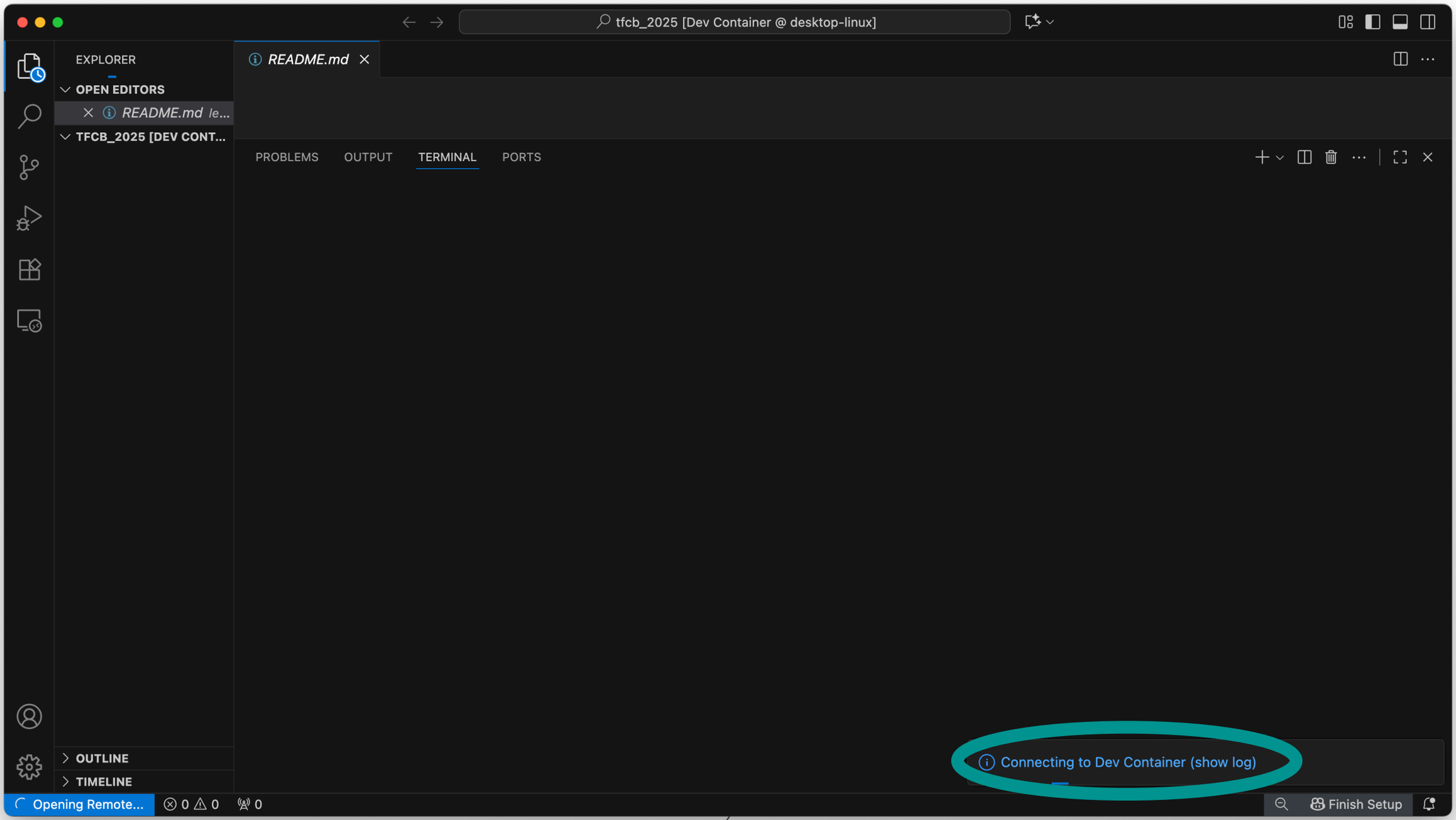
MCB 536: Tools for Computational Biology
Lecture 04: Intro to Command Line

October 07, 2025

Melody Campbell, Fred Hutch

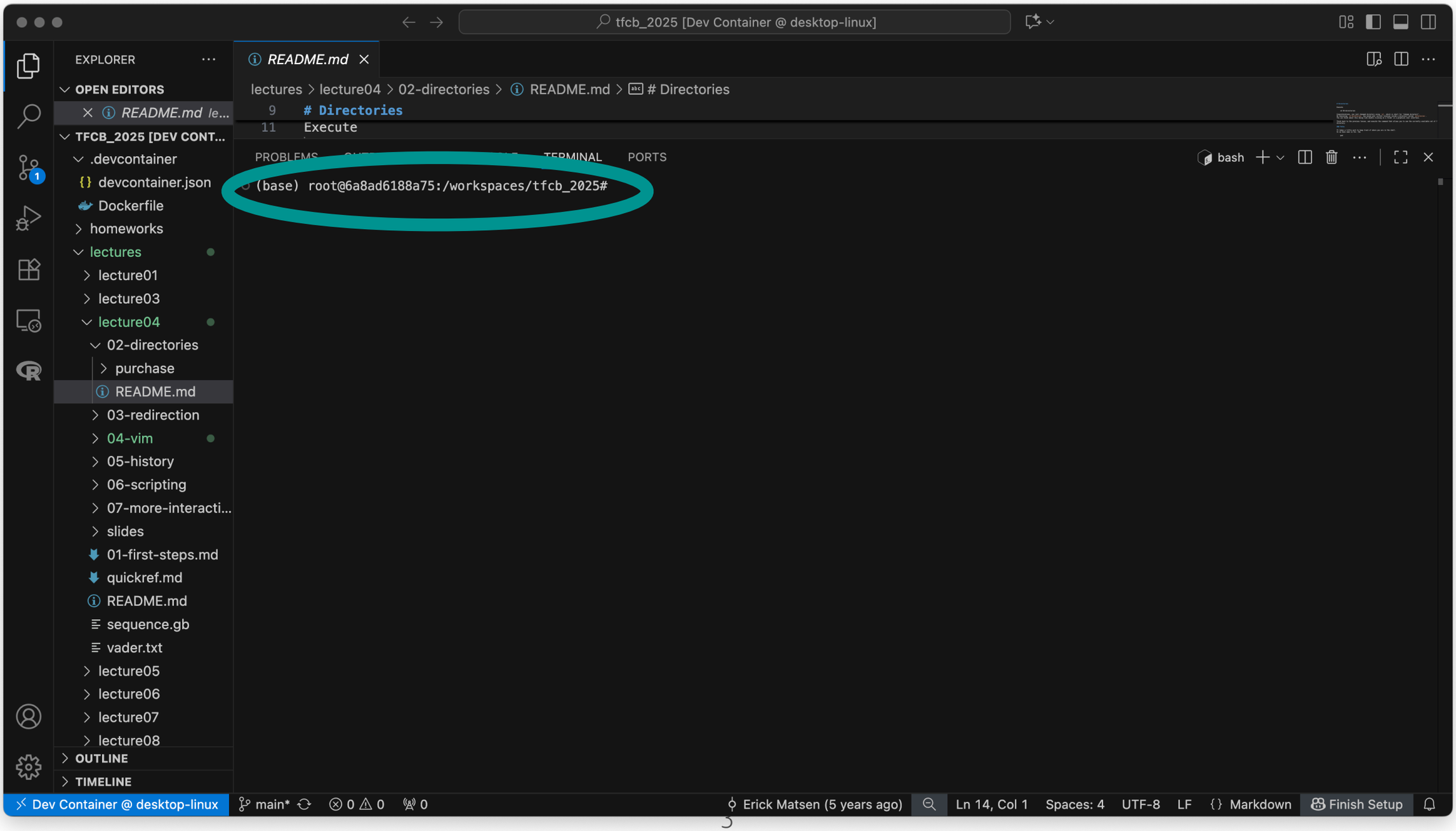
What can you do with a terminal?

- Open Visual Studio Code
- Connect to Dev Container
- Go to Terminal



What can you do with a terminal?

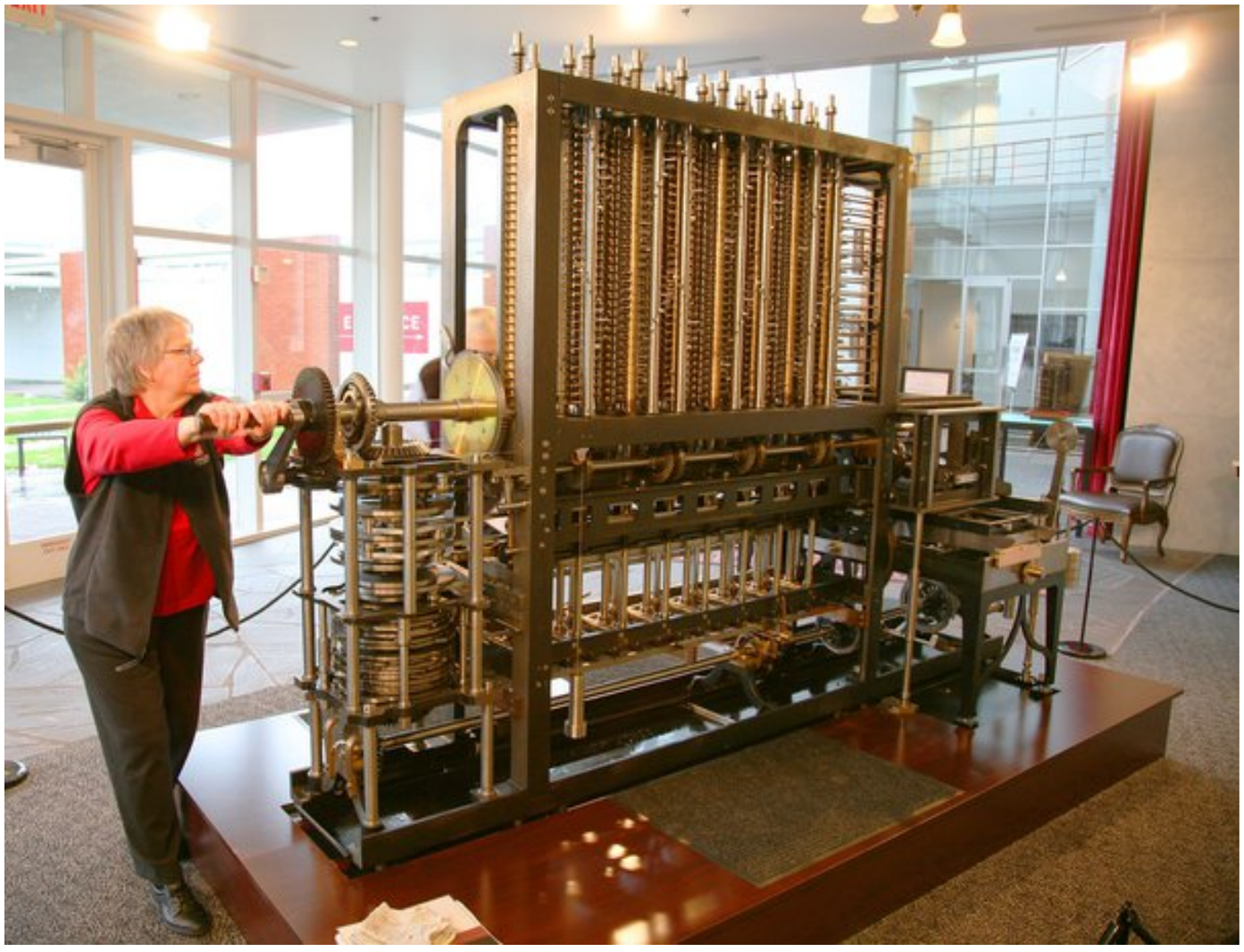
- Open Visual Studio Code
- Connect to Dev Container
- Go to Terminal



Teaching Goals

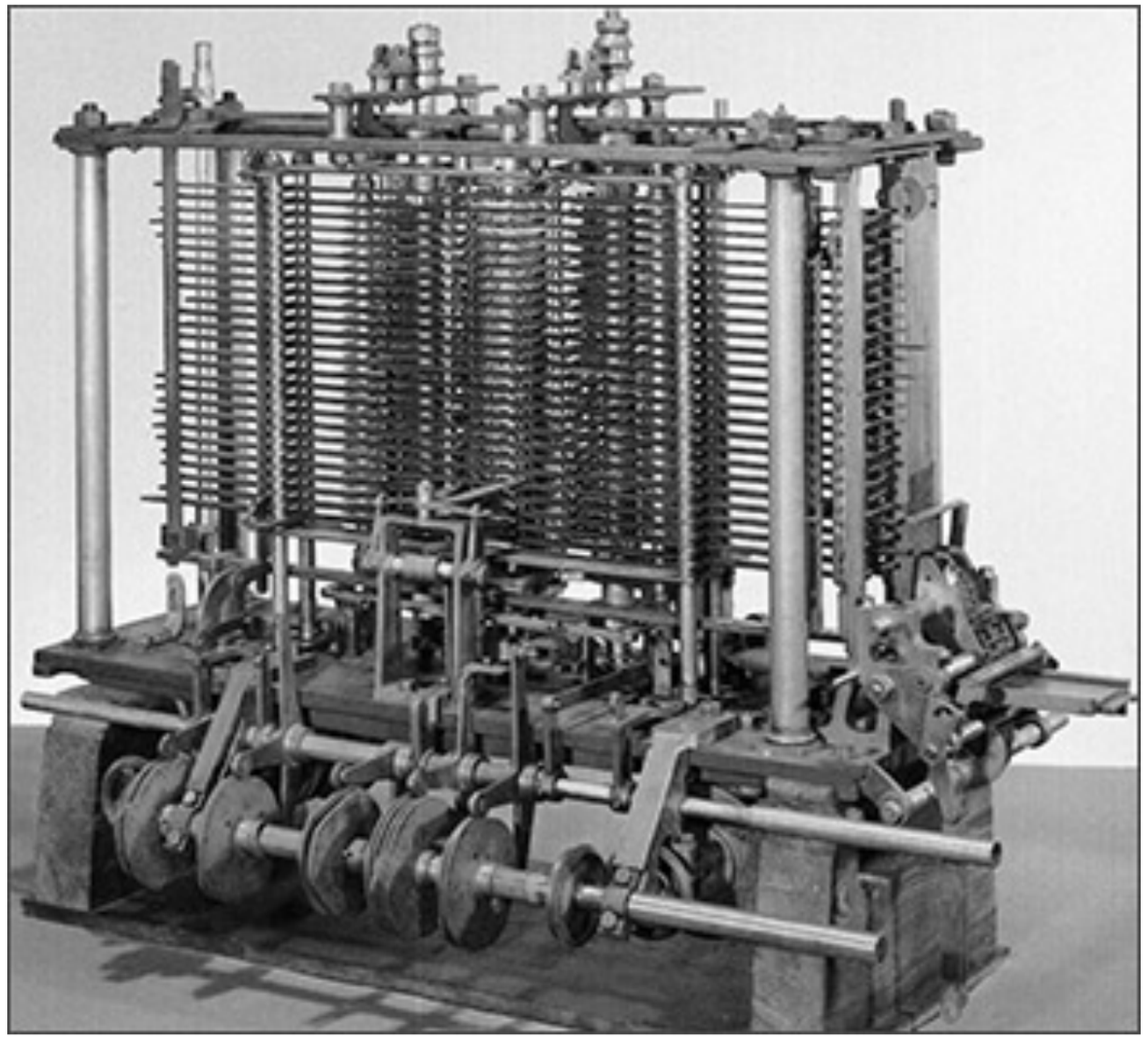
- History: how did we get to the command line?
- Interacting with the command line
  - Commands
  - Flags
  - Wildcards
  - Variables
  - Outputs
  - Autocomplete
  - Terminate

The Interface btwn Human & Computer



Early Interfaces

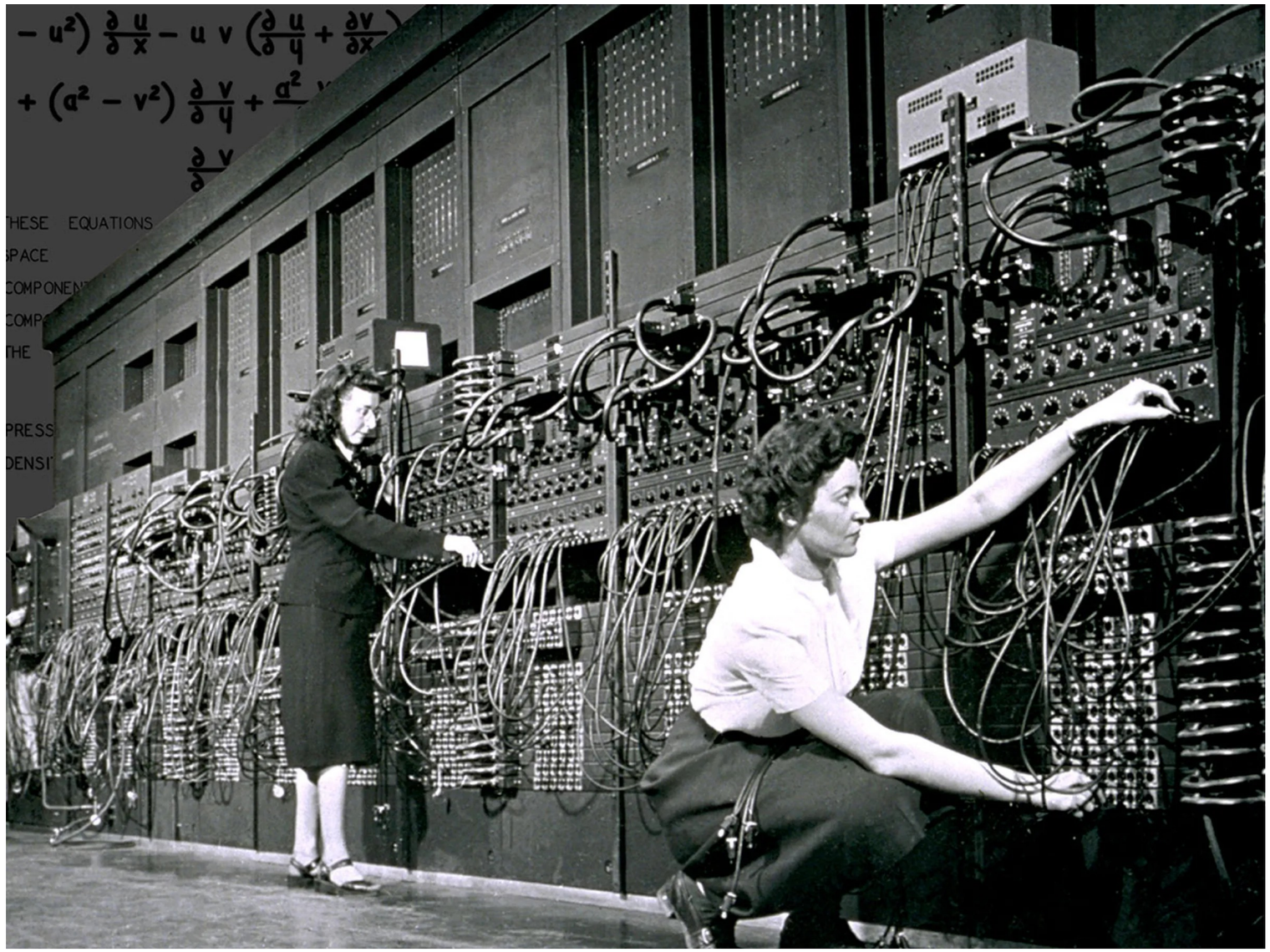
- Use physical controls for inputs and outputs: gears, knobs, switches



1837: Analytical Engine

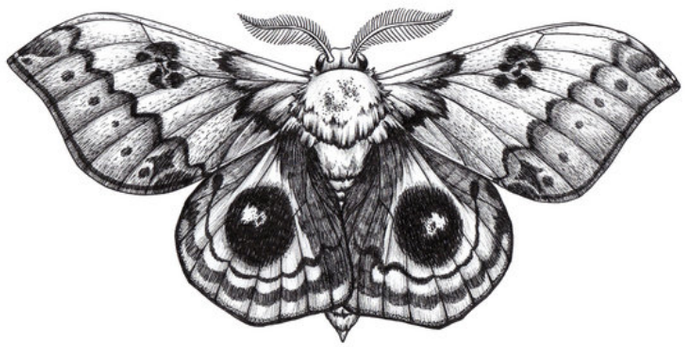
First Electronic Computers

- Still relied on mechanical controls and patch wires
- Took weeks to input a program



1945: ENIAC

Debugging



First Electronic Computers

- Took weeks to run a program
- Output is paper

Early 1950's: Updated Inputs / Outputs

- Inputs: Punch cards and magnetic tape replace mechanical
- Outputs: Still paper

Log files were lights!

Optimized for Computers (not Humans)

- Rolls of punch tape easy to read
  - continuous
  - holes can be read optically or mechanically

also, not interactive!

Programmers have to convert their language and programs to a format that is easy for computers to understand

Late1950's: Back & Forth

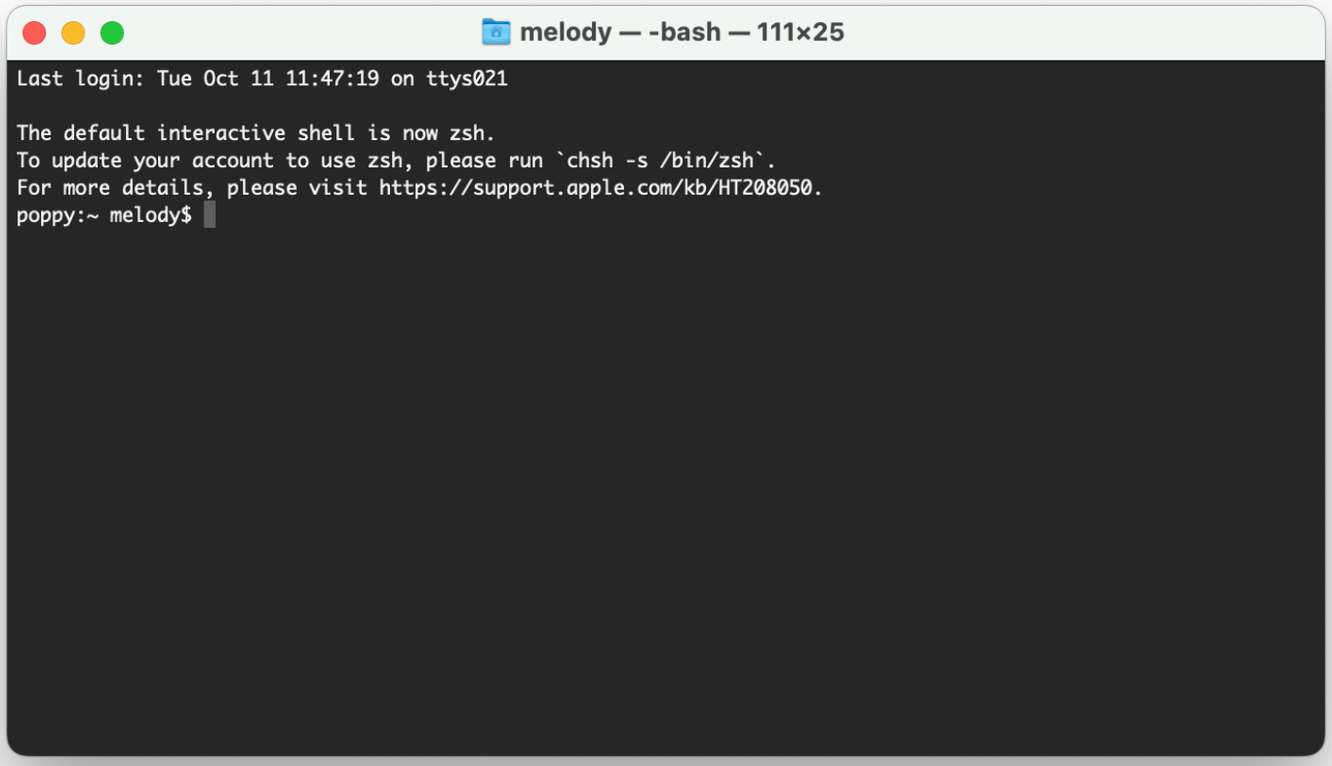
- Computers get smaller & more sophisticated
- Multitasking and Timesharing Systems
- Start using keyboards!

60’s & 70’s: Electronic Teletype

- A electro-mechanically augmented typewriter used for telegraphs: one person types, electronic signal transferred via telegraph line, transmitted to another teletype machine
  - This was easy to adapt to computers: user types, computer ‘types’ back

Command Line Interface

- ^ The name for the adapted teletype human/computer interface
- This was the most prevalent form of human/computer interaction until the 1980's
  - Screens became cheaper in the 1970's

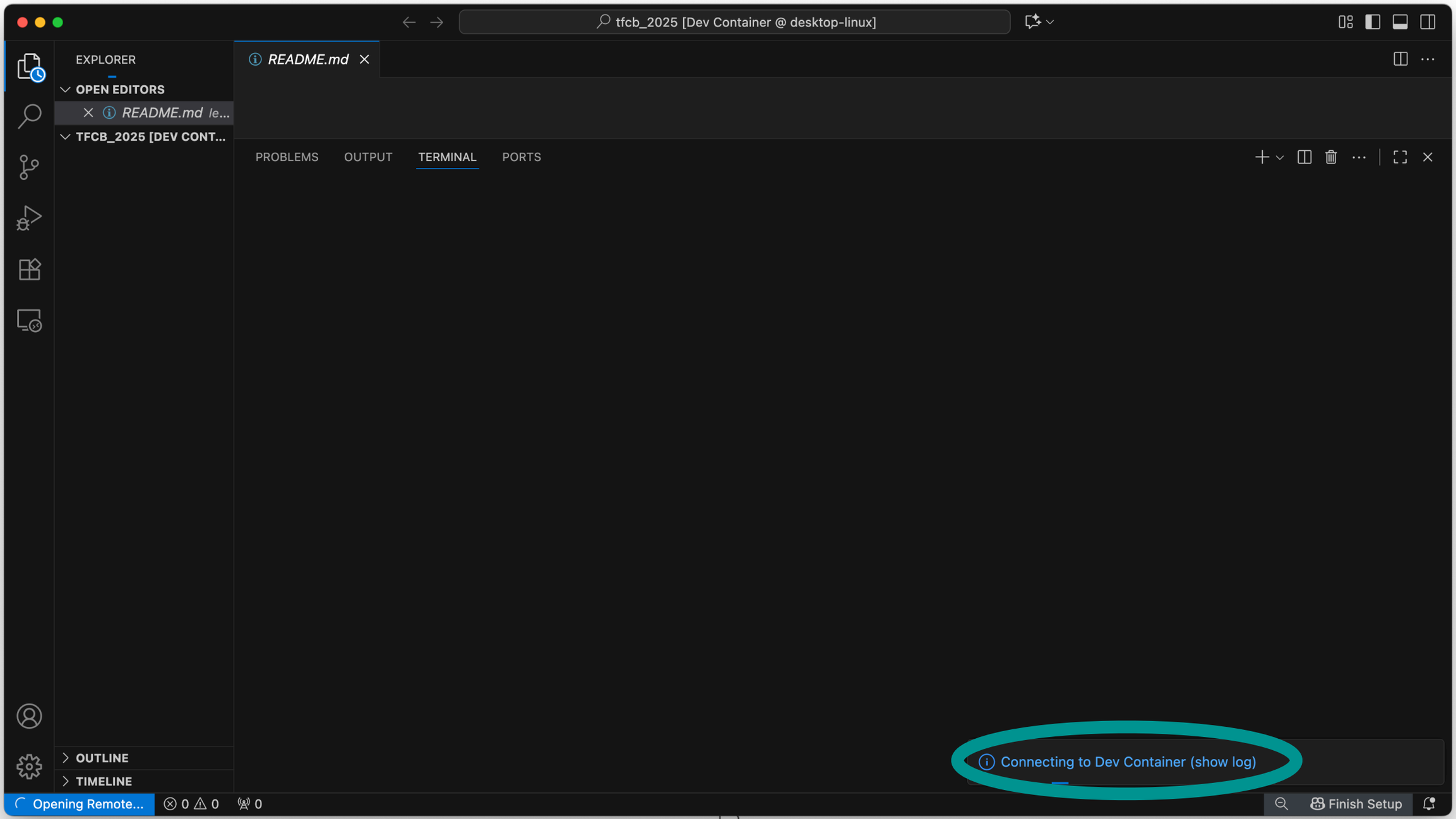


Add a Screen -> Terminal

- We have screens (infinite paper), we still use the ‘electronic teletype procedures’
  - This way engineers didn’t have to design a whole new way for humans to interact with computers
    - text in > text out
- Known as “Glass teletype machines”
- Now called terminals

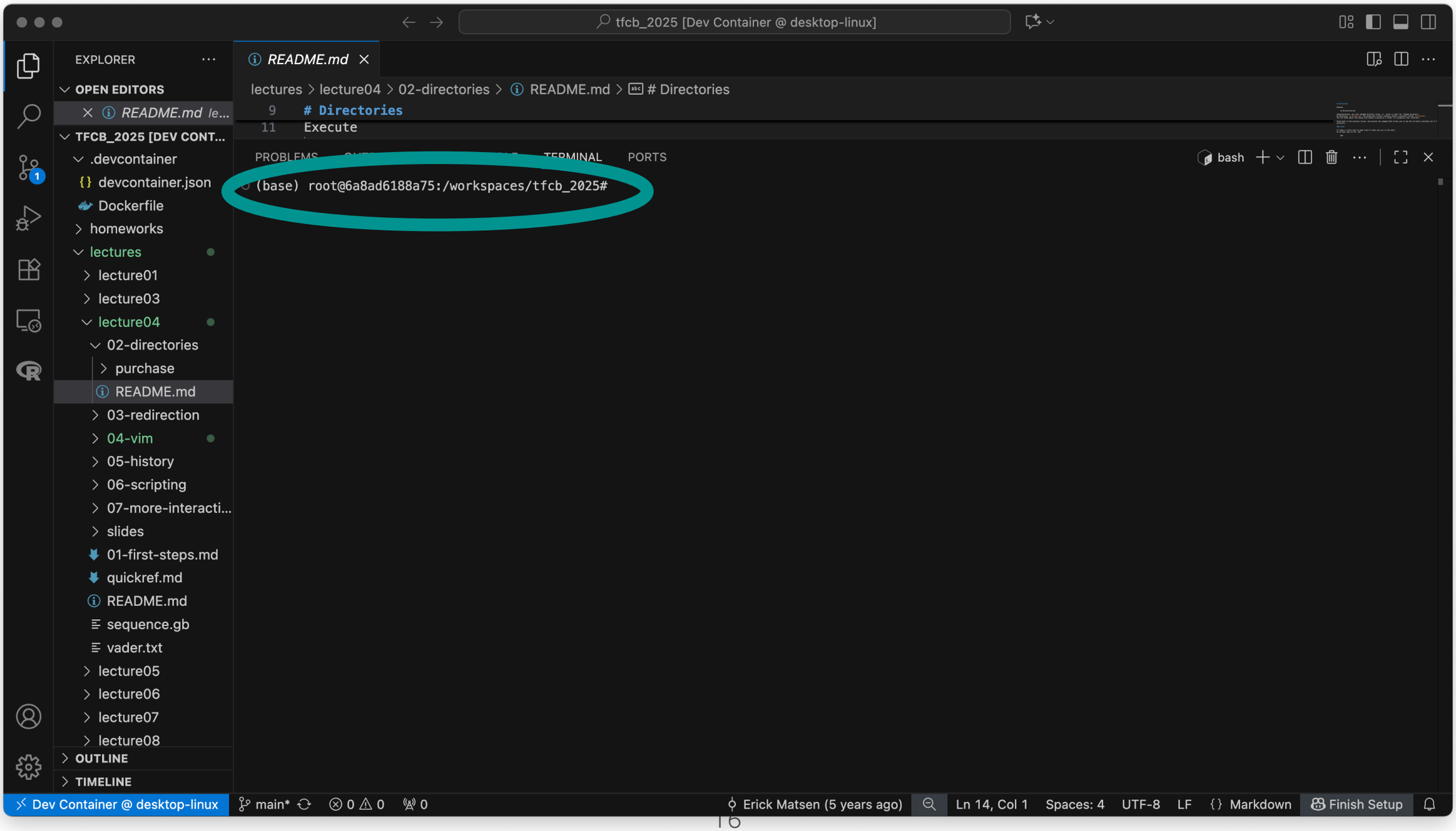
What can you do with a terminal?

- Open Visual Studio Code
- Connect to Dev Container
- Go to Terminal



What can you do with a terminal?

- Open Visual Studio Code
- Connect to Dev Container
- Go to Terminal



pwd

- pwd: print working directory
  - aka where am i?
  - what is a directory?
    - it's just a folder
  - what is a working directory?
    - it's the folder you're in
- if you're using vs code you're here (probably):
  - /workspaces
- if you're in a mac terminal or on rhino you are probably here:
  - /users/$USER/(the name of the folder (directory) you made when you set up VS)
- what does $USER mean?
  - try echo $USER
    - what does echo mean?
      - … what is a variable??
        - what is a command??????

Optimized for Computers (not Humans)

- Rolls of punch tape easy to read
  - continuous
  - holes can be read optically or mechanically

also, not interactive!

Programmers have to convert their language and programs to a format that is easy for computers to understand

You are learning a new language.

Translation: English to Linux

What folder am i in? → pwd

Which user am i logged in as? → echo $USER

Why does this only sometimes work? Slightly different vocabularies…

What is my home directory? → echo $HOME

Different Dialects!

Which user am i logged in as? → echo $USER

Why does this only sometimes work? Slightly different vocabularies…

- Different computers/systems will have different default "language knowledge" (these are not real terms)
- Usually, you can get the info you want. Try instead:
  - whoami
- This should do the same as
  - echo $USER
- Note: echo just repeats what you say. Try with a normal work
  - echo pineapple

pwd

- pwd: print working directory
  - aka where am i?
  - what is a directory?
    - it's just a folder
  - what is a working directory?
    - it's the folder you're in
- if you're using vs code you're here:
  - it depends on how you set up the container, maybe /root
- if you're in a mac terminal you are probably here:
  - /users/$USER/(the name of the folder (directory) you made when you set up VS)
- what does $USER mean?
  - try echo $USER
    - what does echo mean?
      - … what is a variable??
        - what is a command??????

Let’s jump in first & explain later

- This is a reasonable way to approach coding: just try stuff!
- It’s very difficult to break things
  - mash the keyboard: what happens?

Let’s jump in first & explain later

- This is a reasonable way to approach coding
- It’s very difficult to break things
  - mash the keyboard: what happens?

```
poppy:tfcb_2022 melody$
poppy:tfcb_2022 melody$ KSEJHDFISZH IS HFYW A478RYJBS jiwhey83rwy
bash: KSEJHDFISZH: command not found
```

nothing bad!

mkdir

- Figure out where you are: pwd
- If you are not in /workspaces/tfcb_2025
- cd
  - change directory
  - this is what lets you move in and out of folders (like double clicking on a normal computer)
    - cd /workspaces
    - pwd
- mkdir
  - make directory
  - this makes a new folder (like command + shift + N on a mac or file>new folder)
  - make a folder called in_class
    - mkdir tfcb_2025_in_class

Note: I'm not exactly sure how standardized everyone set up their vs code, so if you can't get there, just get into a directory you can write into

Writing Files with >

- cd tfcb_2025_in_class
- Let's make three simple text files 1,2,3 with a single word:
  - echo hello01 >file01.txt
    - The arrow > takes the text "hello01" and outputs it as a file called file01.txt
  - echo hello02 >file02.txt
  - echo hello03 >file03.txt
    - It is annoying to repeat this command 3 times, but we will learn a better way soon…

The Up Arrow

- Recalls the most recent command
- Now use the side arrows to navigate (the mouse won't let you move the cursor)
  - echo hello02 >file02.txt
  - echo hello03 >file03.txt

Overwriting vs Adding: > vs >>

- Be careful! > will overwrite any current file!
- If you want to *add* you need two: > >
  - echo hello04 >file04.txt
  - echo goodbye04 >>file04.txt

cat

- How do we check what is written in each file?
- cat
  - concatenate
    - try: cat file01.txt
    - cat file02.txt
    - cat file04.txt
      - tab complete to make your life easier!

*do example

Commands

- When you type something into a terminal, the computer assumes it is a command
  - try typing your name
- Commands you've used so far: pwd, whoami, cd, mkdir, echo, cat

Argument

- "A value that is passed between programs, subroutines, or functions"
    - cat file01.txt
  - If cat is a verb, than file01.txt is a noun
  - You can have multiple arguments (this depends on the command)
    - cat file01.txt file02.txt

* Wild Cards!

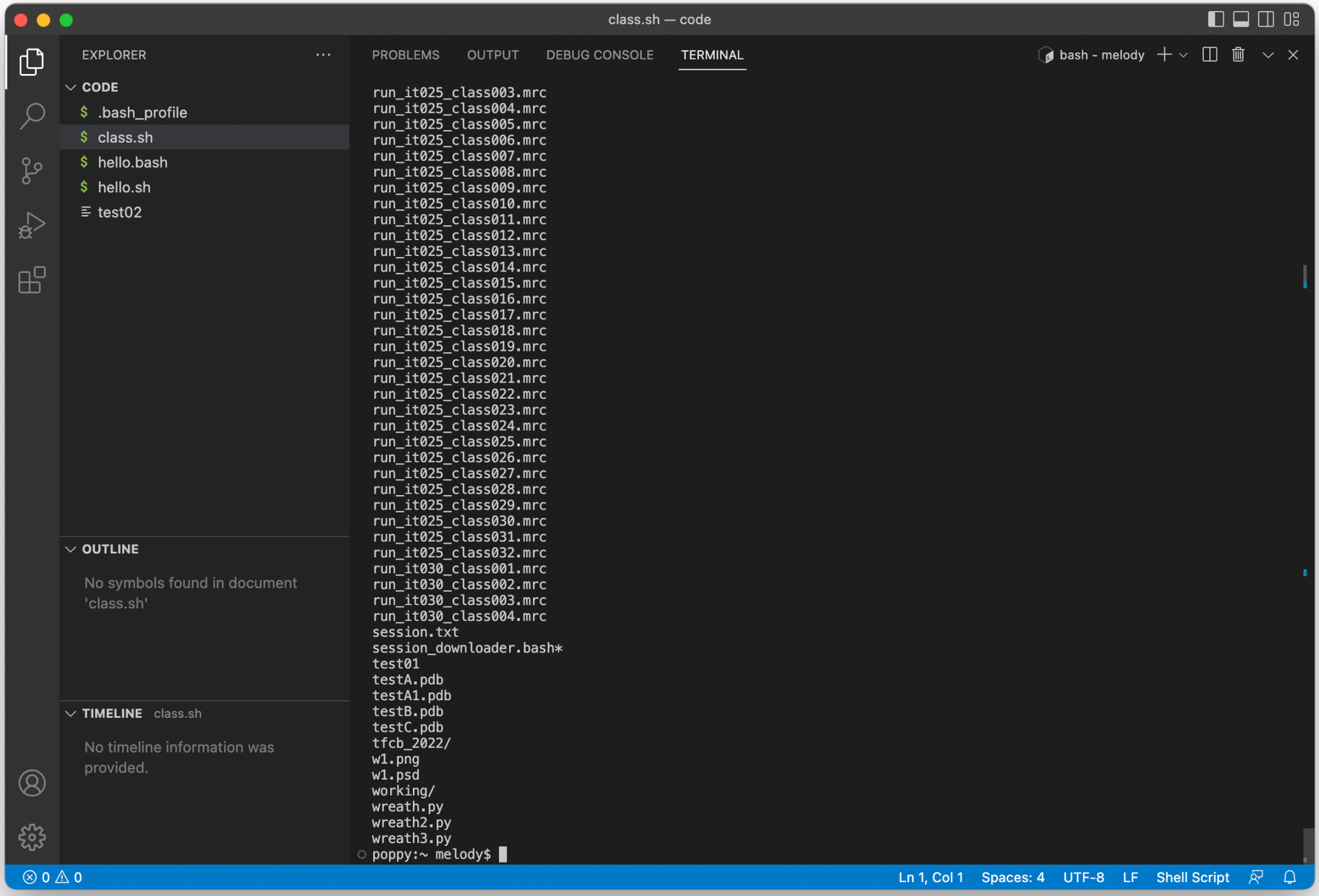
- You can have multiple arguments (this depends on the command)
    - cat file01.txt file02.txt
    - cat file01.txt file02.txt file03.txt file04.txt
- Sometimes you want to read more than one file at once, but you don't want to type it all in. try:
    - cat *.txt
- Also very useful for listing files only of a certain type:
    - ls *.txt
    - ls *.doc
    - ls *.ppt

ls

- ls
  - list directory contents
- cd ..
  - .. means move “up” one directory
    - cd ../.. means move up two directories
      - / delineate folders (e.g. don’t put a / in a file name)
    - . is the directory you’re in
    - shortcut: try cd alone
      - this brings you back to your home dir
- ls
  - what is in this new directory? (/Users/$USER)

Flags

- Flags modify the command
- default ls:

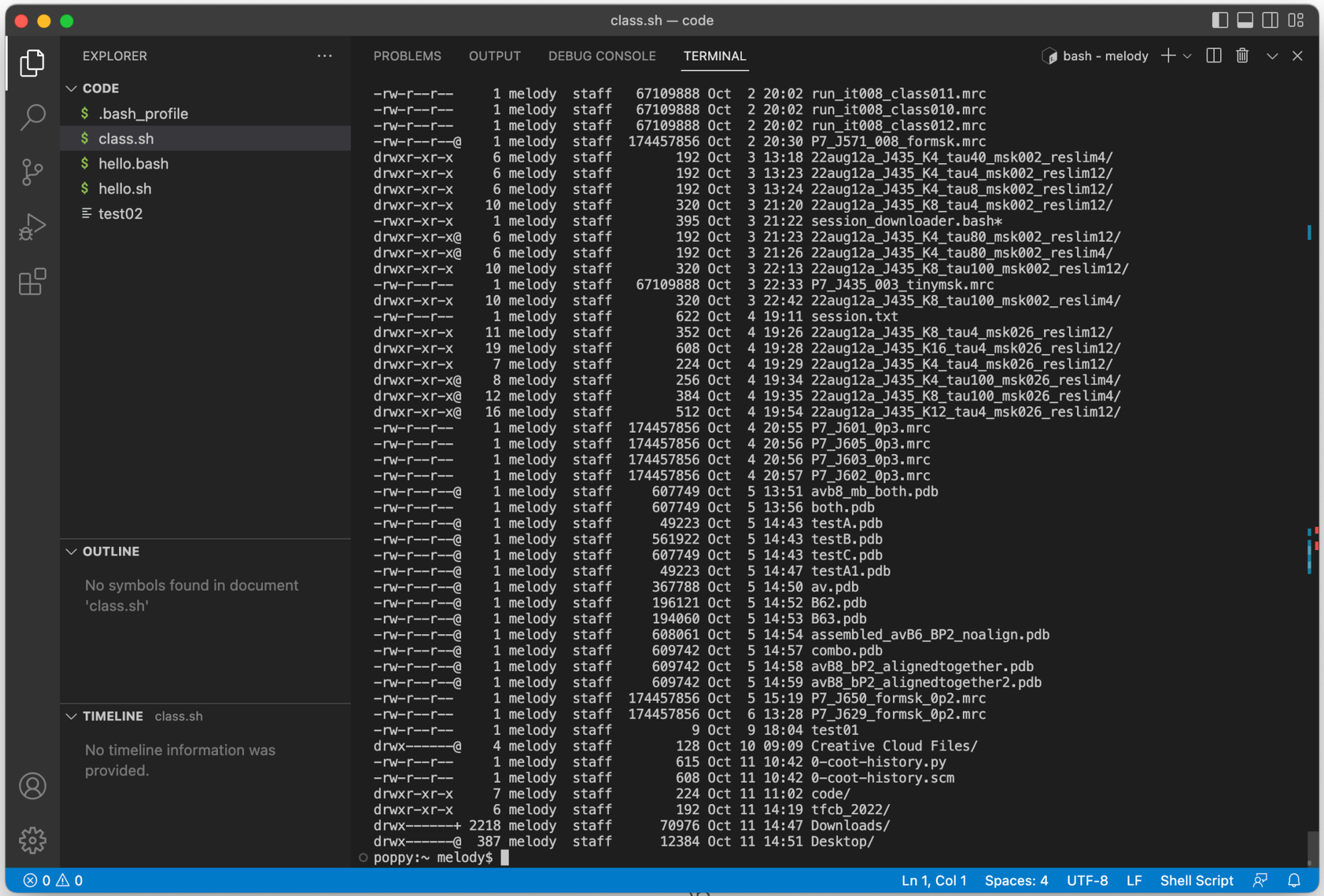


Flags

- Flags modify the command
- type ls
  - default is short format, alphabetical order
- type ls -l
  - -l is a flag
    - it stands for long format
- my favorite: ls -ltr
  - -t sort by depending time (most recently modified files first)
  - -r reverse time (most recently modified files last)

Flags

- my favorite: ls -ltr
  - note how the output has changed



How do I find flags?

- - - help
  - This flag will tell you different options of flags that can be used on a command
  - ls - - help
- flags are different for every command
  - what works on ls will not work on all commands

If this doesn't work, a SIMILAR thing to do is man ls

Syntax (Structure)

command -flag(s) argument

ls -ltr tfcb_2025

what do you want me to do?

what options do you want?

what should i perform it on?

verb adverb noun

english: list out time sorted backwards and fully what is in this folder

Variables

- Variables are shown by having a dollar sign
- Some are set by most systems ($USER $HOME)
  - The docker container does not set $USER
- Others you can set on your own to personalize your computer ~OR~ for writing simple scripts
- They can update and change!
  - they can be commands or flags or arguments
- Example: today_is=october ; echo $today_is
- questions…
  - what is that semicolon?
  - why didn’t you use the space?

; and (escape character)

- This semicolon lets you make multiple commands on one line.
- You can also do this on two lines
  - yesterday_was=september
  - echo $yesterday_was
- Try not to use spaces in file names!
  - Spaces delineate commands, flags and argument
    - If the computer sees a space it thinks it is should move to the next command or argument
  - If you have to use a space or a special character, put a backslash \ first
    - echo hello06 > three\ word\ file.txt
    - this tells the computer that whatever comes next should only be interpreted as text, not the next command

The rm Command

- Be careful with this! Don't use with a wildcard (*) unless you're sure because you can delete whole directories—
- THERE'S NO WAY TO GET THEM BACK!
- Delete one file:
  - rm three (tab complete)

- Delete a whole directory (be careful— this deletes everything in the directory!!) use the man command to read about what -r means and other options
  - rm -r (directory_name)

  - mkdir junk_dir
  - rm -r junk_dir

The mv Command (Move/Rename)

- How to use: (mv input output)

- Rename: Say you type something wrong
- Example:
  - echo hello07 > fiiile07.txt
  - mv fiiile07.txt file07.txt

- Actually move: if you put it in the wrong directory
- Example
  - cd ..
  - hello08 >file08.txt
  - mv file08.txt ../file08.txt
  - mv file08.txt ../. (dot means the same name)
  - mv file08.txt tfcb_2025_in_class/.

Terminate & Cursor shortcuts

- Say you accidentally typed a bunch of things in the terminal that you don't want anymore…
  - Control + C
  - this will not run the command, and give you a fresh new line
- The cursor can be annoying. You have to move via arrows
- Say you typed a bunch of things in the terminal and you made one mistake
  - at the beginning
    - Control + A
  - at the end
    - Control + E

Why are we doing this?

- Many programs and computing clusters don't have nice looking files that you can click and drag to get around
- You now have many of the tools you will need to write scripts and routine and thus automate your work
  - for example: it is much nicer to write a few lines of code that to manually change caats01.jpgs to cats01.jpgs 10,000 times
- Shell scripting helps you get files into the correct format to feed into more sophisticated programs

fournova

COMMAND LINE CHEAT SHEET

presented by **TOWER** › Version control with Git - made easy

DIRECTORIES

```
$ pwd
```
Display path of current working directory

```
$ cd <directory>
```
Change directory to <directory>

```
$ cd ..
```
Navigate to parent directory

```
$ ls
```
List directory contents

```
$ ls -la
```
List detailed directory contents, including hidden files

```
$ mkdir <directory>
```
Create new directory named <directory>

OUTPUT

```
$ cat <file>
```
Output the contents of <file>

```
$ less <file>
```
Output the contents of <file> using the less command (which supports pagination etc.)

```
$ head <file>
```
Output the first 10 lines of <file>

```
$ <cmd> > <file>
```
Direct the output of <cmd> into <file>

```
$ <cmd> >> <file>
```
Append the output of <cmd> to <file>

```
$ <cmd1> | <cmd2>
```
Direct the output of <cmd1> to <cmd2>

```
$ clear
```
Clear the command line window

FILES

```
$ rm <file>
```
Delete <file>

```
$ rm -r <directory>
```
Delete <directory>

```
$ rm -f <file>
```
Force-delete <file> (add `-r` to force-delete a directory)

```
$ mv <file-old> <file-new>
```
Rename <file-old> to <file-new>

```
$ mv <file> <directory>
```
Move <file> to <directory> (possibly overwriting an existing file)

```
$ cp <file> <directory>
```
Copy <file> to <directory> (possibly overwriting an existing file)

```
$ cp -r <directory1>
  <directory2>
```
Copy <directory1> and its contents to <directory2> (possibly overwriting files in an existing directory)

```
$ touch <file>
```
Update file access & modification time (and create <file> if it doesn't exist)

PERMISSIONS

```
$ chmod 755 <file>
```
Change permissions of <file> to 755

```
$ chmod -R 600 <directory>
```
Change permissions of <directory> (and its contents) to 600

```
$ chown <user>:<group> <file>
```
Change ownership of <file> to <user> and <group> (add `-R` to include a directory's contents)

SEARCH

```
$ find <dir> -name "<file>"
```
Find all files named <file> inside <dir> (use wildcards [*] to search for parts of filenames, e.g. "file.*")

```
$ grep "<text>" <file>
```
Output all occurrences of <text> inside <file> (add `-i` for case-insensitivity)

```
$ grep -rl "<text>" <dir>
```
Search for all files containing <text> inside <dir>

NETWORK

```
$ ping <host>
```
Ping <host> and display status

```
$ whois <domain>
```
Output whois information for <domain>

```
$ curl -O <url/to/file>
```
Download <file> (via HTTP[S] or FTP)

```
$ ssh <username>@<host>
```
Establish an SSH connection to <host> with user <username>

```
$ scp <file>
  <user>@<host>:/remote/path
```
Copy <file> to a remote <host>

PROCESSES

```
$ ps ax
```
Output currently running processes

```
$ top
```
Display live information about currently running processes

```
$ kill <pid>
```
Quit process with ID <pid>

fournova

COMMAND LINE TIPS & TRICKS

presented by **TOWER** › Version control with Git - made easy

GETTING HELP

On the command line, help is always at hand: you can either type `man <command>` or `<command> --help` to receive detailed documentation about the command in question.

FILE PERMISSIONS

On Unix systems, file permissions are set using three digits: the first one representing the permissions for the owning user, the second one for its group, and the third one for anyone else.

Add up the desired access rights for each digit as following:

`4` – access/read (r)
`2` – modify/write (w)
`1` – execute (x)

For example, `755` means "rwx" for owner and "rx" for both group and anyone. `740` represents "rwx" for owner, "r" for group and no rights for other users.

COMBINING COMMANDS

If you plan to run a series of commands after another, it might be useful to combine them instead of waiting for each command to finish before typing the next one. To do so, simply separate the commands with a semicolon (`;`) on the same line.

Additionally, it is possble to execute a command only if its predecessor produces a certain result. Code placed after the `&&` operator will only be run if the previous command completes successfully, while the opposite `||` operator only continues if the previous command fails. The following command will create the folder "videos" only if the `cd` command fails (and the folder therefore doesn't exist):

```
$ cd ~/videos || mkdir ~/videos
```

THE "CTRL" KEY

Various keyboard shortcuts can assist you when entering text: Hitting `CTRL+A` moves the caret to the beginning and `CTRL+E` to the end of the line.

In a similar fashion, `CTRL+K` deletes all characters after and `CTRL+U` all characters in front of the caret.

Pressing `CTRL+L` clears the screen (similarly to the `clear` command). If you should ever want to abort a running command, `CTRL+C` will cancel it.

THE "TAB" KEY

Whenever entering paths and file names, the `TAB` key comes in very handy. It autocompletes what you've written, reducing typos quite efficiently. E.g. when you want to switch to a different directory, you can either type every component of the path by hand:

```
$ cd ~/projects/acmedesign/docs/
```

...or use the `TAB` key (try this yourself):

```
$ cd ~/pr[TAB]ojects/
  ac[TAB]medesign/d[TAB]ocs/
```

In case your typed characters are ambiguous (because "ac" could point to the "acmedesign" or the "actionscript" folder), the command line won't be able to autocomplete. In that case, you can hit `TAB` twice to view all possible matches and then type a few more characters.

THE ARROW KEYS

The command line keeps a history of the most recent commands you executed. By pressing the `ARROW UP` key, you can step through the last called commands (starting with the most recent). `ARROW DOWN` will move forward in history towards the most recent call.

Bonus tip: Calling the `history` command prints a list of all recent commands.

HOME FOLDER

File and directory paths can get long and awkward. If you're addressing a path inside of your home folder though, you can make things easier by using the `~` character. So instead of writing `cd /Users/your-username/projects/`, a simple `cd ~/projects/` will do.

And in case you should forget your user name, `whoami` will remind you.

OUTPUT WITH "LESS"

The `less` command can display *and paginate* output. This means that it only displays one page full of content and then waits for your explicit instructions. You'll know you have `less` in front of you if the last line of your screen either shows the file's name or just a colon (`:`).

Apart from the arrow keys, hitting `SPACE` will scroll one page forward, `b` will scroll one page backward, and `q` will quit the `less` program.

DIRECTING OUTPUT

The output of a command does not necessarily have to be printed to the command line. Instead, you can decide to direct it to somewhere else.

Using the `>` operator, for example, output can be directed to a file. The following command will save the running processes to a text file in your home folder:

```
$ ps ax > ~/processes.txt
```

It is also possible to pass output to another command using the `|` (pipe) operator, which makes it very easy to create complex operations. E.g., this chain of commands will list the current directory's contents, search the list for PDF files and display the results with the `less` command:

```
$ ls | grep ".pdf" | less
```

30-day free trial available at
www.git-tower.com

TOWER
Version control with Git - made easy

Extra Commands That Were Requested Previous Years

- How to rename:
  - mv input.txt output.txt
- Copy files
  - cp file.txt file2.txt
- How to delete a file
  - rm file.txt
- How to delete a dir
  - rm -r directory
- Print out contents of dir into new text file
  - ls * > contents.txt
- How to create many sequentially numbered folders/ directories

Any New Command Requests?