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

```
echo "# demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/iamnikhil096/demo.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/iamnikhil096/demo.git
git branch -M main
git push -u origin main
```

- **“git rm -r --cached . “ : this removes all files from the remote repo, we can specify specific File names as well in place of (“.”)**
- **“git rm” vs “git rm --cached” :** The git rm command removes a file from a Git repository. This command removes a file from your file system and then removes it from the list of files tracked by a Git repository. **The --cached flag lets you delete a file from a Git repository without deleting it on your file system**.

GIT :

- It is a distributed version control System.
- It is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

NOTE :-

When we update our MacOS, then git stops working.
We can check that with the following command - “git —version”
We will get some error -
“xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools), missing xcrun at: /Library/Developer/CommandLineTools/usr/bin/xcrun”

This is bcz - Xcode command line tools needs to be updated.
Solution :

1. xcode-select --reset : type this command as it worked for me.

2. xcode-select --reset : this command reinstalls the tools, but it might show some error - "Software not found on server"

Got to below site and search for "Command Line Tools for Xcode 12.x"

https://developer.apple.com/download/more/

in the list of downloads Then click the dmg and download.

Tutorial

- **mkdir** gitWorkspace
- **cd** gitWorkspace
- **ls** (not files in directory now)
- **git init** : creates and initialises a git repository.
  the above command creates a ".git" folder in the current dir.
  In linux/unix based systems any file starting with a ".filename" is hidden, to view hidden files we type "ls -a" .
- This ".git" file stores the entire history of all the changes made in the project.
- **touch** names.txt : creates a text file.
- **git status** : to check the status of untracked changes.
- **git add names.txt** : puts the names.txt file to staging area.
- git status : to check the status of tracked file.
- **git commit -m "names.txt file added"** : commits the changes to save it as
  an event in history.
- **vi names.txt** - opens the vim editor.
- Added some text to the file.
- Esc :wq - to save and exit.
- git status - shows untracked changes in names.txt file.
- **git restore —staged names.txt** : restores the changes added to the staging area, so the changes made in file are again untracked.
- git status : untracked file names.txt
- **git add .**
- git commit -m "names.txt modified again"
- **git log** : tells the history of the repository.
  NOTE : Press 'q' to exit from log.
- **rm -rf names.txt** : deletes the file.
- git status : untracked changes, names.txt file deleted.
- git add .
- git commit -m "names.txt file deleted"
- git status
- git log

Git reset [--hard/ --soft] <HashCode> :

Suppose if we want to undo some changes we made in our repo :

- **git reset <hashcode>** : “hashcode” here should be the hashcode of the commit we want to keep and all the commits after that are then moved to the unstaged area.
- eg : “git reset 859ed783e702ca1c76ee2685b4360a19e0b83f71”
- here we give the hashcode of the first commit where we just created the file, so all other commits after that, there are 3, will be undone and the latest commit is put into the unstaged area.
- git status : says “deleted names.txt” is not staged for commit.
- **git reset --hard <hashcode>** : this commands rests the commits after the hashcode but the changes are not reflected in staging area.
- **git reset --soft <hashcode>** : This command restets the commits but the removed commit is pushed to the staging area, and we can either use it or remove it.

Suppose we want to keep these unstaged commits as a backup for future :

- git add .
- **git stash** : this puts the changes into the stash area for us to use anytime, and we can continue working in clean branch which has only the changes till the commit that we did reset.
- **git stash pop** : bring back the stashed changes into the unstaged area.
- there could be conflicts if we want to apply these changes again.
- **git stash apply** : this applies the changes in stash area to the current working tree, but there might be conflict which needs to be resolved.
- **git stash clear** : this clears the stash area.

Github, origin, master, main, upstream :

- **upstream** : refers to the location through which we clone the repository.
- **origin** : refers to the address of the remote repo where we keep our project.

NOTE : We can give any random names the these addresses instead of “upstream” and “origin”, but it is a general convention used by everyone.

- **master, main** : master is now called as main, main is the main branch of our repo, which will have the complete error free code. and if we wish to try/experiment any changes in our project we can create branches, which later on can be merged with the main branch.

Miscellaneous

- **git diff** :
- **git show <Hashcode>** : shows the changes made in that commit.
- **git branch** : shows the current working branch.
- **git branch [Branch-name]** : creates a new branch with given name.
- **git checkout [Branch-name]** : to switch to the mentioned branch.
- **git checkout -b [Branch-name]** : creates and switch to the mentioned branch-name.
- **git remote add origin https://<PAT>@<git-repo-url>** : to add PAT to

our local git repo.

- **git remote remove origin** : to remove any link saved with origin as name.
- **git remote -v** : to show all the remote url values.
- **git push origin [branch] -f** : This is "**Force Push"**, we do this when in our local we have removed some commit, using **git reset**, and in our remote the changes are not yet applied then we force push these changes to our remote.
- **git log -4** : shows the recent 4 logs.

Note :

- When we fork any project, then a new clone is created in our github profile.
- Now, to add/work on any new feature in the project we should always create a new branch and work on that new branch.
- when we commit and push the changes to this new branch in remote repo, then we get a message to "compare and pull" the changes bcz this new branch is commit ahead of the main branch of forked project.
- So when we compare and pull a request is sent to the forked project's owner and so any further push to our branch in our forked repo will now not create new pull requests, this is why we don't commit to main branch.
- So, for new pull request we need to create new branches.

Git pull :

git pull does two things.

1. Updates the current local working branch (currently checked out branch)
2. Updates the remote tracking branches for all other branches.

git pull fetches (git fetch) the new commits and merges (git merge) these into your local branch.

 # Pull from specific branch

- **git pull [remote-name] [branch-name]**

Scenario : if Upstream is 2 steps ahead of origin.

git pull upstream main :

(pull from main branch of upstream)
- the changes are reflected in our local repo.

git push origin main :

(push to our own remote repo origin's main branch).

Git squash :

- we use **git rebase -i <hashcode>** to squash the commits into 1 commit, we get option to select what to pick and what to squash and

also we an add a message.

- use **“i”** to insert, **“esc + : x”** to create a message for the commit.

Git config :

- **git config --list** :
- **git config --global user.name “Nikhil Gautam”** or **git config user.name “Nikhil Gautam”** :
- **git config --global user.email [your email address here]** :