Git and GitHub

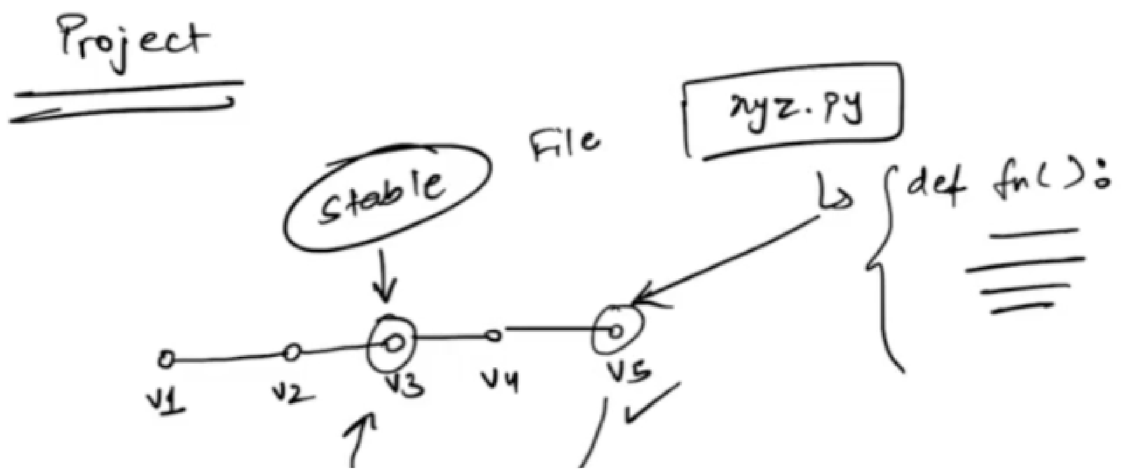
Content

- Git with GitHub Desktop
  - Version Control System
  - Git vs. GitHub
  - Basic Terminologies
  - Creating a repository
  - Pushing and Collaboration
- Git with CLI
  - Install & Initialize Git
  - Staging Environment
  - Git Commands
  - Git Branches
  - Creating a github repository
  - Git Push & Pull
  - Git Rebase & Merge
  - GitHub Fork & Clone

Git using GitHub

Intuition for Git

- Suppose you start working on a long-term project, and you're making changes to the project daily.
- One day you realize that the previous version of a function made more sense and that you have wanted to return to that version.
- Only if we had a place where we kept all versions of our code, we could have done it easily.



What is Git?

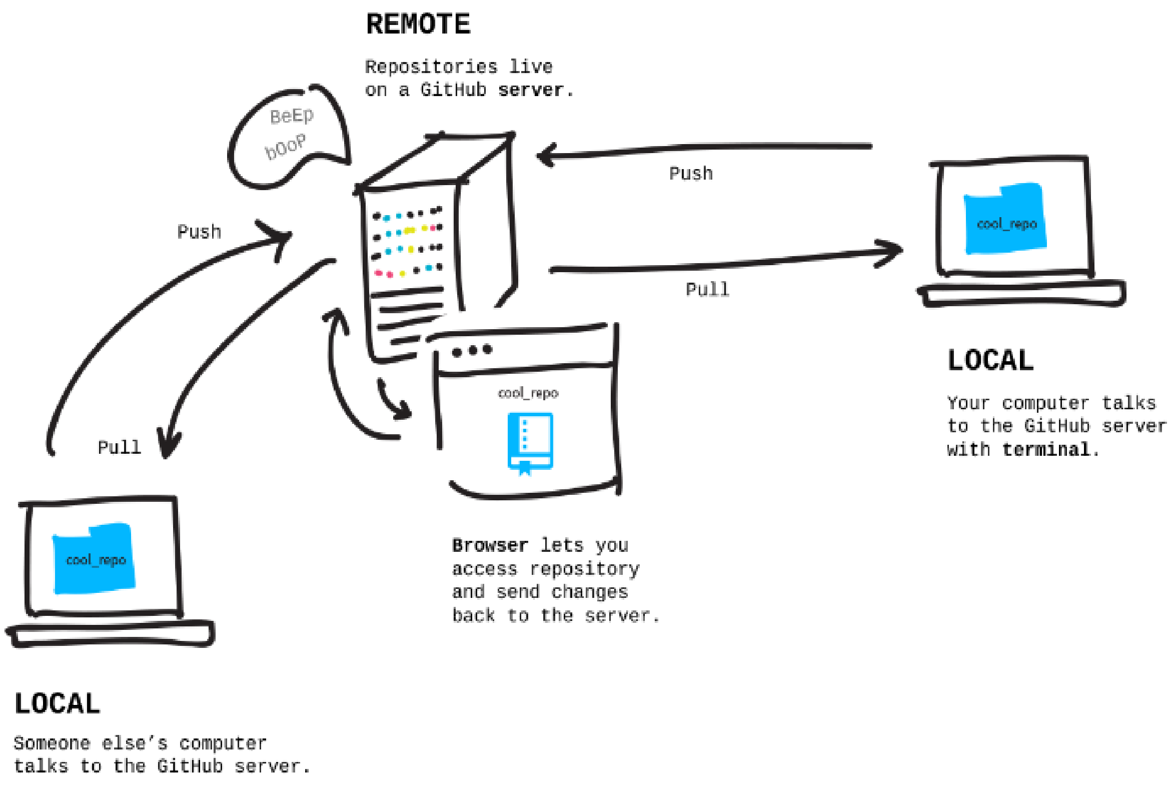
- Git is a VCS (version control system).
- But Git isn't any VCS, it's a distributed VCS which means that
    - Every collaborator of the project will have a history of the changes made on their local machine.
- It is distributed so you can access your code files from another computer and so can other developers.
- People can work on different features of the project without having to communicate with each other or the server hosting the project.

What is GitHub?

- It is a platform for version control that uses Git at its core.
- It lets you host the remote version of your project from where all the collaborators can have access to it.
- Not just your own team members but any member of GitHub can contribute to your code (if it is a public project and you accept the changes proposed).
- It also serves as a platform where people can find a plethora of open-source projects with their codes.

There are other platforms as well that offer this:

- Bitbucket
- GitLab


REMOTE
Repositories live on a GitHub server.
BeEp bOoP
Push
Pull
Push
Pull
cool_repo
cool_repo
cool_repo
LOCAL
Your computer talks to the GitHub server with terminal.
Browser lets you access repository and send changes back to the server.
LOCAL
Someone else's computer talks to the GitHub server.

Here's how it all fits together:

- You write your story on your computer using Git to keep track of changes.
- You save snapshots (called commits) of your story with Git so you can go back in time if needed.
- You put your story on GitHub, making it available on the internet for others to see.
- Others can make copies of your story from GitHub to work on their versions.
- They can suggest changes or improvements to your story and send them back to you (this is called a pull request).
- You review their changes and decide if you want to include them in your story.
- GitHub helps manage this collaboration, making it easier for everyone to work together on the same project.

Difference between Git and GitHub

- Git is a VCS that manages and keeps track of your code.
- GitHub is a service that lets you host, share, and manage your code files on the internet.

Please make sure -

- You have GitHub Desktop installed,
- Created and signed into your GitHub account.
- Ref : https://docs.github.com/en/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop

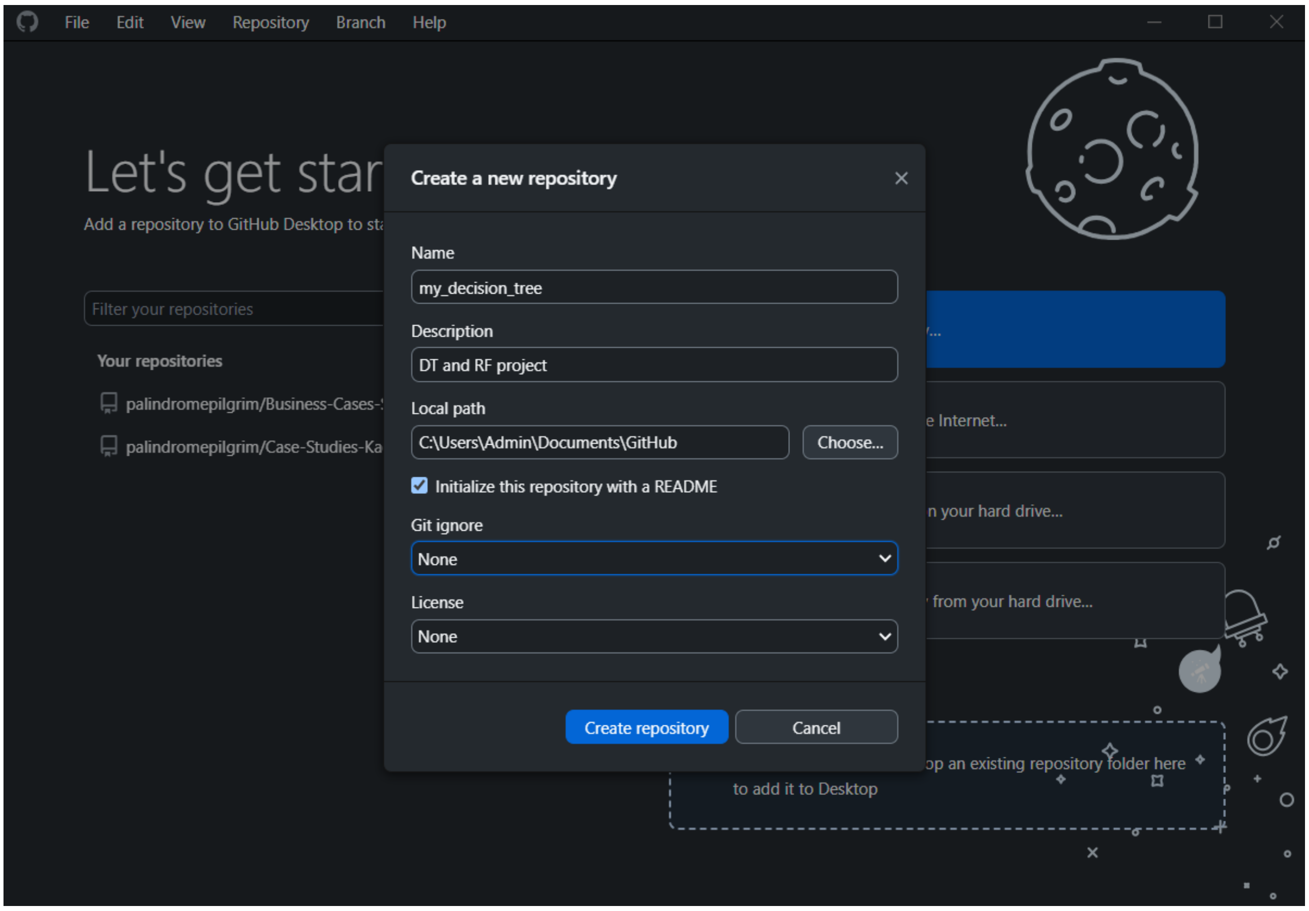
What is a Repository?

A **Repository** is like a folder where you keep all the files for your project.

- **Local Repository**: Think of this as your personal folder on your computer. It's where you work on your project, make changes, and save your progress.
- **Remote Repository**: This is like a backup or a shared folder on the internet. It's where you store your project online, so others can see it and collaborate with you.

Create a repository -

1. Click "Create a New Repository on your Hard Drive..."
2. In the “Create a New Repository” window, fill in the fields -
    - **Name** - Defines the name of your repository both locally and on GitHub.
    - **Description** - Self explanatory. Optional field.
    - **Local path** - Location of the repository on your computer.
3. **Initialize this repository with a README** - Creates an initial commit with a README.md file.
    - README file helps people understand the purpose of your project.
4. **Git ignore** drop-down menu lets you add a custom file to ignore specific files in your local repository.
5. **License** drop-down menu lets you add an open-source license to a LICENSE file in your repository.
    - You don't need to worry about adding a license right away.
6. Click Create repository.



Create a new repository

Name: my_decision_tree

Description: DT and RF project

Local path: C:\Users\Admin\Documents\GitHub (Choose...)

☑ Initialize this repository with a README

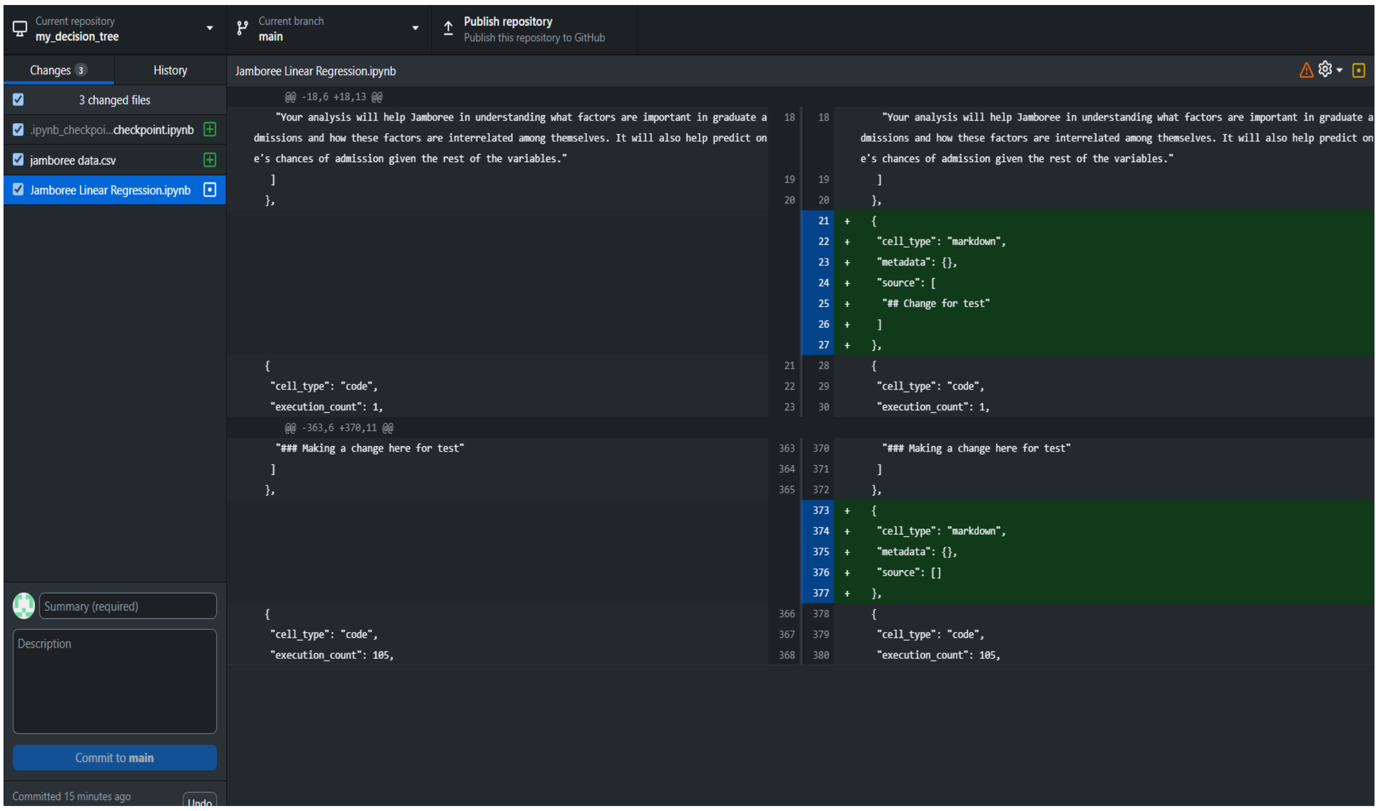
Git ignore: None

License: None

Create repository | Cancel

Working on the repository -

1. We can now go to the git folder that we want to work with and open a file in the code editor.
2. We can update our code or even add/remove some files.
3. If we then go back to the repository in github desktop, it will show the difference in the two versions of the updated file.



Jamboree Linear Regression.ipynb

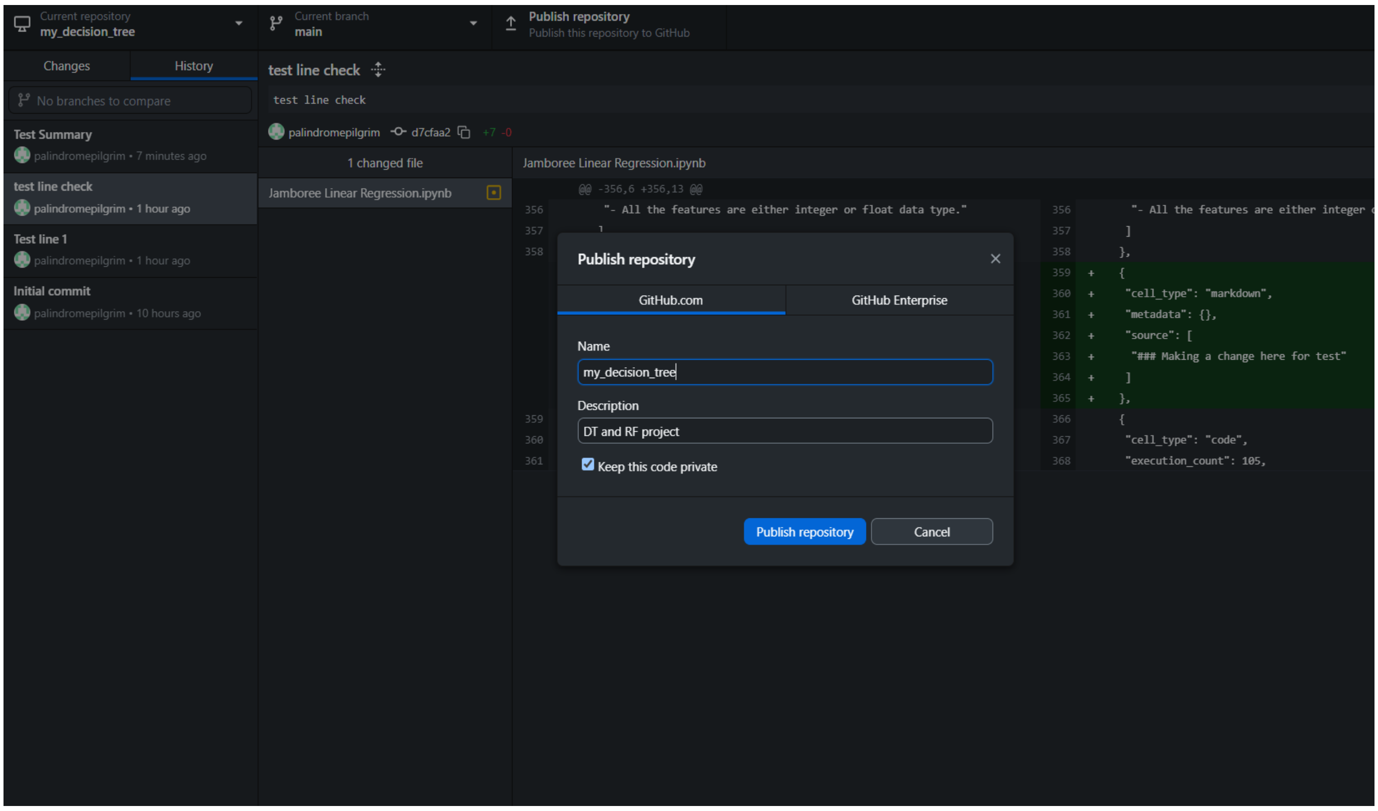
```
@@ -18,6 +18,13 @@
    "Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables."
   ]
  },
+ {
+  "cell_type": "markdown",
+  "metadata": {},
+  "source": [
+   "## Change for test"
+  ]
+ },
  {
   "cell_type": "code",
   "execution_count": 1,
@@ -363,6 +370,11 @@
    "### Making a change here for test"
   ]
  },
+ {
+  "cell_type": "markdown",
+  "metadata": {},
+  "source": []
+ },
  {
   "cell_type": "code",
   "execution_count": 105,
```

4. Now we can add a summary, description and click on '**Commit to main**'.
5. Our history is saved. We can check it in the **History** tab.

But these changes are only available on your local machine because we haven't pushed anything to the remote repository yet.

You can publish your repository to GitHub to keep it synchronized across multiple computers and allow other people to access it.

1. To publish your repository, **push** your local changes to GitHub.
2. In the **Publish Repository** window, enter details for your new repository.
3. **Keep this code private** lets you control who can view your project.
4. The **Organization** drop-down menu, if present, lets you publish your repository to a specific organization that you belong to on GitHub.
5. Click Publish Repository.



Current repository: my_decision_tree
Current branch: main
Publish repository — Publish this repository to GitHub

No branches to compare

Test Summary — palindromepilgrim • 7 minutes ago
test line check — palindromepilgrim • 1 hour ago
Test line 1 — palindromepilgrim • 1 hour ago
Initial commit — palindromepilgrim • 10 hours ago

test line check

test line check

palindromepilgrim d7cfaa2 +7 -0

1 changed file

Jamboree Linear Regression.ipynb

```
@@ -356,6 +356,13 @@
    "- All the features are either integer or float data type."
   ]
  },
+ {
+  "cell_type": "markdown",
+  "metadata": {},
+  "source": [
+   "### Making a change here for test"
+  ]
+ },
  {
   "cell_type": "code",
   "execution_count": 105,
```

Publish repository

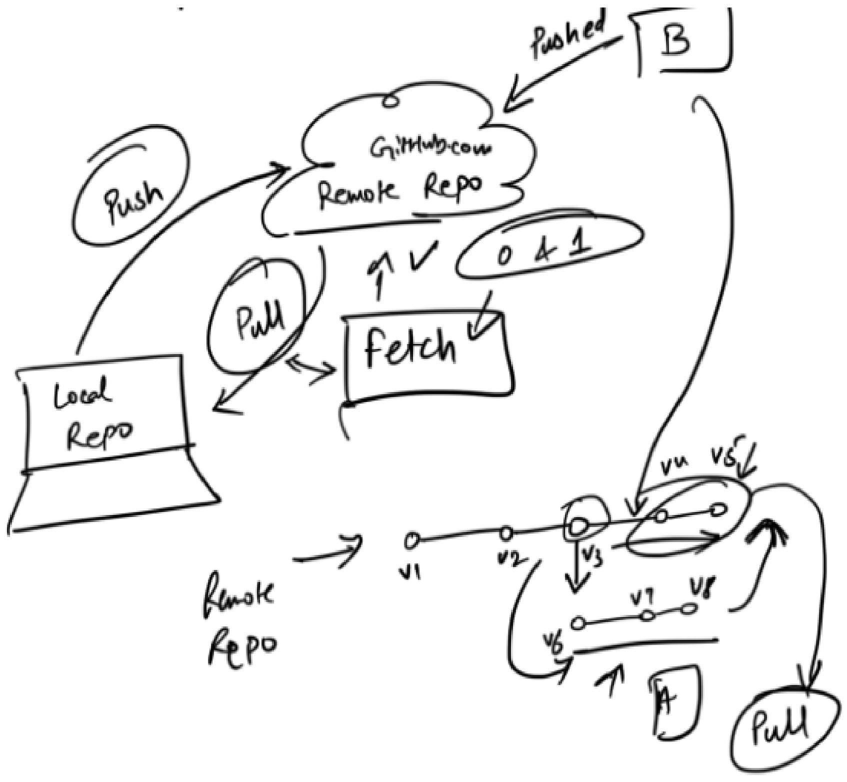
Name: my_decision_tree

Description: DT and RF project

☑ Keep this code private

Pushing and Collaboration

- **Pull**: Pull not only downloads changes from a remote repository but also merges them into your current working branch automatically.
- **Fetch**: Fetch only downloads changes from a remote repository and stores them in your local repository but doesn't automatically merge them into your current working branch.
    - Use “fetch” when you want to see what changes exist in the remote repository but don't want to merge them immediately.
    - This allows you to review changes before deciding to merge.
- **Conflict**: A conflict occurs in Git when multiple people make changes to the same part of a file, and Git can't automatically decide which changes to accept.
    - To resolve a conflict, you need to manually edit the file, keeping the changes you want and removing the conflicting parts.
    - After resolving the conflict, you can commit the resolved file.

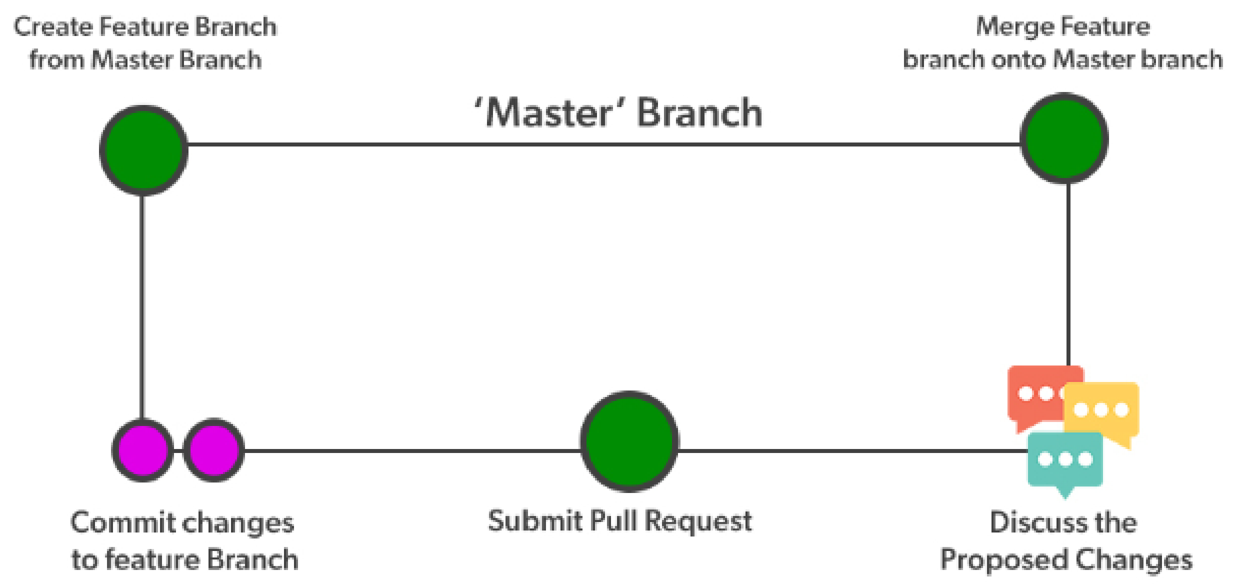


Collaborative Development

The main branch (often called “master” or “main”) in a Git repository typically represents the stable and production-ready version of your project.

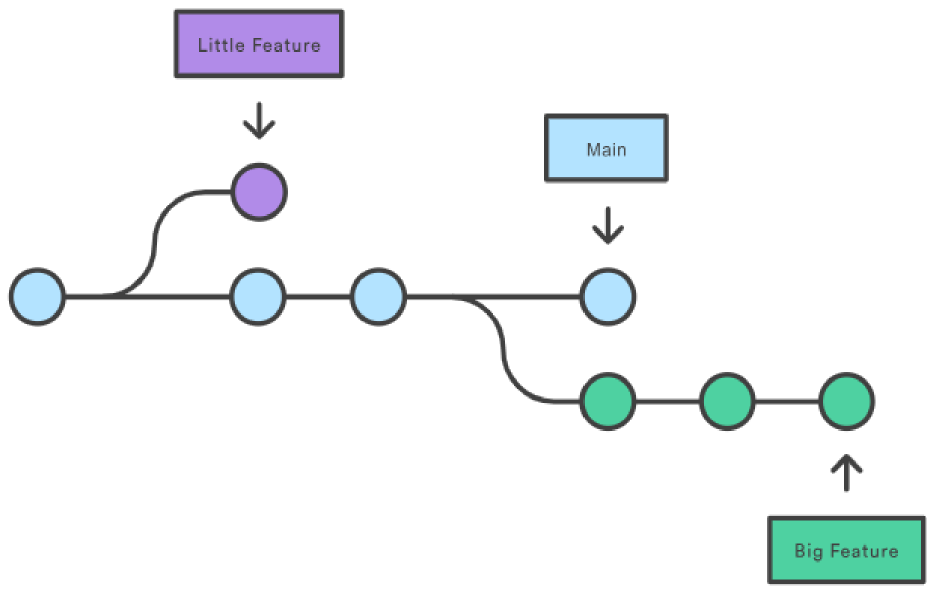
The complete pipeline theoretically looks like -

1. **Pulling Code**: Start by making sure you have the latest code from the main branch.
2. **Creating a New Branch**: If you’re working on a new feature or bug fix, create a new branch from the main branch. This branch will contain your changes.
3. **Making Changes**: Work on your code changes within your feature branch.
4. **Committing Changes**: Return to GitHub Desktop. You should see your changes listed under “Changes.” Write a descriptive commit message and click the “Commit” button to save your changes.

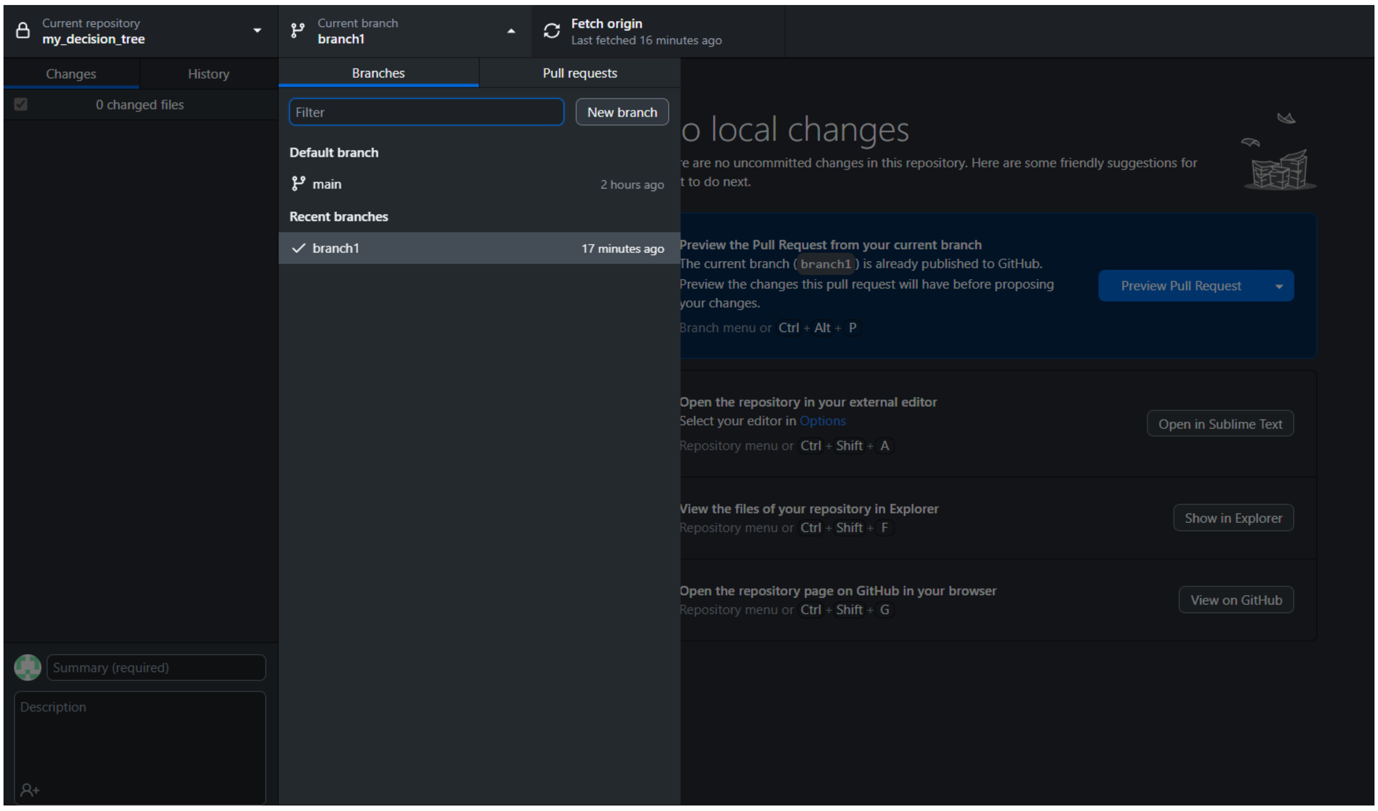

Create Feature Branch from Master Branch
'Master' Branch
Merge Feature branch onto Master branch
Commit changes to feature Branch
Submit Pull Request
Discuss the Proposed Changes

Branching and Merging

- Git branches are effectively a pointer to a snapshot of your changes.
- When you want to add a new feature or fix a bug, no matter how big or how small, you spawn a new branch to encapsulate your changes.



1. Let's create a new branch say `branch1` and make some changes in the README file. Then commit to `branch1` and push it.
2. If we go to our github repository, we can see that there is a new branch and an option to compare and pull request.



o local changes

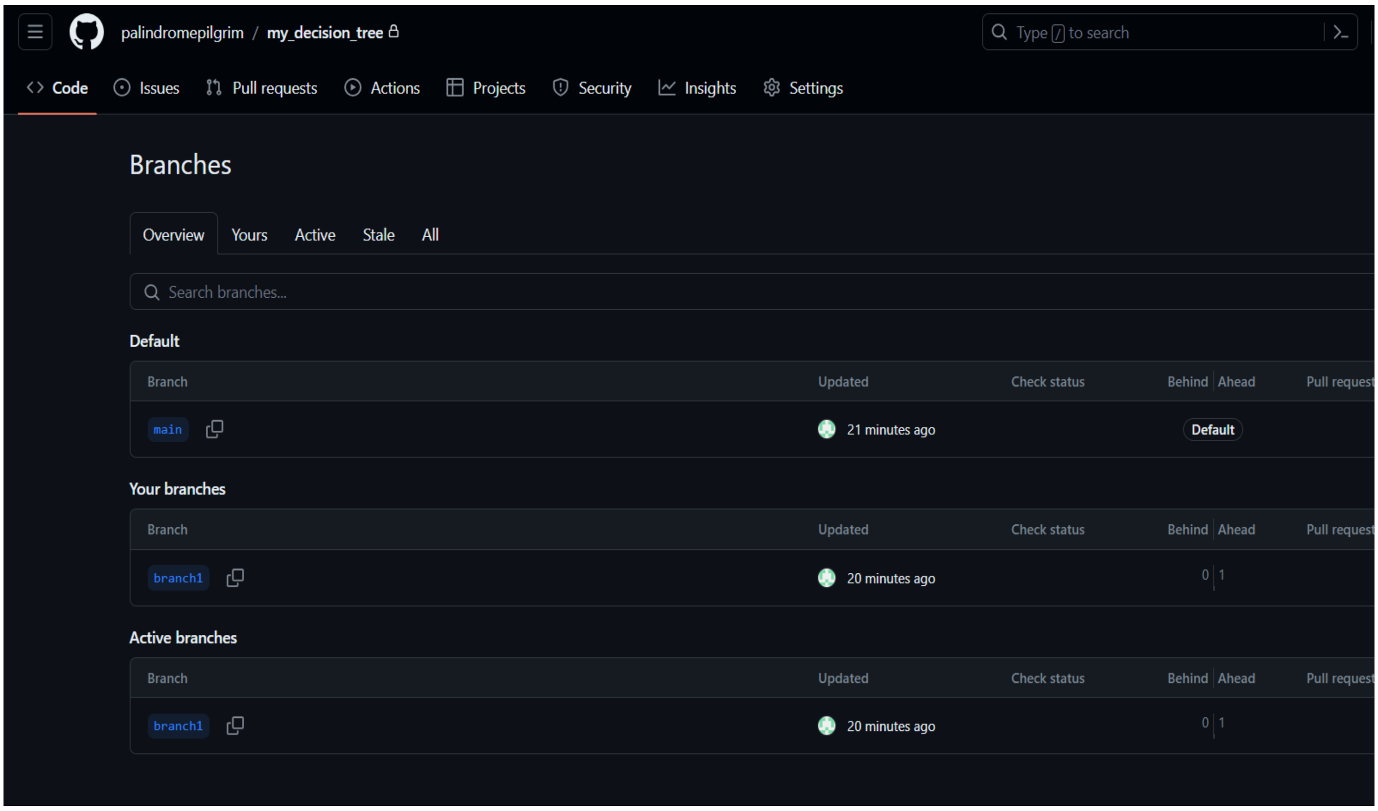
e are no uncommitted changes in this repository. Here are some friendly suggestions for t to do next.

Preview the Pull Request from your current branch
The current branch (`branch1`) is already published to GitHub. Preview the changes this pull request will have before proposing your changes.
Branch menu or Ctrl + Alt + P

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl + Shift + A

View the files of your repository in Explorer
Repository menu or Ctrl + Shift + F

Open the repository page on GitHub in your browser
Repository menu or Ctrl + Shift + G



Branches

Default

Branch	Updated	Check status	Behind | Ahead	Pull request
`main`	21 minutes ago		Default	

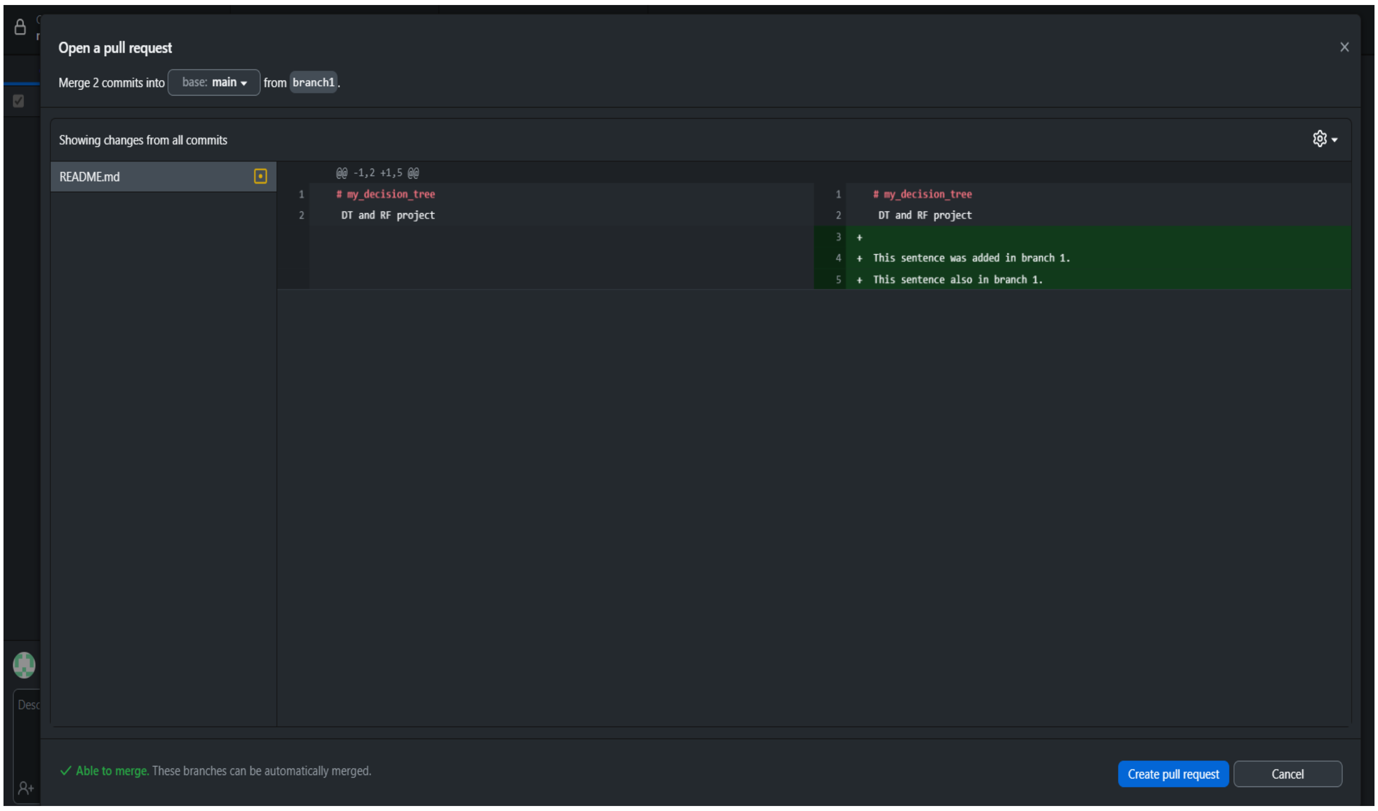
Your branches

Branch	Updated	Check status	Behind | Ahead	Pull request
`branch1`	20 minutes ago		0 | 1	

Active branches

Branch	Updated	Check status	Behind | Ahead	Pull request
`branch1`	20 minutes ago		0 | 1	

3. The first thing we want to do is create a pull request to compare the changes of `branch1` with the `main` branch.
4. Because you are the owner of the repository, you can review and merge the pull request.


Open a pull request
Merge 2 commits into base: main from branch1.
Showing changes from all commits
README.md
Able to merge. These branches can be automatically merged.
Create pull request
Cancel

```
@@ -1,2 +1,5 @@
# my_decision_tree                  # my_decision_tree
 DT and RF project                   DT and RF project
                                    +
                                    + This sentence was added in branch 1.
                                    + This sentence also in branch 1.
```

5. Once merged, you have the option to delete the `branch1` and you will see that the changes in README file are visible in the `main` branch.

Resolving Conflicts

- If while merging, there are two conflicting or different changes to same line of a file, GitHub is not sure which one to keep.
- In that case, you must manually check that to resolve the conflict. After that you can mark as resolved and merge.

Contributing to a Project

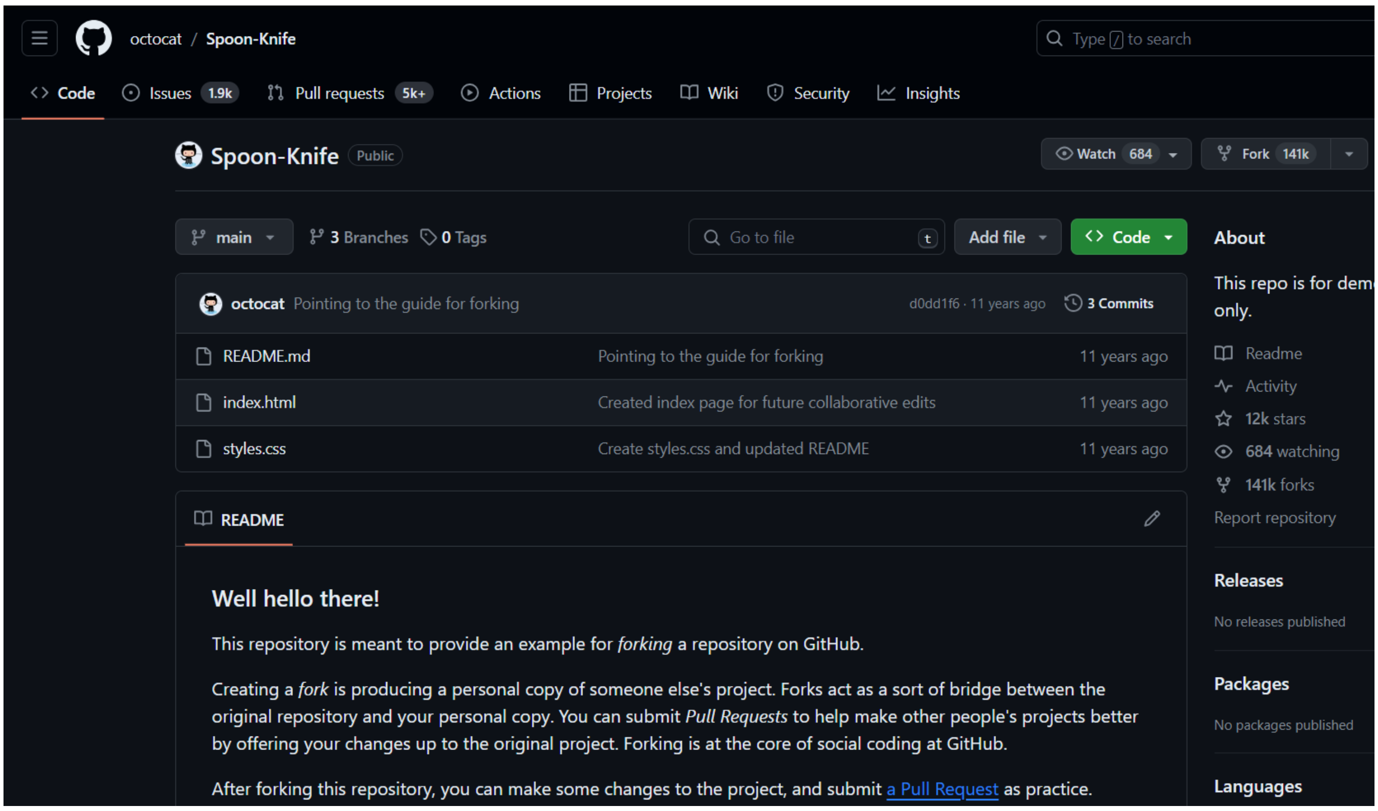
Forking

- It is a way to make a personal copy of a public or open-source repository on a platform like GitHub.
- When you fork a repository, you create an identical copy of that repository in your GitHub account.

Cloning

- You can create a local copy of any repository on GitHub that you have access to by cloning the repository.
- [Read this](#) to see how to clone a repository to GitHub Desktop.

Try forking and cloning : https://github.com/octocat/Spoon-Knife
This is a demo repository meant for trying out forking.



octocat / Spoon-Knife
Type / to search
Code | Issues 1.9k | Pull requests 5k+ | Actions | Projects | Wiki | Security | Insights

Spoon-Knife Public
Watch 684 | Fork 141k

main | 3 Branches | 0 Tags | Go to file | Add file | Code

octocat Pointing to the guide for forking d0dd1f6 · 11 years ago 3 Commits

README.md	Pointing to the guide for forking	11 years ago
index.html	Created index page for future collaborative edits	11 years ago
styles.css	Create styles.css and updated README	11 years ago

README

Well hello there!

This repository is meant to provide an example for *forking* a repository on GitHub.

Creating a *fork* is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit *Pull Requests* to help make other people's projects better by offering your changes up to the original project. Forking is at the core of social coding at GitHub.

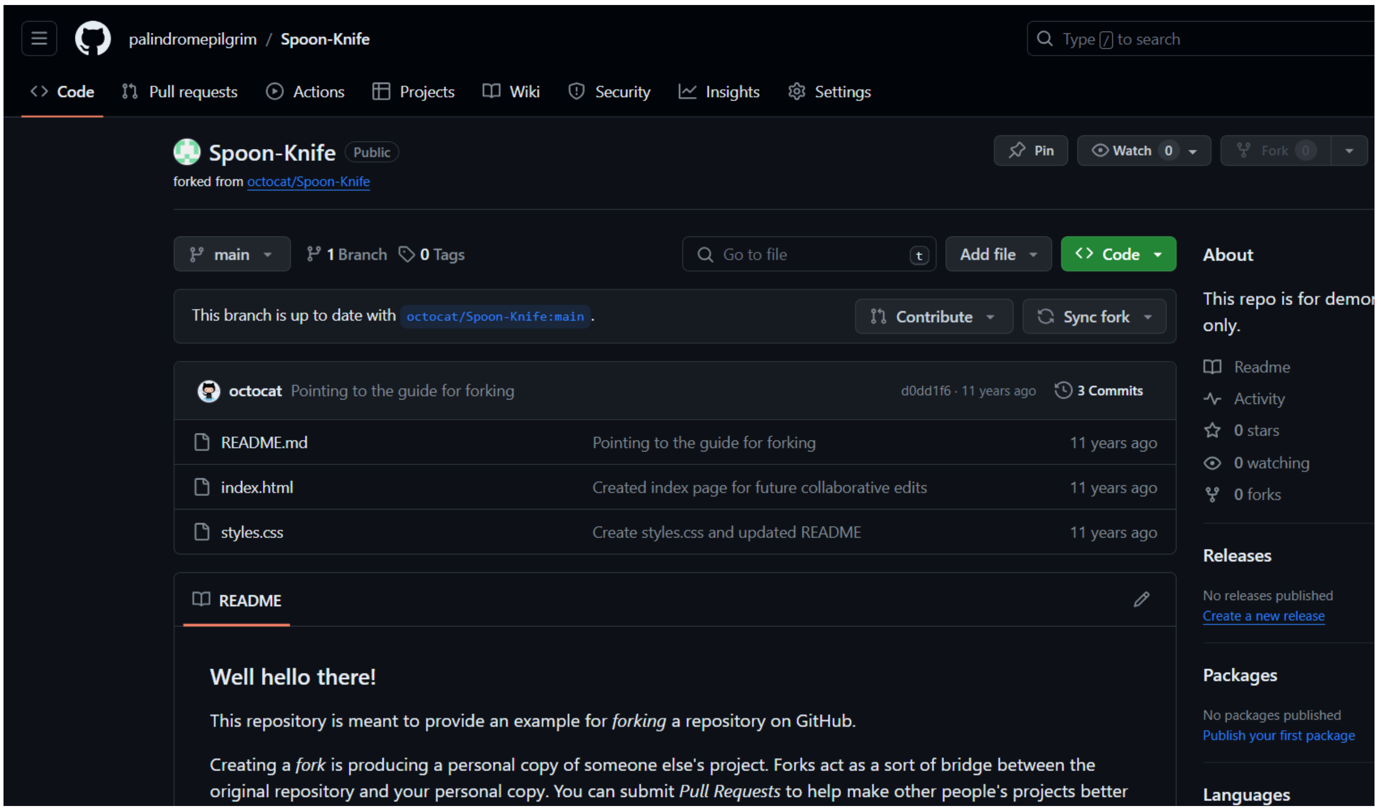
After forking this repository, you can make some changes to the project, and submit a Pull Request as practice.

About
This repo is for dem
only.
Readme
Activity
12k stars
684 watching
141k forks
Report repository

Releases
No releases published

Packages
No packages published

Languages



palindromepilgrim / Spoon-Knife
Type / to search
Code | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

Spoon-Knife Public
forked from octocat/Spoon-Knife
Pin | Watch 0 | Fork 0

main | 1 Branch | 0 Tags | Go to file | Add file | Code

This branch is up to date with `octocat/Spoon-Knife:main`.
Contribute | Sync fork

octocat Pointing to the guide for forking d0dd1f6 · 11 years ago 3 Commits

README.md	Pointing to the guide for forking	11 years ago
index.html	Created index page for future collaborative edits	11 years ago
styles.css	Create styles.css and updated README	11 years ago

README

Well hello there!

This repository is meant to provide an example for *forking* a repository on GitHub.

Creating a *fork* is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit *Pull Requests* to help make other people's projects better

About
This repo is for demo
only.
Readme
Activity
0 stars
0 watching
0 forks

Releases
No releases published
Create a new release

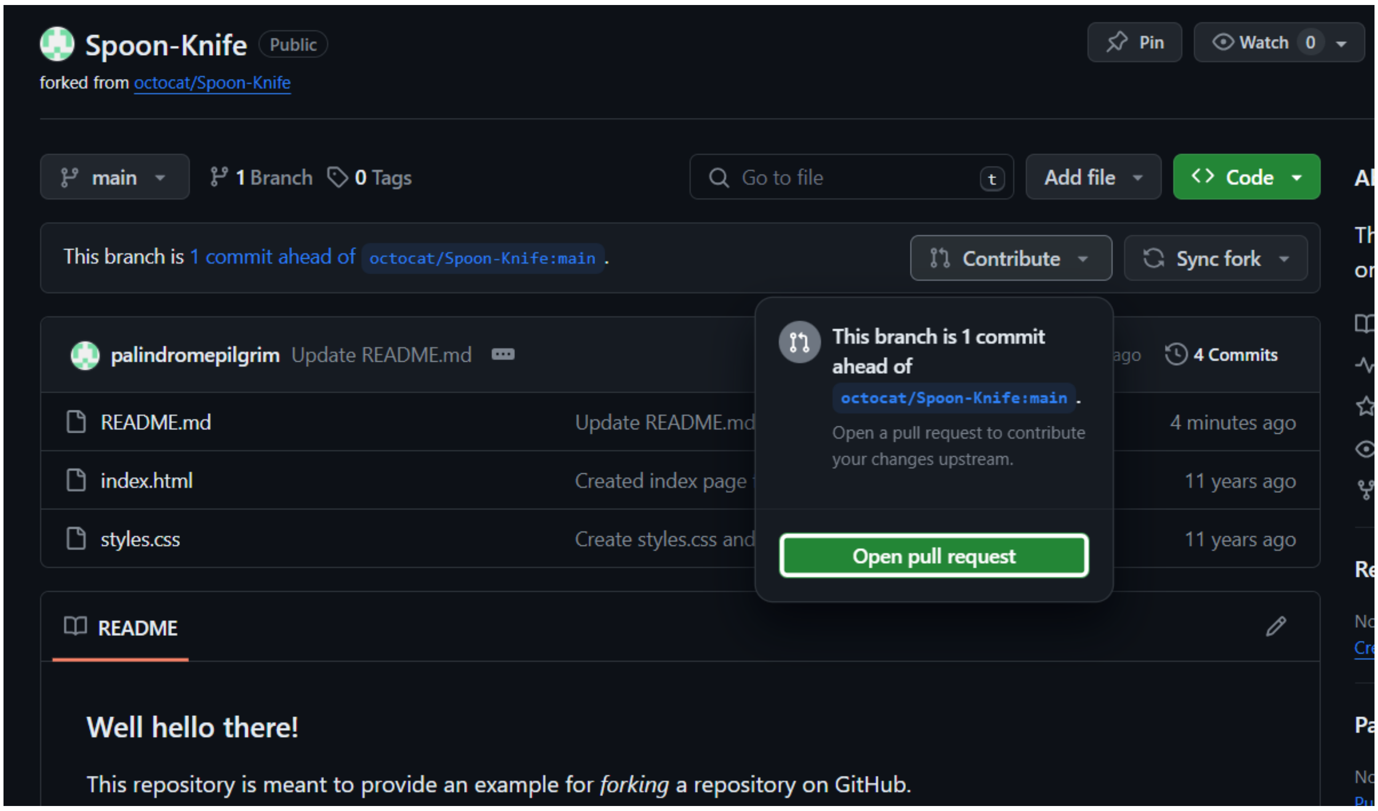
Packages
No packages published
Publish your first package

Languages

- When you clone a repository, any changes you push to GitHub will affect the original repository.
- To make changes without modifying the original project, you can create a separate copy by forking the repository.
- You can create a pull request to propose that maintainers incorporate the changes in your fork into the original upstream repository.

When you're ready to propose changes into the main project -

1. Head on over to the repository on GitHub where your project lives.
2. Click **Contribute** and then Open a pull request.
3. GitHub will bring you to a page that shows the differences between your fork and the original repository.
4. Click **Create pull request**.
5. GitHub will bring you to a page where you can enter a title and a description of your changes.
6. It's important to provide as much useful information and a rationale for why you're making this pull request in the first place.
7. Click **Create pull request**.



This branch is 1 commit ahead of octocat/Spoon-Knife:main.

palindromepilgrim Update README.md — 4 Commits

README.md	Update README.md	4 minutes ago
index.html	Created index page	11 years ago
styles.css	Create styles.css and	11 years ago

This branch is 1 commit ahead of octocat/Spoon-Knife:main.

Open a pull request to contribute your changes upstream.

Well hello there!

This repository is meant to provide an example for *forking* a repository on GitHub.

Git using CLI

Installing Git

Git comes pre-installed in some Macs and Linux-based systems, but you can always check if you have Git installed in your machine by typing `git --version` in your terminal.

https://www.atlassian.com/git/tutorials/install-git

We will start by creating a folder `github_test` and open that in the terminal.

We will add some files into that folder

- `python_file.py` : which has one statement print("python file")
- `text_file.txt` : that has single line text file

Now, these files are in our local system. How to get them on github?

Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line or see all the available options for the specific command.

- `git help <command>`
- `git <command> help`
- `git help --all`

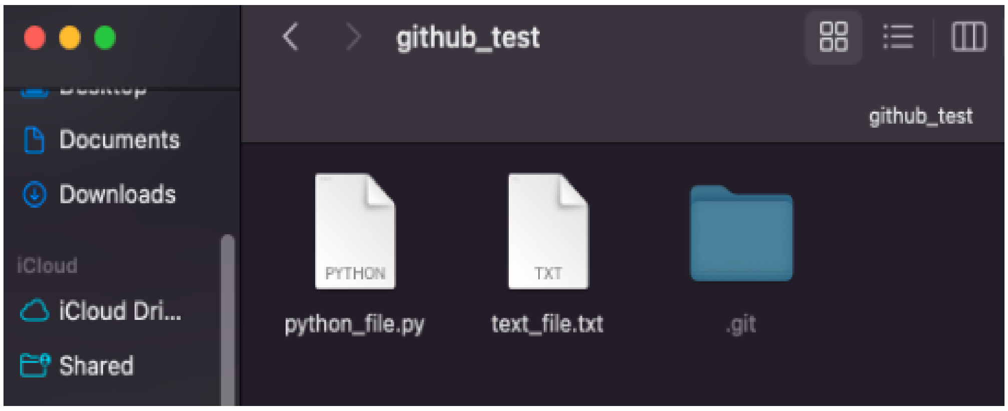
Initialize Git

Once you have navigated to the correct folder which in our case is the github_test folder with our two files, you can initialize Git on that folder:

- `git init`

The `git init` command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.

We can check what's in our folder now.



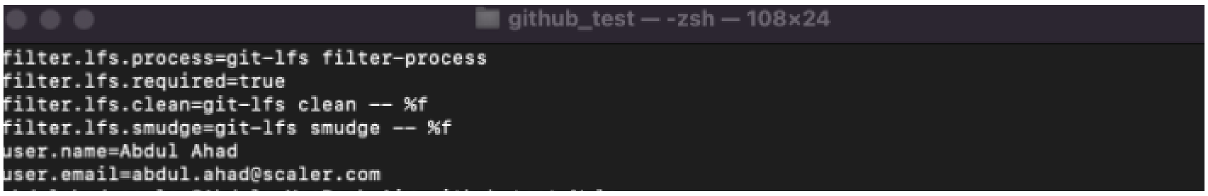
Staging Environment

- One of the core functions of Git is the concepts of the `Staging Environment`, and the `Commit`.
- As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.
- Git has three main states that your files can reside in: **modified**, **staged**, and **committed**.
  - **Modified** means that you have changed the file but have not committed it to your database yet.
  - **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.

Git Status

The `git status` command displays the state of the working directory and the staging area.

It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.

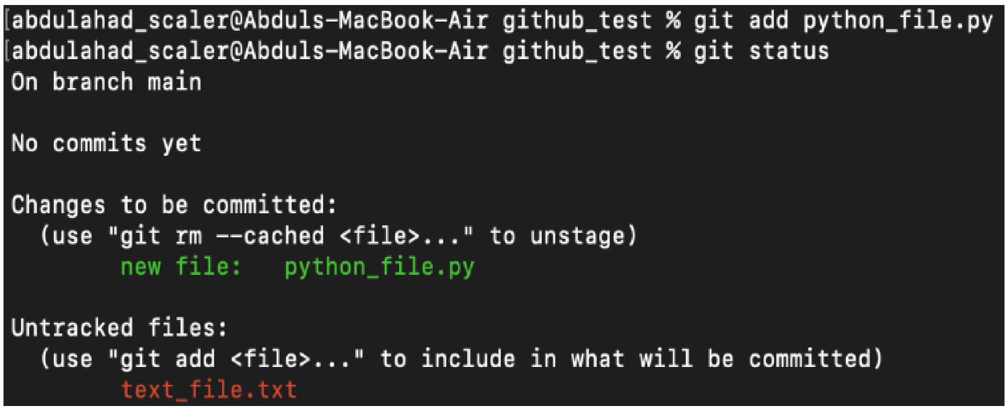


```
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
user.name=Abdul Ahad
user.email=abdul.ahad@scaler.com
```

```
abdulahad_scaler@Abduls-MacBook-Air github_test % ls
```

Git Add

- The `git add` command adds a change in the working directory to the staging area.

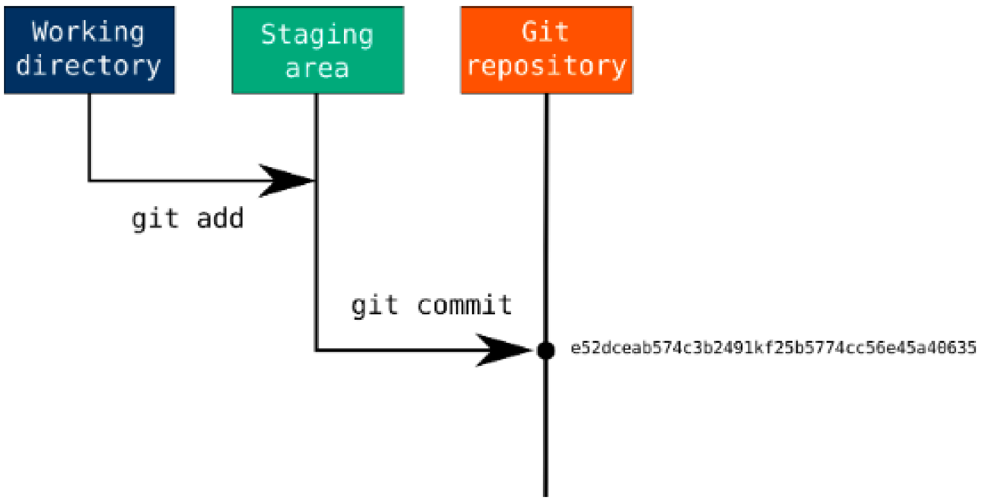


```
[abdulahad_scaler@Abduls-MacBook-Air github_test % git add python_file.py
[abdulahad_scaler@Abduls-MacBook-Air github_test % git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   python_file.py

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

- Using `--all` instead of individual file names will stage all changes (new, modified, and deleted) files.

Git Commit

- Adding commits keep track of our progress and changes as we work. Git considers each commit change point or `Save Point.



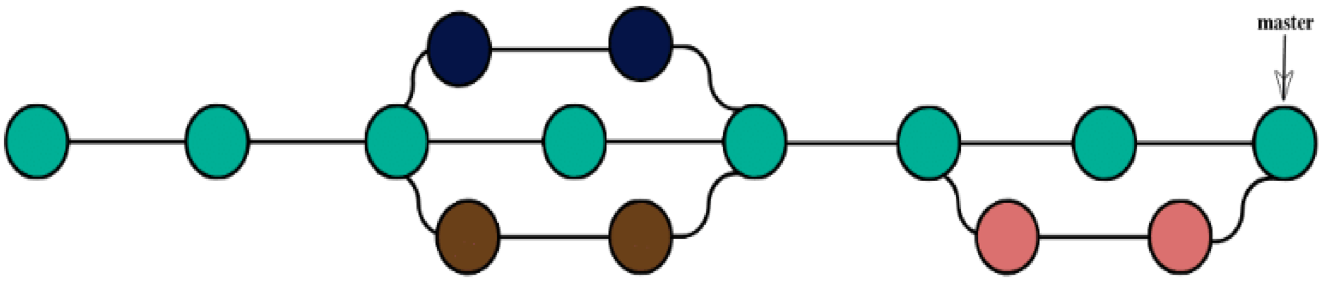
```
git commit -m "first commit"
```

has changed and when

Git Reset

- `git reset` is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

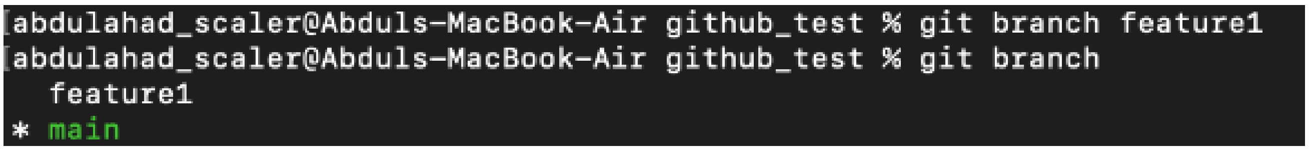
Git Branches



- One of Git's biggest advantages is its branching capabilities.
- **This allows someone to branch off of the master branch (where the production-quality code typically remains) and work on a feature or fix independently of the rest of the project.**

New Git Branch

- Now we want to add some new features to our `python_file.py` page.
- We are working in our local repository, and we do not want to disturb or possibly wreck the main project.
- So we create a new branch: `git branch feature1`



```
[abdulahad_scaler@Abduls-MacBook-Air github_test % git branch feature1
[abdulahad_scaler@Abduls-MacBook-Air github_test % git branch
  feature1
* main
```

- We can see that we have two branches now. The the `*` beside main specifies that we are currently on that branch.

Git Checkout

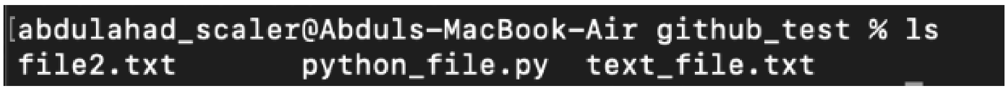
- The `git checkout` command lets you navigate between the branches created by git branch.
- Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch.
- And moves us from the current branch, to the one specified at the end of the comman.

```
git checkout feature1
```

Now let's see just how quick and easy it is to work with different branches, and how well it works.

We are currently on branch feature1. We added a file to this branch, so let's list the files in the current directory using `ls`.

```
ls
```

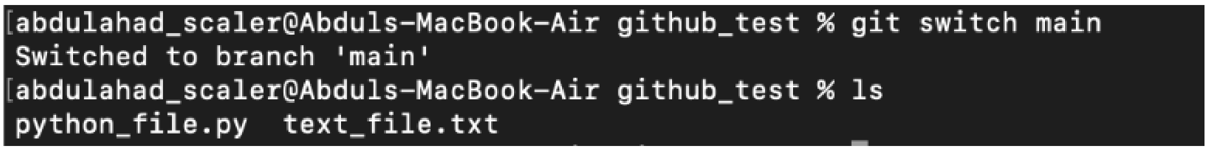


```
abdulahad_scaler@Abduls-MacBook-Air github_test % ls
file2.txt       python_file.py  text_file.txt
```

Git Switch Branch

- The git switch command is one of the latest commands being added to the list of Git commands. The git switch command was added in Git version 2.23.
- It can be seen as an alternative to its ancestor `git checkout` command.

```
git switch main
```



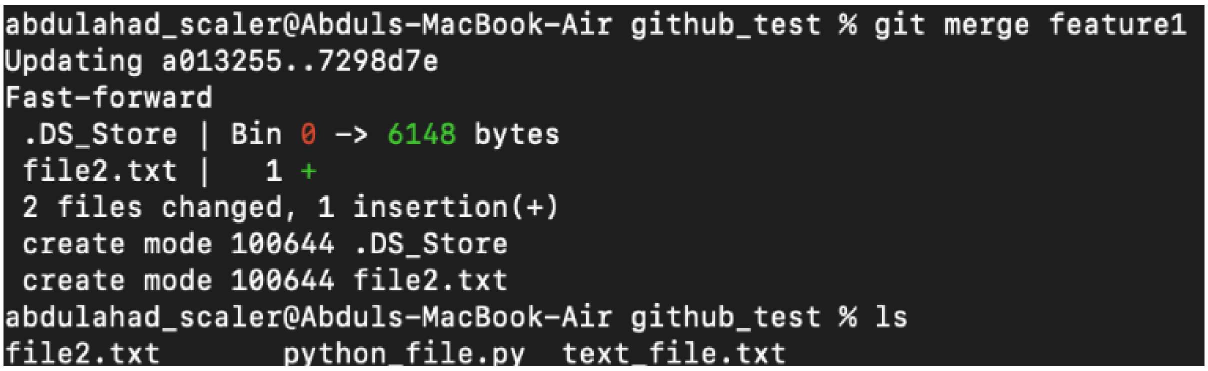
```
abdulahad_scaler@Abduls-MacBook-Air github_test % git switch main
Switched to branch 'main'
abdulahad_scaler@Abduls-MacBook-Air github_test % ls
python_file.py  text_file.txt
```

- We can see there that git manages the files. We can see that if we list files we don't see the `file2.txt`.

Git Merge Branch

- Let's say now you have worked on the feature and tested it.
- Now we need to add the code to the main branch.

```
git merge feature1
```



```
abdulahad_scaler@Abduls-MacBook-Air github_test % git merge feature1
Updating a013255..7298d7e
Fast-forward
 .DS_Store | Bin 0 -> 6148 bytes
 file2.txt |   1 +
 2 files changed, 1 insertion(+)
 create mode 100644 .DS_Store
 create mode 100644 file2.txt
abdulahad_scaler@Abduls-MacBook-Air github_test % ls
file2.txt       python_file.py  text_file.txt
```

- We can now see that when we list files in the main branch as well, we can see the new file there.

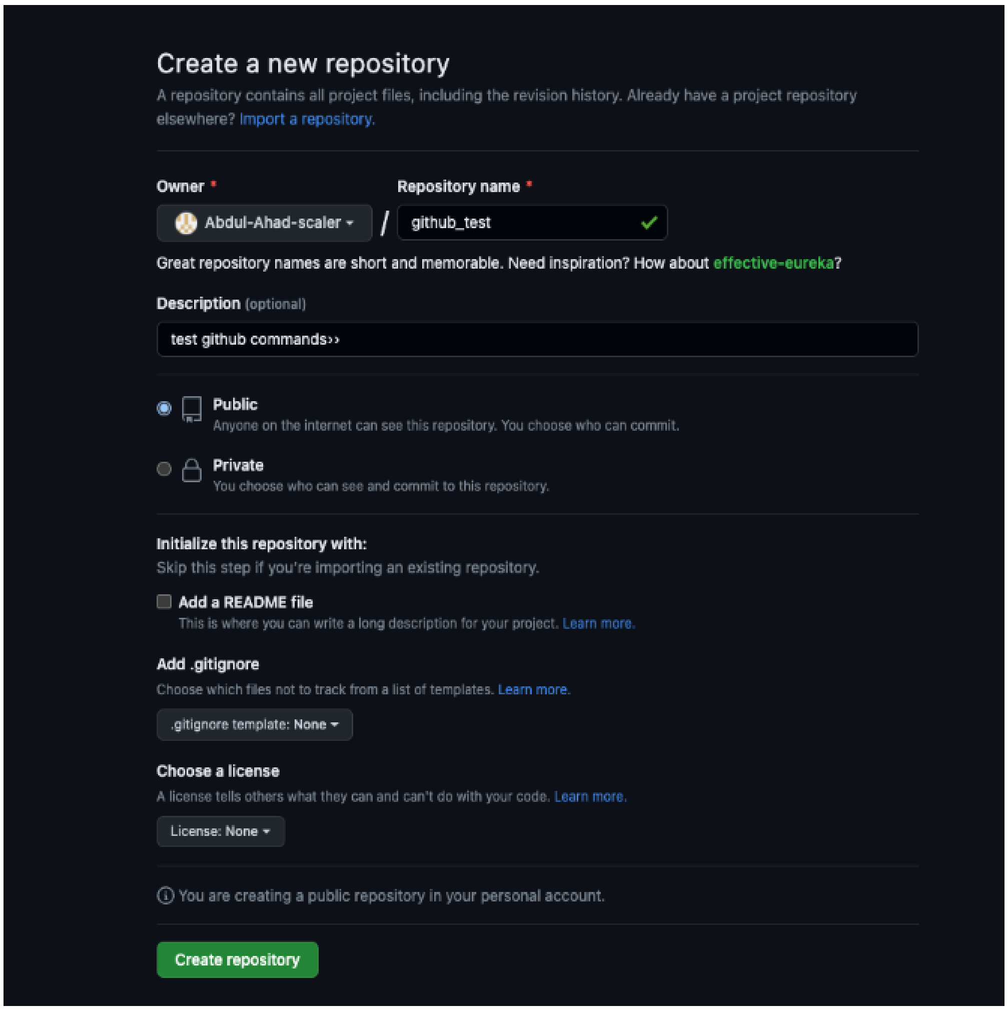
Git Stash

- `git stash` temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.
- Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

Creating a github repository -

Make sure you have a GitHub account.

- A repository contains all of your project's files and each file's revision history.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Abdul-Ahad-scaler / Repository name * github_test

Great repository names are short and memorable. Need inspiration? How about effective-eureka?

Description (optional): test github commands››

(● Public) Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

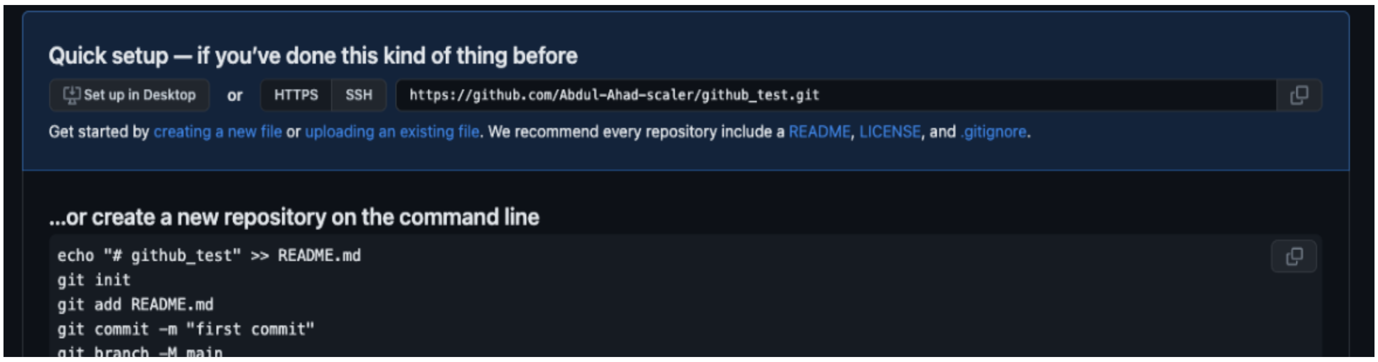
☐ Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None

You are creating a public repository in your personal account.

- Name the repo as `github_test`.
- Change it to private / public, add README and click create.



Quick setup — if you've done this kind of thing before

```
https://github.com/Abdul-Ahad-scaler/github_test.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# github_test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
```

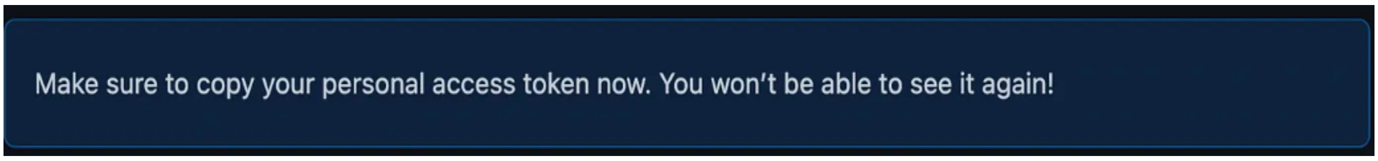
Creating Access Token on GitHub

First of all, you must create a personal Access Token on GitHub.

1. Click on your GitHub profile icon on the top right corner
2. Click Settings
3. From the menu shown on the left, click Developer Settings
4. Click Personal access tokens classic
5. Click Generate new token

6. Add a note that will help you identify the scope of the access token to be generated
7. Choose the Expiration period from the drop down menu
8. Select the scopes you want to grant the corresponding access to the generated access token
9. Finally click Generate Token

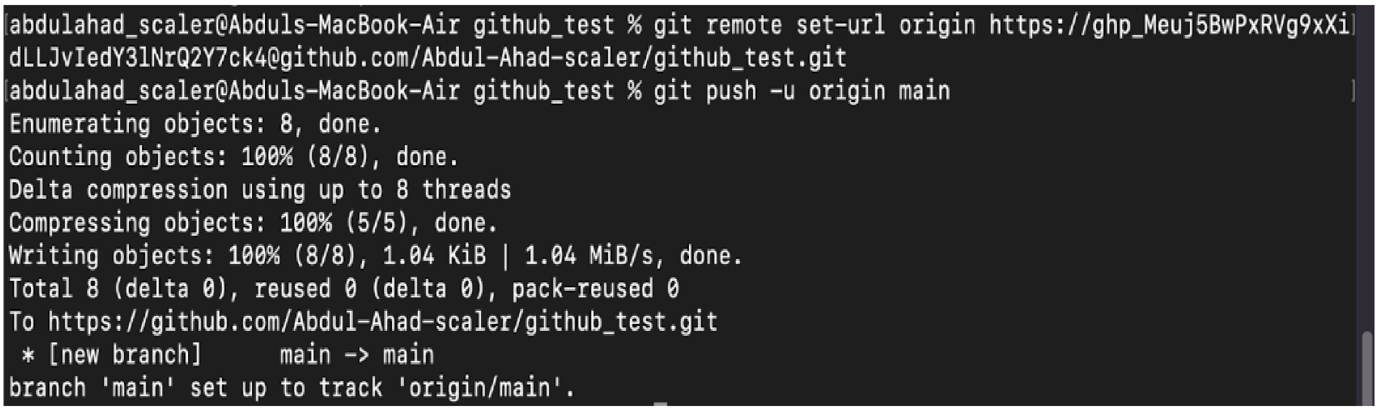
By now, you should have generated your personal access token successfully and the following message should be visible on your screen.



Make sure to copy your personal access token now. You won't be able to see it again!

To add a new remote, use the git remote add command on the terminal, in the directory your repository is stored at.

```
git remote set-url origin
https://<githubtoken>@github.com/<username>/<repositoryname>.git
```



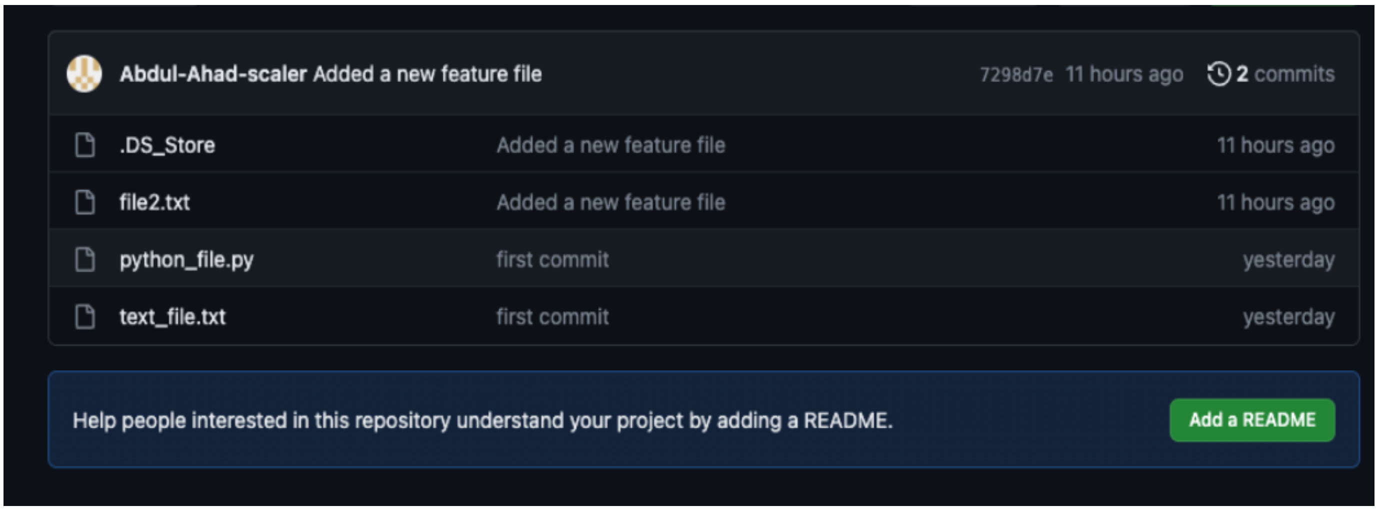
```
abdulahad_scaler@Abduls-MacBook-Air github_test % git remote set-url origin https://ghp_Meuj5BwPxRVg9xXi
dLLJvIedY3lNrQ2Y7ck4@github.com/Abdul-Ahad-scaler/github_test.git
abdulahad_scaler@Abduls-MacBook-Air github_test % git push -u origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 1.04 KiB | 1.04 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Abdul-Ahad-scaler/github_test.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Git Push & Pull

Git Push

```
git push -u origin main
```

- The `git push` command is used to upload local repository content to a remote repository.
- Pushing is how you transfer commits from your local repository to a remote repo.
- `-u` is a flag and used to set origin as the upstream remote in the git config.

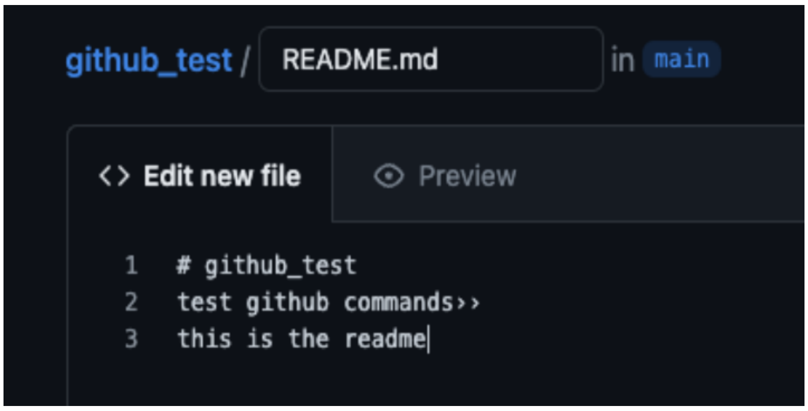


```
Abdul-Ahad-scaler Added a new feature file        7298d7e  11 hours ago   2 commits
.DS_Store          Added a new feature file                  11 hours ago
file2.txt          Added a new feature file                  11 hours ago
python_file.py     first commit                              yesterday
text_file.txt      first commit                              yesterday
```

Help people interested in this repository understand your project by adding a README.

- In Git, "origin" is a shorthand name for the remote repository that a project was originally cloned from

Add a readme

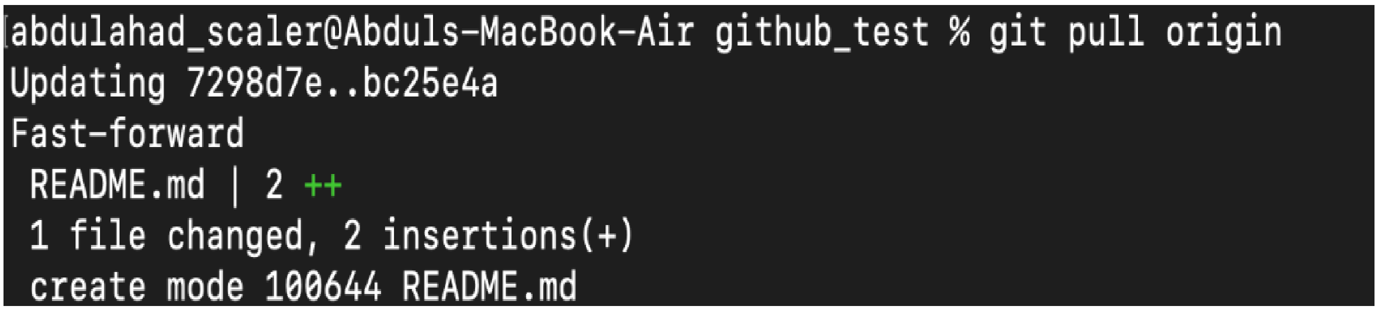


```
# github_test
test github commands>>
this is the readme
```

Git Pull

- The `git pull` command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.
- `pull` is a combination of 2 different commands:
  - Fetch
  - Merge

```
git pull origin
```



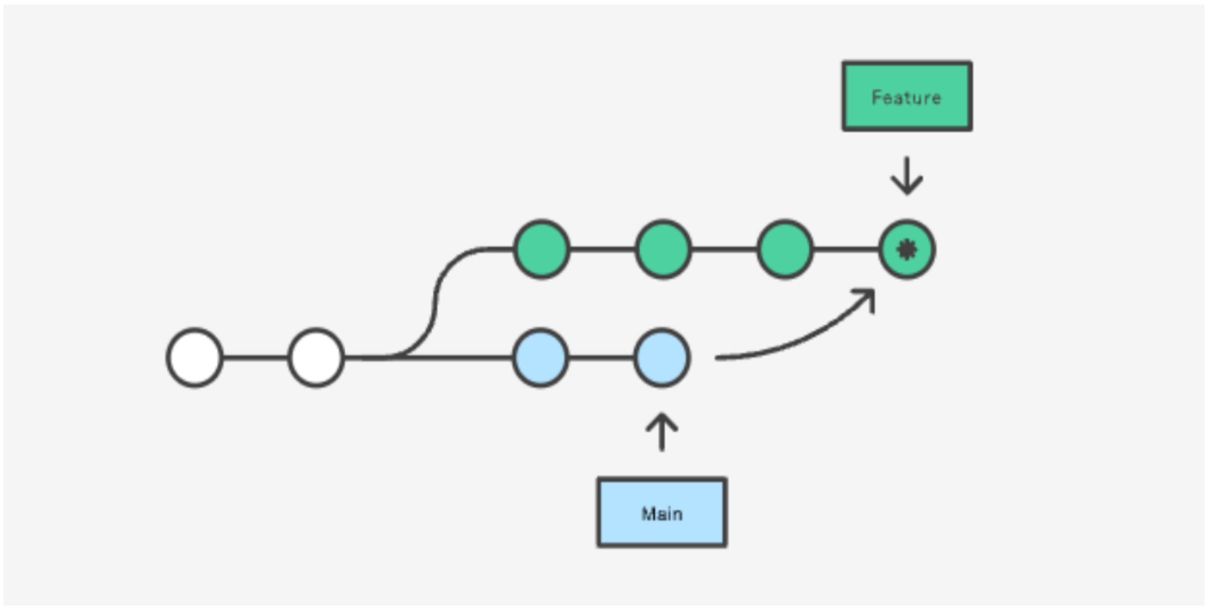
```
abdulahad_scaler@Abduls-MacBook-Air github_test % git pull origin
Updating 7298d7e..bc25e4a
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 README.md
```

Git Merge & Rebase

Git Merge

```
git merge feature main
```

- This creates a new “merge commit” in the feature branch that ties together the histories of both branches, giving you a branch structure that looks like this:



- Merging is nice because it's a non-destructive operation. The existing branches are not changed in any way.

Git Rebase

- You can rebase the feature branch onto main branch using the following commands:
  - `git checkout feature`
  - `git rebase main`
- This moves the entire feature branch to begin on the tip of the main branch, effectively incorporating all of the new commits in main.
- But, instead of using a merge commit, rebasing re-writes the project history by creating