Week 1: Introduction to Computation

POP77001 Computer Programming for Social Scientists

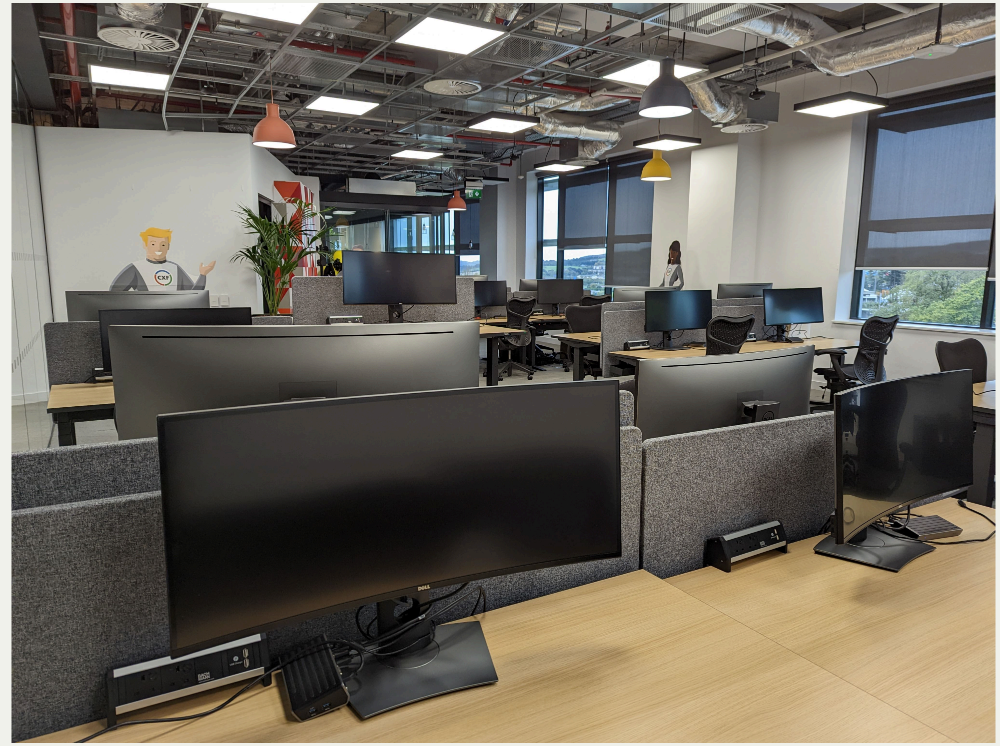
Tom Paskhalis

Overview

- Computers and Computational thinking
- Algorithms
- Programming languages and computer programs
- Debugging
- Command-line Interfaces
- Version controlling with Git/GitHub

Computers

1920s



2020s

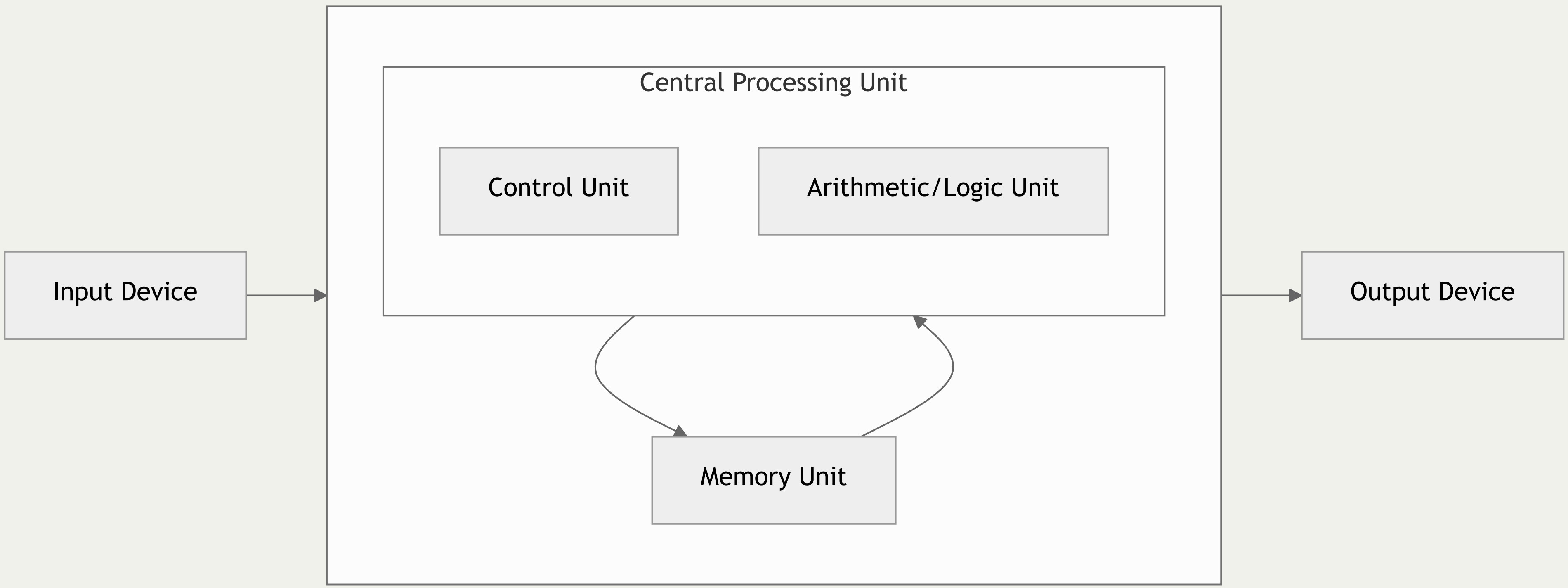
Who Do Computers Do?

Two things:

1. **Perform calculations**

2. **Store results of calculations**

von Neumann Architecture



Computational Thinking

- *Conceptualizing, not programming* - multiple levels of abstraction
- *A way, that humans, not computers, think* - creatively and imaginatively
- *Complements and combines mathematical and engineering thinking*

Wing, Jeannette M. 2006. “Computational Thinking.” Communications of the ACM, 49 (3): 33–35. doi: 10.1145/1118178.1118215

Computational Thinking

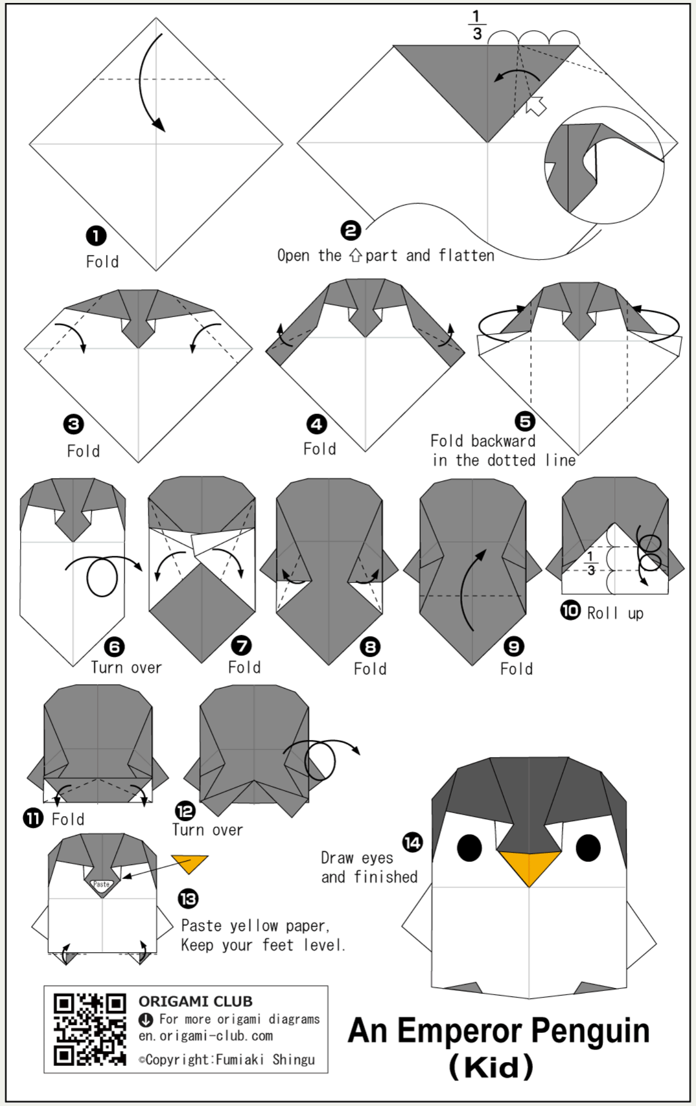
All knowledge can be thought of as:

1. Declarative (statement of fact)
   E.g. square root of 25 equals 5
2. Imperative (how to)
   E.g. to find a square root of x, start with a guess g, check whether g*g is close, …

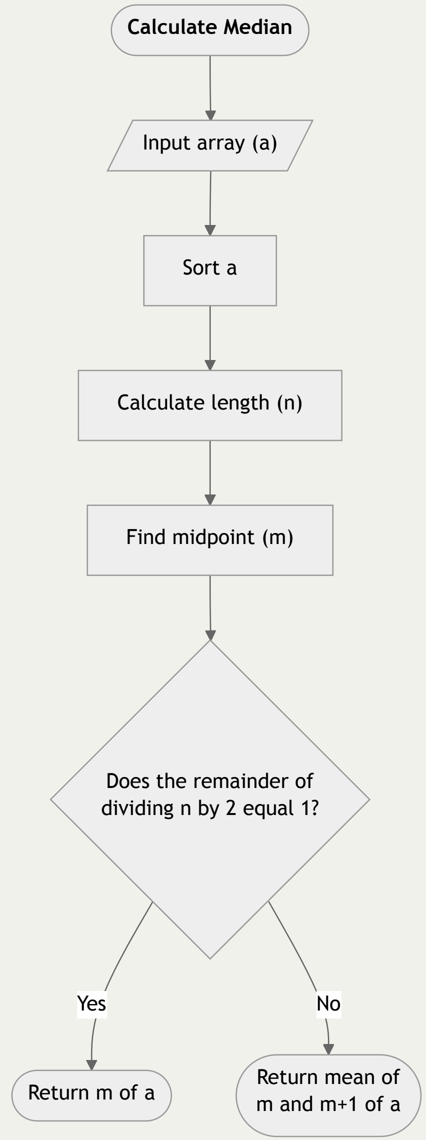
Algorithm

- *Finite list of well-defined instructions that take input and produce output.*
- Consists of a sequence of simple steps that start from input, follow some control flow and have a stopping rule.

Algorithm Example



Algorithm Example



Programming Language

Formal language used to define sequences of instructions (for computers to execute) that includes:

- Primitive constructs
- Syntax
- Static semantics
- Semantics

Types of Programming Languages

- Low-level vs high-level
  - E.g. available procedures for moving bits vs calculating a mean
- General vs application-domain
  - E.g. general-purpose vs statistical analysis
- Interpreted vs compiled
  - Source code executed directly vs translated into machine code

Primitive Constructs

Just like natural languages are made up of different elements such as words and punctuation marks, programming languages are composed of:

- Literals

```
77001
```

```
'POP'
```

- Operators

```
+
```

Syntax

- Defines which combinations of symbols form valid expressions.
- E.g. in English “Animals drink water” is valid, while “Animals water drink” isn’t.
- While in Irish “Ólann ainmhithe uisce” is valid, and “Ainmhithe ólann uisce” isn’t.

```
# Infix operator (used in most programming languages)
77001 + 23
```

```
[1] 77024
```

```
# Prefix operator (used in Lisp-based programming languages)
+ 77001 23
```

```
Error in parse(text = input): <text>:2:9: unexpected numeric constant
1: # Prefix operator (used in Lisp-based programming languages)
2: + 77001 23
           ^
```

```
# Postfix operator (very rare in programming languages)
77001 23 +
```

```
Error in parse(text = input): <text>:2:7: unexpected numeric constant
1: # Postfix operator (very rare in programming languages)
2: 77001 23
         ^
```

Static Semantics

- Defines which syntactically valid sequences have a meaning
- E.g. in English "Animals drunk water" is invalid, while "Animals drank water" is.

R:

```r
'POP' + '77001'
```

```
Error in "POP" + "77001": non-numeric argument to binary operator
```

Python:

```python
'POP' + '77001'
```

```
'POP77001'
```

Semantics in Programming Languages

- Associates a meaning with each syntactically correct sequence of symbols that has no static semantic errors.
- Programming languages are designed so that each legal program has exactly one meaning.
- **This meaning, however, does not, necessarily, reflect the intentions of the programmer**.
- Syntactic errors are much easier to detect.

Expressions & Statements

- Text written in natural languages consists of certain grammatical structures such as clauses and sentences.
- Programs implemented in programming languages consist of:
  - **Expressions** - syntactic entities that may be evaluated to determine a value.

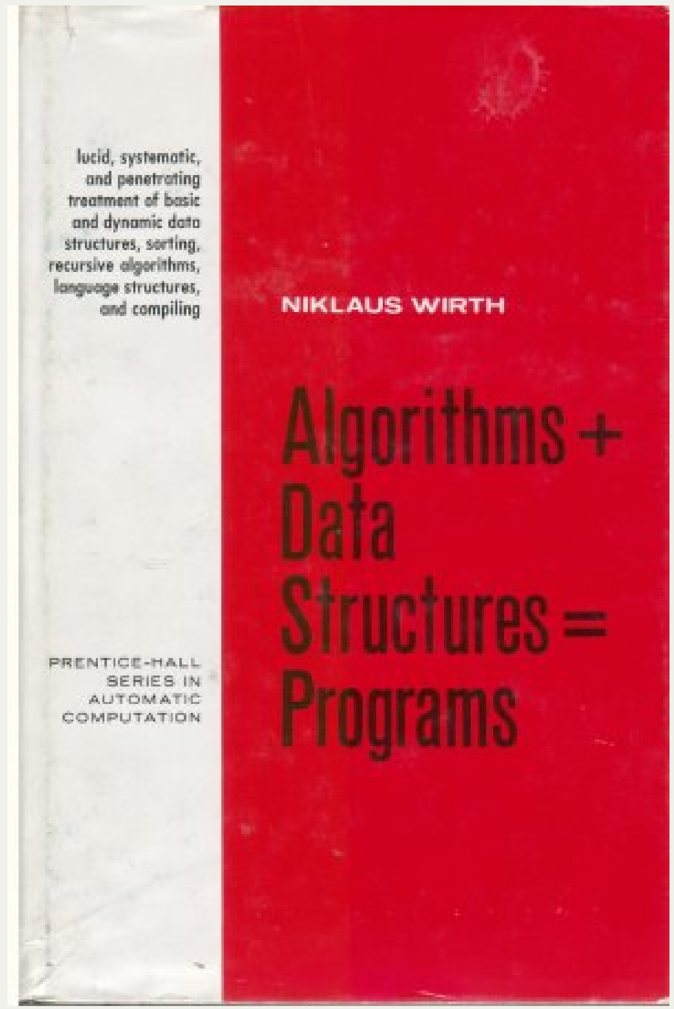
    ```r
    77001 + 23
    ```

    ```
    [1] 77024
    ```

  - **Statements** - syntactic entities that may be executed to change a state.

    ```r
    x <- 77001 + 23
    ```

Algorithms + Data Structures = Programs



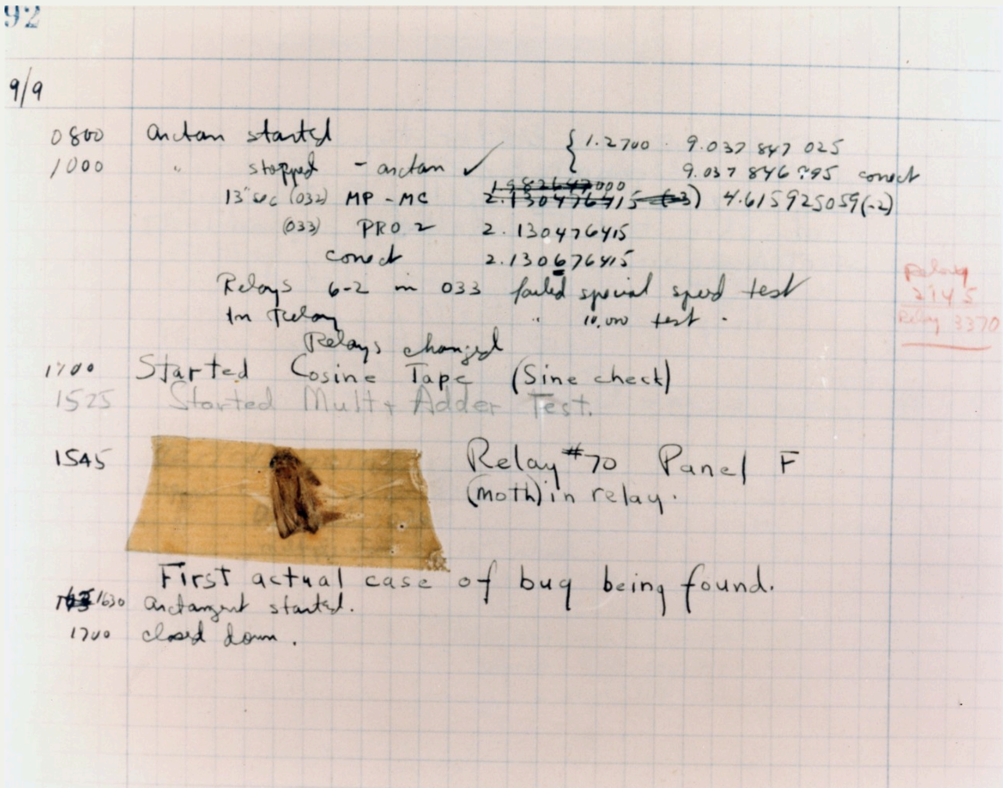
Computer Program

- A collection of instructions that can be executed by computer to perform a specific task
- For interpreted languages (e.g. Python, R, Julia) instructions (source code)
  - Can be executed directly in the interpreter
  - Can be stored and run from the terminal

Programming Errors

- Often, programs would run with errors or behave in an unexpected way
- Programs might crash
- They might run too long or indefinitely
- **Run to completion and produce an incorrect output**

Computer Bugs



US Naval History and Heritage Command

US Navy

Grace Murray Hopper popularised the term *bug* after in 1947 her team traced an error in the Mark II to a moth trapped in a relay.

How to Debug

- Search error message online (e.g. StackOverflow or, indeed, #LMDDGTFY)
- Insert `print()` statement to check the state between procedures
- Use built-in debugger (stepping through procedure as it executes)
- More to follow!

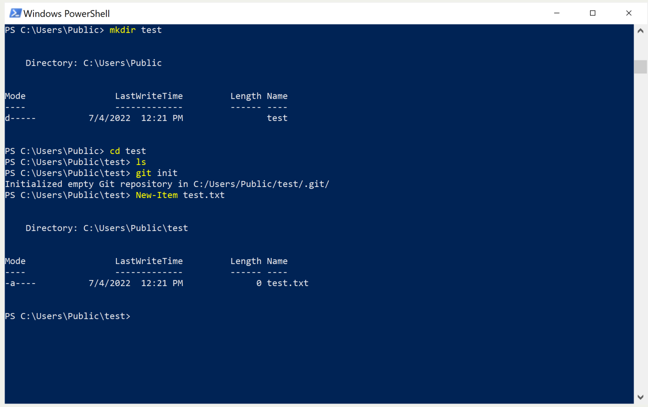
Command-line Interface

- aka terminal/console/shell/command line/command prompt
- Most users today rely on graphical interfaces.
- Command line interpreters (CLIs) provide useful shortcuts.
- Computer programs can be run or scheduled in terminal/CLI.
- CLI/terminal is usually **the only available interface** if you work in the cloud (AWS, Microsoft Azure, etc.).

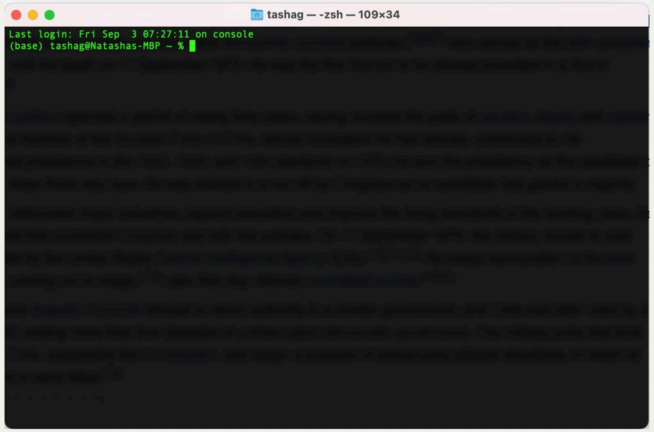
Extra

Five reasons why researchers should learn to love the command line

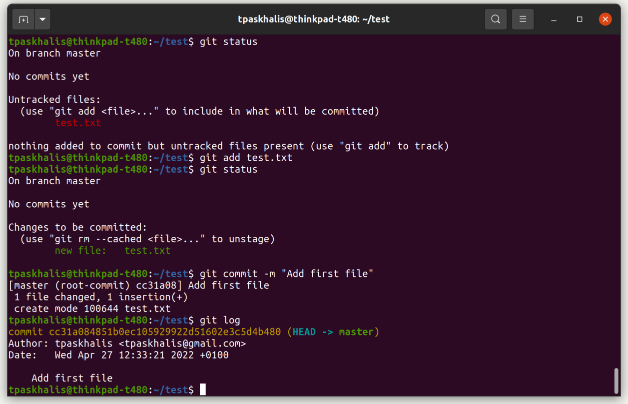
CLI Examples



Microsoft PowerShell (Windows)



Z shell, zsh (macOS)



bash (Linux/UNIX)

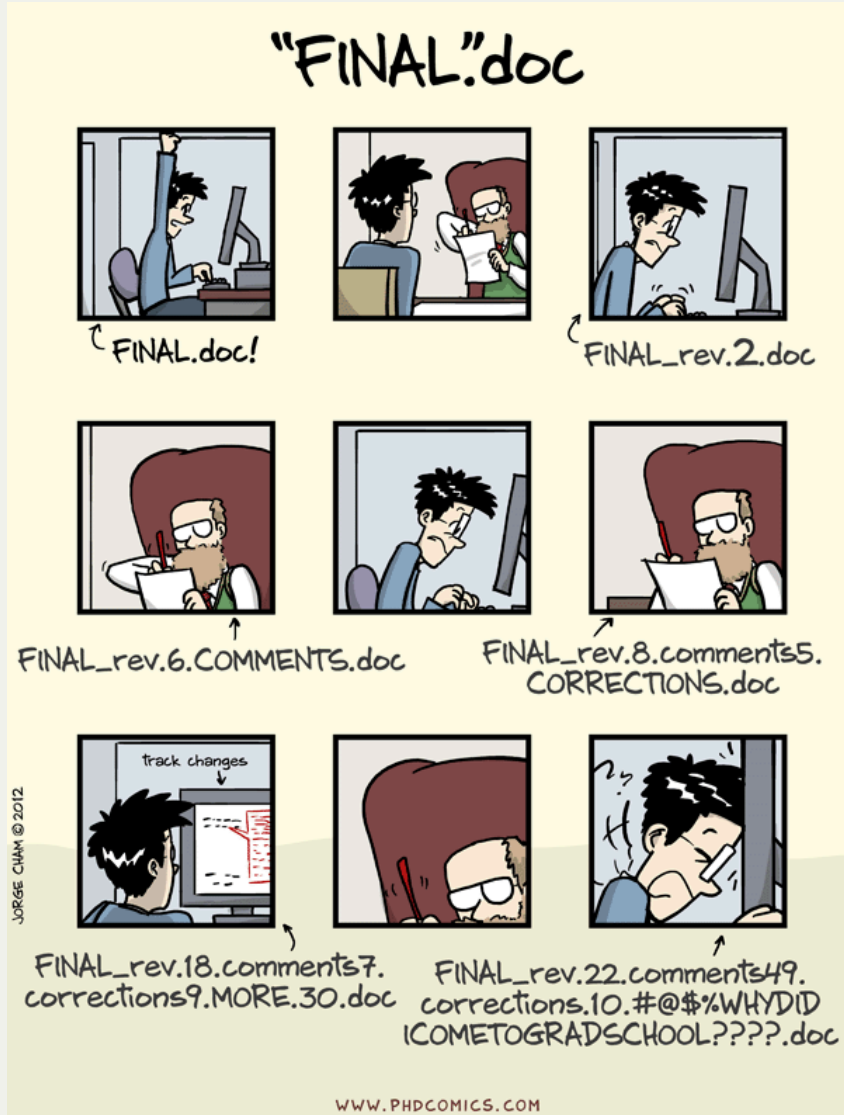
Some Useful CLI Commands

Command (Windows)	Command (macOS/Linux)	Description
`exit`	`exit`	close the window
`cd`	`cd`	change directory
`cd`	`pwd`	show current directory
`dir`	`ls`	list directories/files
`copy`	`cp`	copy file
`move`	`mv`	move/rename file
`mkdir`	`mkdir`	create a new directory
`del`	`rm`	delete a file

Extra

Introduction to CLI

Version Control



Jorge Cham, PhD Comics

Version Control and Git

- Version control systems (VCSs) allow automatic tracking of changes in files and collaboration.
- Git is one of several major version control systems (VCSs, see also Mercurial, Subversion).
- GitHub is an online hosting platform for projects that use Git for version control.

Some Useful Git Commands

Command	Description
`git init <project name>`	Create a new *local repository*
`git clone <project url>`	Download a project from *remote repository*
`git status`	Check project status
`git diff <file>`	Show changes between *working directory* and *staging area*
`git add <file>`	Add a file to the *staging area*
`git commit -m “<commit message>”`	Create a new *commit* from changes added to the *staging area*
`git pull <remote> <branch>`	Fetch changes from *remote* and merge into *merge*
`git push <remote> <branch>`	Push local branch to *remote repository*

Extra

Git Cheatsheet

Things to Do

- Make sure you have installed R, Python and Jupyter.
- Make sure you can run R code through Jupyter (requires extra steps).
- Do the readings.

Things to Try (CLI)

- Identify an appropriate CLI for your OS.
- Try navigating across folders and files using CLI.
- Try creating a test folder and test file inside it.

Things to Try (git)

- Register on GitHub and GitHub Education (for free goodies!)
- Create a test repository in CLI and initialise as a Git repository
- Or create a repository on GitHub and clone to your local machine
- Create test.txt file, add it and commit
- Push the file to GitHub

Next

- Tutorial: Jupyter Notebooks, CLIs, Git/GitHub
- Next week: R Basics