Git

* Git is an opensource, distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

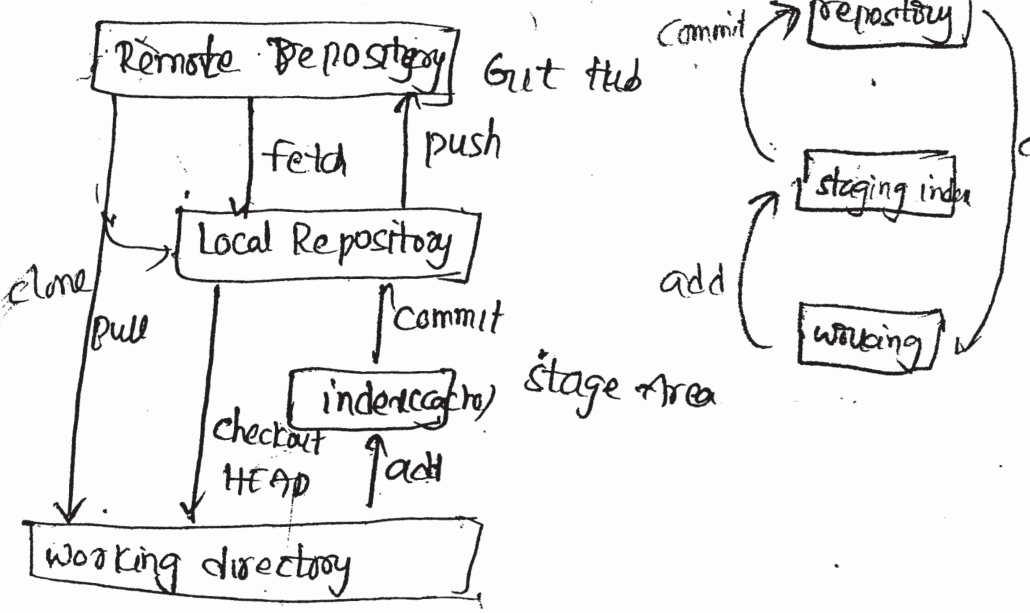
* Git is a distributed version revision control & source code management system with an emphasis on speed.

* Git was initially designed & developed by linus Torvalds for Linux kernel development. Git is a free s/w distributed under the terms of the GNU General public licence version2.

* It's commonly used for source code management, with sites like Git-Hub offering a social coding experience & popular projects such as perl, Ruby on Rails, and the linux kernel using it.

Git.

Git Data Flow



Git work-flow

* Fetch or clone (create a copy of the remote depository) (compare to cvs checkout)

* Modify the files in the local branch

* Stage the files (no cvs comparision)

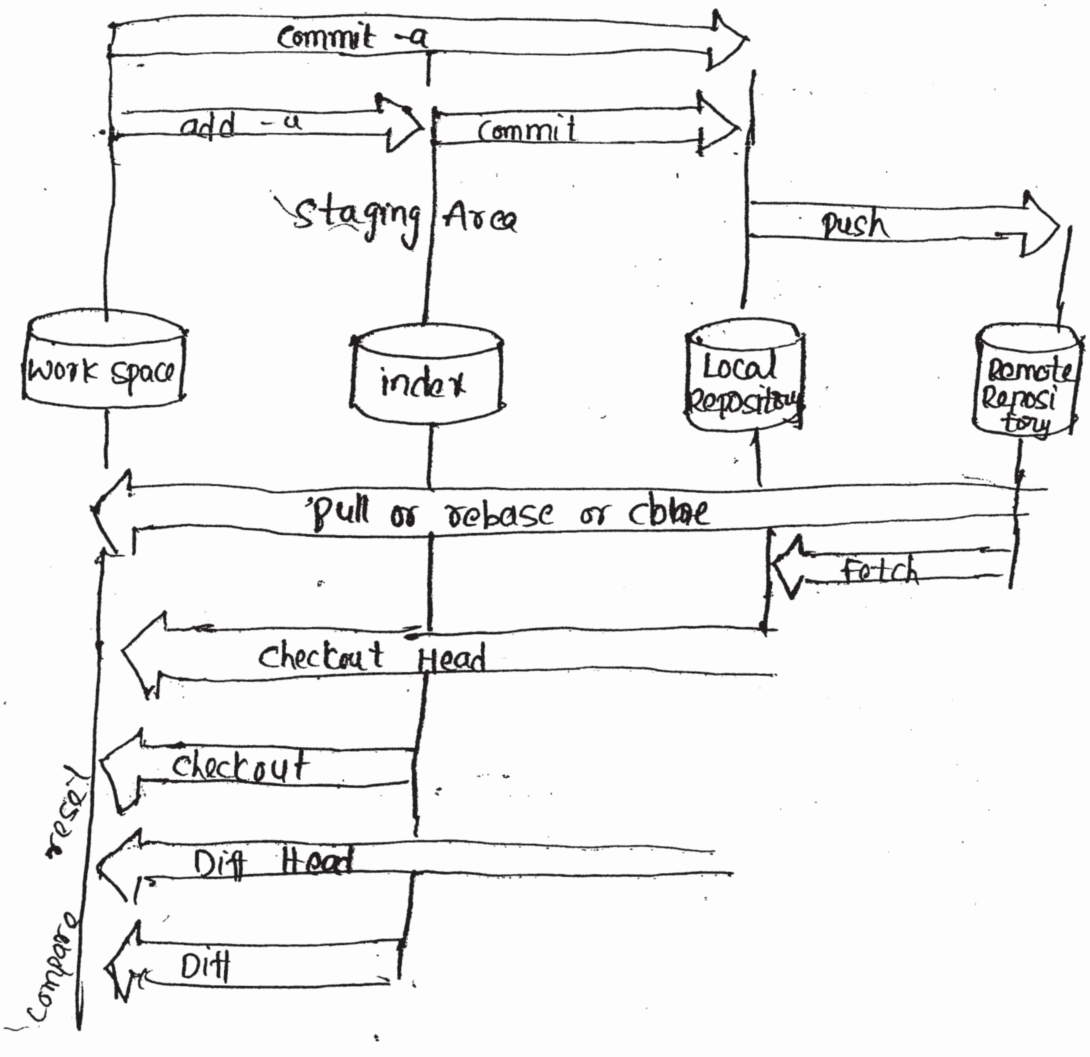
* Commit the files locally (no cvs comparision)

* push changes to remote repository (Compare to cvs commit)

Diff b/w clone & Fetch

* when we perform clone it will directly come to Directory (if ~~It~~ u ~~will~~ ~~get~~ new copy)

* when we perform fetch it will come to Global repository to Local repository.



Branch : A version of the repository that diverges from the main working project

clone : A clone is a copy of a repository or the acting action of copying a repository. when cloning a repository into another branch, the new branch becomes a remote tracking branch that can talk upstream to its origin branch.

Master : The primary branch of all repositories.. All committed & accepted changes should be on the master branch. you can work directly from the Master branch or create other branches.

Checkout : The git checkout command is used to switch branches in a repository

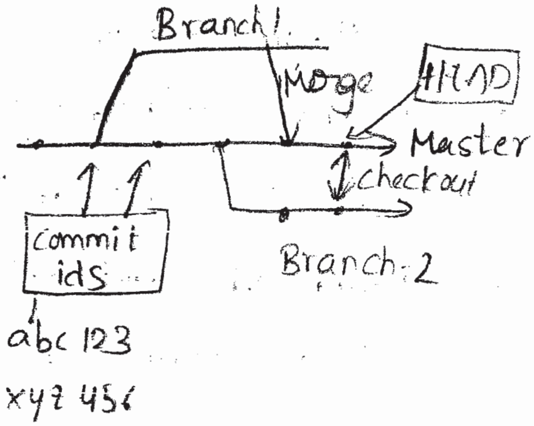
Merge : Taking the changes from one branch and adding them to another branch.

Head : Head is a reference variable used to denote the most current commit of the repository in which you are working. when you add a new commit, Head will then become that new commit push : updates a remote branch with the commit made to the current branch. your literally "pushing" your changes into the remote.

Creating account

* Initialize this repository with 'README'

Install Git on Ubuntu 14.04



step1 : Installation

```
#apt get update
#apt-get install git-core-y
#git --version
```

step2 : configuration

```
#git config --global user.name sathyadevops
#git config --global user.email sathyadevops1@gmail.com
#cat .gitconfig (or) #git config --list
```

ls -a

- Go to Git hub

Create account

start a project

* public

step3 : create GIT repository

```
#mkdir /repos
```

Give any name

```
#cd /repos
#git init
#ls -a
#git clone
https://github.com/sathyadevops/myproj.git
```

Step A : working with Git repository

```
#echo "welcome to Git">>README.md
# git status -s
```

* to add a file to cache (staging Area)

```
# git add README.md
# git status
```

* to move a file from staging Area to Local Repo

```
# git commit -m "initial commit"
```

* to Add and commit a file at a time

```
# git commit -a -m "initial commit"
```

* to push the code to central Repo (master)

```
# git push -u origin master
```

-u → optional

* To changed files in your working repository

```
#git status
```

* To show all git commits.

```
#git log
#git log -p
#git log --since = 12-03-2017 --until =13-03-2017
#git log --oneline
```

git log → commit id

git log -p → commit id along with new code & Message

optional

git log --oneline → commit id & Messages

* To made changes to tracked files

```
# git diff
# git log
# git diff 57af6S43d..9wg5C2ys3
```

→ to compare the two commit id's

cd / repos

After creating acnt go to settings → Deploy a key → we want to add a key for that we

```
# ssh-keygen
cd ~/.ssh/
ls
```

two files will be there

```
cat id_rsa.pub
```

copy pub key

Go to Git repository

Add Deploy key

Title – My key

Key - pub key paste

Add write ☑

```
cd / repos
cd / dev 6pm
```

* To list all branches

```
git remote set-url origin git@
github.com:Sathy&devops
devop
```

```
#git branch
```

To open the file

```
vi demo.c
void func()
{
 print f(" ")
 print f(" ")
}
void main()
{
print f(" ")
}
```

* To work with branches

```
# git branch branch1
#git checkout branch1
#git branch
# vi index
      new line from branch
                   optional
#git commit -a -m "new line from branch"
              optional
#git push -u origin branch1
```

* check in browser → git hub

* To merge the branch code into master

```
 #git branch
#git checkout master
#git merge branch1
#cat index.html
#git push -u origin master
```

Go to browser refresh.

* to delete a branch

```
              delete
#git branch -d branch1
```

* To delete a branch without merging the data,

```
              without merging
#git branch -D branch1
#git branch
```

```
# git tag
 git tag demo
```

```
git tag-list
vi demo.c
 add some code
```

```
git commit -am "newtag"
git push origin demo
```

* Git - Review Changes

git diff

git log

git show

c0fA55906befd1001928A8233fbb896d081e228A

* Git - Remote Server

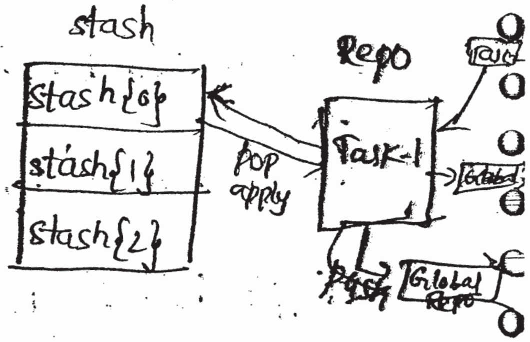
#git remote-v

#git checkout -- (to revert all the changes)

Git Stash

* git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then comeback and re-apply them later on.

* stashing is a way to pause what you are currently working on and come back to it later.



#vi index.html

<h1>Hello World </h1>

<h2>New line is added </h2>

#git diff

* stash your changes away with :

```
# git stash (or)
# git stash save "message"
# git diff
# cat index.html
      <h1> Hello World </h1>
```

* To List multiple layers of stashes

```
# git stash list
# git stash show
```

* You're back to your original working state

```
# git stash apply
# git stash apply stash@{0}
# git stash pop stash{0}
# cat index.html
      <h1> Hello World </h1>
      <h2> New line is added </h2>
```

* we can manually delete stashes

```
                   manually
# git stash drop stash@{1}
```

* delete all of the stored stashes

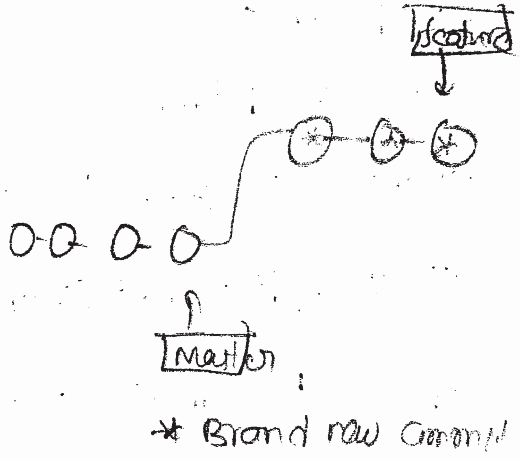
```
# git stash clear
```

This moves the entire feature

```
git log
git branch
git branch features
git branch
git checkout features
git branch
vi newfeature.txt
This is a code
git status
git add newfeature.txt
git commit -m "new feature".
git push origin feature
```

Go to browser & refresh

```
git checkout master
git rebase features
git status
git push origin master.
```



Google [illegible]

To change into ppk

load ppk open putty

Go to connection
ssh : load - add the key by using ssh-key gen method
only add private key

connection data - Auto log in username - name you provide

window
↳ Apperance
Saved sesions - demo → after this we don't want to set this path every time

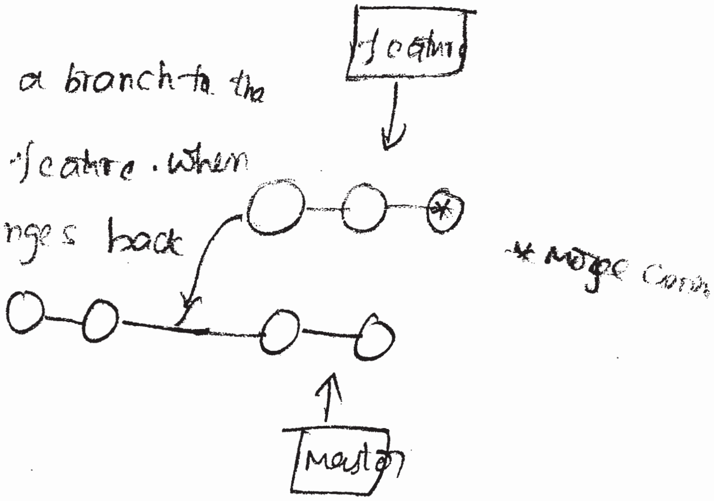
Super putty

Git merge & Rebase

Merge:

Merge takes all the changes in one branch & merges them into another branch in one commit

Let's say you have created a branch to the purpose of developing a single feature. when you want to bring those changes back to



Git rebase

git checkout feature

git rebase master

rebase exists to change the base of branch, which means it's origin commit. It replays a series of commits on top of a new base

Git.

TO Move a file to another Dir

```
#cd git proj
# mkdir mydir
# git mv demo.c mydir/
# git status -s
# git commit -m "new dir"
# git push origin master
```

TO Rename a file :

```
#git mv demo.c sample.c
# git status -s
# git commit -am "file renamed"
# git push origin master
```

TO Remove a file from git repo

```
#git rm sample.c
# git status -s
# git commit -am "file removed"
# git push origin master
```

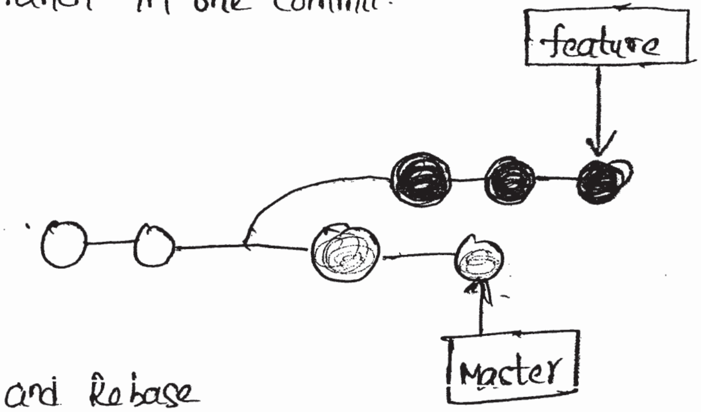
TO pull the changes from git repo:

```
# git pull
# git status -s
```

Git Merge and Rebase

The Merge Option.

Merge takes all the changes in one branch and merges them into another branch in one commit.

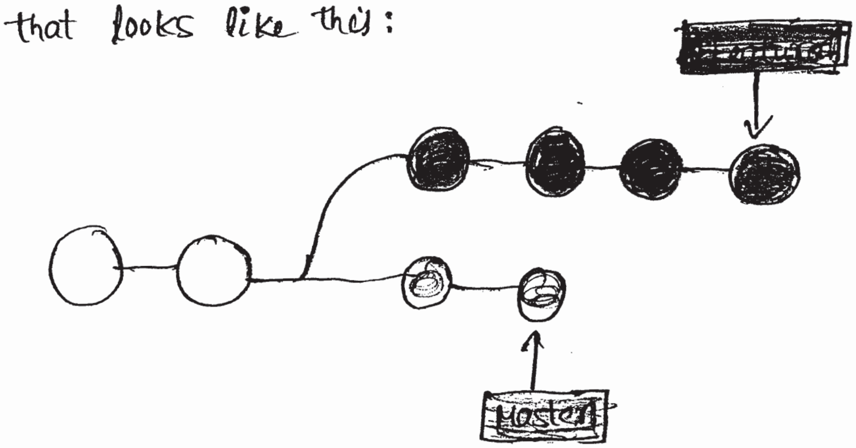


Git Merge and Rebase

let's say you have created a branch for the purpose of developing a single feature. when you want to bring those changes back to master, you probably want merge

```
#git checkout feature
#git merge master
    (OR)
#git merge master feature
```

* This creates a new "merge commit" in the feature branch that ties together the histories of both branches, giving you a branch structure that looks like this:



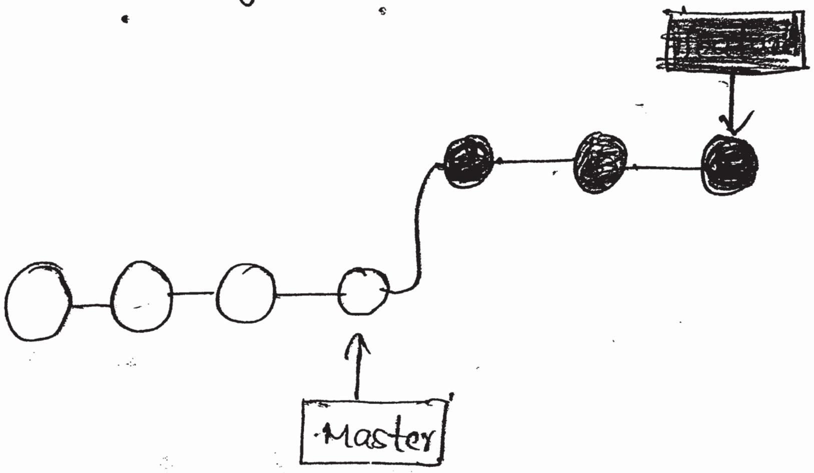
Git Rebase:

* As its name suggests, rebase exists to change the "base" of a 'branch, which means its origin commit. It replays a Series of commits on top of a new base.

* As an alternative to merging, you can rebase the feature branch onto master branch using the following commands:

```
# git checkout feature
# git rebase master
        (or)
# git rebase -i master (interactive rebase)
```

* This moves the entire feature branch to begin on the tip of the master branch, effectively incorporating all of the new commits in master. But, instead of using a merge commit, rebasing re-writes the project history by creating brand new commits for each commit in the original branch.



*Brand New Commt