A cover of a book

Description automatically generated

## Git and Github

## Version Control Unlocked

## 

***Contact Centre Management Fundamentals: Building a Strong Foundation***

*Communicate with Impact: Mastering Contact Centre Communication*

## Document Version

|  |  |
| --- | --- |
| Document Version | 1.0 |
| Date Published | 11 October 2025 |
| Publisher | MINDWORX ACADEMY |

## Document Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Initials | Description |
| 11 October 2025 | 1.0 | MvR | First Edition |

Table of Contents

[Git and Github 1](#_Toc211787426)

[Version Control Unlocked 1](#_Toc211787427)

[Document Version 2](#_Toc211787428)

[Document Change History 2](#_Toc211787429)

[Introduction to Github 4](#_Toc211787430)

[Git and GitHub 5](#_Toc211787431)

[How Git Sees Files 9](#_Toc211787432)

[Common Terms used in GIT and Their Meanings 11](#_Toc211787433)

[Why Git and GitHub Are Different 14](#_Toc211787434)

[Installing Git and GitHub: Step-by-Step for All Devices 16](#_Toc211787435)

[The Basic GitHub Workflow 23](#_Toc211787436)

[A Walk-around GitHub (Key Features) 29](#_Toc211787437)

[Getting Started: Account & First Repository 31](#_Toc211787438)

[GitHub in Action: Practical Experience for Software and Data Students 33](#_Toc211787439)

[Level Up with GitHub: Create, Commit, Collaborate 33](#_Toc211787440)

[Individual Activities 34](#_Toc211787441)

[Team Activities 37](#_Toc211787442)

# Introduction to Github

**Purpose:** Introduce learners to version control and the GitHub platform so they can confidently document, share, and collaborate on their work.

**Learning Outcomes**

By the end of this course, learners will be able to:

1. Explain what Git is and why version control matters.
2. Explain what GitHub is and how it complements Git.
3. Create a GitHub account and set up their first repository.

## Git and GitHub

**What is Git?**

Git is a version control (also called source control) system. It helps you track changes made to your files over time. You or your team can look back at the history, revert to older versions, or branch off to try new ideas safely. That means Git doesn’t care *what’s inside the file*, it just saves the differences between file versions.

So technically, Git can be used to track:

* Code files (.py, .js, .html, .java)
* Text files (.txt, .md)
* Documents (.docx, .pdf)
* Images and designs (.png, .jpg, .fig)
* Data files (.csv, .json, .xml)

Git takes a *snapshot* of all your project files each time you **commit** changes.

**What is GitHub?**

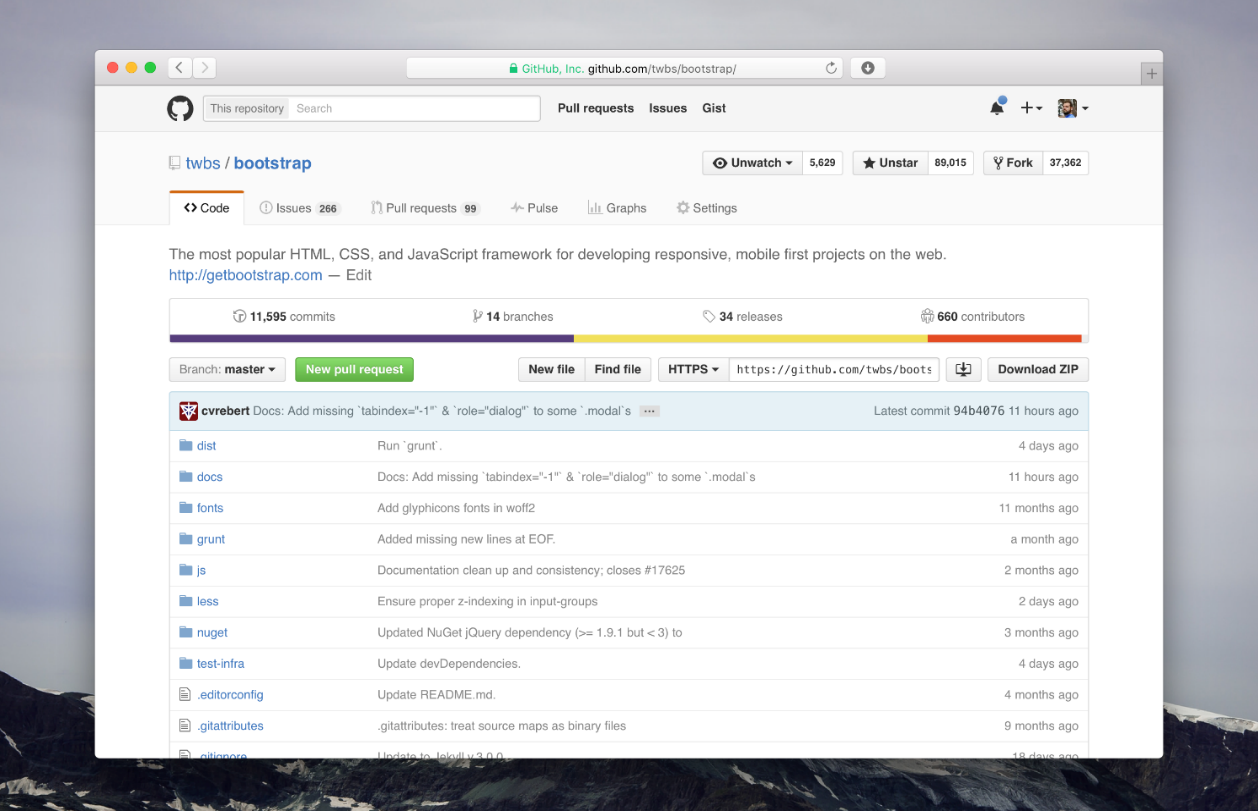
GitHub is an online platform where developers store, manage, and share their code. It uses a tool called **Git**, which helps track changes made to files.

Think of GitHub as a shared folder in the cloud, but with powerful features that allow you to:

* Work with others on the same project
* Keep a record of every change
* Review and merge improvements safely
* Allows everyone on your team can access the same repository in one place.
* Allows the use of features such as *Issues*, *Pull Requests*, *Projects to* help manage work beyond just code.
* Adopt professional habits.

A screenshot of a computer

AI-generated content may be incorrect.



A computer screen shot of a computer

AI-generated content may be incorrect.

**Why Learn the GitHub Process?**

Understanding GitHub is essential in software engineering because it reflects how real software teams work together.

Here are the main reasons:

* **Teamwork:** Developers can work on different parts of a project without interfering with each other’s work.
* **Version control:** You can return to an earlier version of the code if something goes wrong.
* **Code review:** Other team members can check your work before it becomes part of the main project.
* **Automation:** GitHub can connect with tools that test and deploy your code automatically.
* **Career value:** Many employers check your GitHub profile to see how you work and what projects you have done.

**What Git Can Track**

|  |  |  |
| --- | --- | --- |
| File Type | Example Use | Git Behaviour |
| .py, .js, .html | Code files (Python, JavaScript, web) | Perfect, shows exact lines added or removed |
| .md | Markdown / README files | Perfect, used for documentation |
| .txt | Plain text notes or pseudocode | Perfect , easy to see version differences |
| .csv | Data sets | Works, Git tracks changes in data |
| .jpg, .png, .pdf, .docx | Images and binary documents | Works, but Git can’t show what changed inside |
| .exe, .zip | Compiled or compressed files | Works, but not recommended (big and hard to diff) |

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## How Git Sees Files

Git doesn’t read your files like a person would, it simply records changes made to them.  
You can think of Git as a snapshot of your project every time you save (or *commit*) your work. Instead of storing whole new copies each time, Git only remembers what changed, which lines were added, removed, or edited. To Git, every file is part of a bigger picture called a repository, and it keeps track of each version so you can always look back, compare, or restore previous work. Git works best with plain-text files, that’s why programmers love it.  
For example:

* When you change one line of Python code, Git stores only *that line change*.
* When you edit a binary file (like a Word document), Git must store the *entire new version* because it can’t “see” the internal difference.

You *can* version .docx or .jpg files, but Git won’t show “what” changed just that the file changed.

**Think of It Like This**

|  |  |  |
| --- | --- | --- |
| **File Type** | **Git Can Track?** | **Git Can Show Differences (Diff)?** |
| Text-based (code, markdown, config) | Yes | Yes — line by line |
| Binary (Word, Excel, image, audio) | Yes | No — only version history |

Git is used by:

* **Architects / Designers** → to track versions of design assets.
* **Writers** → to track edits in books, scripts, or research papers (in Markdown or LaTeX).
* **Students** → to save essays or projects and revert to earlier drafts.
* **Data Scientists** → to version notebooks, scripts, and CSV files.
* **Educators** → to manage course notes and resources collaboratively.

So Git is really a **universal change-tracking tool**, coding is just its most famous use case.

**Best Practice for Beginners**

If you’re new to Git:

1. Use Git to store your **learning artefacts**:
   * Pseudocode (.txt)
   * Flowcharts (as .png)
   * Notes and reflections (.md)
   * Python scripts (.py)
2. Avoid committing large binary files (videos, large images, etc.) unless necessary.
3. Always include a **README.md** file — this is text-based and perfect for Git.

“With Git … you’re able to see the history of your files and recover the old version if you need it.”  
“With Git … you can create something called a branch … When you finish you can synchronise your branch with the original one … This operation is called merge.”  
(derived from your transcript)

**Its great becasue:**

* If a change breaks something, you can go back.
* You can work on a copy (branch) without affecting the main version.
* You can try features, experiments, and merge them in when ready.

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## Common Terms used in GIT and Their Meanings

| Term | Simple Meaning | In Everyday Words / Example |
| --- | --- | --- |
| Repository (Repo) | A **folder** or **container** that stores all your project files and their version history. | Think of it like a *digital binder* that holds everything for one project. |
| Commit | A **saved snapshot** of your work at a specific point in time. | Like saving a new version of a document — “Save as Version 2.” |
| Branch | A **copy of your project** where you can make changes without affecting the main version. | Like making a duplicate of an essay so you can test edits before submitting. |
| Merge | The process of **combining changes** from one branch into another. | Merging your edited copy back into the final essay once you’re happy with it. |
| Clone | To **make a local copy** of a repository from GitHub onto your computer. | Like downloading a folder from the cloud so you can work on it offline. |
| Push | Sending your local changes **up to GitHub** so they’re saved online. | Uploading your homework to the class drive. |
| Pull | Downloading the **latest changes** from GitHub to your computer. | Refreshing your notes to get your group’s latest updates. |
| Pull Request (PR) | A request to **merge your branch or changes** into the main project. Others can review, comment, or approve it. | Like asking your teacher to review and approve your edits before adding them to the group project. |
| Fork | Creating your **own copy** of someone else’s GitHub repository so you can work on it independently. | Like saving a copy of a shared Google Doc to your own drive to make personal edits. |
| Remote Repository | The **online version** of your repository stored on GitHub. | The version kept in the cloud that your team can access. |
| Local Repository | The **version of your repository on your own computer.** | The version stored on your laptop’s hard drive. |
| Stage / Staging Area | The **middle step** where you prepare files before committing them. | Like reviewing and selecting which files you’re ready to “save” in your next snapshot. |
| Diff (Difference) | The **comparison** between two versions of a file showing what changed. | Like using “Track Changes” in Word to see edits highlighted. |
| HEAD | The **latest commit** or the current place you’re working in your project’s history. | Think of it as your current position or “page” in the project’s timeline. |
| Version Control | The system that **tracks all changes** made to files over time. | Like a detailed timeline of every edit in your work, so you can always go back. |
| .git Folder | A hidden folder where Git stores all its tracking information. | Like the behind-the-scenes control room that remembers every change. |
| README File | A text file that explains **what the project is about**, how to use it, and who made it. | Like the “cover page” or “introduction” of your project. |
| GitHub | A **website** that stores repositories online and allows people to collaborate. | Like Google Drive for code and project files. |
| Git | The **software tool** that tracks and manages changes to files. | Like a history tracker that remembers every version of your work. |
| Commit Message | The **note** you write when you save a commit to describe what you changed. | Like writing “Added new introduction” when saving an updated essay. |
| Conflict (Merge Conflict) | When Git finds **two people changed the same part** of a file differently. | Like when two classmates edit the same sentence in different ways — someone must decide which version stays. |
| Contributor | Someone who **adds to or edits** a project on GitHub. | Any team member helping build the project. |
| Issue | A **task, bug, or feature request** tracked on GitHub. | Like writing a to-do note: “Fix typo on page 3.” |
| Project Board (Kanban) | A **visual task board** on GitHub for tracking progress. | Like sticky notes on a whiteboard labeled “To Do, Doing, Done.” |
| Wiki | A **built-in documentation area** in GitHub repositories. | Like a mini website where you explain how the project works. |
| Action (GitHub Actions) | Automated tasks that run when something happens — e.g., testing code after a push. | Like setting an alarm that checks your work every time you save. |
| License | A document that defines **how others can use your project.** | Like giving permission for classmates to read or copy your notes. |

**Tip:**

When you see unfamiliar Git terms, remember:

* Git = local tool (saving and versioning)
* GitHub = cloud platform (sharing and teamwork)

Together they form your **developer workflow**:

*“Edit → Stage → Commit → Push → Share → Collaborate.”*

## Why Git and GitHub Are Different

Although Git and GitHub are closely related and often used together, they serve different but complementary purposes in the software development process. Git was created to help developers track and manage changes to files on their local computers — it’s all about version control and maintaining a project’s history. GitHub, on the other hand, was built later as a cloud-based platform to make sharing and collaborating on those Git projects easier.  
In simple terms, Git helps you manage your work, while GitHub helps you share it with others.  
The difference exists because developers needed both a local tool for managing versions privately and a remote platform for teamwork, collaboration, and online storage , combining the power of individual control with global cooperation.

**Difference Between Git and GitHub**

| Feature / Aspect | Git | GitHub |
| --- | --- | --- |
| Definition | Git is a **version control system (VCS)** that tracks and manages changes to files on your **local computer**. | GitHub is an **online platform** that hosts Git repositories in the **cloud** for sharing, collaboration, and backup. |
| Type of Tool | Command-line or desktop software. | Cloud-based web service (also has mobile & desktop apps). |
| Purpose | To **track changes**, create versions, and manage branches of your files/projects. | To **store, share, and collaborate** on Git repositories with others. |
| Where It Runs | Installed and runs **locally** on your computer (Windows, macOS, Linux). | Runs **online** — accessed via a web browser or app. |
| Internet Required? | No (works offline — commits and branches can be made locally). | Yes (you need internet to push or pull changes). |
| Who Uses It | Individual developers for version control and local project management. | Teams, organizations, and open-source communities for collaboration. |
| Main Features | - Track file changes - Create commits - Branching and merging - Rollback to older versions | - Host repositories online - Manage permissions - Issues & pull requests - Project boards, Wikis, Actions (automation) |
| Storage Location | Files and version history stored on your **local drive**. | Files and repositories stored on **GitHub servers** in the cloud. |
| Interface Type | Command-line (Git Bash / Terminal) or GUI (GitHub Desktop, SourceTree). | Web interface and apps (browser, mobile, GitHub Desktop). |
| Collaboration | Mostly local — you can share using exported files or network drives. | Designed for teamwork — multiple people can contribute, comment, and review code. |
| Examples of Use | - Save project versions - Create branches for new features - Merge or revert changes | - Host and share project repos - Manage bugs and issues - Review pull requests - Publish open-source projects |
| Account Needed? | No — Git works locally without an account. | Yes — GitHub requires you to create an account at [github.com](https://github.com). |
| Created By | Linus Torvalds (2005) | Founded by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett (2008) |
| Ownership | Open-source software (free for everyone). | Owned by Microsoft (since 2018). |
| Example Command / Action | git commit -m "added login feature" | Creating a pull request or viewing code on GitHub.com |
| File Types Supported | Any file type (code, text, images, docs). | Any Git-tracked files uploaded to repositories. |
| Typical Workflow | Edit file → Stage changes → Commit → (Later push to GitHub). | View repo → Review commits → Merge pull request → Manage issues. |

## Installing Git and GitHub: Step-by-Step for All Devices

Now that you know the **difference between Git and GitHub**, let’s get your computer or mobile device ready. You only need to do this **once**. After setup, you can use Git and GitHub for all your future projects.

WINDOWS

**For Windows Users**

**Step 1: Download Git**

1. Open your browser and go to <https://git-scm.com/download/win>
2. The download will begin automatically. You’ll get a file named something like:
3. Git-2.x.x-64-bit.exe
4. When it finishes downloading, double-click that file to start installation.

**Step 2: Install Git**

1. Click **Next** on all screens, the default settings are perfect for beginners.
2. Keep the default editor as **Vim** or change it to **Notepad** if you prefer.
3. When you see “Adjusting your PATH environment”, choose:  
   “Git from the command line and also from 3rd-party software.”
4. When installation completes, click **Finish**.

**Step 3: Verify Installation**

1. Click the **Start** menu.
2. Type **Git Bash** → open it (it looks like a black command window).
3. Type this command and press **Enter**:
4. git --version
5. You should see something like:
6. git version 2.44.0

That means Git is ready!

**Step 4: Install GitHub Desktop (Optional but Recommended)**

If you prefer using buttons instead of typing commands, install the visual Git app.

1. Go to <https://desktop.github.com/>
2. Click **Download for Windows**.
3. Open the downloaded file (GitHubDesktopSetup.exe).
4. Once it installs, open **GitHub Desktop** from your Start menu.
5. Log in with your GitHub account (you can create one at <https://github.com>).

Now you can manage your repositories, make commits, and sync to GitHub **without typing any commands**.

**Video:**  
 [How to Install Git on Windows – Kevin Stratvert (YouTube)](https://www.youtube.com/watch?v=2j7fD92g-gE)

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**For macOS Users**

**Step 1: Check if Git is Already Installed**

MAC OS

1. Open **Terminal** (use Spotlight search and type “Terminal”).
2. Type:
3. git --version
4. If it says something like:
5. git version 2.x.x

You already have Git installed!

If you get a message saying “Command not found,” move to Step 2.

**Step 2: Install Git**

There are two easy ways:

**Option A – Easiest:**

1. In Terminal, type:
2. xcode-select --install
3. Click **Install** when prompted.
4. Wait for it to complete (it may take a few minutes).

**Option B – Manual Download:**

1. Go to <https://git-scm.com/download/mac>
2. Choose the macOS installer and follow the on-screen prompts.

**Step 3: Install GitHub Desktop (Recommended for Beginners)**

1. Visit <https://desktop.github.com/>
2. Click **Download for macOS**.
3. Open the .zip file, drag **GitHub Desktop** into your **Applications** folder.
4. Open GitHub Desktop and sign in with your GitHub account.

**Video:**  
[Install Git on macOS – Beginner’s Guide (YouTube)](https://www.youtube.com/watch?v=9XRGWfFzRXE)

**For Linux Users**

Git usually comes pre-installed, but you can check or install it manually.

LINUX

**Step 1: Check if Git is Installed**

1. Open the **Terminal**.
2. Type:
3. git --version
4. If you see “git version 2.x.x”, you’re done.

**Step 2: Install Git (if not found)**

**For Ubuntu / Debian systems:**

sudo apt update

sudo apt install git

**For Fedora:**

sudo dnf install git

**For Arch Linux:**

sudo pacman -S git

**Step 3: Verify Installation**

git --version

**Video Tutorial:** [Install Git on Linux (Ubuntu Example) – YouTube](https://www.youtube.com/watch?v=0fKg7e37bQE)

**For Android Users**

You can’t install the full Git software like on a computer,  
but you can use **apps that connect to GitHub** to view, comment, or manage projects.

ANDROID

**Option 1: GitHub Official App (Recommended)**

1. Open **Google Play Store**.
2. Search for **GitHub**.
3. Tap **Install**.
4. Open the app and sign in (or create an account).

*What you can do:*

* Browse repositories
* View and comment on issues or pull requests
* Star and fork projects
* Read files and code

GitHub for Android (Play Store)

**Option 2: Termux (Advanced)**

If you’re comfortable with a command line:

1. Install **Termux** from the Play Store.
2. Open it and type:
3. pkg install git
4. You can now use Git commands like on a computer.

**Option 3: Pocket Git (Paid)**

* For developers who want full Git control with visuals.
* Paid app on Google Play.  
   Pocket Git

**Beginner Tip:**  
Start with the **official GitHub app**, it’s great for exploring repositories and understanding how GitHub looks and feels.

**For iPhone / iPad Users**

iPHONE

**Option 1: GitHub Official App**

1. Open the **App Store**.
2. Search for **GitHub**.
3. Tap **Get** and install.
4. Open and sign in with your GitHub account.

*What you can do:*

* View your repositories
* Comment on issues
* Review pull requests
* Read project files and docs

GitHub on App Store

**Option 2: Working Copy (Best for Full Git on iOS)**

If you want to commit and push code or files:

1. Go to Working Copy on App Store
2. Install the app.
3. Sign in with GitHub (when prompted).
4. You can clone repos, commit changes, and push updates — just like on a computer!

**Beginner Tip:**  
Use **GitHub App** for browsing and feedback,  
and **Working Copy** if you plan to actively edit files on iPad.

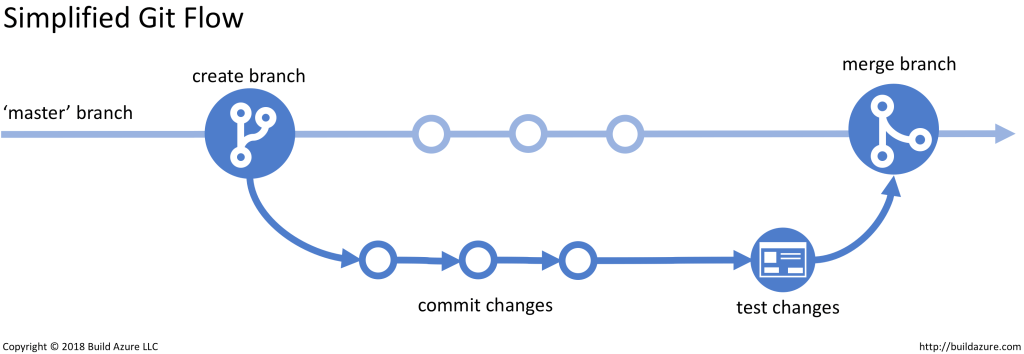
|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Install Git?** | **Install GitHub Desktop?** | **Use GitHub Website / App?** |
| **Windows** | Yes | Optional | Yes |
| **macOS** | Yes | Optional | Yes |
| **Linux** | Yes | X | Yes |
| **Android** | Limited (via Termux) | X | GitHub App |
| **iPhone / iPad** | Use Working Copy | X | GitHub App |

**Helpful Links**

|  |  |
| --- | --- |
| **Purpose** | **Link** |
| Download Git (Windows, macOS, Linux) | <https://git-scm.com/downloads> |
| Download GitHub Desktop | <https://desktop.github.com/> |
| GitHub App (Android) | Google Play Store |
| GitHub App (iOS) | App Store |
| Working Copy (iOS Git Client) | <https://workingcopyapp.com/> |
| Practice Git (in browser – no install) | <https://learngitbranching.js.org/> |
| GitHub Learning Portal | <https://skills.github.com/> |

## The Basic GitHub Workflow

The following workflow shows the standard process developers follow when using GitHub.



**Step 1: Create or Fork a Repository**

A repository (often called a “repo”) is like a folder for your project.  
You can start a new repository for your own project or make a copy (fork) of someone else’s project to work on it yourself.

Example:  
You find a simple calculator project online and fork it to add a scientific calculator mode.

**Step 2: Clone the Repository**

Once you have a repository on GitHub, you can make a copy of it on your own computer. This is called *cloning*.  
Cloning lets you edit files locally (on your computer) without changing the online version yet.

**Step 3: Create a New Branch**

A branch is a separate version of the project where you can safely make changes.  
You should not work directly on the main branch (usually called “main” or “master”) because that branch should always contain the working version of the project.

Creating a branch allows you to experiment and add new features without breaking anything.

**Step 4: Make Changes and Commit Them**

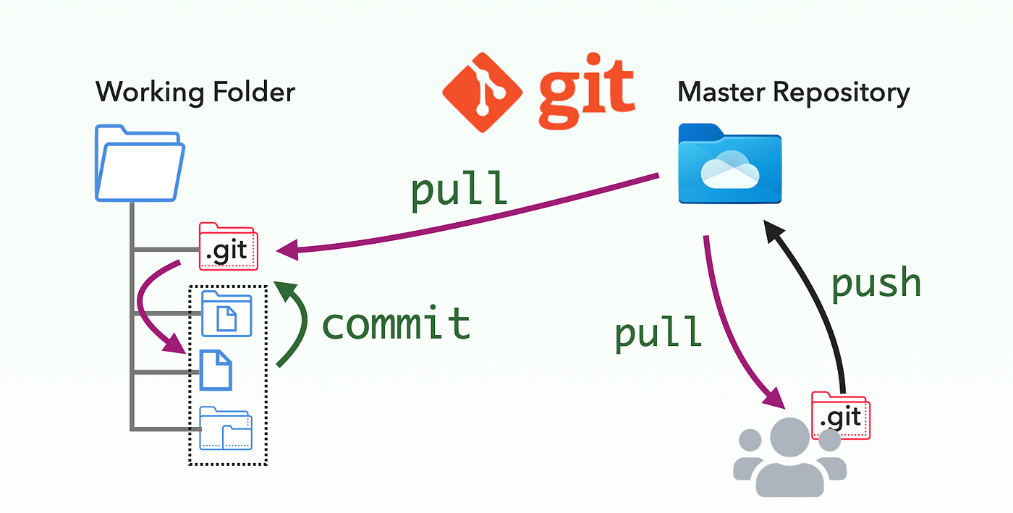
After editing your files, you save your progress by creating a “commit.”  
A commit is like a checkpoint — it records what you changed and when you did it.  
Each commit should include a short message describing the change, such as “Added login page design.”

**Step 5: Push Changes to GitHub**

When you are ready to share your changes, you “push” them to GitHub.  
This means you upload your local changes to the online version of the repository so that others can see your work.

**Step 6: Open a Pull Request**

After pushing your branch to GitHub, you open a “pull request.”  
A pull request asks the project owner or team members to review your changes and decide whether to merge them into the main branch.

During the review, others can:

* Comment on your code
* Suggest improvements
* Approve your changes

**Step 7: Merge the Pull Request**

Once your changes are approved, they are “merged” into the main branch.  
This means your new work officially becomes part of the main project.  
After merging, the branch you used can usually be deleted, since it has served its purpose.

**Step 8: Pull Updates from Main**

If other people have been updating the project, you can bring their latest changes into your local copy by “pulling” updates.  
This keeps your work up to date and prevents conflicts when you make new changes.

**Example of a Real Project Workflow**

Imagine a small team building an online bookstore.

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Task | Branch | Pull Request |
| Alice | Adds a search feature | feature-search | PR #12 |
| Bob | Fixes a checkout error | bugfix-checkout | PR #13 |
| Carol | Creates an admin dashboard | feature-dashboard | PR #14 |

Each person creates a branch, makes changes, opens a pull request, and after review, their code is merged into the main branch. This process ensures that the main project stays stable while allowing everyone to contribute.

**Summary of the GitHub Process**

The GitHub process usually follows this order:

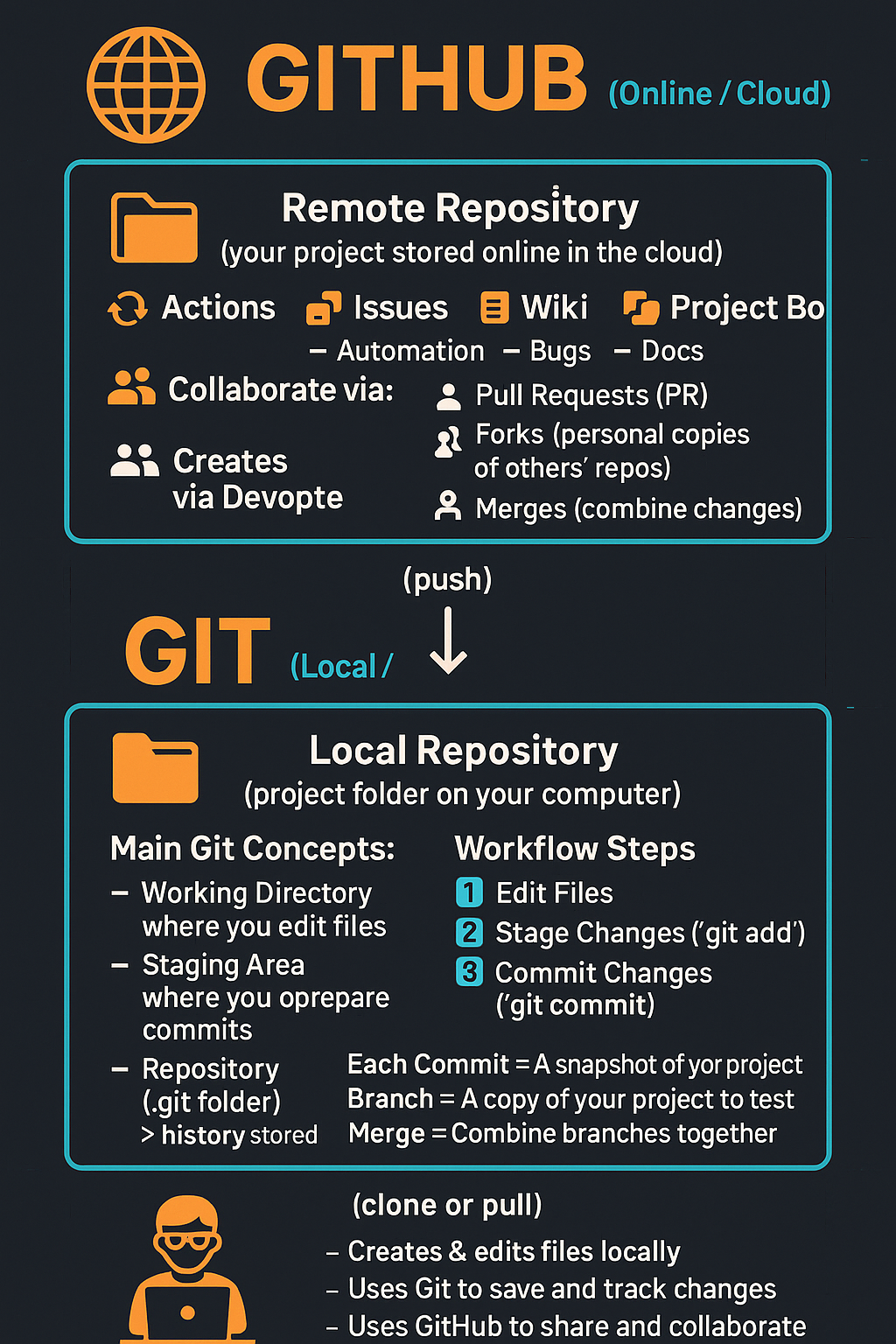
1. Create or fork a repository
2. Clone it to your computer
3. Create a branch
4. Make and commit changes
5. Push your changes to GitHub
6. Open a pull request
7. Merge after review
8. Pull updates from the main branch

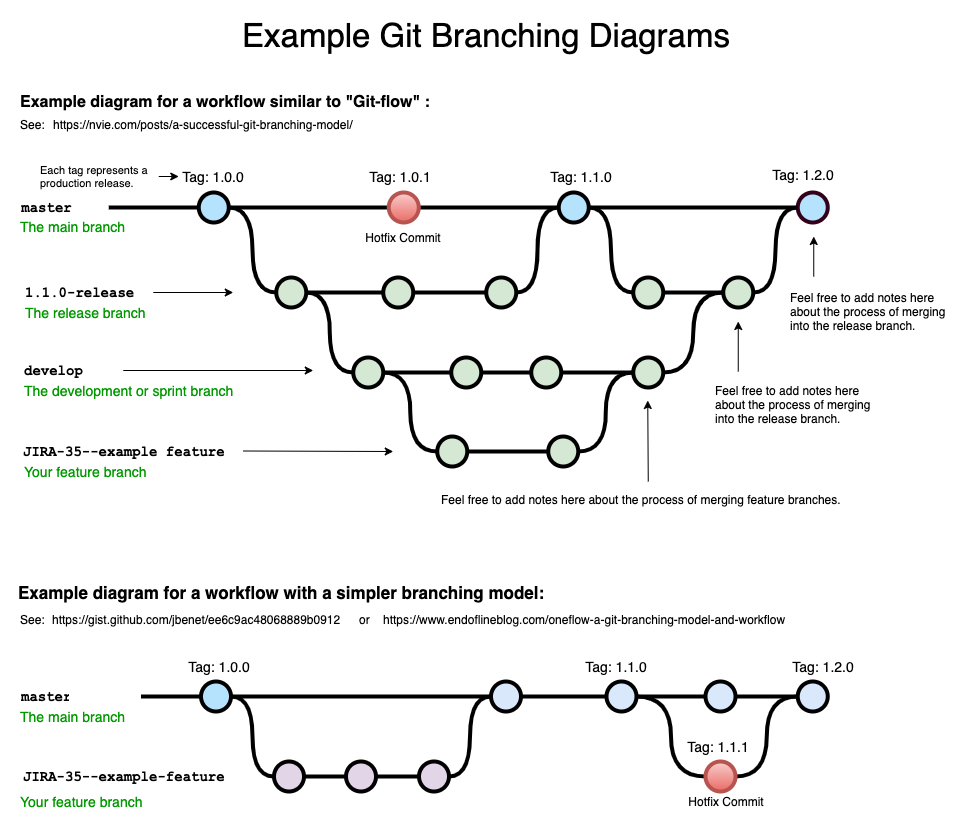
**Why This Matters in Software Engineering**

Learning GitHub is more than just learning a tool. It teaches you how professional software development teams manage code, work together, and maintain quality.

In advanced software engineering, GitHub represents:

* The foundation of collaborative development
* The practical side of version control systems
* The bridge between programming, teamwork, and project management





**Video to watch:**

[](https://www.youtube.com/embed/tRZGeaHPoaw?feature=oembed)

**Suggested activity:**

* Ask learners to think of a file they edit often (e.g., a document) and describe: “What happens if I change it, then want the old version back?”
* Show a simple branch example: main → branch “new-feature” → merge back.

Notes:

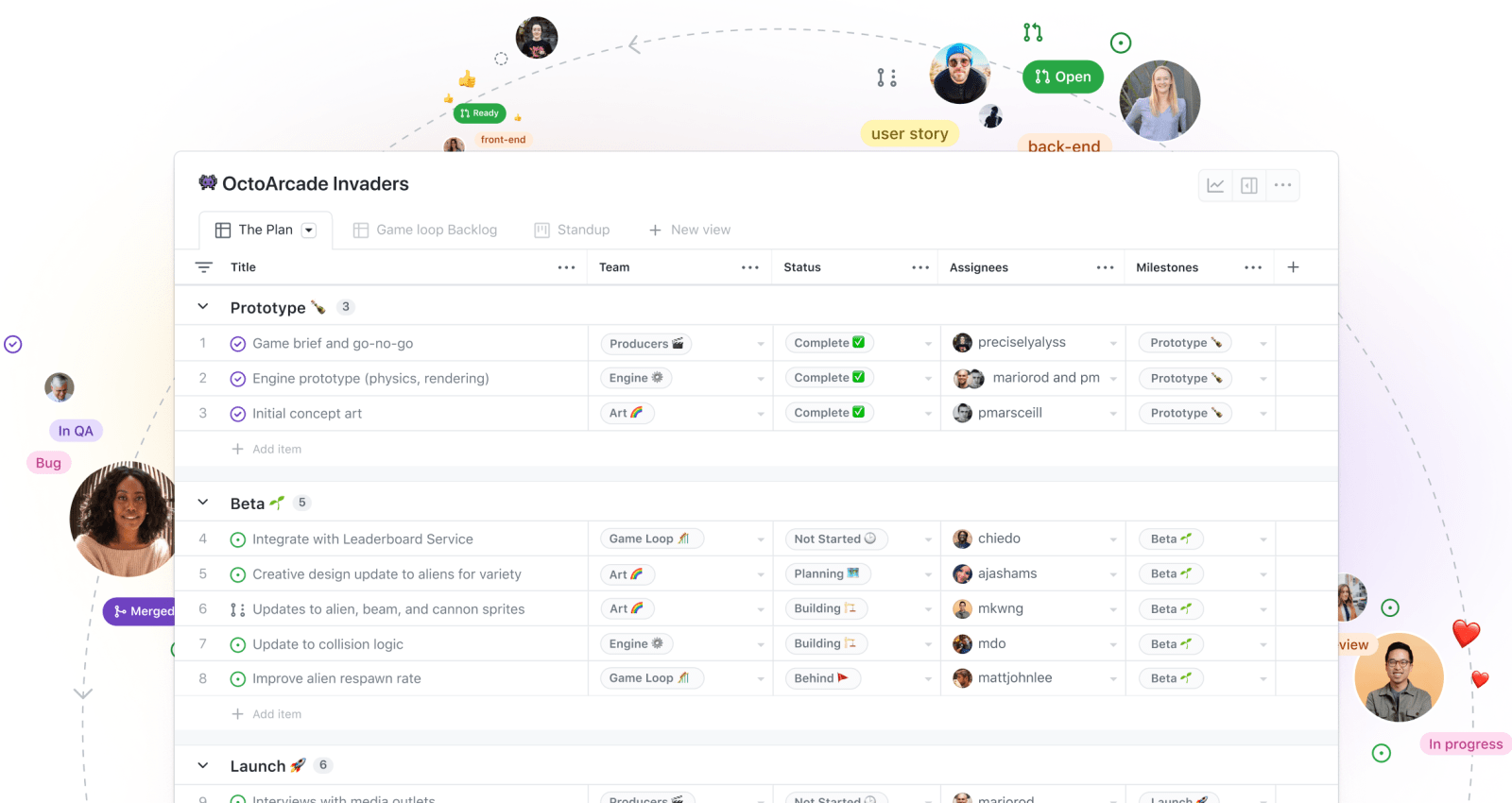
|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## A Walk-around GitHub (Key Features)

**Explanation:**  
This section explores core parts of a GitHub repository and what learners should know at this stage. From your transcript, the features covered:

* *Code tab* — view files/folders; click into README.
* *Editing files in browser* — modify README via web interface.
* *Issues* — track bugs, feature requests.
* *Pull Requests* — proposed changes that can be reviewed & merged.
* *Actions* — automation (CI/CD) (intro-level mention).
* *Projects* — Kanban board style for tasks.
* *Wiki* — documentation area.
* *Security/Insights* — overview of project health/activity.

Understanding these features means students will know *what* GitHub can do for collaboration and documentation — even before coding.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

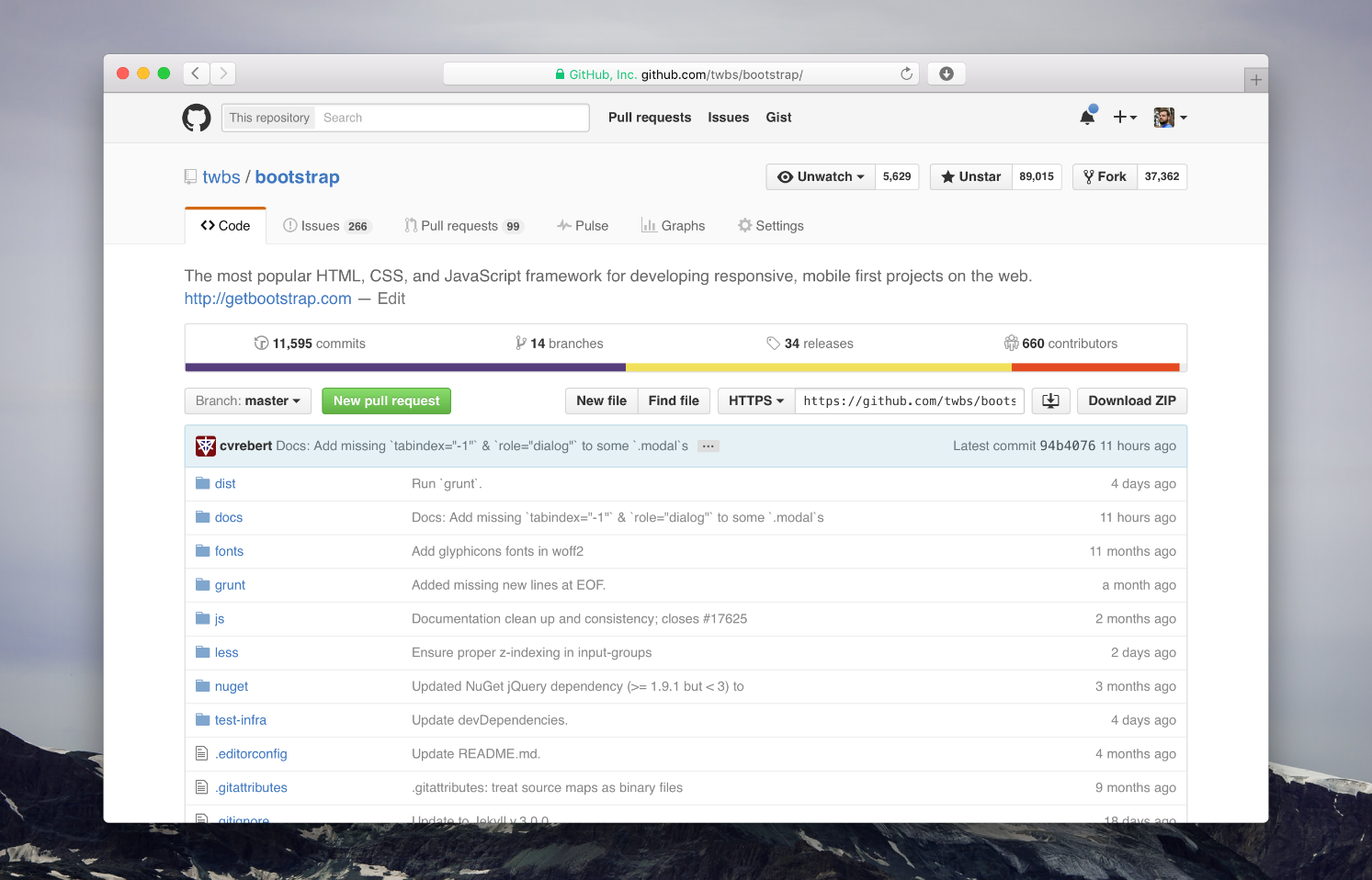
## Getting Started: Account & First Repository

**Explanation & Steps:**

1. Go to [github.com](https://github.com) → click **Sign up**.
2. Provide email, password (at least 8 chars, include number/lowercase).
3. Choose a unique username (professional).
4. Confirm email via code sent.
5. After logged in, click **New repository** → name it something like CTPS\_[YourName] (CTPS = Computational Thinking & Problem Solving).
6. Choose initialisation options: README file (create).
7. Upload your first file: e.g., my-goals.txt or a flowchart image.
8. Create folder structure (optionally): flowcharts/, pseudocode/, reflections/.
9. Write your README.md: explain what the repository is for, what you will upload here.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

# GitHub in Action: Practical Experience for Software and Data Students

## Level Up with GitHub: Create, Commit, Collaborate

**Purpose**

To provide learners with hands-on experience using Git and GitHub through practical, guided activities. This module moves from conceptual understanding to real-world usage, helping students apply version control in individual and collaborative contexts.

**Learning Outcomes**

By completing this module, learners will be able to:

* Create and manage repositories on GitHub
* Use branches and commits to track changes
* Work with GitHub Pull Requests for code review and merging
* Collaborate with peers through issues and project boards
* Simulate a real-world development or data science project workflow using GitHub

**Tools Required**

* A GitHub account (sign up at <https://github.com>)
* GitHub Desktop (download from <https://desktop.github.com>) or Git installed locally
* A text/code editor (e.g., Visual Studio Code)
* Internet access

## Individual Activities

**Activity 1: Set Up and Explore GitHub**

**Objective:** Learn to navigate GitHub and personalize your account.

**Instructions:**

1. Go to <https://github.com> and sign in.
2. Update your profile: add a profile picture, bio, and set a professional username.
3. Search for three open-source repositories that interest you. Star them.
4. In a plain text file (starred-projects.txt), write:
   * The names of the repositories
   * One reason each project appealed to you
5. Save and upload the file to a personal repository (see next activity).

**Where to get help:** Use GitHub’s official documentation https://docs.github.com or their search bar to look up “Edit profile”, “Starring repositories”, etc.

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Activity 2: Create a Personal Repository**

**Objective:** Set up a GitHub repository with initial content and structure.

**Instructions:**

1. On GitHub, click “New Repository”.
2. Name the repository my-portfolio.
3. Select **Public**, and check the box for “Initialize with README”.
4. In the README file, add:
   * A short bio
   * Learning goals
   * Topics or technologies of interest
5. Create the following folders using the “Add file” > “Create new file” interface:
   * projects/
   * notes/
   * images/
6. Upload a .jpg or .png file (e.g., a screenshot or project diagram) to the images/ folder.
7. Create a new file reflection.txt and write one paragraph about why version control matters in software or data projects.

**Where to get help:** Use GitHub Docs, or refer to the “Getting Started” section in your course notes.

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Activity 3: Practice Branching and Pull Requests**

**Objective:** Understand the branching model and how changes are reviewed.

**Instructions:**

1. Go to your my-portfolio repository.
2. Click “Branch: main” → create a new branch called add-project-section.
3. On this branch, edit the README.md to add a new section titled “My Favourite Project” and add a placeholder sentence.
4. Commit the change with a message like:  
   Added placeholder for project section in README
5. Open a Pull Request from add-project-section to main.
6. Merge the pull request.

**Where to get help:** Look up “Create a branch”, “Open a pull request” on <https://docs.github.com>

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## Team Activities

**Activity 4: Collaborate with a Partner**

**Objective:** Simulate a collaboration using GitHub’s workflow.

**Instructions:**

1. Pair with a classmate.
2. One partner creates a new GitHub repository called team-collab-demo.
3. Under repository settings, add the other partner as a **Collaborator**.
4. Each partner adds a file (e.g., john.md, maria.md) with:
   * Their name
   * A short paragraph about their background or interest in tech
5. Each partner creates a branch for their addition and submits a Pull Request.
6. Partners review and merge each other’s Pull Requests.

**Where to get help:** GitHub’s help section on “Managing access to your repository” and “Reviewing Pull Requests”.

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Activity 5: Simulated Team Project**

**Objective:** Work as a group to complete a mini-project using GitHub.

**Project Options (choose one based on your stream):**

* *Software Engineering*: Build a basic to-do list web app using HTML/CSS/JavaScript.
* *Data Science*: Create a small dataset (CSV) and generate a visualization using Python or Jupyter Notebook.

**Instructions:**

1. Form teams of 3–5 students.
2. One team member creates a repository: group-project-[topic-name].
3. Enable GitHub **Projects** and create a task board (To Do, In Progress, Done).
4. Create Issues for major tasks and assign them to team members.
5. Each member creates a branch for their task, commits work, and opens a Pull Request.
6. After review, team members merge their Pull Requests into main.
7. Add or update the README.md to include:
   * Project overview
   * Team member names
   * How to run the code or open the notebook
8. At the end, pull all updates and clean up branches.

**Where to get help:** GitHub Docs > Projects, Issues, Collaboration.

Notes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Git & GitHub Course – Selected Bibliography**

* **Pro Git (2nd Edition)** by Scott Chacon and Ben Straub
  + Free online book covering Git fundamentals in depth
  + Available at: <https://git-scm.com/book/en/v2>
* **GitHub Docs – Official Documentation**
  + Comprehensive help guides and tutorials for GitHub users
  + Available at: <https://docs.github.com>
* **GitHub Skills – Interactive Learning Platform**
  + Self-paced, hands-on GitHub learning modules for all levels
  + Available at: <https://skills.github.com>
* **Learn Git Branching – Interactive Visual Tutorial**
  + Popular visual tool for mastering Git commands and branching
  + Available at: <https://learngitbranching.js.org>
* **Atlassian Git Tutorials**
  + Git concepts, workflows, and best practices from Atlassian
  + Available at: https://www.atlassian.com/git
* **GitHub YouTube Channel – Video Tutorials**
  + Demos, walkthroughs, and GitHub product updates
  + Available at: <https://www.youtube.com/github>
* **Git CLI Reference Manual**
  + Authoritative command-line reference for all Git functions
  + Available at: <https://git-scm.com/docs>
* **Stack Overflow & GitHub Community Forums**
  + Peer-to-peer Q&A, troubleshooting, and best practice sharing
  + <https://stackoverflow.com> | https://github.community