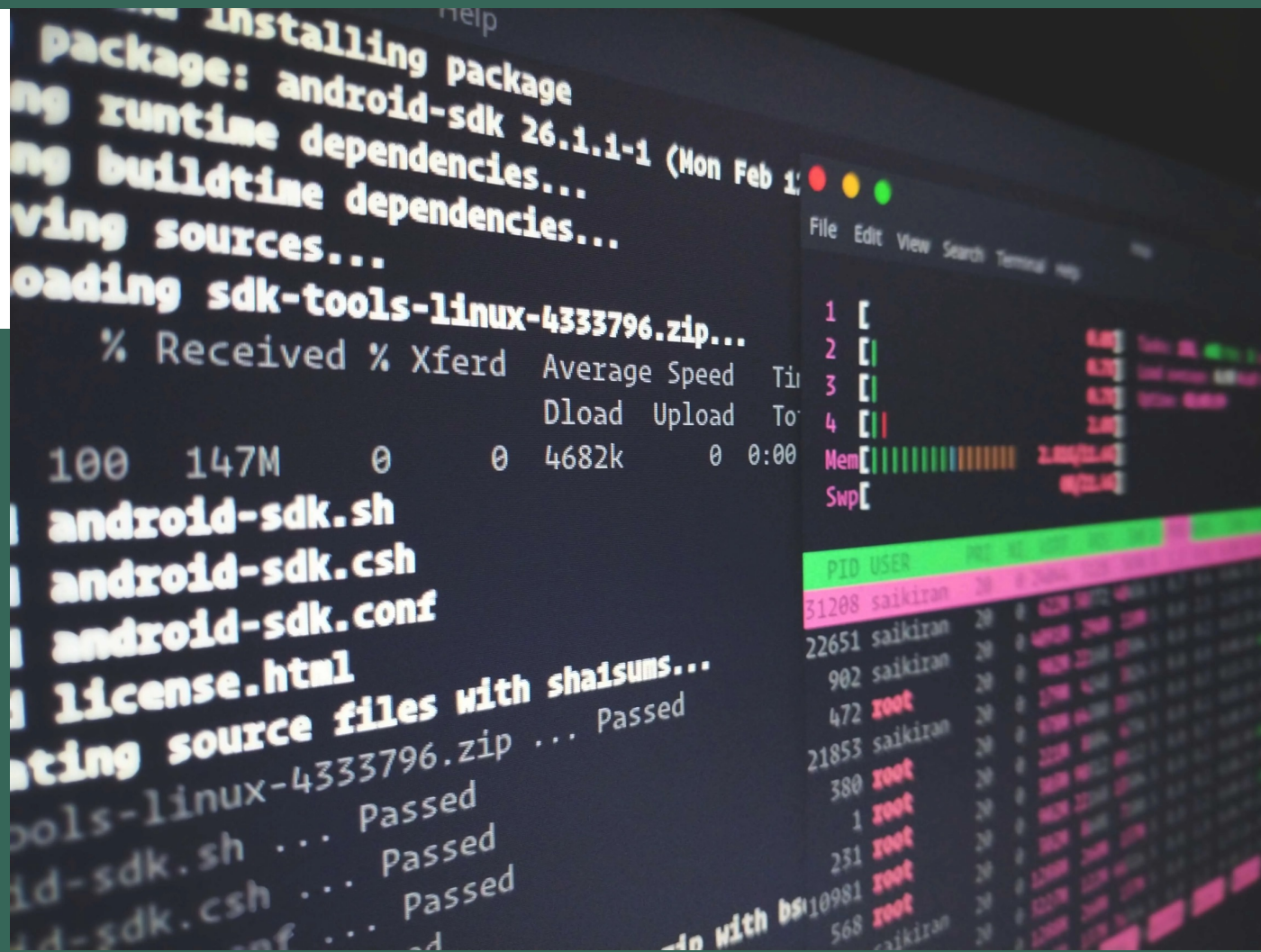


COMMAND LINE & GIT




The image is a composite of two screenshots from a Linux environment. The background is a terminal window showing the installation progress of the Android SDK. The text in the terminal includes: "Installing package", "package: android-sdk 26.1.1-1 (Mon Feb 1", "ng runtime dependencies...", "ng buildtime dependencies...", "ving sources...", "loading sdk-tools-linux-4333796.zip...", and a progress bar showing "100 147M 0 0 4682k 0 0:00". Below the progress bar, there are several lines of text: "android-sdk.sh", "android-sdk.csh", "android-sdk.conf", "license.html", and "ating source files with sha1sums...". The foreground is a system monitor window showing a list of processes. The table has columns for PID, USER, and various resource usage metrics. The processes listed include "saikiran" and "root".

```
Installing package
package: android-sdk 26.1.1-1 (Mon Feb 1
ng runtime dependencies...
ng buildtime dependencies...
ving sources...
loading sdk-tools-linux-4333796.zip...
% Received % Xferd Average Speed Time
Dload Upload To
100 147M 0 0 4682k 0 0:00
android-sdk.sh
android-sdk.csh
android-sdk.conf
license.html
ating source files with sha1sums...
ools-linux-4333796.zip ... Passed
id-sdk.sh ... Passed
id-sdk.csh ... Passed
id-sdk.conf ... Passed
```

PID	USER	PR	NI	U	St	VSZ	SSZ	RES	TIME	COMMAND
31208	saikiran	20	0	0	0	4228	3872	4096	0:00	...
22651	saikiran	20	0	0	0	4028	2960	2048	0:00	...
902	saikiran	20	0	0	0	1628	2240	1792	0:00	...
472	root	20	0	0	0	1796	4240	3072	0:00	...
21853	saikiran	20	0	0	0	1796	4240	3072	0:00	...
380	root	20	0	0	0	2228	8040	4736	0:00	...
1	root	20	0	0	0	3428	9012	4928	0:00	...
231	root	20	0	0	0	1628	2240	1792	0:00	...
10981	root	20	0	0	0	12408	12240	2048	0:00	...
568	saikiran	20	0	0	0	12276	2440	2048	0:00	...

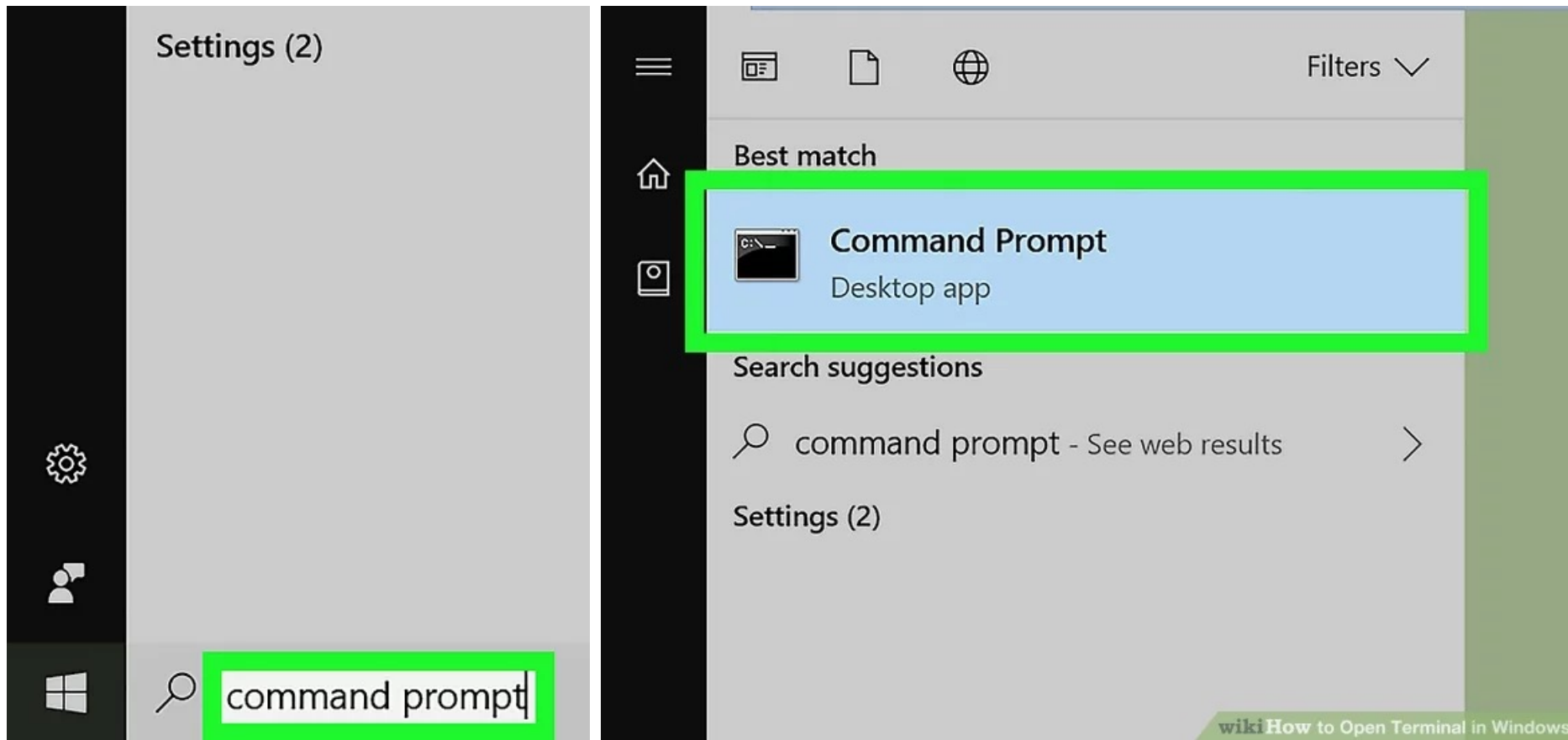
Lecturer: Ngo Tra

COMMAND LINE (CLI)

- Command line or CLI (command line interface)
- The CLI is the interface in which we enter commands for the computer to process.
- Windows:  Win → type **cmd** or **Command Prompt**

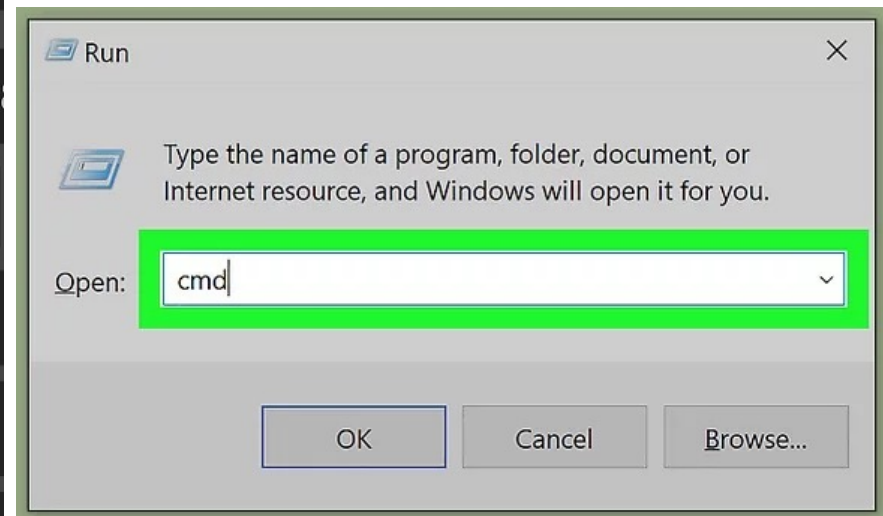
OPEN TERMINAL IN WINDOWS

■ Win → type **cmd** or **Command Prompt**



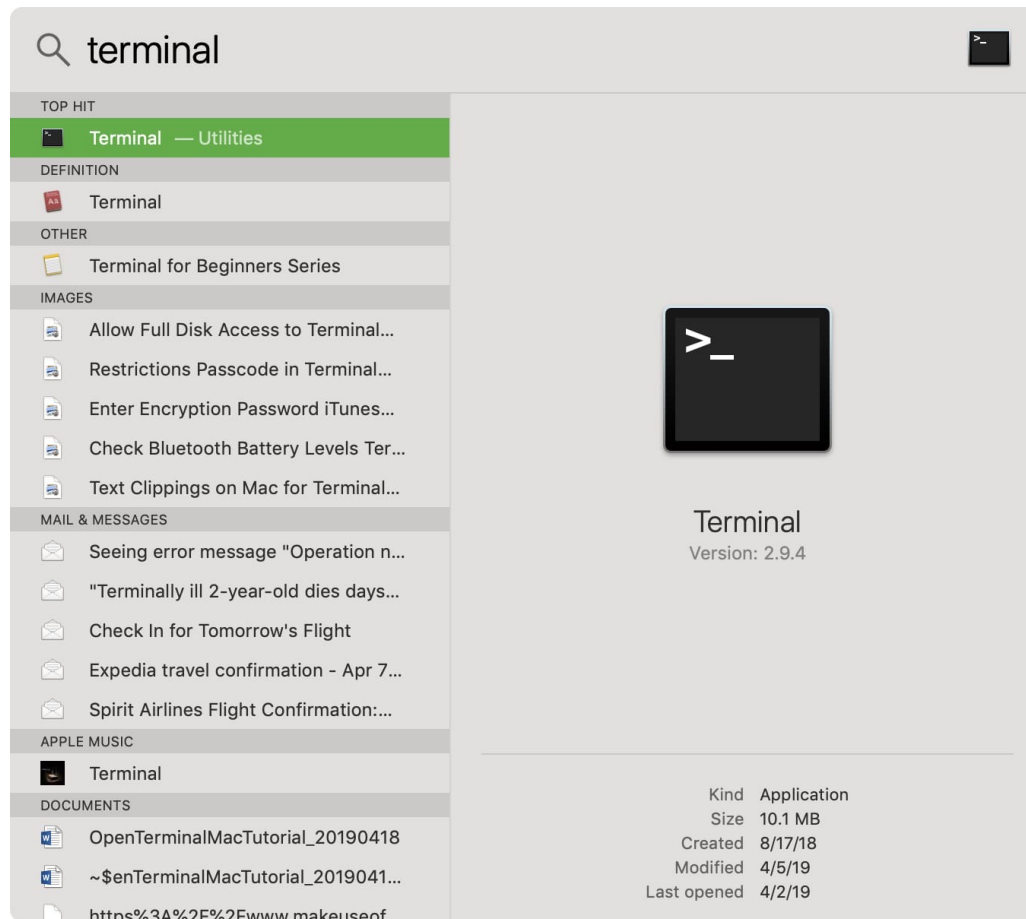
OPEN TERMINAL IN WINDOWS

■ **Win+R** → type **cmd** → Enter



OPEN TERMINAL IN MACOS

■ **Command + Space** → type **Terminal**



MOST COMMON AND USEFUL COMMANDS

- **echo** prints in the terminal whatever parameter we pass it.

```
echo Hello freeCodeCamp! // Output: Hello freeCodeCamp!
```

- **pwd (Mac)/ cd (Win)** stands for print working directory and it prints the "place" or directory we are currently at in the computer.

```
pwd // Output: /home/German
```

MOST COMMON AND USEFUL COMMANDS

- **ls (Mac)/ dir (Win)** presents you the contents of the directory you're currently in. It will present you with both the files and other directories your current directory contains.

```
ls // Output:
```

```
node_modules  package.json  package-lock.json  public  README.md  src
```

- **ls -a (Mac) or dir/a** It will also show you hidden files or directories. Like .git or .gitignore files

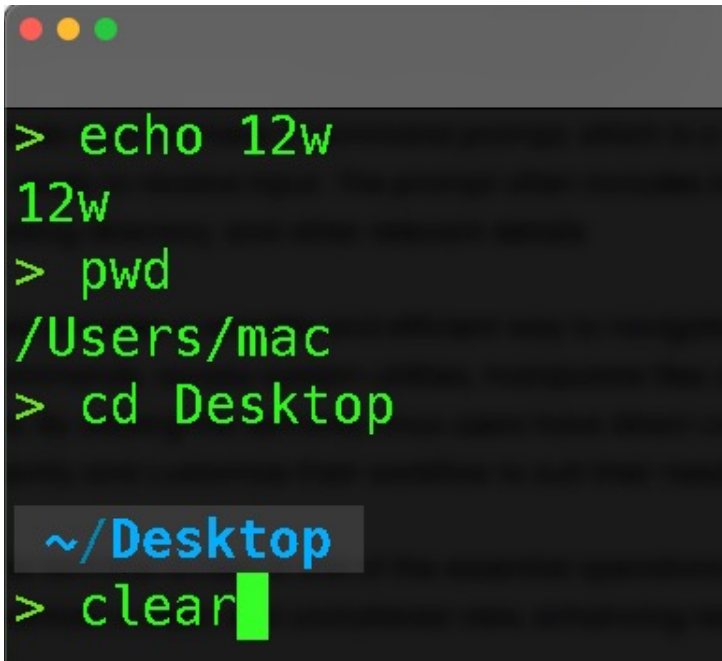
```
ls -a // Output:
```

```
.  .env  .gitignore  package.json  public  src  
.. .git  node_modules  package-lock.json  README.md
```

MOST COMMON AND USEFUL COMMANDS

■ Clear Terminal:

- Mac: **clear** or **command + L**
- Window: **clear**

A screenshot of a macOS Terminal window. The window has a dark gray title bar with three colored window control buttons (red, yellow, green) on the left. The terminal background is black with green text. The commands and output shown are: > echo 12w, 12w, > pwd, /Users/mac, > cd Desktop, ~/Desktop (highlighted in a gray box), and > clear (with a green cursor block at the end).

```
> echo 12w
12w
> pwd
/Users/mac
> cd Desktop

~/Desktop
> clear
```


MOST COMMON AND USEFUL COMMANDS

- **cd** is short for Change directory and it will take you from your current directory to another.
- While on my home directory, I can enter **cd Desktop** and it will take me to the Desktop Directory.
- If I want to go up one directory, meaning go to the directory that contains the current directory, I can enter **cd ..**

A terminal window with a dark background and a title bar with three colored buttons (red, yellow, green). It shows the command `> cd Desktop` in green text.

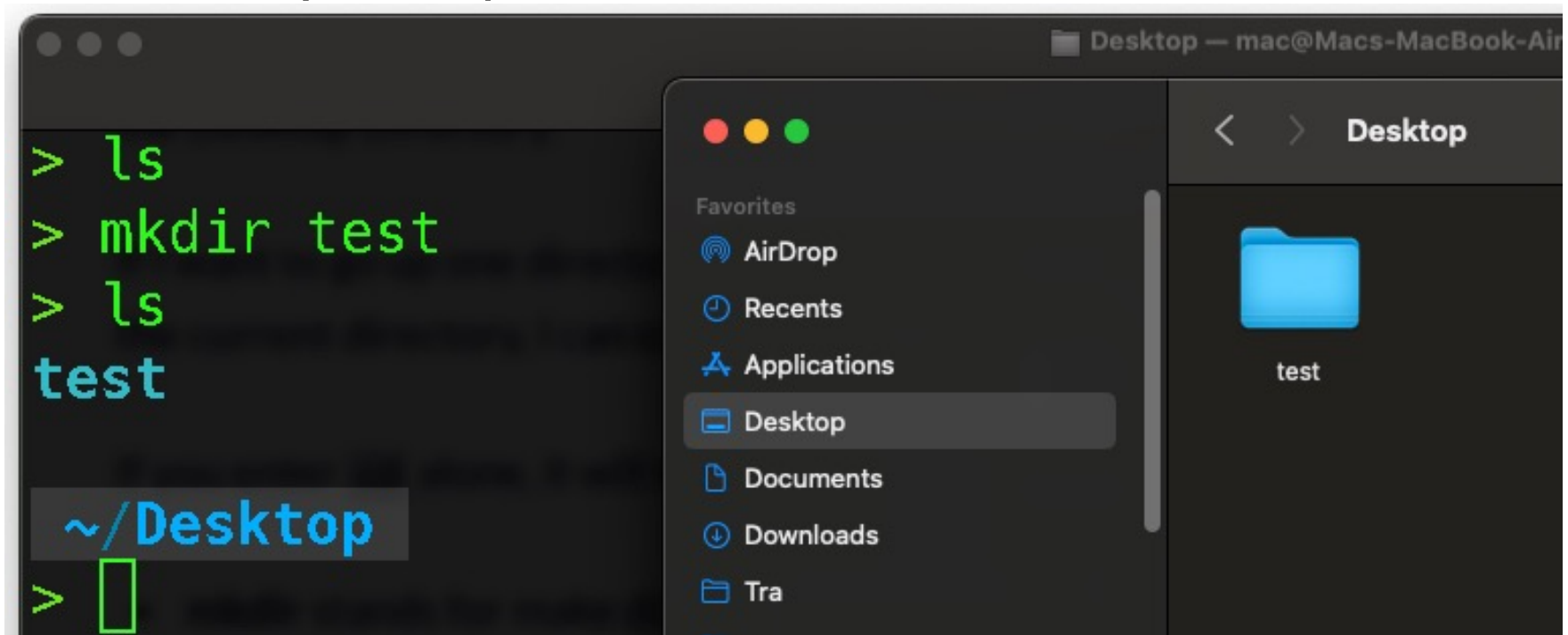
```
> cd Desktop
```

`~/Desktop`

```
> cd ..
```

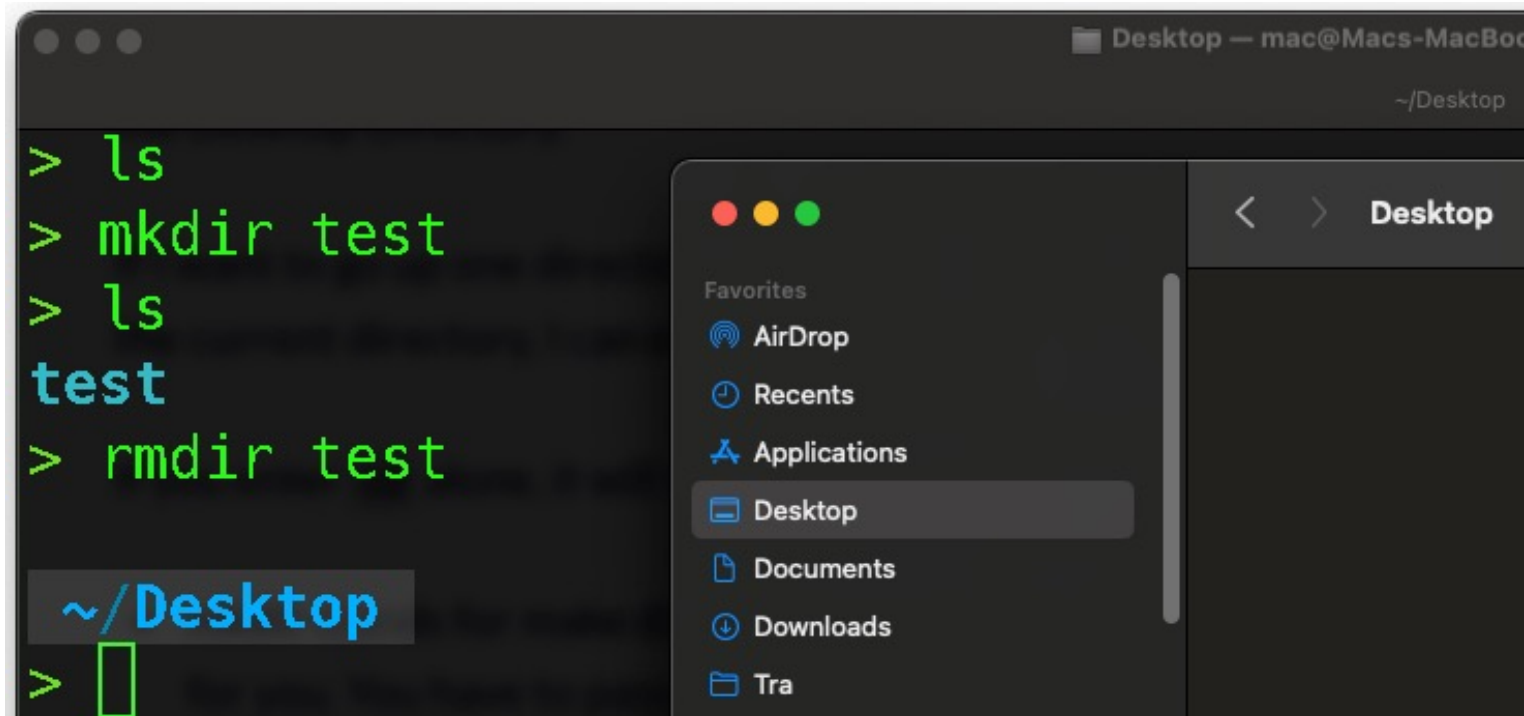
MOST COMMON AND USEFUL COMMANDS

- **mkdir** stands for make directory (folder) and it will create a new directory for you. You have to pass the command the directory name parameter.



MOST COMMON AND USEFUL COMMANDS

- **rmdir** stands for Remove directory and it does just that. It needs the directory name parameter just as mkdir: **rmdir test**



A screenshot of a macOS Terminal window titled "Desktop — mac@Macs-MacBoo" with the current directory set to "~/Desktop". The terminal shows the following sequence of commands and their output:

```
> ls
> mkdir test
> ls
test
> rmdir test
```

Below the terminal window, a blue prompt indicates the current directory: `~/Desktop`. To the right of the terminal, a portion of a Finder window is visible, showing the "Favorites" sidebar with "Desktop" selected.

MOST COMMON AND USEFUL COMMANDS

- **touch** allows you to create an empty file in your current directory. As parameters it takes the file name, like **touch test.txt**.
- **rm** allows you to delete files, in the same way **rmdir** allows you to remove directories. **rm test.txt**

MOST COMMON AND USEFUL COMMANDS

- **cp** allows you to copy files or directories. This command takes two parameters: the first one is the file or directory you want to copy, and the second one is the destination of your copy (where do you want to copy your file/directory to).

```
cp test.txt testCopy.txt
```

```
cp test.txt ../testFolder/
```

```
cp test.txt ../testFolder/testCopy.txt
```

MOST COMMON AND USEFUL COMMANDS

- **mv** is short for move, and lets us move a file or directory from one place to another. That is, create it in a new directory and delete it in the previous one (same as you could do by cutting and pasting).

```
mv test.txt ./testFolder/
```

```
mv test.txt ./testFolder/testCopy.txt
```

MOST COMMON AND USEFUL COMMANDS

- **head** allows you to view the beginning of a file or piped data directly from the terminal.

```
head test.txt // Output:  
this is the beginning of my test file
```

- **tail** works the same but it will show you the end of the file.

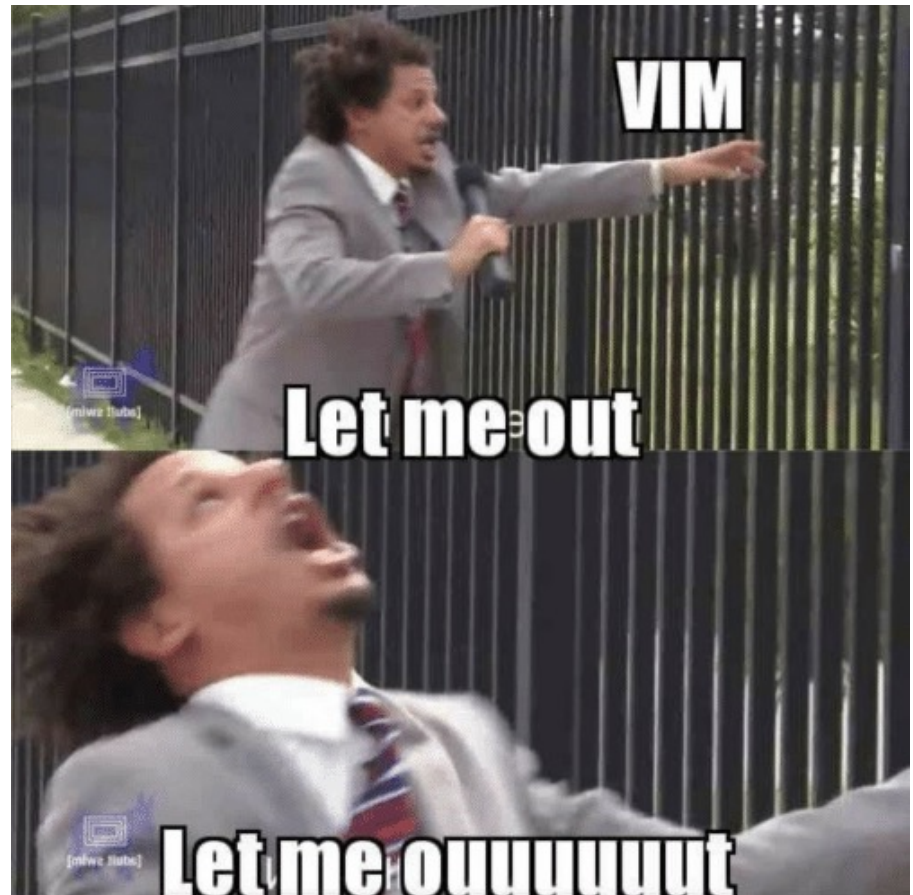
```
tail test.txt // Output:  
  
this is the end of my test file
```

MOST COMMON AND USEFUL COMMANDS

- **code** will open your default code editor. If you enter the command alone, it just opens the editor with the latest file/directory you opened.
- You can also open a given file by passing it as parameter:
code test.txt
- You can also open a given folder (project) by: **cd testFolder**
→ **code .**

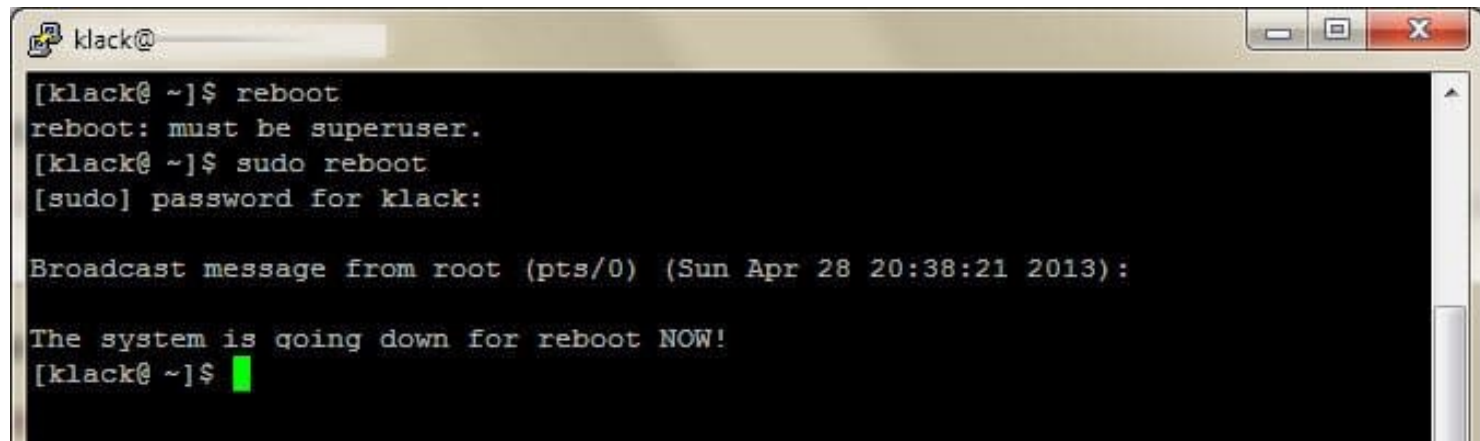
LET ME OUTTTT

- Type **:q!** and hit enter
- **ctrl+c** allows you to exit the current process the terminal is running.
- For example, if you're creating a react app with **npx create-react-app** and want to cancel the build at some point, just hit **ctrl+c** and it will stop.



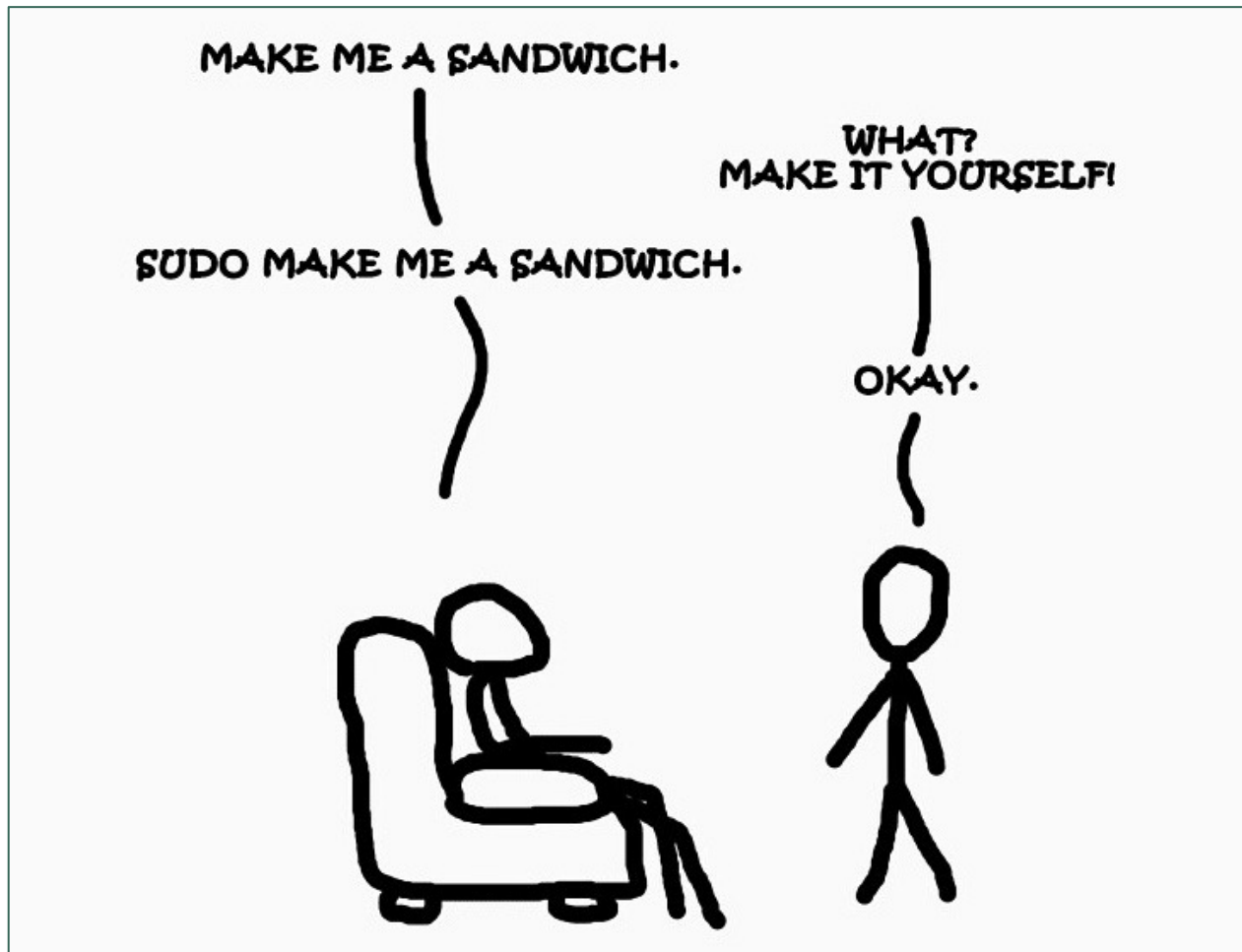
SUDO (SUPERUSER DO)

- It allows the permitted user (the administrator user) to run commands in the Mac Terminal as a superuser or another user with extra security privileges.
- In windows, you just need to open the command line



```
klack@  
[klack@ ~]$ reboot  
reboot: must be superuser.  
[klack@ ~]$ sudo reboot  
[sudo] password for klack:  
  
Broadcast message from root (pts/0) (Sun Apr 28 20:38:21 2013):  
  
The system is going down for reboot NOW!  
[klack@ ~]$
```

SUDO (SUPERUSER DO)



ABOUT VERSION CONTROL

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to:
 - revert selected files back to a previous state
 - revert the entire project back to a previous state
 - compare changes over time
 - see who last modified something that might be causing a problem
 - who introduced an issue and when
- Using a Version Control system (VCS) also generally means that if you screw things up or lose files, you can easily recover.

Local Computer

Checkout

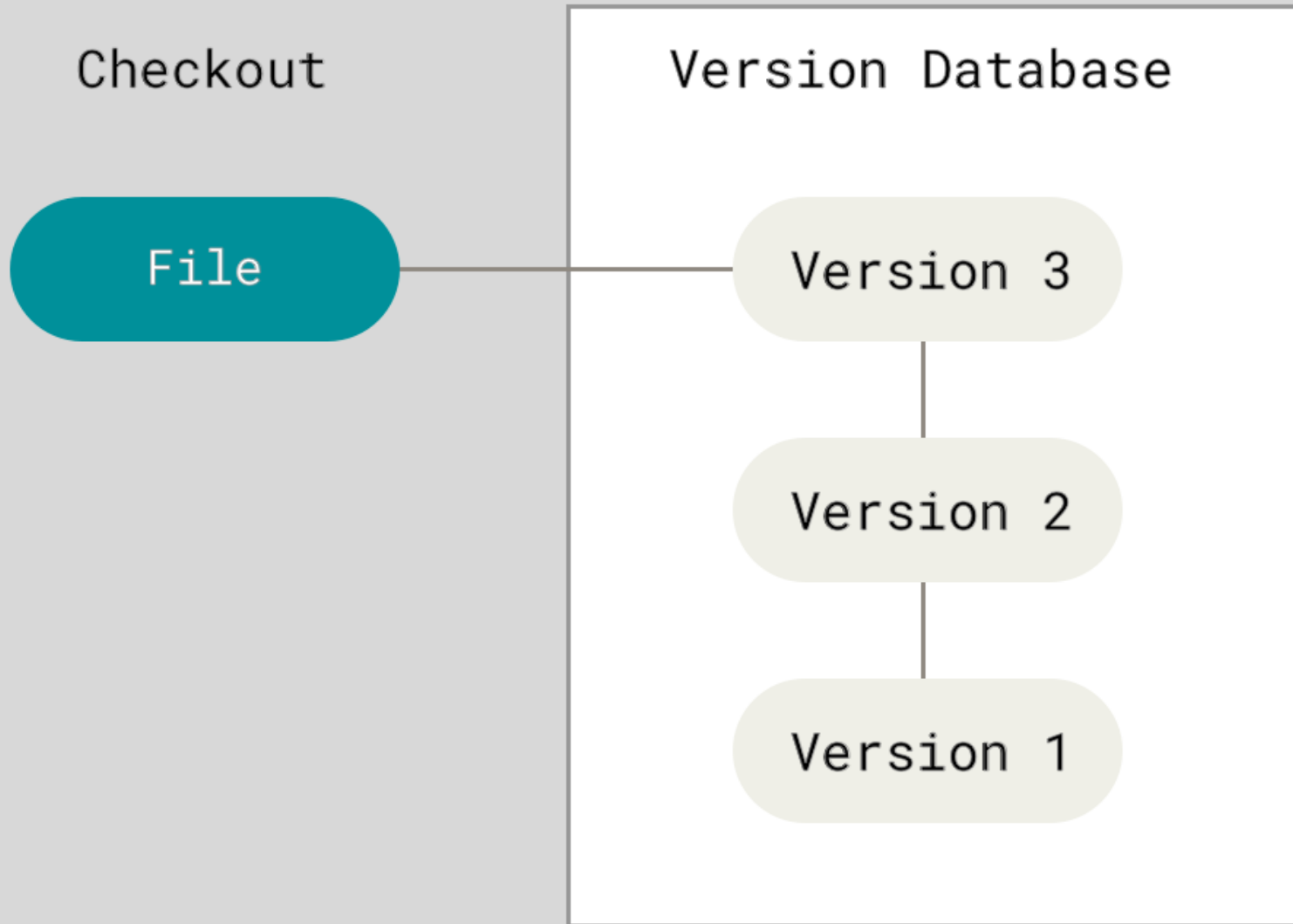
File

Version Database

Version 3

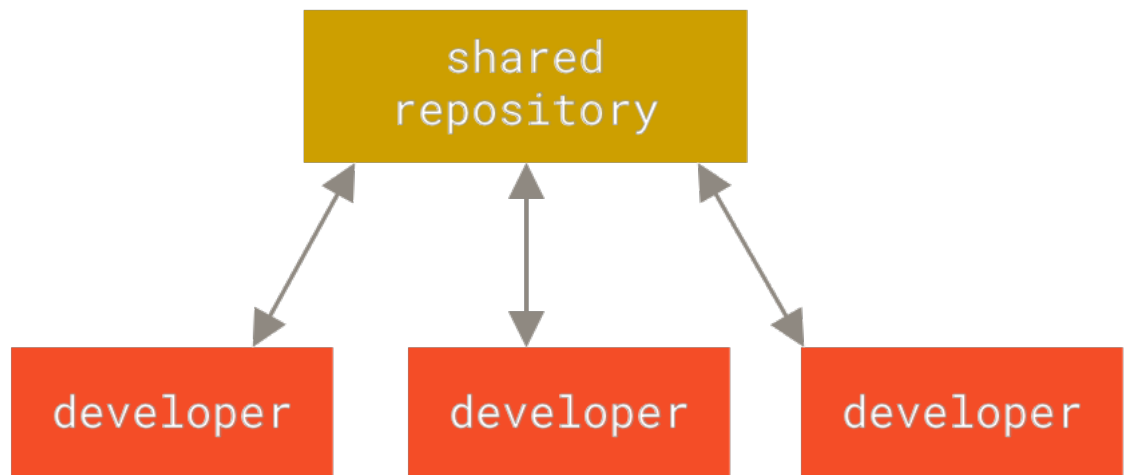
Version 2

Version 1



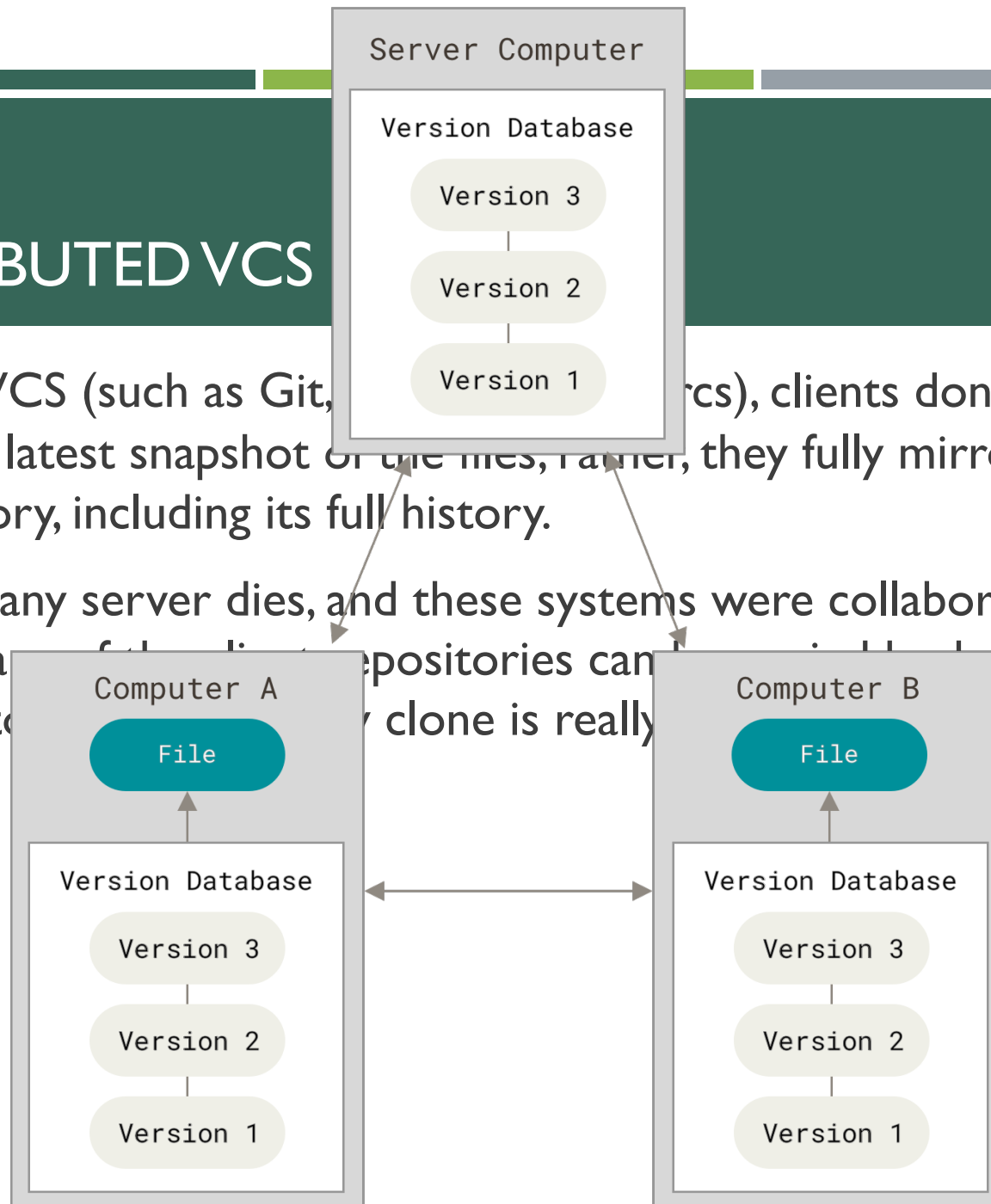
CENTRALIZED VCS

- The next major issue that people encounter is that they need to collaborate with developers on other systems.
- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- Benefits: everyone knows to a certain degree what everyone else on the project is doing
- Drawback: single point failure



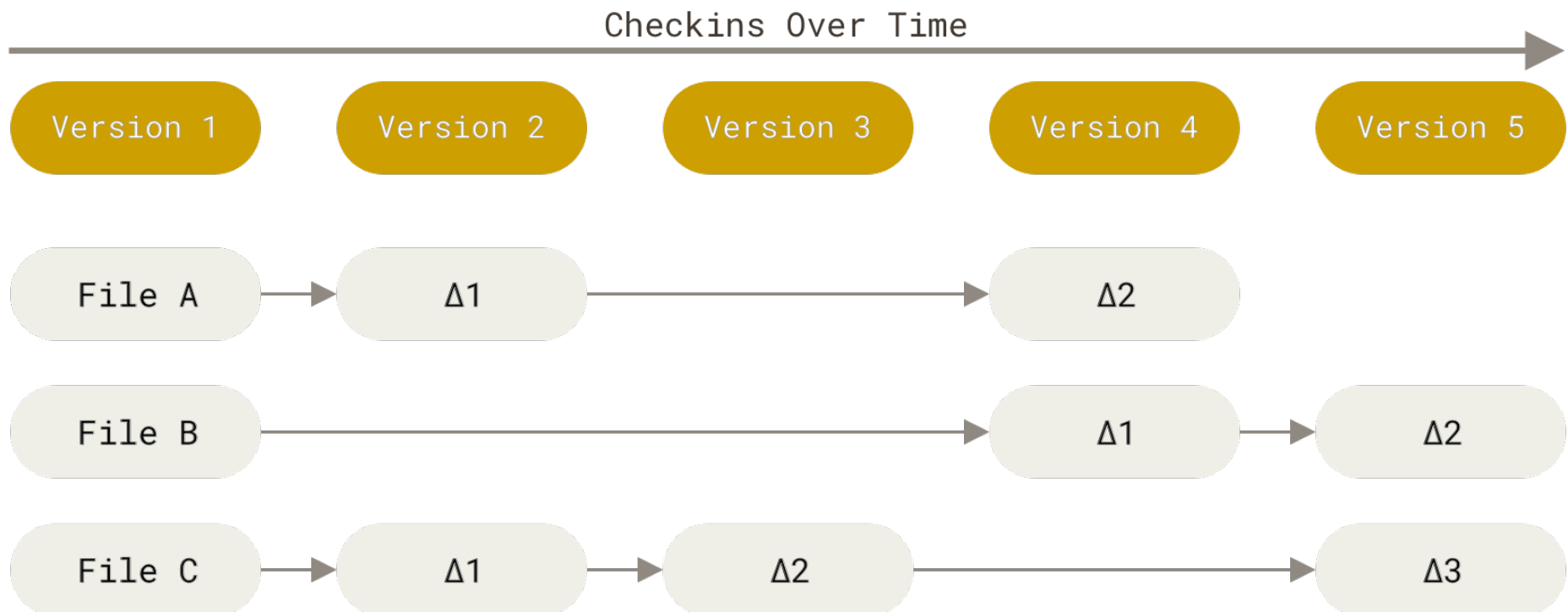
DISTRIBUTED VCS

- In a DVCS (such as Git, Mercurial, etc), clients don't just check out the latest snapshot of the files, rather, they fully mirror the repository, including its full history.
- Thus, if any server dies, and these systems were collaborating via that server, all client repositories can still sync up to the server to get a clone is really all the data.



WHAT IS GIT?

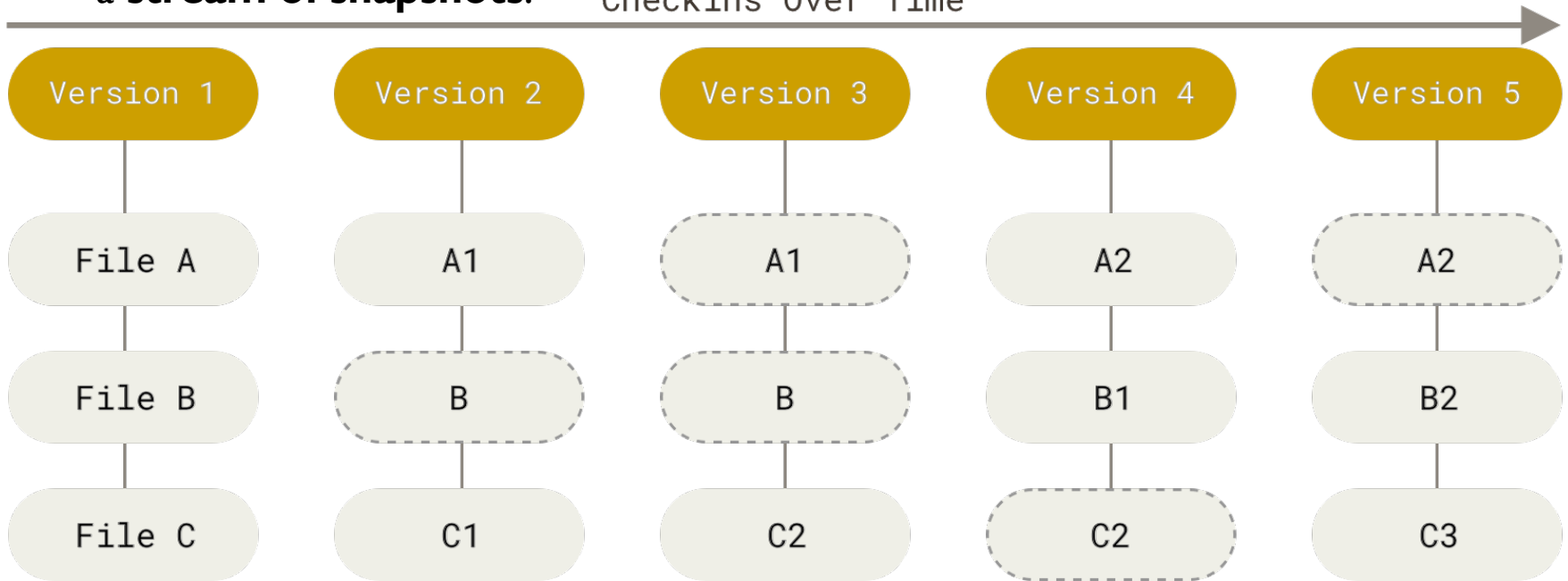
■ Other VCS Snapshots



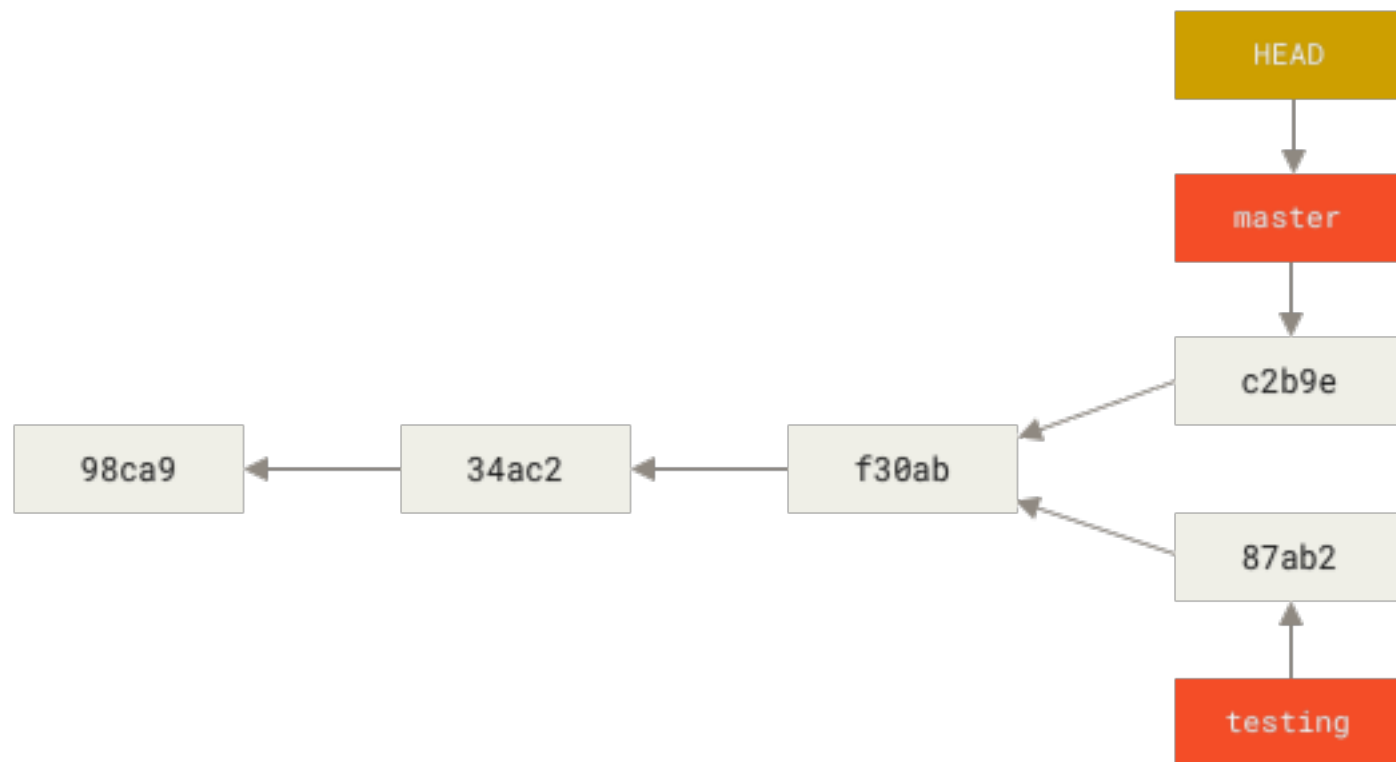
GIT SNAPSHOTS

- With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a **stream of snapshots**.

Checkins Over Time



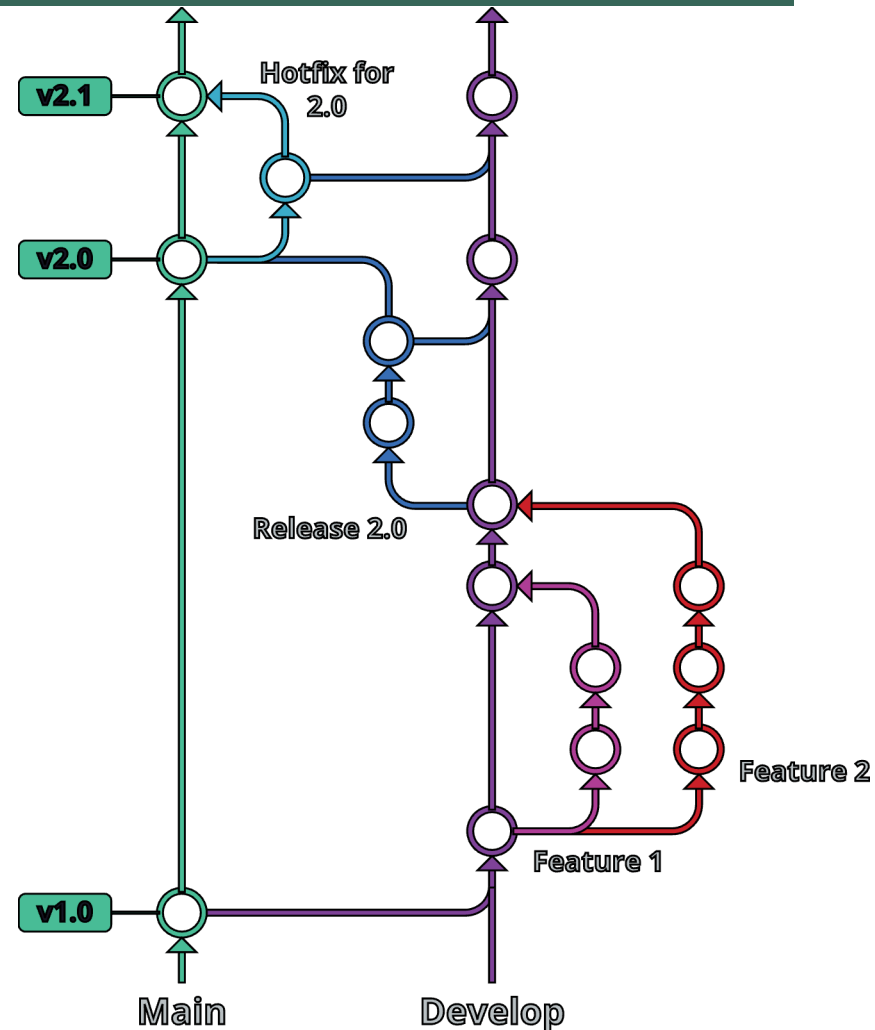
GIT BRANCHES



GIT FLOW

- Fundamentally, Git flow involves isolating your work into different types of Git Branches.
- In the Git flow workflow, there are five different branch types:

- Main
- Develop
- Feature
- Release
- Hotfix



GIT FLOW – MAIN BRANCH

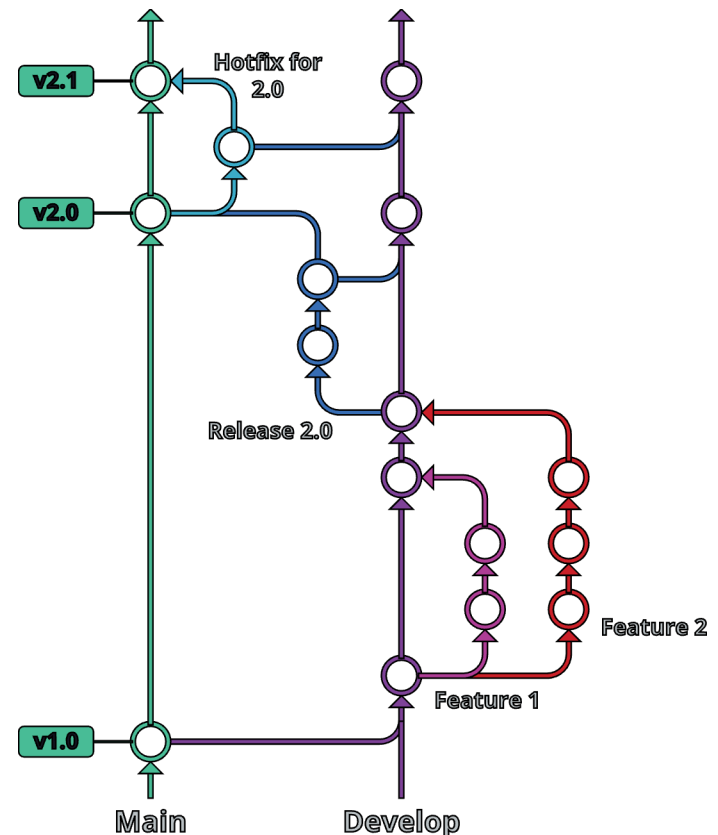
- *Please note: the main branch is commonly referred to as “master”; we have made an intentional decision to avoid that outdated term and have chosen to use “main” instead.*
- The purpose of the main branch in the Git flow workflow is to contain production-ready code that can be released.
- In Git flow, the main branch is created at the start of a project and is maintained throughout the development process. The branch can be tagged at various commits in order to signify different versions or releases of the code, and other branches will be merged into the main branch after they have been sufficiently vetted and tested.

GIT FLOW – DEVELOP BRANCH

- The develop branch is created at the start of a project and is maintained throughout the development process, and contains pre-production code with newly developed features that are in the process of being tested.
- Newly-created features should be based off the develop branch, and then merged back in when ready for testing.

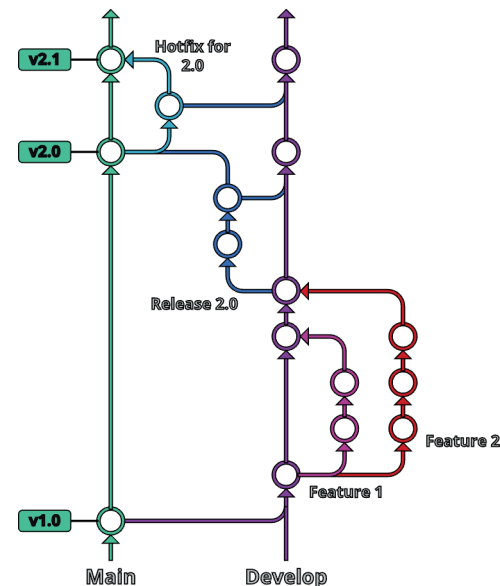
GIT FLOW – SUPPORTING BRANCH

- When developing with Git flow, there are three types of supporting branches with different intended purposes: feature, release, and hotfix.



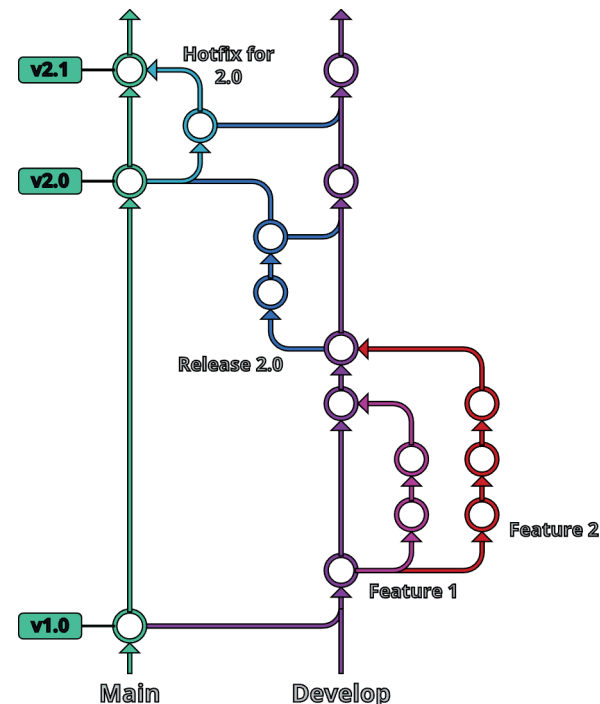
GIT FLOW – FEATURE BRANCH

- The feature branch is the most common type of branch in the Git flow workflow. It is used when adding new features to your code.
- When working on a new feature, you will start a feature branch off the develop branch, and then merge your changes back into the develop branch when the feature is completed and properly reviewed.



GIT FLOW – RELEASE BRANCH

- The release branch should be used when preparing new production releases. Typically, the work being performed on release branches concerns finishing touches and minor bugs specific to releasing new code, with code that should be addressed separately from the main develop branch.



GIT FLOW – HOTFIX BRANCH

- In Git flow, the hotfix branch is used to quickly address necessary changes in your main branch.
- The base of the hotfix branch should be your main branch and should be merged back into both the main and develop branches. Merging the changes from your hotfix branch back into the develop branch is critical to ensure the fix persists the next time the main branch is released.



GIT

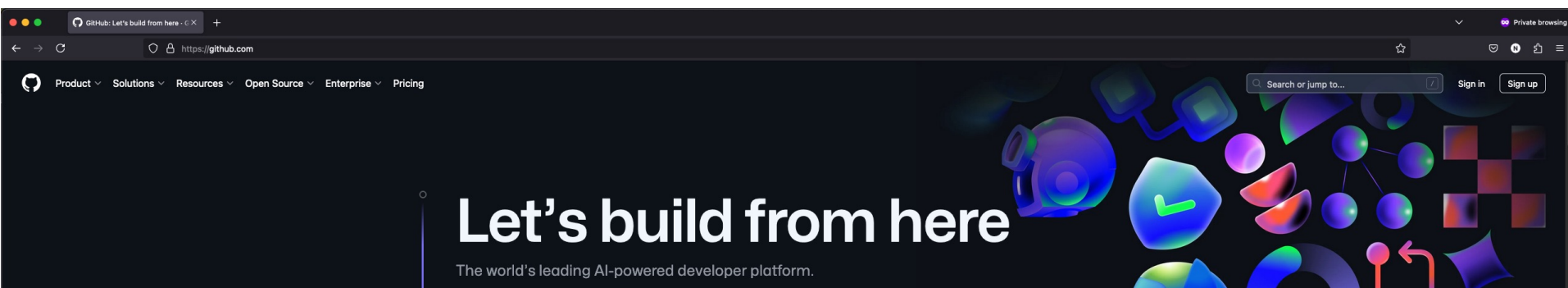
■ Installation:

- **Mac:** <https://git-scm.com/downloads/mac>
- **Window:** <https://gitforwindows.org/>

■ Type **git version** to verify Git was installed

GIT

■ Create a Github account: <https://github.com/>



GIT CONFIG

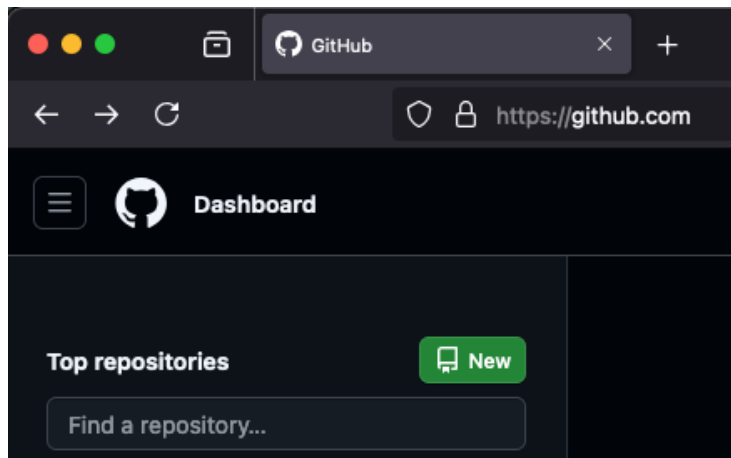
- You can view all of your settings and where they are coming from using: **git config --list --show-origin**

```
$ git config --list --show-origin
```

- Setup your identity

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

GIT NEW PROJECT



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner *

Repository name *

 /

Great repository names are short and memorable. Need inspiration? How about [improved-octo-couscous](#) ?

Description (optional)

☒ **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: **None**


A license tells others what they can and can't do with your code. [Learn more about licenses](#).

i You are creating a public repository in your personal account.


Create repository

GIT NEW PROJECT

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 **fgw** Public


[Pin](#) [Unwatch](#) 1 [Fork](#) 0 [Star](#) 0



Set up GitHub Copilot

Use GitHub's AI pair programmer to autocomplete suggestions as you code.

[Get started with GitHub Copilot](#)



Add collaborators to this repository

Search for people using their GitHub username or email address.

[Invite collaborators](#)

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) `https://github.com/fgw.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# fgw" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/fgw.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/fgw.git
git branch -M main
git push -u origin main
```

GIT CLI

- **cd** into your project folder
- **git init** will create a new local repository for you.

```
git init // output:
```

```
Initialized empty Git repository in /home/German/Desktop/testFolder/.git/
```

- **git add** adds one or more files to staging. You can either detail a specific file to add to staging or add all changed files by typing **git add .**

GIT CLI

- **git commit** commits your changes to the repository. Commits must always be accompanied by the -m flag and commit message.

```
git commit -m 'This is a test commit' // output:  
[master (root-commit) 6101dfe] This is a test commit  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 test.js
```


GIT CLI

- And as last step, we rename our master branch to main, add the remote origin we just obtained, and push our code to GitHub

```
git branch -M main  
git remote add origin $GIT_URL  
git push -u origin main
```

GIT SSH KEY

- If you don't have a private token yet, you can generate it in GitHub in **User icon > Settings > SSH and GPG keys**
- <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

GIT STATUS

- **git status** tells you what branch are you currently on and whether you have changes to commit or not.

```
git status // output:  
On branch master  
nothing to commit, working tree clean
```

GIT CLONE

- **git clone** allows you to clone (copy) a repository into the directory you're currently in. Keep in mind you can clone both remote repositories (in GitHub, GitLab, and so on) and local repositories (those that are stored in your computer).

```
git clone https://github.com/coccagerman/MazeGenerator.git // output:  
Cloning into 'MazeGenerator'...  
remote: Enumerating objects: 15, done.  
remote: Counting objects: 100% (15/15), done.  
remote: Compressing objects: 100% (15/15), done.  
remote: Total 15 (delta 1), reused 11 (delta 0), pack-reused 0  
Unpacking objects: 100% (15/15), done.
```

GIT CLONE

The screenshot shows the GitHub interface for the 'react' repository. At the top, the repository name 'react' is displayed with a 'Public' badge. To the right, there is a 'Watch' button and a count of '6619' watchers. Below this, the main branch 'main' is selected, with '324 Branches' and '143 Tags' also visible. A search bar labeled 'Go to file' is present. The 'Code' button is highlighted in green and has a dropdown menu open. This menu has two tabs: 'Local' and 'Codespaces'. Under the 'Local' tab, there is a 'Clone' option with a question mark icon. Below 'Clone', there are three sub-options: 'HTTPS' (which is selected and underlined), 'SSH', and 'GitHub CLI'. A text input field contains the URL 'https://github.com/facebook/react.git', and a copy icon is to its right. Below the input field, it says 'Clone using the web URL.'. There are also two other options in the menu: 'Open with GitHub Desktop' and 'Download ZIP'. In the background, the repository's file list is visible, showing folders like '.codesandbox', '.github', 'compiler', 'fixtures', 'packages', 'scripts', and a file '.editorconfig'.

react Public

Watch 6619

main 324 Branches 143 Tags

Go to file

Add file

<> Code

Local Codespaces

Clone ?

HTTPS SSH GitHub CLI

https://github.com/facebook/react.git

Clone using the web URL.

Open with GitHub Desktop

Download ZIP

.codesandbox Codesandbox: upg

.github [playground] Deco

compiler [compiler] Rename

fixtures [Flight] Enable Ser

packages Define HostInstanc

scripts fix[scripts/devtools

.editorconfig https link to editorconfig.org (#18421) 4 years ago

GIT PULL

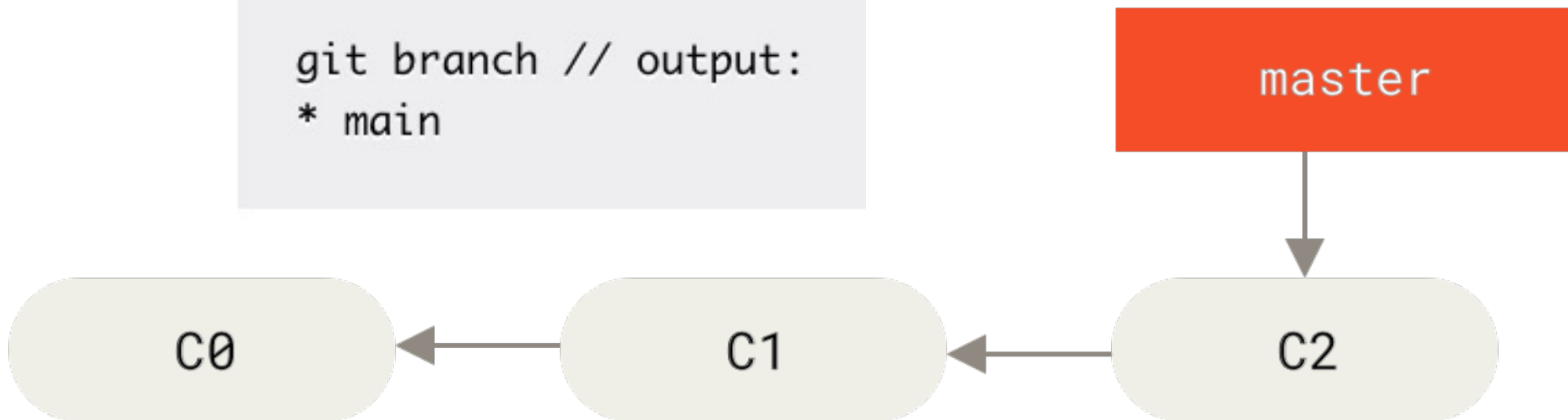
- **git pull** pulls (downloads) the code from your remote repository and combines it with your local repo. This is particularly useful when working in teams, when many developers are working on the same code base. In this case each developer periodically pulls from the remote repo in order to work in a code base that includes the changes done by all the other devs.

```
git pull // output:  
Already up to date.
```

GIT BRANCH

- **git branch** lists all the available branches on your repo and tells you what branch you're currently on. If you want to create a new branch, you just have to add the new branch name as parameter like `git branch <branch name>`.

```
git branch // output:  
* main
```

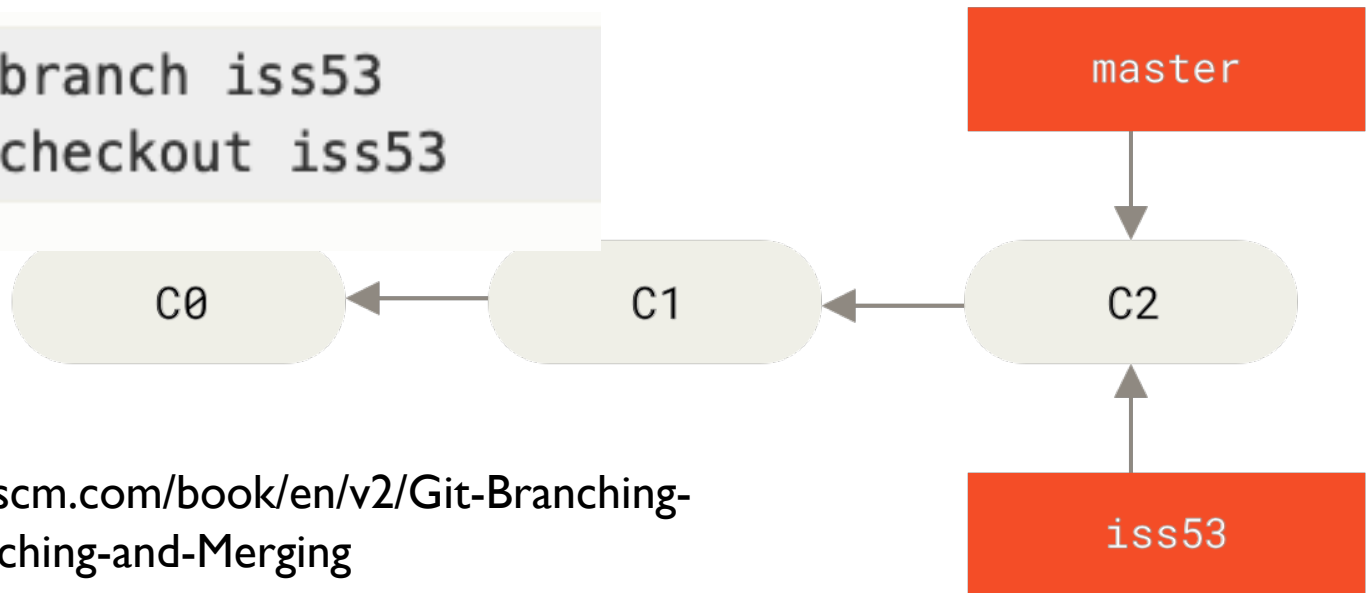


GIT BRANCH

- You've decided that you're going to work on issue #53 in whatever issue-tracking system your company uses. To create a new branch and switch to it at the same time, you can run the git checkout command with the -b switch:

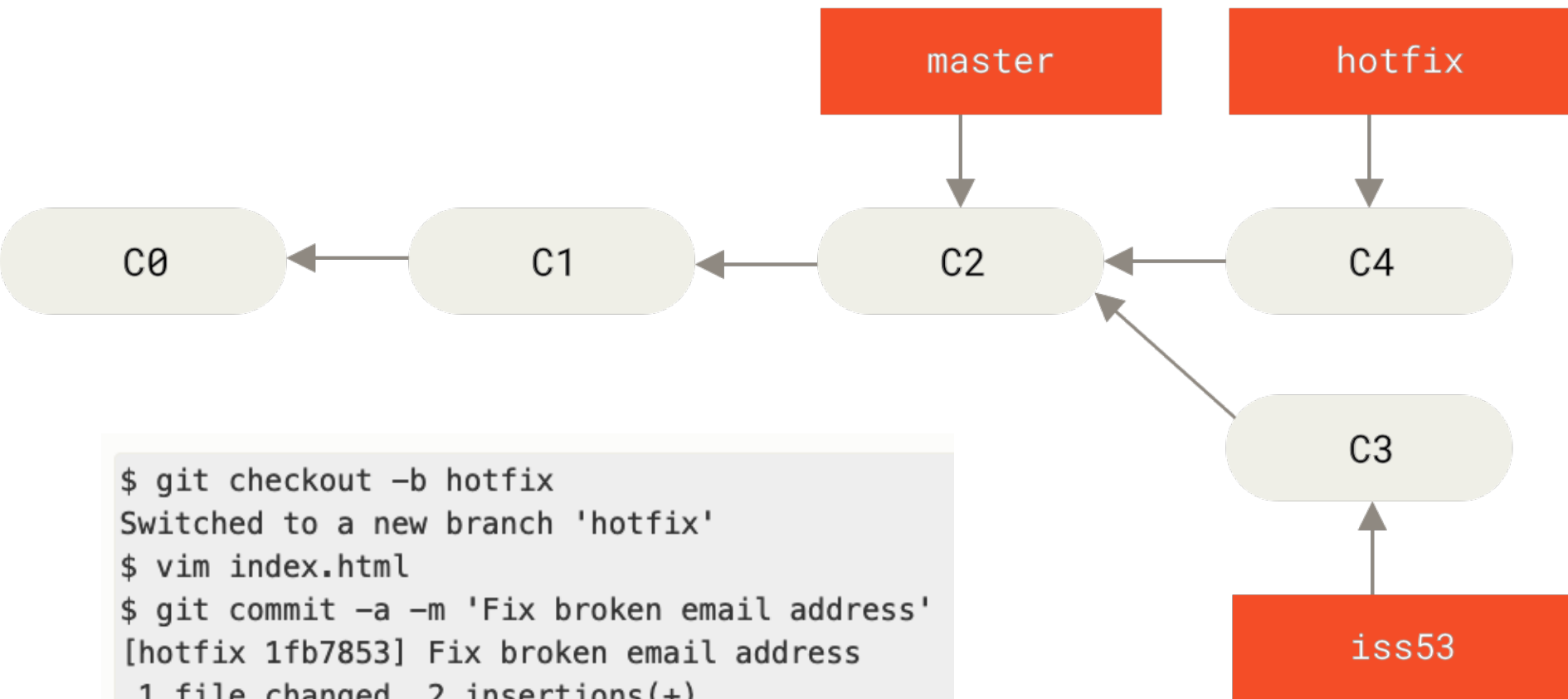
git checkout -b iss53

```
$ git branch iss53  
$ git checkout iss53
```



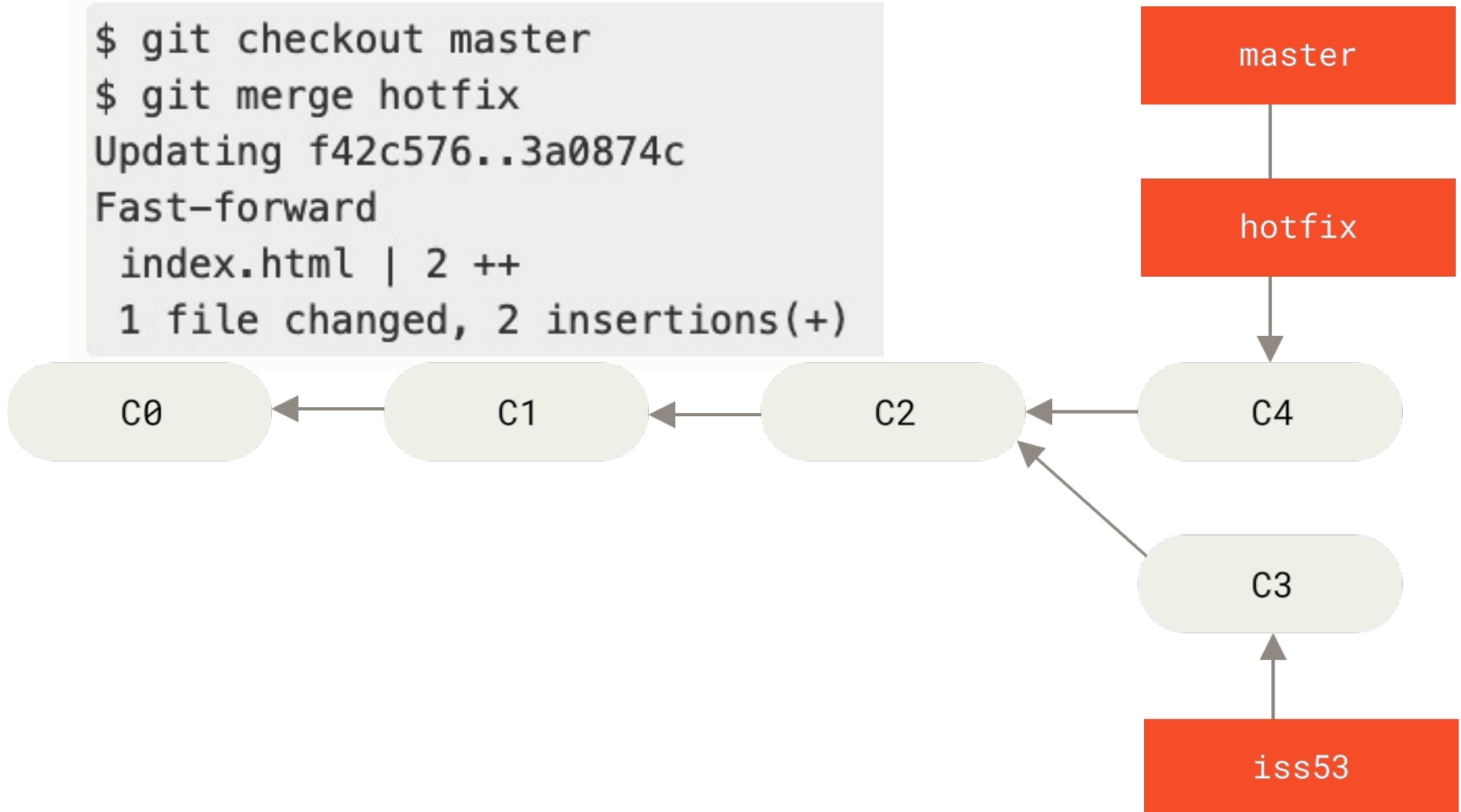
<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

GIT BRANCH

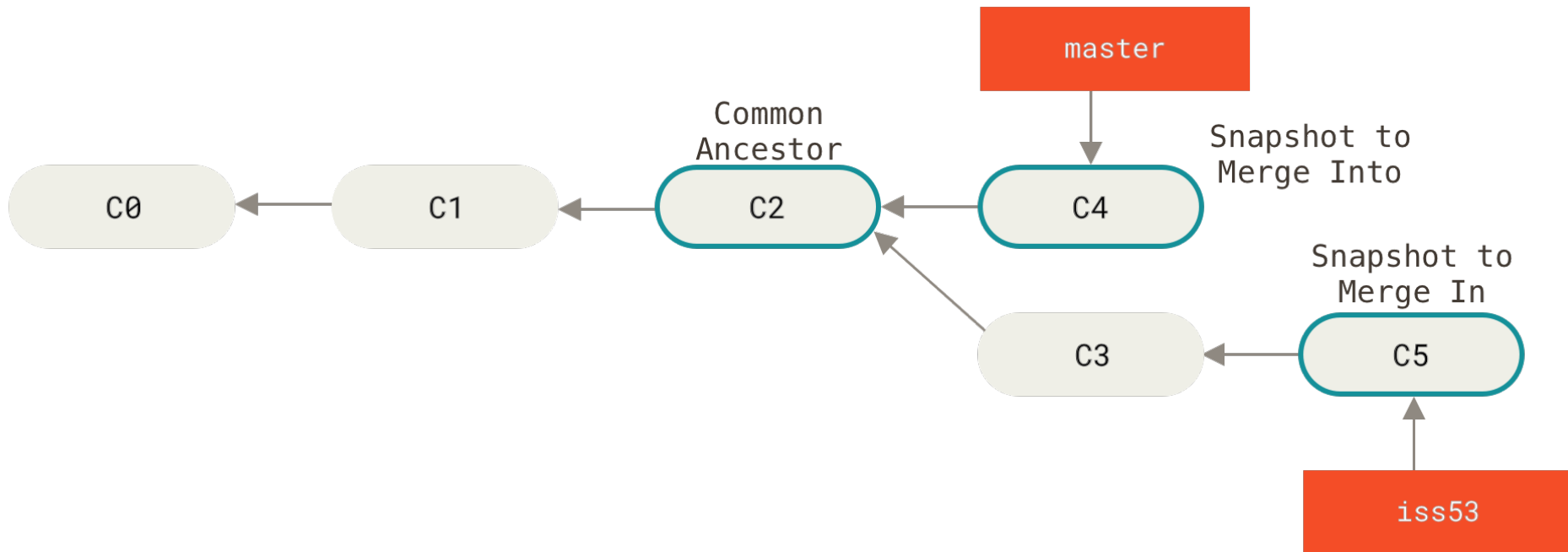


GIT MERGE

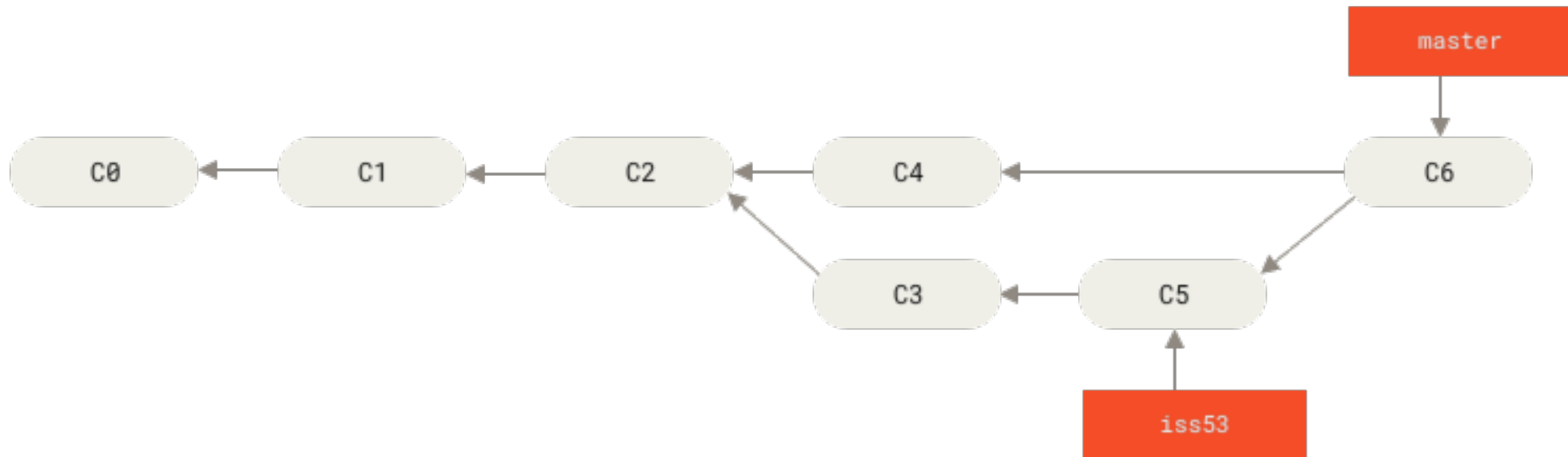
```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```



GIT MERGE



GIT MERGE



LEARNING GIT GAMES

<https://learngitbranching.js.org/>

The screenshot displays the 'Learn Git Branching' website in a web browser. The browser's address bar shows 'learngitbranching.js.org'. The website has a light blue background. On the left, there is a terminal window titled 'Learn Git Branching' with a green header bar. The terminal shows the following commands and their outputs:

```
$ level intro1
$ hint
Just type in 'git commit' twice to finish!
$ delay 2000
$ show goal
$ objective
```

Each command is followed by a blue box with a checkmark. The main area of the website features a commit graph. It shows two commits, C0 and C1, connected by an upward arrow. A pink arrow points from C1 to a label 'main*'. On the right, there is a pink window titled 'Goal To Reach' with the text 'You can hide this window with "hide goal"'. Inside this window, a vertical commit graph shows four commits: C0, C1, C2, and C3, connected by upward arrows. A pink arrow points from C3 to a label 'main*'. The browser's bookmark bar at the top shows several folders: FGW, Broove, Paper, LuanVan, Nisse, and PhD. The bottom right corner of the browser window shows a status bar with a question mark, a language selector (A2), and a refresh icon.