GIT

git init

git config --global user.name "Yogesh Soniwal"

git config --global user.email "yogeshsoniwal@gmail.com"

git add <filename> —> staging

git rm --cached <filename> —> unstaging

git log -- all the git commit history

git log --stat —> stats for commits

git clone <Path> — cloning a remote repo to local machine

git branch —> gives all branches as output

git push <remote repo name> <branch name> —> remote repo name is usually ‘origin’ (when we clone a repo, it is automatically named as origin)

eg.: git push origin master

git show <commit id> —> shows all the changes (diff) in this commit

git diff <commit hash id> <commit hash id>

git reset --hard <commit id> —> resets the working directory and commit log to the commit id

git reset --soft <commit id> —> resets the commit log

git pull —> updates the working directory to latest commit in master branch

HEAD —> identifier of most recent commit (newest)

HEAD~1 —> identifier of second most recent commit and so on

git rev-parse HEAD —> return the commit id as well

To remove a file

<first delete the file from repo>

git rm <filename>

git commit

**BRANCHES**

git branch <new-branch-name> —> creating

git checkout <the-branch-name> —> switching

**git merge <branch-name-1> —> merges branch-name-1 in branch (we first need to switch to the branch where we are merging**

Deleting branch:

git branch -d <branch-name>

Checking out branch from remote

* git fetch —> will fetch all the branches and commits from remote. Won’t make any changes in the working directory. But will update Git’s list of branch names and commits
* git checkout <branch-name> —> will look in local and remote repo, since it is only in the remote repo, Git will copy it into local repo and will also make <branch-name> as current branch

The typical Git workflow looks like this:

* Create a branch off of master with the name of your feature. Let's say feature/better-algo.
* Make your changes on the branch and create commits.
* Push the branch to the remote repo.
* Ask others to review and evaluate your branch.
* Merge the branch into master once everyone thinks it looks okay.
* Delete the branch.

When we use GitHub as our remote repo, [pull requests](https://help.github.com/articles/using-pull-requests/) will show us the differences between branches in an attractive interface, and allow other developers to add comments.

But for git command line we use

git diff <older-branch> <new-branch>

Branch Naming conventions

* Feature - feature/happy-bot
* Fix - fix/remove-error
* Chore - chore/add-analytics - Chores are things that end users won't necessarily notice, but help us reorganize the project or make the code more efficient
* When a new branch is created, it inherits the commit history from the commit it was created from

**MERGE conflicts**

* If merge conflict happens, git adds markups to the conflict files, to remove these, we do
* git merge —abort
* git mergetool --tool-help —> for graphical merging
* git checkout --ours <filename> —> keeps files from master (where we are merging)
* git checkout --theirs <filename> —> keeps files from where we are merging
* Add .gitignore file
* If we want to not track a file from future commits and also want to keep in the repo then “DON’T DELETE IF FROM THE REPO, USE FOLLOWING COMMAND (CACHE THE FILE) AND COMMIT THE CHANGES)
* git rm --cache bot.py