
Git Fork

- **A fork creates a completely independent copy of Git repository. In contrast to a fork, a Git clone creates a linked copy that will continue to synchronize with the target repository.**
- **The developer who performs the fork will have complete control over the newly copied codebase. Developers who contributed to the Git repository that was forked will have no knowledge of the newly forked repo. Previous contributors will have no means with which they can contribute to or synchronize with the Git fork unless the developer who performed the fork operation provides access to them.**

Git Configurations

- **System, user and project level configs**

1. **git config —system**
2. **git config —global**
3. **git config —list to list all configurations**

You can find the configurations you have applied in the .config file in the user directory
cat .config

Git

Git Auto-completion

- **When Tab is pressed git will complete the commands, file paths, and branch names.**
- **Separate file**

Git Help

git help

man ls

- **provides information about the ls command, so this is similar to the git help command**

Git init - initialize a project to use git

git add .

- Add all the chances in the current directory(.)
- Either the names of files to be added or a period (.) must be passed as an argument to git add so git knows which files to stage.

To make those changes permanent we commit the changes.
git commit -m “My commit”

1. Make Changes
2. Add the changes
3. Commit changes to the repository with a message

Writing Good Commit messages

- A short single-line summary (less than 50 characters)
- Optionally followed by a blank line and a more complete description
- Keep each line to less than 72 characters
- Write commit messages in present tense, not past tense
- Good - “Fix for a bug” or “Fixes for a bug” not “fixed a bug”
- Can add tracking numbers from bugs or support requests

View commit log

- git log - gives a log of all the commits done till now
- Every commit has a unique commit id and this is displayed at the top of each commit
- The author of the commit is displayed this information is stored in the config file.
- Latest commit is at the top of the commit

git log -n 5

- Will show the 5 most commits made

git log —since=2021-01-01
git log —author=“Kevin”
git log —grep=“Init”
(grep - globally search for regular expressions)
Will return all the commits with the init keyword.

tree architecture(File Structure)
2 tree

- repository
- working

Three tree architecture

1. **Working**
2. **Staging(when git add file.name is executed)**
3. **Repository(git commit file.name)**

We can add all the necessary changes to the stage and then commit them at the end.

Git commit values
Git labels it commit names using hash values SHA-1

Which best describes the typical basic Git workflow?

- **create new or edit existing files, stage the files, commit the files**
  Correct
  Though the workflow might differ according to the needs of the project, the essence of the workflow is change files, stage, and commit.

Head

- **A pointer to tip of the current branch in repository**
- **Last state of repository, what was last checked out**
- **Points to parent of next commit when writing commits takes place.**
- **We can move the head from one branch to another branch, git will keep track of that for us and we do not need to keep track of it.**
- **This is a file in the git repository**

```
cat .git/HEAD
        ref: refs/heads/master
```

You can find info in refs directory
```
cat .git/refs/heads/master
```

- **289102212f003df2fe1b922ad2e0dcce099b19f6**

We find the Hash value of the commit that is the current head.

Master Branch

- **Main branch**

Status

- **git status**
- **provides us with a status of the files after the commit**

untracked - Not in the git repository, if any changes are made to them, we cannot track them.

When file has been added to the staging area using the git add file_name command we can see that the status has been changed and now it is showing changes to be committed from untracked files

Edit files

After changes have been made to a file which has been committed to the repository when git status command is invoked we get the status that the file changed is modified.
to add the changes from this file we will follow the same procedure as before.

Git diff - TO VIEW CHANGES staging REPOSITORY AND LOCAL REPOSITORY
This command is used when we have to view the difference between the versions of the committed file and the file in the repository.

+++ means new changes have been added to this version.

When there are multiple files that have been changed, git shows that they have been changed in the sequential order.

---

git diff —staged/ git diff —cached

- **Shows the differences between the versions of the working repository and the version in the staging repository.**
- **staged is an alias for cached, therefore it does the same thing as cached.**

git diff --color-words

Deleted Files

- **Using git to track files when they are deleted**

When a file has been removed/deleted from the directory, we get the above message displayed

To remove it from the repository completely so that we do not need its previous changes, we can make use of
git rm file_name.txt
git commit -m "my message"

To perform both the above steps at once, we can directly use git rm file_name.txt command.

Move and rename files:

1.

State and Commit Shortcut

git commit -a -m
git commit —all -m
Stages and commits all the changes to tracked files.
Does not include untracked files

git commit -am "My message" or git commit -a -m "My message"

View Previous commit changes

git show 89fd2

- **This will display all the changes made in the previous commits**
- **Here 89fd2 is the hash value of the previous commit**

git show 94bd22 —color-words

- **Color words also works with git show**

Comparing 2 different versions of commits
git diff 89fd2..5ab4b

To compare with the latest commit (HEAD)
git diff 89fd2..HEAD

—---

Undo Changes

To undo changes in a file, we can make use of the git checkout command

git checkout — means that we want the changes to be made in the current branch not another branch.

git checkout — resources.html

To revert to a previous commit which we have made.

git checkout — .
To revert changes from all the modified files

To Unstage files
To unstage files we will make use of
git reset HEAD file_name

- This will remove the file_name from stage and back to the working repository

Amend Commits

- git commit —amend -m “Including new commit in the previous commit”

Suppose we have made some new change but we want to include it in the previously made commit we can do this using the above command
When we are making use of Amend comment, we are modifying the previously created commit and then including our current changes in that commit and then creating a new commit.

Retrieving old commits

- git checkout 51b4b86 — explorers.html
- - - current branch
- file_name - file to retrieve an old version of

Revert a commit

- Undo a change completely from the previous commit
- git revert 51b4b86
-

Remove Untracked Files

git clean

- Deletes the files
- -n command specifies what will happen if the git clean command is executed
- -f will delete the files
- -i will give us an interactive window to specify what operations we need to perform

git clean -n
Will tell us what untracked files will be deleted if the clean -n command is executed.

To ignore files

- Project/.gitignore
- List of rules to determine which files to ignore
- Changes made to ignored files will be ignored by Git

Rules

1. Writing Name of file
2. Pattern Matching
3. Negative Matching
4. Trailing Slash
5. Comments

.gitignore
access.log

the changes in access.log are ignored by git