Session 2
Computing and R

R for Social Data Science

Jeffrey Ziegler, PhD

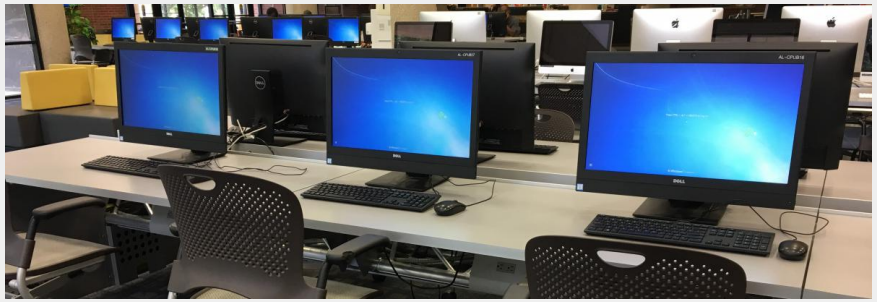
Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

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

Computers

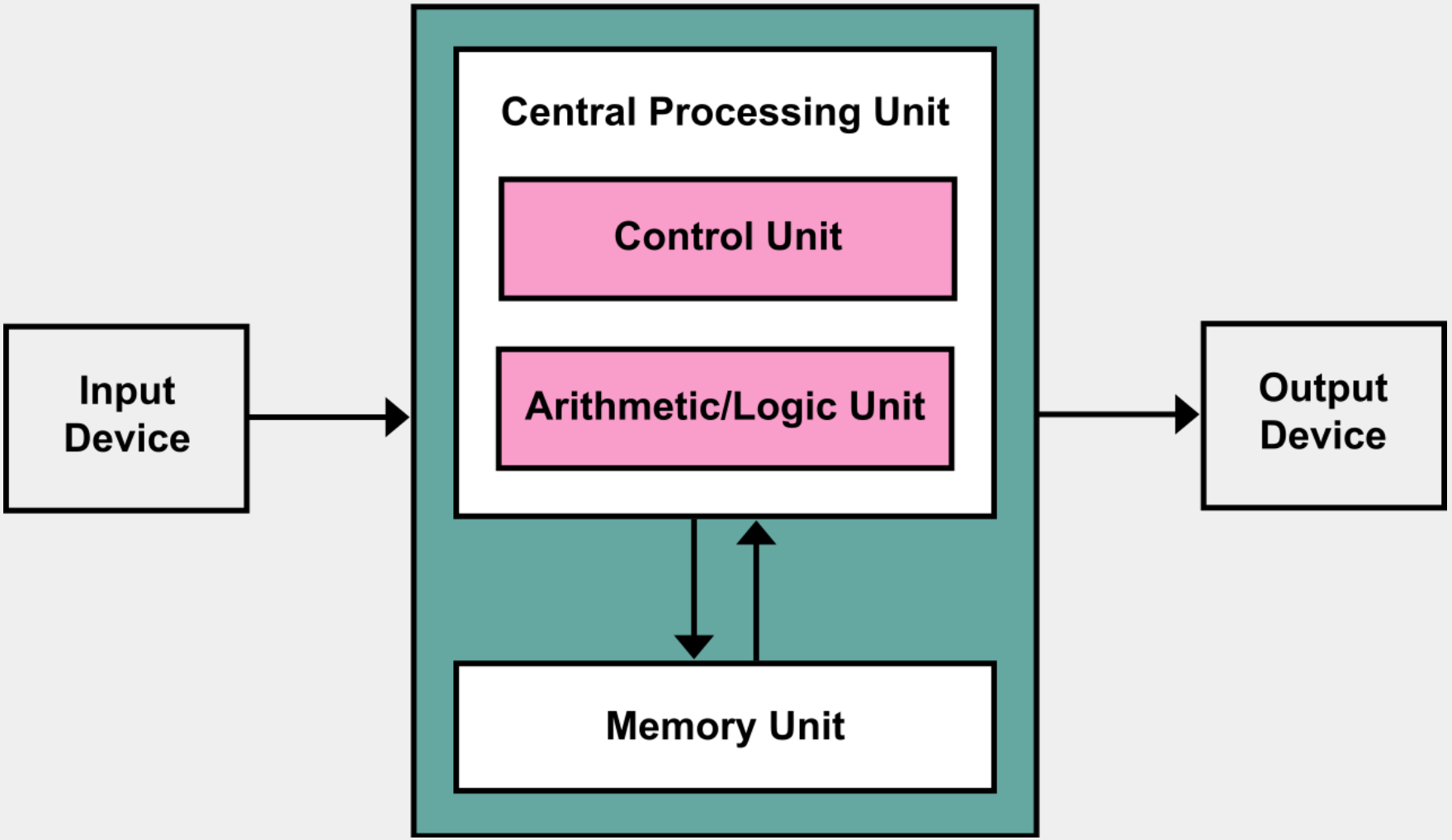


Computers

Do two things:

1. Perform calculations
2. Store results of calculations

von Neumann Architecture



Source: Wikipedia

Computational Thinking

"Computational thinking is breaking down a problem and formulating a solution in a way that both human and computer can understand and execute."[1]

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

[1] Wing, Jeannette M. 2006. Computational Thinking. Communications of the ACM, 49 (3): 33–35.

Computational Thinking

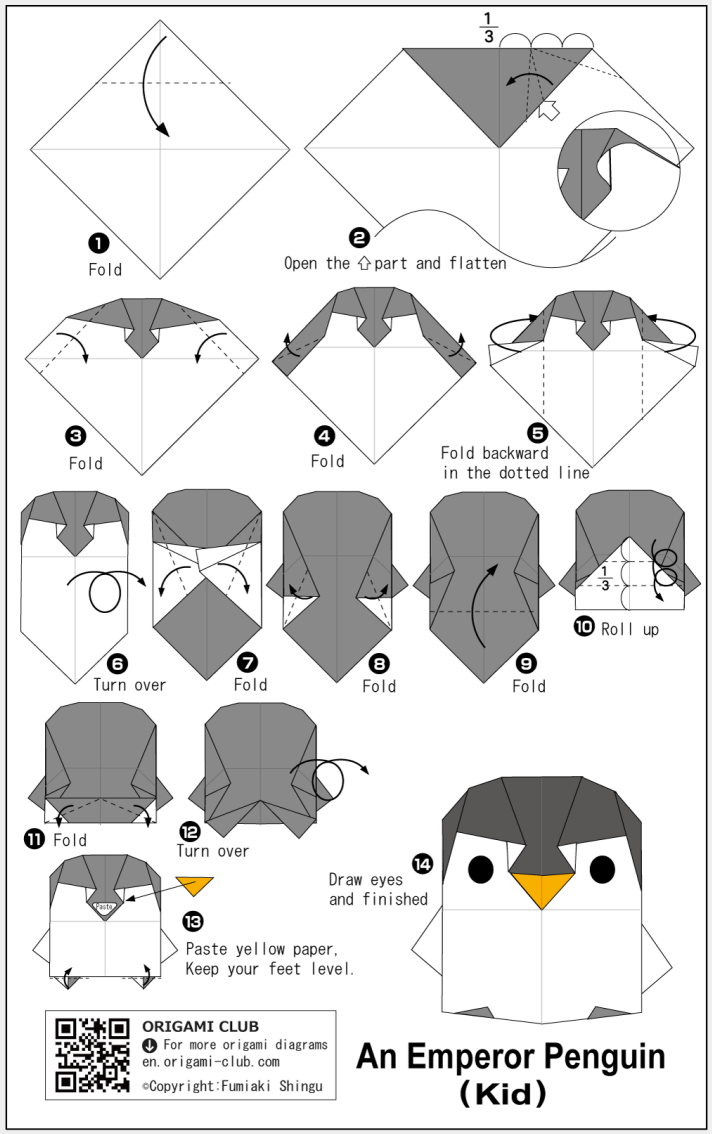
All knowledge can be thought of as:

- Declarative (statement of fact, e.g. $\sqrt{25} = 5$)
- Imperative (how to, e.g. to find $\sqrt{x}$, start with a guess g, check whether g^* is close, ...

Algorithm

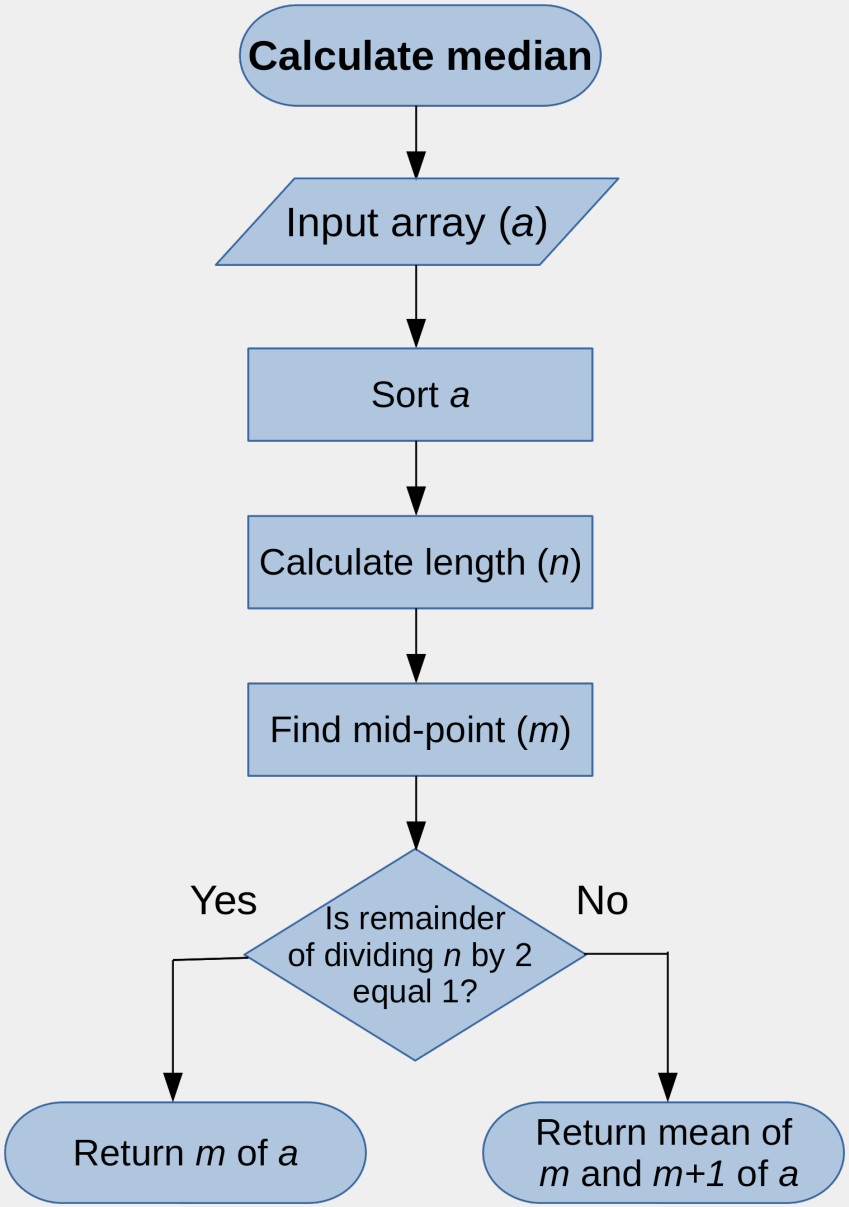
- Finite list of well-defined instructions that take input and produce output
- Consists of a sequence: Simple steps that start from input and follow some control flow and have a stopping rule

Algorithm Example



Source: Origami Club

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
  - Available procedures for moving bits vs calculating a mean
- General vs application-domain
  - General-purpose vs statistical analysis
- Interpreted vs compiled
  - Source code executed directly vs translated into machine code

Primitive Constructs in R

- Literals

```
77001
```

```
77001
```

```
'POP'
```

```
'POP'
```

- Infix operators

```
77001+23
```

```
77024
```

Syntax in R

- Defines which sequences of characters and symbols are well-formed
- E.g. in English sentence "Cat dog saw" is invalid, while "Cat saw dog" is

```
77001+23
```

```
77024
```

```
77001 23 +
```

```
Error: unexpected numeric constant in "77001 23"
```

Static Semantics in R

- Defines which syntactically valid sequences have a meaning
- In English sentence "Cat seen dog" is invalid, while "Cat saw dog" is

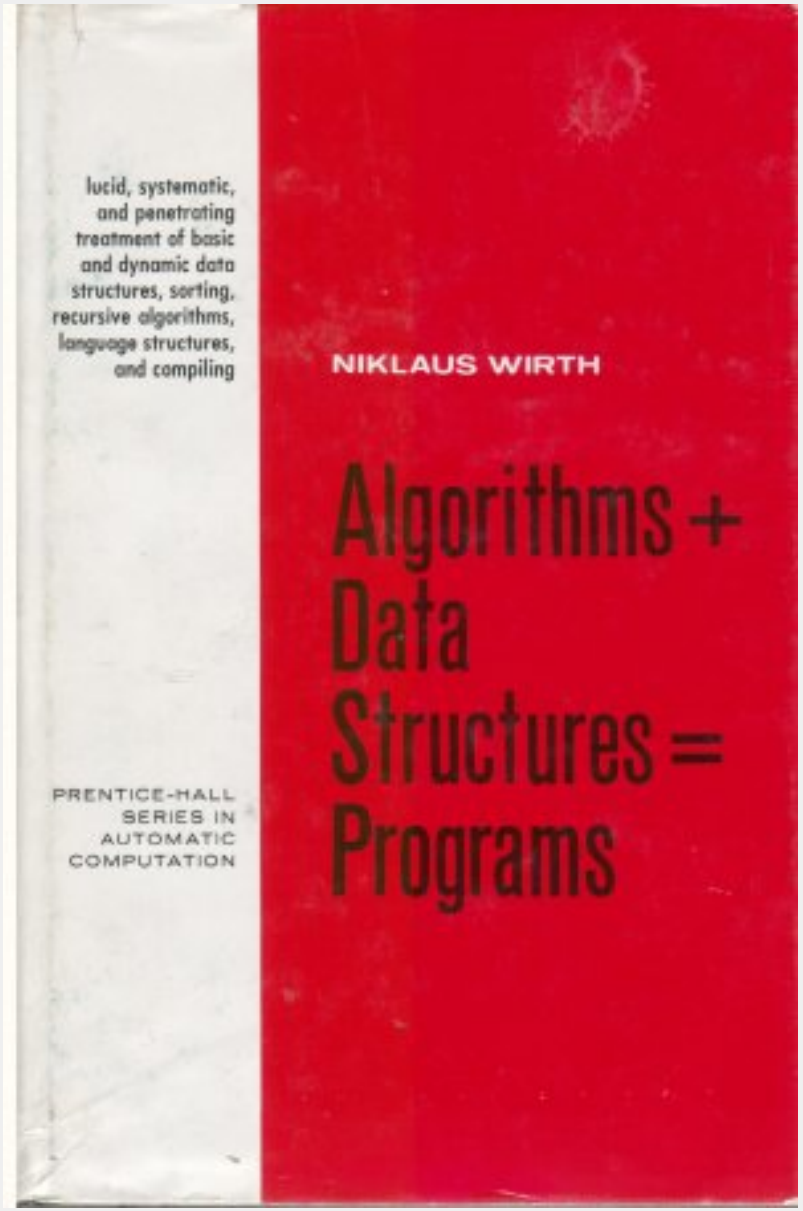
```
1       'POP'+77001
```

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

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 intentions of programmer
- Syntactic errors are much easier to detect

Algorithms + Data Structures = Programs



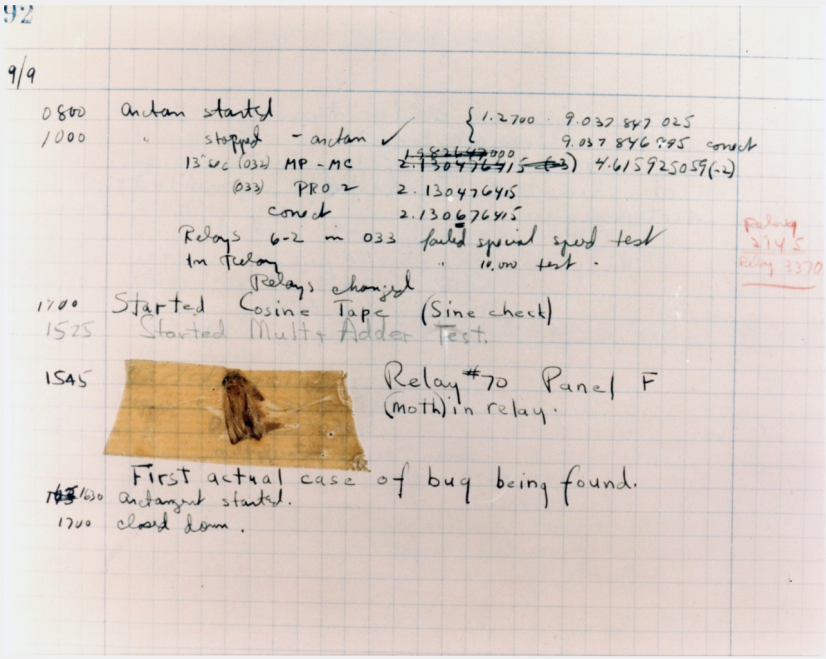
Computer Program

- 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



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

Source: US Naval History and Heritage Command

How to Debug

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

Debugging

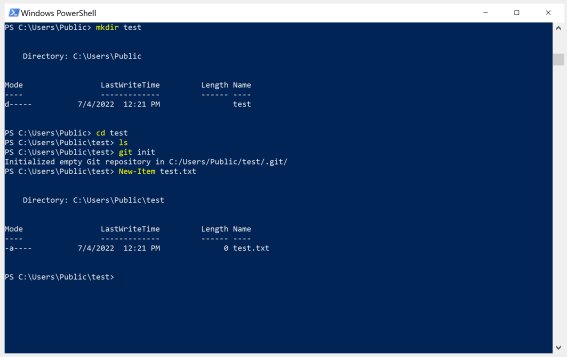
Source: Julia Evans

Command Line Interface (terminal/console/shell)

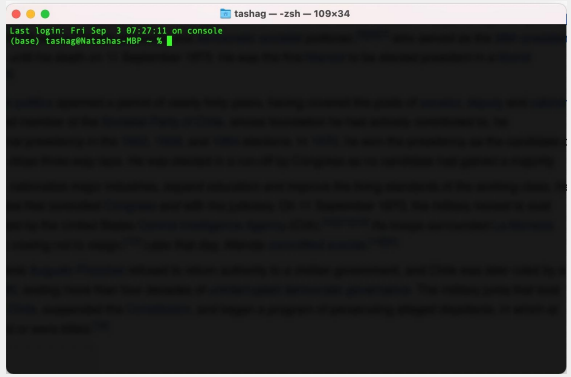
- Most users today rely on graphical interfaces
- Command line interpreters (CLIs) provide useful shortcuts
- Computer programs can be run or scheduled in terminal/CLI
- CLI/terminal is usually the only available interface if you work in the "cloud" (AWS, Microsoft Azure, etc.)

Extra: Five reasons why researchers should learn to love the command line

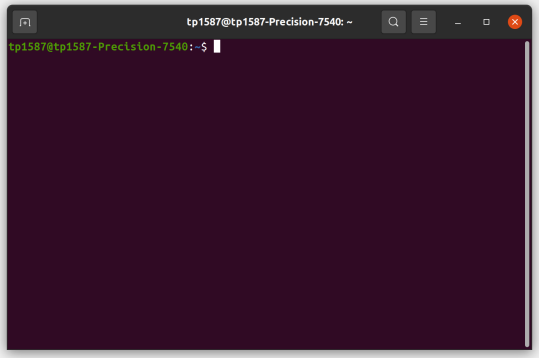
CLI Examples



(a) PowerShell (Windows)



(b) Z shell, zsh (macOS)



(c) bash (Linux/UNIX)

Some Useful CLI Commands

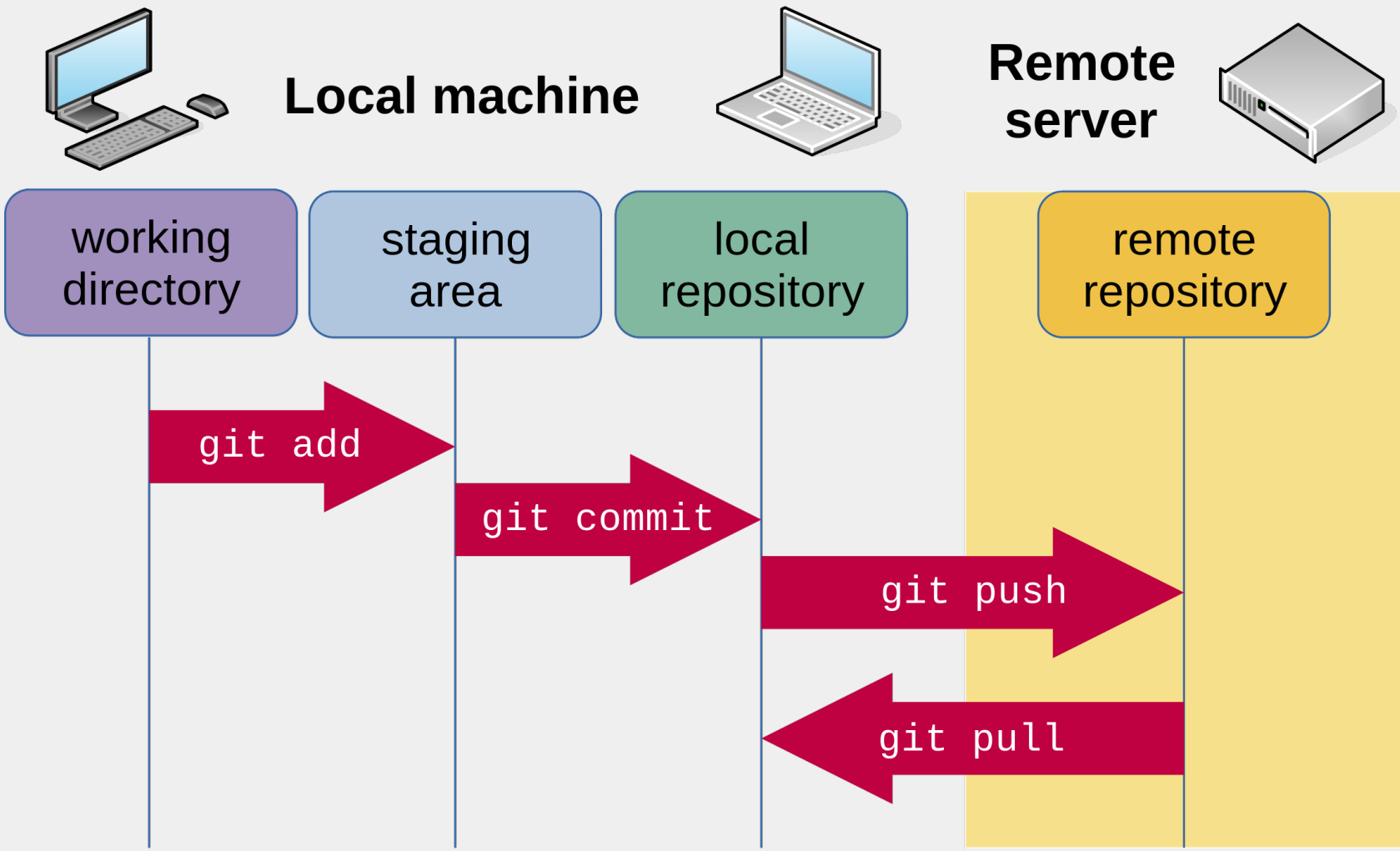
Command (Windows)	Command (macOS/Linux)	Description
`exit`	`exit`	close the window
`cd`	`cd`	change directory
`cd`	`pwd`	show current directory
`dir`	`ls`	list directories/files
`copy`	`cp`	copy file
`move`	`mv`	move/rename file
`mkdir`	`mkdir`	create a new directory
`del`	`rm`	delete a file

Extra: Introduction to CLI

Version Control 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

Git/GitHub Workflow



Some Useful Git Commands

Command (Windows)	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

Creating local Git repository

- Let's create a test project and track changes in it
- Create a test directory by typing 'mkdir test' in your CLI/Terminal
- Go into the newly created directory with 'cd test' command
- To make Git track changes run 'git init' command in this directory
- Congratulations! You now have a local repository for your test project

Making a commit: Creation to Staging

- Open your text editor of choice (Notepad, Sublime Text, Atom, Visual Studio Code, Vim, Emacs, ...)
- Create a file called ‘test.txt‘ in your local test repository
- Type whatever you like in this file
- Add this file to your staging area (make Git aware of its existence) by running ‘git add test.txt‘ command

Making a commit: Staging to Committing

- Commit this file to your local repository by running ‘git commit -m "Added first file"‘
- Note that all files that were added at the previous stage with ‘git add <file>‘ would be commited
- Check status of your repository by running ‘git status‘ (it should say ’nothing to commit, working tree clean’)
- Check history of your repository by running ‘git log‘ and make sure that you see your commit

Remote Git repository: GitHub

- Hosting platform for projects that rely on Git fo version control
- Bought by Microsoft in 2018
- Provides extensive tools for collaborative development and search functionality
- Helpful for troubleshooting more narrow problems (check GitHub Issues of the package/library that you have a problem with)
- GitHub is far from the only platform for hosting Git projects
- Popular alternatives to GitHub include GitLab, SourceForge

Creating remote repository on GitHub

- Register and login into your account on GitHub
- Create a new GitHub repository (choose private repository)
- You should see a similar page with the project URL of the form:

Synchronising local Git repository with GitHub

- Go to your local Git repository (the one created in the previous step)
- Add link from your local Git repository to remote repository on GitHub by running: ‘git remote add origin <project_url>‘

  where:
  - ‘git remote add‘ is the command,
  - ‘origin‘ is the name given to this link (‘<remote>‘), and
  - ‘<project_url>‘ is the URL of the repository on GitHub
- Check the status of links between your local Git repository and remotes by running ‘git remote -v‘

  where:
  - ‘git remote‘ is the command, and
  - ‘-v‘ is the argument ’verbose’

Pushing local Git changes to GitHub

- Your local Git repo is now linked to remote repo hosted on GitHub
- Let's bring the changes made locally to the remote repository
- We will use the 'git push' command for that
- One last thing to check before doing so is which branch we're on
- Run 'git branch' to see name of branch you're on ('master' or 'main')
- Finally, run 'git push <remote> <branch>' (e.g. 'git push origin master')

  where:
  - 'git push' is the command,
  - '<remote>' is the name of the remote link, and
  - '<branch>' is the name of the branch
- Visit your GitHub repository to check that your commit is reflected there

Cloning module repository

- All module materials are hosted on GitHub in this repo
- You can clone this repository to your local machine by running: ‘git clone https://github.com/jeffreyziegler/R_social_DS‘
- This will create a folder called ‘R_social_DS‘ within the directory where you ran this command
- To keep up to date with changes in the remote repository you can run: ‘git pull origin main‘

  where
  - ‘origin‘ is the remote address of the repository - ‘https://github.com/jeffreyziegler/R_social_DS‘
  - ‘main‘ is the name of the branch (recall the discussion about ‘main‘/‘master‘ change)

"Tutorial": 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

"Tutorial": CLI + Git

- 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

"Tutorial": Forking Module Repository

- Fork online repository
- Download your forked repository using GitHub desktop
- Create "test.txt" file to your new local GitHub repository, add it and commit
- Push the file to GitHub, and check

Class business

Today, we talked about...

- Computers, computational thinking, algorithms
- Programming languages and computer programs
- Debugging
- Command-line interfaces (CLI)
- Version controlling with Git/GitHub

Next week...

- R basics