**GIT**

**Overview**

Git- First Generation

1. Single file
2. No networking
3. EG SCCS, RCS

Second Generation

1. Multi Filed
2. Centralized
3. SVN, TFS, CVS

Third Generation

1. Change sets – shifted around
2. Distributed VCS
3. GIT, BitKepper, Hg

**Advantages of DVCS**

* + Centralized
    - Developers push changes to one central repository
  + Hierarchical
    - Developers push changes to subsystems-based repos
    - Subsystem repos are periodically merged into a main repo. (Linux Kernel development)
  + Distributed
    - Developers push changes to their own repo
    - Project maintainers pull changes into the official repo.
  + Backups are easy
    - Each clone is a full backup
  + Reliable branching/merging
    - Feature branches
    - Always work under version control
    - Apply fixes to different branches
  + Full local history
    - Computing repo statistics
    - Analyze regression
* New ideas
  + Deployment
* Push to a server

**Installing Git**

- Go to https://git-for-windows.github.io/

- Don’t check additional icons

- Don’t check windows explorer integration

Adjusting your PATH environment

* Use git Unix tools

Check for installation

# git --version

**Configuring Git**

1. SYSTEM-LEVEL CONFIG
   1. Git config --system
   2. Config File is stored in [ c:\Program Files(X86)\Git\etc\gitconfig]
2. USER LEVEL CONFIG
   1. Git config—global
   2. Config File is stored in [c:\users\<NAME>\.gitconfig]
3. REPOSITORY LEVEL CONFIG
   1. Git config
   2. Config File is stored in [.git/config]

**Useful Configuration commands**

1. Show the global configurations

#git config –global --list

- If not file is not present then use

#git config --global user.name “Prabuddha”

#git config –global user.email [vajrapb@Gmail.com](mailto:vajrapb@Gmail.com)

#git config –global –list

1. To change the core editor

#git config –global core.editor vim

1. To do a guess match and autocorrect your mistakes

#git config –global help.autocorrect 1

1. To color code

#git config --global color.ui auto|true|false

1. To undo changes

#git config –unset user.name

1. Change the line feed options

#git config –global core.autocrif true|false|input

Line feed options

* On windows use true
* Input on Mac or Linux

1. View configurations

#git config –global –list

OR #cat .gitconfig

**Working Locally with Git**

1. Create a local repository
   1. On an empty directory

#git init

* 1. Add a file to the empty directory

#echo “Hello Git” > Readme.txt

#echo “Hello Git” >> Readme.txt

* 1. Check the status of the repository

#git status

1. Adding files to the staging area

#git add Readme.txt

* 1. Check the status of the repository

#git status

1. Commit changes to the repository

# git commit

* 1. add a comment to the vim logs

**Viewing History**

1. View history with:

# git log

* + 1. you can use the sha1 hash use abbreviated 5-8 first characters.

1. Update the Readme.txt file
2. Check the repos status

#git status

1. Add all the updated files using -u option

#git add -u

1. Check the status of the repository

#git status

1. Commit changes to the repository with message

#git commit –m “message in quotes”

1. Check the status of the repository

#git status

1. View history with:

# git log

* Recent at the top and oldest at the bottom

**Viewing Diff**

1. View Diff with

#git diff 023ref 054324(newer)

* Compare older version to newer version

#git diff HEAD~1..HEAD

This means from last commit to the latest commit

#git diff HEAD ~1..

This assumes last commit to the latest commit if the 2nd argument is not given

**Staging changes as multiple files**

1. Create file1.txt

#touch file1.txt

1. Create file2.txt

#touch file2.txt

1. Check the status

#git status

1. Add all the updated files

#git add -u

* This will only work on already tracked files

#git add -A

* This will add all files

1. Commit the changes to repository

#git commit -m “Added file1.txt and file2.txt”

1. Check the log

# git log

1. Check the diff between the last commit and one before

#git diff HEAD~1

1. Edit file1.txt

#vim file1.txt

1. Edit Readme.txt

#vim README.TXT

1. Check the status

#git status

1. Add file 1 to staging

#git add file1.txt

1. Check the status

#git status

1. Commit changes to repository

#git commit -m “edited file1.txt”

1. Add file 1 to staging

#git add Readme.txt

1. Check the status

#git status

1. Commit changes to repository

#git commit -m “edited Readme.txt”

**Deleting files**

1. Check the status

#git status

1. Delete file2.txt

#rm file2.txt

1. Check the status

#git status

1. Add all the updated files

#git add -u

1. Create file3.txt

#vim file3.txt

1. Check the status

#git status

1. Add file3.txt to staging

#git add file3.txt

1. Rename file1.txt to newfile1.txt

#mv file1.txt newfile1.txt

1. Check the status

#git status

1. Add all files to staging

#git add -A

1. Check the status

#git status

1. Commit changes to repository

#git commit -m “removed file2.txt added file3.txt renamed file1.txt”

1. Check the log

# git log

**Undo your changes**

1. Make changes to Readme.txt

#vim Readme.txt

1. Undo Changes

#git checkout Readme.txt

1. Check the status

#git status

1. Check the Readme.txt

#cat Readme.txt

1. Make changes to Readme.txt

#vim Readme.txt

1. Delete newfile1.txt

#rm newfile1.txt

1. Check the status

#git status

1. Hard reset to last commit

#git reset –hard

* Remove all the changes to last commit

1. Check the log

# git log

1. Make changes to Readme.txt

#vim Readme.txt

1. Do a soft reset

#git reset –soft HEAD~1

* This will remove commits to staging and move back to the previous commit

1. Check the status

#git status

1. Commit changes to repository

#git commit -m “soft reset and reogranized”

1. Check the status

#git status

1. Check the log

#git log

1. **Do a hard reset**

**#git reset --hard HEAD~1**

* **This is going to move my head to last commit and discard all changes.**

**Cleaning local repo**

1. Create 2 new files

#touch temp1.txt

#touch temp2.txt

1. What would you clean

#git clean -n

1. Clean the repo

#git clean -f

1. Check the status

#git status

**Ignoring files with gitignore.**

1. Create a Directory

#mkdir logs

1. Create 2 log files

#touch log.log

#touch log.txt

1. Open the .gitignore file

#vim .gitignore

1. Add the directory or files

~/logs

~/logs/\*.txt

~/logs/\*.logs

1. Check the status

#git status

1. Add file 1 to staging

#git add .gitignore

1. Commit changes to repository

#git commit -m “changed .gitignore”

1. Check the status

#git status

1. Check the log

# git log

**WORKING REMOTELY WITH GIT**

**Cloning a remote repository**

1. Clone the jQuery Repository

<https://github.com/jquery/jquery>

* 1. - Clone or download dropdown
  2. - use https link

1. https://github.com/jquery/jquery.git
2. Create a new directory
3. Clone the JQuery Repo
   1. #git clone <https://github.com/jquery/jquery.git>
4. Go into the Jquery Directory
5. Check the logs

#git log

1. One line log option

#git log –oneline

1. One line log option with graph

#git log –oneline –graph

1. Short Logs with just authors and commit messages

#git shortlog

1. Short Logs with summary, ordered numerically and include users email

#git shortlog -sne

* We can also look at GitHub for statistics
* Git insight

1. We can also look at the last commit
   1. #git show HEAD
   2. #git show HEAD~1
   3. #git show 5642645

**Listing remote repo**

1. Check which branch you are on

#git remote

#git remote -v

1. List the contents of directory

#ls -la

1. view .git/config file

#cat .git/config

* We are working on remote origin and branch master

1. To see the local branch we are working on
   1. #git branch
2. To see the remote branch
   1. #git branch -r
3. To see tags or stable versions of your repos/builds
   1. #git tag

GIT PROTOCOLS

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Protocol | 1. Port | 1. Ex | 1. Notes |
| 1. HTTP(S) | 1. 80/443 | 1. <https://github.com> | 1. Read Write 2. Password For Auth 3. Firewall Friendly |
| 1. Git | 1. 9418 | 1. Git://github.com | 1. Read Only 2. Anonymous only |
| 1. ssh | 1. 22 | 1. [git@github.com](mailto:git@github.com) | 1. Read-Write 2. SSH for auth |
| 1. File | 1. n/a | 1. /user/names/ | 1. Read-Write 2. Local Only |

**Fetching changes from remote**

1. Create a github repo with your local test directory

1.Switch back to your local test directory

2. Check for remote branch

#git remote -v

1. Add a remote destination
   1. #git add remote add origin https://github.com/...
2. Remove a remote destination
   1. #git remote rm origin
3. Push existing repo to remote

#git push origin master

1. Pull down any changes from remote repo
   1. #git fetch
   2. #git fetch origin
      1. It you have multiple remotes
2. Create a remote file and commit it
3. Check the logs

#git log

1. Check the logs for remote

#git log origin/master

**Merging changes**

1. Merge the changes
   1. #git merge origin/master master
2. Check the logs

#git log

1. Check the remote branch we merged from

#git branch -r

**Pulling from a remote**

15.What we did earlier was fetch the changes and merge to origin/master. To do this there is a short cut.

#git pull

1. We have to setup tracking branch here. Tracking branch is the remote repo which local is going to mirror
   1. #git branch --set-upstream-to origin/master
   2. 17. Do a git pull now
      1. #git pull

**Pushing changes remotely**

Create and Edit the readme file

Git Status

Git commit

#git commit -m “2nd local commit. Added and edited Readme.txt”

#git status

Branch is ahead of origin/master

#git push

Add the user name

Add the password

This is not a good practice

Remove the origin

#git remote rm origin

Check for remote repo

#git remote -v

Add the ssh version of the URL

#git remote add origin [git@github.com:vajrapb/gitfundamentals.git](mailto:git@github.com:vajrapb/gitfundamentals.git)

-uses ssh keys

Add the ssh keys to your git

#git push

#git push –set-upstream origin master

**Working with tags**

**Tagging your repo**

#git tag

#git tag V1.0

#git tag

#git log – commits will point to the stable commit

#git tag -a V1.0 -m”with message”

#git push

Git will not push tags by default

#git push –tags

Go to github and check using tags

-Stable points in the git server.

**Branching, Merging and Rebasing**

1. **Visualizing branches**
   1. **We used** 
      1. **#git log –graph –online**
      2. **to visualize with graph and with oneline options**
   2. **We will use**

**#git log –graph –oneline –all --decorate**

* + 1. **To visualize with one-line option, all branches and apply labels**
    2. **It is cumbersome to type this all the time so we will use git config.**

**#git config –global alias.lga “log –graph –oneline –all –decorate”**

**You can also directly add it to .gitconfig file in your home directory**

**#git lga**

1. **Creating local branches**
   1. **#git branch feature 1**
   2. **#git lga**
   3. **Git checkout frature 1**
   4. **#git lga**
2. **Add modifications to readme**
   1. **#echo “Feature 1” >> Readme.txt**
   2. **Git status**
   3. **Git commit -am “Feature 1 added”**

**Git lga**

**Now feature 1 and master have different branch**

1. **Checkout master**
   1. **#git checkout master**
   2. **#git lga**
2. **Branch off of 3rd commit**
   1. **#git branch fix 1 0328cbd**
      1. **Usually you will create fix branches from well-known tags**
   2. **Git lga**
   3. **Git checkout fix 1**
   4. **Git lga**
   5. **echo "Fix 1 applied" >> Readme.txt**
   6. **git commit -am "6th commit locally on fix1 branch"**
   7. **git lga**
   8. **git checkout master**

**Rename a branch**

**#git branch -m fix1bug1234**

**#git branch -d fix1bug1234**

**It will not let you deleted due to not being merged to master**

**#git branch -D fix1234 to force delete it**

**Add a new branch**

**#git branch feature2 or git checkout -b feature2**

**#git lga**

**#edit Readme.txt**

**#git commit**

**#git lga**

**Undo Branch deletes**

**#git reflog**

**#git branch fix1bug1234 (commit sha)**

**Log of where head has pointed**

**Git by default keeps it around 30 days**

**#Git checkout feature 2**

**Echo “Feature 2 changes” >> Readme.txt**

1. **Stashing Changes**

**Git status**

**Git stash**

**Git status**

**Cat Readme.txt (changes are gone)**

**Git stash list (holding area)**

**Git checkout fix1**

**Git branch**

**"Another fix for fix1" >> Readme.txt**

**Cat readme.txt**

**Git checkout fix1**

**Git stash apply- to get back to your stash**

**Git lga**

**Git stash list**

**Cat Readme.txt**

**Git reset –hard HEAD**

**Git status**

**Cat Readme.txt**

**Git stats pop**

**Pops the top item off of the stash and applies to the working copy. It removes from stash list as well**

**Git stash**

**echo "More Changes" > Additionalfile.txt**

**Git add Additionalfile.txt**

**Git stash**

**Git status -working copy is clean**

**Git stash list**

**Git stash drop**

**Drops the first index**

**Git stash list**

**Git stash branch feature2\_additional**

**Git stash list**

**git commit -am "8th commit locally additional features to feature 2"**

**Merging**

**Git checkout master**

**Git merge feature 1**

**Fast forward move without any changes**

**Git branch -d feature1**

**Git lga**

**Git merge feature2\_additional**

**Auto-merging Readme.txt**

**CONFLICT (content): Merge conflict in Readme.txt**

**Automatic merge failed; fix conflicts and then commit the result.**

**Cat Readme.txt**

**Manually Resolve with text editor**

**Or git mergetool**

**git diff –cached**

**shows difference between repo to the staging area**

**git commit -am**

**rm Readme.txt.orig**

**Feature 3 and Rebasing**

**Git branch feature3 V1.0**

**>git checkout feature3**

**Git lga**

**Edit file1.txt**

**Git lga**

**git commit -am" 10th commit added feature 3"**

**We need to rebase it now to avoid conflicts with master and feature 3**

**Git rebas master**

**Git lga**

**Git checkout master**

**Git merge feature3**

**Git lga**

**Rebasing doesnot always go as planned**

**Git checkout fix1**

**Git rebase master**

**Git mergetool kdiff3**

git config --global --add merge.tool kdiff3

git config --global --add mergetool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"

git config --global --add mergetool.kdiff3.trustExitCode false

git config --global --add diff.guitool kdiff3

git config --global --add difftool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"

git config --global --add difftool.kdiff3.trustExitCode false

**Fix the conflict**

**Git status**

**Git rebase –continue**

**Git checkout master**

**Git merge fix1**

**Git lga**

**Git branch -d feature 3**

**Git lga**

**Git branch -d fix1**

**Git lga**

**Git branch -d Feature2\_additional**

**Git branch -d feature2**

**Cherry-Picking changes**

**Git lga**

**Git branch V1.0 fixes V1.0**

**Git checkout V1.0\_fixes**

**Echo “Fix 1” >> file 1.txt**

**Git commit**

**Echo “Fix 2” >> file 2 .txt**

**Git commit**

**Now you need one of these fixes on master branch**

**Git checkout master**

**Now you want to merge but you only want commit from that branch**

**You want to grab fix 1**

**Rebase doesnot also work coz you will be applying whole piece.**

**Option**

**#Git merge sha (that particular commit)**

**#git rebase sha**

**But you only want that 1 commit**

**#** **git cherry-pick a9fb4f7**

**It grabs only one commit**

**Not do a merge to the branch and see**

**It will not try to reapply the commit**

**Pushing Changes to origin/master**

**Git push**

**Git push orign V1.0 Fixes**

**Git branch -r**

**Deleting remote branches use caution**

**Git lga**

**Git push origin V1.0 fixes:V1.0Fixex\_remote\_branch**

**Git push origin :V1.0\_fixes\_remote**