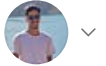


Open in app ↗

Get unlimited access



Search Medium



Fayssal Elaazouzi

Mar 23 · 5 min read · Listen



Save



Day 8: Basic Git & GitHub 👍

This is #90DaysofDevops challenge under the guidance of Shubham Londhe sir.

Day 8 TASK

check this for task:

90DaysOfDevOps/tasks.md at master · LondheShubham153/90DaysOfDevOps

This repository is a Challenge for the DevOps Community to get stronger in DevOps. This challenge starts on the 1st...

github.com





What is Git?

Git is a popular distributed version control system used for software development.

Git allows developers to **track changes** made to their code, **collaborate with others**, and **revert to earlier versions** if necessary. It uses a decentralized model, which means that each developer has their own copy of the repository and can work independently without disrupting others.

Git works by creating a repository, which is a directory that contains the project's files and the entire history of changes made to those files. Developers can make changes to the files, and Git records those changes as "**commits**" along with a message describing the changes made.

Git also provides various tools to manage the repository, such as **branching** and **merging**. Branching allows developers to create separate copies of the codebase to work on different features or fixes, while merging allows them to combine those changes back into the main codebase.

Git has become an essential tool for software development due to its powerful features, ease of use, and widespread adoption.

What is Github?

GitHub is a web-based platform that offers version control hosting for software development. It is built on top of Git and provides a user-friendly interface for developers to manage their Git repositories. With GitHub, developers can store their code, track changes, and collaborate with others on projects.

What is Version Control? How many types of version controls we have?

Version control is a system that records changes to a file or set of files over time, allowing developers to recall specific versions later. The main purpose of version control is to allow multiple people to work on a project simultaneously and to keep a history of changes that have been made to the codebase. This makes it easier to collaborate on a project, revert to previous versions of the code, and track the history of the development of a project.

There are two main types of version control systems: centralized version control systems and decentralized (or distributed) version control systems.

1. **A centralized version control system (CVCS)** uses a central server to store all the versions of a project's files. Developers "check out" files from the central server, make changes, and then "check in" the updated files. Examples of CVCS include Subversion and Perforce.
2. **A distributed version control system (DVCS)** allows developers to "clone" an entire repository, including the entire version history of the project. This means that they have a complete local copy of the repository, including all branches and past versions. Developers can work independently and then later merge their changes back into the main repository. Examples of DVCS include Git, Mercurial, and Darcs.

Why we use distributed version control over centralized version control?

1. **Better collaboration:** In a DVCS, every developer has a full copy of the repository, including the entire history of all changes. This makes it easier for developers to work together, as they don't have to constantly communicate with a central server to commit their changes or to see the changes made by others.
2. **Improved speed:** Because developers have a local copy of the repository, they can commit their changes and perform other version control actions faster, as they

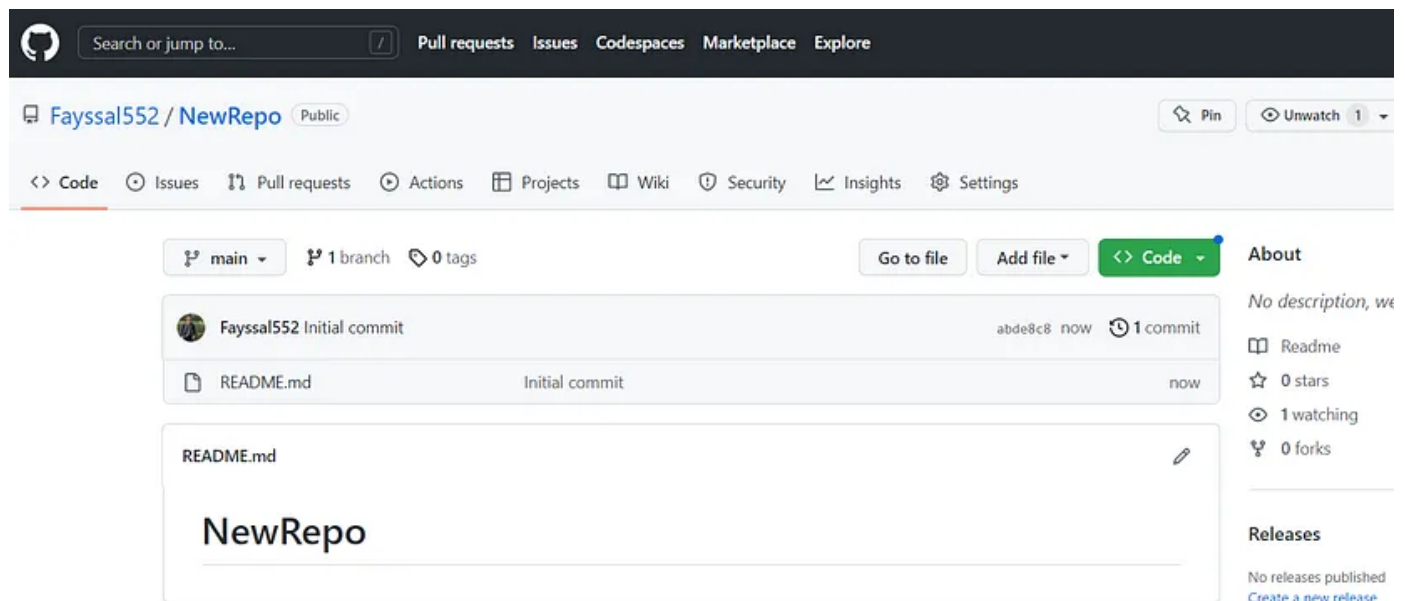
don't have to communicate with a central server.

3. **Greater flexibility:** With a DVCS, developers can work offline and commit their changes later when they do have an internet connection. They can also choose to share their changes with only a subset of the team, rather than pushing all of their changes to a central server.
4. **Enhanced security:** In a DVCS, the repository history is stored on multiple servers and computers, which makes it more resistant to data loss. If the central server in a DVCS goes down or the repository becomes corrupted, it can be difficult to recover the lost data.

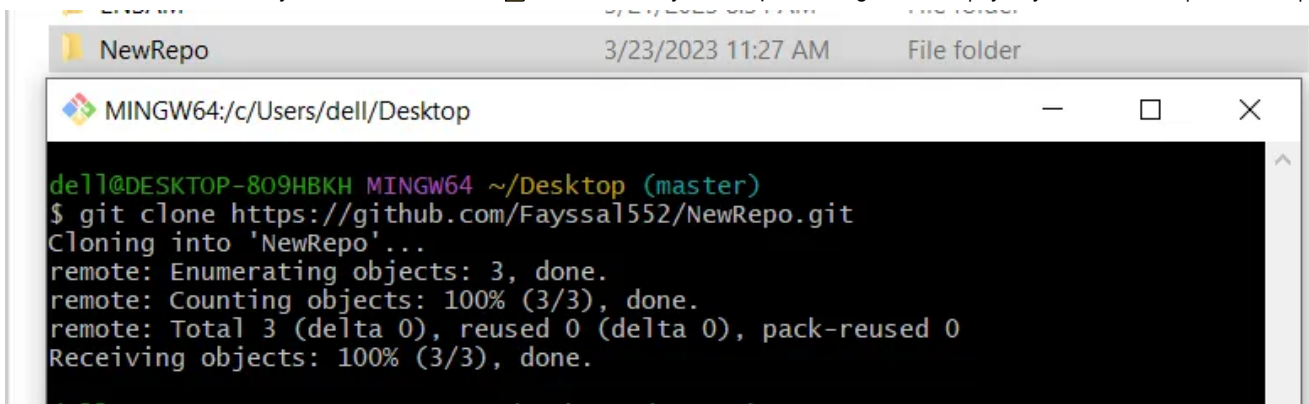
Overall, the decentralized nature of a DVCS allows for greater collaboration, flexibility, and security, making it a popular choice for many teams.

Task:

1. Create a new repository on GitHub and clone it to your local machine



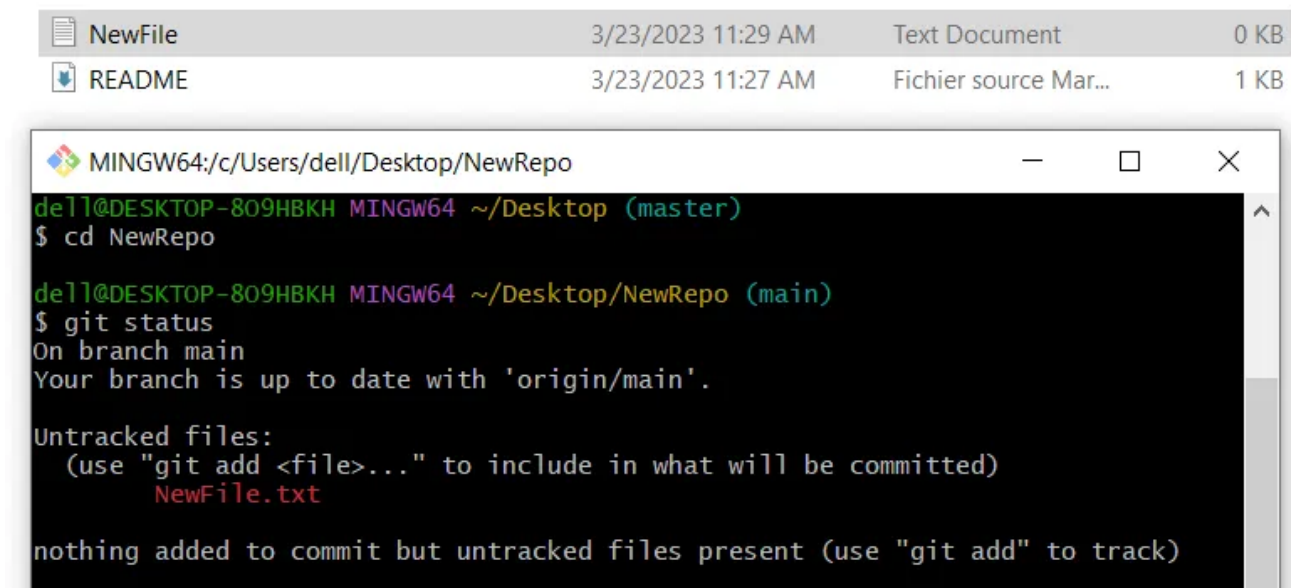
Now we will clone it on the local.



```
MINGW64:/c:/Users/dell/Desktop
dell@DESKTOP-809HBKH MINGW64 ~/Desktop (master)
$ git clone https://github.com/Fayssal552/NewRepo.git
Cloning into 'NewRepo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

2. Make some changes to a file in the repository and commit them to the repository using Git

I did add a new file on the local. Now check status of the repo.



NewFile	3/23/2023 11:29 AM	Text Document	0 KB
README	3/23/2023 11:27 AM	Fichier source Mar...	1 KB

```
MINGW64:/c:/Users/dell/Desktop/NewRepo
dell@DESKTOP-809HBKH MINGW64 ~/Desktop (master)
$ cd NewRepo

dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

Now we will use the git add command

```
dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git add .

dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   NewFile.txt
```

The git add command **adds a change in the working directory to the staging area**. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way — changes are not actually recorded until you run git commit .

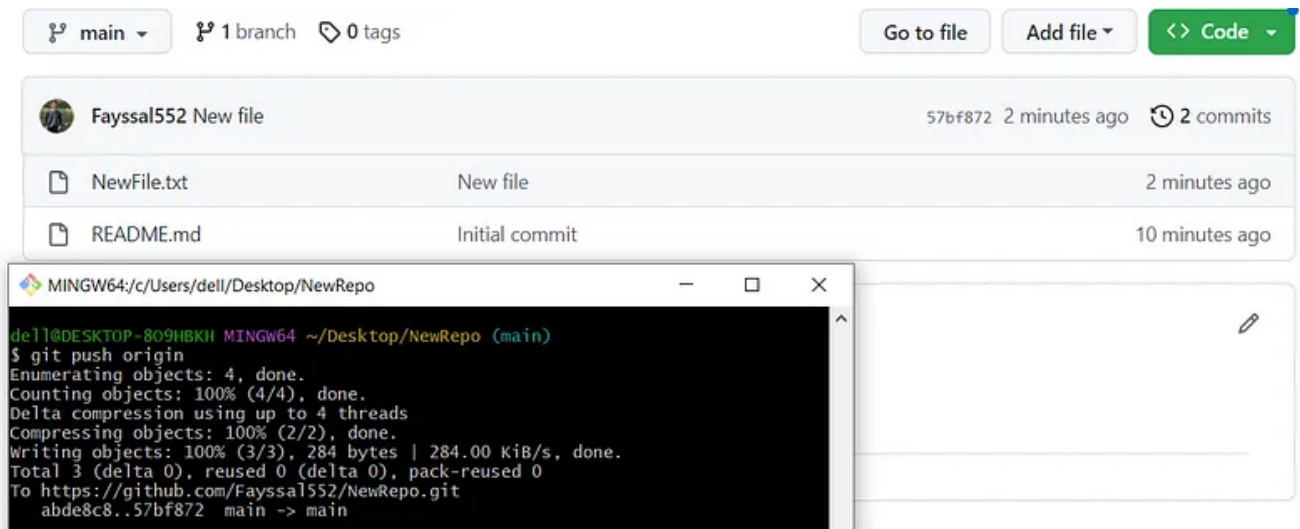
Now we will use the git commit command to save your changes to the local repository and add meaningful message.

```
dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git commit -m "New file"
[main 57bf872] New file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 NewFile.txt

dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

3. Push the changes back to the repository on GitHub



The screenshot shows the GitHub web interface for a repository named 'Fayssal552 New file'. The repository has 1 branch and 0 tags. The commit history shows two commits: 'NewFile.txt' (2 minutes ago) and 'README.md' (10 minutes ago). Below the repository view, a terminal window shows the output of the 'git push origin' command, indicating a successful push to the 'main' branch on GitHub.

```
dell@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git push origin
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Fayssal552/NewRepo.git
abde8c8..57bf872 main -> main
```

Deleting the file from the local and restoring the same.

The “git restore” command is used to restore changes in a Git repository to a previous state


```
de11@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

no changes added to commit (use "git add" and/or "git commit -a")

de11@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ git restore NewFile.txt

de11@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ sls
bash: sls: command not found

de11@DESKTOP-809HBKH MINGW64 ~/Desktop/NewRepo (main)
$ ls
NewFile.txt  README.md
```

If this post was helpful, please do follow and click the clap 🙌 button below to show your support 😊

_ Thank you for reading ❤️

_Fayssal 🙌

Git

Linux

AWS

DevOps

Docker