Version Control System (VCS)

TEKsystems Global Services

WORKFORCE DEVELOPMENT

LOGISTICS

Class Hours:

- Instructor will set class start and end times.
- There will be regular breaks in class.

Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous:

- Courseware
- Bathroom
- Fire drills

VCS

Day 1 Learning Outcomes:

- Comprehensive understanding of Version Control Systems
- Ability to use Git CLI to make checkpoints and version code in a local Git repository
- Create and connect a remote GitHub repository to a local one
- Update code in a remote VCS system, download code, and resolve versioning conflicts
- Make use of Git CLI commands for managing repositories and local code files

VCS

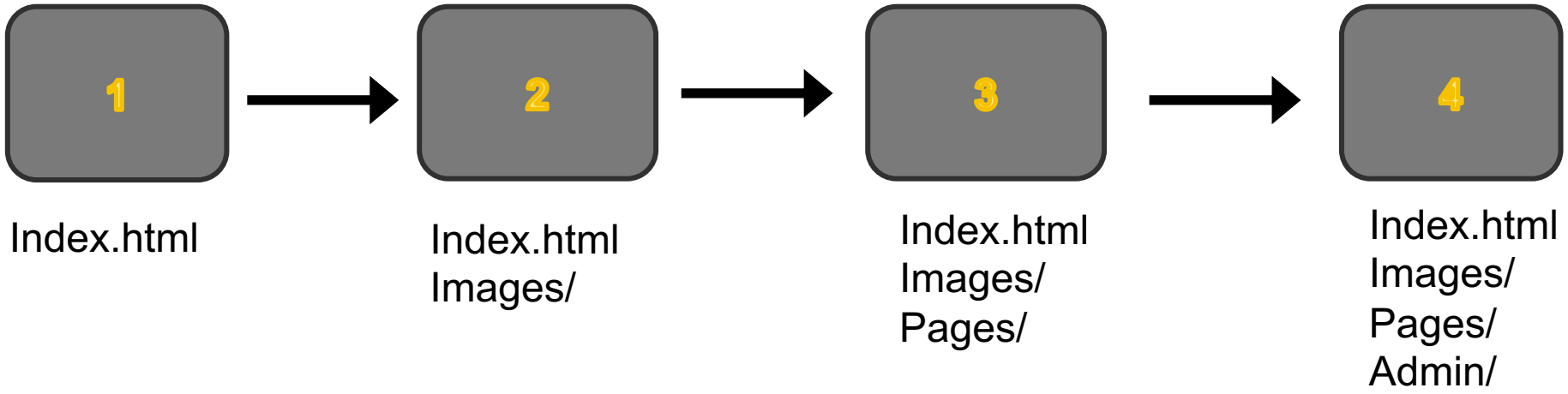
Version control is the practice of tracking and managing changes to files over time.
It allows multiple people to work on the same project simultaneously, review past changes, and restore previous versions if something breaks.

Tools like Git record every change as a commit, creating a detailed history of the project.
This makes collaboration safer and more transparent—teams can experiment, fix bugs, and merge improvements without overwriting each other's work.

VERSION CONTROL

Version control systems let developers create **checkpoints**, called **commits** in Git, that record the state of a project at a specific moment. Each commit acts like a **snapshot** of your files, capturing what changed and who made the change.

These commits allow you to **go back to previous versions**, **compare differences**, and **trace the history** of a project. This makes it easier to fix mistakes, understand how code evolved, and collaborate safely without losing progress.



GIT VCS

Git is a distributed version control system used to track and manage changes in source code.

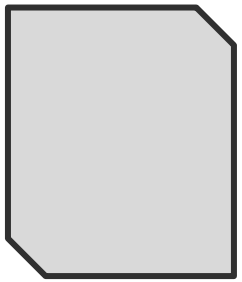
It was created in 2005 by Linus Torvalds, the same developer who built the Linux kernel, to help manage its massive and fast-changing codebase.

VERSION CONTROL

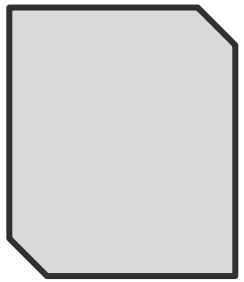
Git tracks any changes made to files and marks them as **Modified (M)**, **Deleted (D)**, or **Added (A)**.

The **staging area** is a preparation zone where you choose which changes to include in your next commit.

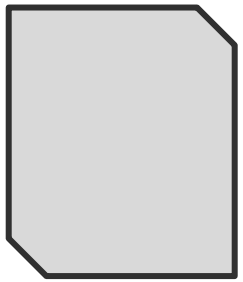
In the example below, all three files have been modified. Suppose we only want to save the updates to *index.html* and *start.sh*. We can **add these two files to the staging area**, preparing them for our next commit while leaving the others untracked for now.



index.html (M)



README.md (M)



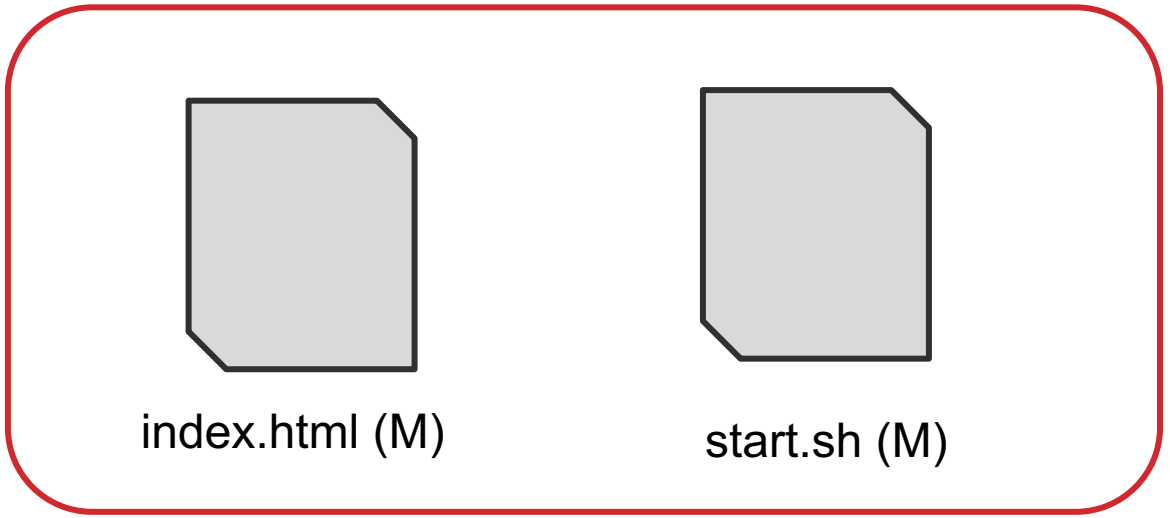
start.sh (M)

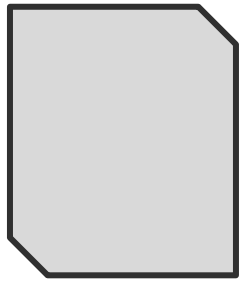
VERSION CONTROL

Lets add two of these files to the staging area

VERSION CONTROL

Staging Area





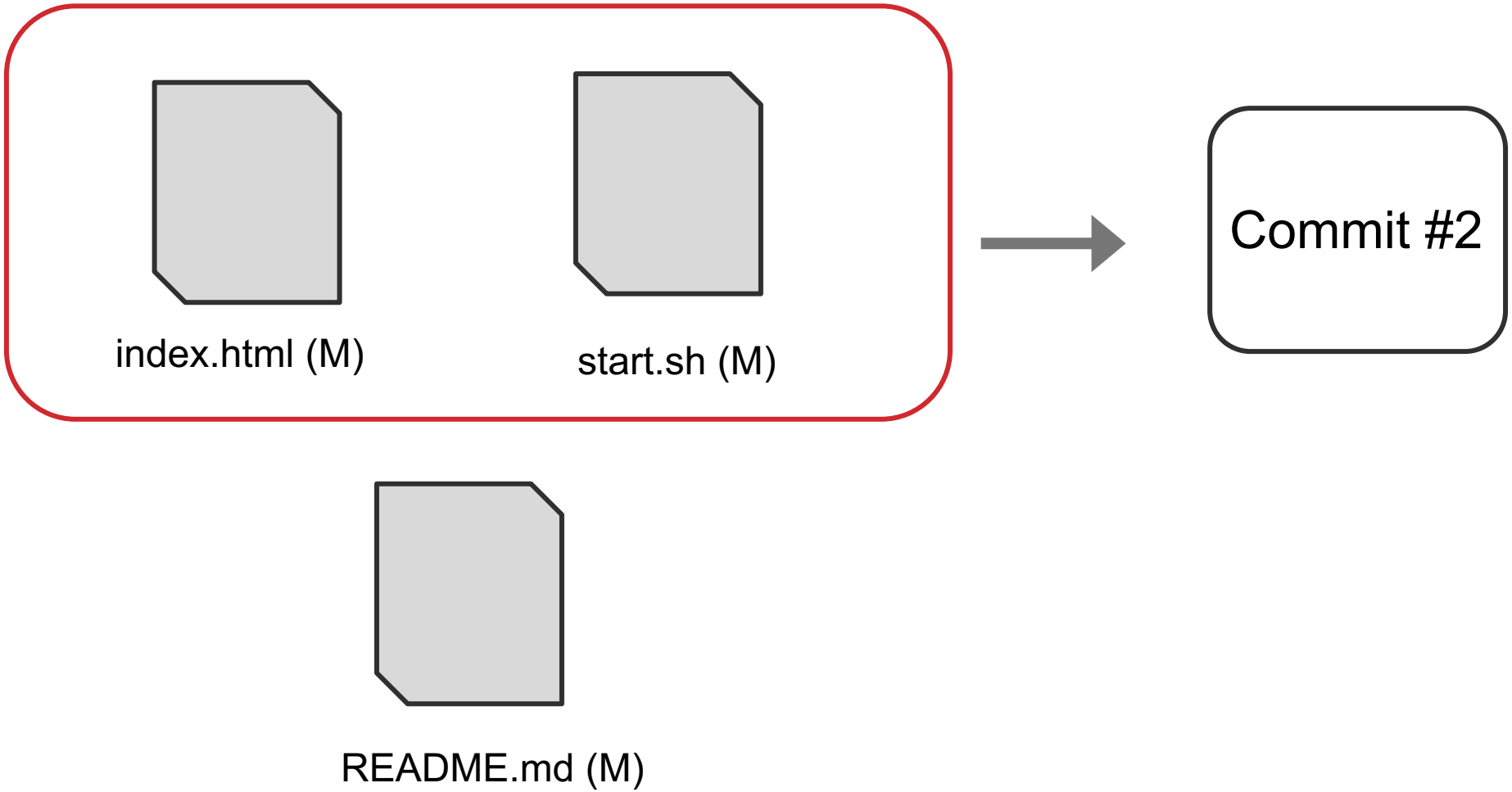
README.md (M)

VERSION CONTROL

Next we can make a commit from the staging area

VERSION CONTROL

Staging Area

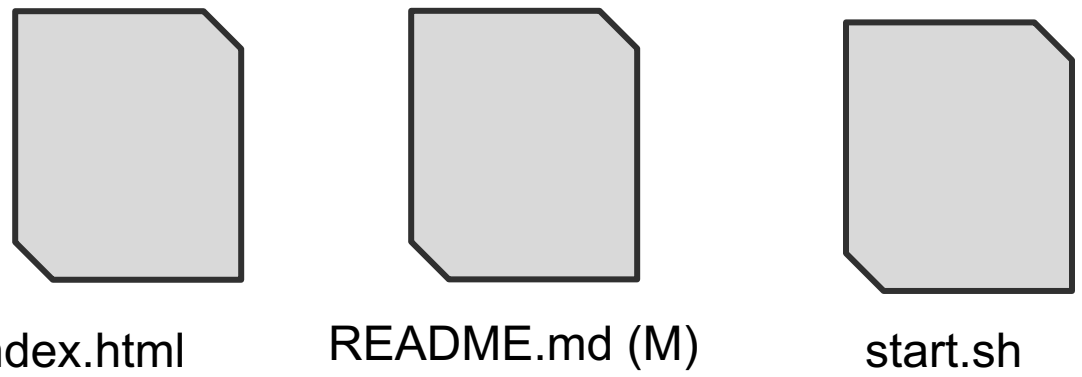


VERSION CONTROL

The commit is now a checkpoint we can always go back to.
The staging area is cleared.

VERSION CONTROL

Staging Area



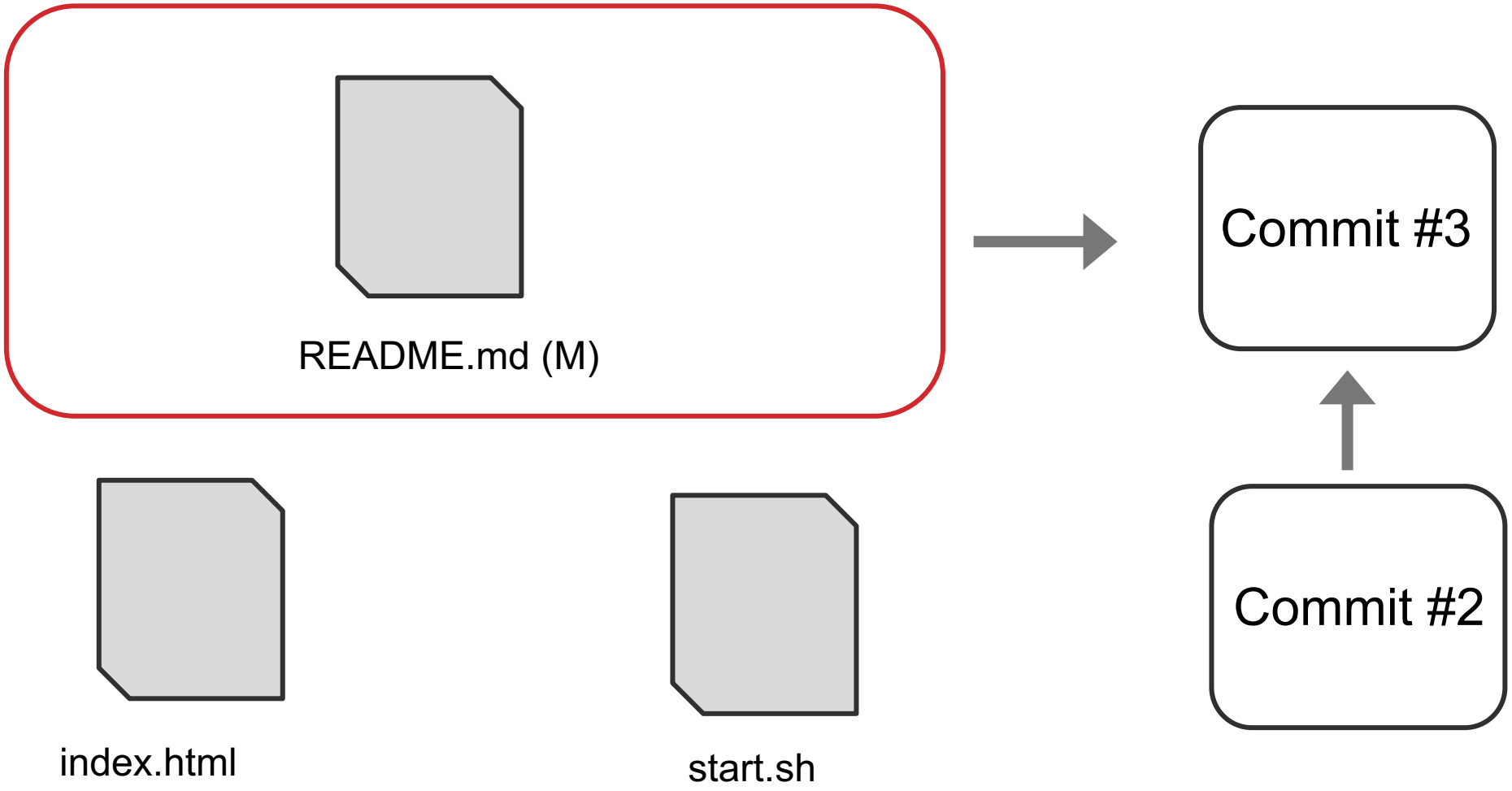
Commit #2

VERSION CONTROL

A third commit will create another checkpoint, this one being the most recent.

VERSION CONTROL

Staging Area

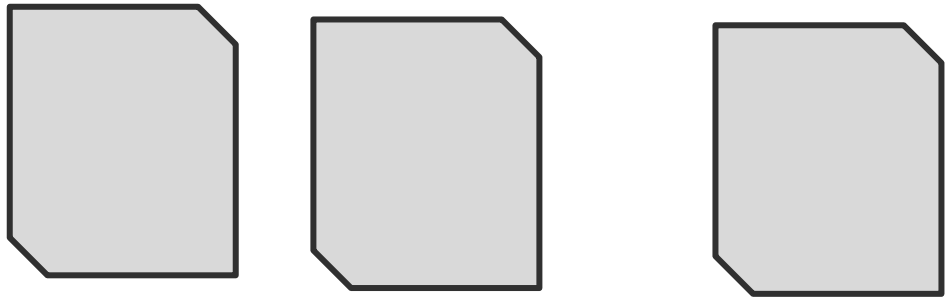


VERSION CONTROL

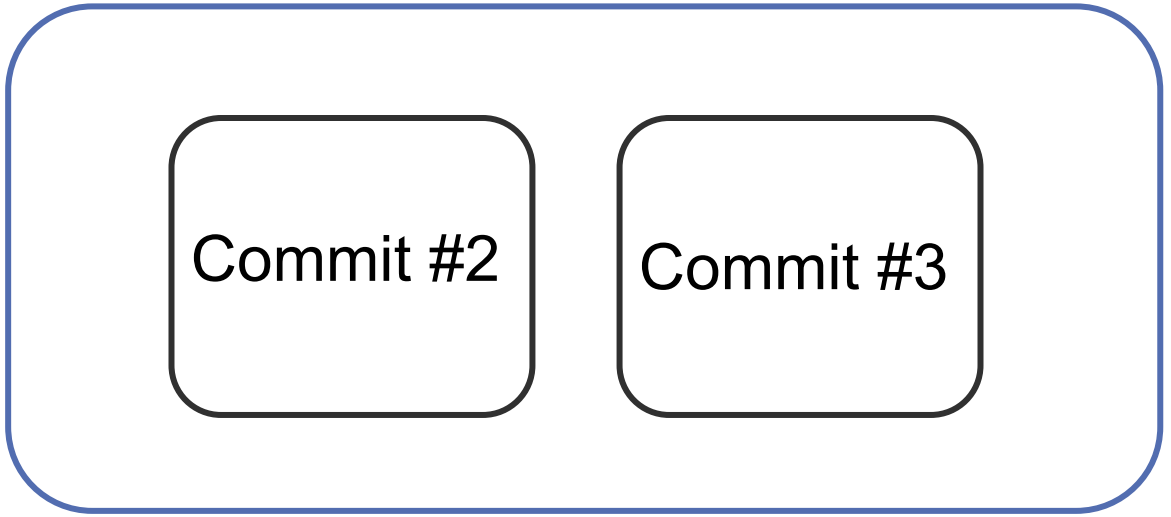
Both commits exists on our local repository only.

VERSION CONTROL

Staging Area



Local Repository

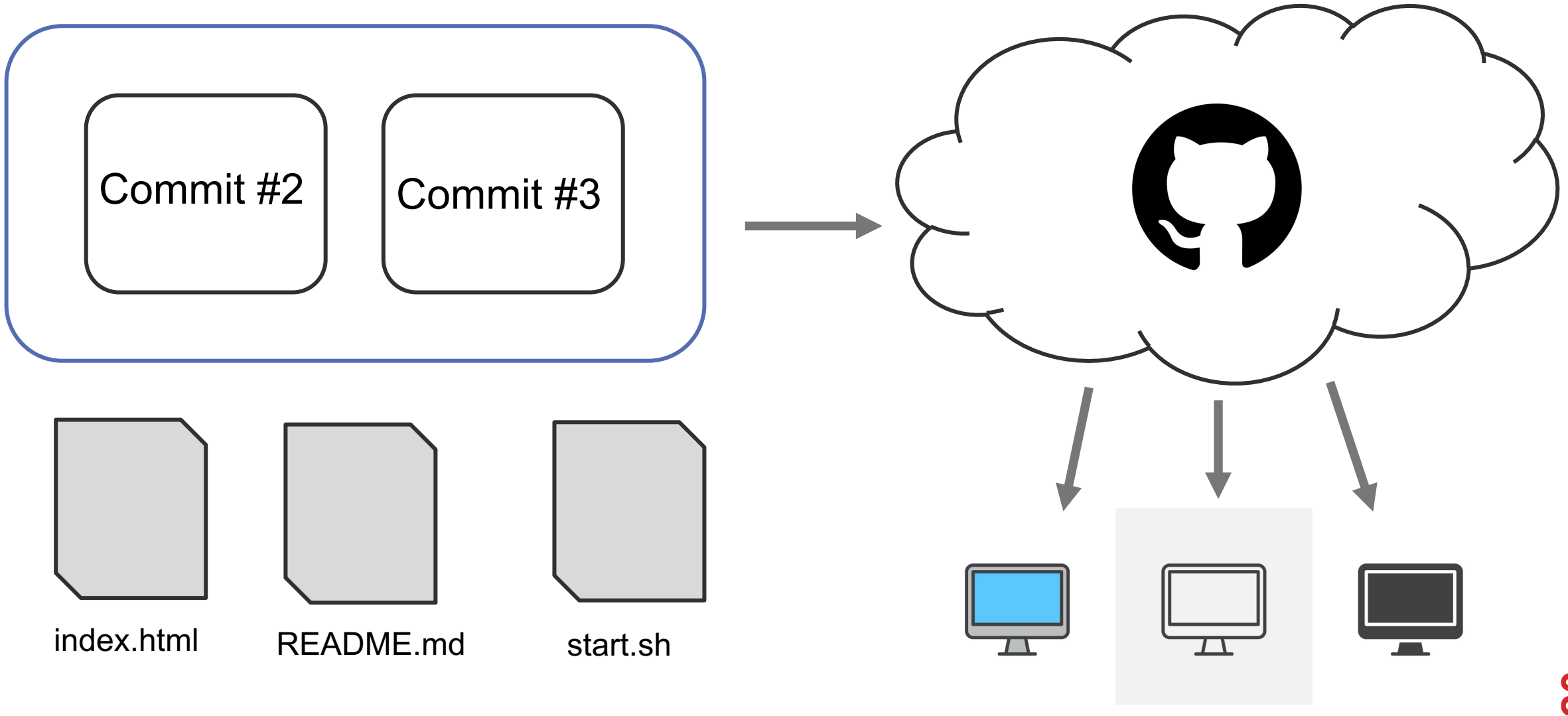


VERSION CONTROL

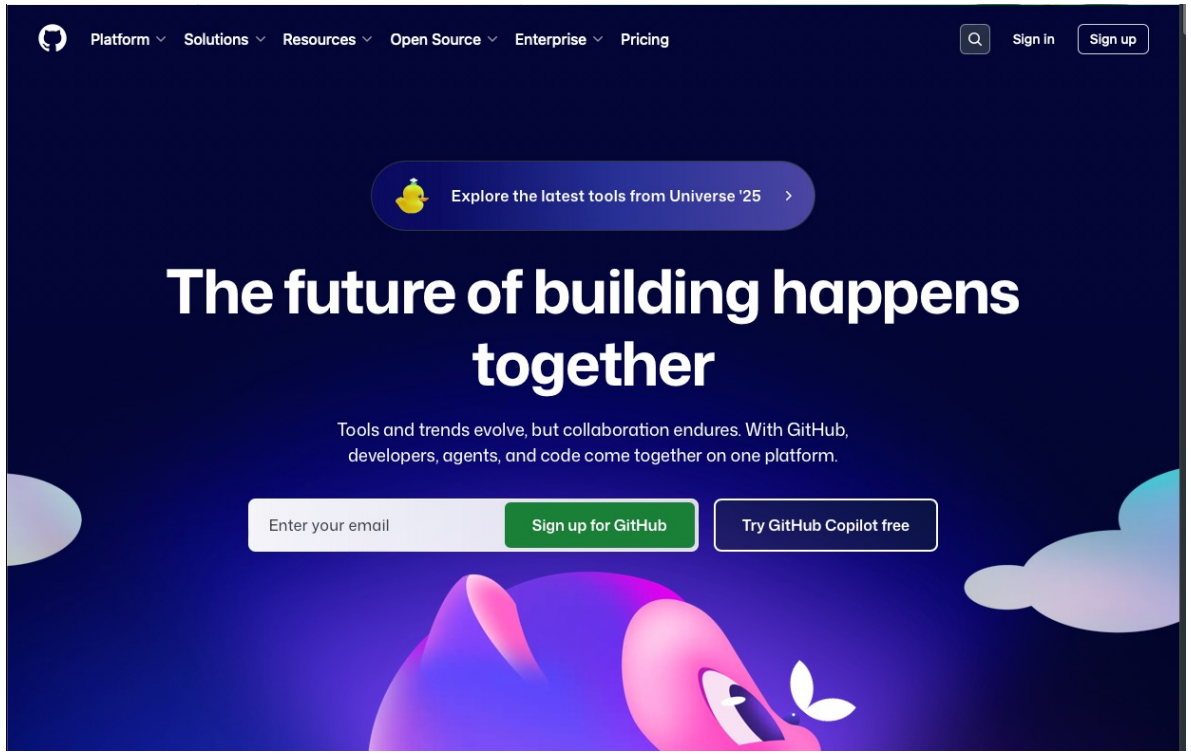
When we push our local commits to the remote repository, it is online and other collaborators can access our updates.

VERSION CONTROL

Local Repository



GITHUB



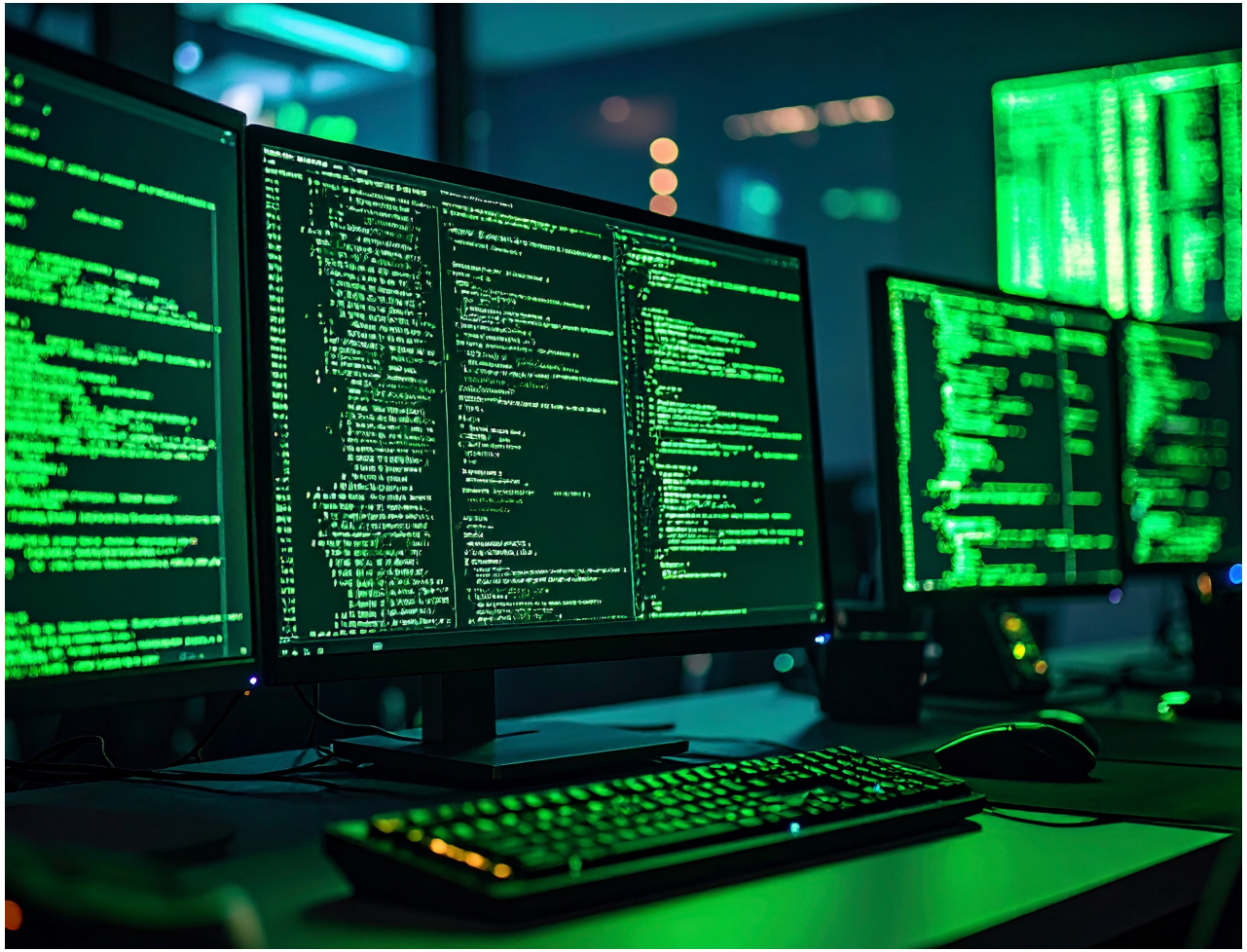
GitHub is a cloud-based platform for version control and collaboration built around Git, a distributed version control system.

It allows developers to host, manage, and track changes to code in remote repositories, collaborate through branches and pull requests, and maintain code quality through features like branch protection, reviews, and integrations.

Visit GitHub Website:

https://github.com

GIT CLI



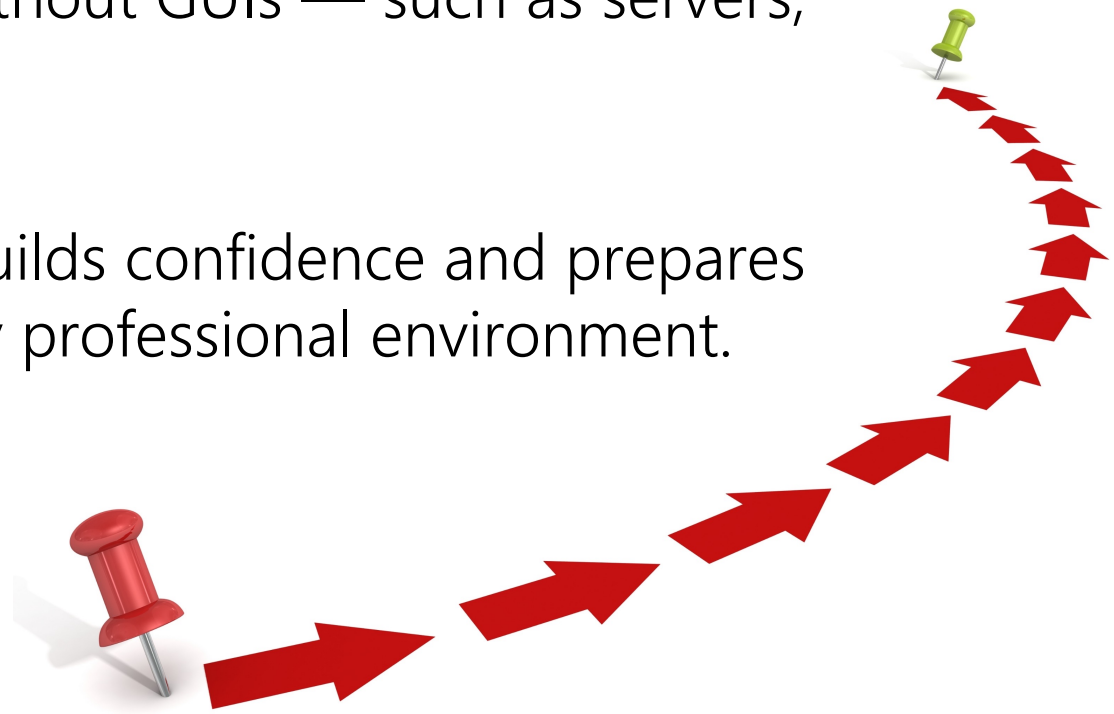
- The Git CLI is the primary way developers interact with Git. It provides full control over version tracking — from creating repositories to managing branches and merging changes.
- Using the CLI helps you understand how Git actually works under the hood, offering more flexibility and precision than graphical tools.

GIT CLI

Graphical interfaces simplify Git, but the CLI reveals its core concepts and workflows.

It helps developers troubleshoot issues, automate tasks, and operate Git in environments without GUIs — such as servers, containers, or CI/CD pipelines.

Learning Git through the CLI builds confidence and prepares you to use Git effectively in any professional environment.



GIT CLI

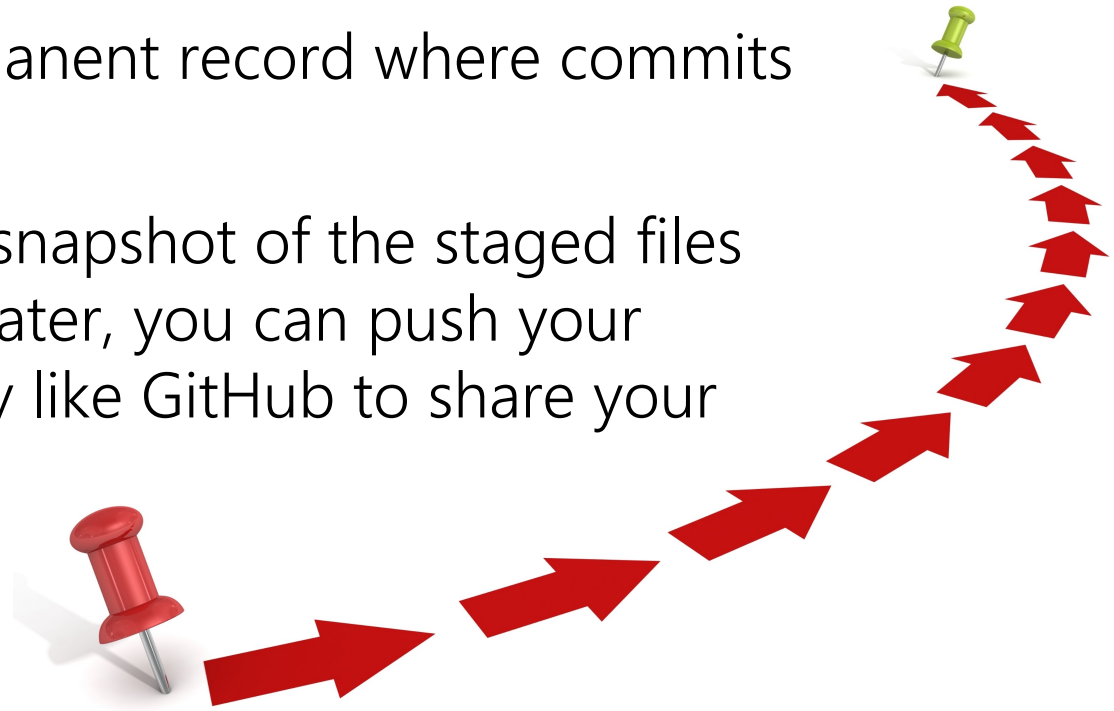
Git organizes your work into three main areas:

Working Directory – where you edit and modify files.

Staging Area (Index) – a holding zone where you prepare specific changes for your next commit.

Repository (History) – the permanent record where commits are stored.

When you commit, Git takes a snapshot of the staged files and saves it in the repository. Later, you can push your commits to a remote repository like GitHub to share your work with others.



LAB 1.1 – GIT CLI

LAB INTRODUCTION

In this lab, you'll learn how to use Git Bash to navigate files, create and edit content, and manage a project using Git.

You'll initialize a repository, stage and commit changes, and use .gitignore to exclude files—all while gaining familiarity with a Unix-like terminal on Windows.

Try referring to the docs if you need assistance:

https://docs.github.com/en/get-started/git-basics

POP QUIZ: GIT CLI

How can all modified files in a git repository be added to staging area?

A. git add .
B. git stage --all
C. git add --all
D. git stage add --all

5 minutes

POP QUIZ: GIT CLI

How can all modified files in a git repository be added to staging area?

A. git add .
B. git stage --all
C. git add --all
D. git stage add --all

5 minutes

POP QUIZ: GIT CLI

What folder contains the meta data for the local repository?

A. .git
B. git
C. .info
D. git_practice

5 minutes

POP QUIZ: GIT CLI

What folder contains the meta data for the local repository?

A. .git
B. git
C. .info
D. git_practice

5 minutes

POP QUIZ: GIT CLI

What command shows you the files you have modified or staged?

A. git commit
B. git log
C. git info
D. git status

5 minutes

POP QUIZ: GIT CLI

What command shows you the files you have modified or staged?

A. git commit
B. git log
C. git info
D. git status

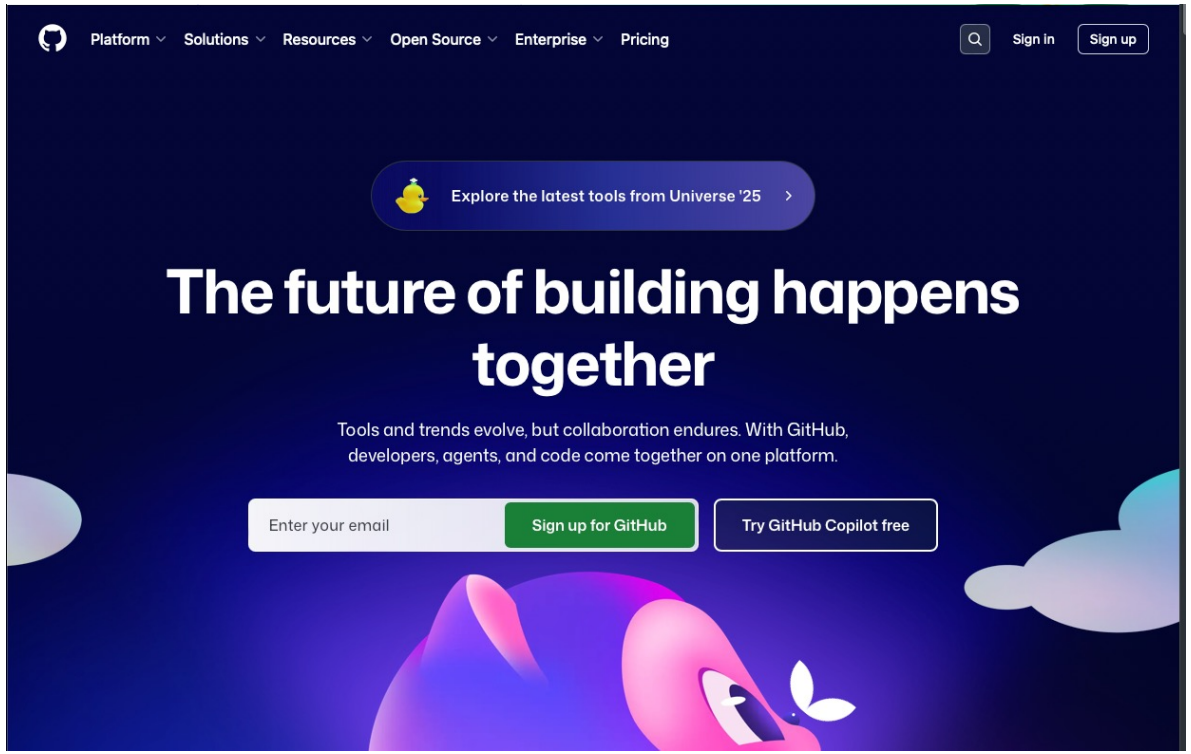
5 minutes

VCS

Intro to GitHub

2

GITHUB



GitHub is a cloud-based platform for version control and collaboration built around Git, a distributed version control system.

It allows developers to host, manage, and track changes to code in remote repositories, collaborate through branches and pull requests, and maintain code quality through features like branch protection, reviews, and integrations.

Visit GitHub Website:

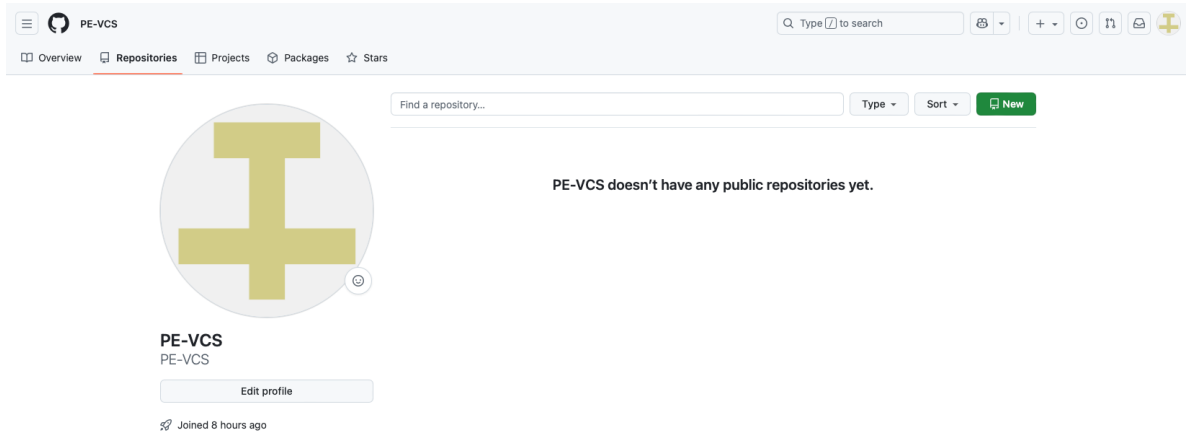
https://github.com

GITHUB

While Git manages version control locally, GitHub extends those capabilities to the cloud, enabling collaboration and visibility. It acts as a central hub where multiple developers can share code, review changes, and track progress.

Using GitHub also provides powerful features like pull requests, issue tracking, Actions for automation, and integrated security tools — making it easier to manage projects, collaborate with teams, and showcase your work publicly or privately.

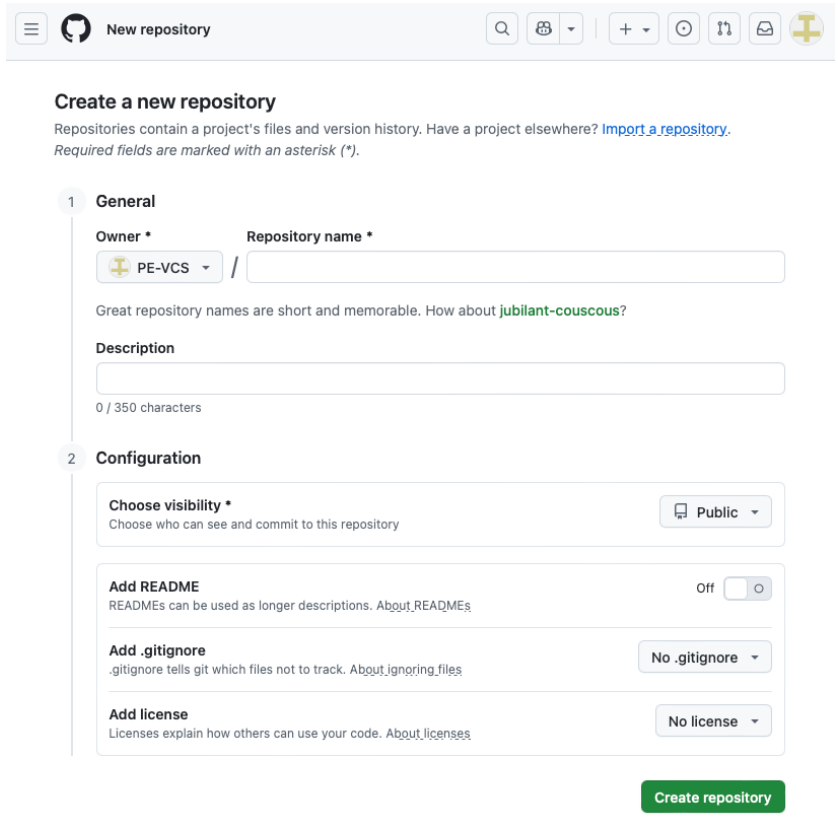
GITHUB



After logging in to GitHub.com, your profile acts as your public developer page. It displays your name, bio, activity, and repositories, helping others see your work and contributions.

Your repositories are where your projects live. Each one holds your files, history, and settings. From your profile, you can create new repositories, explore others, and start collaborating with teams or the open-source community.

GITHUB

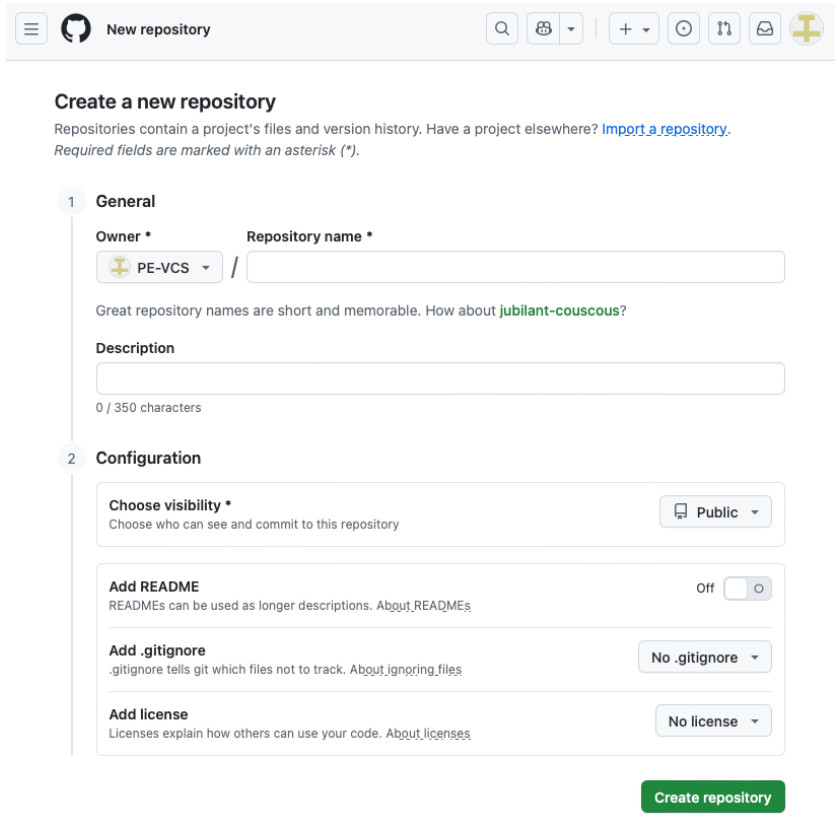


When creating a new repository on GitHub, you can choose to add a README.md file. This file serves as the main description of your project and helps initialize the repository with its first commit, making setup much easier.

If you skip the README, GitHub will create an empty repository. In that case, you'll need to create a local repositoryon your computer, add files, and link it to GitHub before you can push any content.

You can also place a .gitignore file from a template.

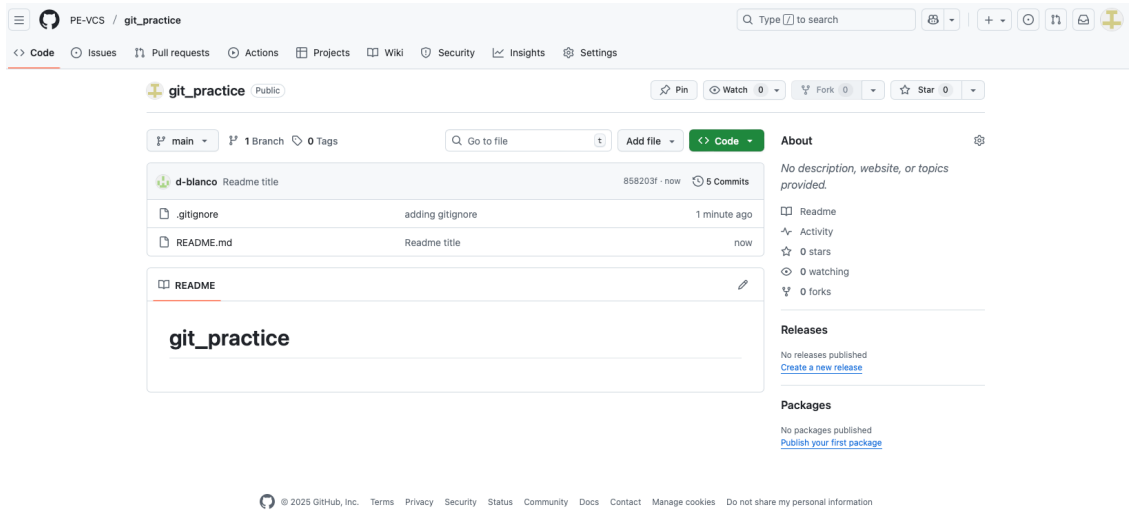
GITHUB



When creating a repository, you can also choose a license, which defines how others can use, modify, and share your code. Adding a license is important if you plan to make your project public — it protects your work and clarifies permissions for contributors.

Common licenses include the MIT License, which allows others to freely use and distribute your code with attribution, and the GNU GPL, which requires anyone who redistributes modified code to share it under the same terms. Choosing the right license depends on how open you want your project to be

GITHUB



A README.md file is the first thing people see when they visit your repository. It explains what your project does, how to use it, and any setup instructions. Writing a clear, simple README helps others understand and contribute to your work.

A good README typically includes a project title, a brief description, and basic usage or installation steps. You can edit it directly on GitHub or in your code editor. Since it uses Markdown, you can format text with headers, lists, and links to make it easy to read.

GITHUB

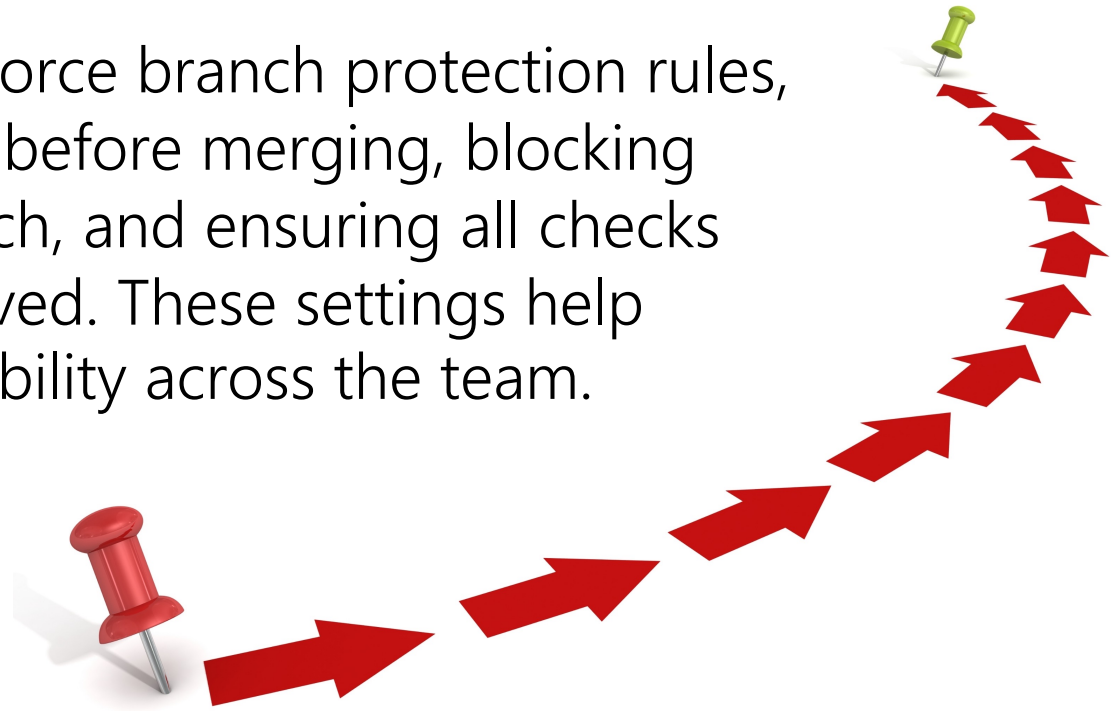
When working in a team, it's important to pull often to keep your local repository up to date with the latest changes from the remote repo. This ensures you're working with the most current version of the project.

Failing to pull regularly can lead to merge conflicts. If someone else has updated a file you also edited, Git will try to auto-merge the changes — but if both of you modified the same lines, it will stop and require you to resolve the conflict manually before pushing your updates.

GITHUB

GitHub provides powerful collaboration settings to help teams manage code quality and communication. You can create issues or discussions to report bugs, request features, or plan upcoming work — keeping all project conversations in one place.

Repository admins can also enforce branch protection rules, such as requiring code reviews before merging, blocking direct pushes to the main branch, and ensuring all checks pass before updates are approved. These settings help maintain stability and accountability across the team.



LAB 1.2 – GITHUB

LAB INTRODUCTION

In this lab, you'll take the local Git repository you created earlier and connect it to GitHub, the cloud-based platform for sharing and collaborating on code. You'll create a GitHub account, set up a remote repository, and push your existing files online.

You'll also explore GitHub's interface — viewing commits, editing files directly on the site, and pulling updates back to your local machine. By the end, you'll understand how local and remote repositories work together to support modern, collaborative development.

POP QUIZ: GITHUB

What is the command to download a GitHub repository?

A. git pull
B. git clone
C. git get repo
D. git restore

POP QUIZ: GITHUB

What is the command to download a GitHub repository?

A. git pull
B. git clone
C. git get repo
D. git restore

POP QUIZ: GITHUB

How can I make a level 2 (h2) title in Markdown?

A. <h2>Title</h2>

B. ## Title

C. **Title**

D. L2 Title

POP QUIZ: GITHUB

How can I make a level 2 (h2) title in Markdown?

A. <h2>Title</h2>

B. ## Title

C. **Title**

D. L2 Title

VCS

Push, Pull, Conflicts

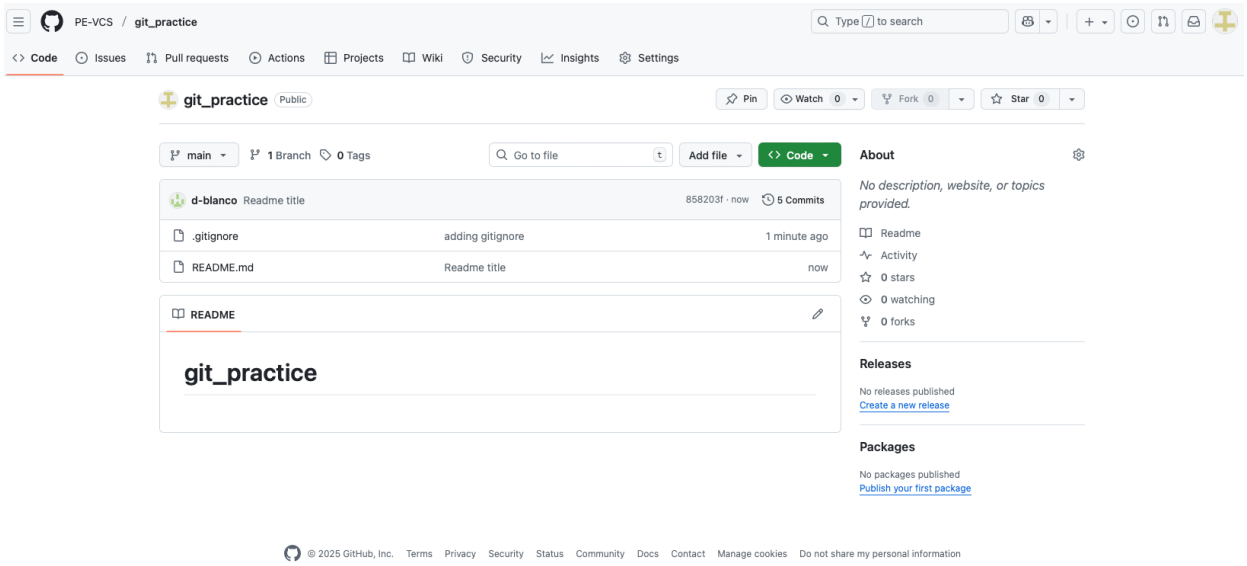
2

PUSH AND PULL

The push and pull cycle connects your local Git repository to GitHub. When you make changes locally and commit them, you use git push to upload those changes to the remote repository, keeping your online version up to date and shareable with others.

When teammates make updates on GitHub, you use git pull to download their latest commits to your local environment. Regularly pulling before pushing prevents conflicts and ensures your work stays in sync with the team's progress.

PUSH AND PULL



After pushing your code, visit your repository on GitHub.com to confirm everything uploaded correctly. You'll see your files and commit history.

You an edit files directly in the browser, notice it will create a commit.

CONFLICT

A **merge conflict** occurs when Git can't automatically merge changes — typically because two people edited the **same lines** in a file on different branches or commits. Git stops the merge so you can decide which version to keep.

The first time you encounter a conflict, you may need to tell Git what **merge strategy** to use. Run the command before # merge

```
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

CONFLICT

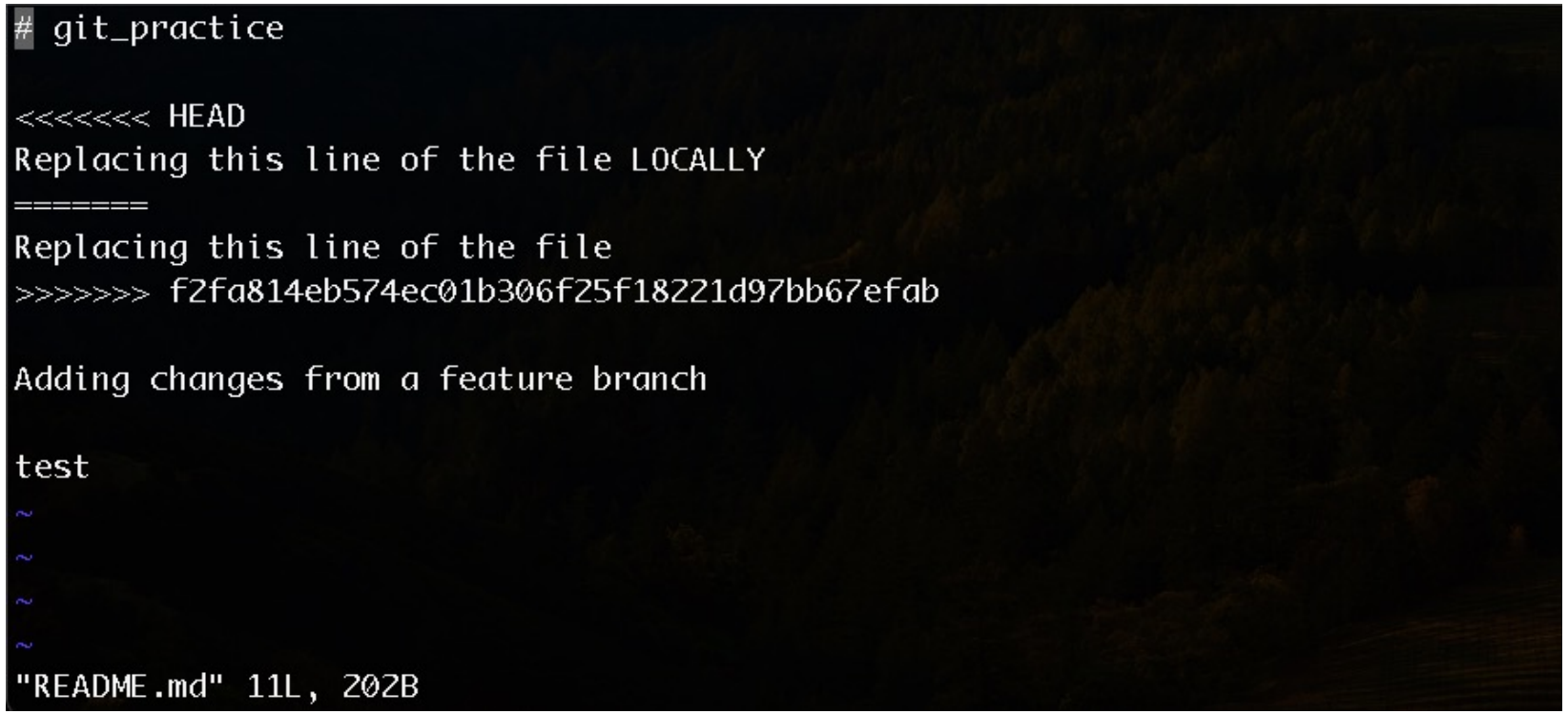
When you pull or merge, Git detects the overlap and pauses to ask you which version should stay.

You must open the file, review the changes, decide what to keep, and then stage and commit the resolved version. This ensures that no work is lost and that the final version accurately reflects the team's intentions.

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

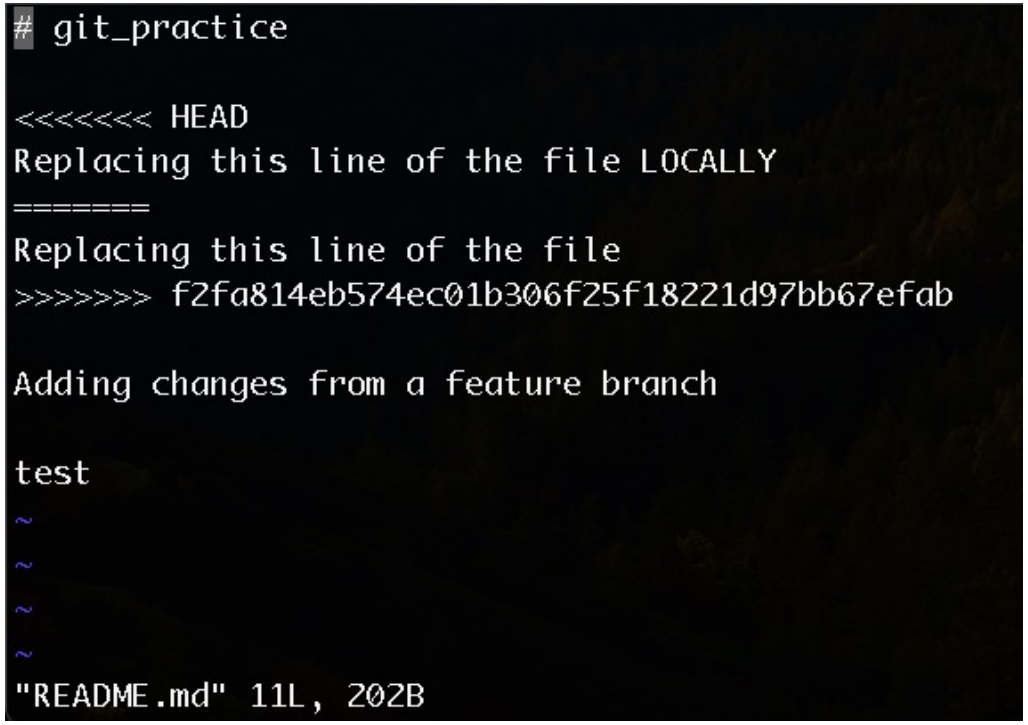
CONFLICT

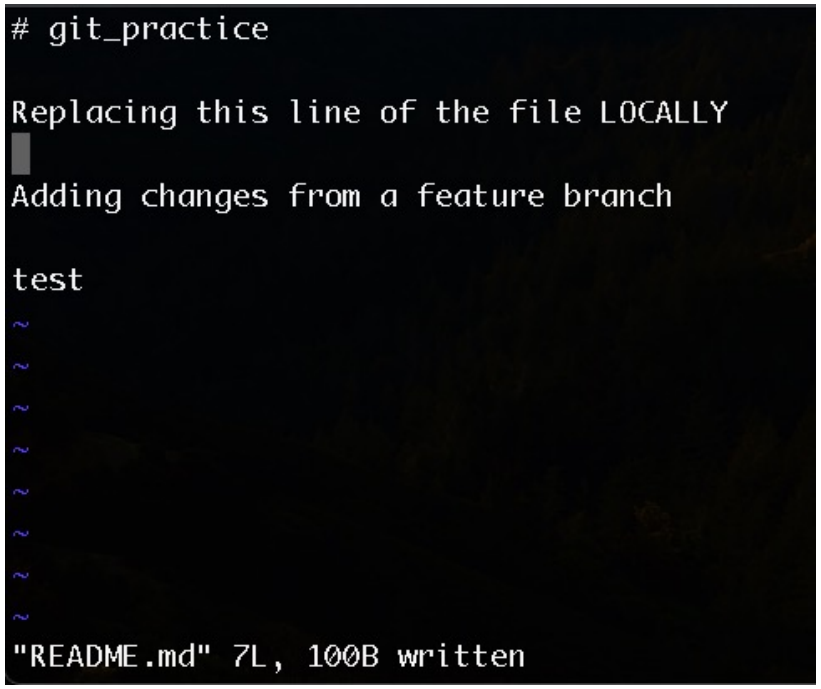
During a conflict, Git marks the conflicting lines inside the file with special symbols (<<<<<<<, =======, >>>>>>>).



CONFLICT

You simply select which version of the file you want to keep, delete the lines with the special symbols, and save the file. Add the file to staging area, commit and push again.

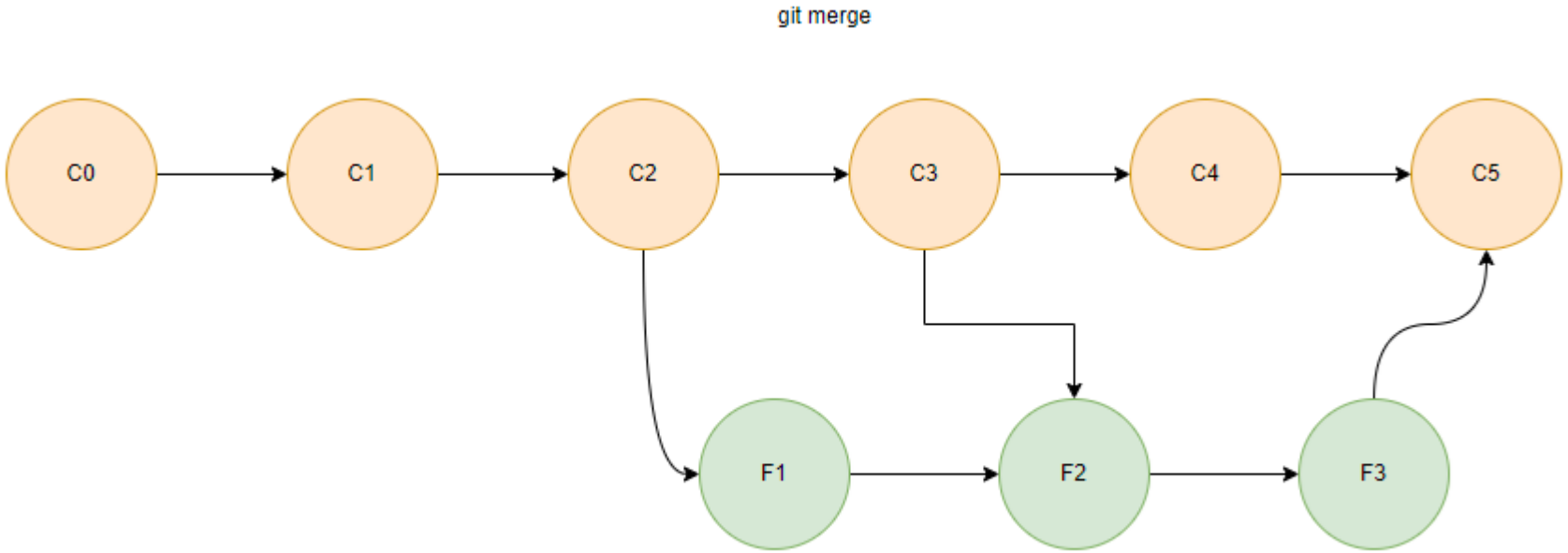




MERGING

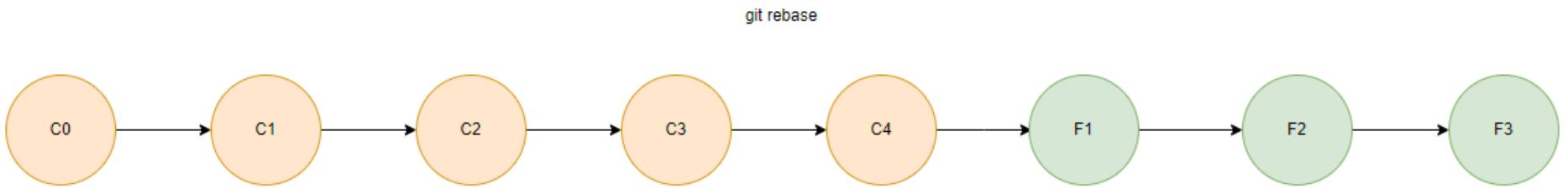
Merging combines changes from different branches into one consistent project history. When branches don’t overlap, Git merges automatically.

if merges do, you’ll be prompted to resolve conflicts before completing the merge.



REBASE

Rebasing moves your local commits to the tip of another branch, creating a linear history with no merge commits. It's useful for cleaning up your commit history before pushing or when you want your changes to appear as if they were made after the latest updates.



LAB 1.3 – PUSH,PULL,CONFLICT

LAB INTRODUCTION

In this lab, you'll practice handling situations where multiple people update the same repository, leading to potential push and pull conflicts. You'll see how Git reacts when your local branch is out of sync with the remote and learn how to resolve these situations safely.

By the end, you'll understand how to pull the latest changes, merge or rebase when needed, and push updates without overwriting others' work — key skills for real-world team collaboration.

POP QUIZ: VCS

What can help prevent merge conflicts?

A. Push often
B. Clone often
C. Pull often
D. Not communicating with team members

POP QUIZ: VCS

What can help prevent merge conflicts?

A. Push often
B. Clone often
C. Pull often
D. Not communicating with team members

POP QUIZ: VCS

Which conflict strategy is best for large teams?

A. Merging
B. Rebasing
C. Fast-forward
D. All of above

POP QUIZ: VCS

Which conflict strategy is best for large teams?

A. Merging
B. Rebasing
C. Fast-forward
D. All of above

VCS

Essential Commands

2

GIT REVERT

Git revert safely undoes a previous commit by creating a new one that reverses its changes. It doesn't rewrite history — instead, it preserves a clear record of what was changed and later undone.

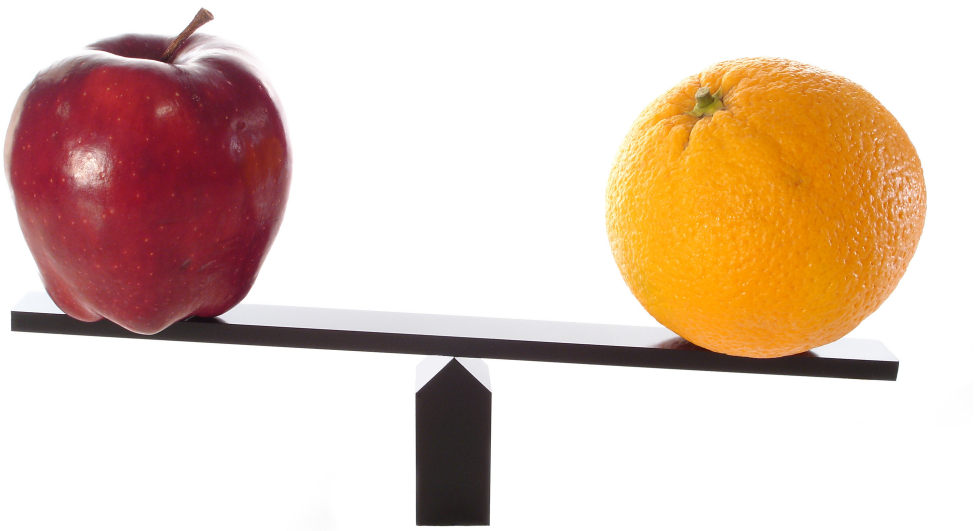
Use this when you've already pushed a commit to GitHub and need to fix it without affecting the repository's shared history.

GIT LOG

Git log shows the full history of your repository — every commit, author, date, and message. You can use options like --oneline, --graph, and --author to filter and visualize changes easily.

Learning to read logs helps you trace when features were added, who made them, and when bugs appeared.

GIT DIFF



Git diff compares changes between commits, branches, or your working directory.
It highlights what's been added (+) and removed (-), helping you review changes before committing.

This is one of Git's most valuable tools for code review and debugging — see exactly what's different before pushing.

GIT RESTORE

git restore discards unwanted local changes and restores files to their last committed state. It's useful when you make edits you don't want to keep or need to unstage files before committing.

git restore <file> → undo changes in working directory

git restore --staged <file> → unstage file but keep edits

GIT RESET

git reset moves the branch pointer and can change history — use carefully!
It's ideal for undoing commits you haven't pushed yet.

--soft → keep changes staged

--mixed → unstage changes but keep in working directory

--hard → delete all changes permanently

GIT FETCH

git fetch updates your local copy of the remote repository without changing your files.
It's a safe way to see what others have pushed before merging their changes.

Use before git pull to preview new commits

Combine with git log main..origin/main to compare local vs remote

GIT COMMANDS

Category	Command	Purpose
Stashing	git stash push -m "message"	Temporarily save work in progress
	git stash pop	Restore and remove stash
	git stash list	View saved stashes
Fetching / Pulling	git fetch	Download changes without merging
	git pull	Fetch + merge changes
Pushing	git push	Upload local commits to remote
Exploring History	git checkout <commit>	View old version (detached HEAD)
	git switch main	Return to main branch

Category	Command	Purpose
Configuration	git config --global user.name "Name"	Set username for all repos
	git config --global user.email "email@example.com"	Set email for all repos
	git config --list	View configuration
Viewing History	git log	View detailed commit history
	git log --oneline --graph --all	Compact, visual history
	git show	View details of last commit
Comparing Changes	git diff	Show unstaged changes
	git diff --cached	Show staged changes
Undoing Changes	git restore <file>	Discard working directory edits
	git restore --staged <file>	Unstage file
	git revert <commit>	Create new commit that undoes a previous one
	git reset --soft HEAD~1	Undo last commit, keep staged changes
	git reset --hard HEAD~1	Undo and delete changes (dangerous)

LAB 1.4 – ESSENTIAL COMMANDS

LAB INTRODUCTION

In this lab, you'll practice using Git's essential commands to manage your code more effectively. You'll learn how to view commit history, compare versions, undo mistakes, and safely revert or reset changes without losing progress.

You'll also explore advanced tools like stash, fetch, and restore, which help you switch tasks quickly, preview remote updates, and recover from common version control issues. By the end, you'll be comfortable handling nearly any situation you'll face in a real development workflow.

POP QUIZ: VCS

Which Git command safely undoes a previous commit by creating a new commit that reverses its changes?

A. git reset
B. git revert
C. git restore
D. git checkout

POP QUIZ: VCS

Which Git command safely undoes a previous commit by creating a new commit that reverses its changes?

A. git reset
B. git revert
C. git restore
D. git checkout

VCS DAY 1 COMPLETE

You've learned the core skills every developer needs to work with Git and GitHub — from creating repositories and collaborating online to resolving merge conflicts and using essential Git commands.

You now understand how to commit, push, pull, revert, restore, and manage changes confidently across local and remote environments. These skills form the foundation for real-world teamwork, version control, and professional software development.