**Git and Git Hub**

**What is Git?**

* Git is a version control system.
* Manages code history.
* Track changes.

**What is Git hub?**

* Git hub is largest development platform.
* Cloud hosting and collaboration provider.
* Git Repository Hosting.

**How Git and Git hub are connected?**

The local capabilities of Git with managing that code efficiently with managing the history and also with tracking changes and the capabilities of Git hub with being able to host the code on a cloud service and to enable that great collaboration capabilities, well this is how git and git hub is connected.

**Commands related branches:**

|  |  |
| --- | --- |
| **Command** | **Usage** |
| git branch | To get all branches |
| git branch <new-branch> | To create a new branch |
| git checkout <branch name> | To checkout to another branch |
| git checkout -b <third-branch> | It creates a branch with name third-branch and checkout it. |
| git push origin test\_branch | After creating a branch use this command to push the branch to github |
| git merge <third-branch> | It merges the changes from third-branch to the current checkout branch |
| git branch -M “Vishal” | Changes the master branch name to Vishal |
| git switch <third branch> | Switch to another branch, same as checkout. |
| git switch -c fourth-branch | Created a new branch with name of fourth-branch and switched to the branch. |
| git branch -d “branch name” | Deletes the branch only if the branch is merged. |
| git branch -D “branch name” | Force delete the branch even if the branch is not merged.  Ex: git branch -D fourth-branch |
| git checkout -b feature  git push -u origin feature | Creating a branch in local and pushing to github |

**Most used commands:**

git status : To see tracked and untracked files.

git add : To add to stage area.

git add . : To add all files to stage area.

**Undo unstaged changes:**

git checkout <filename> & git restore <filename>: If we want to re-do the changes(not staged yet) then we can use this command.

Ex: git checkout initial-commit.txt

git checkout . & git restore . : If we want to re-do the changes(not staged yet) for the all files then we can use this command.

git clean -dn : To delete the untracked or un staged newly added file. ‘n’ shows the files you want to remove.

git clean -df : To delete the untracked files which are deleted.

**Undo Staged changes:**

git restore --staged <filename>

**Undo commits:**

Head : When we checkout to the new branch or any existing branch, the last commit is referred as head.

Detached-Head : When we checkout on a specific commit from any branch then temporary branch will be created in which head is pointing to that specific commit. When we checkout to any existing branch the temporary branch will lost.

Soft & Hard reset/undo commit:

* ***git reset --soft HEAD~1*** : A soft reset, commit will be deleted (last commit) but files will be in staging area.
* ***git reset HEAD~1*** : It will reset the last commit and point the head to correct commit and delete the staging area files.
* ***git reset --hard HEAD~1:***

Removes the latest commit.

Removes the staging area changes.

Removes the file if any related in working directory.

**Important Point To remember:**

After resetting the commit if we have to push the reset commit change to git hub use below command.

***git push --force origin master***

**Detached head branch creation** :

When we checkout with a commit and in the detached head if you made changes then you need to create a branch with that changes before checking out to other branch, otherwise the changes will be gone.

***git branch <new branch name> [detached head id]***

**STASH :**

The stash is kind of an internal memory you could say where you can save uncommitted unstaged changes.

|  |  |
| --- | --- |
| **Stash commands** | **Use case** |
| git stash | To save changes |
| git stash apply | To bring back changes |
| git stash list | To see all stash in git repo |
| git stash push -m “message” | To save with a message |
| git stash pop | Removes stash from stash list & brings back the change. |
| git stash drop 1 | Deletes the stash at index 1 position |
| git stash clear | Deletes all stashes |

**Bring back lost information** : {commits}

***git reflog*** : We can see all the changes we have done.

1. I will delete a commit by doing.

**git reset --hard HEAD~1**

1. I will do ‘git reflog’ and copy the ‘ID’ of the commit which is deleted.

**Ex** : 4810efa **HEAD@{1}:commit: file2 added**

1. If I want to bring back the commit which I was deleted before using hard rest, I can use the below command.

**git reset --hard 4810efa**

**Bring Back lost information** : {branches}

* Assume, currently we are on master branch
* We created a new branch and switched to that feature branch.
* In that feature branch we added a file & staged it & committed in that branch.
* Now we switched back to master and deleted that feature branch.
* Now we can use ***git reflog*** and get the last committed ID.
* Then use *git checkout <ID>* and then use git switch -c feature branch.
* TADA, we got the branch again.

**Merge Types** :

* Fast-Forward
* Non Fast-Forward

1. Recursive
2. Octopus
3. Ours
4. Subtree

**Fast-Forward merge & UNDO merge** :

The fast-forward merge only works if we have no additional commit in the master branch after we created that feature branch.

Master m1 m2 ?

Feature m1 m2 f1 f2

So, our m2 commit in the master branch is the last change we had there. If that’s the case, the fast-forward merge simply moves the HEAD forward to our f2 commit here, but it does not create a new commit.

* git switch master
* git merge feature

To reset the HEAD from f2 to m2 (or) we can say undo the merge we can use the below command.

* git reset --hard HEAD~2

Here 2 is because we must go back m2 but after merging we moved our HEAD to f2.

**Recursive merge** :

If in master, we made another commit m3 and now we are merging with feature branch which has m1 & m2 & f1 & f2 commit we can use below command.

**git merge feature**

In master (git log) we can see a new commit is added “merge branch feature” .

To undo the merge in this recursive case we can use below command.

**git reset --hard HEAD~1**

It reverts f1 & f2 & “merge branch feature ” commits from master.

**Rebase** :

In the master we made m3 commit. But feature branch cloned from master after m2 commit & added f1 & f2 commits in the feature branch. Now in the feature branch we can bring the m3 commit by using rebase we can use below command.

***git rebase master***

And now if we do **git log** we can use below structure in the feature branch commits.

f2

f1

m3

m2

m1

Note : But commits f1 & f2 will be added newly, i.e. commit Id’s change and history also change so it is actually not preferred in real world application.

Finally, we can use switch to master and use

***git merge feature***

This makes a fast-forward merge.

Cherry pick

If we made change in master in m3.txt file with a typo & staged and committed.

Consider a new branch is created on the master branch let it be feature-2 branch and checkout to this branch.

In the feature-2 branch we added some files and committed them.

And we found that ‘typo’ in master file we made and corrected that typo and committed.

Now again we added a new file in the feature-2 branch and committed, we use git log and copy the commit-ID related to ‘typo fix’.

Now we switched to master and we can use **git cherry-pick <commit-id>** (commit-id related to feature-2 branch(typo-fixed commit-id))

***git cherry-pick 2aetfer54535***

Now the typo changes are come from feature-2 branch to master branch.

But copies commit with new -ID in master branch.