Git Basics for Students

This guide will help you get started with Git, a powerful version control tool, and GitHub, a popular platform for hosting Git repositories. By following these steps, you'll learn the essential Git commands needed to manage your code effectively.

Table of Contents

To use Git on your local machine, you first need to install it. Most machines will have it preinstalled.

Windows: Download and install Git from https://git-scm.com/download/win.

macOS: Git is often pre-installed. Run git –version in Terminal to check. If it's not installed, run:

xcode-select –install

Linux: Use the package manager to install Git. For example:

sudo apt install git # Debian/Ubuntu sudo yum install git # CentOS/Fedora

Verify your installation by running:

git –version

Creating a GitHub Account

Go to https://github.com/. Sign up for a free account. After creating your account, you can host repositories, collaborate with others, and showcase your work.

Initial Setup

Step 1: Configure Git Profile

Before using Git, configure your Git profile:

git config –global user.name “Your Name” git config –global user.email “your_email@example.com”

This information will be attached to your commits.

Step 2: Verify Your Installation

To confirm that Git is installed and configured correctly, use:

git –version git config –list

Basic Git Commands

Initializing a Repository To start tracking a project with Git, navigate to your project’s root directory and initialize Git:

cd path/to/your/project git init

This creates a hidden .git folder in your project directory.

Creating a .gitignore File

A .gitignore file tells Git which files or directories to ignore. Create a .gitignore file in the root of your project and add file patterns to ignore, e.g.,

node_modules/ *.log .DS_Store

Adding Files to the Staging Area

The staging area is where you gather changes before committing. To add a file to the staging area, use:

git add filename # add specific file

git add . # add all files in the directory

Committing Changes

Once your files are staged, commit them with a message:

git commit -m “Your descriptive message”

Checking Repository Status

To see which files are staged, unstaged, or untracked, use:

git status

Viewing Commit History

To view the history of commits:

git log

Press q to exit the log view.

Creating and Switching Branches

Branches allow you to work on features independently.

Create a new branch:

git branch branch_name

Switch to a branch:

git checkout branch_name

Create and switch to a new branch:

git checkout -b branch_name

Merging Branches

To merge changes from another branch into the current branch:

Switch to the branch you want to merge into, typically main:

git checkout main

Merge the other branch:

git merge branch_name

Link your local project to the GitHub repository:

git remote add origin https://github.com/username/repo_name.git

Push your code to GitHub:

git push -u origin main

For subsequent pushes, you can use:

git push

Pulling Changes from GitHub:

To get changes from the remote repository, use:

git pull origin main

Working with GitHub

Forking a Repository:

On GitHub, find the repository you want to contribute to. Click Fork to create a copy under your account.

Cloning a Repository:

Copy the repository URL on GitHub.

Clone the repository to your local machine:

git clone https://github.com/username/repo_name.git

Creating Pull Requests:

After making changes on your branch, push to GitHub. Go to your GitHub repository and click New Pull Request. Add a title and description, then click Create Pull Request. Extra Tips Undo Last Commit: git reset –soft HEAD~1

Remove File from Staging Area:

git reset filename

Delete a Branch:

git branch -d branch_name

Additional Git Activities with Syntax

Some optional extras if you manage to finish everything. ## Create and Work with a New Branch This activity teaches students to create and switch branches, make changes, and then switch back without merging.

Create a new branch called feature-update using the command git branch feature-update. Switch to the feature-update branch with git checkout feature-update. Create a new file in this branch called feature.txt using your text editor, and add some content such as “This is a new feature.” Add and commit this new file with the commands git add feature.txt and git commit -m “Add feature.txt to feature-update branch”. Switch back to the main branch without merging by using git checkout main.

Merge Branches and Resolve Conflicts

This activity teaches students how to handle merge conflicts and resolve them.

While on the main branch, open the feature.txt file and add a different line of text, for example, “This line is from the main branch.” Add and commit this change in the main branch with git add feature.txt and git commit -m “Add conflicting line in feature.txt on main branch”. Switch back to the feature-update branch with git checkout feature-update. Merge the main branch into the feature-update branch using git merge main. Git should detect a conflict in feature.txt. Open feature.txt in your text editor and manually resolve the conflict by choosing which changes to keep, or by combining them. Save the file. After resolving the conflict, add the resolved file with git add feature.txt and complete the merge with git commit -m “Resolve conflict in feature.txt”. Push the changes in feature-update to GitHub using git push origin feature-update.

Collaborate Using GitHub

This activity helps students practice collaborating on GitHub by forking and cloning repositories.

Pair up with another student or use a secondary GitHub account if working alone.

One student should create a new repository on GitHub, initialize it with a README file, and invite their partner as a collaborator from the repository settings. The second student should clone the repository to their local machine with git clone . Once cloned, create a new branch called collaboration using git checkout -b collaboration. Create a new file named collab.txt and add your name and a fun fact. For example, “Name: Alex, Fun Fact: I love hiking.” Add and commit this file with git add collab.txt and git commit -m “Add collab.txt with name and fun fact”. Push the collaboration branch to GitHub with git push origin collaboration. The original repository owner should pull down the new branch using git pull origin collaboration, review collab.txt, and merge collaboration into main using git checkout main and git merge collaboration. ## Create and Use a .gitignore File This activity helps students learn how to create and use a .gitignore file to prevent tracking unnecessary files.

In your project, create a directory named logs and inside it, add a file called error.log. Add some example content to error.log such as “Error: Missing file.” In the root of your project, create a .gitignore file using your text editor and add the following lines: logs/ to ignore the entire logs directory. *.log to ignore all .log files. Save and close .gitignore. Verify that error.log does not appear in the staging area by running git status.

Fetch and Pull Changes from a Remote Repository

This activity reinforces fetching and pulling changes from a shared repository.

Work in pairs for this activity.

Student A should add a new file in the shared GitHub repository named shared.txt with a message like “This is a shared file.” Student A commits and pushes the changes to GitHub using git add shared.txt, git commit -m “Add shared.txt”, and git push origin main. Student B should fetch the new changes by using git fetch origin main to see the new branch in the remote repository. Student B should then pull down the changes with git pull origin main to retrieve and view the shared.txt file.

Review and Revert Commits

This activity helps students learn how to review commit history and revert changes.

Create a new file named experiment.txt and add content such as “Experimenting with Git revert.” Add and commit this file using git add experiment.txt and git commit -m “Initial experiment commit”. View the commit history with git log to identify the commit ID of experiment.txt. To undo the last commit without deleting the actual changes in the file, use git reset –soft HEAD~1. Check the status with git status to confirm that the changes are still staged but the commit is undone.

Now it’s time to start building a portfolio, with regular commits to show employers your work and your version control skillset!