GITHUB NOTE BOOK

ParameshSPS

JANUARY 1, 2022

ParameshSPS

Dharmavaram, Anantapur Dist. 515671

Git & GitHub

HISTORY

Linus Torvalds

Git (Apr 6th 2005)

MAIN ROLE

Managing Complex/Large projects.

Useful everyone

Very important

GIT

Free and open-source version control system.

FEATURES

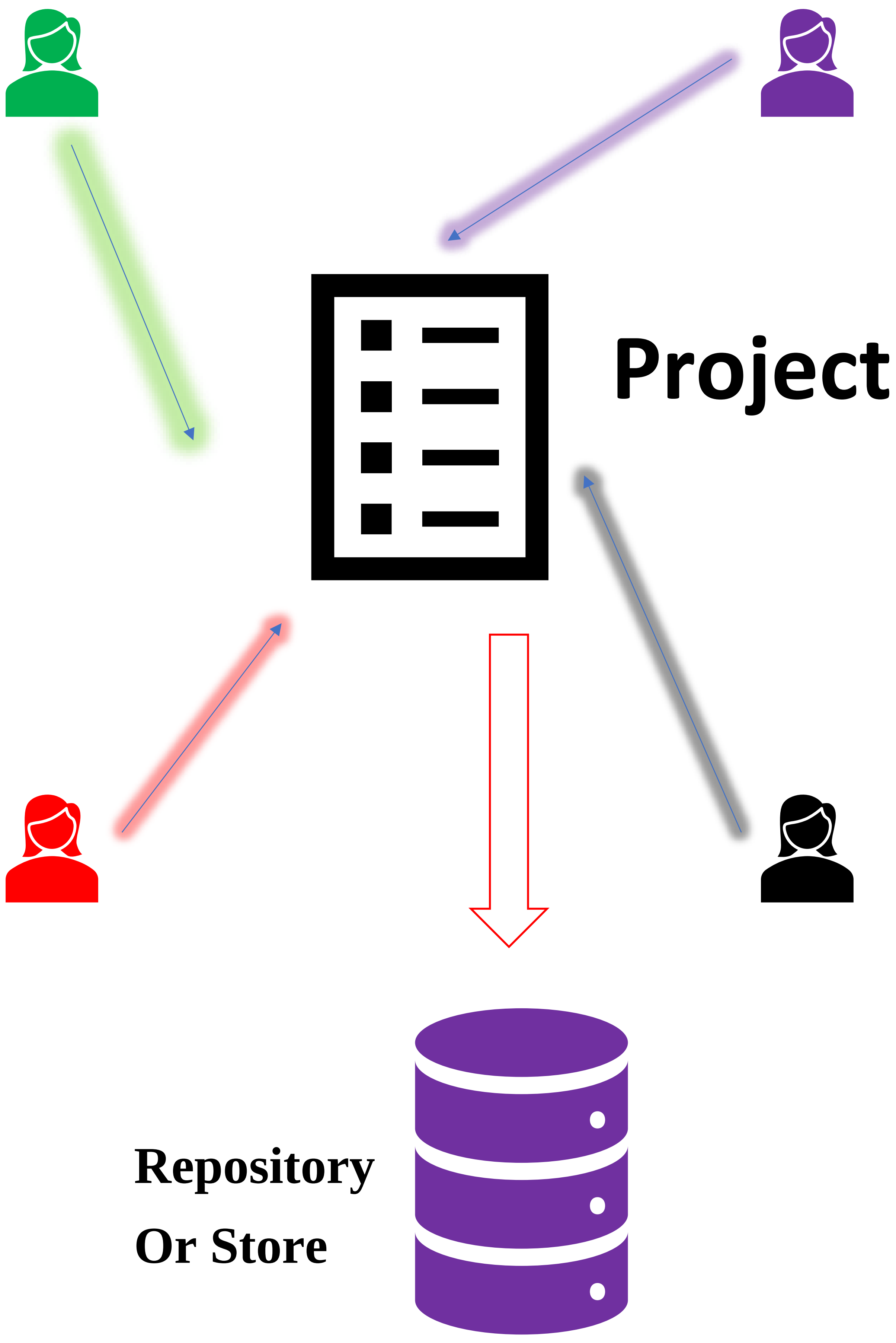
Distributed source control system

Open source

Large and active community

REPOSITORY

Means storage space (repository or directory)


Project
Repository
Or Store

GIT CONFIG

- **--global** ---- .git config - folder - user personal data
- --system ---- all users data
- --local ---- project repository data

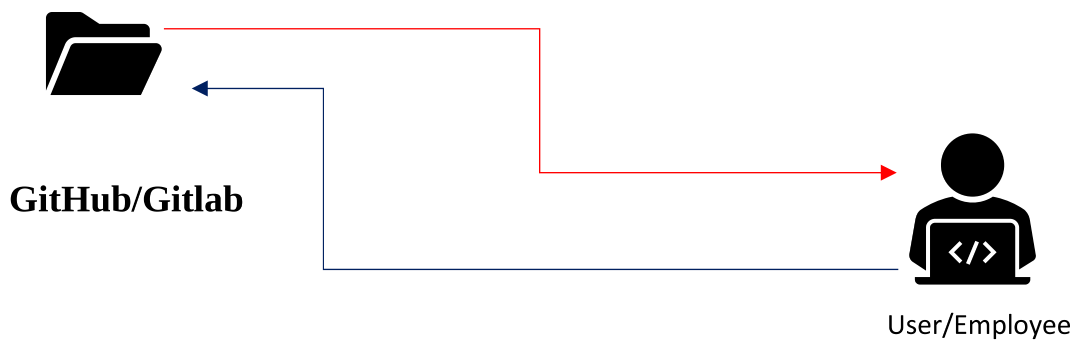
GIT BASH

```
$ git version----git version 2.35.0.windows.1
$ git config –global (Data) files visible
$ git config –global -e (Edit) file data visible
:q (Back)
clear
$ git config –list
$ git config –global user.name “Your name”
$ git config –global user.email “Your email id(GitHub’s)”
```

HELP

```
$ git help <verb> ex: $ git help config
$ git <verb> --help ex: $ git config --help
$ man git -<verb> Not working (Few Systems)
```

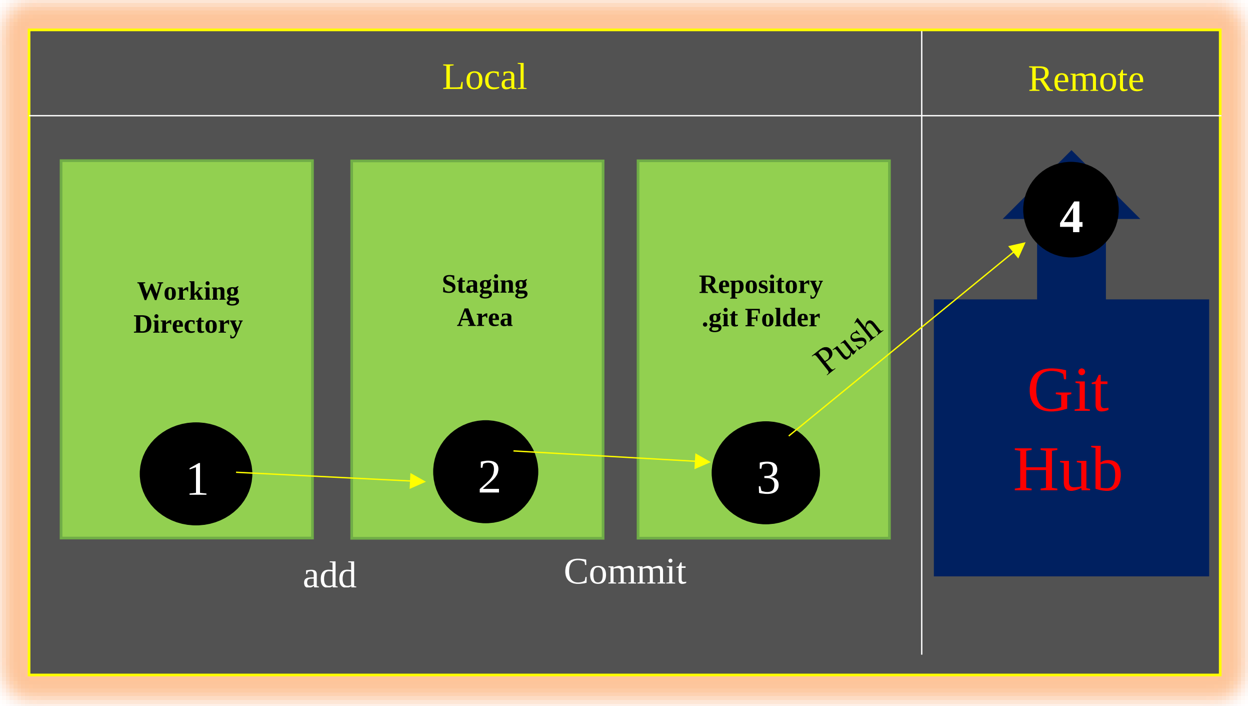
WORK FLOW



TERMS

- Commit ---------- SAVE YOUR FILES IN GIT
- Clone ----------------------- COPY
- Tracked/Untracked ------------------ .git (FLODER)
- Branch

GIT WORK FLOW



TWO NEW TERMS

- Master (Branch) or (Clone File is master)
- Origin (Server default name)

EDIT REPOSITORY FILE

Create a folder in local system (Direct)

Or

USING GIT BASH

```
$ cd D:
$ mkdir <Folder name>
$ cd <Folder name>
$ git clone <HTTPS link> or
$ git clone <HTTPS link> <preferred name>
$ dir
$ cd <Repository>
$ dir
$ dir -al
Changes in files
$ git status
$ git add README.md
$ git status
$ git commit -m “Changes”
$ git push origin master or main or …….
```

MAIN BRANCH TO MASTER BRANCH

```
$ git checkout main
$ git branch -m master
$ git push origin master or $ git push -u origin master
```

CLONING

```
$ git clone <Repository HTTPS link> or
$ git clone <Repository HTTPS link> <preferred name>
```

LOCAL PROJECT => GIT REPOSITORY

First create a new Repository folder in GitHub

Enter Repository name

Description (optional)

Select Public or Private

Finally Create Repository

After open Quick Setup

Go to Git bash

$ git init (Auto create an empty git)

$ dir & $ dir -al

$ git status or $ git status -s/ -m

$ git add .

$ git status

$ git commit -m “Initial commit”

$ git remote add origin https://github.com/ParameshSPS/Local-Repository.git

$ git push -u origin master (u means upstream)

REFRESH THE GITHUB

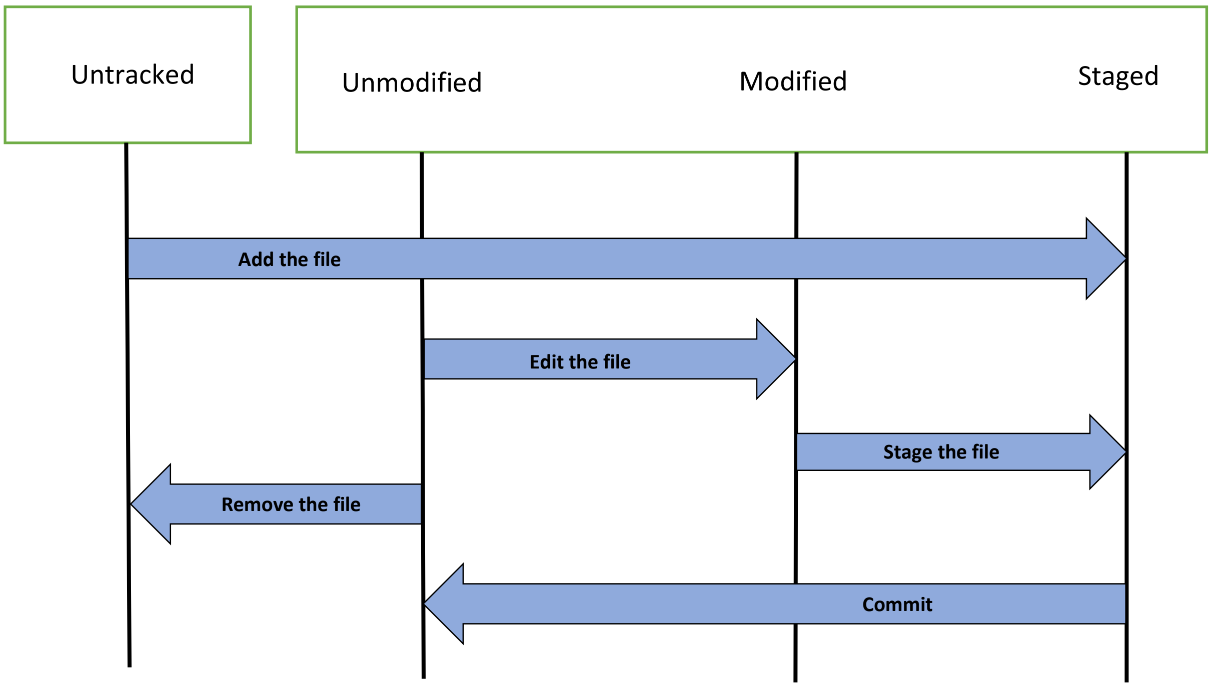
LIFE CYCLE

- Untracked
- Tracked

Unmodified

Modified

Staged



Create project in local system

Add 3 or 4 files and 1 folder (optional)

$ git dir or dir -al

$ git status (notic: add .git file)

$ git init (auto adding .git file)

$ git status (Untracked **files**)

$ git add . or choose 1 file

$ git status (1 file staged stage and 2 files are untracked)

$ git commit -m “adding home file”

$ git status (staged file going to unmodified stage)

Edit the commit file or changes the data in files

$ vi <file name> or npm <file name> or direct local file

$ vi <file name>

I (Enter) use arrows and edit after esc option

```
:wq (Enter)
$ git status (unmodified to modified)
$ git add <file name>
$ git status (file adding modified to staged stage)
$ git commit -m “updated file”
$ git status (again file is staged stage)
```

Git Status

```
$ git status –short or -s
?? – Untracked files
A – staged Area
M – Modified files
```

GIT IGNORE

Create project in local system

Add 3 or 4 files and 1 folder (optional)

```
$ dir
$ git init
$ git status
```

Create a file in project ---- .gitignore

```
$ vi .gitingore
    I (Enter) (ignore file add) esc
    :wq (Enter)
$ git dir -al
$ git status
```

<u>RULES</u>

#Comments or blank lines

Specific file: intro.html

File pattern: *.txt/.js/.html/.css/….

!main.js

Folders: images/

/images (Currect Directory)

<u>STATUS VS DIFF</u>

Create project and add file

```
$ dir
$ git init
$ git status
$ git add <file name>
$ git status
```

Changes the data

```
$ git status
$ git diff
$ git diff –staged/--cached
```

Status = Just files

Diff = file data or content changes

COMMIT

$ git commit

$ git commit -v

$ git commit -m “Initial Commit” (single line commit)

LOG (HISTORY)

1. Clone the project
2. git log
3. git log -p
4. git log -2 (last two commits)
5. git log –pretty=oneline
6. git log –pretty=short
7. git log –pretty=full
8. git log –pretty=fuller

GIT BRANCHING

Create a project in local system

Create three 3 files (optional)

$ git init

$ git status

$ git add .

$ git status

$ git commit -m “--------”

Changes the file data

$ git status

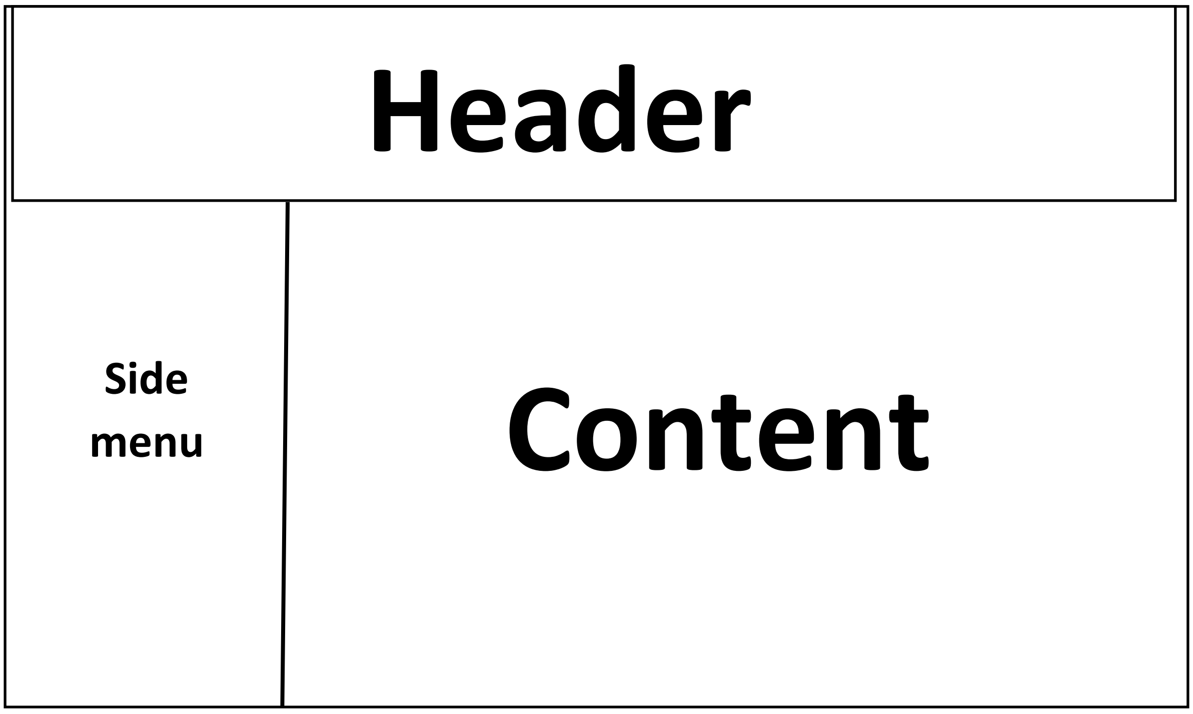
$ git add <file name>

```
$ git commit -m “Initial Commit”

$ git log

$ git log –oneline
```

Continued….

BRANCH

Branch is a pointer.



CREATE A NEW BRANCH

```
$ git branch <branch name>

$ git log –oneline

$ git checkout <side-name> (change branch)

$ git log –oneline
```

Note: $ git checkout -b <branch name>

Add one 1 file in project (new branch)

```
$ dir
$ git status
$ git add <new file name>
$ git status
$ git commit -m “add nav bar”
$ git log –oneline
$ dir
$ git checkout <old branch>
$ git log –oneline
$ git log –oneline –all
```

Note: Switching branches changes files in your working directory.

```
$ dir
```

Changes the file data (old branch)

```
$ git status
$ git add <file name>
$ git status
$ git commit -m “update about”
$ git log –oneline --graph
$ git log –oneline –graph --all
```

$ git checkout -b <branch name>

Edit or change file data

```
$ git status
$ git add <file name>
$ git status
$ git commit -m “update main section”
$ git status
$ git checkout master
$ git status
$ git log –oneline --graph
$ git log –oneline –graph --all
```

Note: Merge

```
$ dir
```

$ git checkout -b <email-fix>

Fix the task

```
$ git status
$ git add <file name>
$ git status
$ git commit -m “Fixed problem”
```

<u>**MERGE**</u>

```
$ git checkout master (to choose merge branch)
$ dir
$ cat <file name (changed file name)>
$ git merge email-fix (fast forward)
$ dir
$ cat <file name (changed file name)>
$ git log –oneline –graph –all
```

<u>DELETE A BRANCH</u>

```
$ git branch -d email-fix
$ git status
$ git log –oneline –graph –all
```

Main section branch work continued….

Again edit or change file data

```
$ git status
$ git add <file name>
$ git status
$ git commit -m “again update main section”
$ git status
```

Main Section branch merge to master

```
$ git checkout master (to choose merge branch)
$ dir
$ cat <file name (changed file name)>
$ git merge main section
```

(Note: Merge made by the 'ort' strategy) 2 commits are merge

```
$ dir
$ cat <file name (changed file name)>
```

MERGE CONFLICTS

Merge yes or no confirm.

```
$ git branch -a
$ git branch –merge (check)
$ git branch –no-merge
```

Note: Edit or change in same file and same line with 2 branches

Process is same (add, commit).

Finally merge -------- notice: merge conflict

Choose one change and save the file

```
$ git status
$ git add <file name>
$ git status
$ git commit -m “merging”
```

REMOTE BRANCHING

Clone 1 project

```
$ git clone <HTTPS>
$ dir
$ cd <Repository name>
$ git remote
$ dir
```

Changes or edit

```
$ git status
$ git log –oneline –graph --all
$ git add .
$ git commit -m “update”
$ git log –oneline –graph --all
```

Changes the remote(GitHub) adding one file

Local not update the changes

FETCH & PULL

FETCH

```
$ git fetch origin
$ dir
$ git branch -a
$ git merge origin/master
$ dir
```

PULL

Remote add file

```
$ git pull origin master (auto adding and merge)
$ dir
$ git log –oneline –graph --all
```

PUSH

Any changes or edit

```
$ git add .
$ git commit -m “#”
$ git log –oneline –graph –all
$ git pull origin master
$ git push origin master
```

TOOLS

P4merge

(perforce application)

Set path

```
$ git config –global diff.tool p4merge
$ git config –global difftool.p4merge.path “path to p4merge”
        Ex: c:/……./……./p4merge.exe
$ git config –global difftool.prompt false
$ git config –global merge.tool p4merge
$ git config –global mergetool.p4merge.path “path to p4”
$ git config –global mergetool.prompt false
```

Git bash close and open

```
$ git difftool
$ git mergetool (save)
```

GIT ALIASES

It is using short.

```
$ git config –global alias.<name> “log –oneline –graph –all”
$ git <name>
```

GIT REBASE

```
$ git init
$ dir
$ git status
$ git add .
$ git status
$ git commit -m “#”
 $ git status
$ git allcommts
```

Create branch

```
$ git checkout -b <name>
$ dir
Changes the data
$ git status
$ git add <name>
$ git status
$ git allcommts
```

Move to master branch

```
$ dir
Changes the data another file
$ git status
$ git add <name>
$ git status
$ git allcommts
```

REBASE

```
$ git rebase <branch name>
```

GIT STASHING

Any files changes after git status notice is modified.

```
$ git stash or
$ git stash push
```

The changes are delete

Status is clean

```
$ git stash list
$ git stash apply (latest changes)
```

changes is applied but stash list is not updated.

```
$ git stash apply stash@{1} (particular stash file)
$ git stash drop (stash list updated)
$ git stash pop (apply $ drop)
$ git stash -u (untracking files)
$ git stash branch < new branch name> (new branch is created
and apply stash)
```

CLEANING

Note: Be careful

Only untracked files are deleted

```
$ git clean
$ git clean -f (only files)
$ git clean -f -d (folder delete)
$ git clean -f -d -x (.gitignore and ignore files)
$ git clean -n or
$ git clean –dry-run
```

TAGGING (MARKING)

```
$ git tag or
$ git tag -l
$ git tag –list
$ git tag –list “v1.*”
$ git tag <tag name>
$ git allcomits
$ git tag

$ git commit -am “#”
$ git tag -a <tag name> -m “#”
```

COMPARE

```
$ git diff <1st tag> <2nd tag>
```

DELETE TAG

```
$ git tag -d <tag name>

$ git allcomits

$ git tag -a <tag name> <commit id> -m “#”
$ git tag -a <tag name> -f  <update commit id> -m "#"
```

```
$ git allcomits
```

```
$ git push origin master <tag name>
$ git push origin master –tags (all tags)
```

END