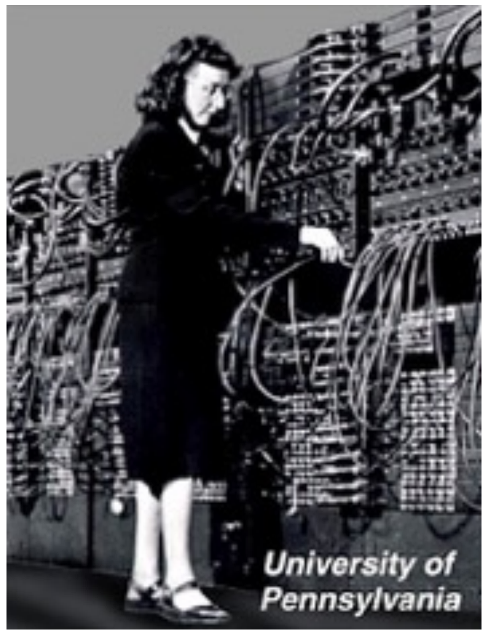
we have morse code

we also have typewriters

“damn,” says Emile Baudot and a bunch of other cool people, “what if we could build a machine that combined the two”

one solution: literally 128 wires, actuate the correct letter on the other end
another: one drum, like a rotary telephone, move it the requisite dist each time
another: use some sort of … “encoding” … to translate 5 wires into 32 different characters (gasp proto-ASCII what)

Teletype (TTY)



(via plyojump.com)

After we left the “Grace Hopper is god”-era of computing with ENIAC in the 40s

(via plyojump.com)

we started hooking up those teletypes (TTYs) to mainframes.

Terminal

we called these typewriters “terminals”

(via wikipedia)

then we started putting the computer in the same package as the old “terminal”— no more mainframes

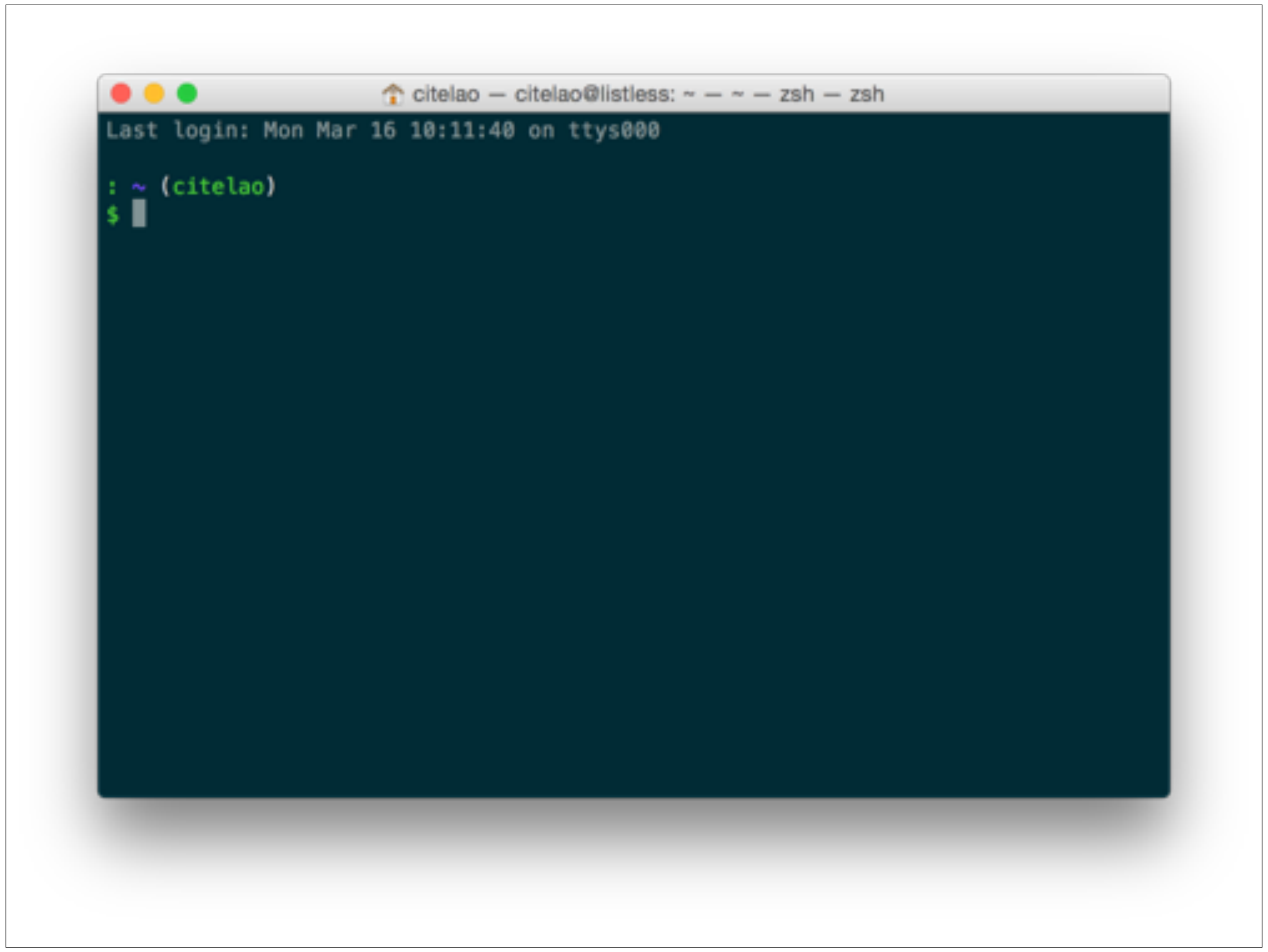
it’s not really a terminal connected to a larger device (in the strict English defn sense) so a…

Terminal Emulator

a “terminal emulator”.

Shell

We also have some intelligence in the text-entry bit: you know, you can *backspace* and *view history* and get tab-suggestions because we wrapped the interface in a “shell.”



```
Last login: Mon Mar 16 10:11:40 on ttys000

: ~ (citelao)
$ 
```

and now we have “terminal windows” within GUIs

Terminal Window

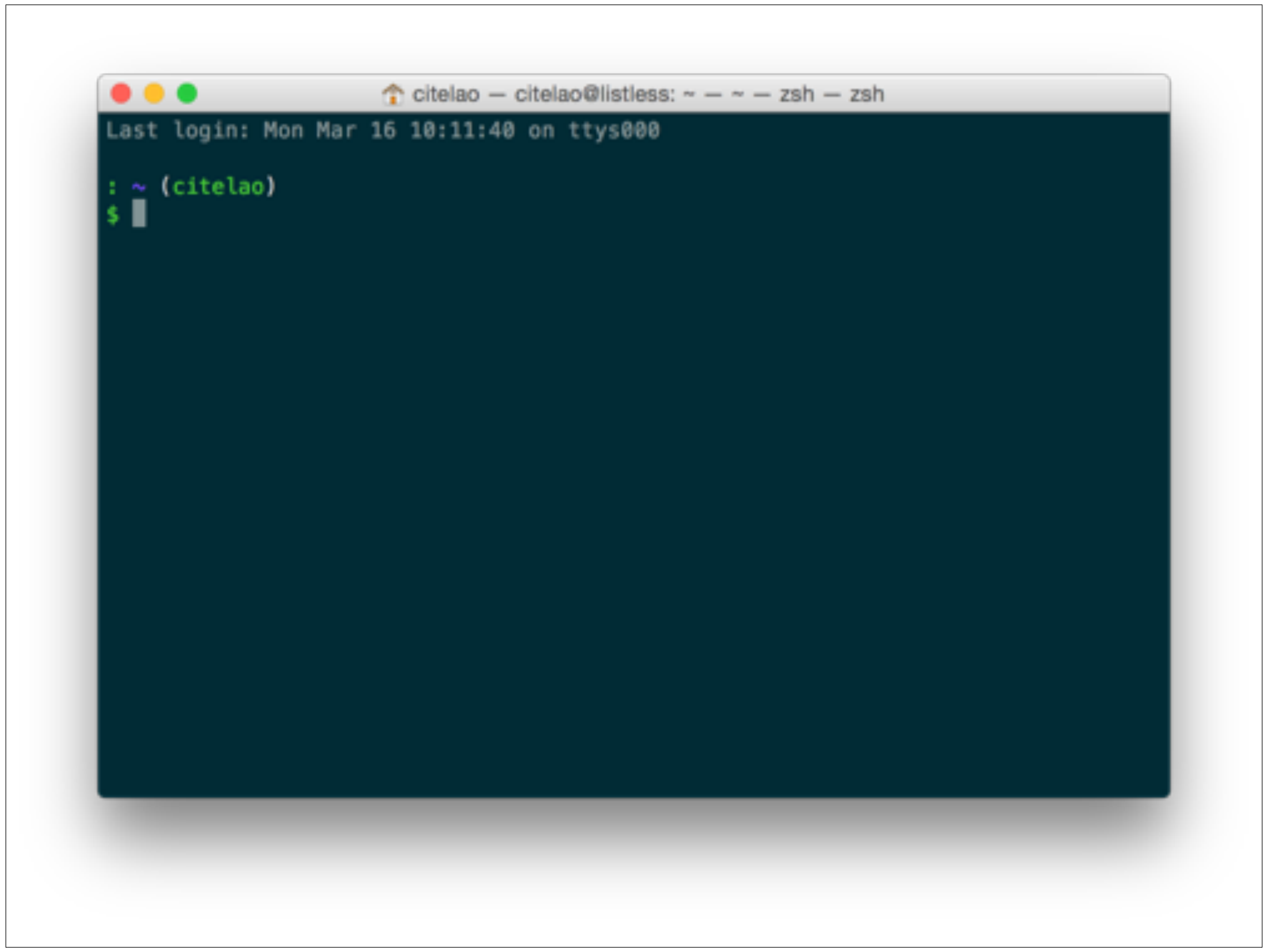
congratulations you pedantic jerk

Terminal vs TTY vs Terminal Emulator vs Terminal Window vs Shell

you now know the difference b/w these things

you can go around being a pedantic UNIX nerd. aren’t you proud of yourself

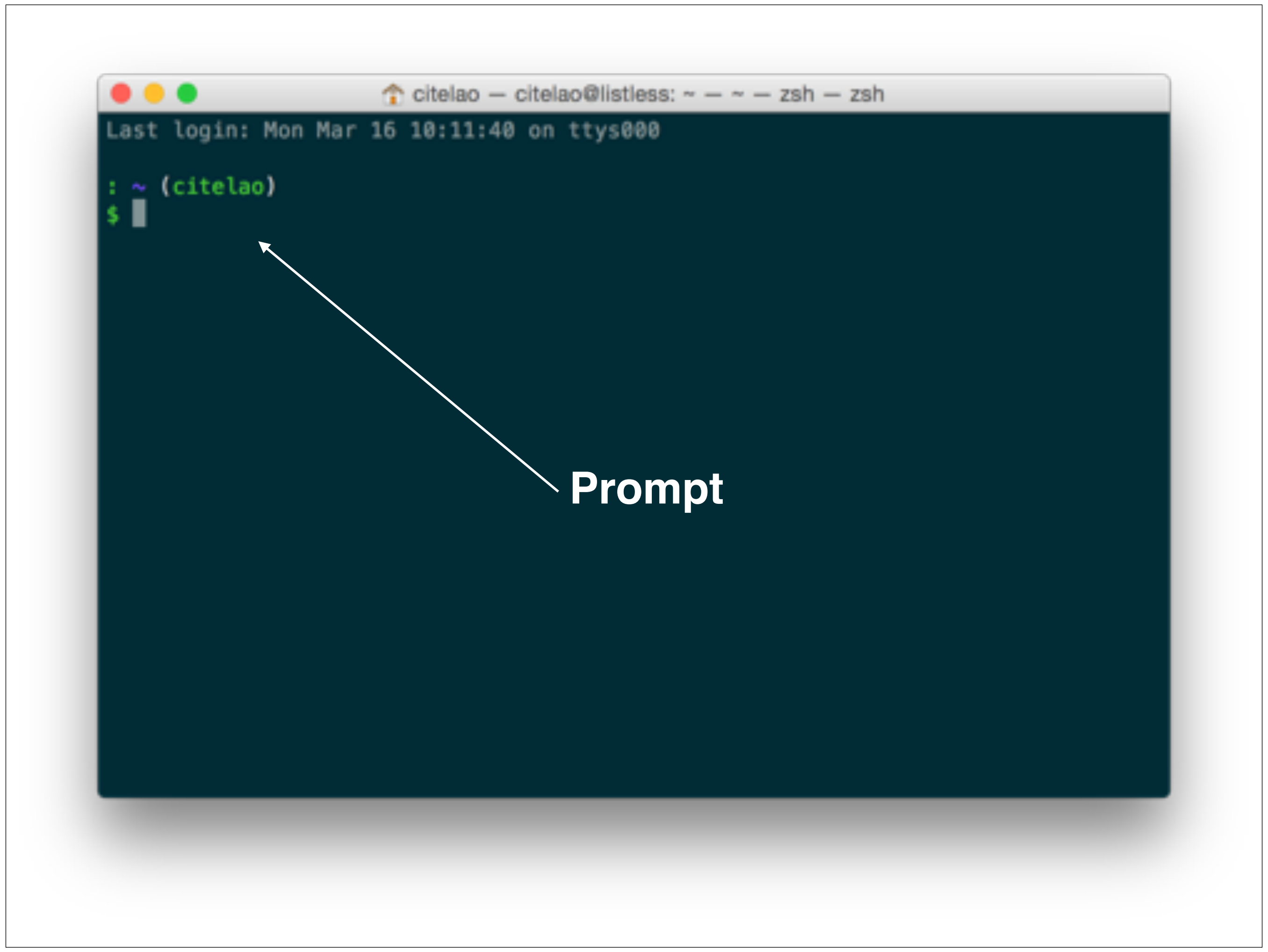
they are used pretty much interchangeably by most people. I didn’t bold any of these words because they all refer to similar things.



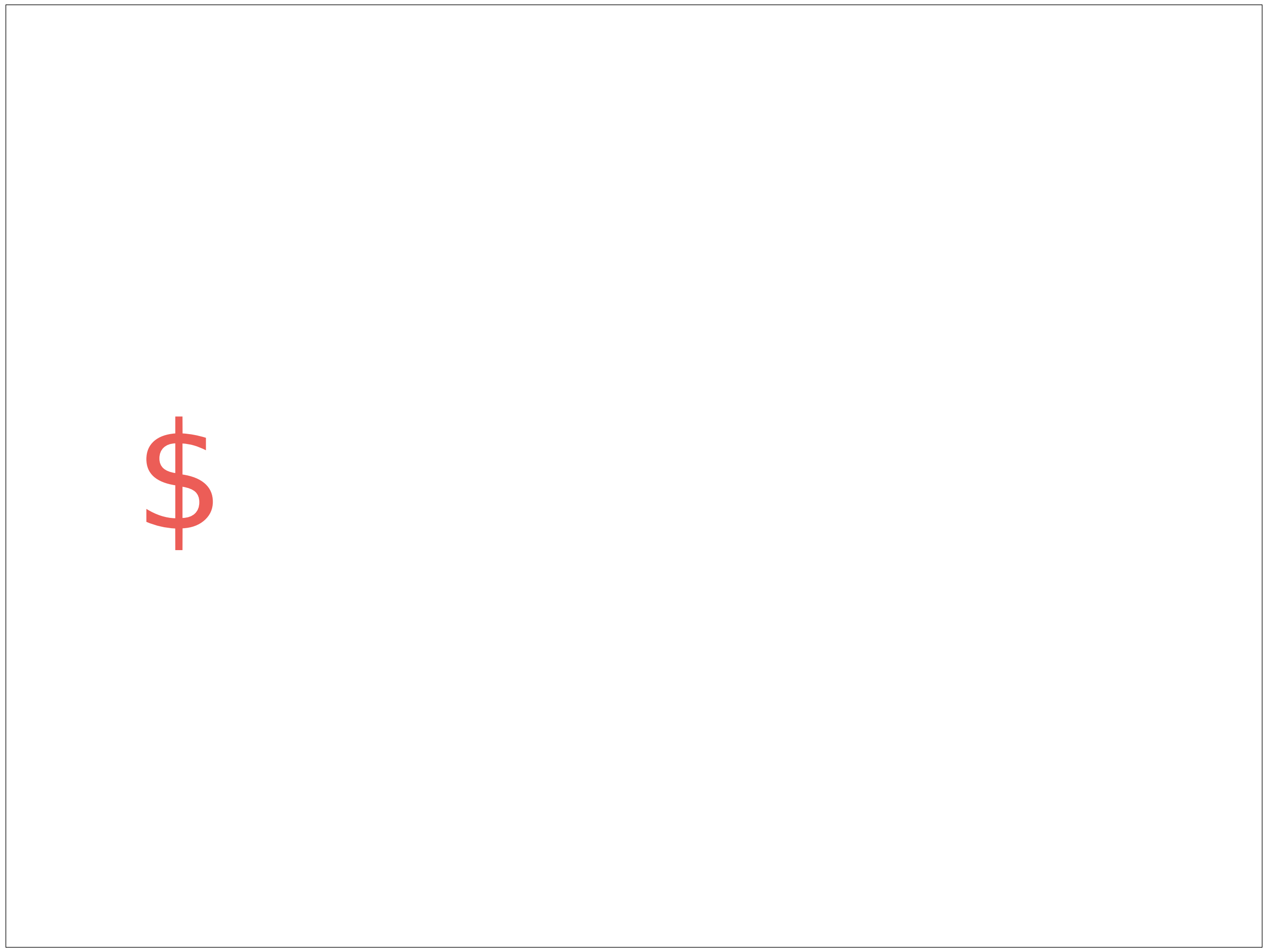
```
Last login: Mon Mar 16 10:11:40 on ttys000

: ~ (citelao)
$ 
```

anyway back to my terminal

this is Mac OS's terminal application. I have customized it because I am a pedantic UNIX nerd.


```
Last login: Mon Mar 16 10:11:40 on ttys000

: ~ (citelao)
$ 
```

Aside from custom colors and a custom shell, I have a customized prompt telling me I can enter a command.

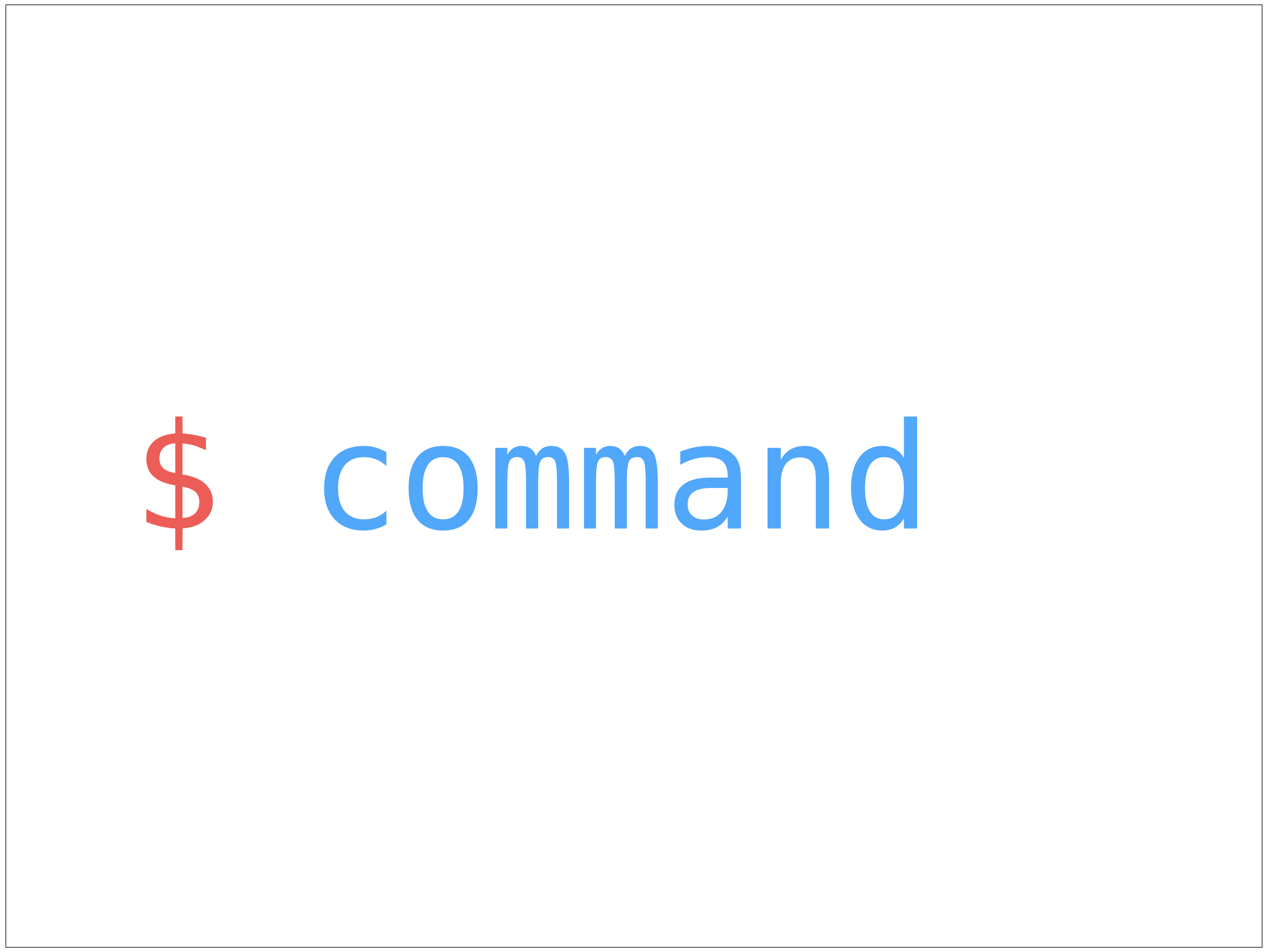


$

but I have something in common with most prompts.

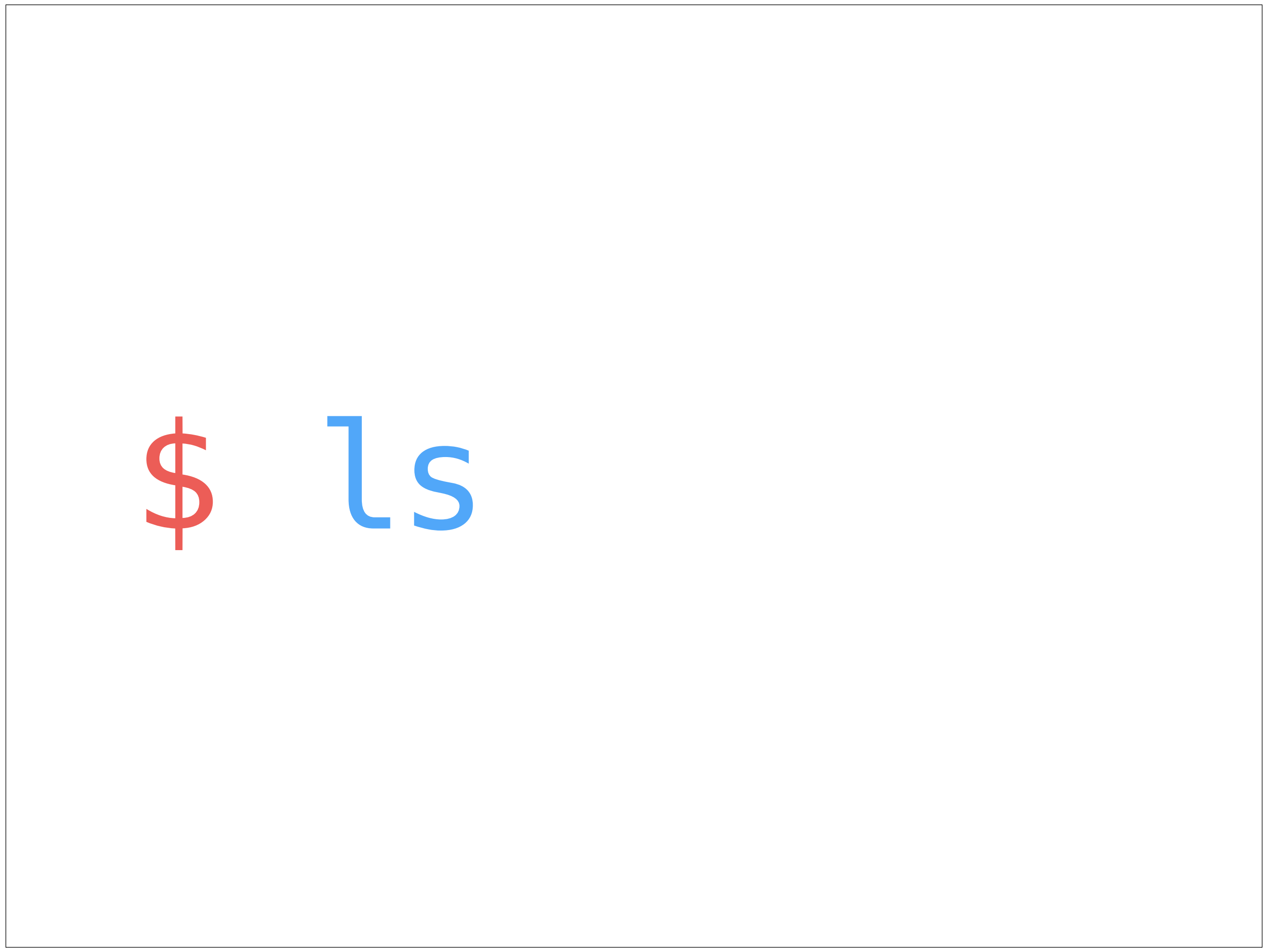
the $.

for some reason that’s the universal symbol for a prompt line.



```
$ command
```

if you see “$ command” that tells you that “command” is a shell command. You execute it in a terminal.

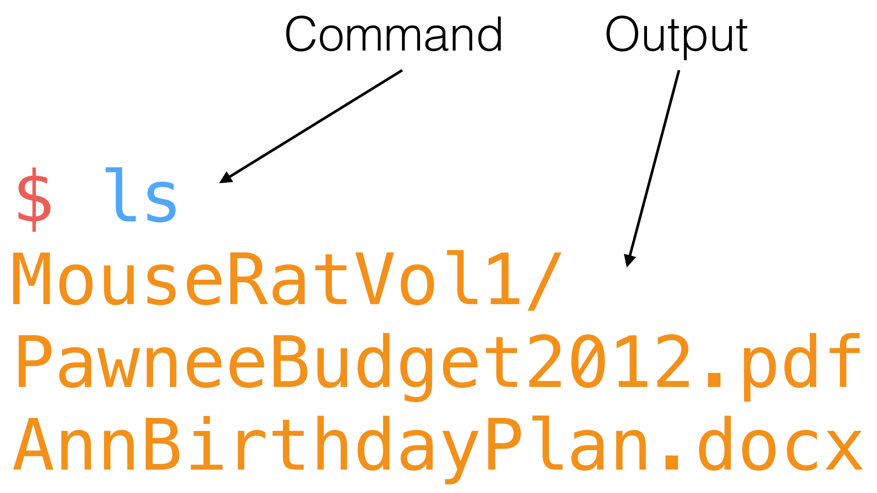


```
$ ls
```

so when I open my terminal and run “ls”, I don’t actually *type* the $.

```
$ ls
MouseRatVol1/
PawneeBudget2012.pdf
AnnBirthdayPlan.docx
```

It just indicates to me that this is a command and not, say, output from one



```
$ ls
MouseRatVol1/
PawneeBudget2012.pdf
AnnBirthdayPlan.docx
```

How do I do things?

How do I do things?

run commands.

“Commands”

commands.

CSE330 wiki

“Linux” article, “Command Reference”

this article has a list of some common commands.

http://classes.engineering.wustl.edu/cse330/index.php/Linux#Command_Reference

it also talks a lot about linux as a whole

“Commands”

commands.

~~“Commands”~~
Files

Let’s talk about files.

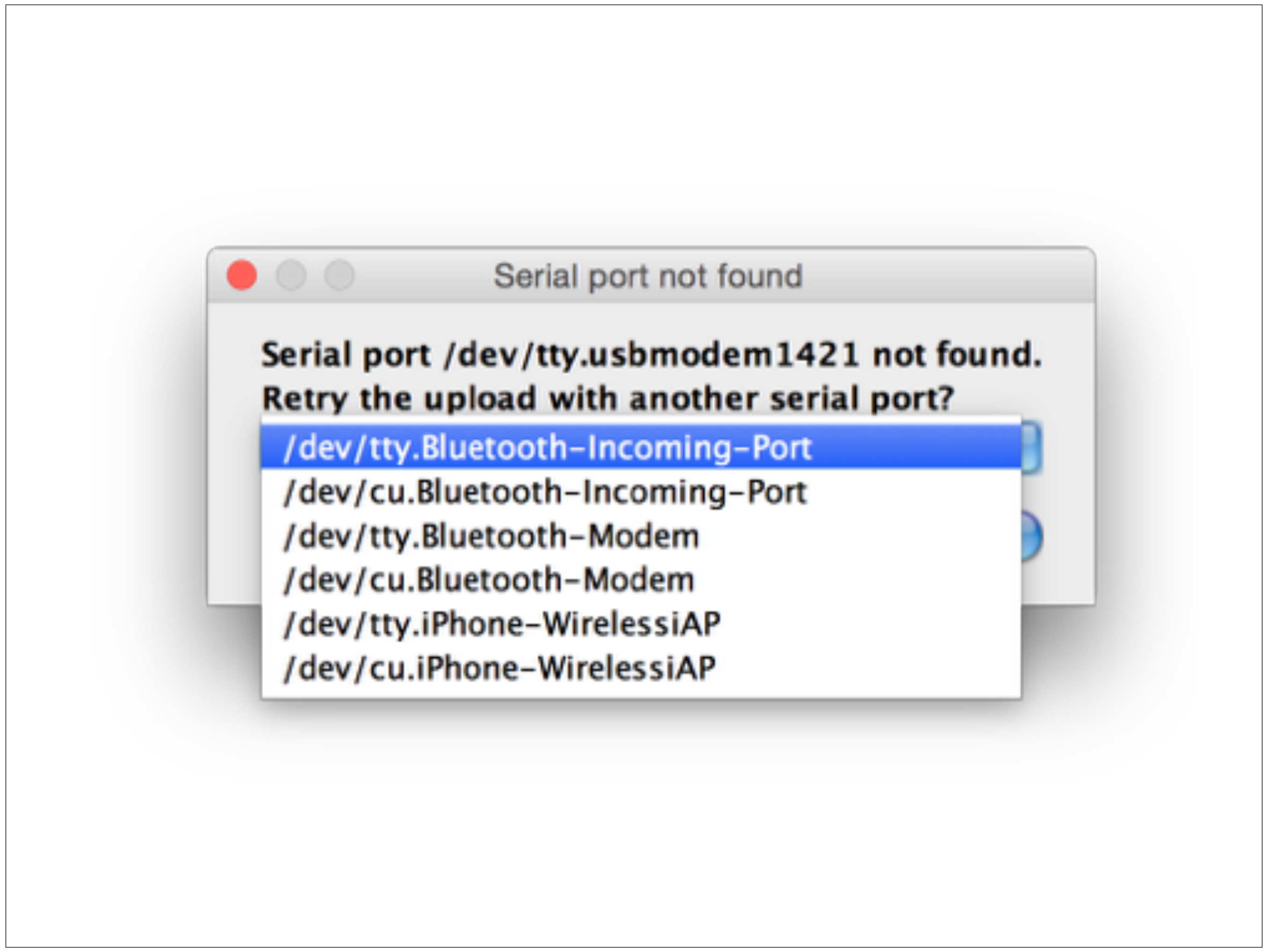
one big revelation behind UNIX is that *everything* is a file.

you execute commands— that happen to be files— on other files— and the output is written to a file.

so that’s one of the reasons for so many commands. I can just write one in 5 minutes, combining *other, existing* commands. (never mind the chicken/egg problem)

Often, of course, things aren’t really “files” with plaintext. Some might be programs that generate output to *look* like a file.

http://www.howtogeek.com/117939/htg-explains-what-everything-is-a-file-means-on-linux/

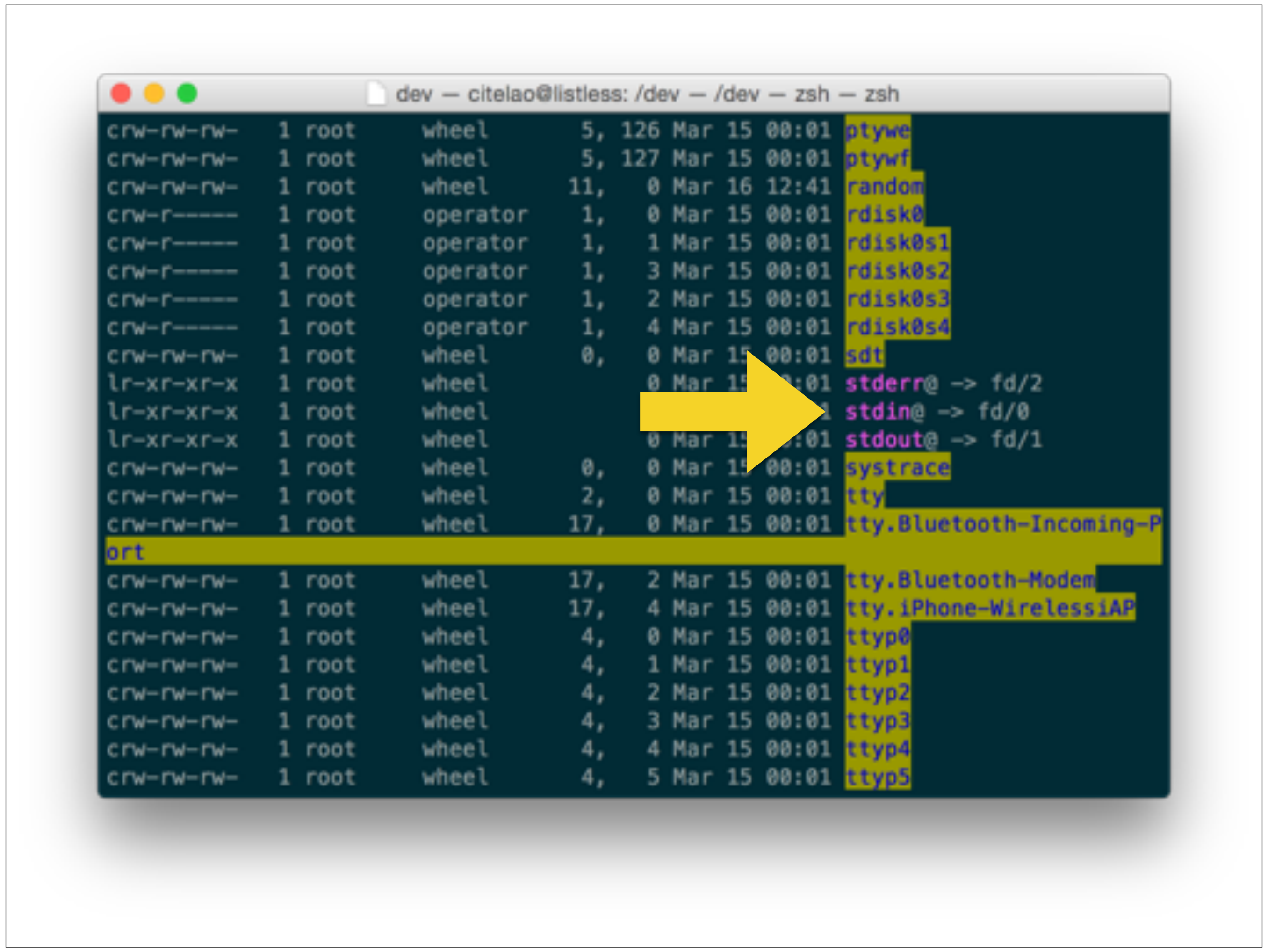


Serial port not found

Serial port /dev/tty.usbmodem1421 not found.
Retry the upload with another serial port?

```
/dev/tty.Bluetooth-Incoming-Port
/dev/cu.Bluetooth-Incoming-Port
/dev/tty.Bluetooth-Modem
/dev/cu.Bluetooth-Modem
/dev/tty.iPhone-WirelessiAP
/dev/cu.iPhone-WirelessiAP
```

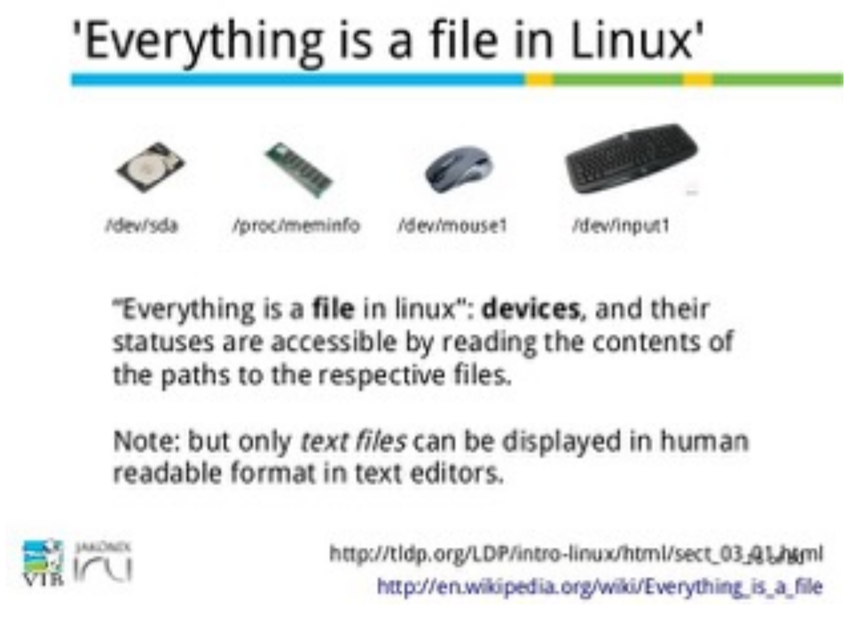
some are meant only to be written to; like the ports you write to on your Arduino, which are handled by the OS to write programs to the device



```
dev — citelao@listless: /dev — /dev — zsh — zsh
crw-rw-rw-  1 root  wheel       5, 126 Mar 15 00:01 ptywe
crw-rw-rw-  1 root  wheel       5, 127 Mar 15 00:01 ptywf
crw-rw-rw-  1 root  wheel      11,   0 Mar 16 12:41 random
crw-r-----  1 root  operator    1,   0 Mar 15 00:01 rdisk0
crw-r-----  1 root  operator    1,   1 Mar 15 00:01 rdisk0s1
crw-r-----  1 root  operator    1,   3 Mar 15 00:01 rdisk0s2
crw-r-----  1 root  operator    1,   2 Mar 15 00:01 rdisk0s3
crw-r-----  1 root  operator    1,   4 Mar 15 00:01 rdisk0s4
crw-rw-rw-  1 root  wheel       0,   0 Mar 15 00:01 sdt
lr-xr-xr-x  1 root  wheel            0 Mar 15 00:01 stderr@ -> fd/2
lr-xr-xr-x  1 root  wheel                         1 stdin@ -> fd/0
lr-xr-xr-x  1 root  wheel            0 Mar 15 00:01 stdout@ -> fd/1
crw-rw-rw-  1 root  wheel       0,   0 Mar 15 00:01 systrace
crw-rw-rw-  1 root  wheel       2,   0 Mar 15 00:01 tty
crw-rw-rw-  1 root  wheel      17,   0 Mar 15 00:01 tty.Bluetooth-Incoming-P
ort
crw-rw-rw-  1 root  wheel      17,   2 Mar 15 00:01 tty.Bluetooth-Modem
crw-rw-rw-  1 root  wheel      17,   4 Mar 15 00:01 tty.iPhone-WirelessiAP
crw-rw-rw-  1 root  wheel       4,   0 Mar 15 00:01 ttyp0
crw-rw-rw-  1 root  wheel       4,   1 Mar 15 00:01 ttyp1
crw-rw-rw-  1 root  wheel       4,   2 Mar 15 00:01 ttyp2
crw-rw-rw-  1 root  wheel       4,   3 Mar 15 00:01 ttyp3
crw-rw-rw-  1 root  wheel       4,   4 Mar 15 00:01 ttyp4
crw-rw-rw-  1 root  wheel       4,   5 Mar 15 00:01 ttyp5
```

even your little terminal windows are secretly attached to files

“stdout” for output
“stdin” for input
“stderr” for errors



'Everything is a file in Linux'

/dev/sda /proc/meminfo /dev/mouse1 /dev/input1

"Everything is a **file** in linux": **devices**, and their statuses are accessible by reading the contents of the paths to the respective files.

Note: but only *text files* can be displayed in human readable format in text editors.

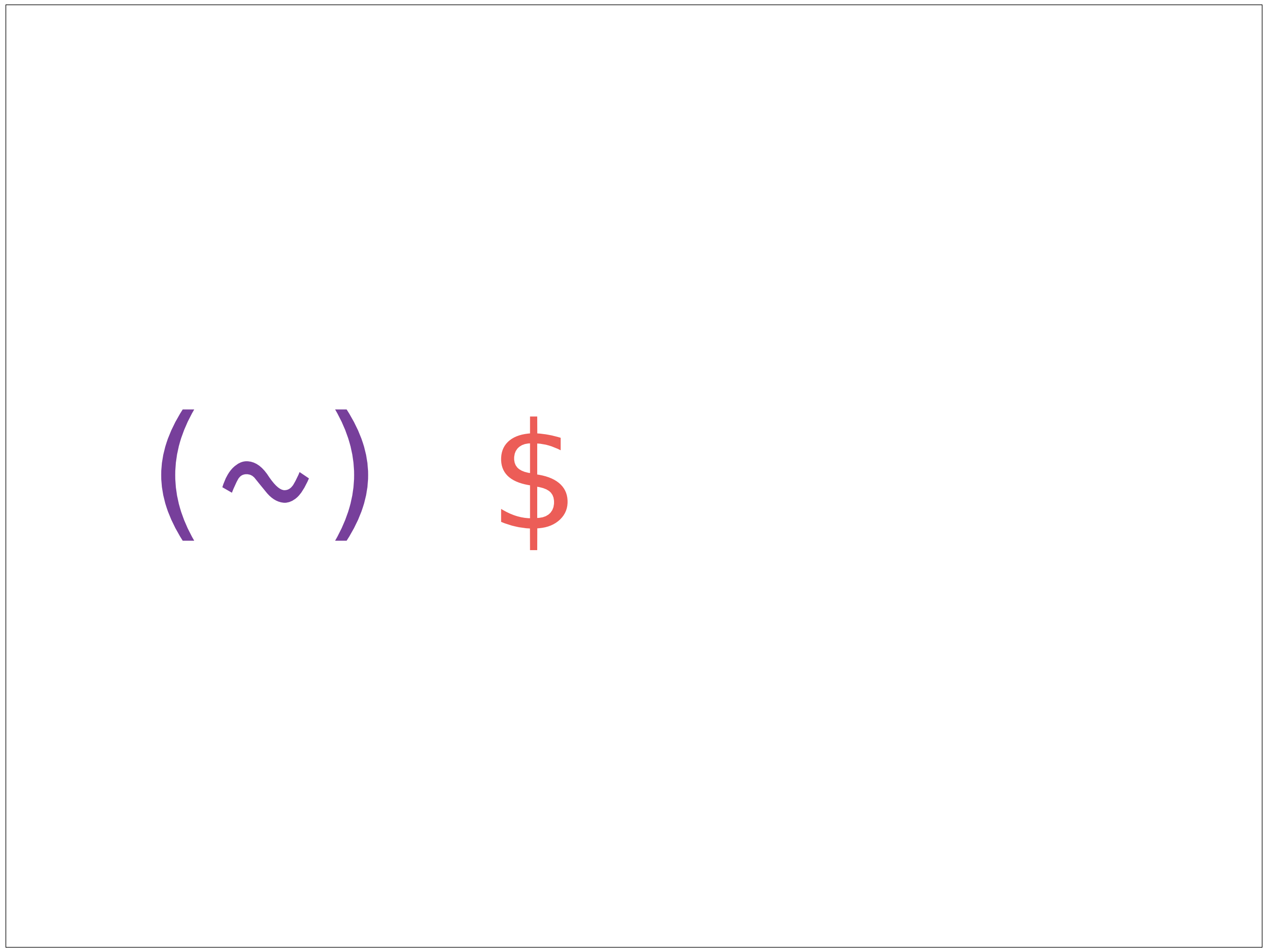
http://tldp.org/LDP/intro-linux/html/sect_03_01.html
http://en.wikipedia.org/wiki/Everything_is_a_file

(via Joachim Jacob)

I stole this slide.

We can get keyboard information by reading `/dev/inputWhatever`, we could parse it and write it to a file on our hard drive.

Ok, but how do I do things?



```
(~) $
```

Let’s talk about your terminal.

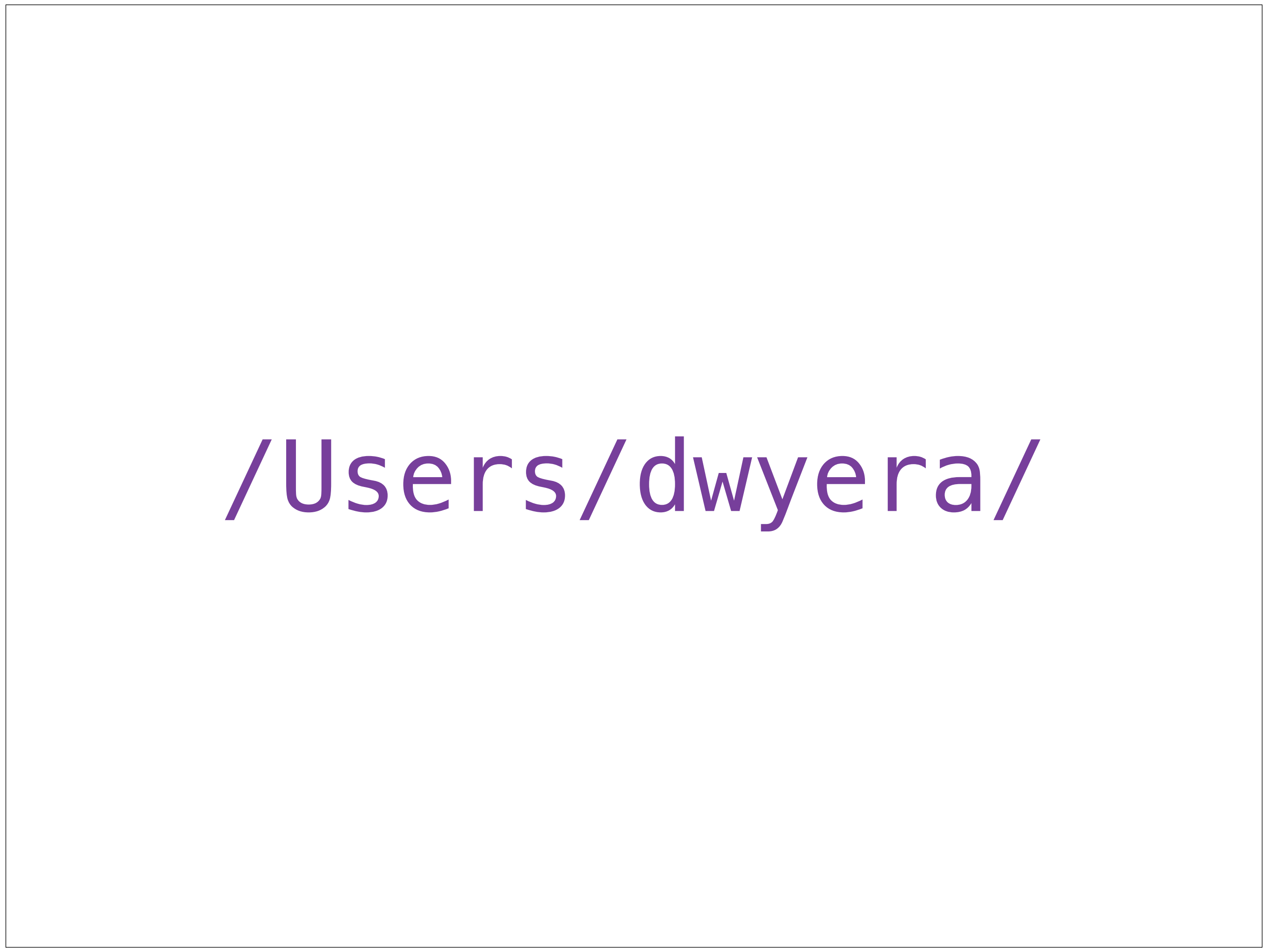
your prompt probably has something else in it, besides a $.

‘~’ == ‘/Users/citelao’

It might be a tilde, but it is a path. It’s your home folder. On Mac, it’s what holds “Documents” “Photos”… all the stuff specific to your user.

/

all paths are relative to the “root directory”… just a slash.



```
/Users/dwyera/
```

like my home folder

`/Applications/`

or my applications folder

```
/Users/knopel/
Documents/
BenLoveLetter1042.txt
```

or this text file

Absolute vs Relative

`/Users/haverfordt/Documents/BusinessPlan42.pdf`

vs

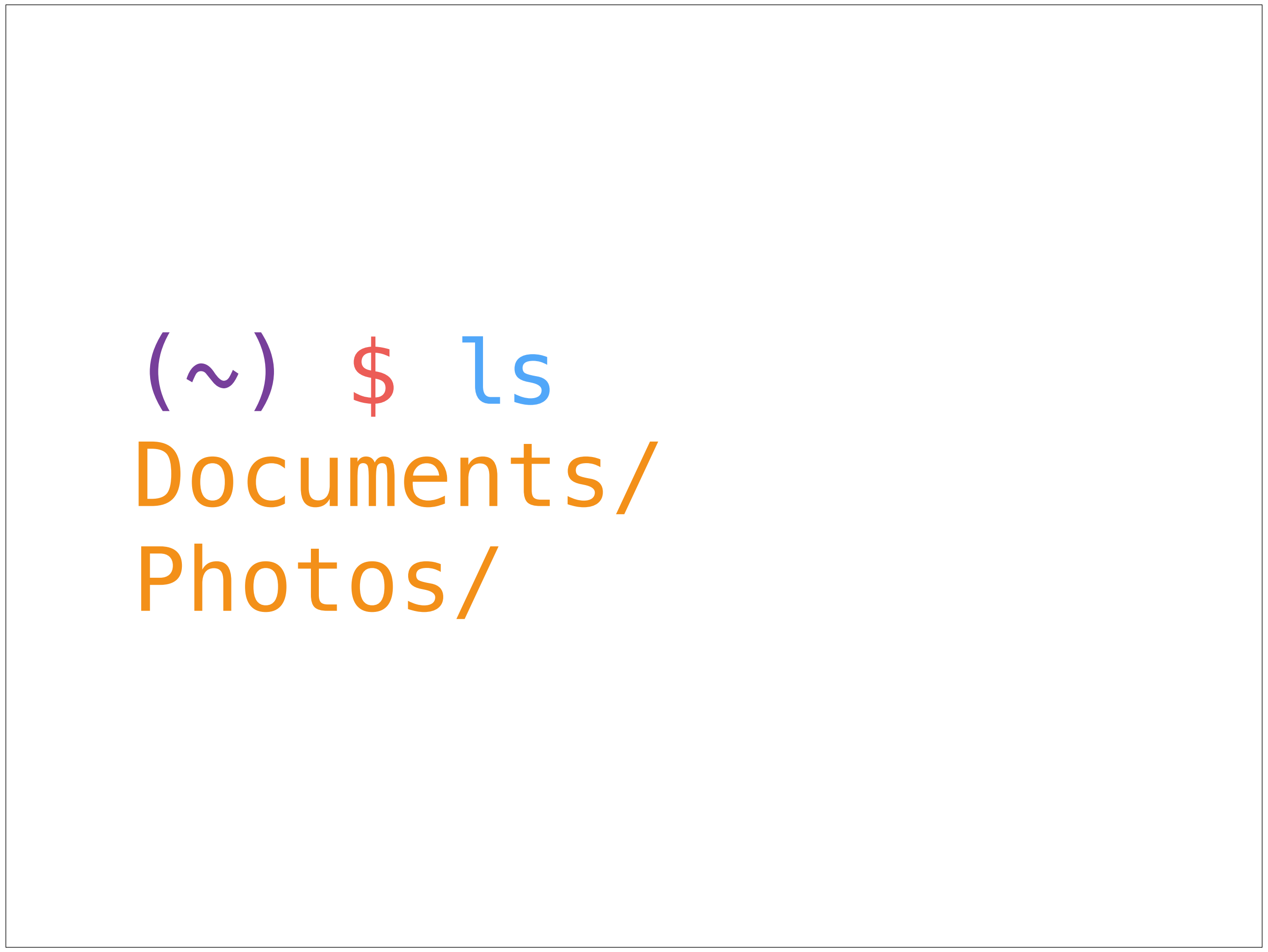
`Documents/BusinessPlan42.pdf`

those were all absolute paths. They started from the very top, the root directory, and showed where the file was.

a relative path would be *relative* to some other directory. Usually absolute paths have a “/“ at the beginning.

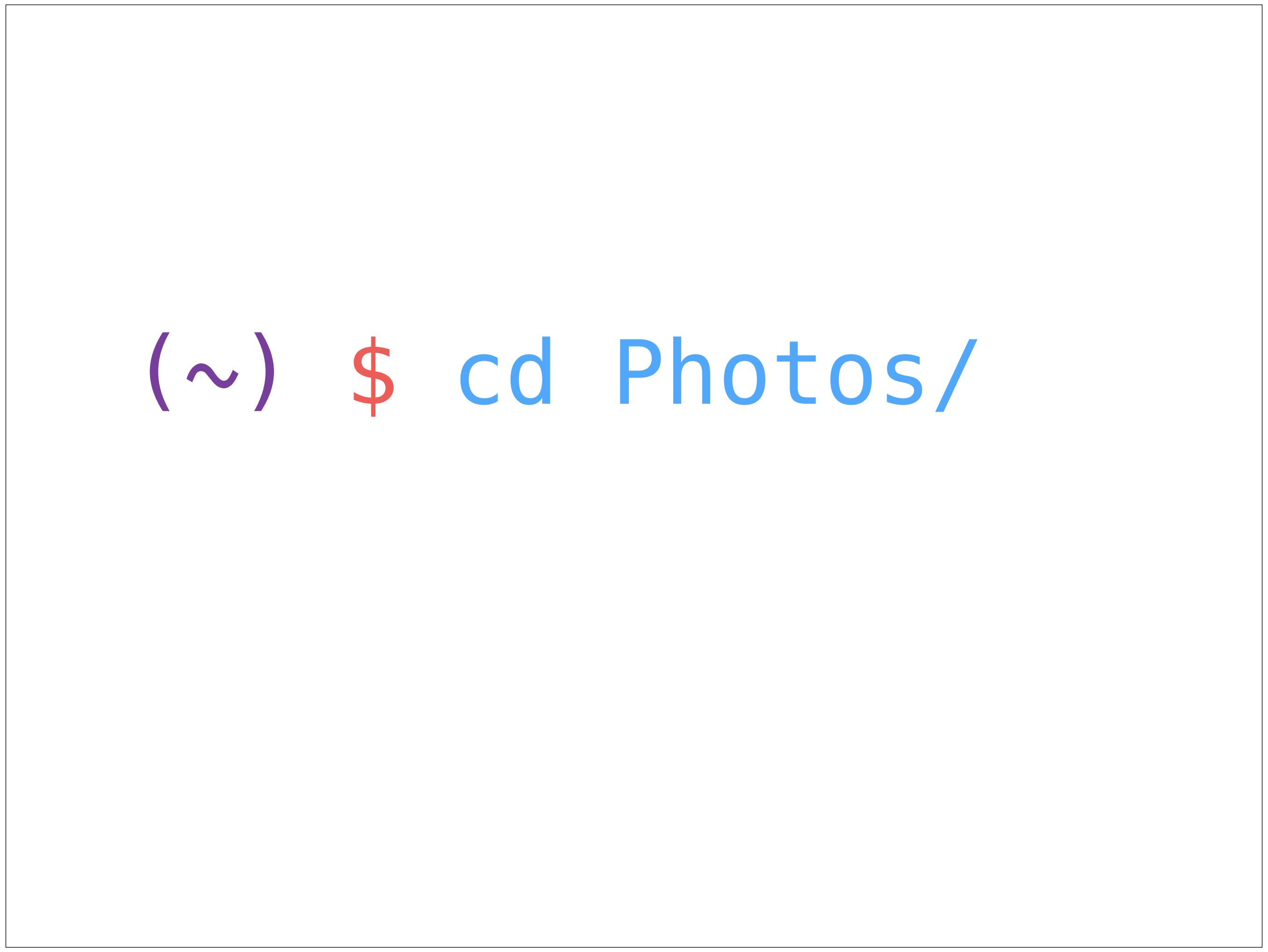
Your terminal is a file browser

you can navigate around your computer with your terminal



```
(~) $ ls
Documents/
Photos/
```

ls lists files in your current directory



```
(~) $ cd Photos/
```

cd changes to a directory relative to your current one.

```
(~/Photos) $ ls
JerryBlackmail/
HarvestFest001.jpg
```

```
(~/Photos) $ cd ..
```

and “..” means parent.

(∼) $

/, ~, ., ..

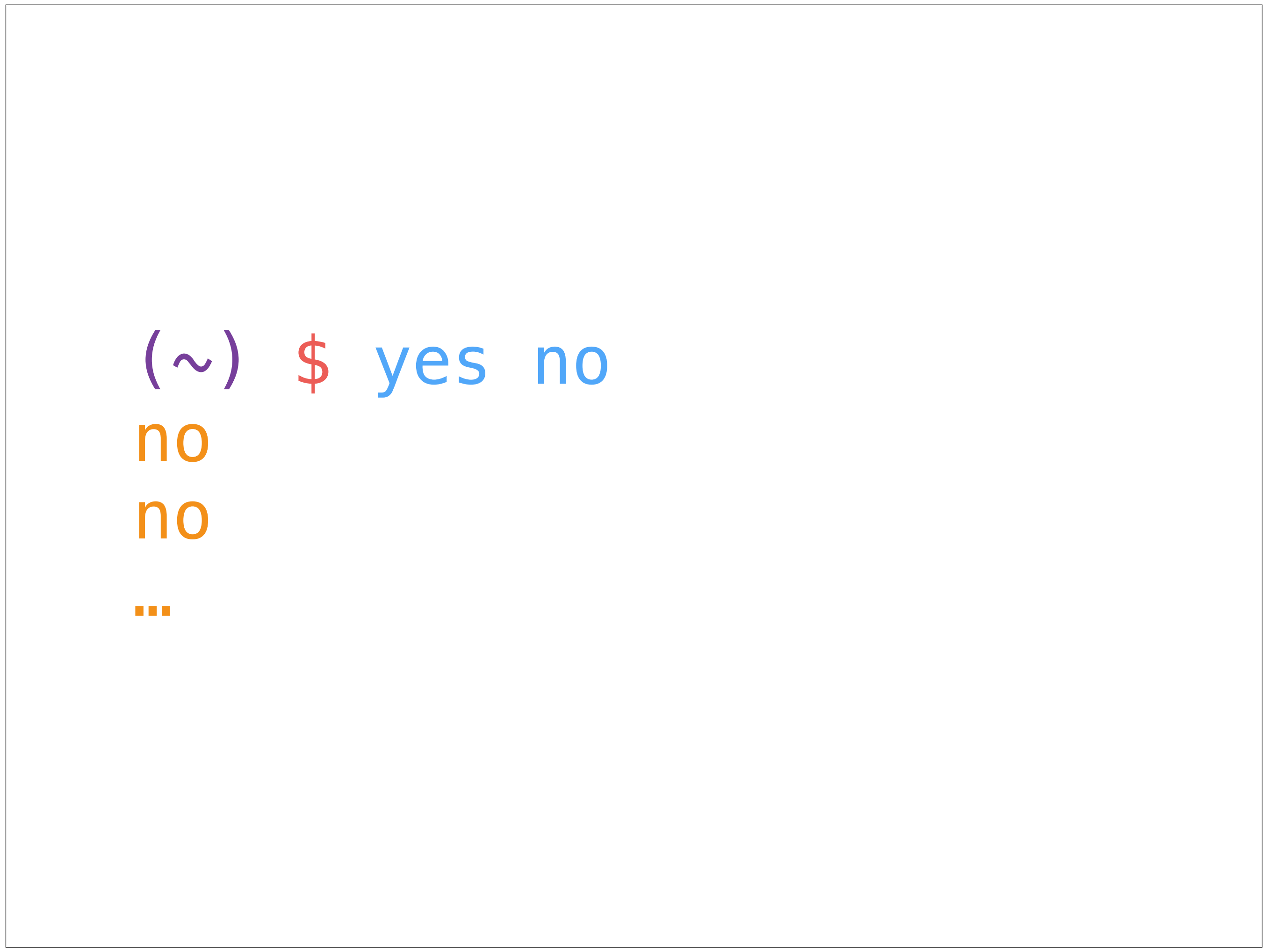
these are the confusing new symbols.

root directory
home directory
current directory
parent directory

```
(~) $ canIPutAnythingHere
```

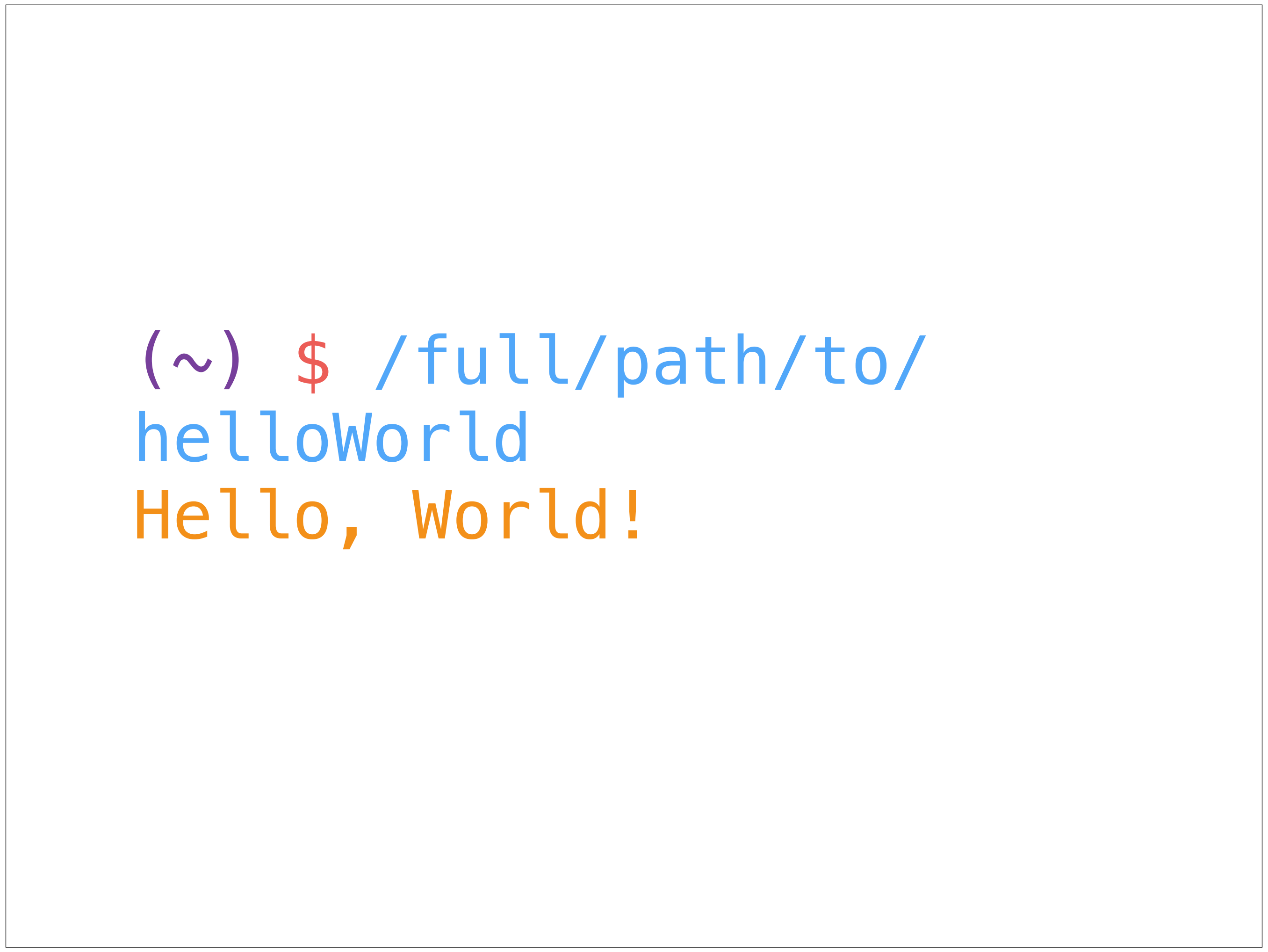
so what about the `.`, when is that useful

what if you want to execute a file as a command?



```
(~) $ yes no
no
no
…
```

no. you cannot put anything in as a command.

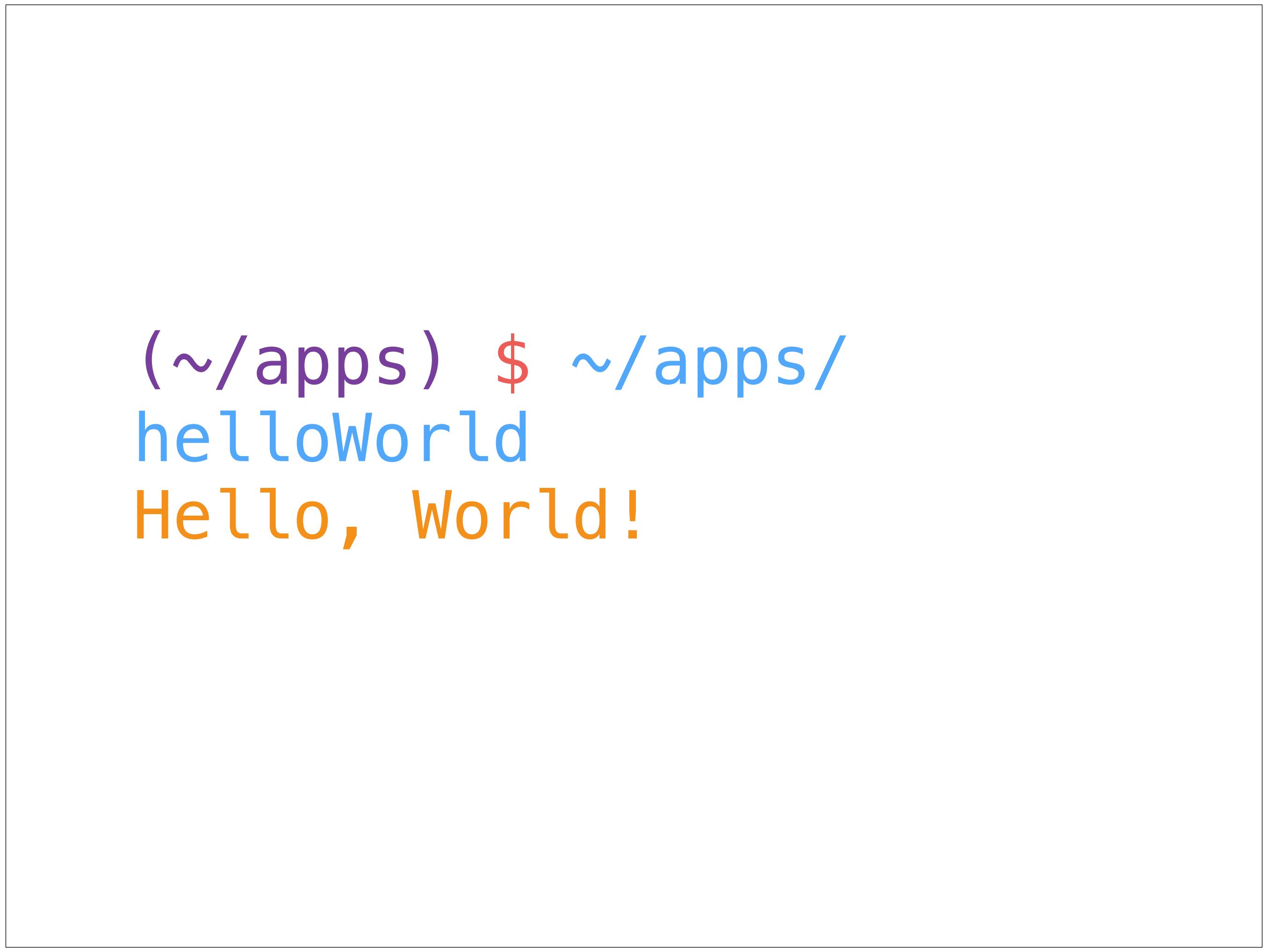


```
(~) $ /full/path/to/
helloWorld
Hello, World!
```

but you *can* run any file as a command. you just need the full path to the file.

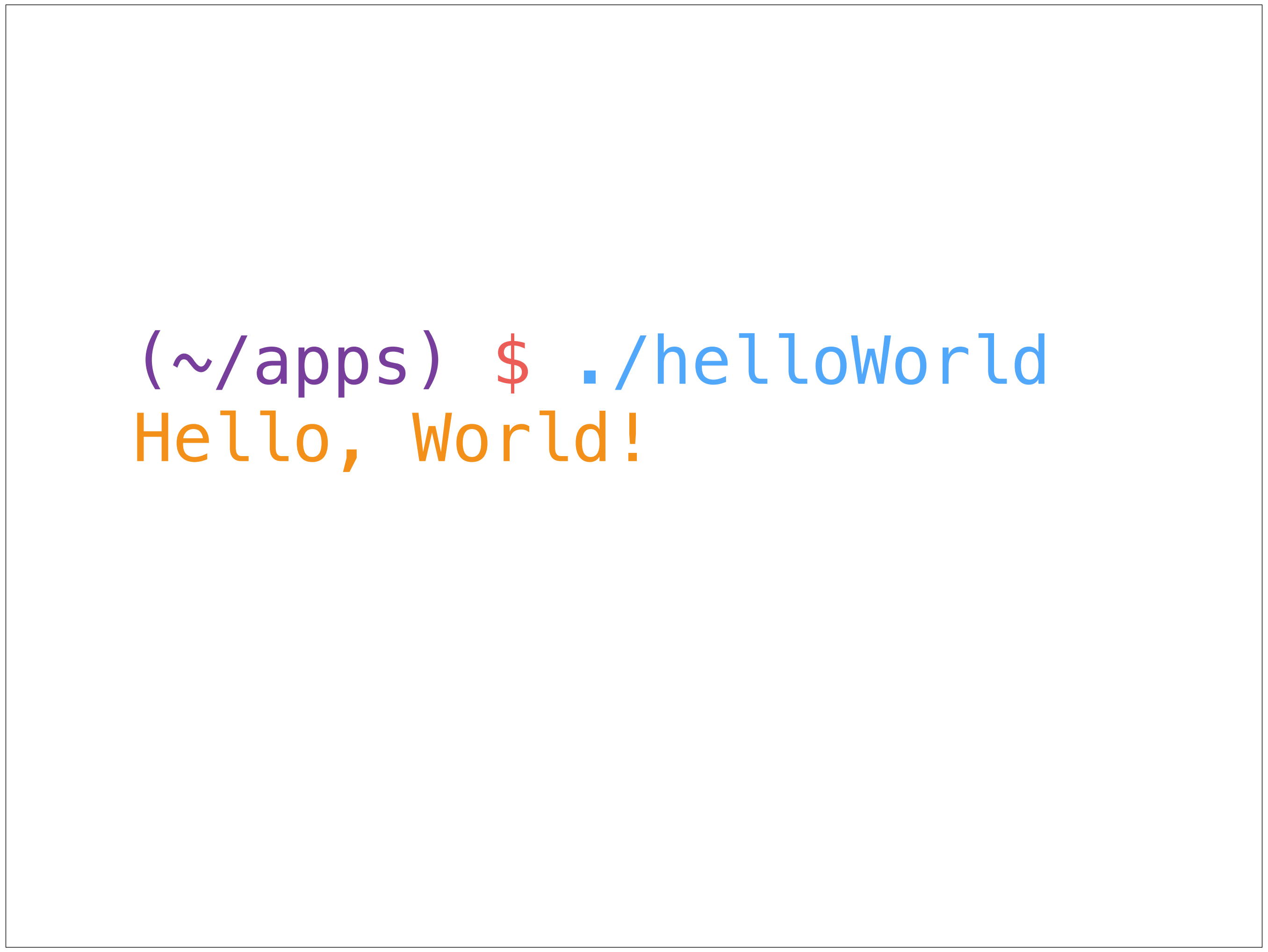
```
(~/apps) $ /Users/dwyera/
apps/helloWorld
Hello, World!
```

so if my hello world app is in a folder in my home directory, I could run it like this



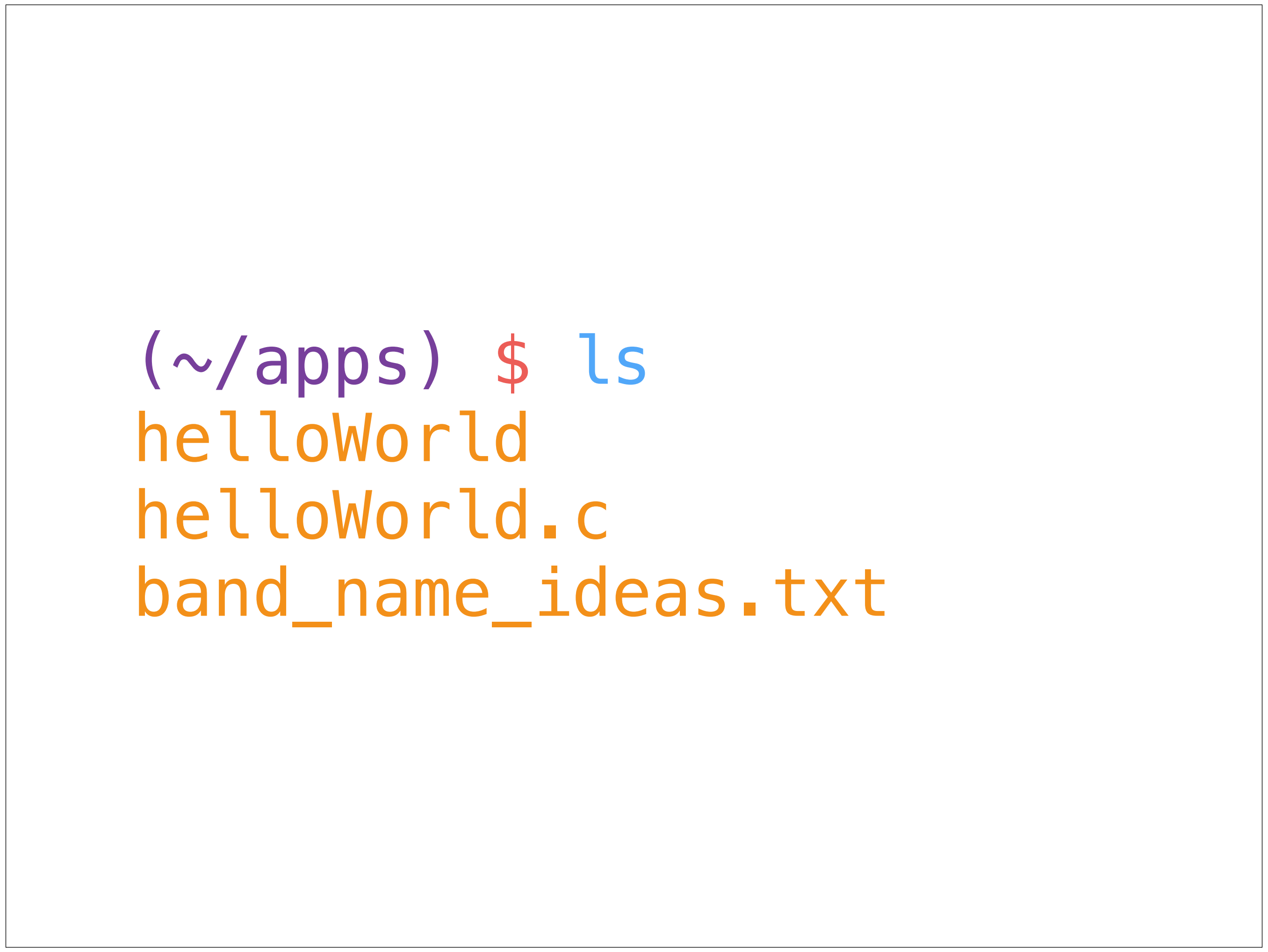
```
(~/apps) $ ~/apps/
helloWorld
Hello, World!
```

or this



```
(~/apps) $ ./helloWorld
Hello, World!
```

or this (look at my current directory before the $)



```
(~/apps) $ ls
helloWorld
helloWorld.c
band_name_ideas.txt
```

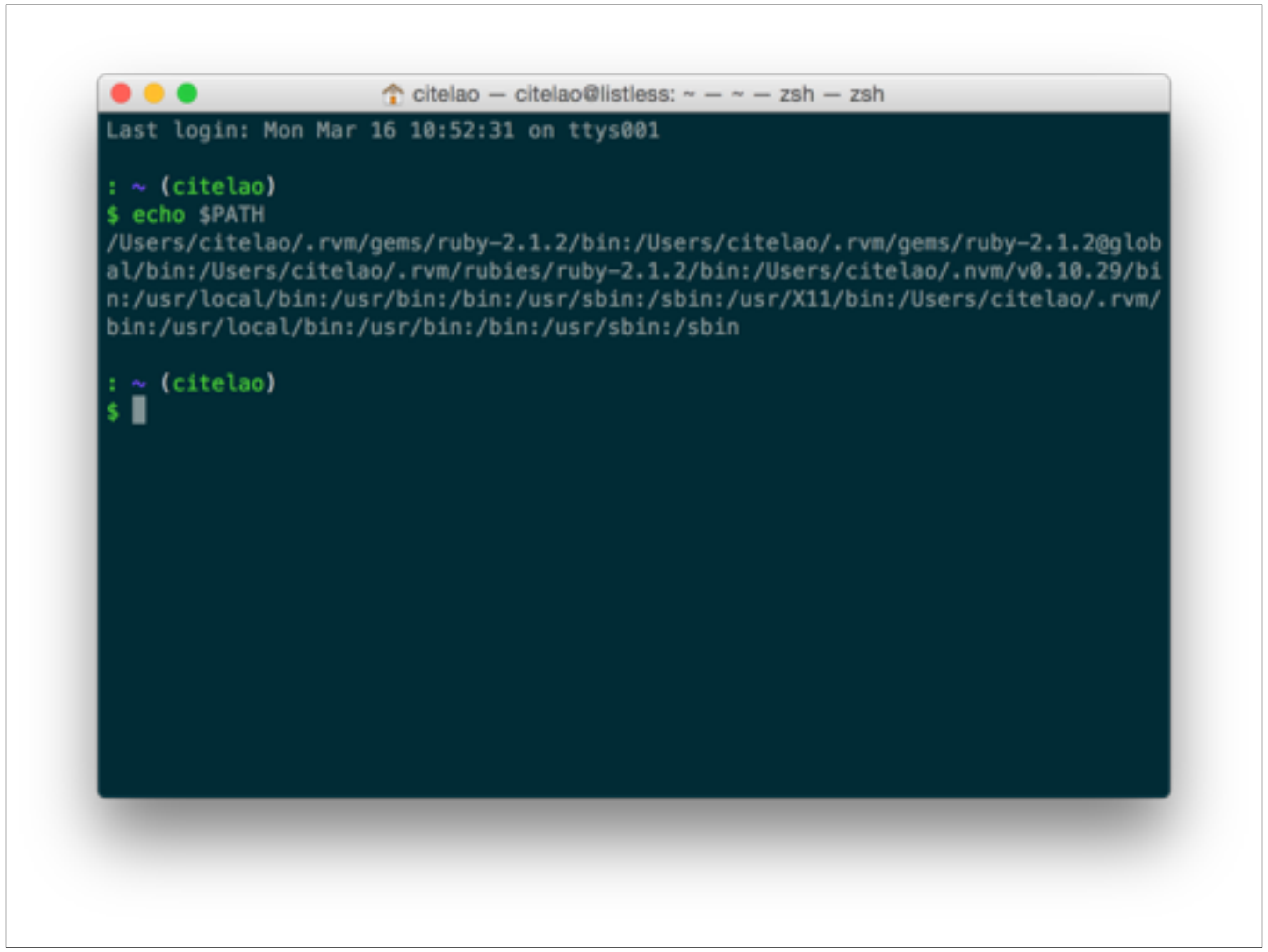
so why can I run certain commands without prefixing their path?

$PATH

There’s a shell variable called PATH.

Remember, the shell handles sending your commands to the OS.

It looks at each directory in the $PATH for a file with the same name as the command you seek. If it doesn’t find the command, it assumes the command is an absolute path.

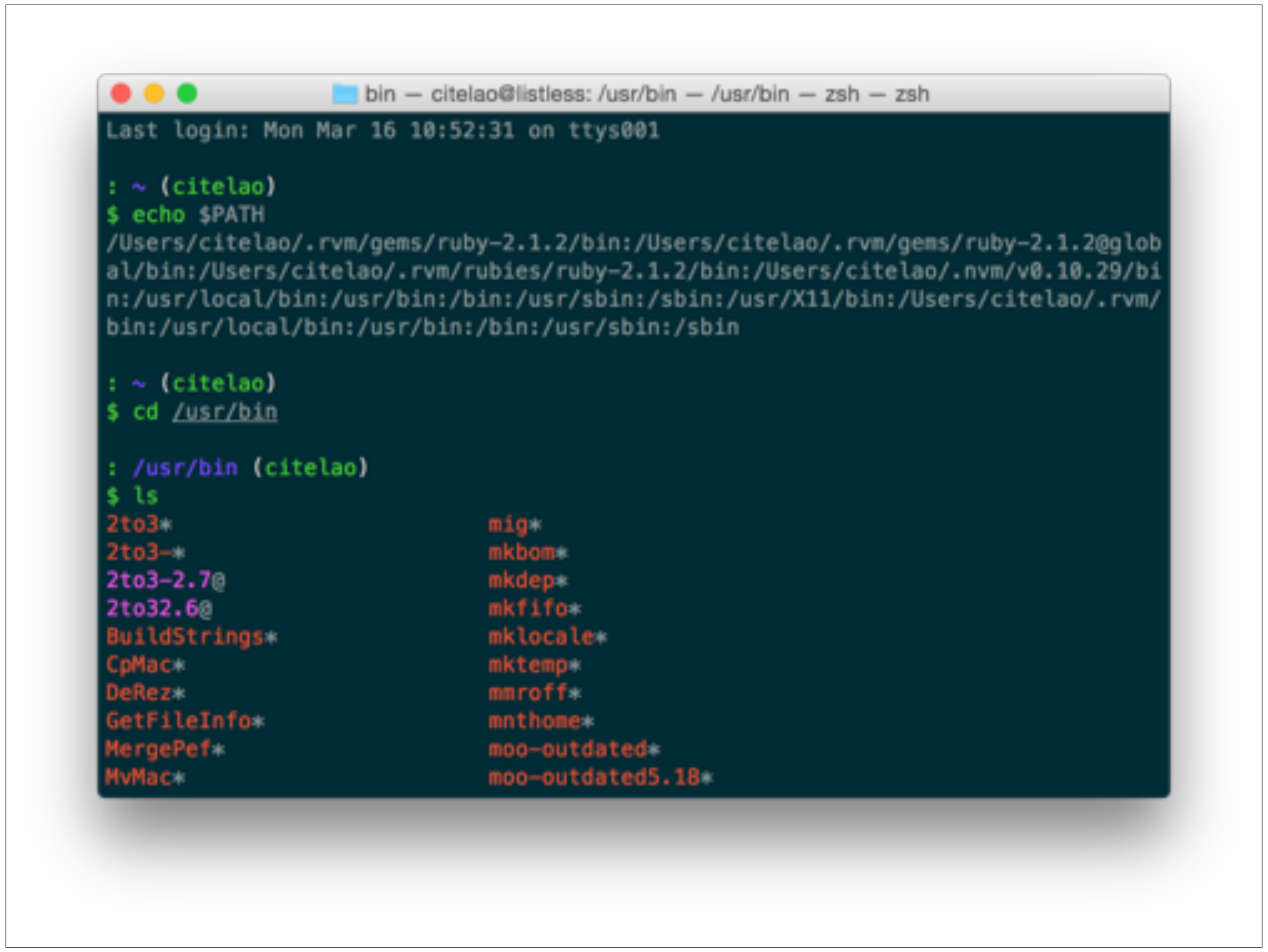


```
Last login: Mon Mar 16 10:52:31 on ttys001

: ~ (citelao)
$ echo $PATH
/Users/citelao/.rvm/gems/ruby-2.1.2/bin:/Users/citelao/.rvm/gems/ruby-2.1.2@glob
al/bin:/Users/citelao/.rvm/rubies/ruby-2.1.2/bin:/Users/citelao/.nvm/v0.10.29/bi
n:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11/bin:/Users/citelao/.rvm/
bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin

: ~ (citelao)
$ 
```

this is $PATH on my machine.

it checks the `/Users/citelao/.rvm/gems/ruby-2.1.2/bin` directory first, then `/Users/citelao/.rvm/gems/ruby-2.1.2@global/bin`, etc.

each directory is separated by a colon.

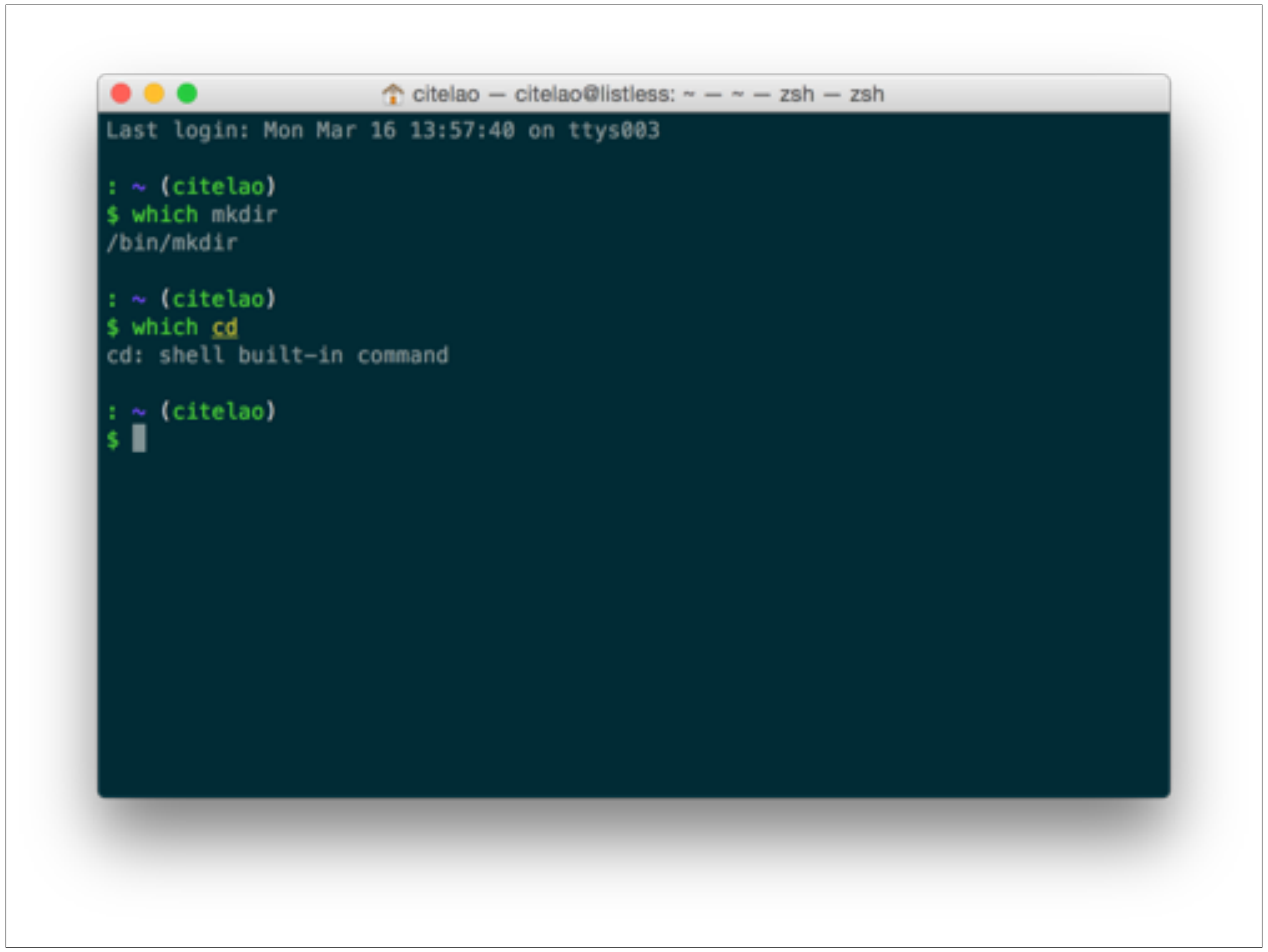


```
Last login: Mon Mar 16 10:52:31 on ttys001

: ~ (citelao)
$ echo $PATH
/Users/citelao/.rvm/gems/ruby-2.1.2/bin:/Users/citelao/.rvm/gems/ruby-2.1.2@glob
al/bin:/Users/citelao/.rvm/rubies/ruby-2.1.2/bin:/Users/citelao/.nvm/v0.10.29/bi
n:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11/bin:/Users/citelao/.rvm/
bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin

: ~ (citelao)
$ cd /usr/bin

: /usr/bin (citelao)
$ ls
2to3*                           mig*
2to3-*                          mkbom*
2to3-2.7@                       mkdep*
2to32.6@                        mkfifo*
BuildStrings*                   mklocale*
CpMac*                          mktemp*
DeRez*                          mmroff*
GetFileInfo*                    mnthome*
MergePef*                       moo-outdated*
MvMac*                          moo-outdated5.18*
```

If you navigate to one of these directories, you can see all the different commands as files.

By adding directories to the $PATH, you add directories where you can just type the command name.

I have a lot of directories in my path because I have a lot of sources of programs, something we'll cover in a later lecture.



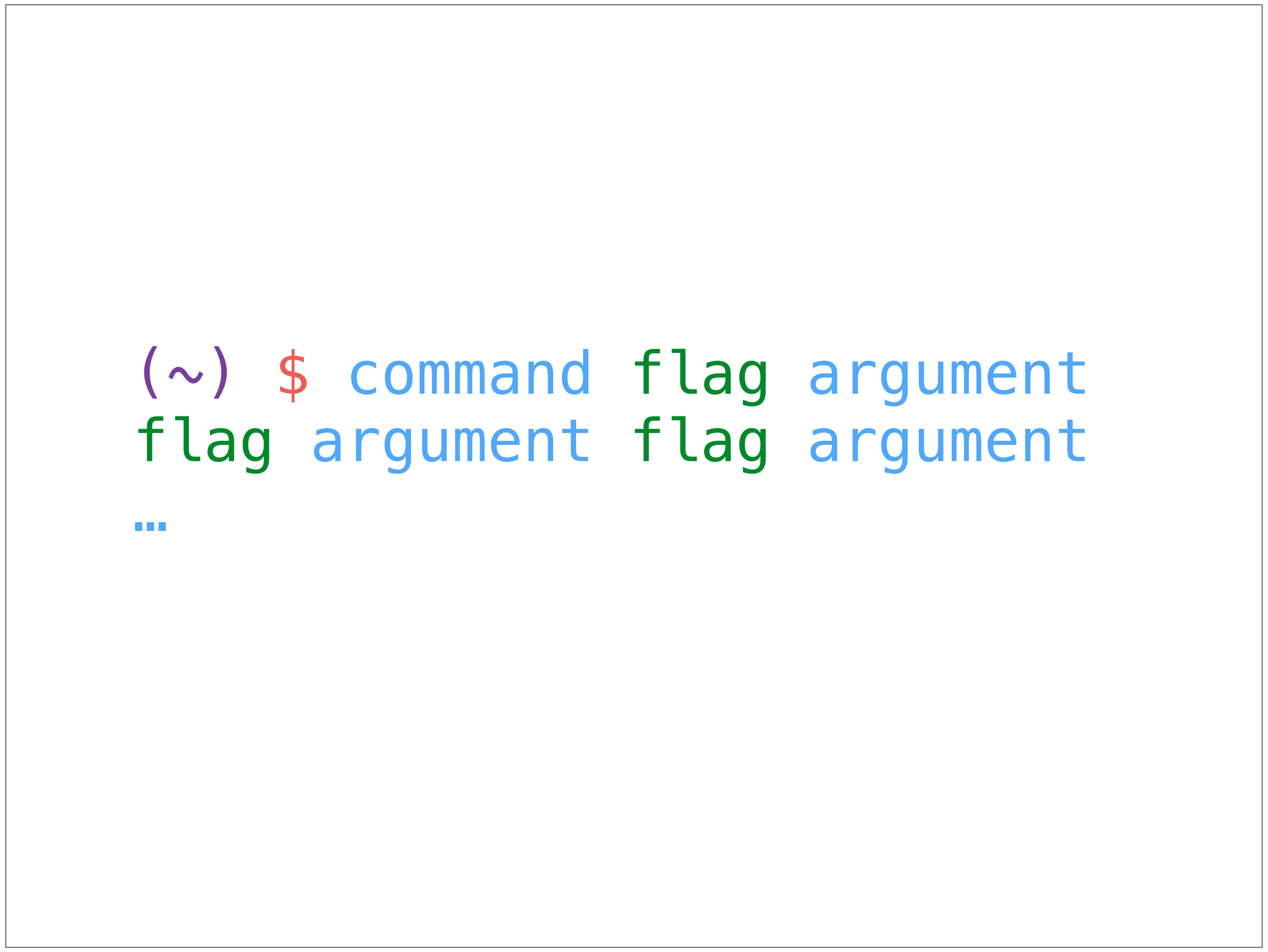
```
Last login: Mon Mar 16 13:57:40 on ttys003

: ~ (citelao)
$ which mkdir
/bin/mkdir

: ~ (citelao)
$ which cd
cd: shell built-in command

: ~ (citelao)
$ 
```

you can see what directory each command lives in by typing “which cmd”

most are actually files

some are secretly not files because operating systems are complex. Essentially, stuff that is unique to the shell— that handles moving around the operating system, all that jazz— is built-in to the shell, and is not an actual file.

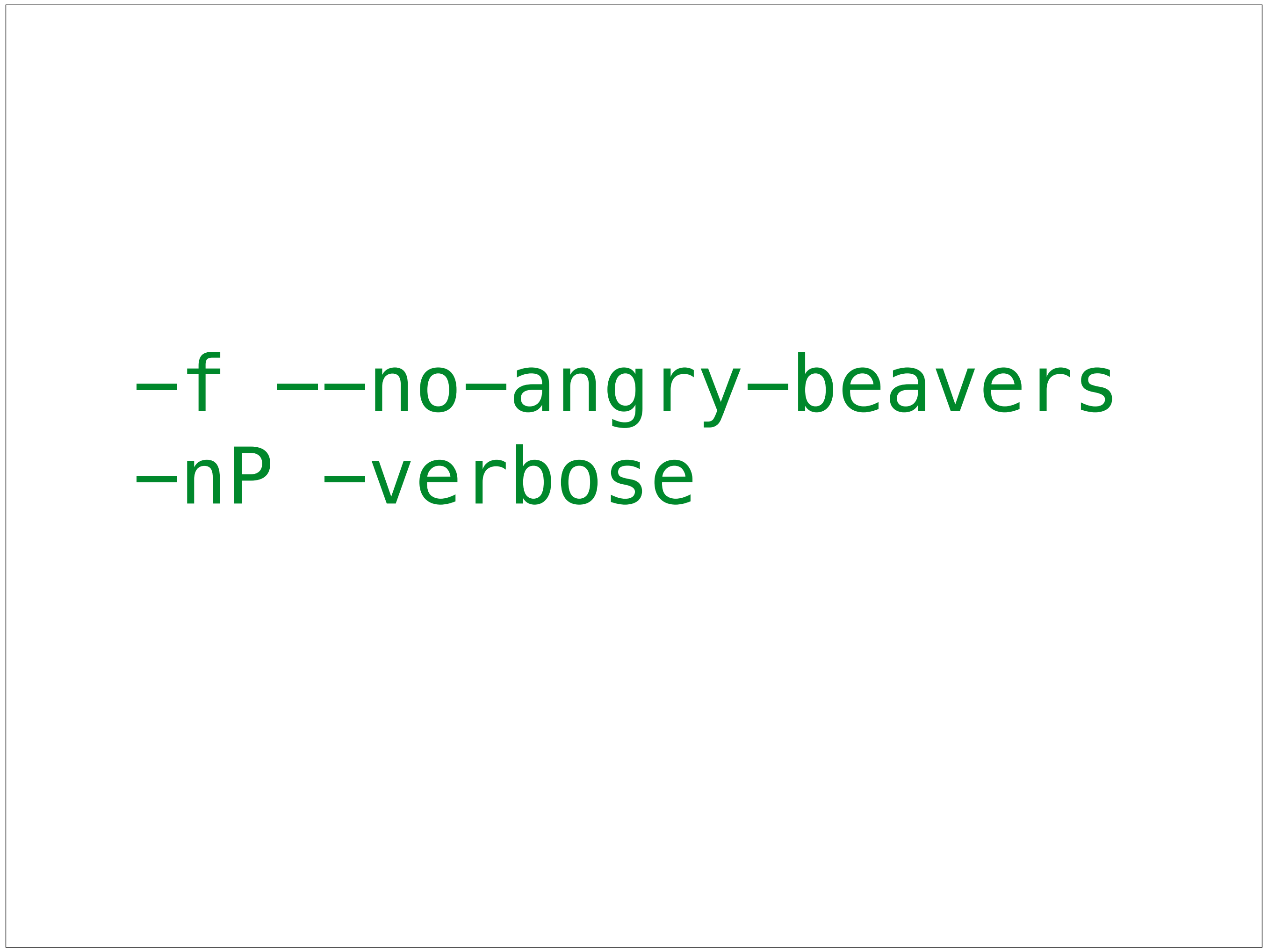
Ok… but how do I do things?



```
(~) $ command flag argument
flag argument flag argument
…
```

this is the basic syntax for commands

it’s really hard to get specific about it because you’ll see it’s really up to the program how to handle arguments.

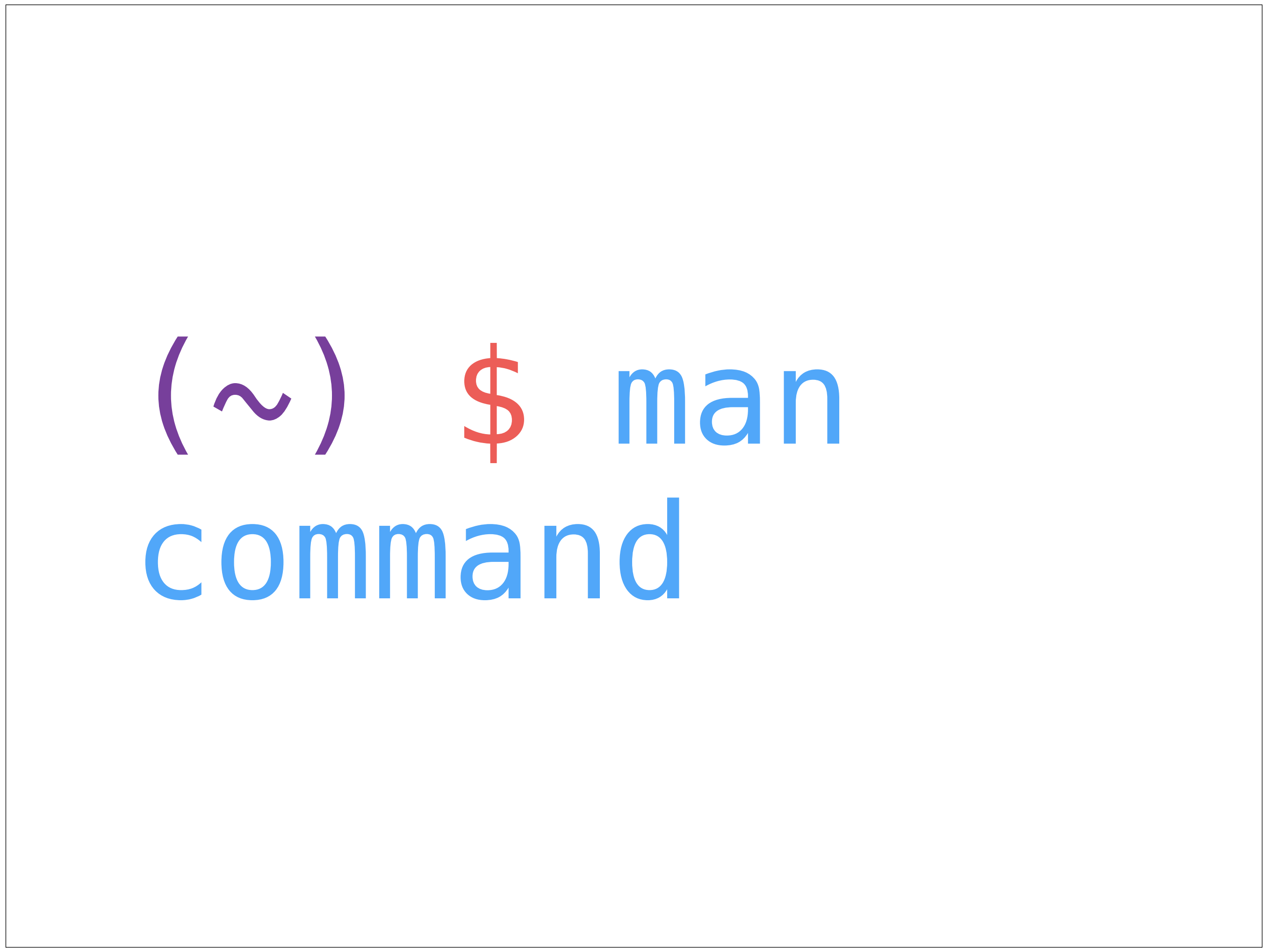


```
-f --no-angry-beavers
-nP -verbose
```

a flag, also called a switch can be a letter, or a word, prefixed with one dash or two

it might be grouped together
it might take an argument…

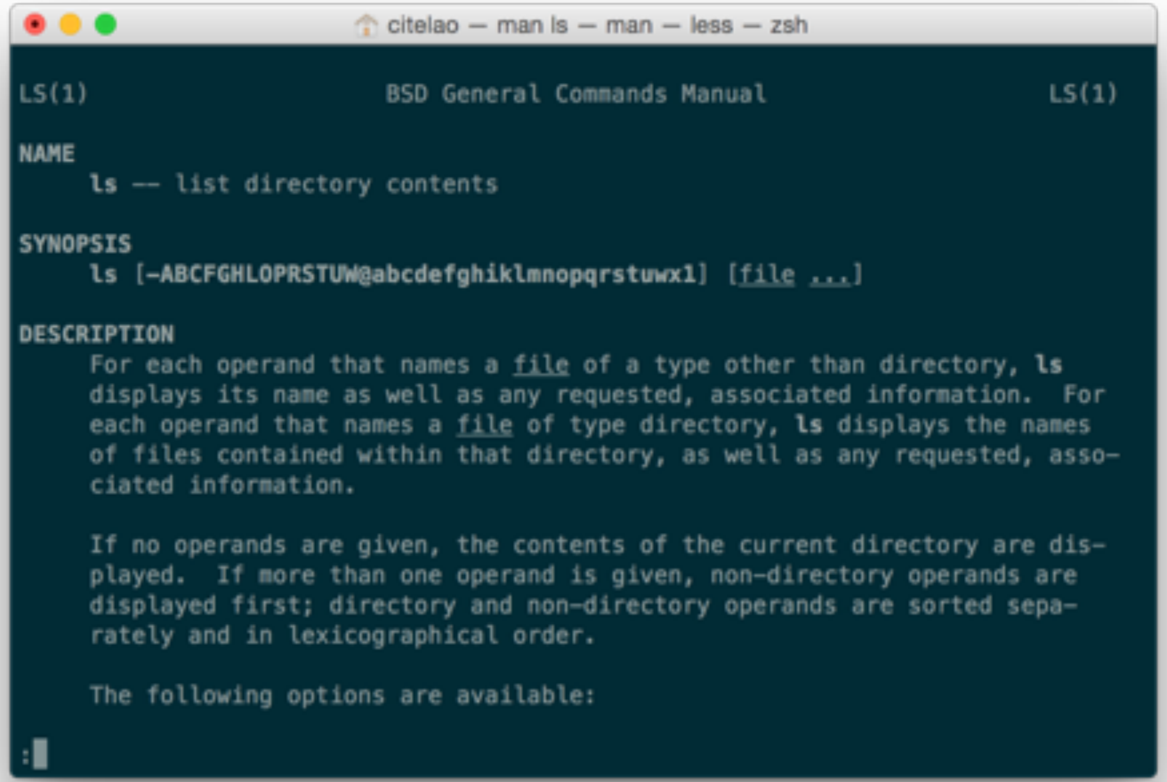
varies by command



```
(~) $ man
command
```

that’s why we have the “manual” command.

Type “man command name” and you get documentation for that command
also you could google it



```
LS(1)                     BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names
     of files contained within that directory, as well as any requested, asso-
     ciated information.

     If no operands are given, the contents of the current directory are dis-
     played.  If more than one operand is given, non-directory operands are
     displayed first; directory and non-directory operands are sorted sepa-
     rately and in lexicographical order.

     The following options are available:

:
```

You see that’s a lot.

`j` and `k` scroll down and up, q quits.

that’s a lot of things

that’s how you do things.
It’s overwhelming.

you learn the features you use.

most standard commands are really powerful.

that’s why I’m not gonna talk any more about them

```
highlight --syntax=sh
-O rtf
```

but first, here's some commands I've used recently

this is what I use to highlight code for this class

```
git commit -a -m
"preparing to write
sad, sad code"
```

a git commit!

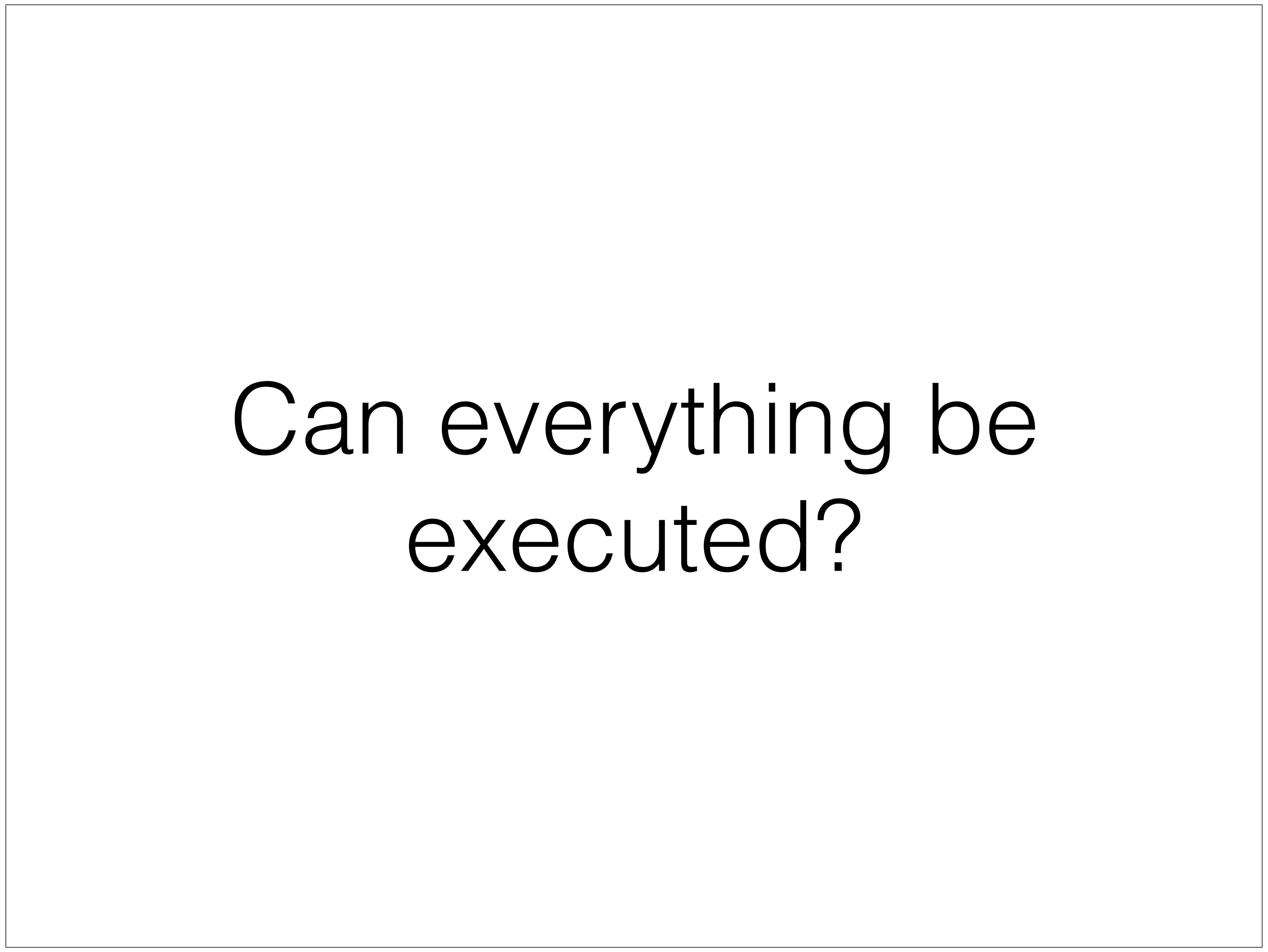
Yes, git and svn are usable via command line

```
/Applications/Arduino.app/Contents/
Resources/Java/hardware/tools/avr/bin/
avr-g++ -c -g -Os -Wall -fno-exceptions
-ffunction-sections -fdata-sections -
mmcu=atmega328p -DF_CPU=16000000L -MMD
-DUSB_VID=null -DUSB_PID=null -
DARDUINO=105 -I/Applications/
Arduino.app/Contents/Resources/Java/
hardware/arduino/cores/arduino -I/
Applications/Arduino.app/Contents/
Resources/Java/hardware/arduino/
variants/standard /var/folders/7b/
308r51sn3wnd56f88mm71_w40000gn/T/
build8876472275854087584.tmp/
sketch_feb20a.cpp -o /var/folders/7b/
308r51sn3wnd56f88mm71_w40000gn/T/
build8876472275854087584.tmp/
sketch_feb20a.cpp.o
```

I was testing Arduino stuff

```
kill -9 25793
```

and murdering processes.



Can everything be executed?

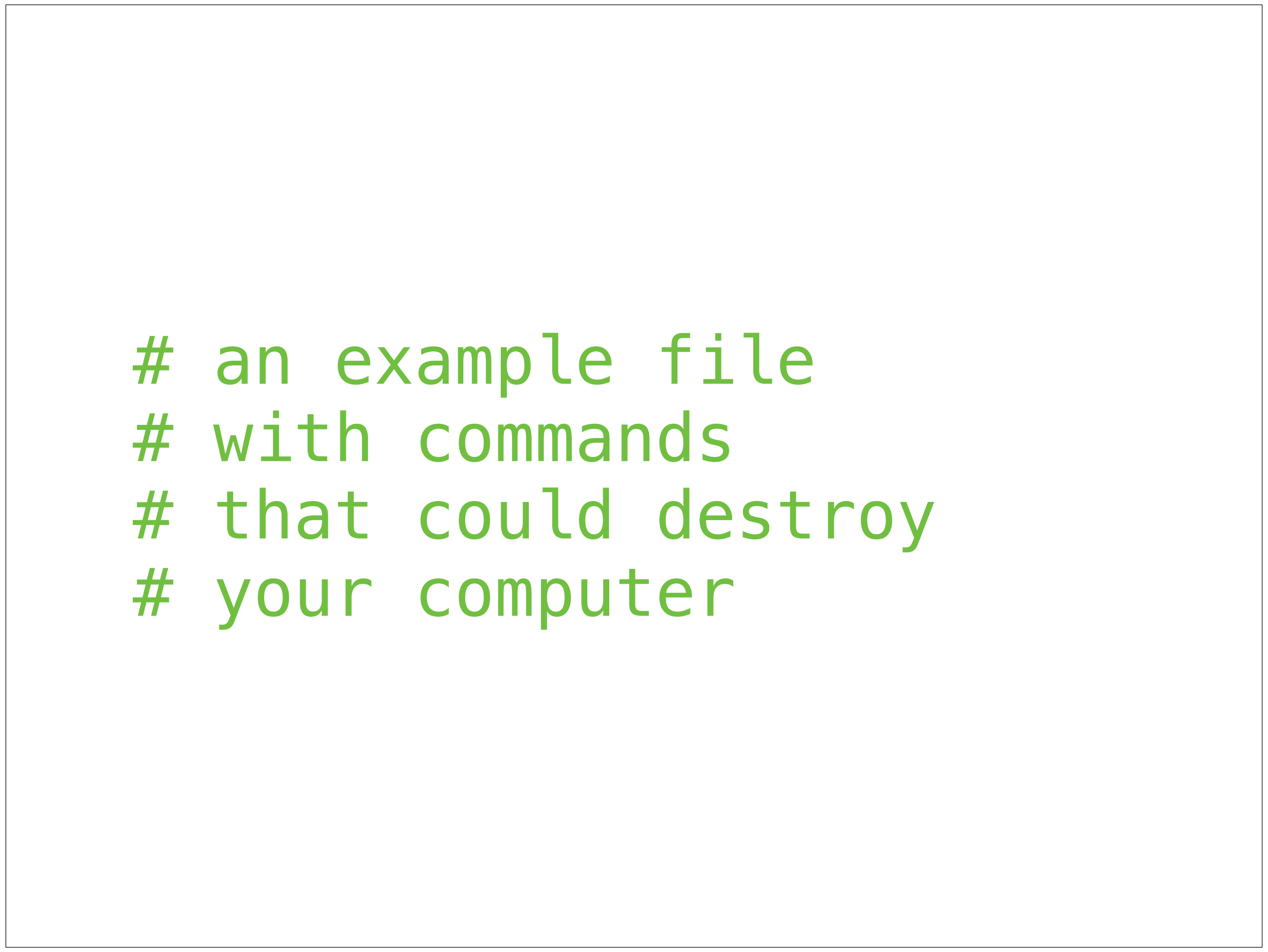
everything is a file, and you can execute files from the shell.

Can you execute every file?

Is that even reasonable?

```
# this is a
comment
```

this is a shell script comment. like //, etc



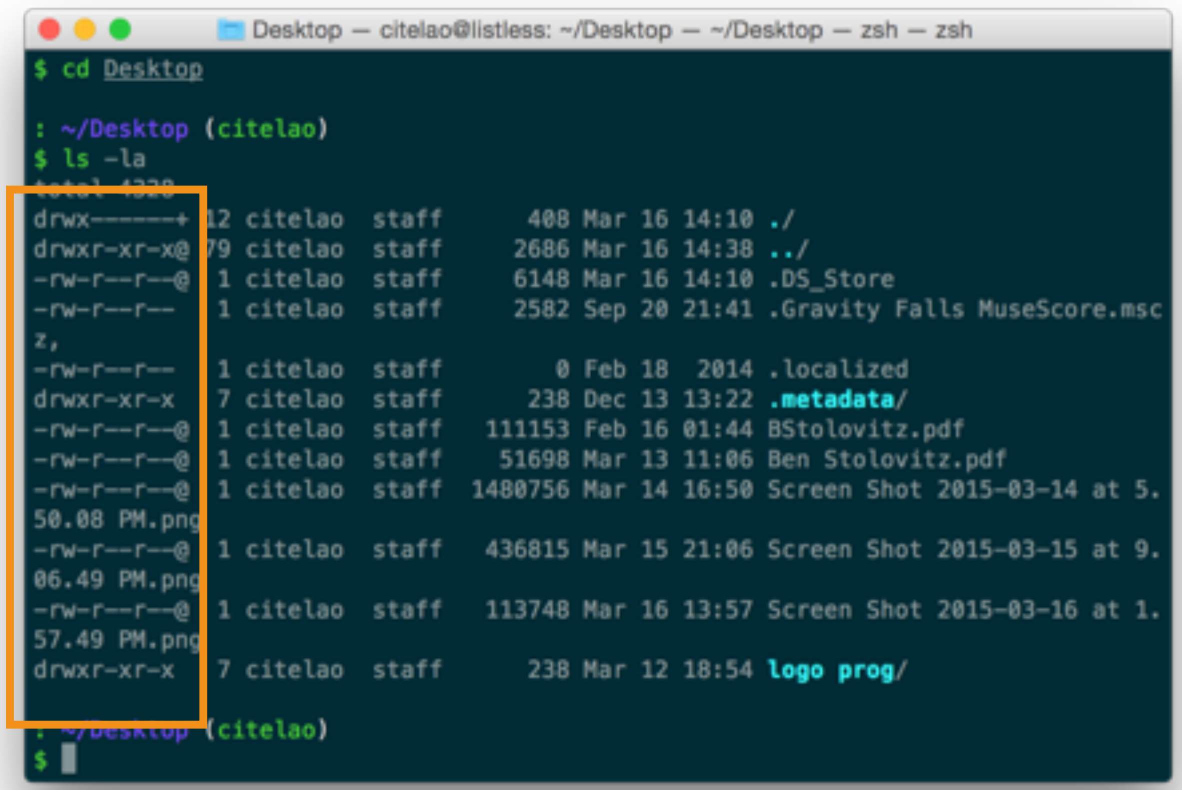
```
# an example file
# with commands
# that could destroy
# your computer
```

what if I have an example file of what not to do

these commands exist, but I am too scared of them to type them. rm with the right flags removes things, dd writes to a disk

Permissions!

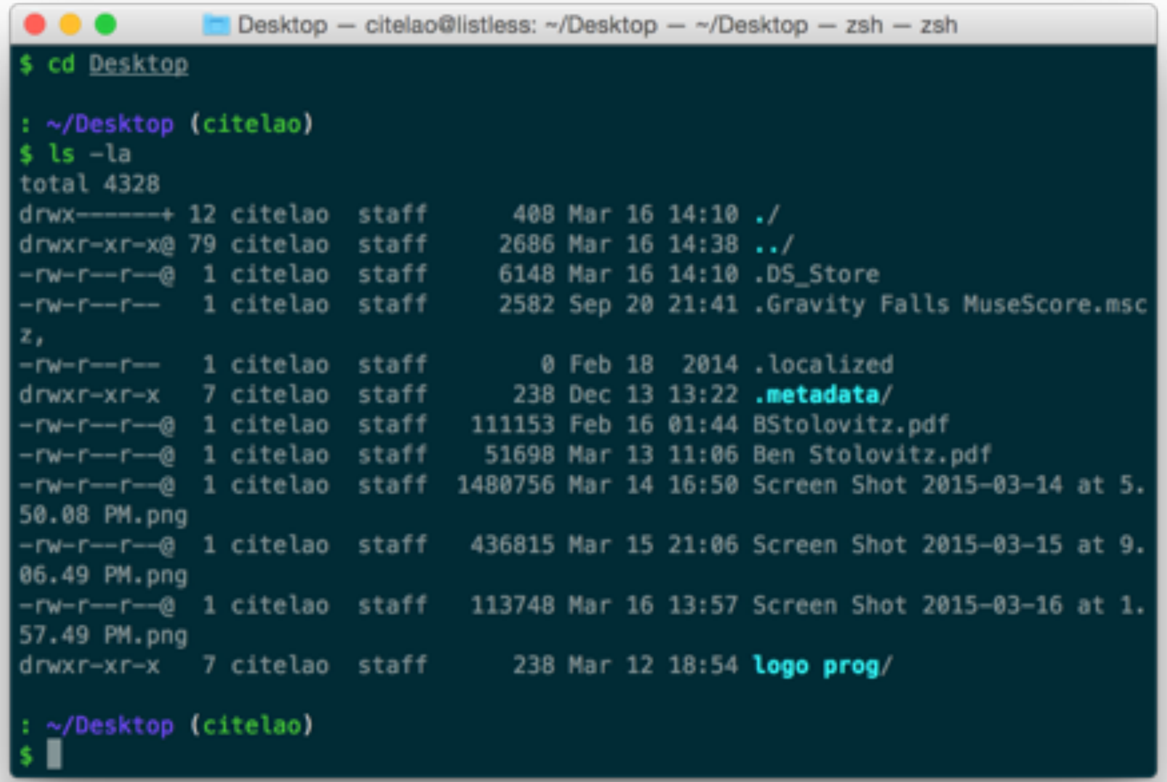
thank Unix, then, for file permissions



```
Desktop — citelao@listless: ~/Desktop — ~/Desktop — zsh — zsh
$ cd Desktop

: ~/Desktop (citelao)
$ ls -la
total 4320
drwx------+ 12 citelao  staff      408 Mar 16 14:10 ./
drwxr-xr-x@ 79 citelao  staff     2686 Mar 16 14:38 ../
-rw-r--r--@  1 citelao  staff     6148 Mar 16 14:10 .DS_Store
-rw-r--r--   1 citelao  staff     2582 Sep 20 21:41 .Gravity Falls MuseScore.msc
z,
-rw-r--r--   1 citelao  staff        0 Feb 18  2014 .localized
drwxr-xr-x   7 citelao  staff      238 Dec 13 13:22 .metadata/
-rw-r--r--@  1 citelao  staff   111153 Feb 16 01:44 BStolovitz.pdf
-rw-r--r--@  1 citelao  staff    51698 Mar 13 11:06 Ben Stolovitz.pdf
-rw-r--r--@  1 citelao  staff  1480756 Mar 14 16:50 Screen Shot 2015-03-14 at 5.
50.08 PM.png
-rw-r--r--@  1 citelao  staff   436815 Mar 15 21:06 Screen Shot 2015-03-15 at 9.
06.49 PM.png
-rw-r--r--@  1 citelao  staff   113748 Mar 16 13:57 Screen Shot 2015-03-16 at 1.
57.49 PM.png
drwxr-xr-x   7 citelao  staff      238 Mar 12 18:54 logo prog/

: ~/Desktop (citelao)
$ 
```

these are permissions

Read (R)
Write (W)
Execute (X)



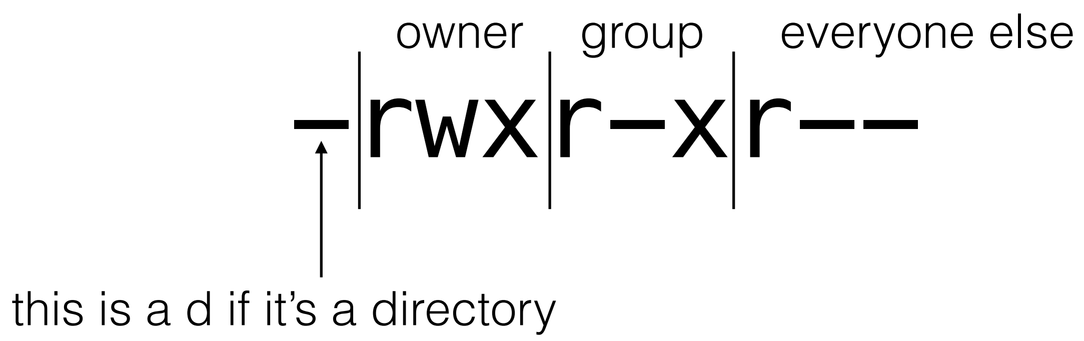
Desktop — citelao@listless: ~/Desktop — ~/Desktop — zsh — zsh

```
$ cd Desktop

: ~/Desktop (citelao)
$ ls -la
total 4328
drwx------+ 12 citelao  staff      408 Mar 16 14:10 ./
drwxr-xr-x@ 79 citelao  staff     2686 Mar 16 14:38 ../
-rw-r--r--@  1 citelao  staff     6148 Mar 16 14:10 .DS_Store
-rw-r--r--   1 citelao  staff     2582 Sep 20 21:41 .Gravity Falls MuseScore.msc
z,
-rw-r--r--   1 citelao  staff        0 Feb 18  2014 .localized
drwxr-xr-x   7 citelao  staff      238 Dec 13 13:22 .metadata/
-rw-r--r--@  1 citelao  staff   111153 Feb 16 01:44 BStolovitz.pdf
-rw-r--r--@  1 citelao  staff    51698 Mar 13 11:06 Ben Stolovitz.pdf
-rw-r--r--@  1 citelao  staff  1480756 Mar 14 16:50 Screen Shot 2015-03-14 at 5.
50.08 PM.png
-rw-r--r--@  1 citelao  staff   436815 Mar 15 21:06 Screen Shot 2015-03-15 at 9.
06.49 PM.png
-rw-r--r--@  1 citelao  staff   113748 Mar 16 13:57 Screen Shot 2015-03-16 at 1.
57.49 PM.png
drwxr-xr-x   7 citelao  staff      238 Mar 12 18:54 logo prog/

: ~/Desktop (citelao)
$
```

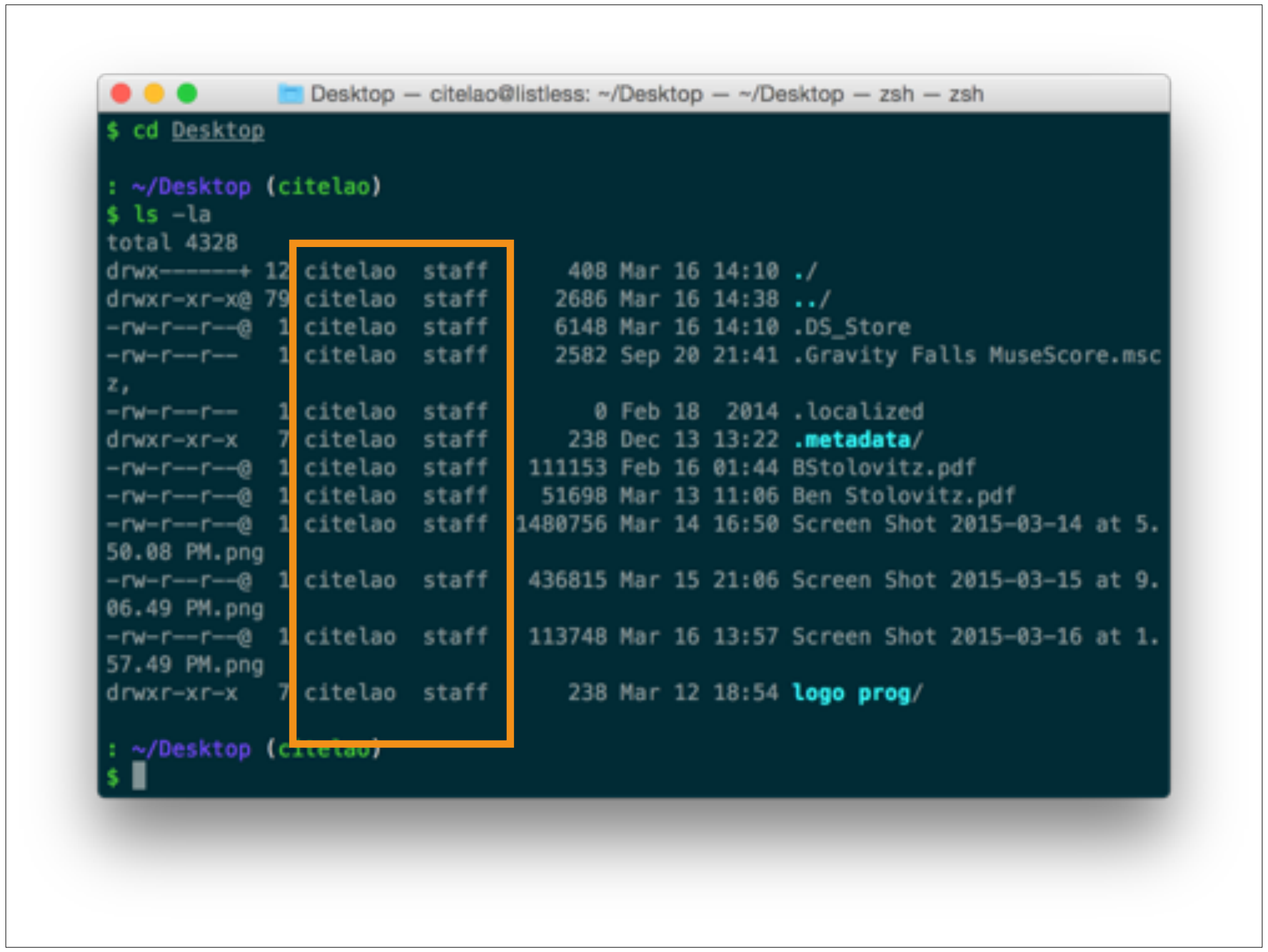
a dash means nothing, symbol means you can

```
-rwxr-xr--
```

what’s the mess though


owner
group
everyone else
-rwxr-xr--
this is a d if it’s a directory

who owns it?



```
Desktop — citelao@listless: ~/Desktop — ~/Desktop — zsh — zsh
$ cd Desktop

: ~/Desktop (citelao)
$ ls -la
total 4328
drwx------+ 12 citelao  staff     408 Mar 16 14:10 ./
drwxr-xr-x@ 79 citelao  staff    2686 Mar 16 14:38 ../
-rw-r--r--@  1 citelao  staff    6148 Mar 16 14:10 .DS_Store
-rw-r--r--   1 citelao  staff    2582 Sep 20 21:41 .Gravity Falls MuseScore.msc
z,
-rw-r--r--   1 citelao  staff       0 Feb 18  2014 .localized
drwxr-xr-x   7 citelao  staff     238 Dec 13 13:22 .metadata/
-rw-r--r--@  1 citelao  staff  111153 Feb 16 01:44 BStolovitz.pdf
-rw-r--r--@  1 citelao  staff   51698 Mar 13 11:06 Ben Stolovitz.pdf
-rw-r--r--@  1 citelao  staff 1480756 Mar 14 16:50 Screen Shot 2015-03-14 at 5.
50.08 PM.png
-rw-r--r--@  1 citelao  staff  436815 Mar 15 21:06 Screen Shot 2015-03-15 at 9.
06.49 PM.png
-rw-r--r--@  1 citelao  staff  113748 Mar 16 13:57 Screen Shot 2015-03-16 at 1.
57.49 PM.png
drwxr-xr-x   7 citelao  staff     238 Mar 12 18:54 logo prog/

: ~/Desktop (citelao)
$ 
```

a dash means nothing, symbol means you can

`chown` & `chmod`

this is how you change that

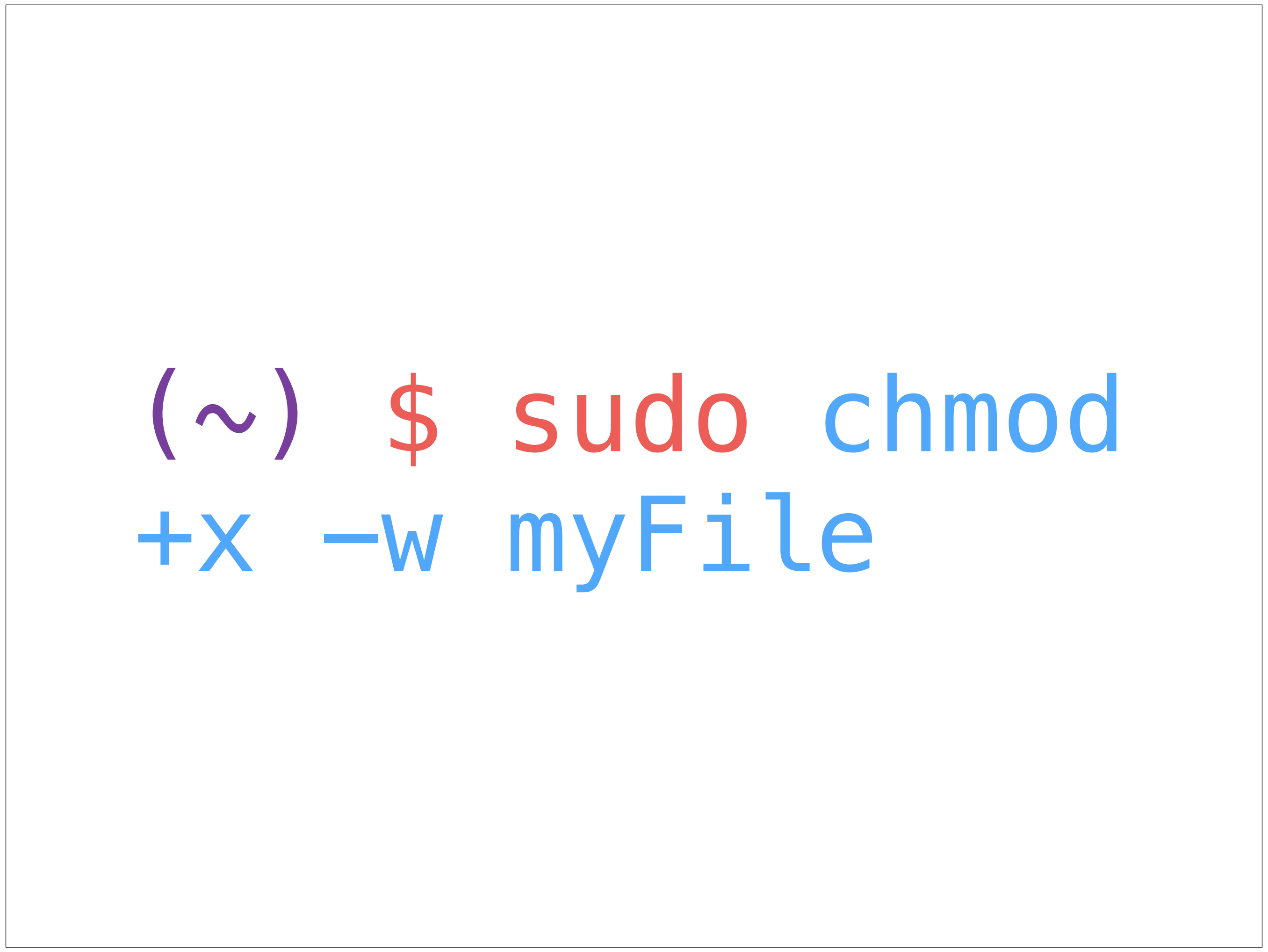
```
(~) $ chmod +x -w myFile
```

syntax kinda looks like this.

+ is the opposite of what you mean; disables

You can’t do this
unless you own it

Unless you're the admin

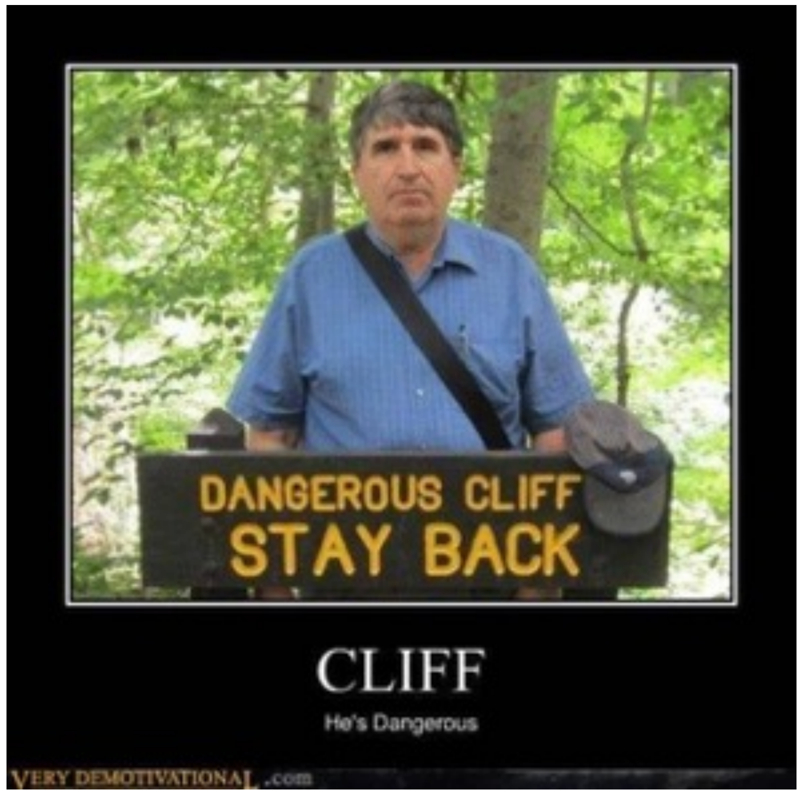


```
(~) $ sudo chmod
+x -w myFile
```

SUDO

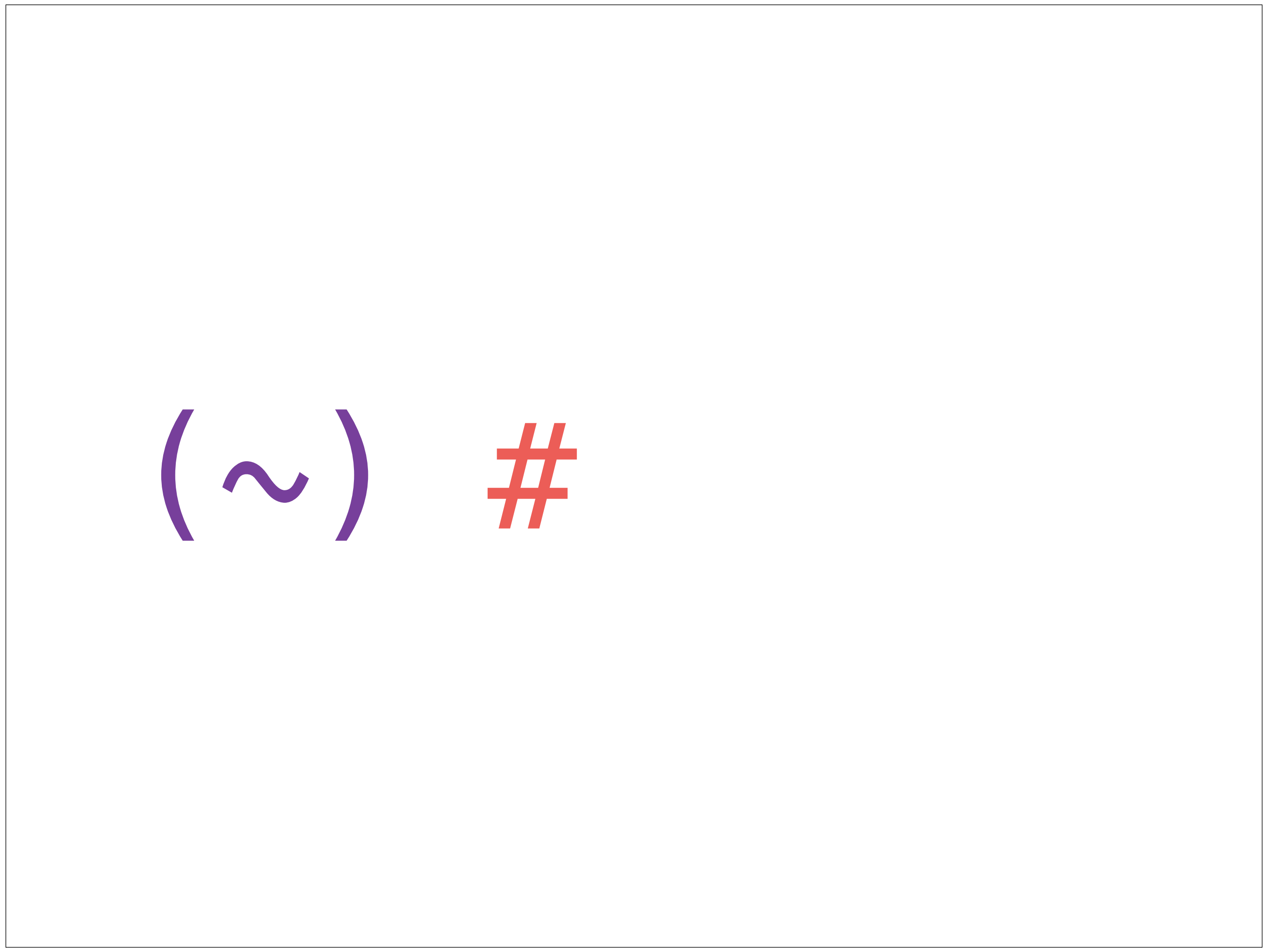
super-user do

you need your password for this, runs the command as the administrator (or another specific user, with the right args)



DANGEROUS

you can break things



```
(~)  #
```

if you’re logged in as super user, the prompt generally changes to look like this

no $. # instead.

because unix loves you.

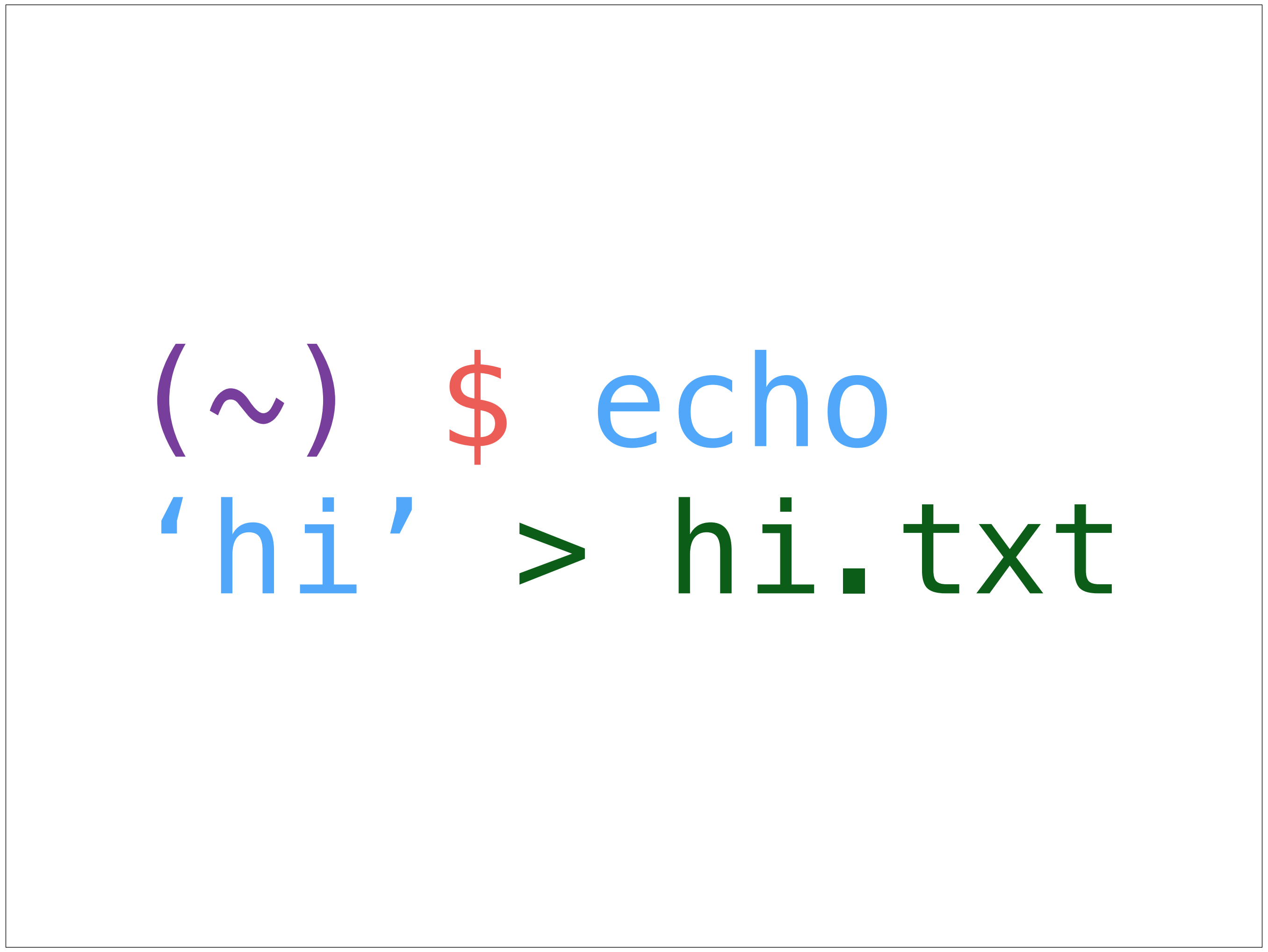
How do I do things?

like make files and read them?

```
(~) $ cmd > outputFile
```

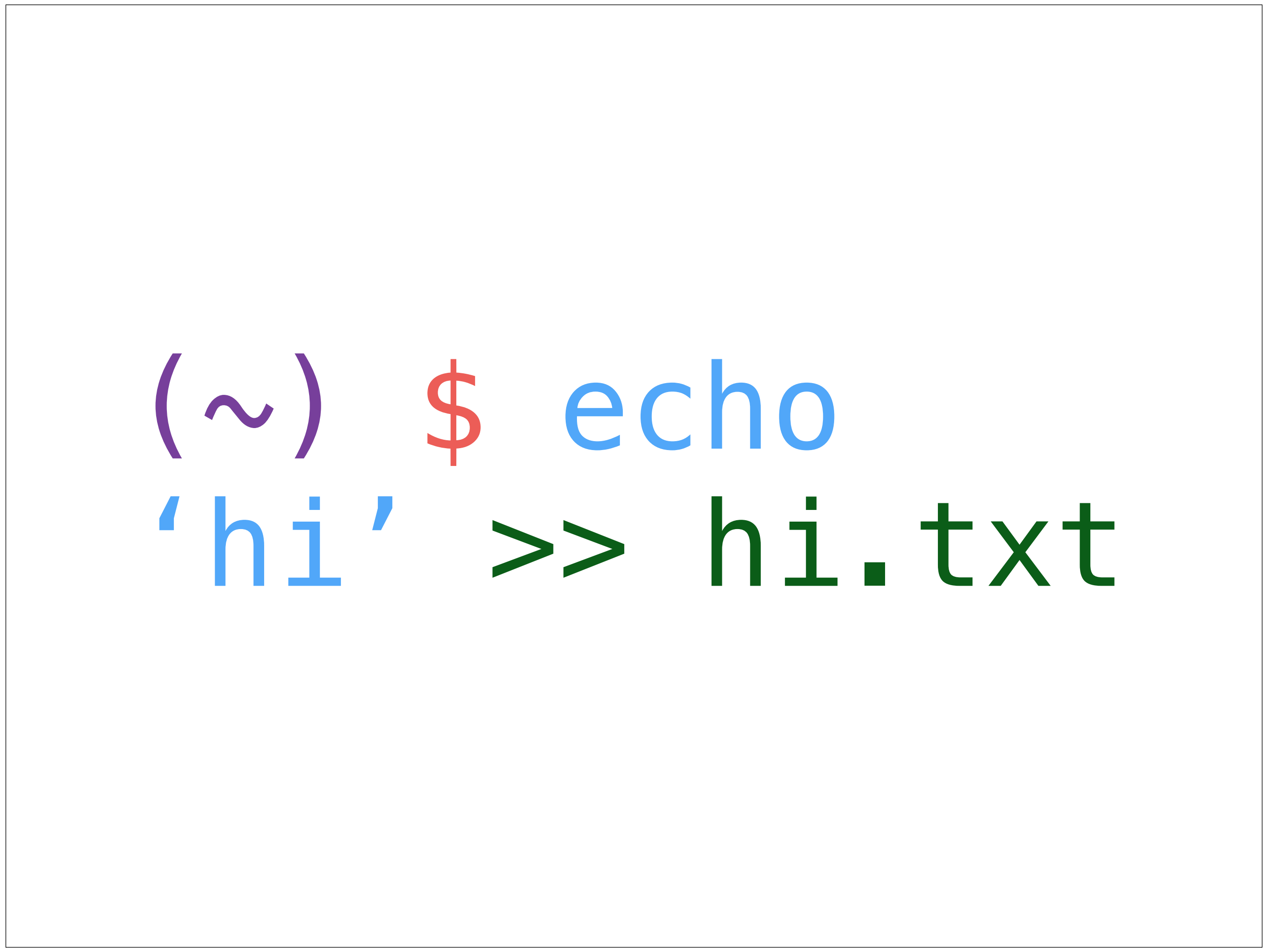
the bracket redirects output

you can redirect errors and input too, but that’s a little out of scope



```
(~) $ echo
'hi' > hi.txt
```

creates a new file



```
(~) $ echo
'hi' >> hi.txt
```

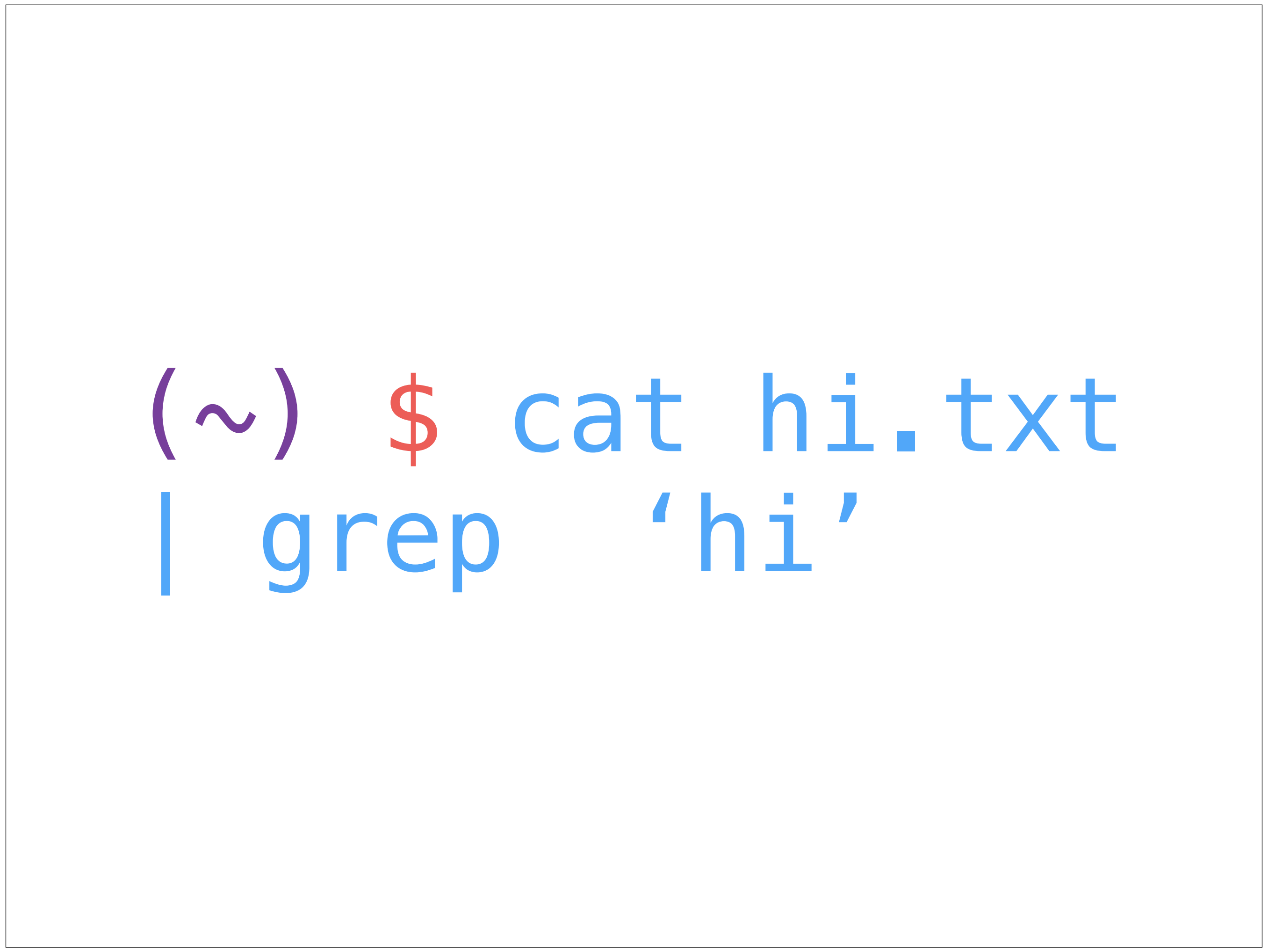
appends to existing

```
(~) $ cat
hi.txt
```

read a files contents out

```
(~) $ cat hi.txt
> new.txt
```

this can be done on any command.

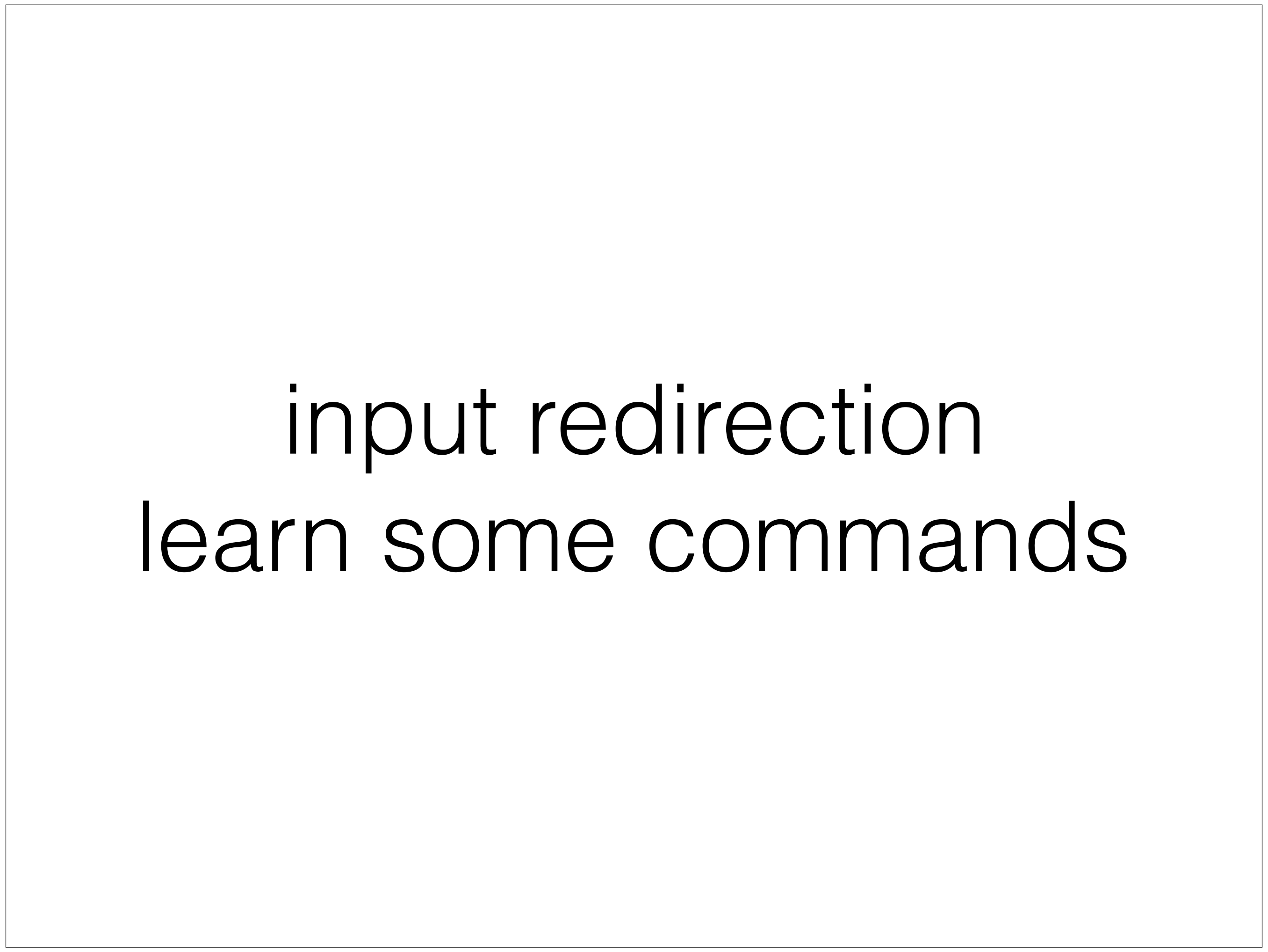


```
(~) $ cat hi.txt
| grep  ‘hi’
```

you can also pipe from one command to another, as if the first command is a file input to the next.

Really, really powerful

We'll talk about package managers, compilers, and other stuff later



input redirection
learn some commands

if you want to succeed at command line, learn this

- permissions
- ownership & users
- piping & redirection
- ctrl codes