**Git fundamentals**

* **Introduction to Git**
* **Working locally with Git**
* **Working Remotely with Git**
* **Branching, Merging and rebasing with Git**
* **Troubleshooting**

**Module1: Introduction to Git**

* Created by Linus Torvalds
* Written perl and c
* Run in all OS
* Design goals:

1. Speed
2. Simplicity
3. Branch and merge support
4. Distributed
5. Scales well for large project

**Installing git:**

Windows:

<https://gitforwindows.org/>

git --version (cmd prompt)

cd git

mkdir test

get init

note: assure that git is working fine

Linux:

Apt-get install git-core (debian, Ubuntu)

Yum install git-core (Fedora)

cd git

mkdir test

get init

note: assure that git is working fine

**Git configuration**

1. System level configuration

2. User level configuration

3. Repository level configuration

Git config --global --list

Git config –global user.name=”Elakkia Manickam”

Git config –global user.email=”elakkiasuganthi@gmail.com”

Git ~/.gitconfig

(Just name value pair)

Git config –list (all operations)

Git config –unset user.name

Vim .git/config

(Editing)

**Module 2: Working locally with Git**

**Overview:**

* **Creating a local repository, adding files, and committing changes**
* **Viewing history and diffs**
* **Staging changes as multiple commits**
* **Deleting and renaming files**
* **Undoing changes to the working copy**
* **Undoing/redoing changes in the repository**
* **Cleaning the working copy**
* **Ignoring files with .gitignore**

#create the folder that you want to use in the git

cd code

git init

(it will create .git folder and add all config files)

#add content to README.txt

Echo “Hello git”>>README.txt

#check status

git status

#not tracked yet

#add to the repo

Git add README.txt

Git status

#(git knew it is new one and added to repo)

Git commit

#(new file will come and edit your commit statement )

Git log (history) (author and date info)

#adding second line in readme

Echo “hello git ”>>README.txt

Git status (git know it README.txt changed)

Git add –u (ADD all files updated)

Git commit –m “updated README.txt” (No need of going text file)

**Viewing history and diffs**

Git log (note:2 commits added.. most recent at the top)

#to find

Git diff commit-no commit-no git diff afds … 5466

#latest commit is HEAD in git

Git diff HEAD~1..HEAD

#(~1 means one commit before head)

Create 2 more file in code folder

Git status

Git add –u

Git status

(Note: add –u will add only updated file and deletion -d)

#add explicitly by name or use –A option

Git add –A

(add all even untracked) (yet to be commit)

Git add –m “ADDED new feature”

**Staging changes as multiple commits**

#if you have change two files 2 different changes and you want to commit one by one. You can do it easily in git

Echo “fix file 1 for the bug 1234”>>file1.txt

Echo “updated file”>>readme.txt

Git add file1.txt git add -A

Git status

(1 to commit and one not added to repo yet)

Git commit –m “fix budg “

Git add README.txt

Git commit –m “change in read”

(Note: Staging is important.. if you want to pull some of the things . but if you mix together then it will tough to pull back)

**Deleting and renaming file**

Rm file2.txt

Vim file3.txt

Mv file3.txt newfile.txt

Git add –A (add new and deletion)

Git status

Git commit –m “Reoragnizing”

**Undoing changes to the working copy**

#if you change something that you don’t want

Vim README.txt

#delete all

Git status

(OOPS , NOT DONE BY ME)

Git checkout README.txt

Git status

(Working fine, get back now)

#if you more than 1 file

Git reset –hard

(Bunch of reset will come back)

Git status

**Undoing/redoing changes in the repository**

#if you want redo particular state again

#reorganizing if I want to remove

Git reset --soft HEAD~1

#omitted now reorganization

#look at the git status, now it moved to staging area

Git status

#now you can do correction

#make the fixes and recommit

#now you can commit if you want

To delete and change and roll back

Git reset --hard HEAD~1

Git reset –hard HEAD~2 (ROLL back until 2 commit before) (HARD will totally roll back no in status also you cant see)

**Cleaning the working copy**

#if you have two files but you don’t to have it here

Touch file1.txt file2.txt

Git status

rm file1.txt file2.txt

(Note: 2 files not added to repo)

#if you want to remove bunch of temp file

Git clean

Git clean –f (remove it forcefully)

Git status

(Removed successfully)

**Ignoring file with .gitignore**

#logs are changing as based on application so I don’t want to commit

Mkdir logs

Tough logs/log1.txt

Git status

(Note: - .gitignore specify file you don’t want to commit )

Vim .gitignore

(note: anywhere in application logs/ )

(note: absolute path /logs/\*.txt,/logs/\*.log,/logs )

Git status

(note : git ignore file)

Git add .gitignore

Git commit –m “ignore “

Git status

**Module 3: Working Remotely with Git**

* **Cloning a remote repository**
* **Listing remote repository**
* **Fetching changes from a remote**
* **Merging changes**
* **Pulling from a remote**
* **Pushing changes remotely**
* **Working with tags**
* **Summary**

**Cloning a remote repository:**

Cloning of remote repository to local machine

Example: jQuery repository on Git hub

git hub jQuery page link : <https://github.com/jquery/jquery>

git hub jQuery source link : https://github.com/jquery/jquery.git

commands :

#cloning (Entire history will be downloaded)

git clone https://github.com/jquery/jquery.git

#go to path

cd jquery

#to see list of commits

git log

#to see all commits in line by line

git log --oneline

**Basic repository statics**

Commands

#to find total number of commits done in that project

git log –oneline | wc –l

#to see different branches merges happened (left hand side)

git log –oneline --graph

#to get the log detailed list of authors and commit messages details

git shortlog (git log --format=short)

#getting statics s = summary n =number of commit e =email address

git shortlog -sne (output : 45 elakkia [elakkiamanickam@outlook.com](mailto:elakkiamanickam@outlook.com))

**Viewing commits**

#what is the last commit made in jQuery

git show head

#reverse the commit

git show head~

#to find total number of commits done in that project

git log –oneline

(note :last two one was reverted one )

git show head~10

git show 567456 (commit number)

#show the origin

git remote

#show the link for fetch and push url

git remote –v

**Git protocols**

**http(s) 80/443**

**git 9418**

**ssh 22**

**file n/a**

**Viewing branches and tags**

#display local branch

git branch

#display remote branch

git branch –r

#tags used by jQuery team

git tag

**Fetching from a remote**

#switch to git fundamentals repository

cd GitFundamentals

ls

git logs (commit added git ignore)

#this is the local repository not communicating with other repo

git remote –v

[Note: to remove remote branch

git remote rm origin ]

git remote add origin <https://github.com/elakkia-manickam/git-fundamentals.git>

https://github.com/Elakkia-Manickam/Git-Fundamentals.git

#fetch remote changes

git fetch

git fetch origin

git log origin/master

(note : you can see the commits done by others also but not in out local working copy )

#to get that remote in our local repo

git merge origin/master

**Pulling from a remote**

#to get merged directory

git branch –r

#to fetch origin and merge

git pull (note : there is no connection to remote)

#track remote branch from origin

git branch –set-upstream master origin/master

git pull

**Pushing to a remote**

Vim Readme.txt

Git status

Git commit –am “sharing”

Git status (branch one step ahead of ‘origin/master ‘ 1 commit)

Git push (need username password)

#remove origin

Git remote rm origin

#check remote

Git remote –v

#add

Git remote add origin [git@github.com:Elakkia-Manickam/Git-Fundamentals.git](mailto:git@github.com:Elakkia-Manickam/Git-Fundamentals.git):

(note it will use ssh key to authenticate) (will not ask for the password)

Updated in your account after push

**Creating and verifying tags**

#craete tag like v1.0

#set the tag

git tag v1.0

#view tag

git tag

#creatimg tag with message

Git tag –a v1.0\_with\_message (file will open write your message)

#signing tag (automatically need msg)

Git tag –s v1.0\_signed (file will open write your message)

Git tag

Git tag –v v1.0\_with\_message (not verified)

Git tag –s v1.0\_signed (verification need in public repo also)

**Pushing tags to remote**

**#**git push (output up to date)

#updating tag in remote

Git push --tags (codes at the time of tag made)

**Module 4: Branching, Merging and rebasing with Git**

* **Working with local branches**
* **Stashing changes (mightn’t commit write now save for latter)**
* **Merging branches**
* **Rebasing branches**
* **cherry-picking commits**
* **Working with remote branches**
* **Summary**

**Visualizing Branches**

#to visualize branches

Git log --graph --oneline -- all –decorate (git log --graph --oneline --all --decorate)

# add this as alias in git config

Git config –global alias.lga “log --graph --oneline -- all –decorate ”(log graph all)

Git lga

Cat ~/.gitconfig

**Creating local branches**

**#**add new feature branch

Git branch feture1

Git checkout feature1 (switched to branch)

Git lga (you can see 2 branches master and feture)

(branches are lables for individual commits )

echo “feature1” >>README.txt

git status

git commit –am “added feature 1”

**Difference between branches and Tags**

Brach: it will follow the commit if you want to fix bug commit it to new one.

Git branch fix1 (old commit no)

Git checkout fix1

Echo “fixing bug #1234” >>README.TXT

Git commit –am “Fixed bug#1234”

Git lga

(now brahes will be pointed to the old commit )

**Renaming and deleting branches**

#Renaming

#check log

Git lga

#go to master

Git checkout master

#change the name

Git branch –m fix1 bug1234

#renaming successful

Git lga

#deletion

Git branch –d bug1234

#force deletion

Git branch –D bug1234

Git lga

#add branch and checkout at the same time

Git checkout –b feature2

#switches to feature 2

Echo feature2 >>README.txt

Git commit –am “Added feature2”

#now we can see feature 1 and feture2 of our master branch

**Recovering deleted commits**

Git reflog (all the references )

Note that branch commit number

Git branch bug1234 commitno

Git lga

(Branch back)

Git checkout bug1234

Git show head

#keep record commit 30 days after it will clean

#reflog to get deleted info back

**Stashing changes**

git checkout feture2

Echo “fetaure2 change” >>README.txt

git status (commit not clear)

git stash

git status (commit clear now)

(Readme file don’t have addition)

Git stash list (stash list shown here)

git checkout bug1234

echo “Another bug fix ” >>README.txt

git commit –am “bug fixing”

git checkout feature2

git stash apply

#now old feature2 readme changed

Git stash list

#roll back stash (note it stash to store data.. apply using git stash apply and roll back and store again in stash using below)

Git reset –hard head

Git status

Cat readme.txt

#clean stash list and roll back (clear and delete store in stash)

Git status pop

#add one more file and save it in stash

Git stash list (2 changes)

Echo “ADDITIONAL FILE” >>additional.txt

Git status

(Note file should add)

Git add addtional.txt

Git stash

Git status (working copy clean)

Git stash list (2 pending changes)

#rollback that addtiona.txt delete

Git stash drop

Git stash list (one pending change)

#if you need a new branch for this change

Git stash branch feture2addiotional

#stash list (Stash pop out from here…added under branch)

Git commit –am “added additional one”

**Merging Branches**

Git lga

Git merge feature1 (FAST ORWARDING merge)

(Note: head moved to feature1)

Git branch –d feature1

(Note:featurwe1 integrated into master)

Git lga

#merge 2 nd branch

Git merge feature\_addtional

(note :merge conflicts)

Git merge tool

(note:kdiff3)

Git status

Git diff --cached

Git commit –m “Feature 2 additional”

Rm README.txt.origin

**Rebasing changes**

Git branch feature3 v1.0

Git checkout feature3

Vim fiel1.txt

git commit –am “added feature3”

git lga

git rebase master (relocate it on top f master)

git checkout master

git merge feature3

(Fast forwarding merge)

Git Checkout bug1234

Git merge master (merge it with bug1234)

(merge conflicts came )

Edit using tool

After edit

Git status

Git diff –-cached

Git rebase –continue

(conflict)

Use merge tool

Git rebase –continue

Git status

Git lga

Git checkout master

Git merge bug1234

Git branch –d bug1234

Git branch –d feature3

Git branch –d feature2

Git branch –d feature2addtional

(After the successful rebase merge it )

(delete the extra branch)

**Cherry picking changes**

#another technique

Git lga

Git branch v1.0fixes v1.0

Git checkout v1.0fixes

Echo “fix1”>>file1.txt

Git commit –am “added fix1”

Echo “fix2’>>file2.txt

Git commit –am “added fix2”

Git checkout master

(Note: don’t want to merge but want to apply fix1 changes in master)

Git cherry-pick

(it select one commit and apply it (command-number) )

Git lga

(patches – like you want to take some commit and apply it on another commit )

Git merge v1.0fixes

Git lga

(it know that fix1 already added)

(git know about that files already added it added only the exclude commits)

In cherry-picking know where it from all.Merging and rebasing easy

**Creating a remote branch**

Git lga (master far behind the origin)

Git fetch origin master

Git lga

(to push changes )

Git push

(Master fast forwarded )

(only one branch in dashboard)

#If we want to push fixv1.0

Git push origin v1.0fixes

(it will have one branch in dashboard)

Git branch –r

(Remote branches will display here)

**Deleting a remote branches**

Git push origin :v1.0fixes

(note :git push origin “don’t specify local branch name leave it empty” : “specify the remote branch”)

**Troubleshooting**

**Refusing to merge unrelated histories**:

Fatal: refusing to merge unrelated histories

1. As clear from the error message, it happens when someone try to merge two unrelated projects (projects which do not know about each other and commit history in one project does not match with other project.). Until now I am aware of two scenarios when this error can occur.

2. You have create a brand new repository, added few commits and now you are trying to pull from some remote repository which already has some commits.

cmd

git pull origin master --allow-unrelated-histories

<https://allaudin.github.io/git-unrelated-histories/>

**GitHub push declined due to email privacy restrictions:**

Unchecking the "Block command line pushes that expose my email" box in your email settings and then pushing again.

