1 Initial-Git

Initialize git on your machine:

```
git init
```

2 Basic-Git

Some basic git commands:

```
git add . # add unstaged file to staged
git restore <filename> # restore staged file to unstaged
git commit -m "Your_commit_message" # commit the staged file
git push # push files
```

3 Git Config

Configure commit history:

```
# global
git config --global user.name "your_name"
git config --global user.email "your_email"
# local
git config --local user.name "your_name"
git config --local user.email "your_email"
# check config lists
git config --global --list
git config --local --list
```

Global configs are located here : `~/.gitconfig` and the Local configs are located at the current directories: `.git/config`

---

4 Some Common Git Command

There are some common git commands which we use regularly

```
git branch  #to check branches
git status  #to check staged and unstaged
git checkout  #changing branch
git log  #to check the commit history
git checkout -b "branch_name"  #to create a new branch from main branch
```

5 SSH Keygen

SSH keygen is most important to connect the local server to the remote server. There are few steps to create SSH key and add it to the remote server .

5.1 Create SSH key

```
ssh-keygen -t rsa -b 2048 -C "comment"
```

This will show you a location where to save the ssh key .To save it on the default location `~/.ssh/id_rsa.pub` press ENTER, ENTER and ENTER . Here the `id_rsa.pub` is your SSH public key. Open the key, copy it and add it to github settings SSH key section .

5.2 Add To Remote

After adding the SSH key to remote , You can now connect your local machine to remote server.

5.2.1 Cloning Github/Gitlab

Go to a particular directory where you want to clone the remote git

```
git clone git@gitlab.com:group.project.git
```

5.2.2 Add Remote and Push

On the other hand, you work in your local machine and now you want to add your project to remote

```
git remote add origin git@gitlab.com:group.project.git
```

5.3 Now The Big Problem Arise

Suppose I have two GitHub accounts:

- `git@github.com:user1/project1.git`
- `git@github.com.user2/project2.git`

Earlier you saw in the previous command that i also have a GitLab account.

- `git@gitlab.com:group.projetc.git`

which i already connected to my local machine using SSH, and its public key is added in `id_rsa.pub` . Now i want to connect these two new GitHub account with SSH without any problem **Procedure**

```
#key generate for user1
ssh-keygen -t rsa -b 2048 -C "user1" -f ~/.ssh/id_rsa_user1
#key generate for user2
ssh-keygen -t rsa -b 2048 -C "user2" -f ~/.ssh.id_rsa_user2
```

Now there will be three key which are

- `~/.ssh/id_rsa.pub` which is for GitLab
- `~/.ssh/id_rsa_user1.pub` which is for GitHub user1
- `~/.ssh/id_rsa_user2.pub` which is for GitHub user2

Open a config file and add these line in it:

```
nano ~/.ssh/config #open config file
```

```
# GitLab account
Host gitlab
   HostName gitlab.com
   User git
   IdentityFile ~/.ssh/id_rsa

# GitHub account user1
Host github-user1
   HostName github.com
   User git
   IdentityFile ~/.ssh/id_rsa_github_user1

# GitHub account user2
Host github-user2
   HostName github.com
   User git
   IdentityFile ~/.ssh/id_rsa_github_user2
```

Check the HostName carefully, cause for git clone and git remote add you need to add this hostname properly. Now open and copy the each SSH key and add them to their account

```
cat ~/.ssh/id_rsa.pub  #add it to GitLab account
cat ~/.ssh/id_rsa_github_user1.pub  #add it to GitHub user1 account
cat ~/.ssh/id_rsa_github_user2.pub  #add it to GitHub user2 account
```

Now you can clone them singley and also add them in remote . You will see the HostName of config file will add after git@ part .

```
#for GitLab the HosName was gitlab
git clone git@gitlab:group/project.git

#for GitHub user1 the HostName was github-user1
git clone git@github-user1:user1/project1.git

#for GitHub user2 the HostName was github-user2
git clone git@github-user2:user2/project2.git
```

```
#for GitLab
git remote add origin git@gitlab:group/project.git

#for GitHub user1
git remote add origin git@github-user1:user1/project1.git

#for GitHub user2
git remote add origin git@github-user2:user2/project2.git
```

6 Fork Git Branch, Edit and Merge Request

Fork the main repository

```
git@gitlab.com/main/merge.git #suppose its the main repository
git@gitlab.com/fork/merge.git #forked repository
```

Clone your fork which you added to your gitlab:

```
git clone git@gitlab.com/fork/merge.git
cd merge
```

Here origin is automatically set to your fork (git@gitlab.com/fork/merge.git).

Add the main repository to upstream:

```
git remote add upstream git@gitlab.com/main/merge.git #upstream add
git remote -v #check origin and upstream set
```

Create a new branch for your feature

```
git checkout -b new-branch
```

Always work on your feature branch , never directly on master.

After completing edit, add, commit, push your feature branch to your fork(origin)

```
git push origin new-branch
```

It will create the branch in your remote fork. Then you need to create Merge Request to your fork master. Whenever merging the branch, it will automatically delete the branch from your remote fork repository. Then Create a Merge Request to the main repository from where you forked the repo.

Now there are some point:

- The commit history of main repository is ahead of your fork repository
- The commit history of your fork repository is behind of your main repository
- If you without doing anything try to push , it will shows conflict

Sync fork with main repository after Merge Request

```
git checkout master #comeback to the master
git branch -d new-branch #delete feature branch
git fetch upstream #fetch latest upstream change
git rebase upstream/master #rebase
git push  #push updated master to your fork
```

Now the commit history of your fork repository will be same as your main repository from where you forked it .

7 Resolve Conflict when two or more person changing the same line

Its from the initial phase, you can also work on a existance directory. Always create a feature branch and work on that. The scenerio is given below: In main repo there is a file1 and into the file the text is:

```
Hello,
My Name is Git.
```

And you fork the repo and clone it your local machine and make a branch named **feature** edit the file like:

```
Hello,
My Name is Git.
And You are HTML
```

So you add it and commit it. On the same time someone changed the main repo file1 like this:

```
Hello,
My Name is Git.
And You are HTML
And you love javascript
```

Now if you want to push the file to remote fork as well as merge to the main. you will see some conflict issue. There is the procedure to resolve the conflict manually. **Fetch upstream**

```
git fetch upstream
```

Checkout feature branch

```
git checkout feature
```

rebase your branch

```
git rebase upstream/master
```

You will see error like it:

```
CONFLICT (content): Merge conflict in file1
```

Now open file1 and manually changed it:

```
#from this
<<<<<<< HEAD
Hello,
My Name is Git.
And You are HTML
----------------
----------------
And you love javascript
>>>>>> feature

#to that
Hello,
My Name is Git.
And You are HTML
And you love javascript #or however you want to do it
```

Now make conflict resolved and continue rebase .

```
git add file1
git rebase --continue
```

Now You can push it

```
git push origin feature -f
```

Now from your fork feature branch you can add merge request and then delete them. And after doing these you should again do these things for make your local, fork and main repos commit history similiar so that if you want to do something there will be no problem occurs. Also do these things everytime before you work. All the procedure here.

```
git checkout feature
git fetch upstream
git fetch upstream/master
git checkout master
git push
git branch -d feature
git push origin --delete feature
```

7.1 Git Reset

We are using git reset using two types:

- Hard Reset
- Soft Reset

```
git reset --hard HEAD~2 #delete 2 commits
git push --force
```

Be careful. Hard Reset always use from the feature branch . Never use Hard reset from master or main branch. From main branch, we can use **git revert**

```
git rever commit-hash #which has you want to go
git push
```

7.2 Merge and Update

Suppose my feature branch is behind from my master branch **Checkout master**

```
git checkout master
```

Pull

```
git pull
```

Checkout feature

```
git checkout feature
```

```
git merge master
```

Push

```
git push
```