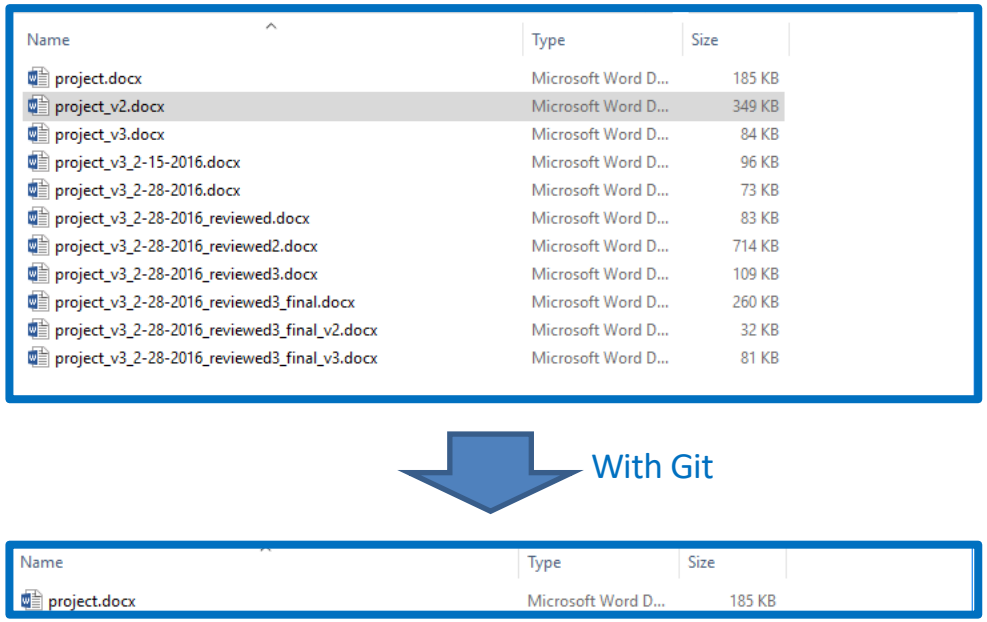
Git

Ergun Simsek

February 3, 2022

Why Do We Use Git?

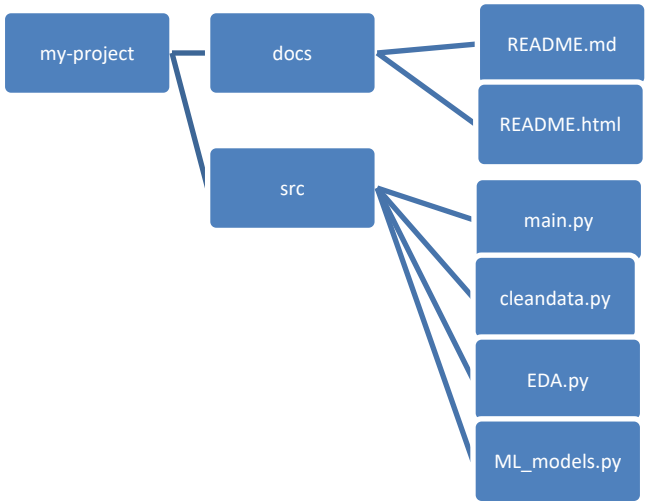


- ✓ Track all your changes
- ✓ Work along with others
- ✓ Share work with others

So, what is git?

- Git is a distributed version-control system, developed in 2005
- Terminology: In git-speak, a “version” is called a “commit.”
- Git keeps track of the history of your commits, so you can go back and look at earlier versions, or just give up on the current version and go back some earlier version.
- You can synchronize your repository with a repository on a server
- If you move from one machine to another, you can pick up the changes by synchronizing with the server.
- You can share the repository with others (to collaborate). If someone else on your team uploads some changes to your files, you can pick those up by synchronizing with the server.

Your files

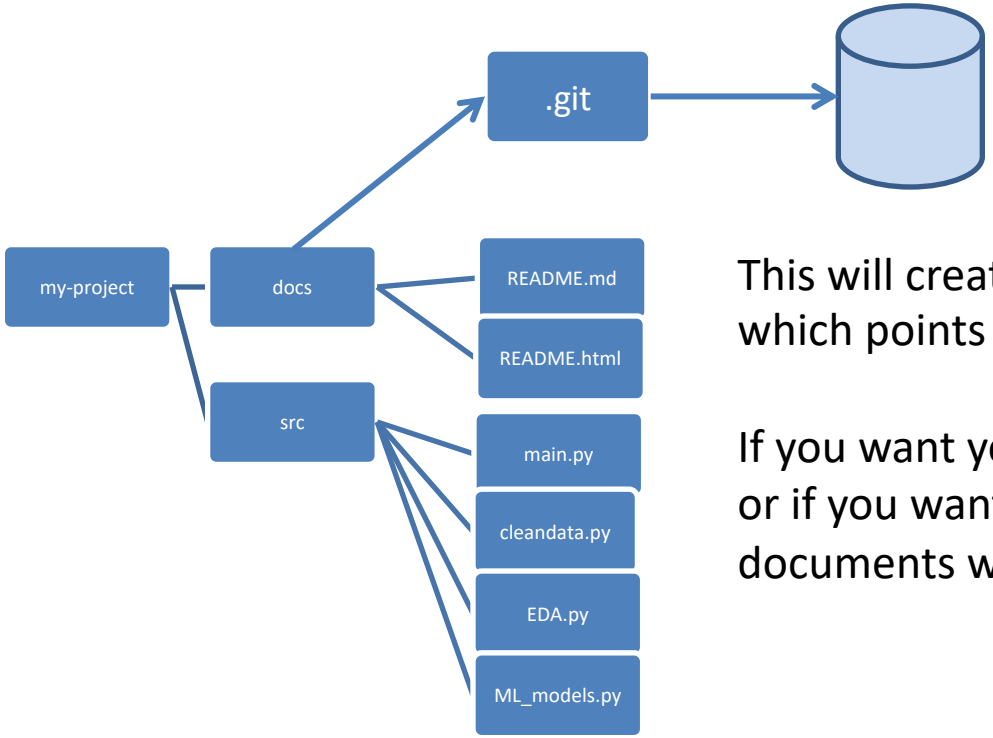


Here are your files, sitting in a directory called my-project

To **initiate** a git
Open a terminal and go to this "my-project" folder and type

```
$ git init
```

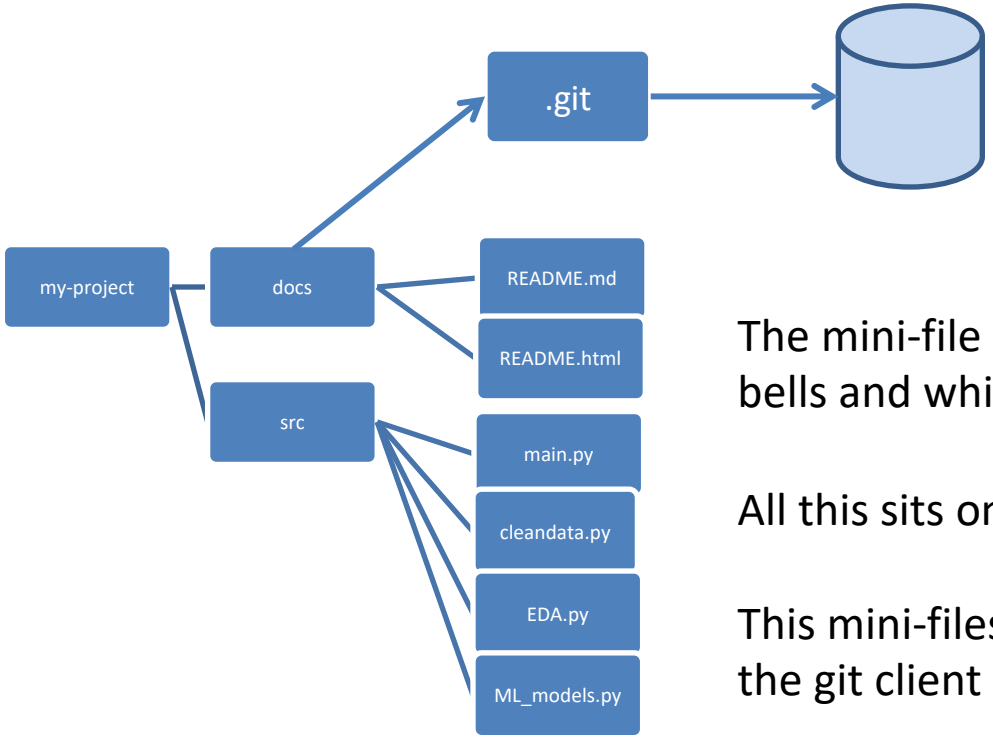
Your files in your git repository



This will create an additional directory called .git, which points at a mini-filesystem.

If you want you can **add** the entire directory to git or if you want you can **add** exactly what documents will belong to git.

Your files in your git repository



The mini-file system keeps all the data, plus the bells and whistles that git needs to do its job.

All this sits on your local machine.

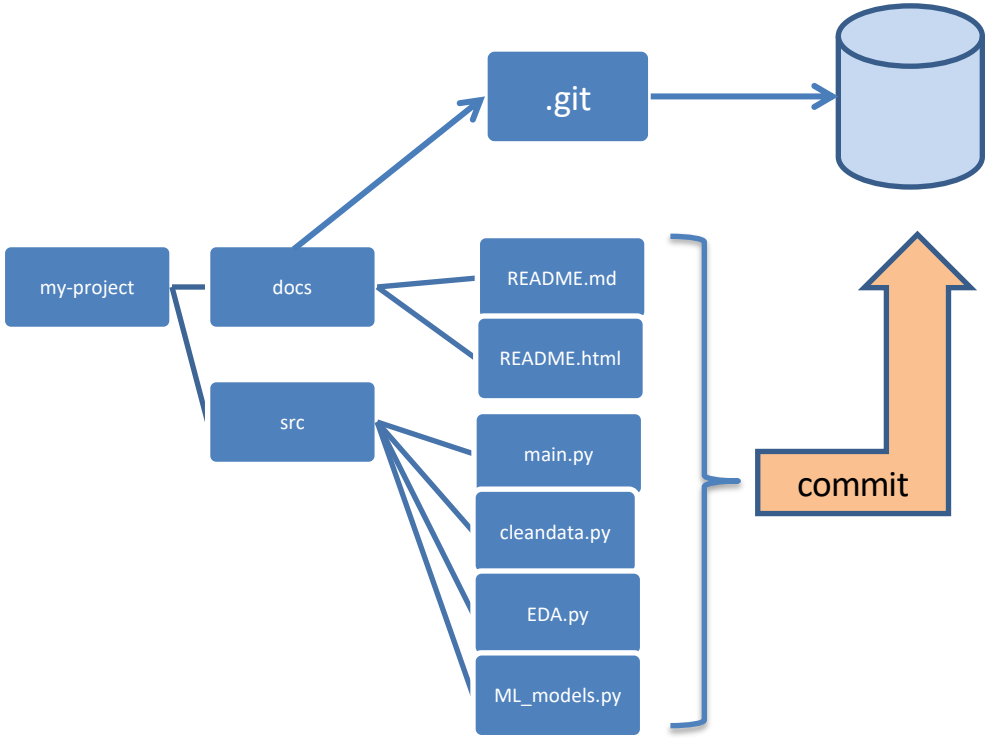
This mini-filesystem is highly optimized. The job of the git client is to manage this for you.

Your workflow (part 1)

- You edit your local files directly.
  - You can edit, add files, delete files, etc., using whatever tools you like.
  - This doesn't change the mini-filesystem, so now your mini-filesystem is behind.

- When you are ready, **commit**!

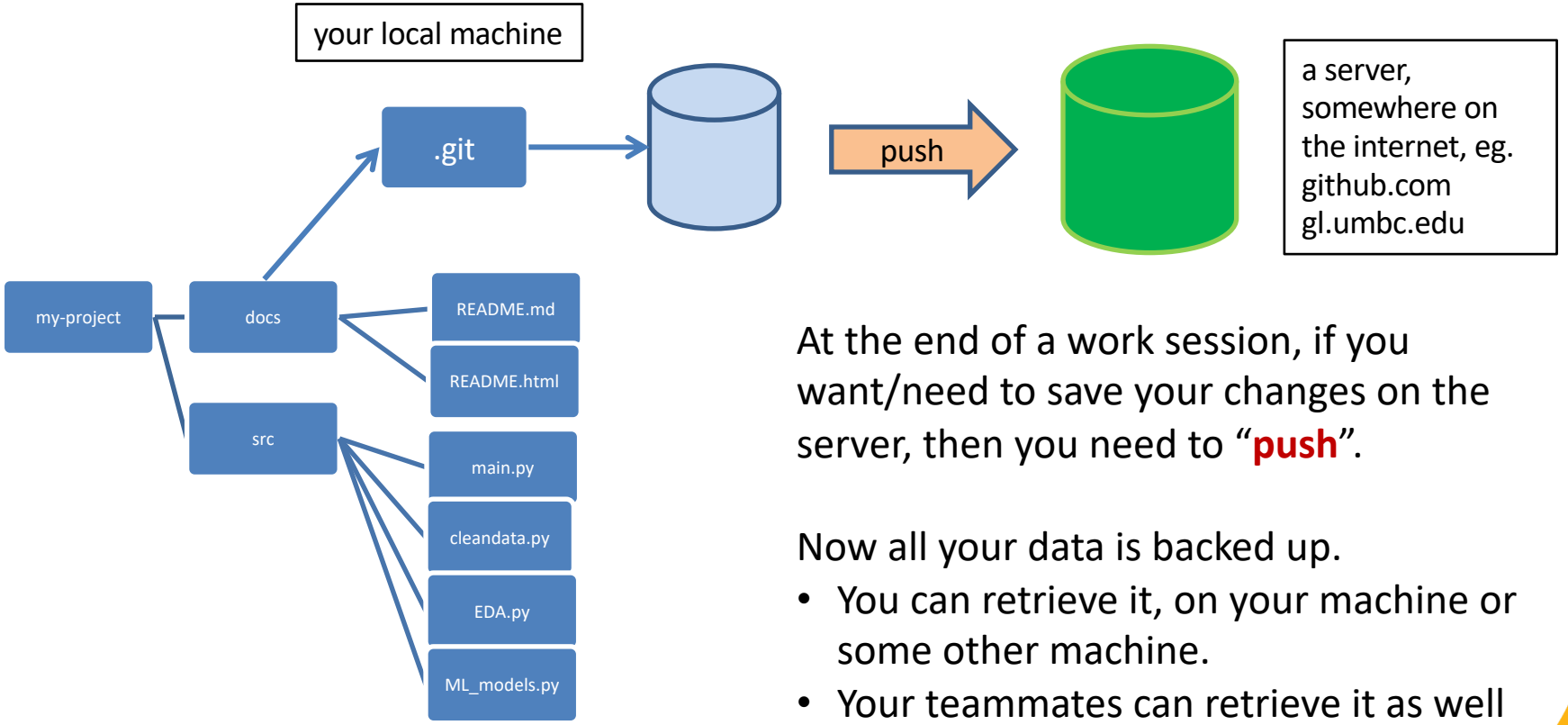
A Commit



When you do a "**commit**", you record all your local changes into the mini-filesystem.

The mini-filesystem is "append-only". Nothing is ever over-written there, so everything you ever commit can be recovered.

Synchronizing with the server (1)

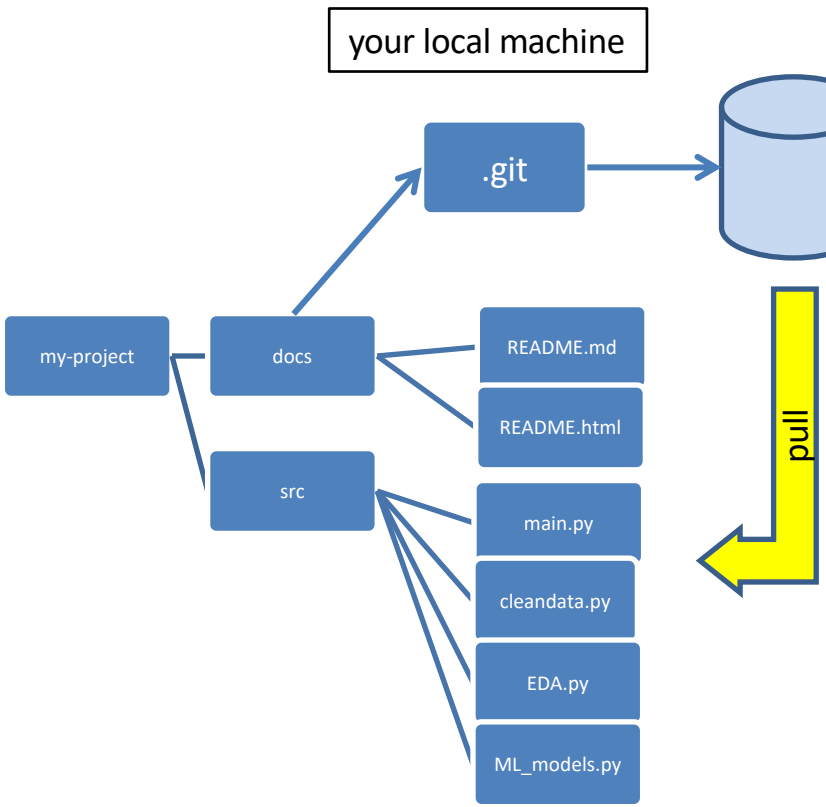


At the end of a work session, if you want/need to save your changes on the server, then you need to "**push**".

Now all your data is backed up.

- You can retrieve it, on your machine or some other machine.
- Your teammates can retrieve it as well

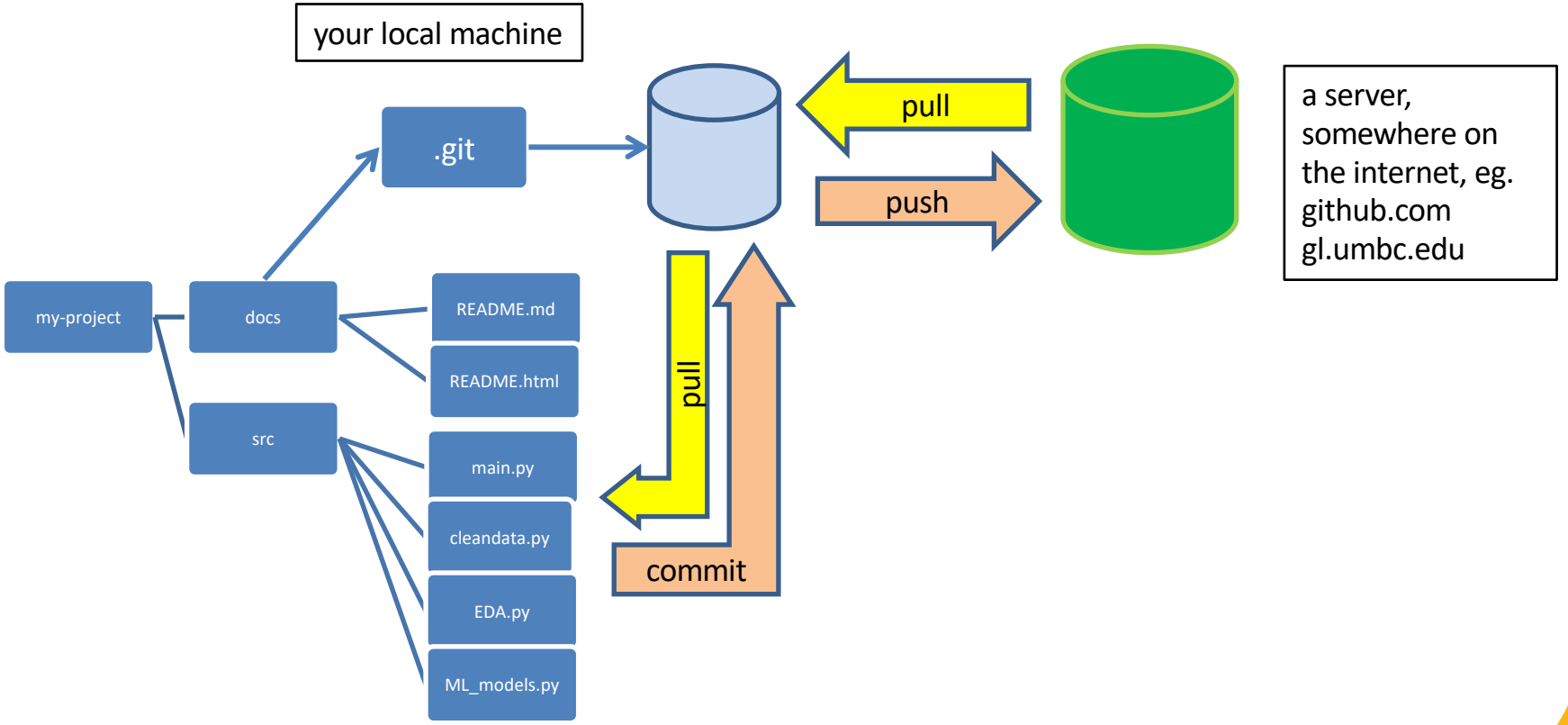
Synchronizing with the server (2)



To retrieve your data from the server, you do a "**pull**". A "pull" takes the data from the server and puts it both in your local mini-fs and in your ordinary files.

If your local file has changed, git will merge the changes if possible. If it can't figure out how to the merge, you will get an error message. We'll talk about this some more a little later

The whole picture



Installing Git

https://git-scm.com/downloads

You can check whether you already have git

```
(base) DPSITs-iMac:~ ergunsimsek$ git --version
git version 2.24.3 (Apple Git-128)
```

You better get the latest version

1. install homebrew,
2. 2. `brew install git`

Step-1: Basic Configuration

Open a terminal

Tell git your name and email:

```
git config --global user.name “Your Full Name”
git config --global user.email your_uni@umbc.edu
```

You can set the default editor, e.g. nano or pico or whatever you have/like

```
git config --global core.editor nano
git config --global content.editor "pico -w"
```

You can set **main** as the default branch name do:

```
git config --global init.defaultBranch main
```

You only have to do this once. You can make changes later, if you want

To check the current configuration

```
git config --list
```

For Example

```
git config --global user.name “Ergun Simsek”
git config --global user.email simsek@umbc.edu
git config --global core.editor nano
git config --global init.defaultBranch main
```

Getting Help

```
git help <verb>
git <verb> --help
man git-<verb>
```

For example

```
git help config
```

How to Get a Git Repository

In git, getting someone else's repo is called cloning

If you know the address, it is very easy

Let’s say, the address is

https://github.com/simsekergun/git_test

Then you just need to type the following (in the terminal, where you want to have a local copy of this git

```
git clone https://github.com/simsekergun/git_test.git
```

Cloning from GitHub

```
git clone https://github.com/simsekergun/git_test.git
```

```
(base) DPSITs-iMac:git ergunsimsek$ git clone https://github.com/simsekergun/git_test.git
Cloning into 'git_test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Check what we get:

```
(base) DPSITs-iMac:git ergunsimsek$ ls
README.md       git_test
```

Checking the Status of Your Files

The main tool you use to determine which files are in which state is the `git status` command. If you run this command directly after a clone, you should see something like this:

```
(base) DPSITs-iMac:git ergunsimsek$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git_test/

nothing added to commit but untracked files present (use "git add" to track)
```

This means you have a clean working directory; in other words, none of your tracked files are modified.

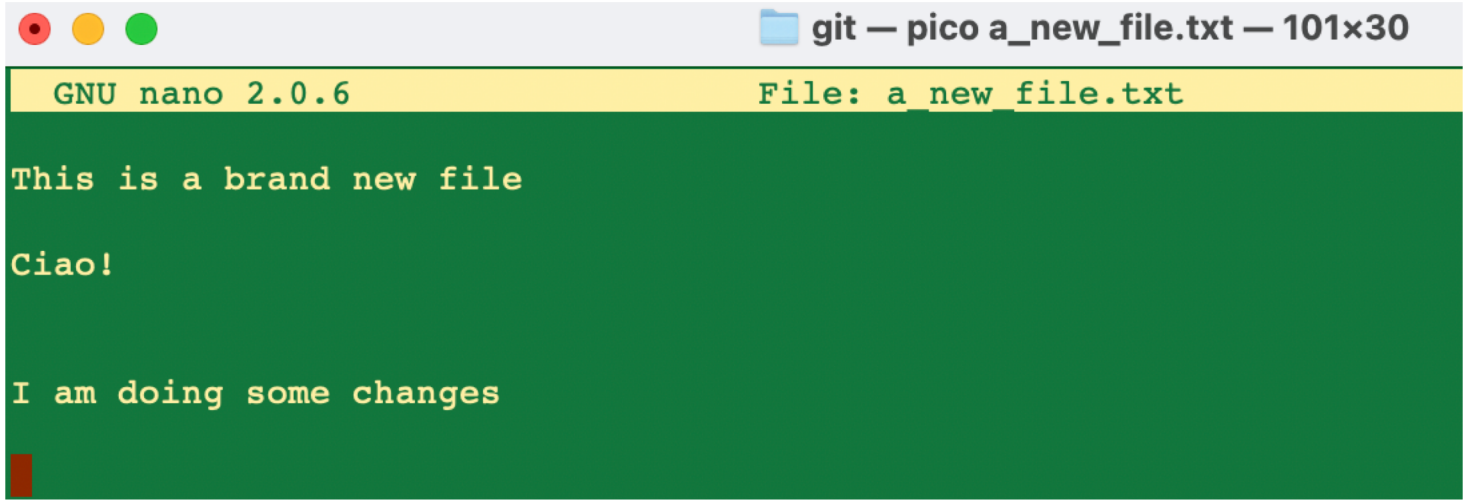
Let’s make a change

Let’s create a new file called “a_new_file.txt”

Then add it to the project

```
(base) DPSITs-iMac:git ergunsimsek$ nano a_new_file.txt
(base) DPSITs-iMac:git ergunsimsek$ git add a_new_file.txt
(base) DPSITs-iMac:git ergunsimsek$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   a_new_file.txt

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

Let’s make some changes



Let’s see what happens to the status

```
(base) DPSITs-iMac:git ergunsimsek$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   a_new_file.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a_new_file.txt

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

Let's Commit

```
git commit -a -m 'first commit'
```

your personal note here

```
(base) DPSITs-iMac:git ergunsimsek$ git commit -a -m 'first commit'
[main 401ba52] first commit
 2 files changed, 11 insertions(+)
 create mode 100644 a_new_file.txt
 create mode 100644 another_file.txt
```

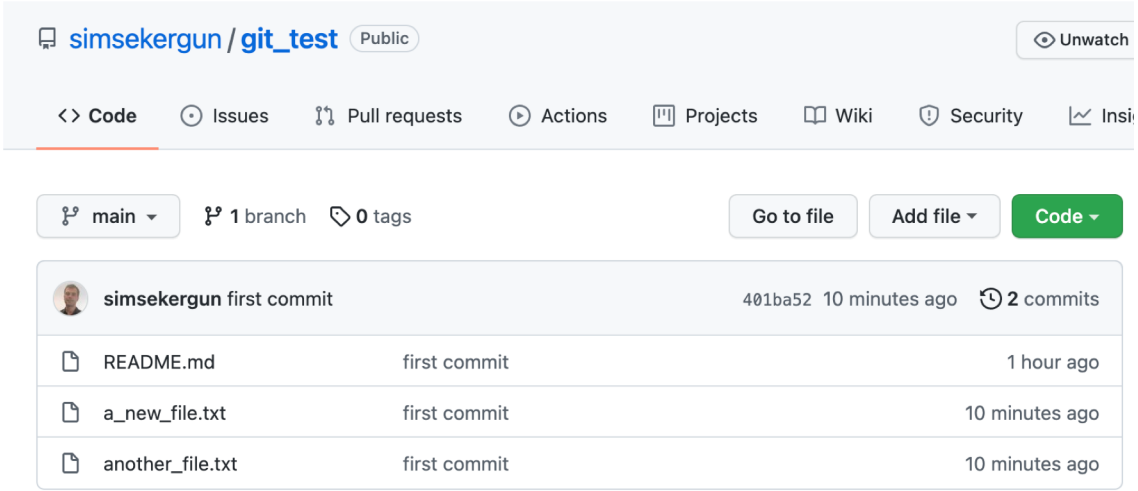
Viewing the Commit History

```
(base) DPSITs-iMac:git ergunsimsek$ git log
commit 401ba5230b4debeb18a6206c305907d243ffe470 (HEAD -> main)
Author: Ergun Simsek <simsek@umbc.edu>
Date:   Fri Dec 10 17:12:41 2021 -0500

    first commit

commit 39eb3c37d97d5ed1cded90231928d1cb5f5eaac1
Author: simsekergun <simsek@umbc.edu>
Date:   Fri Dec 10 16:32:24 2021 -0500

    first commit
```

Pushing to Your Remotes

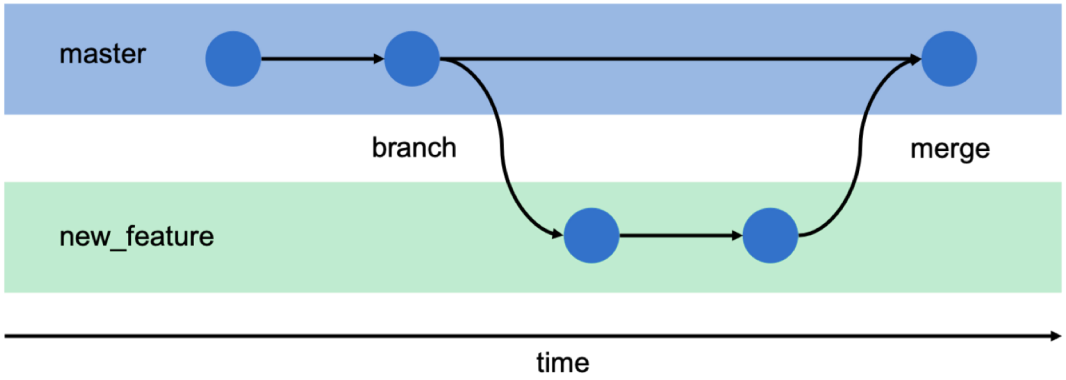
```
(base) DPSITs-iMac:git ergunsimsek$ git push -f origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (7/7), 598 bytes | 85.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/simsekergun/git_test.git
 + 68c2ac2...401ba52 main -> main (forced update)
```

Voilaaaaa!



Branches

- Branches are used to develop features isolated from each other.
- The master/main branch is the "default" branch when you create a repository.
- Use other branches for development and merge them back to the master branch upon completion.



Branching

create a new branch named "feature_x" and switch to it using

```
git checkout -b feature_x
```

switch back to master

```
git checkout master
```

and delete the branch again

```
git branch -d feature_x
```

a branch is not available to others unless you push the branch to your remote repos

```
git push origin <branch>
```

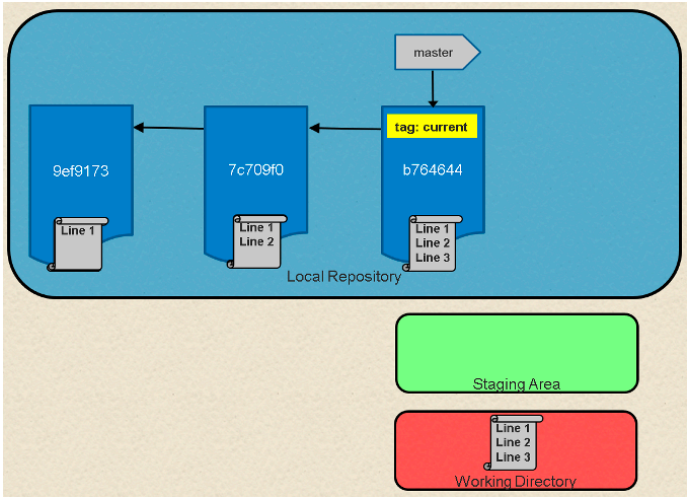
to merge another branch into your active branch (e.g. master), use

```
git merge <branch>
```

How to reset, revert, and return to previous states in Git

- reset = rollback to a previous commit (it points your local environment back to a previous commit)

```
$ git log --oneline
b764644 File with three lines
7c709f0 File with two lines
9ef9173 File with one line
```



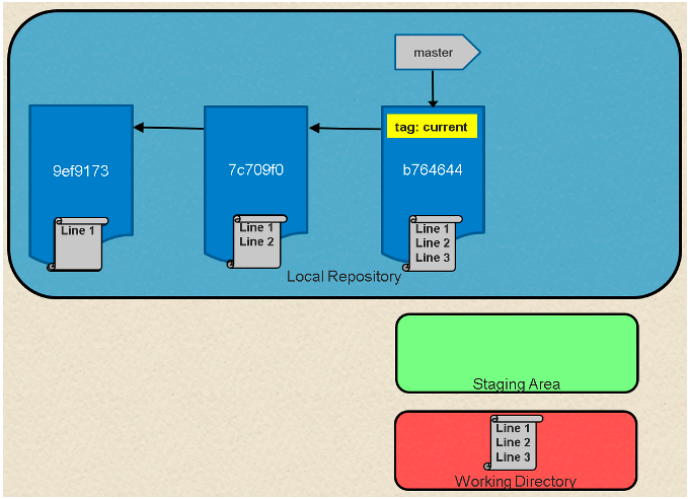
Method-1: **$ git reset 9ef9173**

Method-2: **$ git reset current~2**

(using a relative value -2 before the "current" tag)

How to reset, revert, and return to previous states in Git

- revert = similar to reset (but reset does undo and goes back, revert command adds a new commit at the end of the chain to "cancel" changes)



Method-1: `$ git revert HEAD`

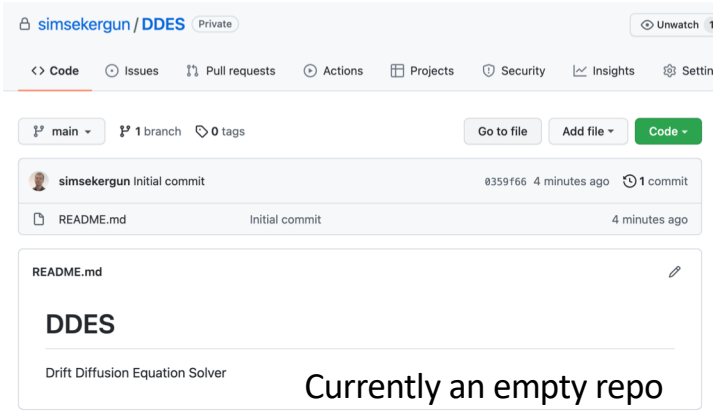
Method-2: `$ git revert 7c709f0`

SOME ADDITIONAL NOTES

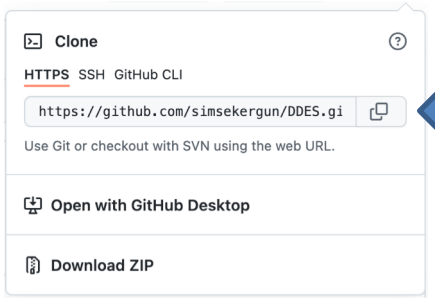
For GitHub First-timers

Using Git with GitHub

- Create a GitHub account at github.com
- Get a Token (Profile > Settings > Developer Settings > Personal Access Tokens > Generate new token)
- Create a repository (Repositories > New) [Public or Private]



Click



Click Copy

On Your Local Machine

Assuming you have `git` on your machine

- Open a terminal
- Go to a folder that you want to `git`
- Clone, e.g.

```
$ git clone https://github.com/simsekergun/DDES.git
```

- This will create a folder with the name of the repo, e.g. DDES, go to that folder

```
$ cd DDES
```

First Steps

- Check what you have

```
$ ls –a

.    ..   .git     README.md
```

- I copy *.m files to this folder

```
$ cp ../*.m .
```

- I add all the *.m files to my `git`

```
$ git add --all
```

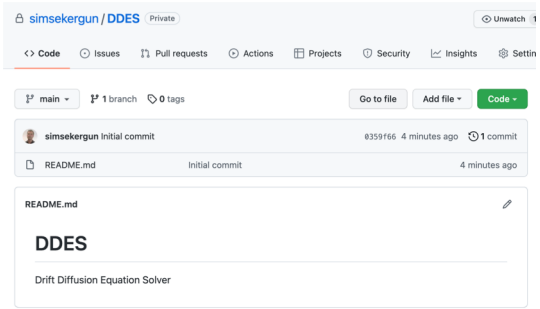
Then

- Commit

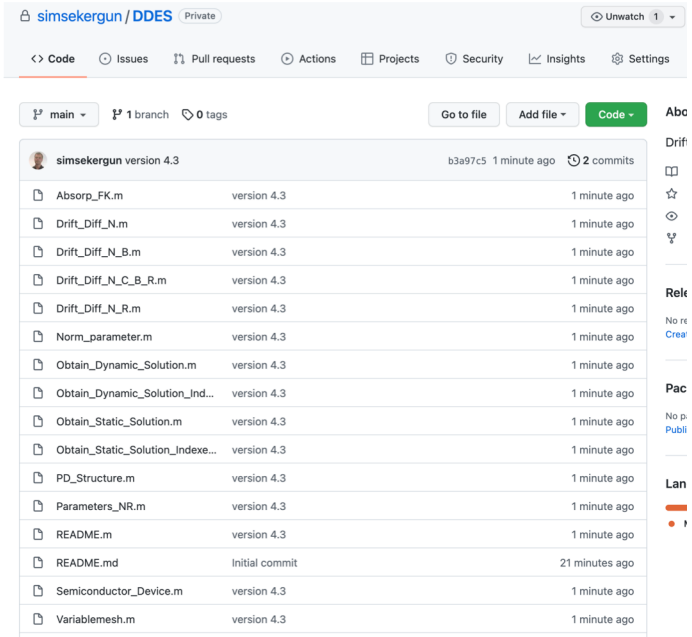
```
$ git commit –m 'version 4.3'
```

- Push

```
$ git push
```



It was empty and after “push ”



How to Collaborate on GitHub

- Under your repository name, click **Settings**
- In the "Access" section of the sidebar, click **Collaborators & teams**.
- Click **Invite a collaborator**.
- In the search field, start typing the name of person you want to invite, then click a name in the list of matches.
- Click **Add NAME to REPOSITORY**.
- The user will receive an email inviting them to the repository. Once they accept your invitation, they will have collaborator access to your repository.

For Example, Now

- Ishraq can clone this repo
- He can make some changes
- He can push these changes to this repo
- I can pull with a single command to see changes made
- If we want to add a new feature
  - We can add this feature to the main code after testing
  - If we make a mistake, we can roll back
- Even though we might have our own copies of the code, there will be one and only code

We Don’t Have to Use GitHub

- UMBC has GL servers, which support `git`
- Just use ssh, e.g.

```
$ ssh simsek@gl.umbc.edu
```

and login with your UMBC password

We Can Use `git` for Teaching As Well

- To collect the e-solutions of students to HW assignments/projects
- Setup

https://www.csee.umbc.edu/~simsek/git/Setting_Git_for_submission.html

- Example

https://www.csee.umbc.edu/~simsek/git/Git.html

Oh no! I've got a git conflict

- If you update your work before you pull, `git` will try to merge the changes when you do pull.
- If `git` can't figure out how to do the merge, it will go into a special mode in which it expects you to resolve the conflicts.
- When you have the files the way you want them, you can commit them and continue working. Here's a guide https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line

Other ways to use git and github

- There are lots of possible ways to use `git` and github.
- There are many resources, interactive git tutorials on the web.

  http://book.git-scm.com/book/en/v2

  http://think-like-a-git.net/