Git Tutorial Revision

Diego Santiago
dsantiagooxy.edu
Occidental College

1 Introduction to Version Control with Git

In modern software development, managing changes to code is essential for collaboration, tracking progress, and ensuring project stability. Version control systems like Git provide a structured approach to managing these changes effectively. In this section, we'll introduce you to the basics of Git and why it's beneficial for your projects.

1.1 What is Version Control?

Version control is a system that records changes to files over time. It allows you to revisit previous versions of your files, track modifications, and coordinate work among multiple developers. Think of it as a time machine for your code, enabling you to revert to a previous state or compare different versions effortlessly.

1.2 Why Use Version Control?

Using version control offers several advantages for both individuals and teams:

- **Collaboration**: Version control enables seamless collaboration among team members. Multiple developers can work on the same codebase simultaneously without interfering with each other's changes. Git provides mechanisms for merging these changes intelligently.
- **History Tracking**: Every change made to the codebase is recorded in the version control system. This history allows you to understand who made what changes, when they were made, and why they were made. It's invaluable for troubleshooting issues and understanding the evolution of your project.
- **Backup and Recovery**: Version control serves as a backup mechanism for your code. If something goes wrong, such as accidental deletion or corruption of files, you can revert to a previous known-good state with ease.
- **Experimentation and Risk-Free Development**: Version control encourages experimentation by providing a safe environment to try out new ideas. You can create branches to isolate changes and test them independently. If an experiment fails, you can discard the branch without affecting the main codebase.

Now that you understand the benefits of version control, let's dive into the fundamentals of Git and how you can start using it for your projects.

2 Installing Git

To use Git locally on your machine, you need to install it. Download the installer for your operating system from Git's official website and follow the installation instructions.

3 Configuring Git

After installing Git, you need to configure it with your name and email address. Open a terminal or command prompt and run the following commands, replacing `Your Name` and `your.email@example.com` with your actual name and email address:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

4 Creating a GitHub Account

If you haven't already, go to GitHub's website and sign up for an account. It's free!

5 Creating a Repository

Now that you have Git installed and configured, you can start creating repositories to store your projects. A repository, or repo for short, is a collection of files and folders that make up a project.

1. Click on the "+" icon in the top right corner of the page.
2. Select "New repository" from the dropdown menu.
3. Enter a name for your repository, a description (optional), choose visibility (public or private), and initialize with a README file if needed.
4. Click on the "Create repository" button.

6 Cloning a Repository

To work with an existing repository on your local machine, you need to clone it. Go to the repository on GitHub and click on the "Code" button. Copy the URL provided.

Open a terminal or command prompt, navigate to the directory where you want to clone the repository, and run the following command:

```
git clone <repository_url>
```

Replace `<repository_url>` with the URL you copied earlier.

7 Making Changes

Now that you have a local copy of the repository, you can make changes to the files in it using your preferred text editor or IDE.

8 Committing Changes

Once you've made changes to the files, you need to commit them to the repository. Open a terminal or command prompt, navigate to the repository directory, and run the following command:

```
git add .
git commit -m "Descriptive commit message here"
```

The `git add .` command stages all changes for the next commit. Replace `"Descriptive commit message here"` with a brief but informative message describing the changes you've made.

9 Pushing Changes

After committing your changes locally, you need to push them to the remote repository on GitHub. Run the following command:

```
git push origin master
```

This command pushes the committed changes from your local `master` branch to the remote repository.

10 Pulling Changes

If changes have been made to the remote repository by other collaborators, you can pull those changes to your local repository. Run the following command:

```
git pull origin master
```

This command fetches the changes from the remote `master` branch and merges them into your local repository.

11 Branching

Branching allows you to work on new features or fixes without affecting the main codebase. To create a new branch, run the following command:

```
git checkout -b <branch_name>
```

Replace `<branch_name>` with the name of your new branch. You can then make changes and commit them to this branch without affecting the `master` branch.

12 Merging

Once you've finished working on a feature or fix in a separate branch, you can merge it back into the `master` branch. First, switch to the `master` branch:

```
git checkout master
```

Then, merge the changes from your branch into `master`:

```
git merge <branch_name>
```

If there are conflicts during the merge, Git will notify you. You can resolve the conflicts by editing the conflicting files manually. After resolving the conflicts, add the files and commit the changes.

13 Stashing

If you need to switch branches but you're not ready to commit your changes, you can stash them. Stashing takes everything in your working directory – that is, your modified tracked files and staged changes – and saves it on a stack of unfinished changes that you can reapply at any time.

To stash your changes, run the following command:

```
git stash
```

To apply the most recent stash, run:

```
git stash apply
```

14 Collaborating with Others

GitHub facilitates collaboration by allowing multiple developers to work on the same project. You can invite collaborators to your repository and manage their access permissions from the repository settings.

To collaborate on a project:

1. Share the repository's URL with your collaborators.

2. Collaborators can clone the repository, make changes, commit them, and push them to GitHub.
3. Use pull requests to review and discuss changes before merging them into the main codebase.

Congratulations! You've learned the basics of using Git and GitHub to manage your projects efficiently. As you continue to use these tools, you'll discover more advanced features and workflows to streamline your development process.