

DevOps Lab

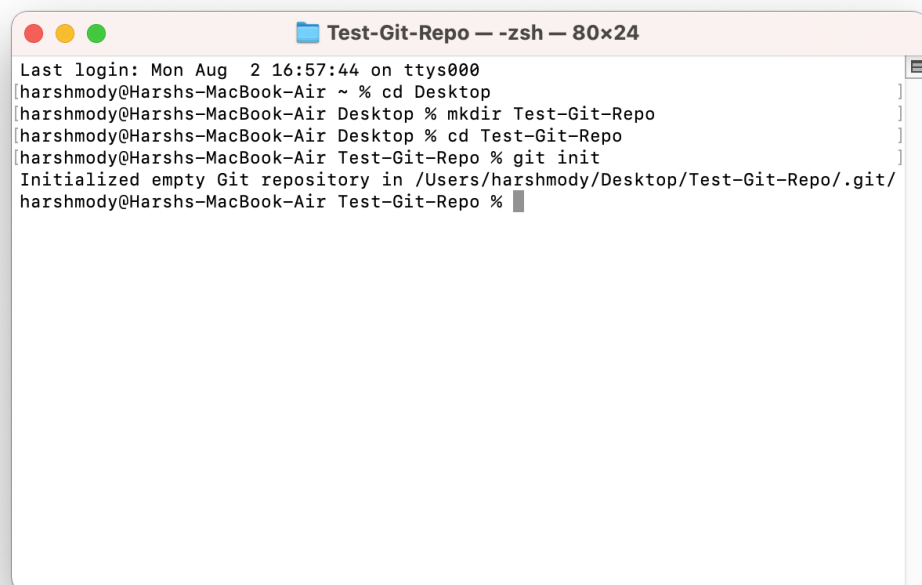
Assignment 3:

Aim: To understand various commands in Git and use GitHub to create a remote copy of our local code.

Commands & Their Execution:

Creating a local repository:

- i) Initialize the repository using the **git init** command.



```
Test-Git-Repo — zsh — 80x24
Last login: Mon Aug  2 16:57:44 on ttys000
harshmody@Harshs-MacBook-Air ~ % cd Desktop
harshmody@Harshs-MacBook-Air Desktop % mkdir Test-Git-Repo
harshmody@Harshs-MacBook-Air Desktop % cd Test-Git-Repo
harshmody@Harshs-MacBook-Air Test-Git-Repo % git init
Initialized empty Git repository in /Users/harshmody/Desktop/Test-Git-Repo/.git/
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

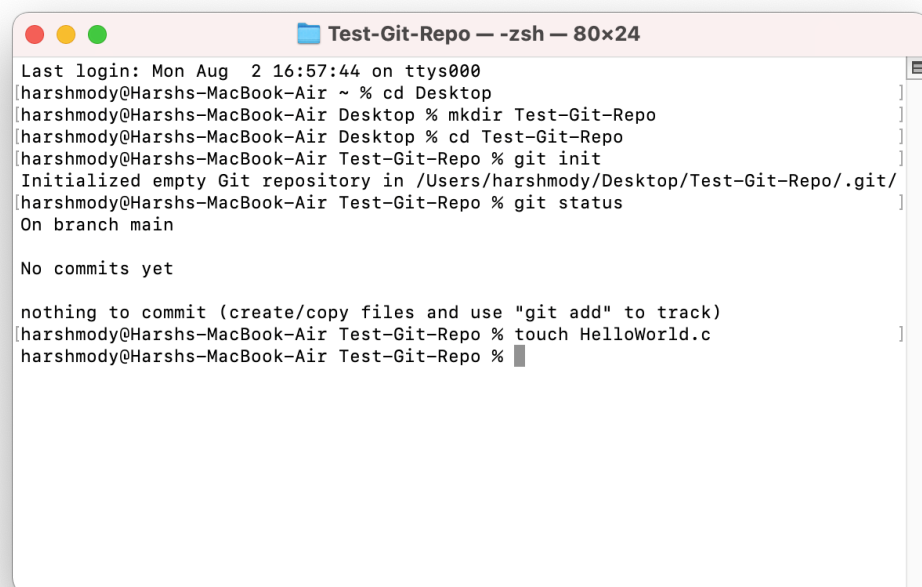
- ii) The **git status** command is used to know the status of files in the current working directory.

There are 4 basic types of files in any repository which git status command tells us about:

- a) **Untracked Files:** These are newly created files in the repository folder which are not yet added to the repository.
- b) **Added Files:** All files which are not ignored by git using .gitignore file are a part of this group. These are called the staged files.

- c) **Modified Files:** These are files which are already added to git repository but have been modified since and are so are not in the latest version of the repository.
- d) **.gitignore:** This file usually tell repository to ignore files of a certain pattern. For e.g. JetBrains IDE often create .idea files, etc. which do not need to be part of our code repository.

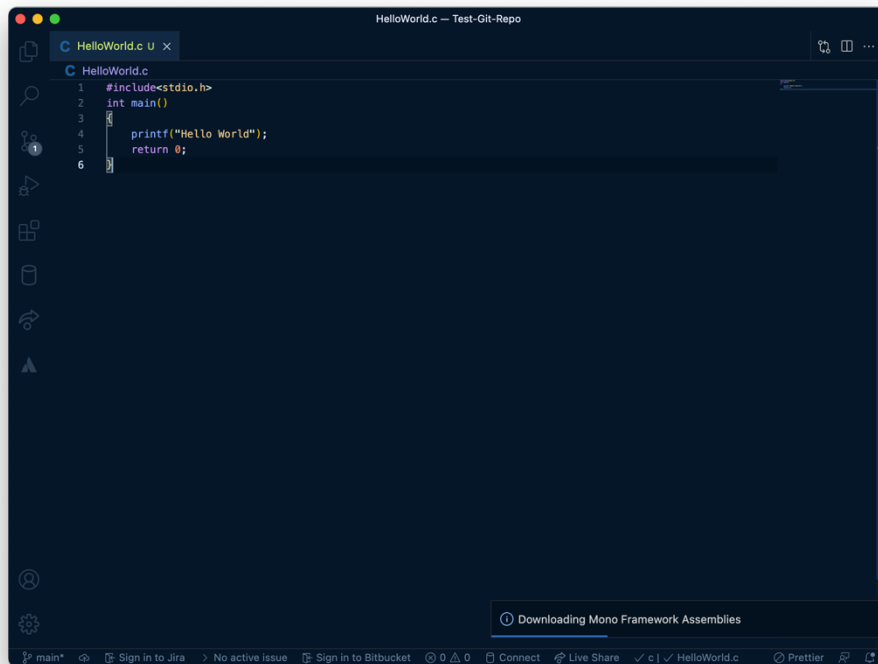
Using touch command we create a new File and Using VS Code, we add our code to the file

A terminal window titled "Test-Git-Repo — zsh — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the following commands and output:

```
Last login: Mon Aug 2 16:57:44 on ttys000
harshmody@Harshs-MacBook-Air ~ % cd Desktop
harshmody@Harshs-MacBook-Air Desktop % mkdir Test-Git-Repo
harshmody@Harshs-MacBook-Air Desktop % cd Test-Git-Repo
harshmody@Harshs-MacBook-Air Test-Git-Repo % git init
Initialized empty Git repository in /Users/harshmody/Desktop/Test-Git-Repo/.git/
harshmody@Harshs-MacBook-Air Test-Git-Repo % git status
On branch main

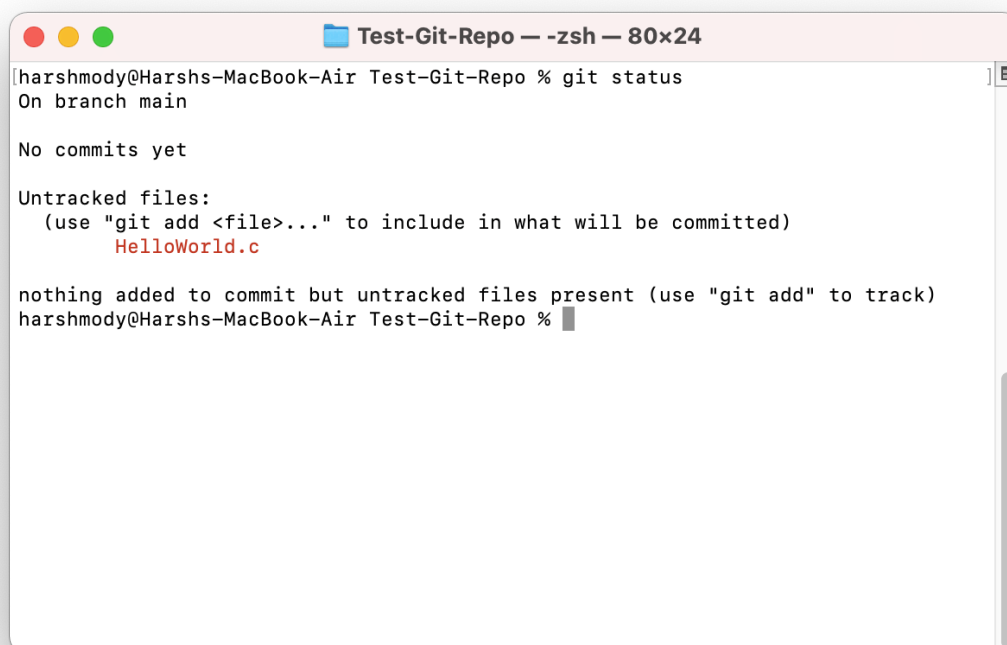
No commits yet

nothing to commit (create/copy files and use "git add" to track)
harshmody@Harshs-MacBook-Air Test-Git-Repo % touch HelloWorld.c
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```



```
1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello World!");
5     return 0;
6 }
```

Now, since we created the file but we didn't add it to our repository, git status shows that's it's a untracked file.



```
Test-Git-Repo — -zsh — 80x24
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git status
On branch main

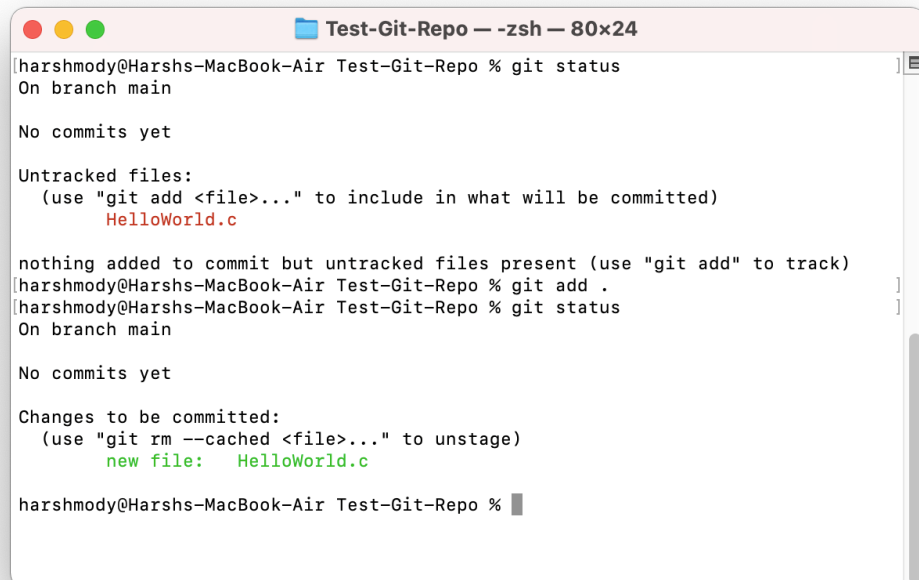
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

- iii) The **git add** command is used to add files in current working directory to our repository.

The **git add .** command adds all the untracked files in current working directory to the repository.

A terminal window titled "Test-Git-Repo — -zsh — 80x24" showing the execution of git commands. The user runs 'git status' and sees that there are untracked files, including 'HelloWorld.c'. Then, the user runs 'git add .' to stage all untracked files. Finally, the user runs 'git status' again, and the output shows that 'HelloWorld.c' is now a new file staged for commit.

```
harshmody@Harshs-MacBook-Air Test-Git-Repo % git status
On branch main

No commits yet

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


nothing added to commit but untracked files present (use "git add" to track)
harshmody@Harshs-MacBook-Air Test-Git-Repo % git add .
harshmody@Harshs-MacBook-Air Test-Git-Repo % git status
On branch main

No commits yet

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

harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

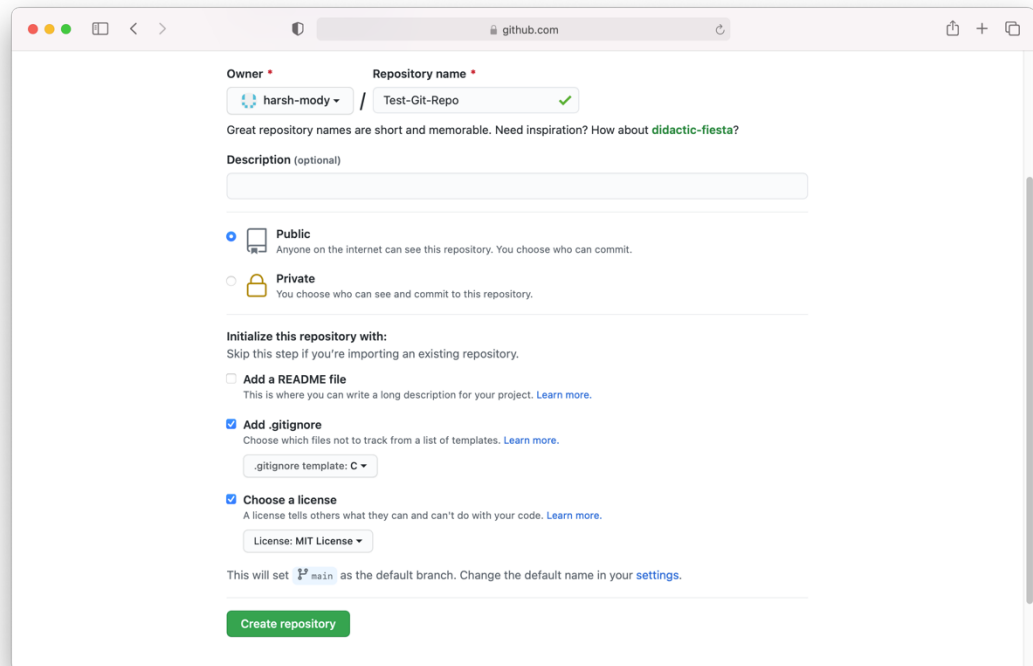
- iv) Now, our added files are in the staging area and to make these changes a permanent checkpoint to where we can revert our repository in case we find bugs in latest version of our code, we use the **git commit** command.

A terminal window titled "Test-Git-Repo — -zsh — 80x24" is shown. It contains the following text:

```
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git commit -m "Add HelloWorld.c"]
[main (root-commit) fde063d] Add HelloWorld.c
 1 file changed, 6 insertions(+)
 create mode 100644 HelloWorld.c
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git status]
On branch main
nothing to commit, working tree clean
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

As we see, after committing changes, our working tree is clean and now we have a checkpoint where we can revert to in-case we need to.

- v) Now, adding these changes to a remote hosted repository on GitHub, we create a new repository.



vi) The **git pull** command is used to pull the changes from the remote repository.

But, before we push, we need to pull changes from initial commit on remote. So, we need to use **git pull** command. But before that we need to set a default

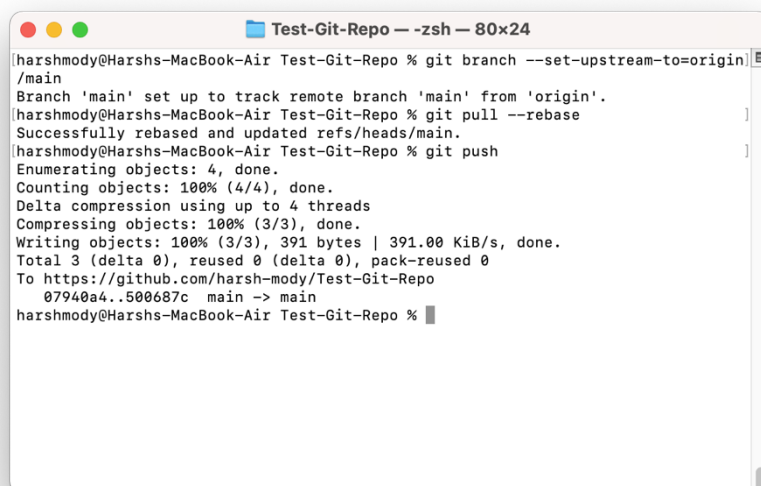
branch to pull from remote repository. So, we use the **git branch --set-upstreamto=origin/main** command.

Here, since I had created an initial commit on remote repository, and the local as well but the version history of the two repository's did not match, we use the **git pull --rebase** command.

A terminal window titled "Test-Git-Repo — zsh — 80x24" showing the execution of two git commands. The first command is "git branch --set-upstream-to=origin/main", which sets the local 'main' branch to track the remote 'main' branch. The second command is "git pull --rebase", which successfully rebases the local 'main' branch onto the remote 'main' branch.

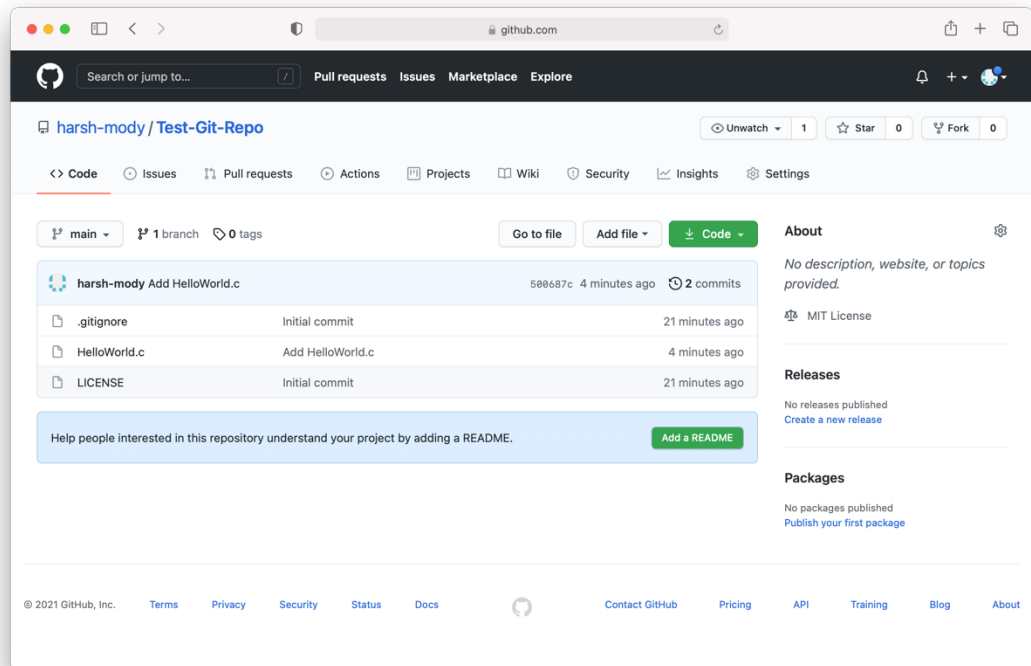
```
harshmody@Harshs-MacBook-Air Test-Git-Repo % git branch --set-upstream-to=origin/main
Branch 'main' set up to track remote branch 'main' from 'origin'.
harshmody@Harshs-MacBook-Air Test-Git-Repo % git pull --rebase
Successfully rebased and updated refs/heads/main.
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

- vii) The **git push** command is used to push the changes from local repository to remote repository

A terminal window titled "Test-Git-Repo — zsh — 80x24" showing the execution of the "git push" command. The output shows the process of enumerating, counting, compressing, and writing objects to the remote repository. The push is successful, and the local 'main' branch is updated to match the remote 'main' branch.

```
harshmody@Harshs-MacBook-Air Test-Git-Repo % git branch --set-upstream-to=origin/main
Branch 'main' set up to track remote branch 'main' from 'origin'.
harshmody@Harshs-MacBook-Air Test-Git-Repo % git pull --rebase
Successfully rebased and updated refs/heads/main.
harshmody@Harshs-MacBook-Air Test-Git-Repo % git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 391 bytes | 391.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/harsh-mody/Test-Git-Repo
07940a4..500687c main -> main
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```

As we can see, our HelloWorld.c file was committed to our remote GitHub repository.

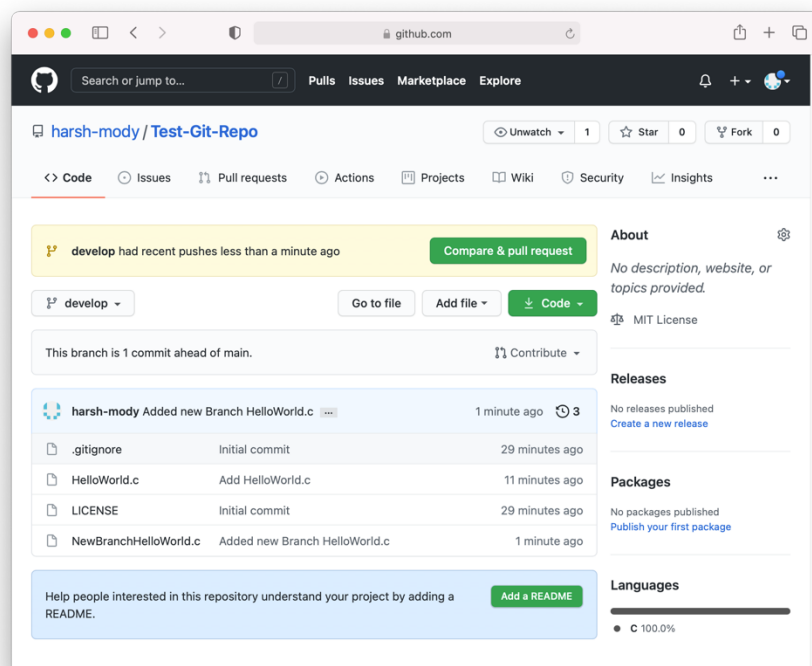


viii) Now, creating a new branch using **git branch develop**, we can create a new branch. The **git checkout develop** command is used to navigate to the newly created branch.

```
Test-Git-Repo — -zsh — 80x24
harshmodiy@Harshs-MacBook-Air Test-Git-Repo % git branch develop
harshmodiy@Harshs-MacBook-Air Test-Git-Repo % git checkout develop
Switched to branch 'develop'
harshmodiy@Harshs-MacBook-Air Test-Git-Repo %
```

Using similar add, push commands, we create a new file and push it to new branch in remote repository. Thus getting our new branch to remote repository as shown below.

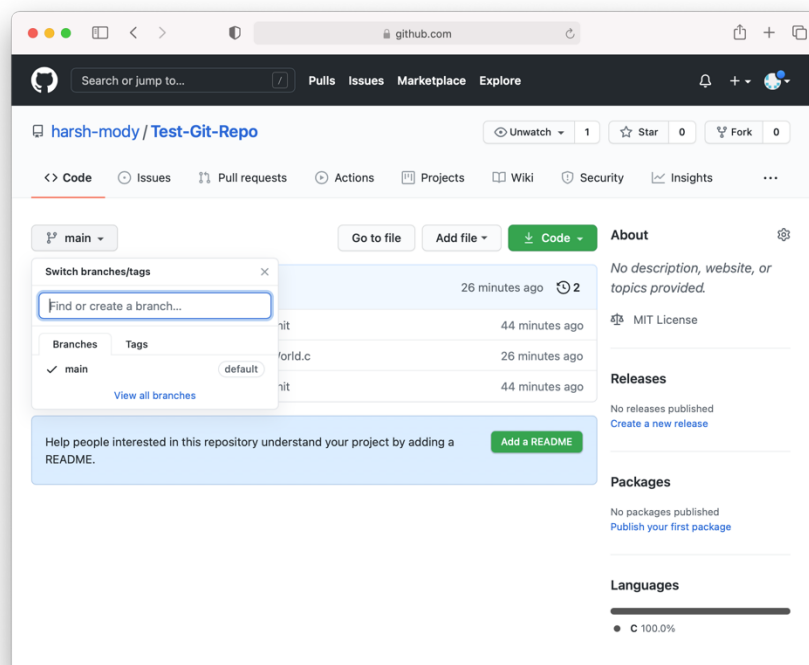

```
Test-Git-Repo — zsh — 80x24
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git checkout develop
Switched to branch 'develop'
[harshmody@Harshs-MacBook-Air Test-Git-Repo % touch NewBranchHelloWorld.c
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git add .
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git commit -m "Added new Branch HelloWorld.c"
[develop e6579cf] Added new Branch HelloWorld.c
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 NewBranchHelloWorld.c
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git push origin develop
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), 296 bytes | 296.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/harsh-mody/Test-Git-Repo/pull/new/develop
remote:
To https://github.com/harsh-mody/Test-Git-Repo
 * [new branch]      develop -> develop
[harshmody@Harshs-MacBook-Air Test-Git-Repo %
```



Now, we delete the created branch and push these new changes to remote repository.

To delete develop from both remote and local repository, use **git push origin --delete develop** to delete it from remote and then **git checkout main** to checkout of main and then **git branch -D develop** to delete the local copy of the branch.

```
Test-Git-Repo — zsh — 80x24
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git push origin --delete develop
To https://github.com/harsh-mody/Test-Git-Repo
- [deleted]          develop
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git branch
* develop
  main
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git branch -D develop
Deleted branch develop (was e6579cf).
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git branch
* main
harshmody@Harshs-MacBook-Air Test-Git-Repo %
```



- ix) The **git clone** command. This command is used to pull/bring latest changes from a remote repository to local repository we just created on GitHub using the command **git clone "https://github.com/harsh-mody/Test-Git-Repo.git"**.

As we see, all the changes added to remote repository were pulled correctly.



```
Test-Git-Repo — zsh — 80x24
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git clone "https://github.com/harsh-
-mody/Test-Git-Repo.git"
Cloning into 'Test-Git-Repo'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 1), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), done.
[harshmody@Harshs-MacBook-Air Test-Git-Repo % cd Test-Git-Repo
[harshmody@Harshs-MacBook-Air Test-Git-Repo % ls
HelloWorld.c    LICENSE
[harshmody@Harshs-MacBook-Air Test-Git-Repo % git log
commit 500687cc7caee21040a3cdc0aa6a22d0cb27282 (HEAD -> main, origin/main, orig
in/HEAD)
Author: Harsh Mody <harsh.mody@icloud.com>
Date:   Mon Aug 2 17:27:26 2021 +0530

    Add HelloWorld.c

commit 07940a48140cba5a4399b38bf6255fca3b02951d
Author: Harsh Mody <56110469+harsh-mody@users.noreply.github.com>
Date:   Mon Aug 2 17:31:55 2021 +0530
```

Conclusion: Thus, successfully understood various basic commands in Version Control Systems Like Git and also understood the use of remote repository hosting services like GitHub .