**Git Commands (Mac)**

**Bangla tutorial:** <https://www.youtube.com/watch?v=cPgIpUraWQo&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=1>

**Git documentation:** <https://github.com/anisul-Islam/github-documentation>

**Git cheatsheet:**

<https://gist.github.com/keeperofthenecklace/3394158>

Git has three stages locally.

1. **Working directory / area**(red files)
2. **Staging area**(green files)
3. **Local repository**(after commit)

All the works we do are in the **working area**. Then we send them to **staging area** and then we commit them to the **local repository** and at last we push them to our **remote repository** or github.

**HEAD** always points at the current commit / branch.

1. Version check: **git –version** or **git -v**
2. Git command help: **git some-command -h**
3. Enter a folder: **cd folder-name**
4. Exit a folder by one step: **cd ..**
5. Exit to root folder: **cd**
6. Remove existing git setup: **rm -rf .git**
7. Check all the files and folders in the current folder: **ls** or **ls -a**(**-a** is used to show hidden files)
8. Initialise git in a folder: **git init**
9. Create new file: **touch file-name.extension**
10. Delete a file: **rm file-name.extension**
11. Open a file:**open file-name.extension**
12. Open a folder: go to the folder and then **open .**
13. Clear terminal screen: **clear** or press **control + L**
14. To see the present working folder or directory: **pwd**
15. Create a directory or folder: **mkdir name**
16. Delete or remove a directory: **rmdir name**
17. To know the status of a working directory: **git status.**It will show all the new (**untracked**) and modified files. Then we send them to staging area with this command. **git add file-name** (for a single file) or **git add .** (for all the new untracked and modified files). From here we send them to local repository with **git commit -m ‘message’**and at last we push them on remote repository with **git push -u origin branch-name**
18. Amend/modify **previous** commit. Make changes then **git add .** Then **git commit --amend** and the previous commit will be amended. If we want to change the previous commit message, **git commit --amend --m ‘messsage’**
19. **git status -s**will show the status in short form. Left column is the **staging** area and the right column is the **working directory**.  
    Green **M** means the file is present in staging area.  
    Red **M** means this file is modified.  
    Green **A** means the file is added in the staging area.  
    Red question (**?**) means it’s a new untracked file.
20. We can stage and commit with a single command **git commit -am ‘message’**.This will not include newly created untracked files.
21. To see all the staged files: **git ls-files**
22. Download or pull all the latest codes from remote repo: **git pull origin branch-name**
23. To check commit history: **git log** then come out of the screen **Q**or**:Q**If we want to see the latest 2 or 3 histories: **git log -2**or**git log -3**If we want to see history briefly: **git log --oneline**, it will show short history of commits where the latest commit being at the top.  
    **git log --oneline --reverse**will show the history in reverse order where the oldest commits will be at the top.   
    **git log --oneline --all** to see all branch commits.  
    **git log --oneline --graph** will show diverged log graphically.
24. To remove all untracked files **git clean -fd**
25. To remove files only from staging area (green color) then:   
    **git rm --cached file-name** or **git rm --cached -r file-name**
26. To discard or undo unstaged file changes (red color files):  
    **git restore file-name**or**git restore .  
    git checkout file-name**or**git checkout .**To remove all the new untracked files **git clean -f** (-f to force as it physically removes files). To remove whole directories **git clean –fd**
27. To unstage files from staging area (green color to red color):  
    **git reset**  
    **git restore --staged file-name** or **git restore --staged .**  
    **git rm --cached file-name** or **git rm --cached .**
28. Remove a folder (bin) from staging area. **git rm --cached bin/** or  
    remove it recursively **git rm --cached -r bin/**
29. To go back to a previous commit **by deleting all the later commits / histories**:  
    **git reset commit-id  
    git log** will show all the commits with a unique **commit-id**.
30. To view all the file changes in the last commit:  
    **git log --stat** or git **log --oneline --stat** (for one liner log)
31. To view the last / latest x number of commits:  
    **git log -number** or **git log --oneline -number**ex: **git log -3**
32. To see details of a particular commit:  
    **git show commit-id** or **git show HEAD~2** (Here 2 means, 2 commits before the latest (**HEAD**) commit)
33. To see a complete snapshot of our project in a previous commit:  
    **git checkout commit-id**, It will show a warning message, **You are in 'detached HEAD' state.** The **HEAD** always points to the latest branch/commit **master**. But when we check out to a particular commit, the HEAD is detached from master branch and points to that commit. We should not make any commit at this state as they will be automatically deleted later. To go back to normal form, **git checkout master**
34. To undo or delete commit:  
    **First:** If you want to just undo commit and keep all the files intact in the staging area (green color) then: **git reset --soft HEAD^  
      
    Second:**If you want to undo commit and remove them from the staging area (red color) then:  
    **git reset HEAD^  
    git reset HEAD~**  
    **Third:** If you want to undo commit and discard or remove all changes and go back to the previous commit state then: **git reset --hard HEAD^**To undo multiple commits serially (latest to oldest):  
    **git reset --soft HEAD~2**or **git reset --soft HEAD~3**
35. If we want to see what did we change / modify in the files, we use **git diff**and it will show us the difference but for this, these files has to be staged (**git add .**) first and then compare the new changes (**red files**).
36. To see details of a commit: **git show**, it will show the details of the latest commit. If we want to see details of another commit:  
    **git show commit-id**(or the first 7 characters of the id) or  
    **git show HEAD~1** or **git show HEAD~2** HEAD is at 0. So 1 means 1 step earlier commit.  
    If we **git log** then it shows all the commit id’s.
37. To move from one commit to another commit: **git checkout commit-id**(It will show **HEAD detached**). If we want to go back to the latest commit then: **git checkout master**
38. To update other latest commits from master branch to the current branch: **git rebase master**

**Remote repository:**

1. To check if there’s a remote repo for this local repo: **git remote** or **git remote -v**, if it returns nothing that means there’s no remote.
2. Add or link remote repository with the local repository:  
   **git remote add origin git-SSH-link**from the git repo.**origin** is actually the HTTPS link’s reference. We can write anything else instead of origin. From next time we don’t have to write the entire HTTP link to push or pull our codes, instead we can only write origin. **git remote** will show the reference name (origin) and **git remote -v** will show the remote details.  
   To rename remote reference, **git remote rename old-name new-name**, ex: **git remote rename origin base**To remove remote reference **git remote rm reference-name**, ex: **git remote rm origin**
3. Clone or copy a remote repository to local: First go to the desired location where the folder will be cloned, then: **git clone git-SSH-link** or **git clone git-HTTPS-link**example: **git clone git@github.com:mamun2016/test.git**, here git will create a local repository by the same name **test**. If we want to change the name, **git clone git@github.com:mamun2016/test.git folder-name**

**Git branch –vv** is used to see if my **local master** branch is synced or behind or ahead from the **origin/master** (remote repository)

**git remote -v** to check if a local repository has a remote origin / repository linked to it.

**Change HTTPS to SSH:** Do this to switch to **SSH**:

**git remote set-url origin SSH-link**

Example:**git remote set-url origin git@github.com:mamun2016/test.git**

**Push, Pull and fetch:**

1. Once the files are in the local repository (after commit), we have to push them to the remote repository. For that we use, **git push -u origin branch-name.** We can omit **origin** and **branch-name** if the branch already exists at the remote and write only **git push** if pushing to the same branch.  
   If we have to force any action, we need to write **git push -f**
2. If we want to update our local repo with the latest codes from the remote repo, we can write, **git pull origin branch-name**. If we pull to the same branch, we can omit **branch-name**. If we omit **origin**, git will automatically assume origin.
3. Cancel or undo git pull merge conflict:  
   **git merge --abort  
   git reset --hard HEAD**
4. **git fetch** syncs the local codes with the remote server but doesn’t merge like **git pull**. With this command we can have all the branches and commits that are present in the remote.  
   **Video:** <https://www.youtube.com/watch?v=Mdo7hvlUJ-U>

**Branching:**

1. Create a branch locally: **git branch branch-name**. It will create a copy of the current / active branch in the new branch.
2. To see all the local branches: **git branch**. The branch with an asterisk (**\***) is my current / active branch. To see all the remote branches **git branch -r**After removing a branch from both remote / origin and local, it is still visible as remote tracking branch with **git branch -r**To delete these branches, **git remote prune origin** and the branch will be pruned.
3. To move / switch to a branch:   
   **git checkout branch-name** or **git switch branch-name**
4. To switch between the last two branches:   
   **git checkout -** or **git switch -**
5. To create a branch and move / switch to the branch immediately:   
   **git checkout -b branch-name** or **git switch -c branch-name**
6. Edit / change / rename a branch name:  
   **git branch --m old-name new-name**
7. Delete a branch: **git branch -d branch-name** or **git branch -D branch-name**. We can’t delete an active or current branch. We have to switch to another branch to delete a particular branch.
8. To view commits on all branches: **git log --oneline --all**

**Stashing:**

If we make changes in a wrong branch and we don’t want to lose them, we need to stash them so that we don’t lose all the changes that we made.

1. To copy all the codes: **git stash**. This command will store all the changes in memory and we can bring them back later. **git stash -u** is used to stash untracked files also.
2. To use the saved stash:  
   **git stash apply** (It will keep the stashed copy in the memory) or  
   **git stash pop** (It will delete / clear the stashed copy in the memory)  
   **git stash pop = git stash apply + git stash drop**
3. We can create multiple stashes with individual names: **git stash save “Name of the stash”**
4. To see the stash list: **git stash list** (It will give all the stashes’ names and ids to be used later. Ex: **stash@{0}**, **stash@{1}**)
5. To use individual stash: **git stash apply** **stash@{0}**or**git stash pop** **stash@{0}**(pop will delete that individual stash)
6. We can use index number to apply stash: **git stash apply 0** or **git stash pop 1**(0 is the latest stash)
7. If we have multiple stashes, **stash@{0}** will always be the latest / last stash.  
   If we don’t mention any stash id then,**git stash pop** will apply the latest stash and drop / delete the latest (**stash@{0}**) stash.
8. To see the changes in a stash  
   **git stash show stash-id**, ex: **git stash show** **stash@{1}** or  
   **git stash show 1**
9. To delete / clear individual stash: **git stash drop stash-id**
10. To delete / clear all the stashes all together: **git stash clear**

**Video:** <https://www.youtube.com/watch?v=KLEDKgMmbBI>

**Merging locally:**

There are two types of merging.  
1. **Fast-forward merging**: If the master branch has not changed since the new branch was created, the new branch can be directly merged with the master branch. This is a fast forward merge.

2. **3-way merging**: If the master branch has new changes after the new branch was created, the branches don’t share same changes. When we run merge command, git creates a new commit combining changes from both the branches and merge them. This is called 3-way merging because this is the combination of three different commits.  
1. The common ancestor of both the branches (when the new branch was created from master).  
2. The latest master branch.  
3. The new branch.

1. To merge one branch to another locally: **git merge branch-name**. This will merge the other branch with the current or active branch. For example, If I am in the **master** branch and want to merge **dev** branch with it, then: **git merge dev**

2. To check which branches are merged locally with master branch:  
**git branch --merged**

3. To check which branches are not merged locally with master branch:  
**git branch --no-merged**4. To abort a merge: **git merge --abort**

5. Reverting a merge commit: **git revert -m 1 HEAD**

**Fast forward merge or 2 way merge:**

Watch the video if needed: <https://www.youtube.com/watch?v=FbX_MpqSLfc&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=19>

**3 way merge:**

Watch the video if needed: <https://www.youtube.com/watch?v=dPSB-UhioJw&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=21>

**Cherry picking:**

If we want to merge some particular changes / commits from other branch to master branch without merging, we have to use cherry pick.

**git cherry-pick commit-id** from the branch where you want to merge the changes (here master)

**Resolve merge conflict on git**

**git rebase** or git merge might show some merge conflicts. We have to resolve the conflicts and then follow the instructions given on the terminal.

**git rebase --continue**, **git commit -am “commit message”**

If it shows text on the command prompt and ask to write commit messages then write message. Click **Ctrl + c** and then **:wq** and press <enter> and it will come to the command prompt again.  
 **Video:** <https://www.youtube.com/watch?v=LapVNu99qU8&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=21>

**Fork and clone:**

If we want to contribute to others’ repo, we use these features.

**Video:** <https://www.youtube.com/watch?v=FH0wptOLukk&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=23>

**Collaboration in team:**

If a team members decide to work together in a single repair, they need to collaborate and share read-write permissions in the repo.

Video: <https://www.youtube.com/watch?v=nS9QbJt4KaQ&list=PLgH5QX0i9K3qAW8DT6I0XOxC23qnA4FL-&index=24>

**Visualizing how git works**: <https://git-school.github.io/visualizing-git/>

For unrelated histories pull

git pull origin master --allow-unrelated-histories