

1. git init --> git local repo

2. git config --list

--> global (available for all the project on the system) ,

system (available for git operation), local level(.git repo)

2 most important config parameter should be set at global level

a) user.name --> commit

b) user.email

> git config --global user.name ankan

> git config --global user.email ankan\_mitra@persistent.com

git config --list --global

user.name=ankan

user.email=ankan\_mitra@persistent.com

>> touch math.py (to create any file)

3. git status

4. git add mabrth.py (adding to staging area)

5. git rm <file> --> remove file from WD and staging area

git rm --cached <file> --> remove from staging area

6. git add ./ \* --> to move all the file to staging area

>> git add + git commit --> git commit -a -m "<msg>" / git commit -am "<msg>" --> use this when file is already tracked.

7. git commit -m <msg> --> commit(save) the changes in local repo along with msg

8. git log (checks the version history of commits)

> git log --oneline --> short of log

> git log -2 --> shows last two commit

> git log -p --> changes done for every commit

> git log -p -2 --> changes of last 2 commit

> git log --grep="Initial" -->shows commit related to search

> git log --author="ankan"

> git log --since="09/16/2021"

> git log --until="09/16/2021"

9. git commit --help

-----

SHA - Secure Hash Algorithm - SHA-1

fixed length hash= SHA(input data) --> data can be in bits/bytes/GBs/Tbs

properties SHA -

1. Deterministic - for same input we always get same output regardless of environment
2. output should be fixed length
3. Avalanche effect - minor change will completely change hash
4. Unique value - no 2 input should have same hash value

-----

10. git restore --staged <file> ---> rollback to previous point

11. git commit --amend -m "Added substract() function in math.py" --> last commit update msg

12. git diff -->gives you the diff between stages(Working Directory,Staging Area,Local Repo) of file

WD

SA

LR

-----

1.

dif diff --> Changes1

--> change1

dif diff head

--> change1

## 2. Put changes to SA

`git diff --> Changes1`      Changes1      --> No output

`git diff HEAD`      --> change1

## 3. Put changes to SA and perform in WD

`git diff --> Changes1+Change2`      Changes1      --> Change2

`git diff HEAD`      --> change1 and change2

`git diff -->` diff btw WD and SA

`git diff HEAD -->` gives the difference b/w WD and LR

## 13. Restoring/undoing local/ stage changes -

`> git restore --staged <file> -->` restore changes in staging area LR

`> git restore <file> -->` restore changes in WD from LR

## 14. Undoing of committed changes

### 1.Safe way

`> git revert <SHA>`

### 2.Unsafe way

`> git reset --hard <SHA>`

`> git reset --hard HEAD~2`

## 15. Ignore (from being tracked) certain types of file -

File that need to be tracked - Documents, project code files being updated manually  
webpages.

File that need not to be tracked -log files ,executable files, library, .o, .tmp

`touch .gitignore`

- 
- 
1. Create math.py with basic code
  2. git init
  3. git add
  4. git commit -m "Initial commit of math.py"
  5. add fn header
  6. git commit -am "Added fn headers"

Execute git log --> should be able to see 2 commits

7. git branch <Branch name> --> create branch
8. git checkout <Branch name> --> to switch to branch
9. git branch --> shows all branches and pointer

2 types of commands -

> porcelain commands - git add, git commit, git status ....

> plumbing commands - core command (never use in projects)

git cat-file -t <hash>

- a. commit
- b. tree
- c. blob - binary large object

## 10. Branching and Merging - scenarios

---

> Scenario#1 - One or more commit on new branch and no extra commit on master branch

Implementation

> git branch Bug123

> git checkout Bug123

> git commit -am "msg" --> this goes on Bug123 branch

Merge -

> git checkout master

> git merge Bug123

> git push origin

Scenario#2 - One or more commit on new branch and one or more commit on master branch. (parallel branches) and without conflict situation

Implementation -

> git branch Bug456

> git commit -am <msg> --> perform 1 commit on master branch

> git checkout Bug456

> git commit -am <msg> --> perform 1 commit on Bug456 branch

Merge/Rebasing -

> git rebase master --> execute this while you are on Bug branch

> git checkout master

> git merge Bug123 --> merge commit from Bug456 into current branch

-----

Scenario#3 - One or more commit on new branch and one or more commit on master branch. (parallel branches) and with conflict situation

Implementation - (implement same fn on both the branches)

> git branch Bug789

> git commit -am <msg> --> perform 1 commit on master branch

> git checkout Bug789

> git commit -am <msg> --> perform 1 commit on Bug789 branch

Merge/Rebasing -

> git rebase master --> execute this while you are on Bug branch > git add <file> >git rebase --continue

> git checkout master

> git merge Bug789 --> merge commit from Bug789 into current branch

Resolve the conflict

-----

Rebasing with remote repo

Implementation -

> perform 1 commit on remote repo on main branch

> perform 1 commit on local repo on main branch

Rebasing

> git fetch origin --> fetch the changes from remote to local repo and not in WD.

> git rebase origin/main --> rebase local main with remote main

> git push origin

11. Deleteing branch :

git branch -d <Branch name>

12. Decorate log:

git log --graph --decorate --oneline

13. Rename branch :

git branch -m <old branch name> <new branch name>

14. Recover :

git branch + git checkout ---> git checkout -b <branch> <SHA>

15. Tagging:

> git tag V0.1

```
> git tag tag v1.0 -m "msg" <SHA> -->msg
```

```
> git tag -l -->tag list
```

16> Stashing: store local changes temporarily

```
> git stash
```

```
> git stash list
```

```
> git stash apply --> recovered
```

```
> git stash -m <msg>
```

```
> git stash pop --> applies on latest stash
```

---

---

Remote repository -

1. Add remote

--> > create new repository in github

```
> git init
```

```
> git remote add <remote_name> <.git(HTTPS)>
```

```
> git fetch origin main
```

```
> git merge origin/main
```

2. Clone

```
> git clone <.git>
```

3.git pull origin --> to push from Remote repo to WD.

origin is the name of remote repository

Case 1: change in remote repo at math.py and change locally on math.py

- > git checkout main
- > git commit -am "Implemented multiply() from local repo"
- > git fetch origin --> not pull as to fetch file in LD not in WD
- > git rebase origin/main
- > git push origin

4. git branch -a --> list all Local and remote branches

git branch --> list only Local branches

- > pull = fetch +merge
- > rebase = linear
- > pull then push

5. git remote origin remote\_repo --> remote repository name change

git remote rm <remote\_branchn\_name> --> remove remote branch

6. Bundle : to create a bundle folder

- > git bundle create <file\_name>.bundle HEAD main --> create a bundle folder
- > git clone <file\_name>.bundle <new\_name> --> to extract file in another machine

7. Push into repository:

- > git checkout <branch\_name>
- > git remote add <repository\_name> <.git HTTPS>
- > git push -u <repository\_name> <branch\_name> --> remotely create branch and add the

files

there after when we need we can to push



```
> git push -u <repository_name>
```