



Git/Github

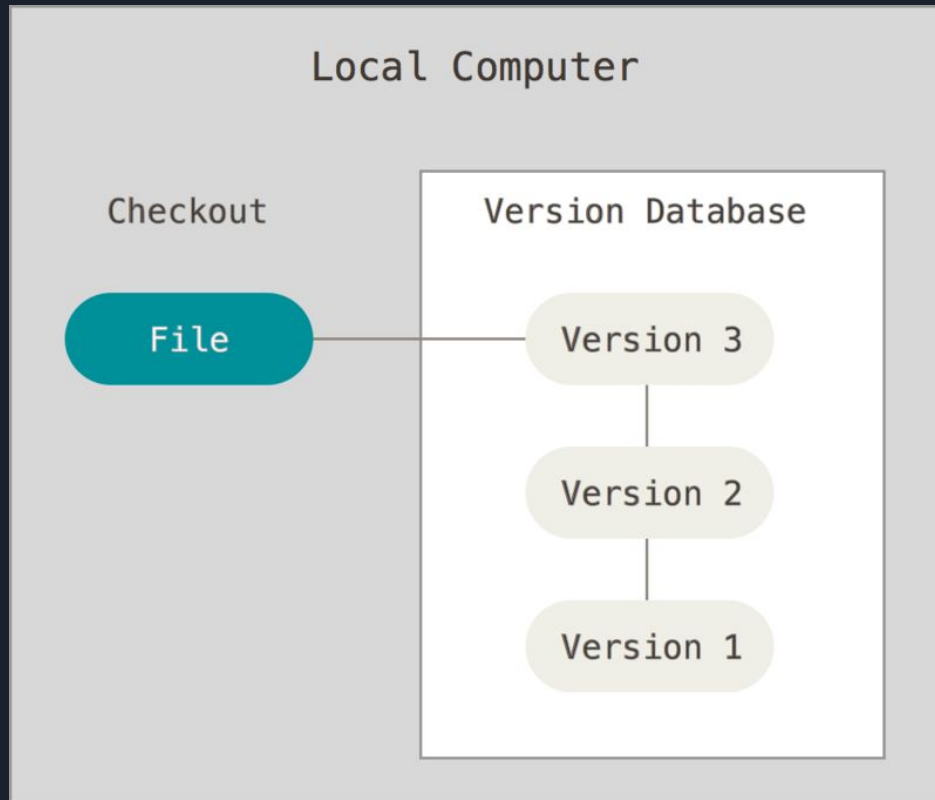
Version Control



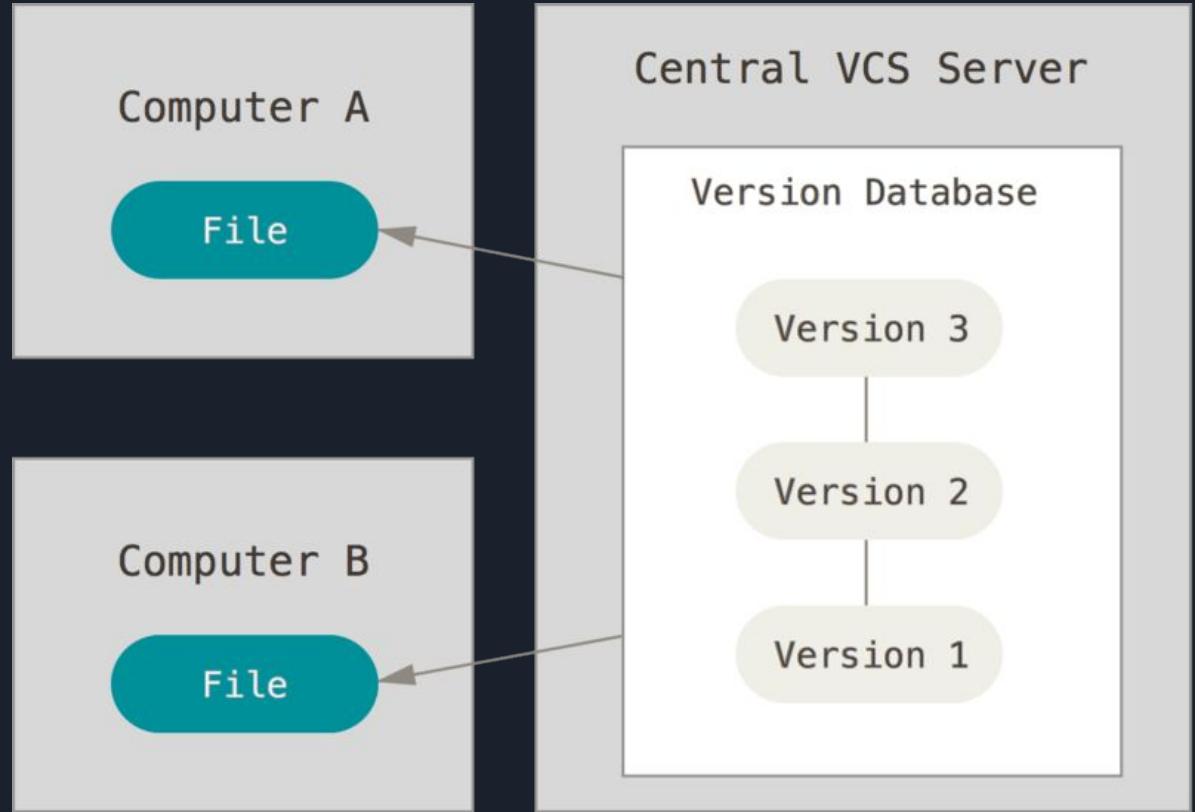


Version control system (VCS) **records changes** to a file or set of files over time so that you can recall **specific versions** later.

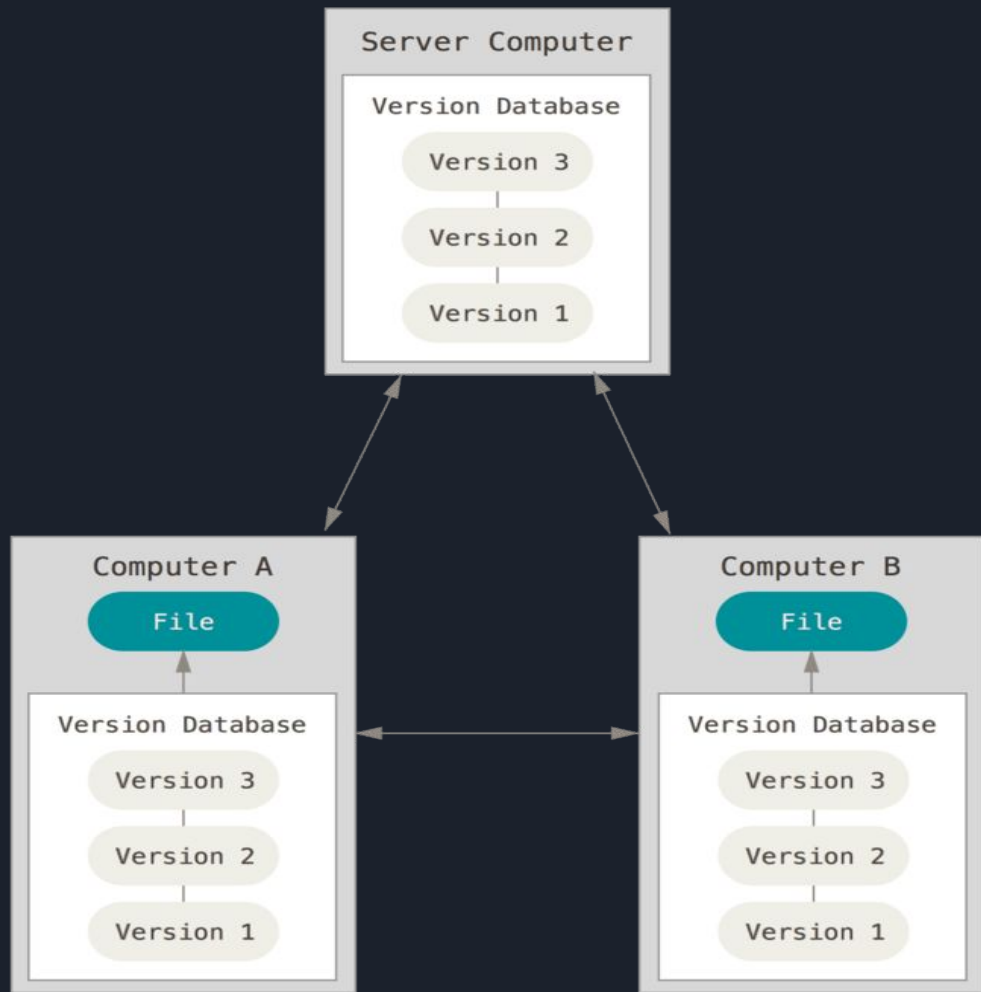
Local file management



Centralized version control systems



Distributed version control systems



Git






Git

Git is a free and open source **distributed version control system**.

Designed with these goals:

Speed, open source, efficiency, support for **parallel branches**, simple design, able to **handle large projects**.

A man with glasses and a green shirt is speaking, gesturing with his right hand. The background is a dark red curtain.

que fue creado para poder
mantener mi primer gran proyecto.

Version control systems



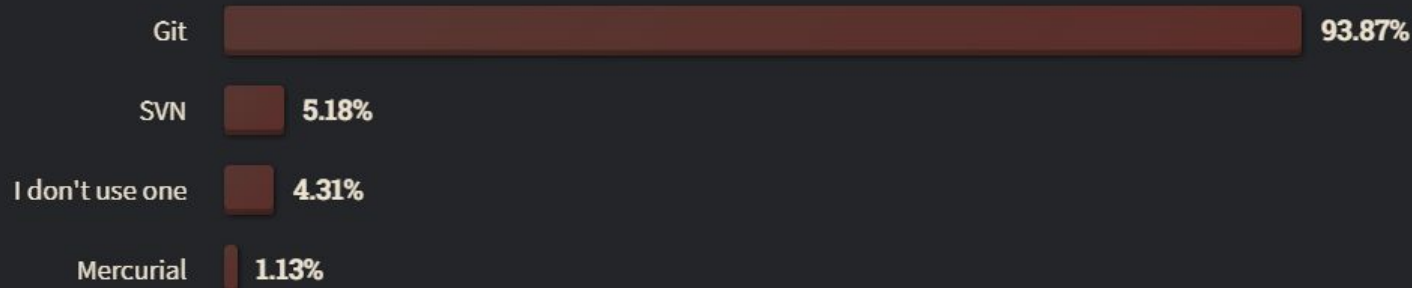
No other technology is as widely used as Git. Especially among Professional Developers. But for those learning to code, 17% still do not use a version control system.

All Respondents

Professional Developers

Learning to Code

71,379 responses



Version control systems



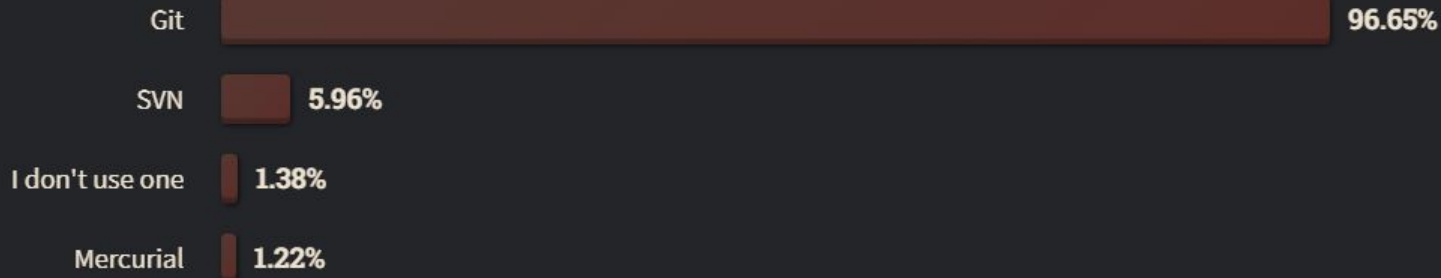
No other technology is as widely used as Git. Especially among Professional Developers. But for those learning to code, 17% still do not use a version control system.

All Respondents

Professional Developers

Learning to Code

53,374 responses



Version control systems



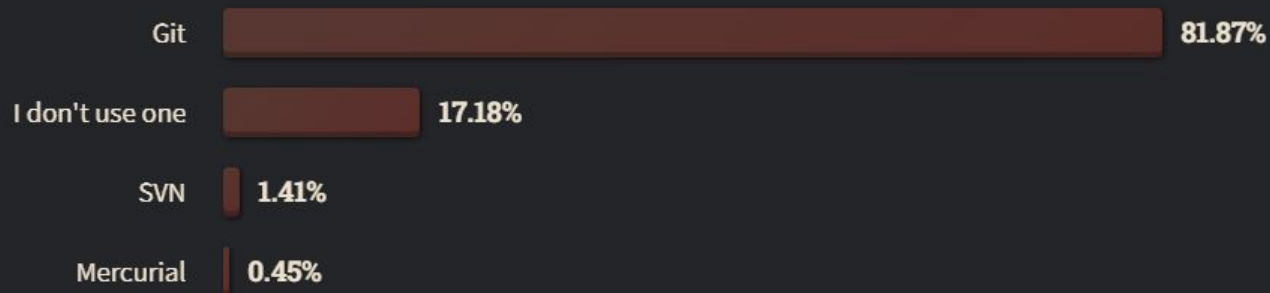
No other technology is as widely used as Git. Especially among Professional Developers. But for those learning to code, 17% still do not use a version control system.

All Respondents

Professional Developers

Learning to Code

6,157 responses



Interacting with version control systems



The command line is the primary way developers interact with their version control system

All Respondents

Professional Developers

Learning to Code

68,156 responses



Interacting with version control systems



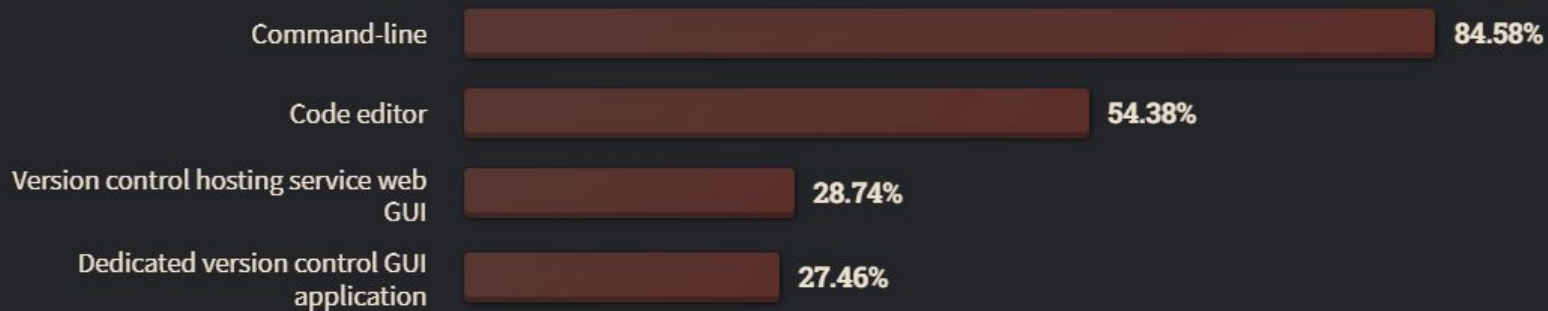
The command line is the primary way developers interact with their version control system

All Respondents

Professional Developers

Learning to Code

52,556 responses



Interacting with version control systems



The command line is the primary way developers interact with their version control system

All Respondents

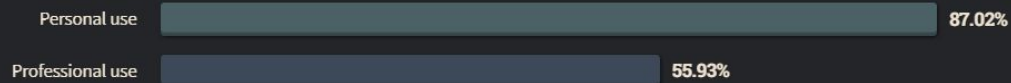
Professional Developers

Learning to Code

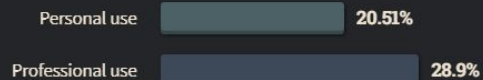
5,054 responses



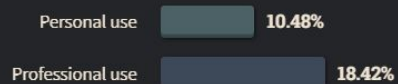
GitHub



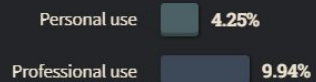
GitLab



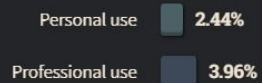
Bitbucket



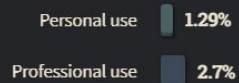
Azure Repos



Custom built solution



AWS CodeCommit





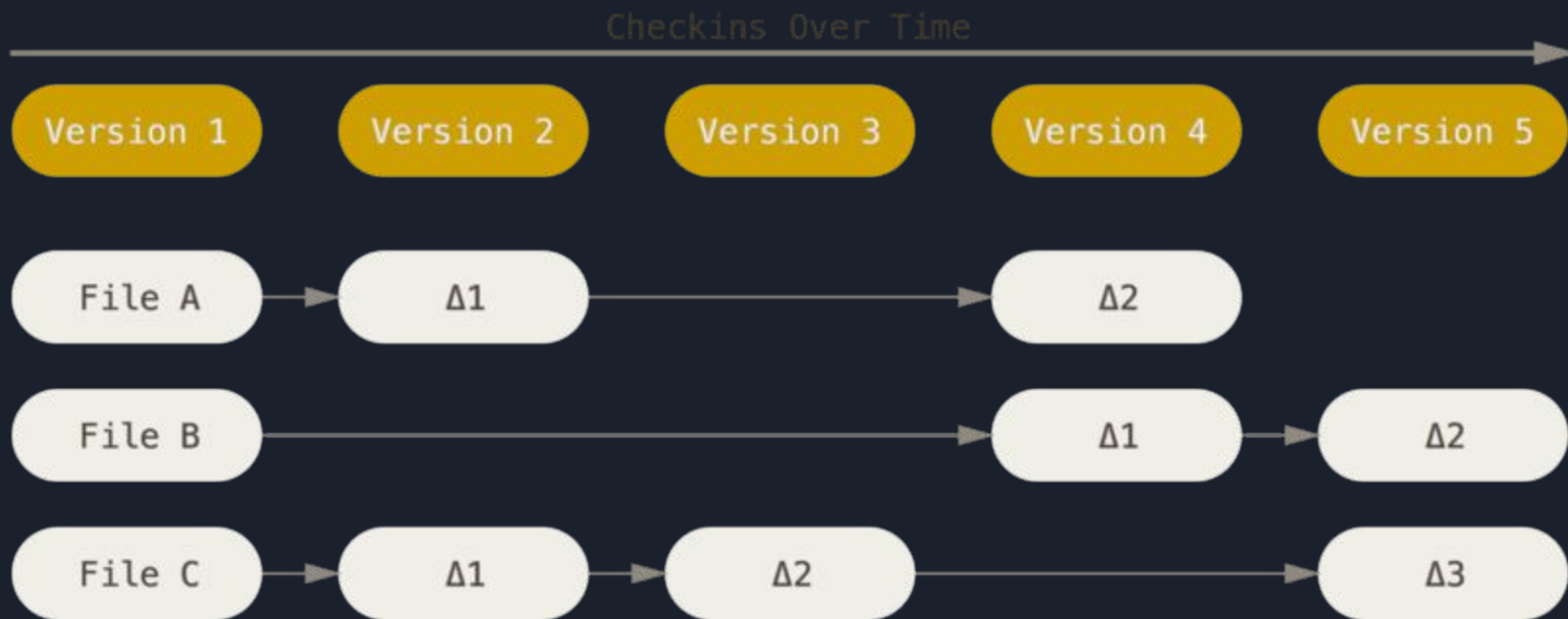
Snapshots, not differences

Most other systems store information as a **list of file-based changes**.

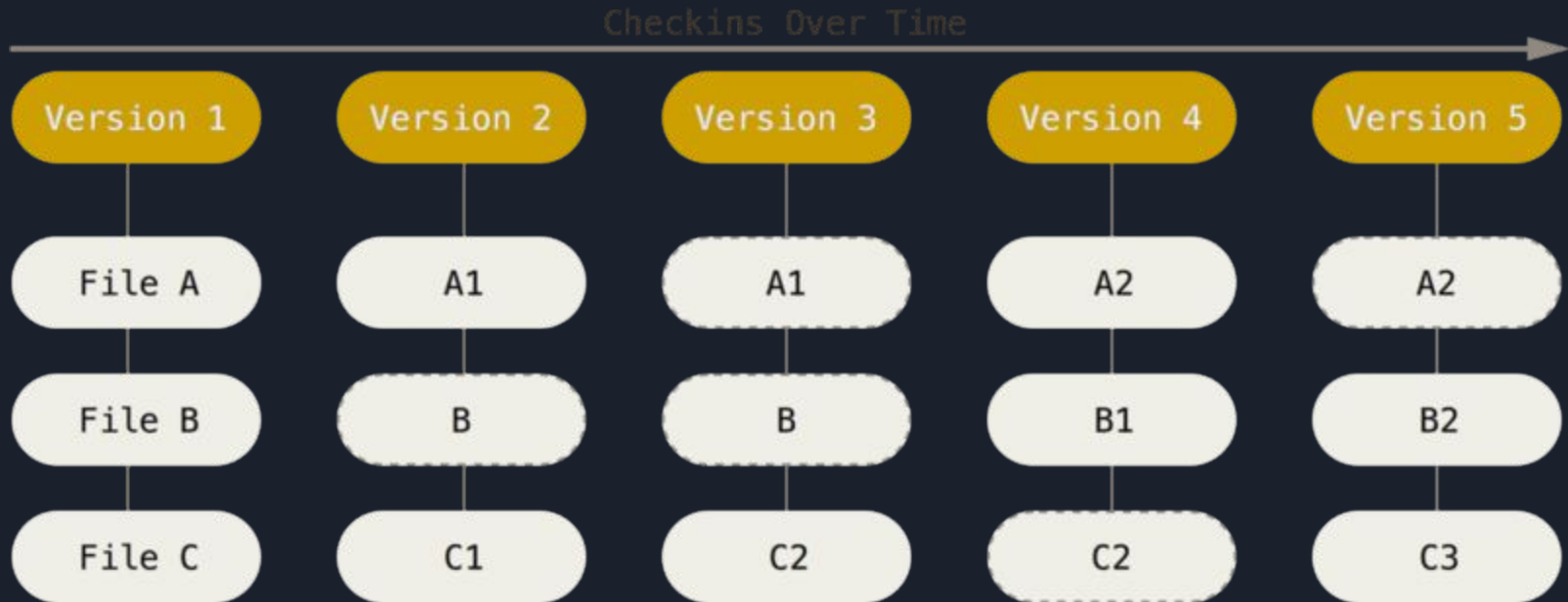
Git thinks of its data as a **series of snapshots of a miniature filesystem**.

If files have not changed, Git doesn't store the file again, just a **link to the previous identical file** it has already stored

Other VCS



Git





Git

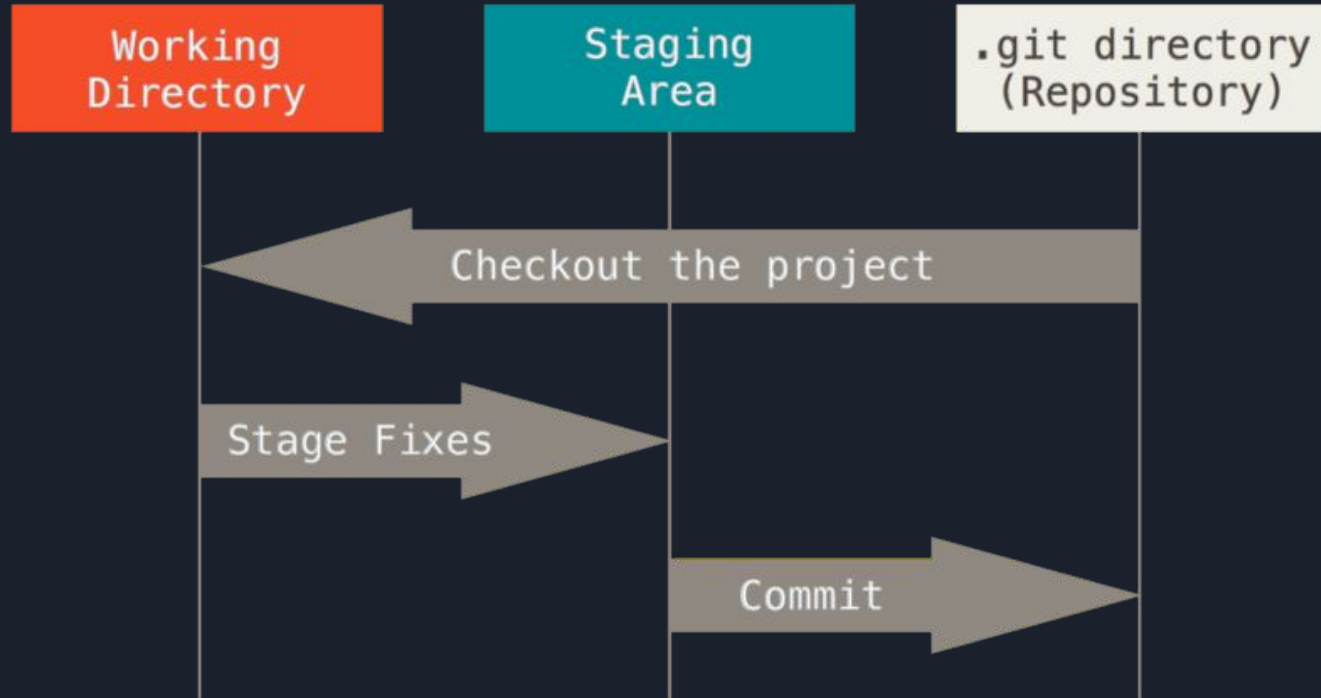
- Nearly Every Operation Is **Local**
- Git Has **Integrity**
 - Everything is checksummed, using SHA-1, before it is stored and is then referred to by that checksum:
`24b9da6552252987aa493b52f8696cd6d3b00373`
- Git Generally **Only Adds Data**
 - Nearly all actions only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way.

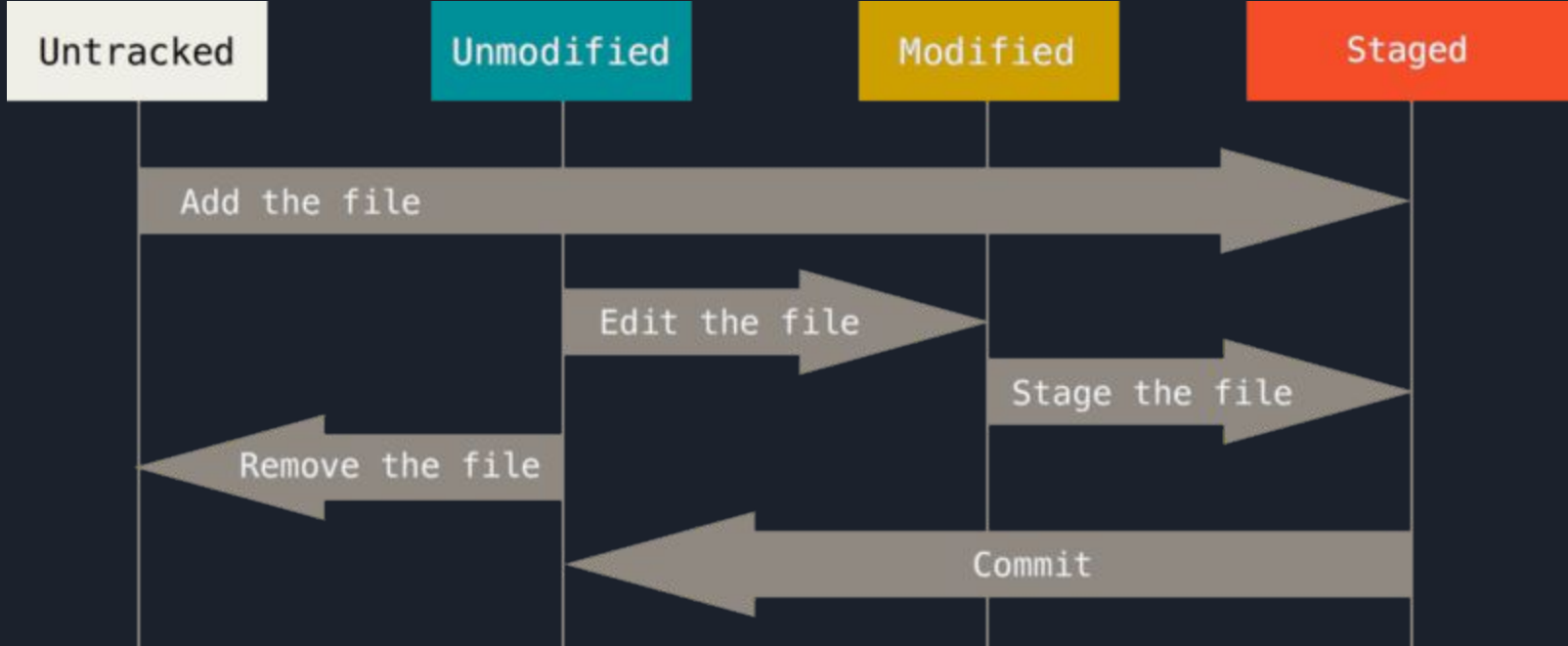


Git Stages

- **Modified:**
 - You have **changed the file but have not committed it** to your database yet.
- **Staged:**
 - You have **marked a modified** file in its current version **to go into your next commit** snapshot.
- **Committed:**
 - Means that the data is safely **stored in your local database**

Git Stages







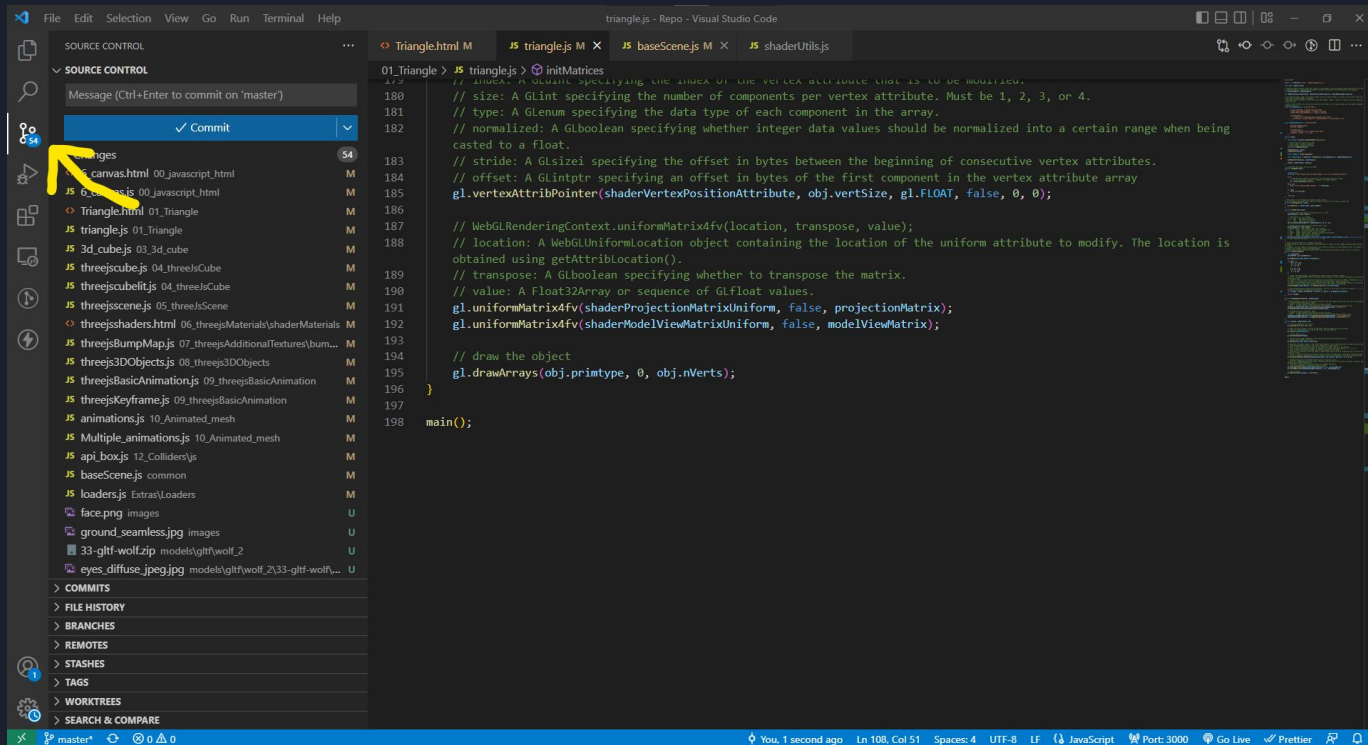
Git Workflow

1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.


Using Git

```
batmung@L03089582L01: ~  
MINGW64 ~/Clases/TC3022/Repo (master)  
$ git status  
Refresh index: 100% (367/367), done.  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified:   00_javascript_html/6_canvas.html  
    modified:   00_javascript_html/6_canvas.js  
    modified:   01_Triangle/Triangle.html  
    modified:   01_Triangle/triangle.js  
    modified:   03_3d_cube/3d_cube.js  
    modified:   04_threeJsCube/threejscube.js  
    modified:   04_threeJsCube/threejscubelit.js  
    modified:   05_threeJsScene/threejsscene.js  
    modified:   06_threeJsMaterials/shaderMaterials/threejsshaders.html  
    modified:   07_threejsAdditionalTextures/bumpMap/threejsBumpMap.js  
    modified:   08_threejs3DObjects/threejs3DObjects.js  
    modified:   09_threejsBasicAnimation/threejsBasicAnimation.js  
    modified:   09_threejsBasicAnimation/threejsKeyframe.js  
    modified:   10_Animated_mesh/Multiple_animations.js  
    modified:   10_Animated_mesh/animations.js  
    modified:   12_Colliders/js/api_box.js  
    modified:   Extras/Loaders/loaders.js  
    modified:   common/baseScene.js  
    modified:   models/json/teapot-claraio.json  
    modified:   models/obj/Penguin_obj/penguin.mtl  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    images/face.png
```

Using Git



Using Git

 **git** --everything-is-local

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire **Pro Git** book written by Scott Chacon and Ben Straub is available to **read online for free**. Dead tree versions are available on **Amazon.com**.

GUI Clients

Git comes with built-in GUI tools for committing (**git-gui**) and browsing (**gitk**), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just [follow the instructions](#).

All

Windows

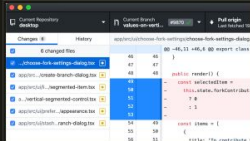
Mac

Linux


Android

iOS

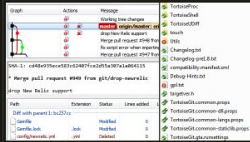
34 Windows GUIs are shown below ↓



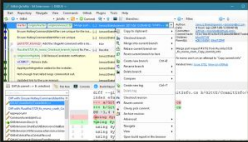
GitHub Desktop
Platforms: Mac, Windows
Price: Free
License: MIT



SourceTree
Platforms: Mac, Windows
Price: Free
License: Proprietary



TortoiseGit
Platforms: Windows
Price: Free
License: GNU GPL



Git Extensions
Platforms: Linux, Mac, Windows
Price: Free
License: GNU GPL



Course materials

https://github.com/gilecheverria/github_fundamentals



Activity: Installing Git

Installing git, and exploring github

Terminal commands



Git Setup

- `git --version`
- `git config`
- `git config --list`
- `git config --list --show-origin`
- `git config --list --system`
- `git config --list --global`
- `git config --list --local`

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



Git Help

If you ever need help while using Git, there are three equivalent ways to get the comprehensive manual:

```
$ git help <verb>
```

```
$ git <verb> --help
```

```
$ man git-<verb>
```

For example, you can get the manpage help for the git config command by running this:

```
$ git help config
```

Git Help

For condensed information on a command, you can use the **-h** parameter after a command.

```
$ git add -h
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run            dry run
    -v, --verbose            be verbose

    -i, --interactive        interactive picking
    -p, --patch              select hunks interactively
    -e, --edit               edit current diff and apply
    -f, --force              allow adding otherwise ignored files
    -u, --update             update tracked files
    --renormalize            renormalize EOL of tracked files (implies -u)
    -N, --intent-to-add      record only the fact that the path will be added later
    -A, --all                add changes from all tracked and untracked files
    --ignore-removal         ignore paths removed in the working tree (same as --no-all)
    --refresh                don't add, only refresh the index
    --ignore-errors          just skip files which cannot be added because of errors
    --ignore-missing         check if - even missing - files are ignored in dry run
    --sparse                 allow updating entries outside of the sparse-checkout cone
    --chmod (+|-)x           override the executable bit of the listed files
    --pathspec-from-file <file> read pathspec from file
    --pathspec-file-nul      with --pathspec-from-file, pathspec elements are separated with \0
```




Activity: Creating a remote repository

Creating repositories and making commits

People use GitHub to build some of the most advanced technologies in the world. Whether you're visualizing data or building a new game, there's a whole community and set of tools on GitHub that can help you do it even better.

A repository is a centralized space where computer files are uploaded, saved, organized, and downloaded. In software engineering they are used to manage projects that involve the use of source code files. On Github you can create several repositories to work with other people on interdisciplinary projects, and share your progress to receive feedback or help others in their respective work.

In this activity you will learn how to create, manage, and make changes to a repository.

What is a repository?: A [repository](#) is a project containing files and folders. A repository tracks versions of files and folders. For more information, see "[About repositories](#)" from GitHub [Docs](#).

Overview

In this activity, you will:

1. [Create a new repository.](#)
2. [Make changes to a file in github](#)
3. [Create a new file, and make changes to it](#)



Activity: Markdown

Communicating with markdown

GitHub is about more than code. It's a platform for software collaboration, and [Markdown](#) is one of the most important ways developers can make their communication clear and organized in issues and pull requests. This course will walk you through creating and using headings more effectively, organizing thoughts in bulleted lists, and showing how much work you've completed with checklists. You can even use Markdown to add some depth to your work with the help of emoji, images, and links.

Overview

In this activity, you will be able to use markdown to:

- [Add headers.](#)
- [Create links.](#)
- [Include images.](#)
- [Quote code examples.](#)
- [Create lists and task lists.](#)



Activity: Working with a repository

Working with local and remote repositories

As we have seen, a repository is a space to save, manage and organize the files of a project. However, if your computer is where you do all the work, how can you download a remote repository and work locally?

This is where the concepts of local and remote repository come in. You see, all your repositories on Github are versions of your project that are hosted on the Internet or any other network, therefore, they are remote spaces.

To access its contents, you need to create a space on your computer from which you can send and receive changes made to it. We will call this place local repositories. In this section, you will learn how to create a local repository on your computer and then connect it to the remote repository you just created on Github.

What is a remote?: This is the version of a repository or branch that is hosted on a server, most likely GitHub.com. Remote versions can be connected to local clones so that changes can be synced.

What is a clone?: A clone is a copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy. When you make a clone, you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online. The repository you cloned is still connected to the remote version so that you can push your local changes to the remote to keep them synced when you're online.

Overview

In this activity, you will:

- [Clone a remote repository in your computer.](#)
- [Making and pushing local commits](#)
- [Amend a commit](#)
- [Unstaging a staged file](#)
- [Unmodifying a Modified File](#)



Undoing changes

- `git diff`
 - Show changes between commits, commit and working tree, etc.
- `git commit --amend`
 - When you commit too early and possibly forget to add some files, or you mess up your commit message.
- `git checkout -- <file>`
 - Revert a file back to the last commit.
- `git restore:`
 - Used for unstaging, unmodifying
- `git reset`
 - Reset current HEAD to the specified state.
- `git reset HEAD~1`



Removing files

- `git rm`
 - Remove files from the working tree and from the index. The next time you commit, the file will be gone and no longer tracked.
- `git rm --cached`
 - Keep the file in your working tree but remove it from your staging area.



Ignoring files

Often, you'll have files that you don't want Git to automatically add or even show you as being untracked. These are generally automatically generated files such as log files or files produced by your build system.

In such cases, you can create a file listing patterns to match them named `.gitignore`.



Ignoring files

```
$ cat .gitignore  
*.[oa]  
*~
```

The first line tells Git to ignore any files ending in “.o” or “.a” — object and archive files that may be the product of building your code.

The second line tells Git to ignore all files whose names end with a tilde (~), which is used by many text editors such as Emacs to mark temporary files.

You may also include a log, tmp, or pid directory; automatically generated documentation; and so on.



Ignoring files

The rules for the patterns you can put in the `.gitignore` file are as follows:

- Blank lines or lines starting with `#` are ignored.
- Standard glob patterns work, and will be applied recursively throughout the entire working tree.
- You can start patterns with a forward slash (`/`) to avoid recursivity.
- You can end patterns with a forward slash (`/`) to specify a directory.
- You can negate a pattern by starting it with an exclamation point (`!`).



Ignoring files - Glob patterns

Glob patterns are like simplified regular expressions that shells use:

- An asterisk (*) matches zero or more characters
- [abc] matches any character inside the brackets
- A question mark (?) matches a single character
- Brackets enclosing characters separated by a hyphen ([0-9]) matches any character between them (in this case 0 through 9).



Ignoring files - Glob patterns

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Git branching



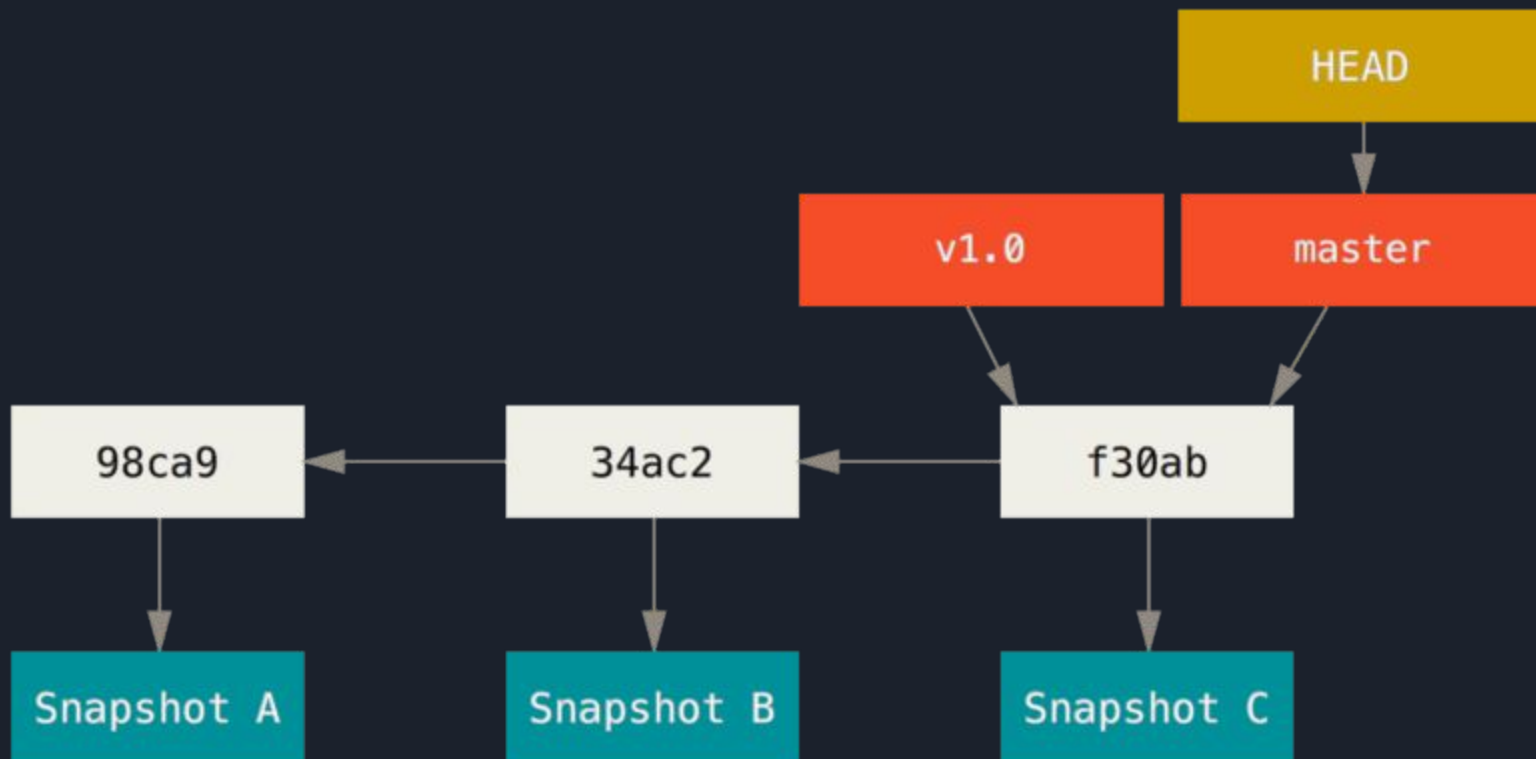


Git branching

A branch in Git is simply a **lightweight movable pointer** to one commit.

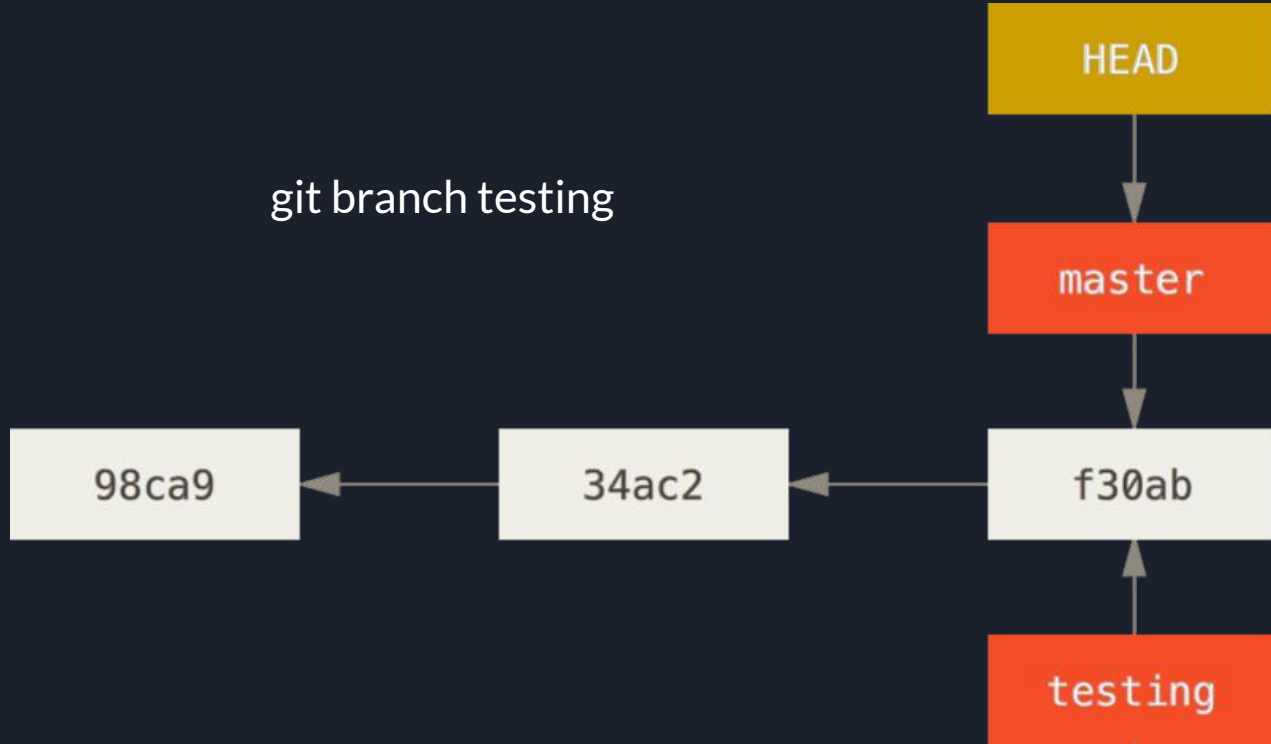
The default branch name in Git is master (main). As you start making commits, you're given a master branch that points to the **last commit** you made.

Every time you commit, **the master branch pointer moves forward** automatically.

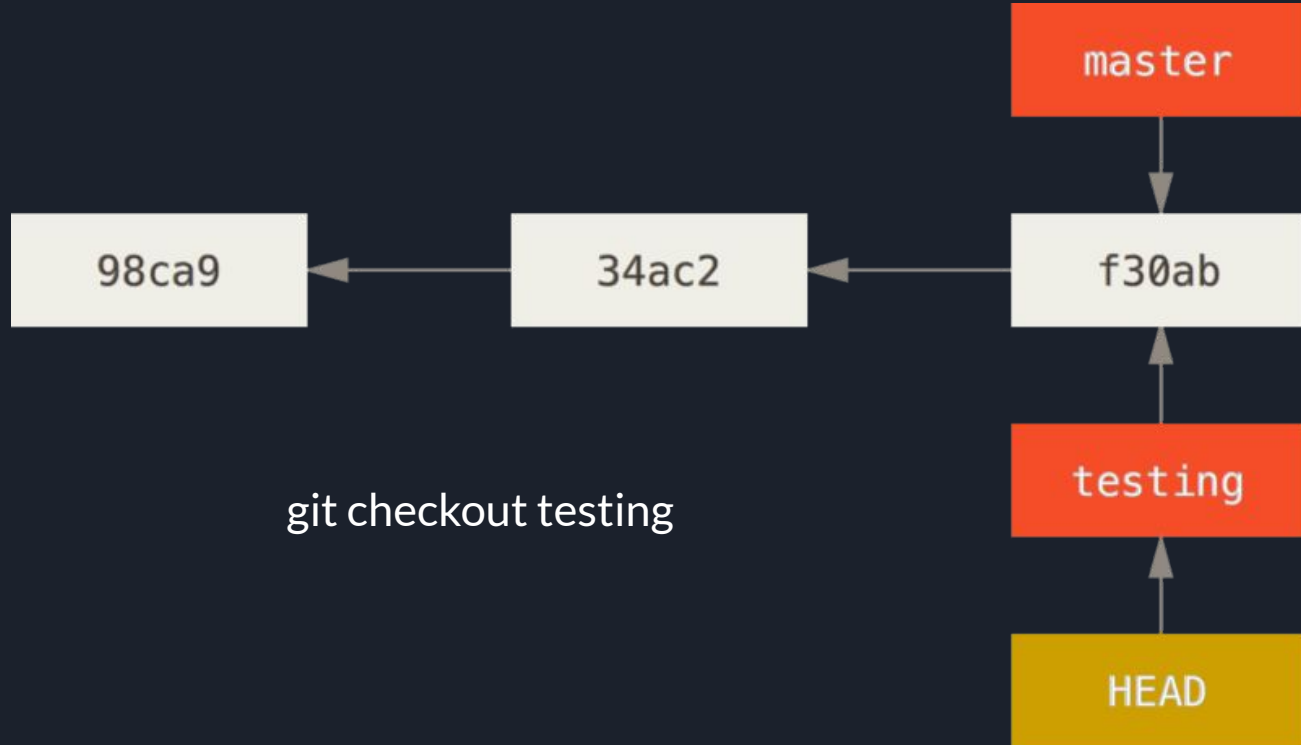


Creating a new branch

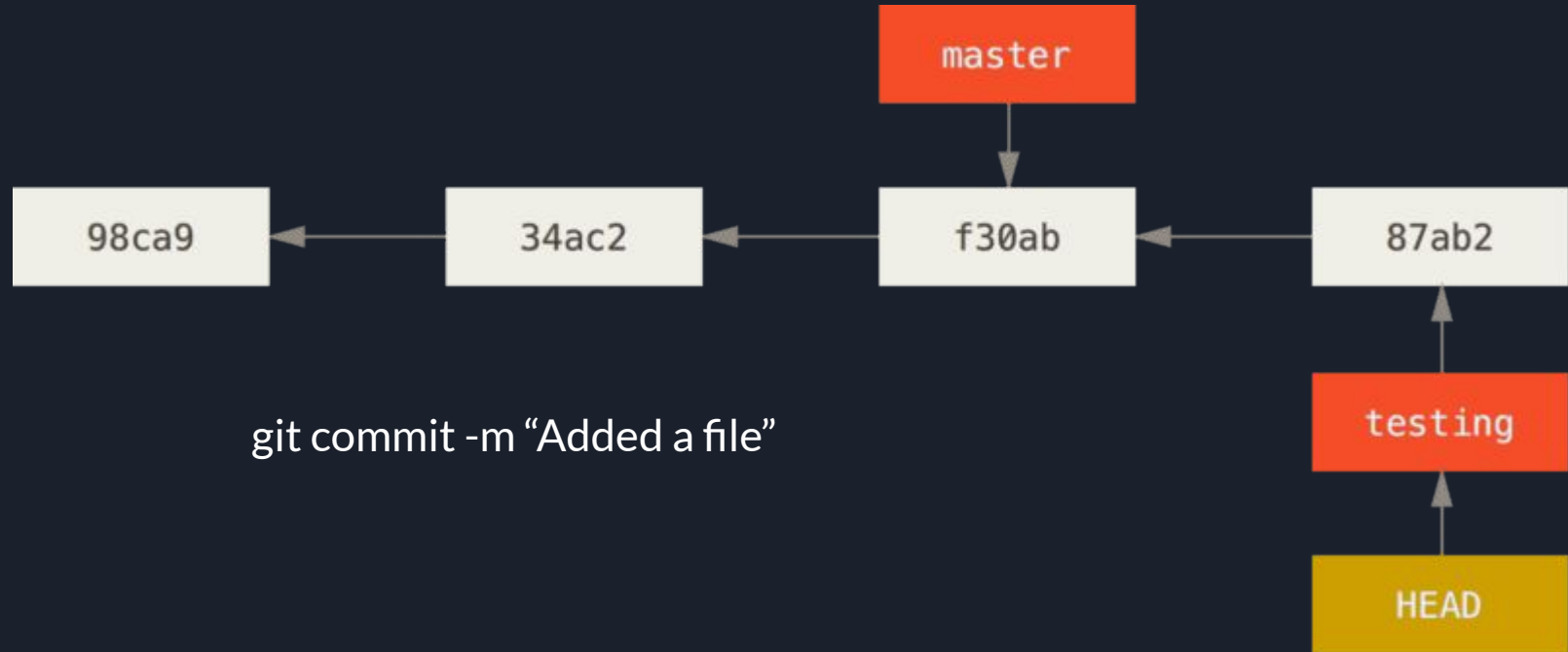
git branch testing



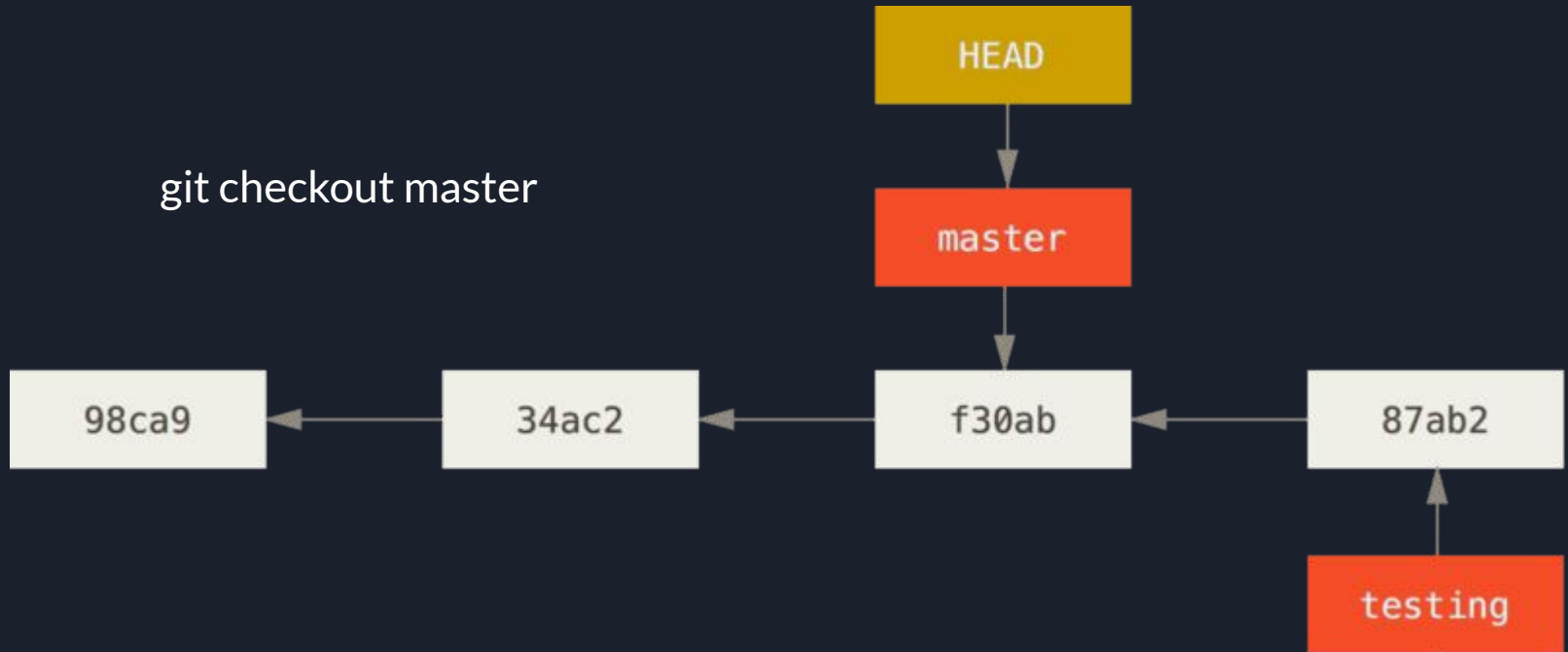
Switching branches



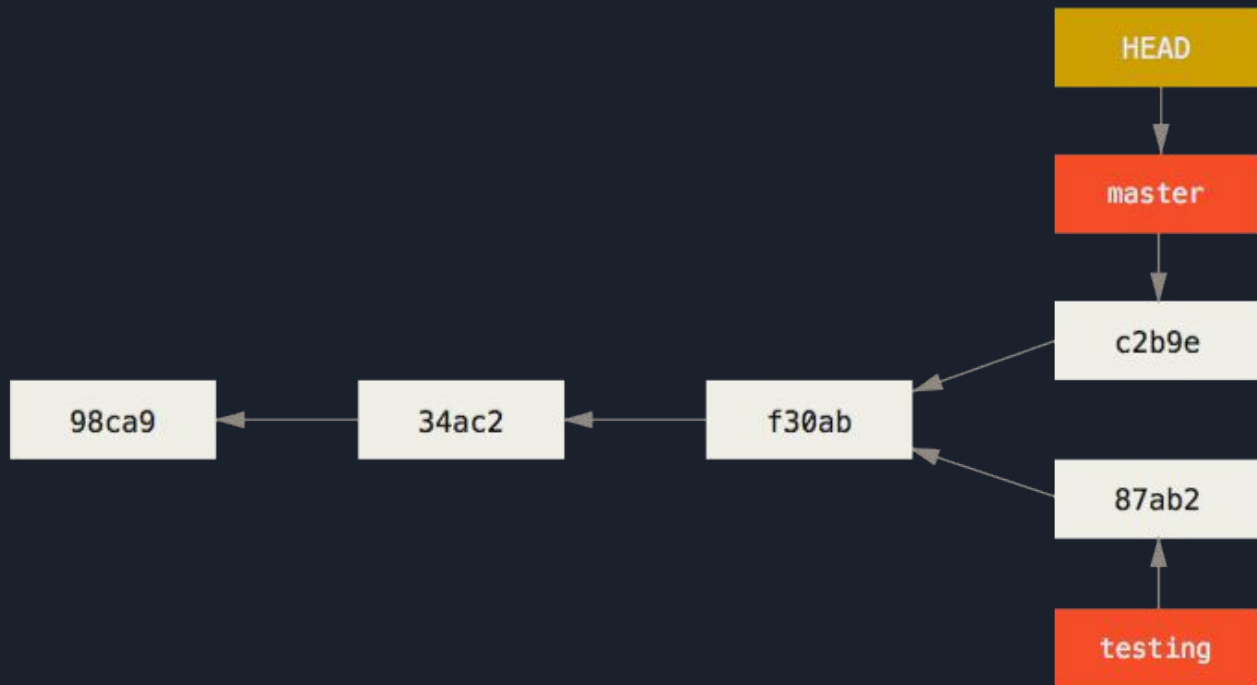
Switching branches

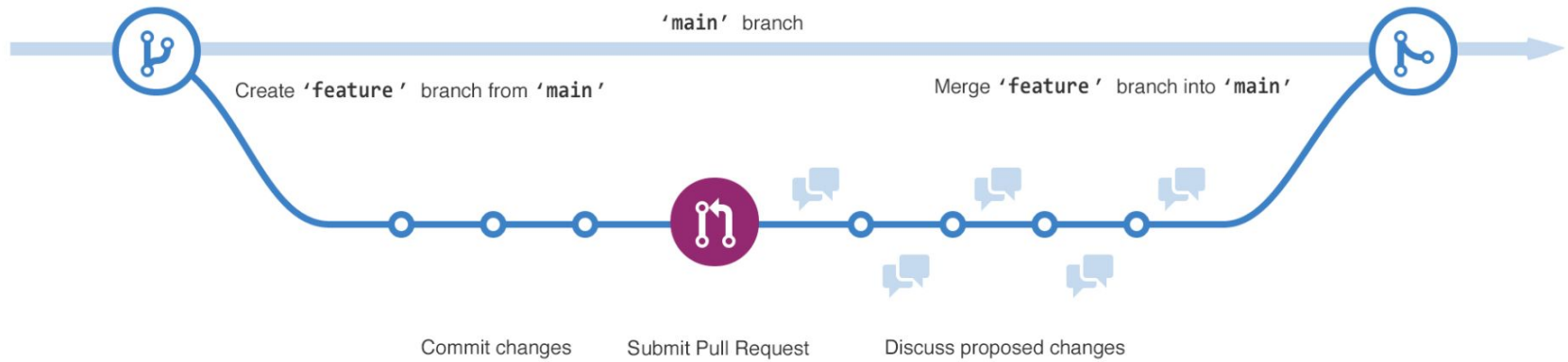


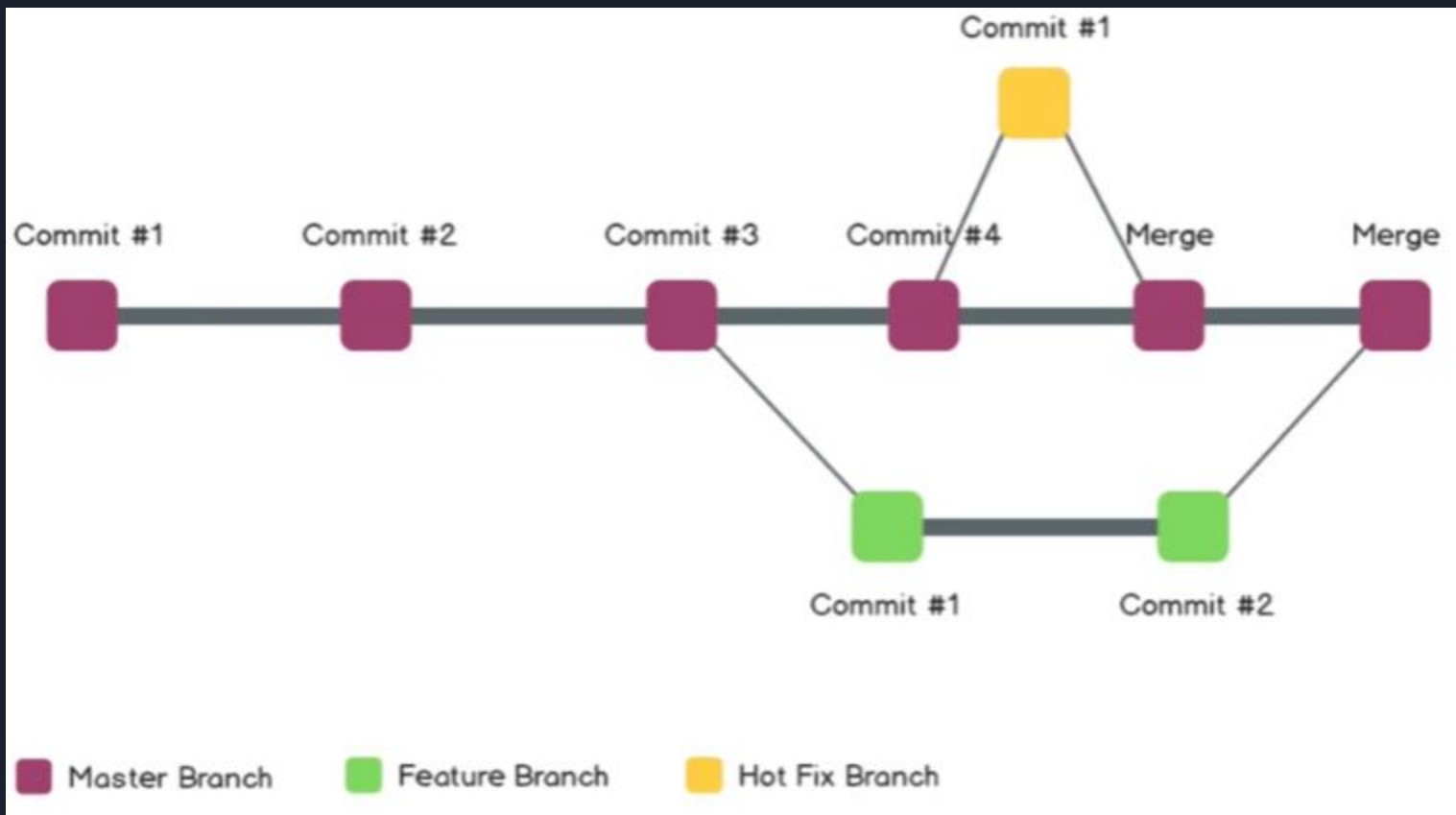
Switching branches



Diverting branches









Activity: Git branching

Git branching

Now that we have the local repository ready on our computer, we can start working. Surely now you are wondering, if several people participate in the same repository, then how can each person carry out their own work without affecting the progress of the others?

Unlike tools like Google Docs or Google Slides, where everyone works on the same file, in a repository, each person creates their own space within it (called a branch) where they create and upload their corresponding part. Once someone or everyone has finished, their progress is combined to integrate the work into a main branch. Next, we will learn everything necessary to create branches and join the content of multiple branches.

What is a branch? A [branch](#) is a parallel version of your repository. By default, your repository has one branch named main and it is considered to be the definitive branch. Creating additional branches allows you to copy the main branch of your repository and safely make any changes without disrupting the main project. Many people use branches to work on specific features without affecting any other parts of the project.

Branches allow you to separate your work from the main branch. In other words, everyone's work is safe while you contribute. For more information, see ["About branches"](#).

Overview

In this activity, you will:

1. [Create branches](#)
2. [Create commits in a new branch](#)
3. [Merge branches](#)
4. [Deleting branches](#)



Activity: Collaboration

Collaborative work and Pull Requests

In the previous exercises we have seen how to create, merge, and delete branches in a repository. However, you have done everything yourself in your own repository. What happens when you have to work with other people? Next, we'll work on a new repository to test what you've learned in these labs with a team.

Overview

In this activity, you will:

1. [Learn how to add contributors to a repository](#)
2. [Practice creating branches, and commits to those branches](#)
3. [Merging remote branches made by others to the main branch](#)
4. [Use github flow to create branches and pull requests](#)

Recommendations

- Don't panic.
- Commit often (but not too often).
- Make useful, descriptive, commit messages.
- Ideally, use past tense.
- Branch before you push to main.
- Commit only related work.
- Choose a workflow ([branching workflows](#) work really well).
- Always fetch and pull.
- Don't commit generated files.
- Don't commit configuration files.
- Avoid committing large binary files.



Documentation and additional resources:

<https://git-scm.com/>

<https://git-scm.com/book/en/v2>

<https://docs.github.com/en/get-started/quickstart/hello-world>

<https://git-scm.com/docs/gittutorial>

https://learngitbranching.js.org/?locale=es_AR