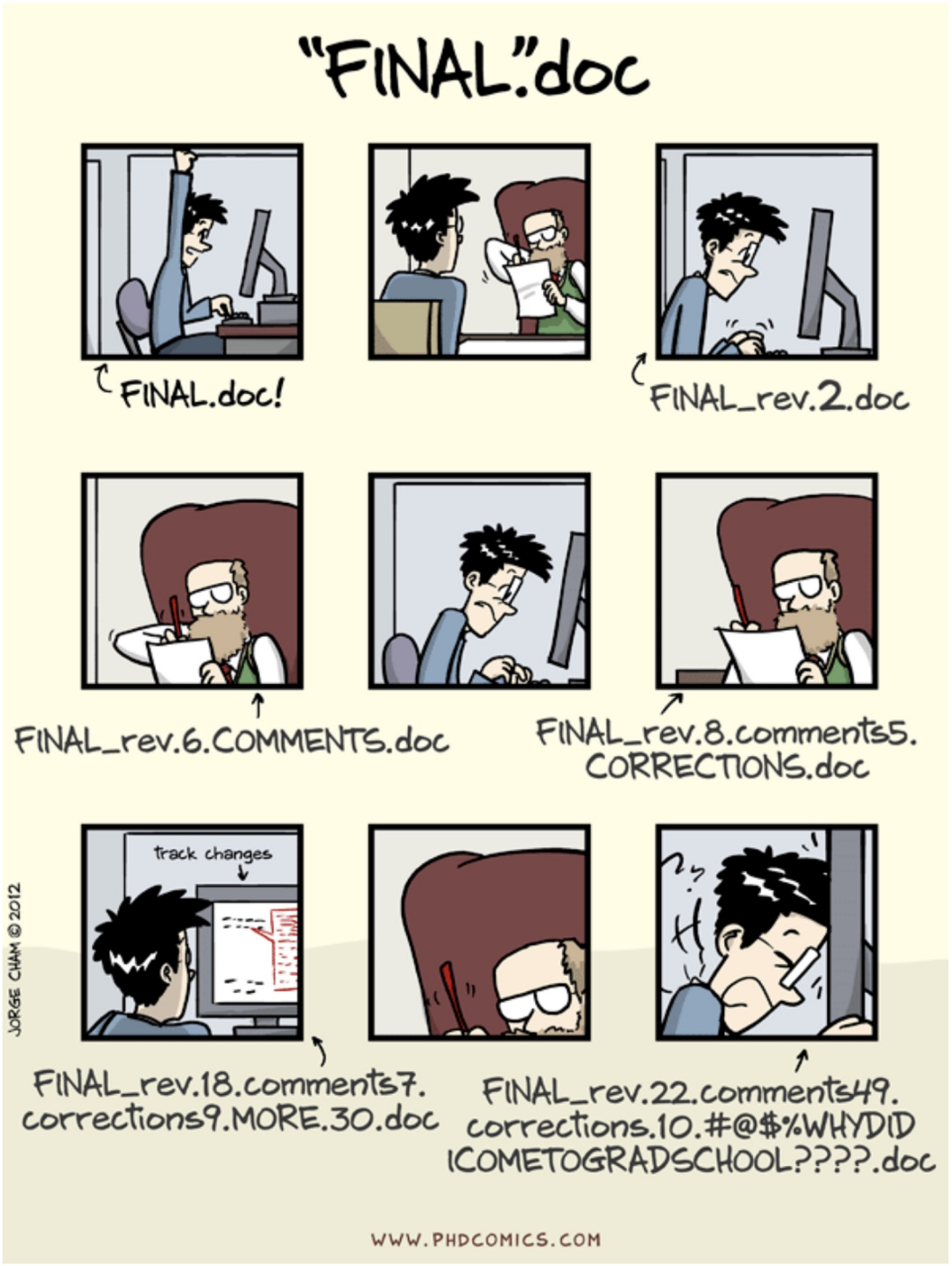
DATA 3401
Python for Data Science

Version Control System

Dr. Max (Masoud) Rostami

Department of Science / College of Science
Data Science Program / College of Science
The University of Texas Arlington, Texas

Version Control System



Version control with git

Version Control Systems (VCS)

are essential for any software development process, be it a small or a large-scale project.

Here are some situations where you'd want to use a VCS:

1. **Collaborative Work**: When multiple developers are working on the same codebase, VCS ensures that there's no code conflict between developers.
2. **Tracking Changes**: With VCS, every change to the code is tracked, including information about who made the change, when they made it, and why they made it. This can be very useful for understanding the history of a project, finding when and where bugs were introduced, and debugging.
3. **Backup and Restore**: VCS also provides a way to back up your projects. If something goes wrong with your current codebase, you can restore it to a previously working state using the VCS.
4. **Release Management**: You can mark certain points in your project history as 'releases' or 'versions'. This allows you to track particular versions of your project and compare them over time.
5. **Remote Working**: Distributed Version Control Systems like Git allows developers to work remotely and asynchronously. Each developer gets a complete copy of the entire codebase on their local machine, enabling them to work independently.

How often does this happen?

- You and one other person need to work on the same dataset, and one of you edits the data.
- You and a group use multiple copies of data/code, and each of you reformats or reorganizes the data
- Someone sends you back a paper but has not used **track changes**

A version control system (VCS), also known as a source control system

- Is a system that records changes to a file or set of files over time so that specific versions can be recalled later.
- It allows you to track modifications, see who made changes, and even revert to an earlier state if necessary.

- Nothing that is committed to version control is ever lost
- Old versions of files are saved
- It’s always possible to go back in time to see exactly who wrote what on a particular day,
    - or to see what version of a program was used to generate a particular set of results.

The version control system

- Have a record of who made what changes when, so we know who to ask if we have questions later on
- We can revert to a previous version, much like the “undo” feature in an editor.
- When several people collaborate in the same project, it’s possible to accidentally overlook or overwrite someone’s changes.
    - The version control system automatically notifies users whenever there’s a conflict between one person’s work and another’s.

Version control with Git

Git is a version control system that lets you

- track who made changes to what
- when,
- And has options for easily updating a shared or public version of your code on github.com.

Version control systems start with a **base version of the document** and then record changes you make each step of the way.

Version control with Git

- **Git** is a version control system that lets you manage and keep track of your source code history.
- **GitHub** is a cloud-based hosting service that lets you manage Git repositories.
- If you have open-source projects that use **Git**, then **GitHub** is designed to help you better manage them.

Linux command line (create and add files)

$ pwd	Print the current working directory
$ cd	Changing directory
$ mkdir	Make directory
$ ls	List
$ touch	Create a new file

Changing to the Previous Directory

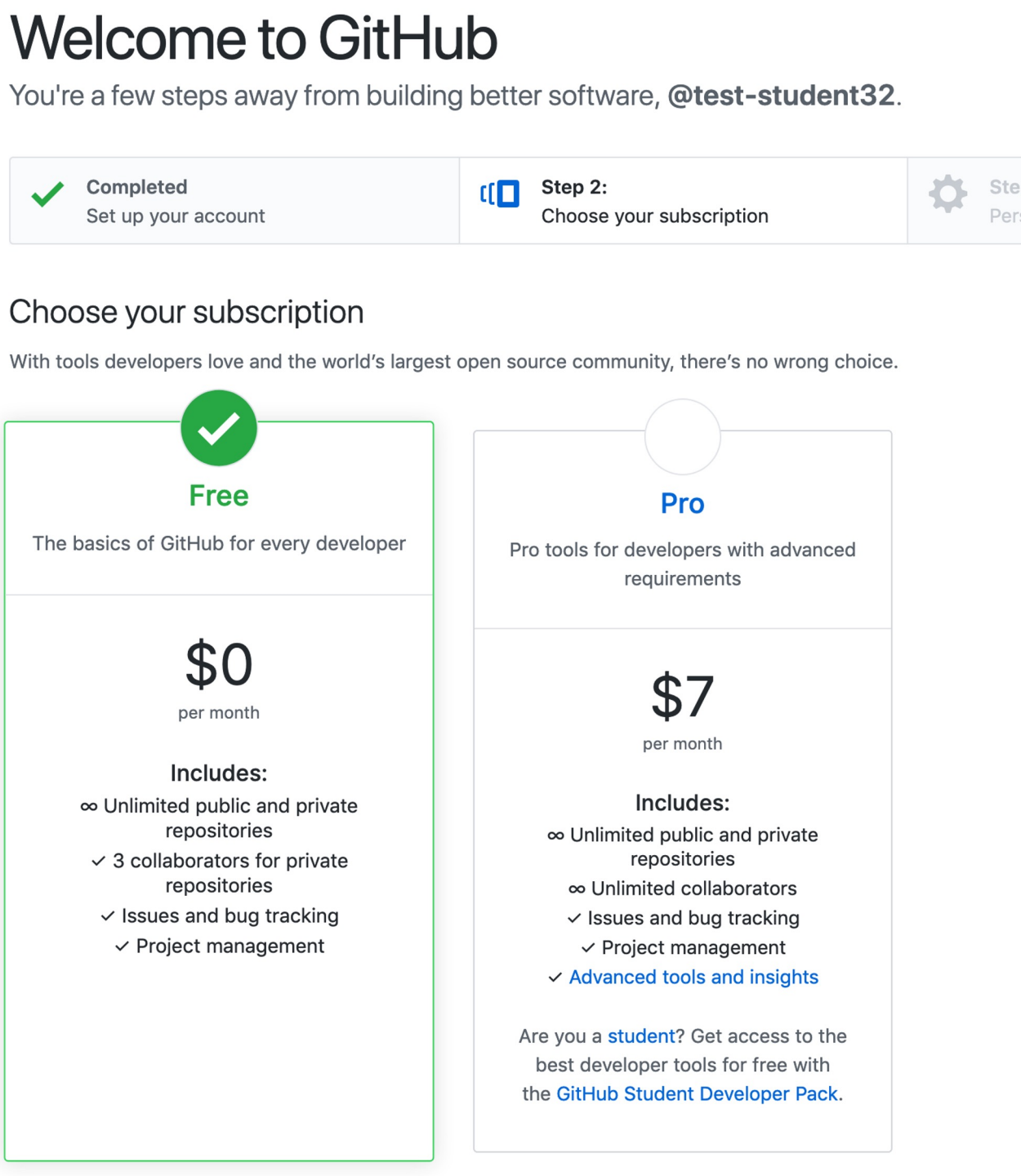
cd -	Returns the shell to the previous working directory.
cd ~	Back to the default working directory
cd /	Change directory to the root directory
cd ..	Move to the parent directory of the current directory

Now, Let's start

Step 1:Let’s Create a GitHub account

Create a Github Account

- Create a username, give an email address and passcode
- Will have to verify it by proving you are human
- Select the free account
- Possibly unclick the ‘send me offers and updates on github’
- Answer or skip survey
- Verify email



Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

test-student32 / **hello-world** Private
generated from github/welcome-to-github

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Security | Insights | Settings

No description, website, or topics provided.

Edit

Manage topics

1 commit | **1** branch | **0** releases

Branch: **master** | New pull request | Create new file | Upload files | Find File | Clone or download

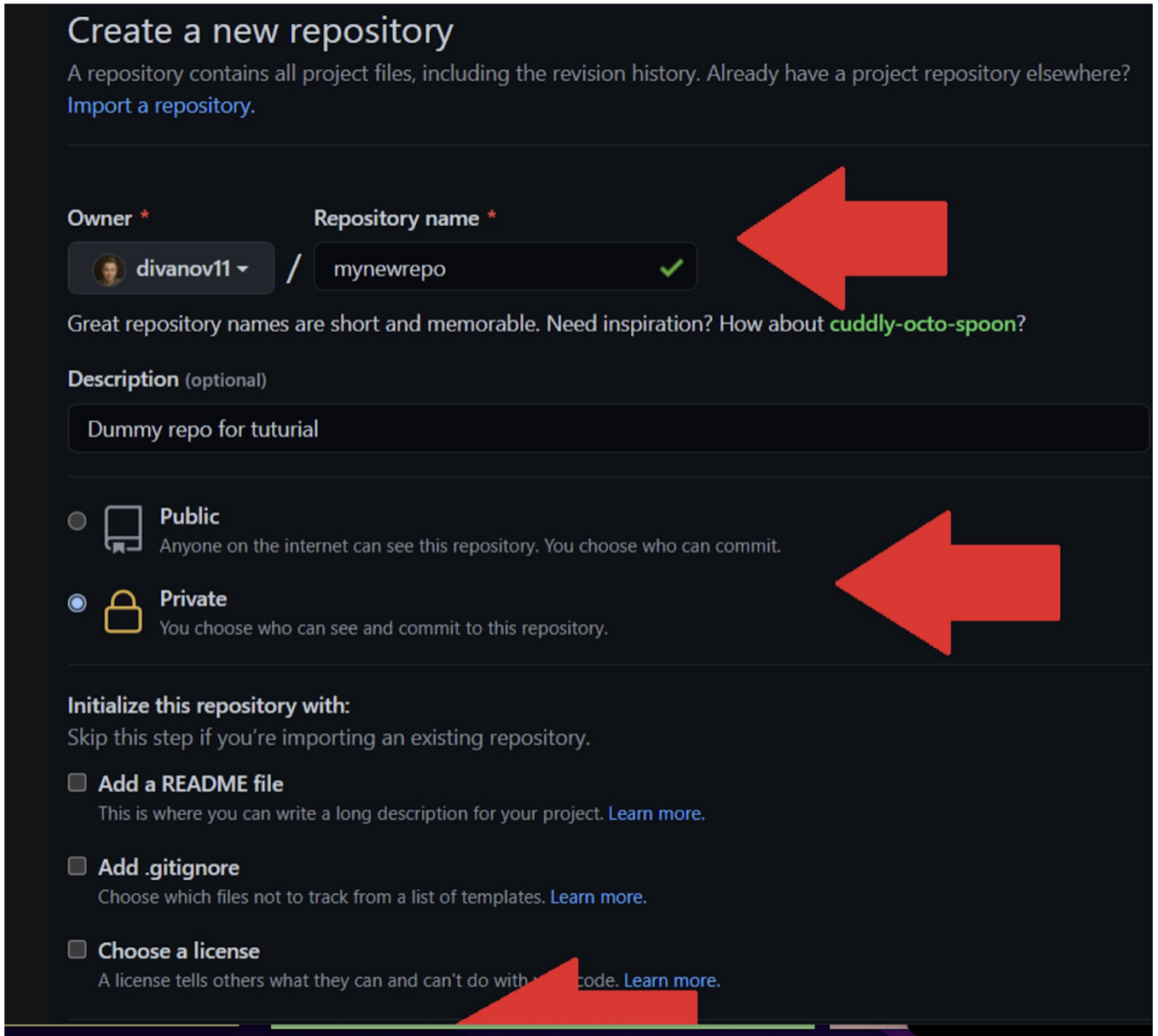
test-student32 Initial commit — Latest commit 46ebee5 10 minutes ago

images	Initial commit	10 minutes ago
README.md	Initial commit	10 minutes ago
index.html	Initial commit	10 minutes ago
styles.css	Initial commit	10 minutes ago

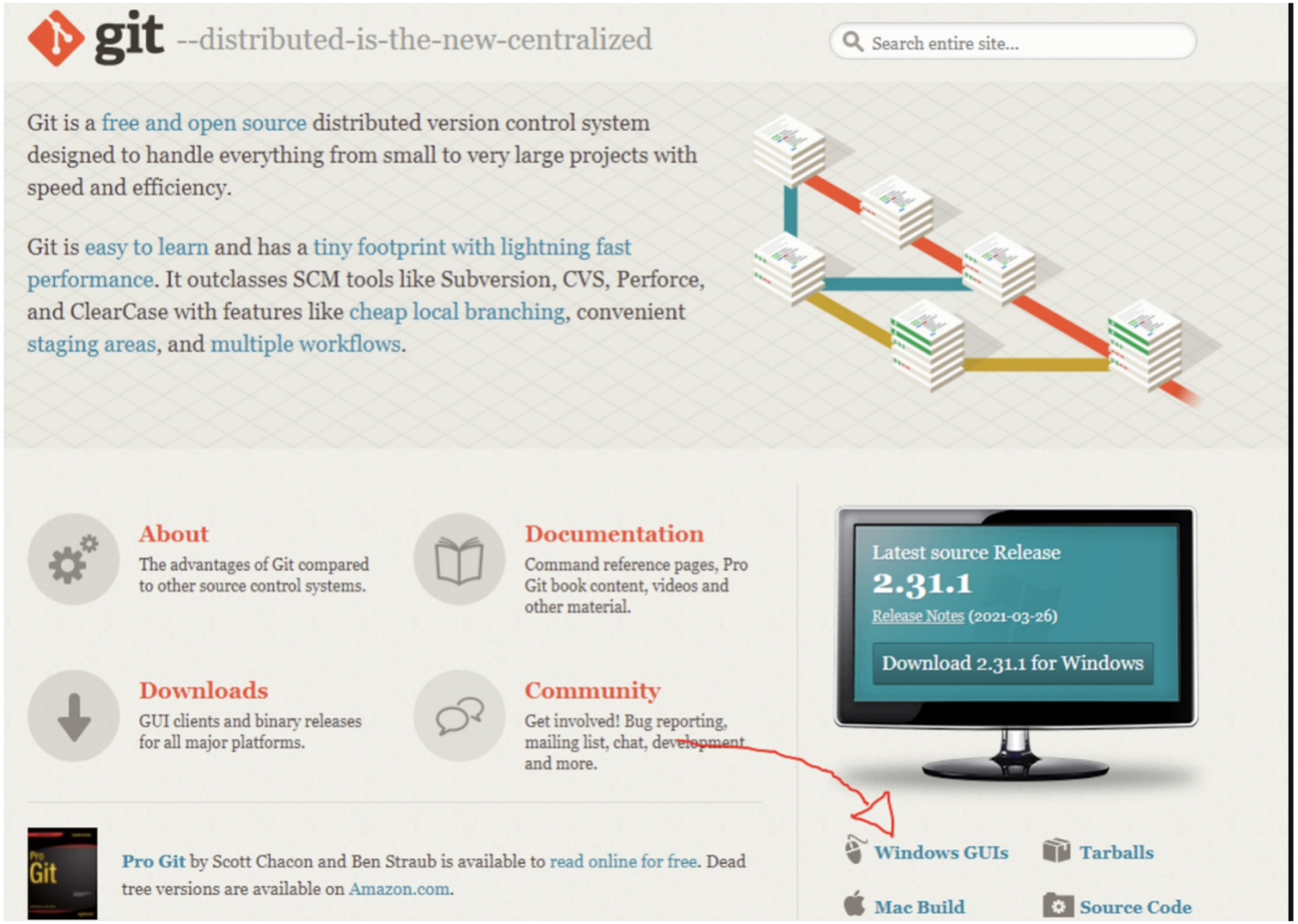
README.md

Welcome to GitHub

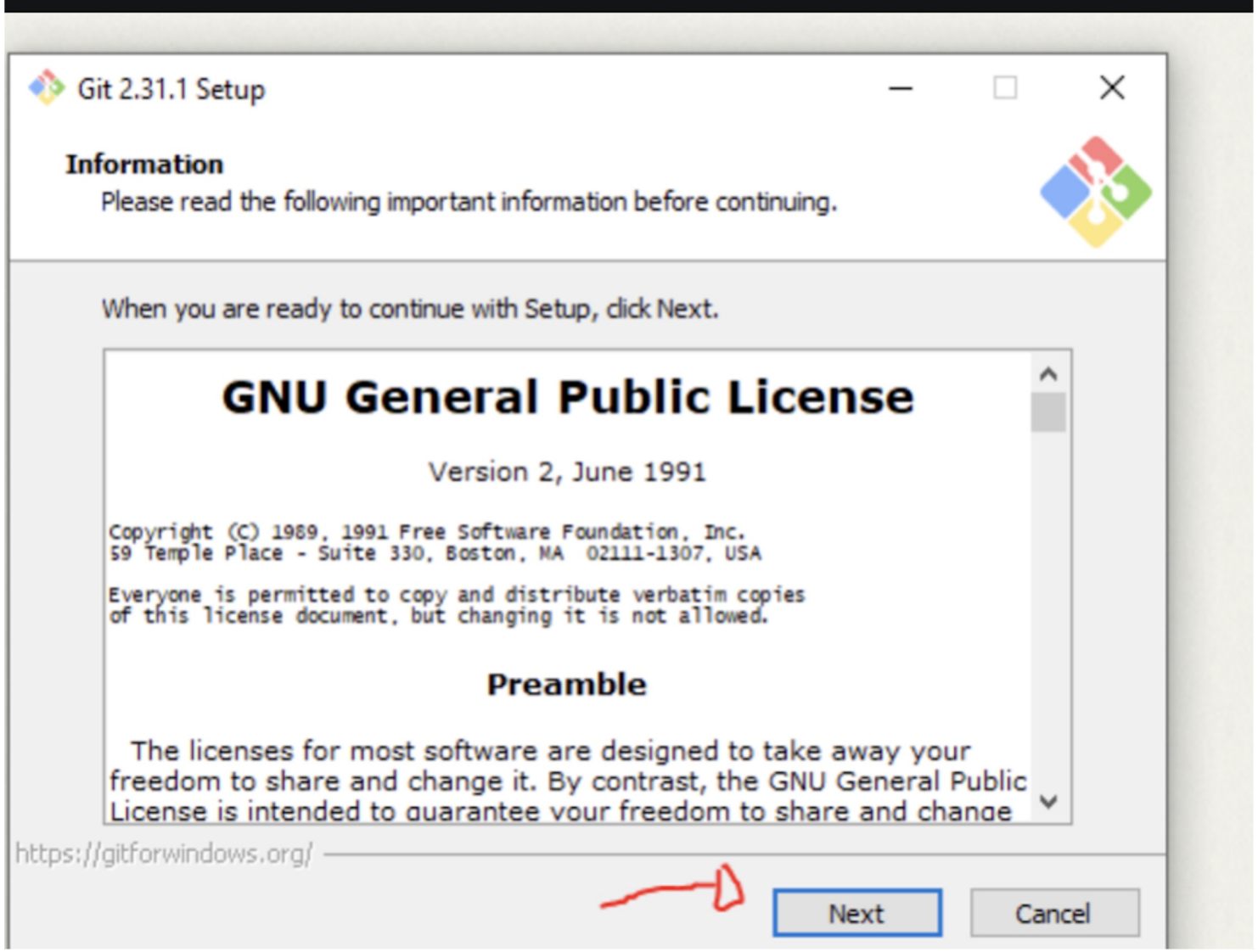
Step 2: Create a new repository



Step 3: Install git (Just for window users, Mac users skip this step)



Download Installer - Got to https://git-scm.com/ and select the downloader for your machine, I will be using windows



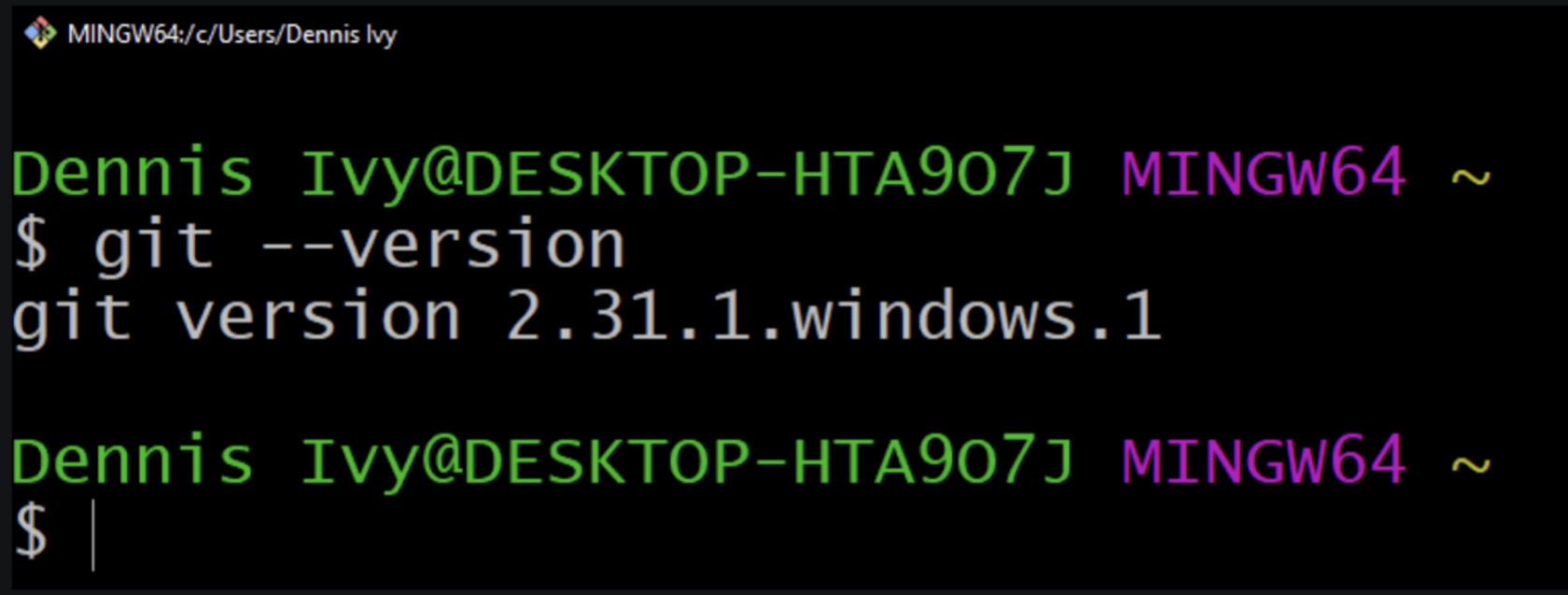
Install git - Once the installer is downloaded go ahead and follow the steps to get things set up, I tend to just leave the default settings so if you don't have a preference just let it guide you.

Check the git version in git bash (Windows users) and terminal (Mac users)

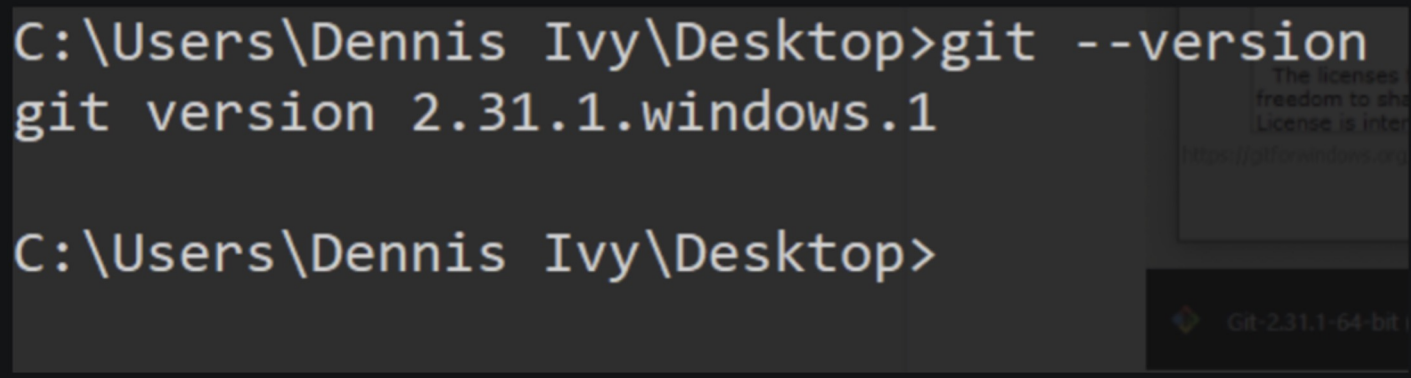
Once things are installed do a quick search on your computer for git bash and open it.

To check the version of git, you have installed go ahead and type

git --version



You can also do this from your command prompt, in fact I will be using the command prompt from now on so go ahead and close git bash and open up a new terminal.If you already had your terminal open you may need to close and reopen to get the updates.



Let's look at a few of the commands we will be using in the next steps:

Step 4: Setting up Git (You need to do this part just one time)

- If you are using git regularly have already done this.
- If not need to set:
  - User Name – *same as the one you set for your github account!*
  - Email address – *same as you will use with github account!*
  - And that we want to use these globally (for every project)

$ git config --global user.name "Your Name"

$ git config --global user.email "youremail@example.com"

Bash

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

Create Local Repository

Step 1: - Initialize a new git repo

git init

When you run git init a new folder will be added to your project files, these files visibility are set to “hidden” by default so you will not see them but rest assured they are there. If you want to see these files open up your folder and select “view”

Check your staging area

This step is not necessary before we add our files but for the sake of seeing the difference and showing exactly what happens lets run “git status”

```
C:\Users\Dennis Ivy\Desktop\newproject>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        manage.py
        newproject/
```

Step 2: - Add files

To add a particular file or folder to your staging area you can run "git add .". In our case we want to add all the files to the staging area.

```
> git add .
```

After running git status I typically like to run git status again to ensure all changes we made.

```
C:\Users\Dennis Ivy\Desktop\newproject>git add .

C:\Users\Dennis Ivy\Desktop\newproject>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   manage.py
        new file:   newproject/__init__.py
        new file:   newproject/asgi.py
        new file:   newproject/settings.py
        new file:   newproject/urls.py
        new file:   newproject/wsgi.py
```

What if we make changes after adding running git “add .”?

No worries, just run git add again and changes will be updated. You can always run

“**git status**” beforehand to see exactly which files have been modified.

Step 3: - Commit Changes

> git commit -m “Your custom note”

Now that we added files to our staging area we need to commit them to our local repository.

Now that we added files to our staging area we need to commit them to our local repository.

```
git commit -m “first commit”.
```

```
C:\Users\Dennis Ivy\Desktop\newproject>git commit -m "First commit"
[master (root-commit) 4a08248] First commit
 6 files changed, 195 insertions(+)
 create mode 100644 manage.py
 create mode 100644 newproject/__init__.py
 create mode 100644 newproject/asgi.py
 create mode 100644 newproject/settings.py
 create mode 100644 newproject/urls.py
 create mode 100644 newproject/wsgi.py
```

Files are now committed and ready to be pushed to our remote

git branch -M main

- **branch:** This Git command is used for various branch operations.
- **-M**: This flag moves/renames a branch. It's used here to rename the current branch to main.
- **main**: The new name of the branch.

Step 4: - Set remote

Files are now committed and ready to be pushed to our remote

git remote add origin <repo url>

Now that all the files are set locally we are ready to push them to our remote repository. We set the remote so when we run **git push**, git knows which remote repo to send these files to.

> git remote add origin https://github.com/divanov11/newproject.git

Step 5: Push to remote

> git push -u origin <branch name>

Now that we have our remote set, we can push our local github repo live. The default branch on github is called “master” unless you set your own. When you first create a repository on github you will see steps to rename this branch to “main”, if you decided to rename the default branch then use that name instead of master.

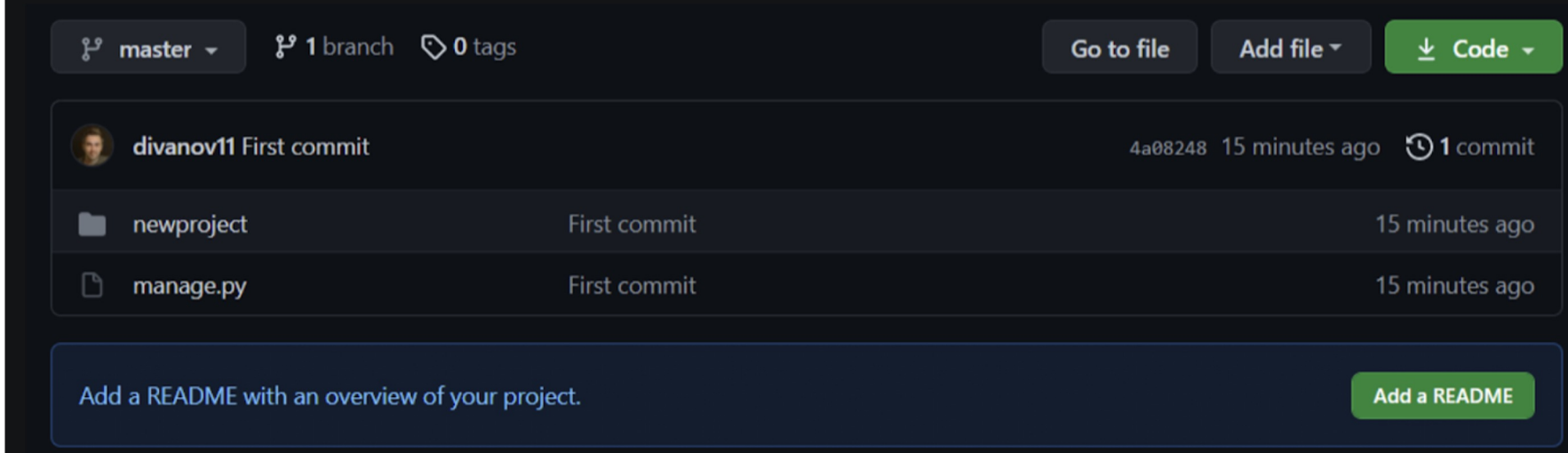
git push -u origin (master or main)

You can use **git branch** to see your branch is main or master

```
git push -u origin master
```

```
C:\Users\Dennis Ivy\Desktop\newproject>git push -u origin master
info: please complete authentication in your browser...
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 2.64 KiB | 2.64 MiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/divanov11/newproject.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Now if you refresh your repository on github.com you should see all your local files here.



After executing **git push -u origin main**, which pushes your local repository's changes to the remote repository and sets the upstream tracking for the **main** branch, your next steps will depend on your project's needs. **Here are some common actions you might take:**

Continue Development:

- Make changes to your code or project files.
- Use **git add <filename>** or **git add .** to stage your changes.
- Commit these changes with **git commit -m "Your commit message".**
- Periodically push your commits to the remote repository with **git push**.

Pull Changes from Remote:

- If you're collaborating with others, they might also make changes to the repository. To incorporate these changes into your local repository, use **git pull**.

Checking Repository Status:

- To see the status of your files (staged, unstaged, untracked), use **git status**.
- To view the commit history, use **git log.**

git clone is a Git command used to create a copy of an existing repository into a new directory on your local machine.

```
git clone https://github.com/username/repository.git
```

Some common problems and their solutions

1. Incorrect Remote URL

Problem: The most common issue is an incorrect URL for the remote repository. This can happen if you mistype the URL when setting up the remote.

Solution: You can correct the remote URL using the git remote set-url command. First, check the current remotes with **git remote -v**, then set the correct URL:

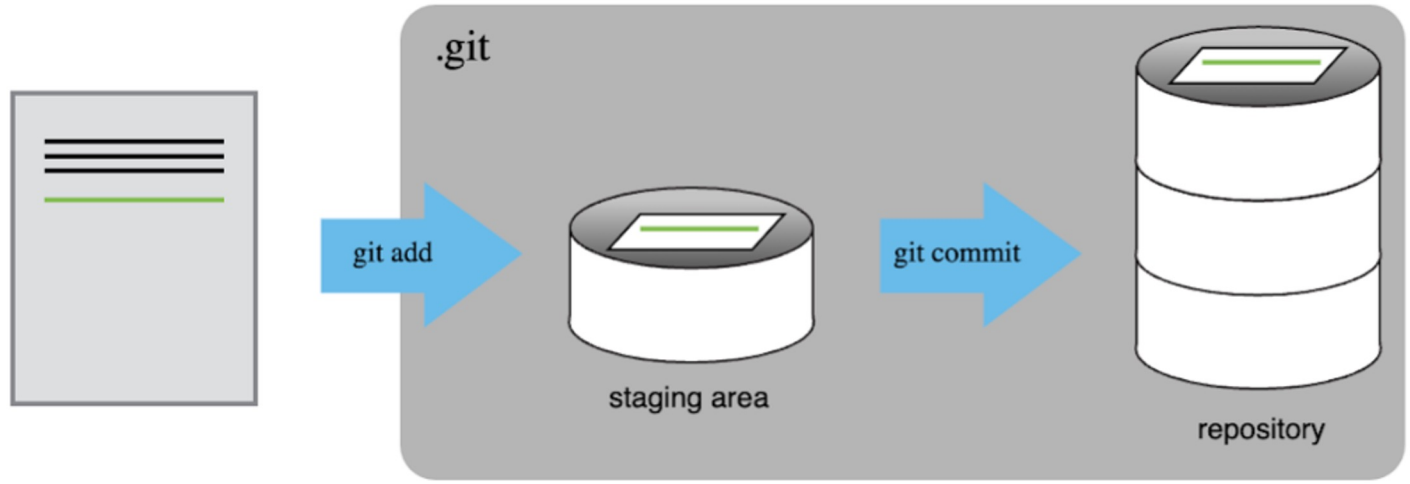
```
git remote set-url origin <correct-remote-url>
```

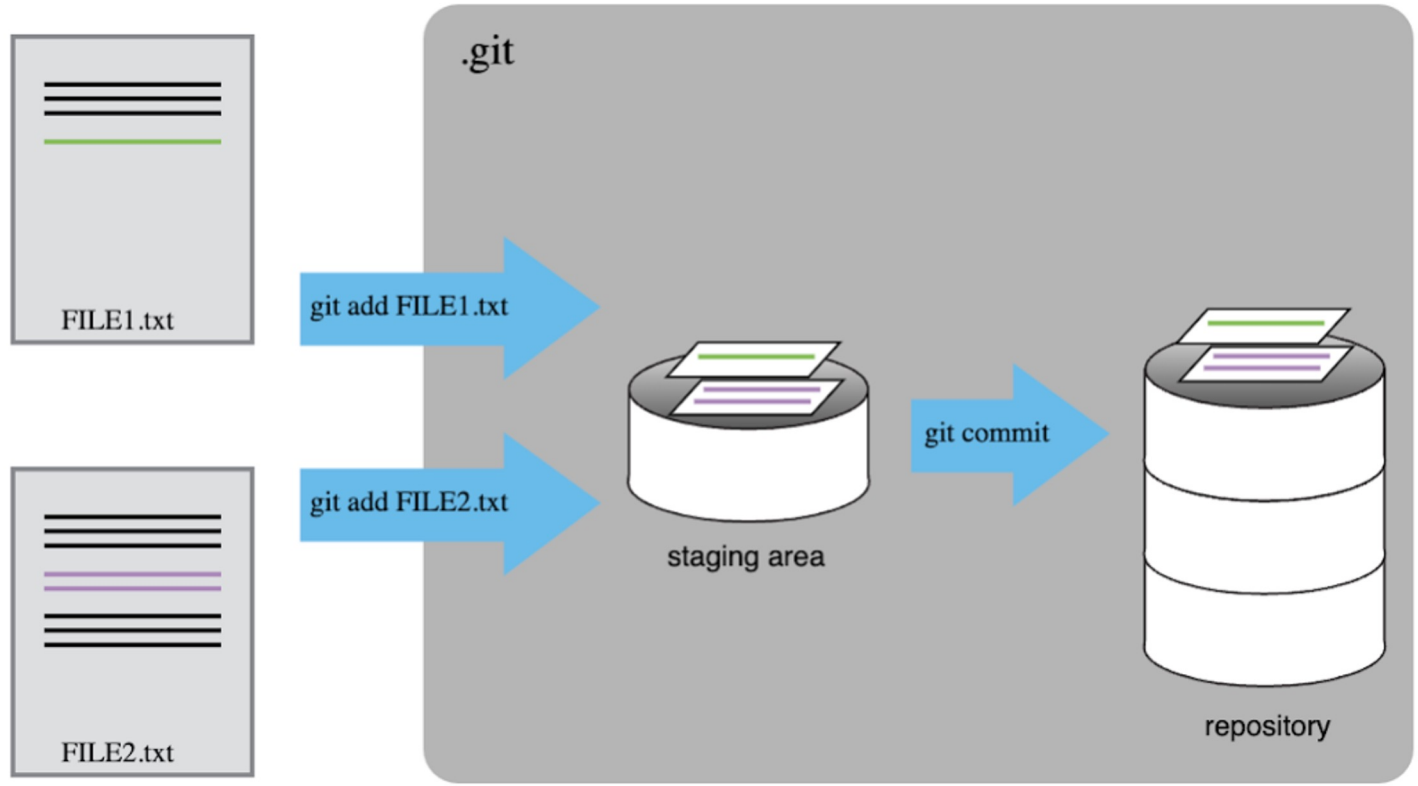
2. Branch Conflicts

Problem: Conflicts between your local and remote branches can prevent successful pushes, especially if the remote contains work that you do not have locally.

Solution:

- Perform a **git pull** followed by a **git merge** to integrate remote changes.
- Alternatively, use **git pull --rebase** to reapply your changes on top of the remote changes.


.git
git add
staging area
git commit
repository


.git
FILE1.txt
FILE2.txt
git add FILE1.txt
git add FILE2.txt
staging area
git commit
repository

Getting started with Git and GitHub:

1.Install Git: Start by installing Git on your computer. Visit the official Git website (https://git-scm.com/) and follow the installation instructions for your operating system.

2.Set up Git: After installation, configure Git with your name and email address using the following commands in the terminal:

$ git config --global user.name "Your Name"

$ git config --global user.email "youremail@example.com"

3.Create a Repository: You can create a new repository on GitHub by signing in to your account and clicking on the "New" button. Alternatively, you can initialize a local repository using the **git init** command in a desired directory.

4.Clone a Repository: To work with an existing repository, you can clone it to your local machine using the **git clone** command followed by the repository's URL. For example:

$ git clone https://github.com/username/repository.git

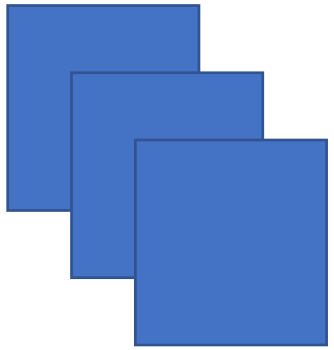
5.Make Changes and Commit: Start making changes to your files within the repository. Use **git add** to stage the changes you want to include in the next commit and **git commit** to create a commit with a meaningful message explaining the changes.

6.Branching and Merging: Create branches using **git branch** and switch between branches using **git checkout**. Use **git merge** to combine changes from one branch to another.

7.Push and Pull: To synchronize your local repository with a remote repository on GitHub, use **git push** to send your changes to the remote repository and **git pull** to fetch and merge changes from the remote repository to your local repository.

8.Collaborate on GitHub: Explore GitHub's features like pull requests, issues, and project management to collaborate with others and contribute to open source projects.

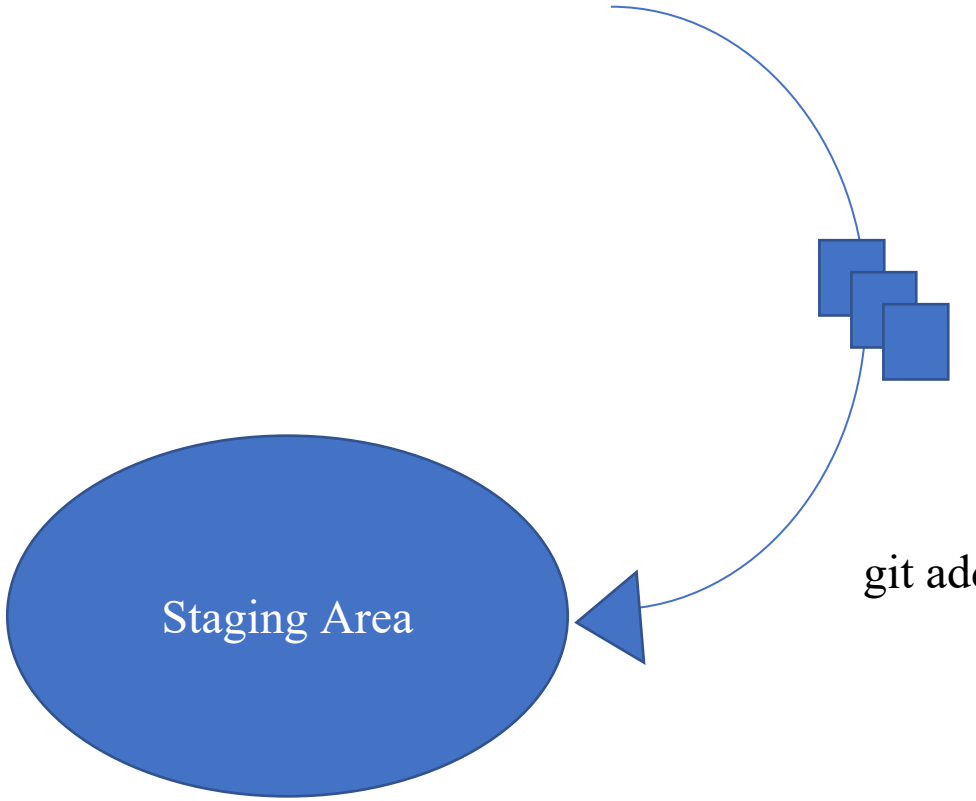
Git Workflow



You have files you want to put into a version control system

```
git init
```

Git Workflow



You have files you want to put into a version control system

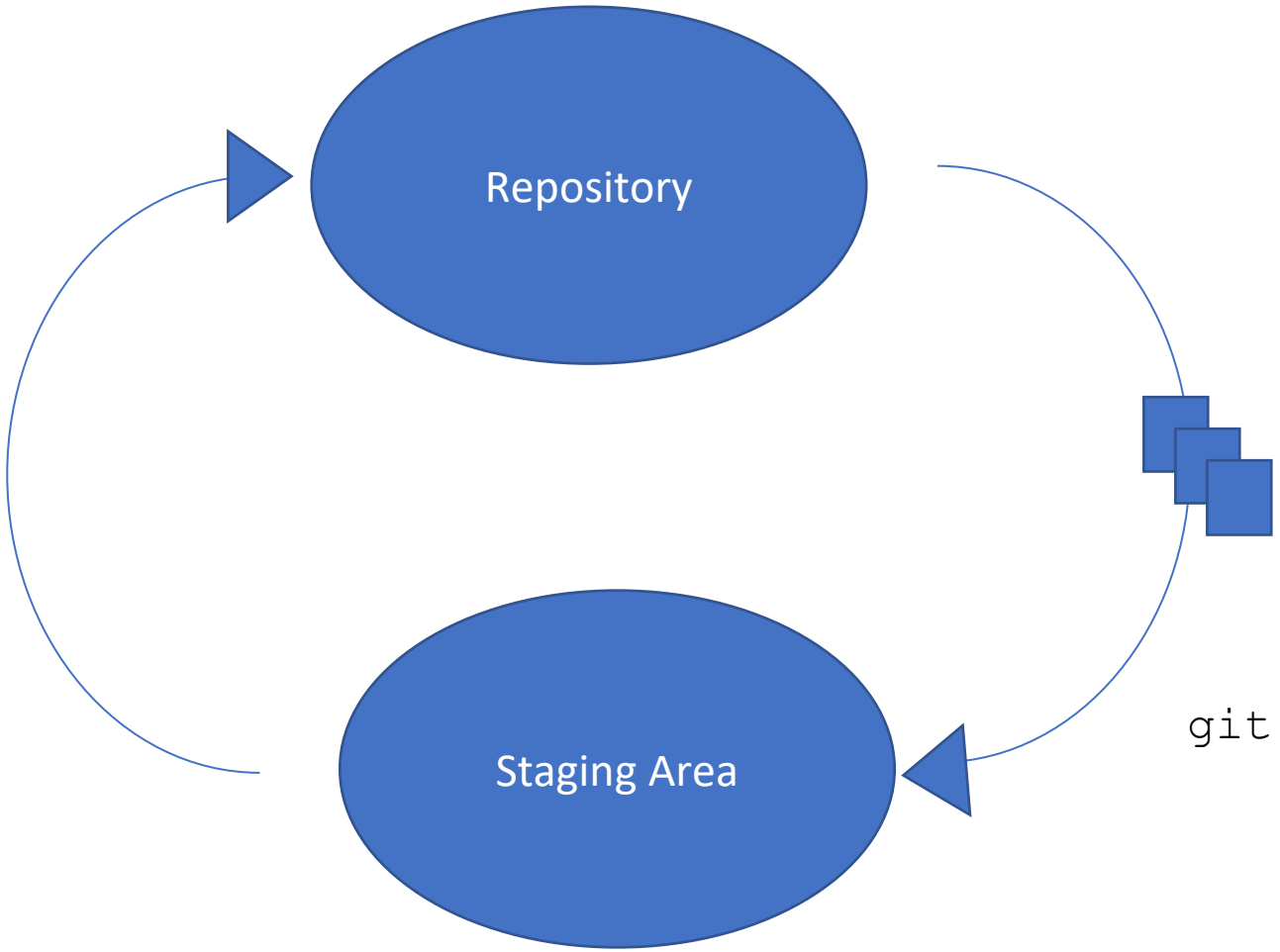
git init

edit files

git add

Add everything to the staging area you want to track

Git Workflow



You have files you want to put into a version control system

`git init`

edit files

`git add`

Add everything to the staging area you want to track

`git commit`

Commit your changes with a commit message (meta-data)

Laptop

Master Copy – on a web host (e.g. Github)