Git

- Purva, IPLUG

Introduction

You must have seen, and probably done something like this yourself:

- Report.docx
- Report-Final.docx
- Report-FinalChanged.docx
- Report-FinalFinal.docx
- Report-Final3.docx
- Report-TheUltimateFinal.docx

Whenever we work on something, it's usually not a job finished in a single sitting. There are changes, additions and deletions. Don't you sometimes wish you could go back to how your work was at a specific point when the Undo button isn't available anymore?

That's where **Version Control Systems** (VCS) come in. A VCS saves your work at different moments, so that you can go back to some point in the past. If you have ever used the History feature in Google Docs, it is similar to that.

Git, Mercurial, SVN are some examples of Version Control Systems which are primarily used for code, but can be used for just about any work. We will be talking about Git, the most popular VCS.

What is Git?

- Git is a VCS.
- It snapshots your work at any given moment that you want it, along with a helpful **commit** message.

- A folder can be initialized to work with Git. Such a folder is called a **Repository**. It is called *Repo* in short.
- You can have Git store your repos at some remote place.

Git vs GitHub

GitHub is like a Google Drive for Git Repositories. GitHub allows you to store any repos that are initialized with Git. It acts as a remote place to store your work.

The primary benefit is that it makes **collaboration** easier and much more feasible. Imagine many people working on a codebase, of course, they can't work on a single computer on a single local folder. Here's where the power of Git as a VCS and GitHub as a storage medium comes in.

GitHub is not the only service which hosts Git Repositories. Some names include

- GitHub (from GitHub Inc, purchased by Microsoft)
- GitLab (from GitLab Inc)
- Bitbucket (from Atlassian)
- Gitee (from Open Source China)

Git Basics

The Idea

- We initialize a folder as a *Git Repository*. This folder is also our **working directory**.
    - The working directory is our live working folder, just a simple folder in the normal sense.
    - The repository is where git stores snapshots of the work.
    - In practice, we call the folder both a repo and a working directory.

- Whenever we do some little work that we feel should be saved, we **add the file(s) to the staging area** and **commit** with a helpful message.
- Git stores a snapshot of the folder with the commit message and a unique id (an SHA256 hash).

Image Source: [Git Basics, Geo Python, University of Helsinki]

- We can also have our work sync with some remote repository or backup. That could be present on any services like GitHub, GitLab, etc.
  - We can **push** any changes on our local repo to the remote repo.
  - We can **pull** any changes from the remote repo to our local repo.
- We can **revert** any changes at a later time.

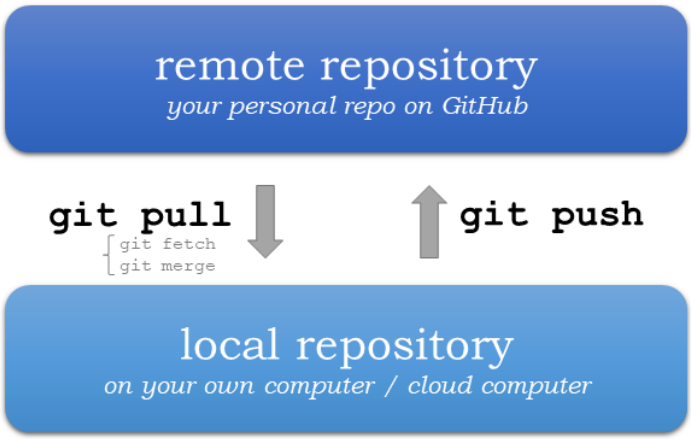


Image Source: [Git Basics, Geo Python, University of Helsinki]

- The snapshots and commits are stored in a Graph Tree like structure. The main tree is simply called **main** or **master** branch.
- We can also work on different things using the feature of **branches**. Essentially, we work on something alongside the main/master branch and we can merge changes to the main/master branch later on.
- How does Git know what branch is it on? There is always a pointer, called the **HEAD**, pointing to the latest commit on the branch we are working on.

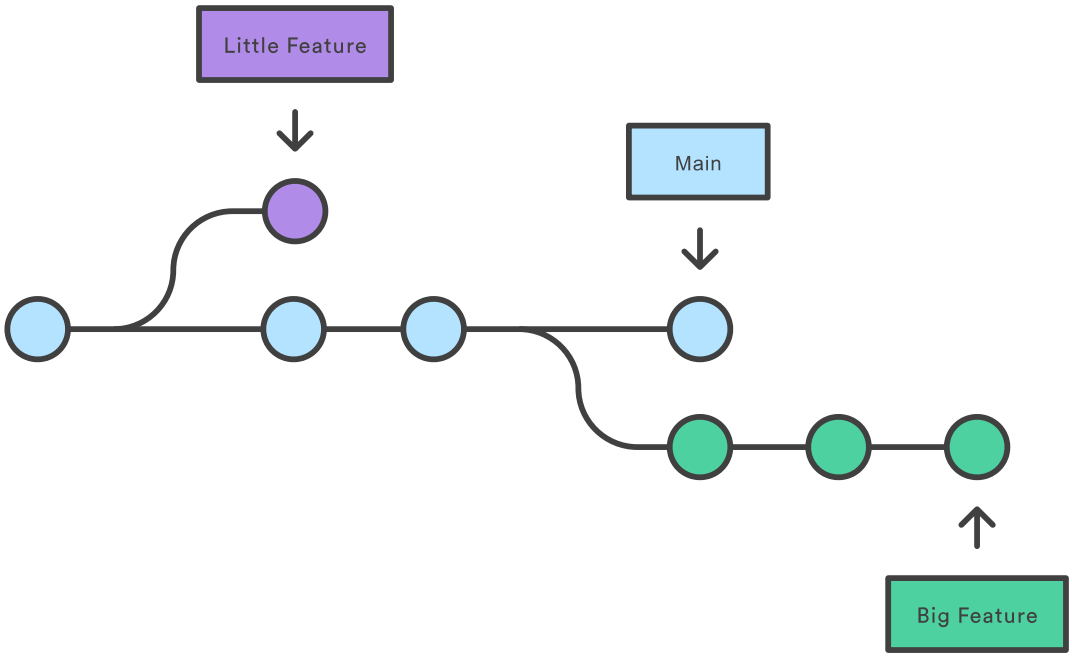


Image Source: [Using Git Branches, Atlassian]

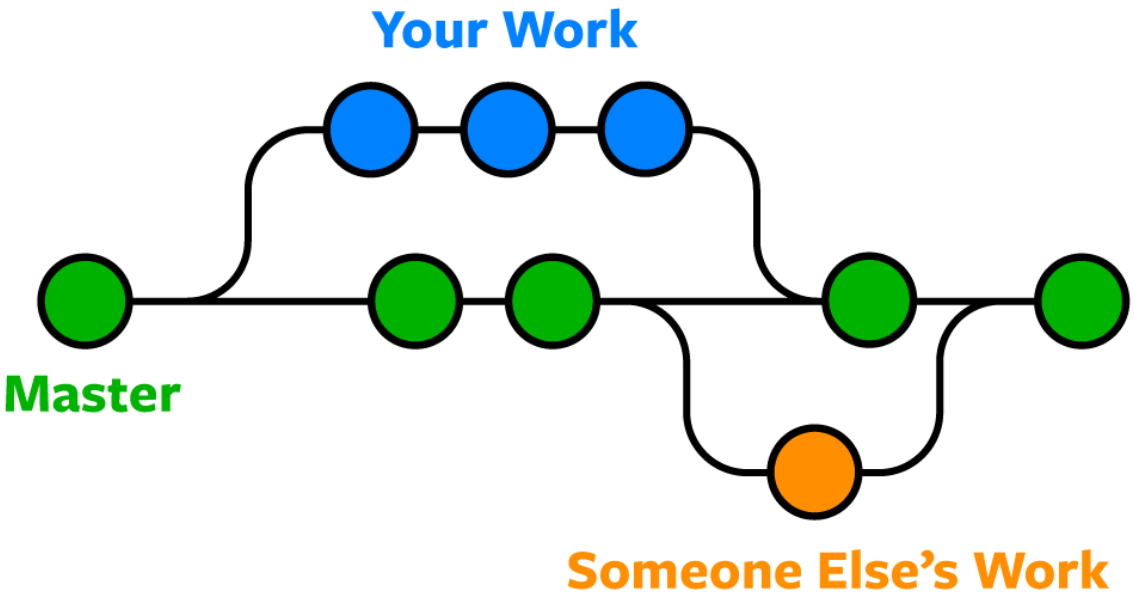


Image Source: [Git Branches, Noble Desktop]

Setting up Git

Downloading Git

Git usually comes installed by default on Linux and Mac systems.

For Windows,

- [Optional but nice] Download Windows Terminal.
  - If the link doesn't work, go to Microsoft Store and search for *Windows Terminal*.
- Download Notepad++ and install it. It is a powerful and fast text editor with many more features than the standard Notepad.
- Download Git.
  - When it asks about the **default editor**, select Notepad++. This will help a lot.
  - For the rest, follow the default options and continue the installation.

Checking that git is installed

Open a terminal and run:

```
git --version
```

It should show something like `git version 2.35.0.windows.1` on Windows, and `git version 2.25.1` on Linux systems. Of course, the exact version number may differ.

The point is, if it works and gives an output and not an error, git is working.

Initial Configuration

Git associates a username and email address to each commit. We need to set it up before working with Git.

```
git config --global user.name "MyName"
```

```
git config --global user.email "mail@example.com"
```

Using Git

Standard Commands

- To **initialize a folder**:

  ```
  git init
  ```

  This can be done in any folder. Either empty or something which already has some work.

- We can **view the difference** from the latest commit and our current work with

  ```
  git diff
  ```

  This will show additions with a green highlighting, deletions with a red highlighting. Press `q` to exit the diff screen.

- We can now do our work. Every time we have done some little thing and we wish to save it with Git, we **add the files to the staging area**
  - ```
    git add filenames
    ```
    ```

    - To add the entire current folder to the staging area, we can simply do

      ```
      git add .
      ```

      The `.` is a shorthand for the current directory. [Similarly, `..` is a shorthand for parent directory.]

- We can **view the current status** of our files with

  ```
  git status
  ```

  This shows the files which are currently untracked or staged.

- To **commit the changes**:

  ```
  git commit -m "Message describing changes or why the changes happened"
  ```

- The work has been saved by Git. We can **view the log of all our commits** as:

  ```
  git log
  ```

  Press `q` to exit the log screen.

We can repeat the steps as required. Here's the previous image shown again.

Image Source: Git Basics, Geo Python, University of Helsinki

- To **discard any changes in the working directory** and restore a file as to how it was in the last commit, we do

```
git restore filenames
```

- To **remove a file from the staging area** (for example, if you added some file by mistake)

```
git restore --staged filenames
```

Using a Remote

As mentioned previously, we can have our Git repo hosted somewhere online as well. GitHub is the most popular service for this, and is used by many developers and hobbyists alike.

Remote simply means our repo is hosted somewhere else too. Same sense as working *remotely.*

Git assigns a simple **nickname** to every remote. The default nickname used by most is `origin`.

> ***NOTE:*** *You need to authorize git to use your GitHub account with SSH. Refer to all the steps in Connecting to SSH Docs on GitHub and add an SSH key to your account before proceeding.*

Creating a new remote repo

- Go to GitHub, create a new repository. For example, suppose my username is `iplug-iiserp` and I create a repository named `hello-world`.
- GitHub will show the URL to use automatically after the creation. Here, in our example, it would be `https://github.com/iplug-iiserp/hello-world.git`.
- To add this as our remote for our local repo, we can do:

```
git remote add origin https://github.com/iplug-iiserp/hello-git.git
```

As you can notice, the command adds the URL as a remote with the nickname `origin`. You can use a different nickname if you like, though.

- To **push our local repo** to the remote:

```
git push origin master
```

If the default branch name is `main`, replace `master` with `main`.

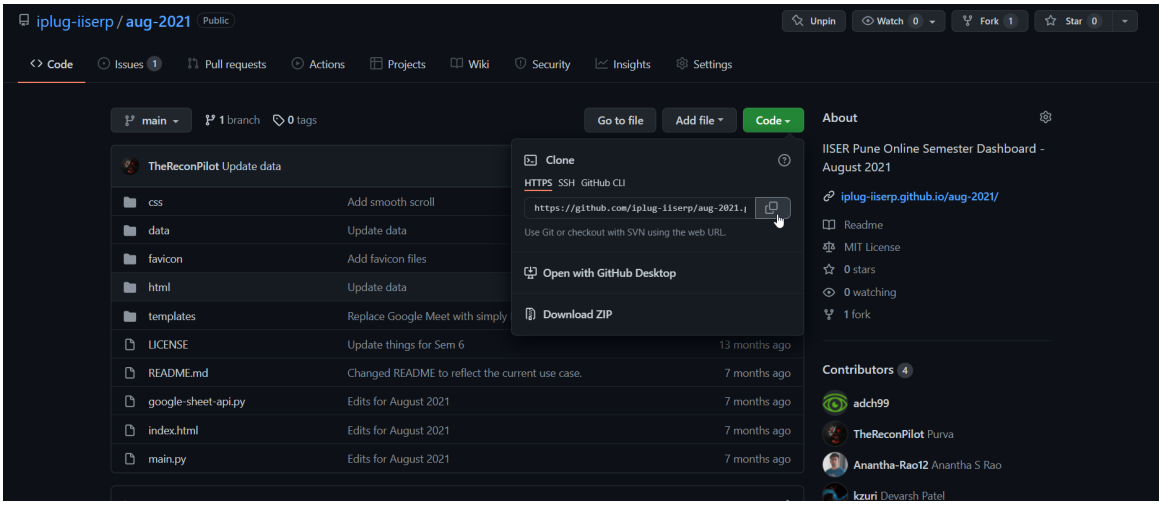
- Similarly, we can **pull data from the remote to our local repo**:

```
git pull origin master
```

As always, make sure to check the default branch name. Replace `master` with `main` if that is your default.

Using an existing remote repo

There are loads of repositories publicly available on GitHub. Here's one for example: iplug-iiserp/aug-2021, an online dashboard for the August 2021 Semester.



Click on the `Code` button and you should see a URL. Copy that.

We will **clone** this repository to our local machine. Here in this case, it would be:

```
git clone https://github.com/iplug-iiserp/aug-2021.git
```

> ***NOTE:*** *If you simply wish to clone a repository and not push any changes to it, you don't need the SSH authentication step.*

We often clone repositories to just use or test out the software/work or see it.

The Power of GitHub - Collaboration

Here's how a typical group work on a codebase looks:

- There is a remote repository hosted on GitHub.
- Everyone has a local copy.
- Everyone works on their own stuff on the respective branches.
    - For example, someone could be working on a `new-feature` or `development` branch, or whatever they have named it.
    - They commit their work as and when needed.
- Everyone pushes their work to the remote repository.
- When someone is done with their work, they open a **Pull Request** (PR) on GitHub. (This is specific to GitHub). The branch is merged after a code review by the head maintainers of the repository.
- If there are any conflicts (like multiple people made changes to the same code lines), then the *merge conflict* needs to be manually sorted out. Someone has to go to each conflict and select which change to keep and which to reject. After all conflicts are resolved, the branches can be merged.
- Git creates a special commit signifying merging of branches.

GitHub and Git allow many people to work on the same thing simultaneously, and each small change is saved to a commit message. This greatly reduces the pain of debugging and reverting any changes if needed.

Resources

Starting with Git

- **git - the simple guide** by Roger Dudler
  https://rogerdudler.github.io/git-guide/
- **Git How To** - a guided tour that walks you through the fundamentals of git.
  https://githowto.com/
- **Meet Git** - A simple introduction to Git in the Geo Python course offered by the University of Helsinki
  https://geo-python.github.io/site/lessons/L2/git-basics.html
- **Introduction to GitHub** - An interactive class by GitHub
  https://lab.github.com/githubtraining/introduction-to-github

Going Deeper

- **Git Branches** - A simple page by Atlassian explaining the concept of Branching
  https://www.atlassian.com/git/tutorials/using-branches
- **The Pro Git Book** - The book by Git, serves as both an introduction and a deep dive
  https://git-scm.com/book/en/v2
- **Visualize Git with D3** - An interactive site which helps you visualize how different git commands may look and how they affect the tree
  https://onlywei.github.io/explain-git-with-d3/
- **Git Explorer** - Find the right commands you need without digging through the web
  https://gitexplorer.com/

Also recommend is looking up GitHub specific things like **Forking a Repository**, **Pull Requests**, **Commit Signatures**, **GitHub PR Review**, **GitHub Pages** and anything you may come across and find interesting.

When Things go Wrong

- **Oh Shit, Git!?!** - Some situations you may get yourself into, and how to resolve them
  https://ohshitgit.com/

- **Git Flight Rules** - Some solutions on what to do when something happens
  https://github.com/k88hudson/git-flight-rules