# basics operation:

git init

git add .

git commit

git pull

git push

# advance operation (use in teams):

branching

merging

rebasing

# basics terms:

repository

working directory

file status (untracked, unmodified, modified, staged)

staging area (virtual area for those files which will add in new commit)

# user, email:

git config --global user.name ["username"]

git config --global user.email ["user email"]

# new git local directory:

git init

{add in staging area}

git add .

git add abc.txt xyz.txt

git add \*.txt

# commit change:

git commit -m "commit message"

{check all history}

git log

# show current status:

git status

git log

# head:

{point to our current branch and their last commit}

{change when we rebasing or reset}

{commit hash 40 character unique}

# un stage:

{before commit remove from staging area}

git reset

{un stage and remove/discard all changing}

git reset --hard

# git ignore:

{ignore specific file mention in “.gitignore”}

\*.txt, path/xyz abc.txt, [directory name]

# git clone with all branch:

mkdir [remo name]

git clone --mirror [repo url] .git

or

git clone --bare [repo url] .git

ls -a

git config --bool core.bare false

git reset --hard

git branch

# branch:

{for each context, do not work in main branch because main is stable or production branch}

{main branch is default branch}

two types of branches

1: short-lived branch (bug fix, features)

2: long-running branch (main, test, development)

production (main)

development (dev)

test (test)

user or context specific or feature branches

only show the current data of that particular branch

# branch commands:

git status

git branch

git branch -v

{new branch will same or inherit from current working branch}

git branch [new-branch-name]

git branch

{show the last commit of all branches}

git branch -v

git checkout branch-name

{show the commit history of current working branch}

git log

{old branch has any change}

git log new-branch..old-branch

{new branch has any change}

git log old-branch..new-branch

{mentioned branch will merge current working branch}

git merge branch-name

# conflict:

{when two branches work same file and same line}

{conflict arise on merging}

best way to resolve conflict is use smart-git or any other UI base git tool

# stash:

{save work temporally and use it whenever you want}

{very useable when switch to other branch during work}

git stash

{save without name and with name}

git stash save [name for that stash]

git stash list

git stash pop

{apply last stash and remove from stash clipboard (history)}

git stash apply [stash name]

or

git stash apply [stash@{0}]

{apply specific stash and don't remove}

git stash clear

# remote repositories:

{github, bitbucket, gitlib}

{clone remote repo}

git clone [remote repo url]

git push [save on remote url]

git fetch [fetch only new change from remote repo]

{must use after fetch otherwise we will not get the new changes}

git merge

{fetch and merge new change from remote}

git pull

{show remote repo link}

git remote -v

{show the detail summary of remote repo}

git remote show origin

{this will add in our local workstation}

git remote add [repo name or link]

git push

git push origin [branch name]

# local repo upload to github:

{create local repo and after that create a repo on github and upload our local repo}

git remote -v

git remote add [set repo name 'hello' default is ‘origin’] [remote repo url]

git remote -v

git status

git remote show [already set repo name 'hello']

git push -u [already set repo name 'hello'] [branch name 'master']

# remote branch:

git branch

git status

git branch [new branch name]

git checkout [new branch name]

git status

{add some change in new branch}

git add .

git commit -m "some message"

git remote -v

git push -u [already set repo name or default name origin] [new branch name]

# merge remote branches:

git checkout master

git merge [new branch name]

git log

git push [already set repo name 'hello'] master

# create branch on github and fetch in local:

git branch

git pull

git branch

git checkout [new remote branch name]

# git workflow [git-flow]:

main (deployment) (tested code for clint)

development (merge all new features and test/review)

feature (foreach feature)

hotfix (for emergency bug fixing)

# hithub feature for pull request:

create pull request for development

{developer create pull request for development after complete feature and senior can review}

senior or team lead will merge after review

{senior can bound for merge}

# forke (github feature):

{we can copy any public repo in our github account}

{and we can change and then merge it into the original repo}

{get update from forked repo}

git remote [can give any name best is: 'upstream'] [remote forked repo url]

git remote -v

git status

{pull and push request for remote forked repo}

git branch upstream/[branch name]

git status

git checkout [branch name]

git merge upstream/[branch name]

# delete branch:

{delete local branch}

git branch -d [branch name]

{delete remote branch from pc}

git branch -dr origin/[branch name]

{delete remote branch from pc and github}

git push origin --delete [branch name]

# undo changes (reset) remote:

{undo from current branch}

git reset --hard

{undo from all branches}

git reset --hard HEAD

{undo commit (delete commit)}

{undo commit => it's created new commit and show previous changing before that commit}

{save all history}

{practically create an addition commit}

git log

git revert <commit hash can give only first 6 char>

git push

{undo commit moves to the given commit and remove all history}

git reset --hard<commit Hash>

git push -f origin master

{undo commit moves to the given commit and remove all history

{but show uncommitted data of that commit}

git reset --keep<commit Hash>

# merge types:

1: fast forward (merge)

2: merge commit (merge)

# fast forward (merge):

if we create new branch 'dev' from 'master branch'

and we do some new commit in dev branch only

but main branch has no extra commit

in this case we do simple merge

and this merge will fast and very easy

because just one branch has extra commit

and in this case no extra commit will do on merge

**example:**

git branch

git log

git branch dev

git checkout dev

git log

{create some new change and commit in dev branch}

git add .

git commit -m "new commit 1 from dev"

{create some new change and commit in dev branch}

git add .

git commit -m "new commit 2 from dev"

git checkout master

{fast-forward this is simple merge}

git merge dev

git log

git push origin master

# merge commit (merge):

if we create new branch 'dev' from 'master branch'

and we do some new commit in dev branch

and we do some new commit in master branch also

now both branches have some new commit

in this case we do merge commit

and this merge will slow and complex

because both branches have new commit

and now a extra commit will do on merge

this extra commit do for combine the both branches on same point

**example:**

git branch

git log

git branch dev

git checkout dev

git log

{create some new change and commit in dev branch}

git add .

git commit -m "new commit 1 from dev"

git checkout master

{create some new change and commit in master branch}

git add .

git commit -m "new commit 1 from master"

git log

{now master and dev have some new commit}

{this will not fast-forward merge}

git merge dev

{and now one extra commit will do}

git log

git checkout dev

{git will not show the new change of master}

git log

{now this merge will be fast-foreword}

{because and addition commit will do on master now no need for extra commit}

git merge master

# rebase:

{before rebase we should understand the merge and its type fast-forward and merge commit}

{merge show the time line or history of every commit}

{rebase will not show the time line or history}

{rebase not show the branches it's show that this all happen in single branch}

{this will add branch-B commit first and then merge branch-A changes}

{this will temporarily discard the branch-A change and merge branch-B and then add branch-A change}

{this will look as that all happen in one branch}

example:

gut branch

git status

git branch dev

{add some new change in master}

git add .

git commit -m "new commit 1 from master"

{add some new change in master}

git add .

git commit -m "new commit 2 from master"

git checkout dev

{add some new change in dev}

git add .

git commit -m "new commit 1 from dev"

{add some new change in dev}

git add .

git commit -m "new commit 2 from dev"

git checkout master

{add some new change in master}

git add .

git commit -m "new commit 3 from master"

{now no history or time line will show in rebase}

git rebase dev

{this will create new commit and show single commit all previous commit now shows}

git log

# Markdown:

{use in readme file github}

**Heading**: #

**style:** \*\*bold\*\*

**quote:** > this text show unique

**code:** backtick `git status command` or ```code ```

**link:** [title](link)

**list:** - unorder, 1. order

**task list:** [space] check box unchecked, [x] checkbox checked