5 Git Basics

CS 181
Object-oriented programming

Alex Petrovici

Content

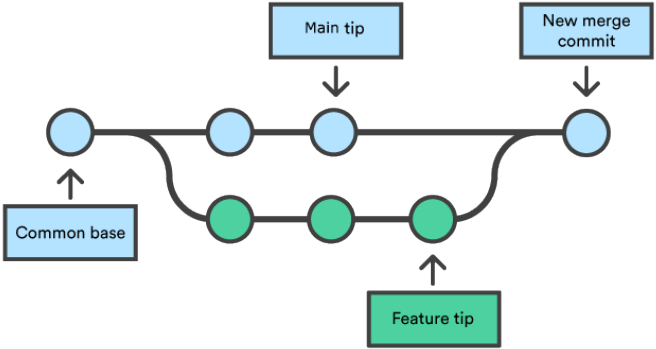
Introduction / Why do we need version control?

- **Track changes:** Version control lets you track and review changes to your code and documents over time.
- **Collaboration:** Multiple people can work on the same project simultaneously without overwriting each other's work.
- **History and rollback:** Easily revert to previous versions or undo mistakes by accessing the project history.
- **Backup:** Your project is safely stored and can be recovered, even if files are accidentally deleted or lost.
- **Experimentation:** Create branches to try out new features or ideas without affecting the main codebase.
- **Accountability:** See who made what change and when, which helps with understanding and reviewing work.

Introduction / Popular Version Control Systems

- **1990 CVS (Concurrent Versions System)** – One of the earliest version control systems, now largely replaced by newer tools.
- **1998 Perforce (Helix Core)** – Centralized system designed for large-scale projects and binary files.
- **2000 Subversion (SVN)** – Centralized version control system, known for its simplicity and atomic commits.
- **2005 Git** – Distributed version control system, widely used for its speed, flexibility, and branching capabilities.
- **2005 Mercurial** – Distributed version control system, similar to Git but with a focus on ease of use.
- **2006 Fossil** – Distributed version control with built-in wiki, bug tracking, and web interface.

Introduction / Popular Version Control Systems



Introduction / Popular Version Control Systems

Why focus on Git?

- **De Facto Industry Standard:** Git is the most widely used version control system in software development today.
- **GitHub, GitLab, Bitbucket:** Major cloud platforms are built around Git.
- **Open source:** Free and supported by a large community with extensive documentation and resources.
- **Powerful branching:** Git's branching model makes it easy to work on features independently and merge them later.
- **Distributed:** Every developer has a complete copy of the repository, enabling offline work and redundancy.

First Git Setup / Install Git

Ubuntu:

```
sudo apt update
sudo apt install git -y
```

macOS:

```
brew install git
```

Other platforms: Follow installation instructions from https://git-scm.com/downloads. Then check that Git has installed correctly:

```
git --version
```

Expectation:

```
git version 2.43.0
```

First Git Setup / Configure Git

Now that we have Git installed, we need to personalize it with our credentials.

Change name and email

```
git config --global user.name "John Doe"
git config --global user.email "johndoe@example.com"
```

Source: https://git-scm.com/book/ms/v2/Getting-Started-First-Time-Git-Setup

Now your commits will be associated with your name and email.

First Git Setup: Log in Github via SSH

Generate a SSH key

Since `Support for password authentication was removed on August 13, 2021` we need to connect git to our github account via SSH access. Steps:
In the terminal:

```
ssh-keygen -t rsa -C your_email@example.com
```

- `ssh-keygen`: This is the program used to create the SSH keys.
- `-t rsa`: This option specifies the type of key to generate.

First Git Setup: Log in Github via SSH

Hit Enter to save the key in the default location

```
~/.ssh/id_rsa
```

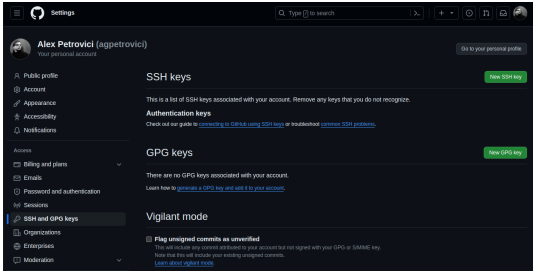
Hit Enter twice for no/empty passphrase.
Your SSH keys will be saved at the default location. Run the following command to view the generated public SSH key (will be pasted in github).

```
cat ~/.ssh/id_rsa.pub
```

First Git Setup: Log in Github via SSH

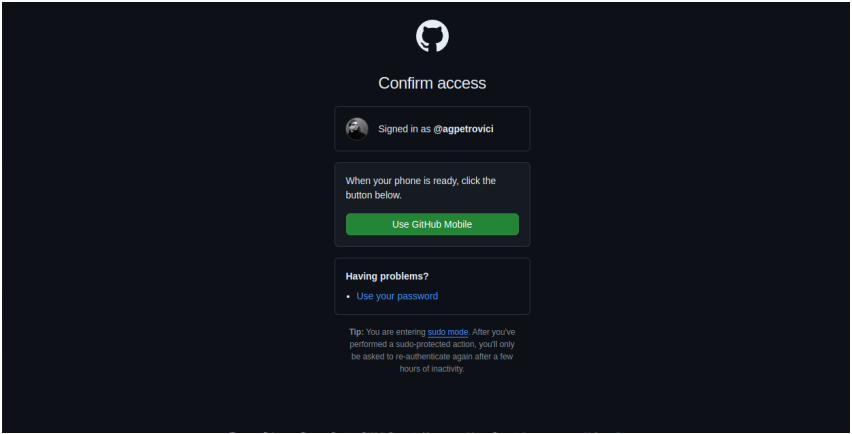
Add the SSH key to GitHub

- 1. Go to SSH Keys: `https://github.com/settings/keys`
- 2. Click on **New SSH key**



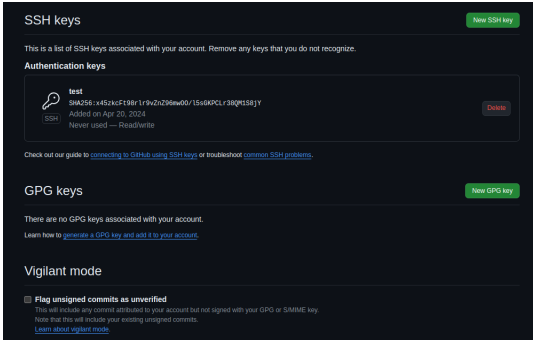
First Git Setup: Log in Github via SSH

Confirm access:



First Git Setup: Log in Github via SSH

Result:



First Git Setup: Log in Github via SSH

Add github to the list of known_hosts

```
ssh-keyscan github.com >> ~/.ssh/known_hosts
```

- `github.com`: This specifies the host from which ssh-keyscan should collect the public SSH keys. You can also list multiple hosts or use IP addresses instead of domain names.
- »: is used to append the output to a file. If the file does not exist, it will be created. If you use a single '>' instead, it would overwrite the file each time, rather than appending to it.

It will copy the content from "GitHub's SSH key fingerprints" https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/githubs-ssh-key-fingerprints and append them to `/.ssh/known_hosts`

Git Basics / Make a local repository

You can either make a new repository from Github and then clone it, or first make it and then point to the remote. We'll do the second option since is more common to start a new project from scratch and later push it to Github.
For example, make a new directory and initialize a repository in it:

Make a repository

```
mkdir MA_UG2025FALL_CS181MAD  # mkdir = make directory with project name
cd MA_UG2025FALL_CS181MAD     # cd = change directory to the new project
git init
```

`git init` initializes a repository in the current directory.

Expectation:

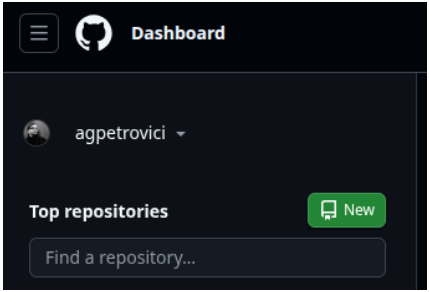
```
Initialized empty Git repository in /path/to/project/MA_UG2025FALL_CS181MAD/.git/
```

Git Basics / Make a remote repository

The local git repository that we just made can point to a remote repository in Github (or different branches to other platforms like GitLab in the same repository)

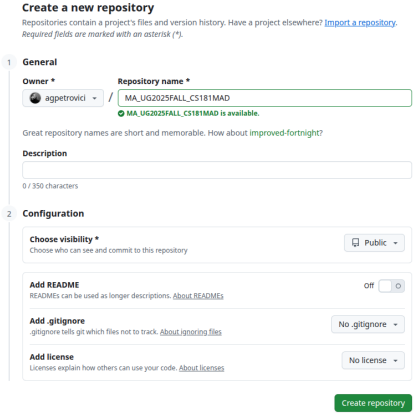
Make a remote repository in Github

Go to `https://github.com/dashboard` and click on **New repository**



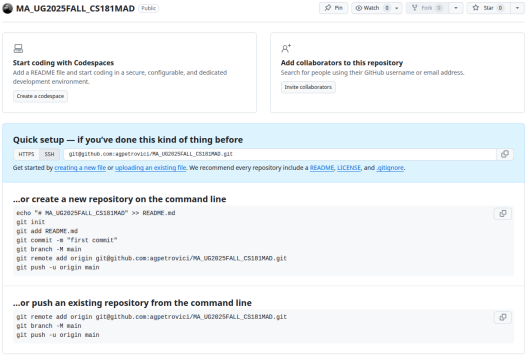
Git Basics / Make a remote repository

Then provide a name and make it either public or private.



Git Basics / Make a remote repository

You should see that the repository is created and now you can clone it as is or point an existing local repository to it (we'll do the second option).



Git Basics / Make a remote repository

In your local repository, point to the remote repository by adding the remote URL: Since we start with a blank repository, we can follow the instructions to create a new repository on the command line

Assumption: we are in the root path of the local repository.

```
echo "# MA_UG2025FALL_CS181MAD" >> README.md  # adds the text content to the README.md file
git add .  # adds all the files to the staging area
git commit -m "initial commit"  # makes a new commit with the message "initial commit"
git remote add origin git@github.com:agpetrovici/MA_UG2025FALL_CS181MAD.git  # points to the
↪  remote repository
git push -u origin main
```

Git Basics / Make a remote repository

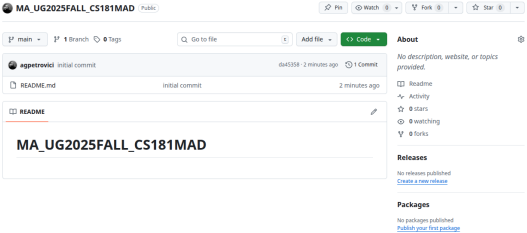
Now you should have something like this in the terminal:

```
❯ echo "# MA_UG2025FALL_CS181MAD" >> README.md
❯ ls
README.md
❯ git add *
❯ git commit -m "initial commit"
[main (root-commit) da45358] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
❯ git remote add origin git@github.com:agpetrovici/MA_UG2025FALL_CS181MAD.git
❯ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 241 bytes | 241.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:agpetrovici/MA_UG2025FALL_CS181MAD.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
❯ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Git Basics / Make a remote repository

And your github should look like this:

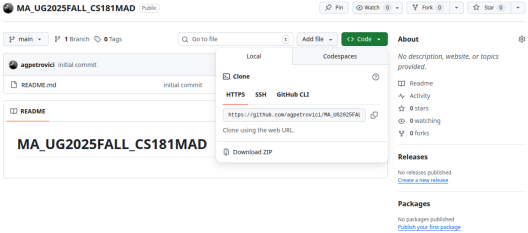


Git Basics / Clone a repository

You can clone a repository from Github to your local machine by using the following command:

```
git clone <repository-url>
```

The <repository-url> is the URL of the remote repository.



Use:

- **SSH** when you have access to the repository.
- **HTTPS** when you don't have access to the repository (if you can see a repository and you don't have access to it, it's because it's public).

Git Basics / Clone a repository

Run the following to clone the repository:

```
git clone https://github.com/agpetrovici/MA_UG2025FALL_CS181MAD.git
```

Expectation:

```
❯ git clone https://github.com/agpetrovici/MA_UG2025FALL_CS181MAD.git
Cloning into 'MA_UG2025FALL_CS181MAD'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
❯ cd MA_UG2025FALL_CS181MAD
❯ ls
README.md
```

Git Basics / Clone a repository

From now on, to be up to date with the content in the repository, you have to pull the changes:

```
git pull
```

Make a commit /

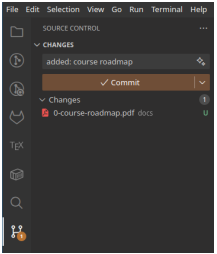
First of all, you have to be in a repository that you cloned with SSH so you permission to push changes.
Open the repository in vscode:

```
code .
```

or from the program itself: **File > Open Folder > [Select the repository folder]**

Make a commit /

Go to **Source Control** and select the files you want to add and add a message to the commit.



This does behind the scenes:

```
git add <file1>
git commit -m "commit message"
```

Make a commit /

Then you can either **Sync Changes** to push the changes to the remote, or run in the terminal:

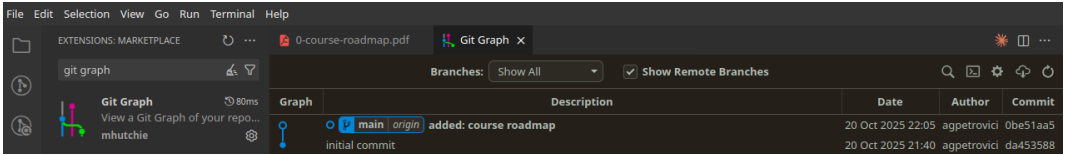
```
git push
```

Make a commit /

Recommendation: install the extension **Git Graph** in vscode.

```
Name: Git Graph
Id: mhutchie.git-graph
Description: View a Git Graph of your repository, and perform Git actions from the graph.
Version: 1.30.0
Publisher: mhutchie
VS Marketplace Link: https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
```

so you can visualize the changes in a visual manner:



Make a commit /

If you have a specific issue to solve with git, you can use the following website:
https://git.gaozih.top/