Git

=============================================================

Day 1

=============================================================

Version Controlling

============================

This is the process of maintaining different versions of the code

All the team members upload their code into the remote version

controlling system. The VCS accepts the code uploads from multiple

team members and creates an integrated project. Next time when the

team members download the code they will be able to see all the

code created by the entire team. The process of uploading into a

VCS is called as checkin and downloading is called as checkout.

VCS's also preserve older and later versions of the code so that

at anypoint we can jump between whichever version that we want

VCS's also keep a track of who is making what kind of changes

VCS's are of two types

1 Centralised Version Controlling System

2 Distributed Version Controlling System

Centralised Version Controlling System

---------------------------------------------

Here we have a remote server where all the team members upload the code

and version controlling happens only on that remote server

Distributed Version Controlling System

------------------------------------------------

Here we have a local repository installed on every team members machine and

version controlling happens here at the level of individual developer,

from the local repository the code is uploaded to the remote repository

where version controlling happens at the level of the entire team

Installing git on Windows

=================================

1 Open https://git-scm.com/downloads

2 Download git for windows

3 Install it

4 Once git is installed we get an application called gitbash

where we can fire the git commands

Installing git on Ubuntu Linux

====================================

1 Connect to a ubuntu linux

2 Update the apt repository

sudo apt-get update

3 Install git

sudo apt-get install -y git

Installing git on RHEL, Centos, Fedora

===========================================

1 Update the yum repository

yum -y update

2 Install git

yum install -y git

-------------------------------------------------------------------

Setting up username and email globally for all users on a system

git config --global user.name "sai krishna"

git config --global user.email "intelliqittrainings@gmail.com

To see the list of configurations done

git config --global --list

----------------------------------------------------------------------

Git uses three sections when working on the local machine

Working directory

Staging Area

Local Repository

Working Directory is the location where the team initially stores the

data and all the files present here are initially called untracked files

Staging Area is the intermediate buffer zone of git where the files

are sent into and these files are called as stagged files

Local Repository is the location where the files are moved into from the

staging area and the files here are called as commited files. Version controlling

happens only when the code comes into the local repository

1 To initialise the current folder as a git repo

git init

This command will create a hidden folder called as ".git" where it

stores all the configurations related to git

2 To send a files from working directory to staging area

git add filename

To send multiple files to stagging area

git add file1 file2 file3

To send all the files and folders from working dir to staging area

git add .

4 To send the files from staging area back to working directory

git rm --cached filename

or

git reset filename

5 To send files from staging area to local repository

git commit -m "Some message"

6 To see the status of untracked and staged files

git status

7 To see the list of version sent into local repository

git log

.gitignore: This is a special configuration file which is used to hide the private files info. Any file whose name is mentioned in .gitignore will no longer be accessed by git

1 Create few files

touch f1 f2 f3 f4 f5

2 Check the status of git

git status

This will show the above 5 files as untracked files

3 Create .gitignore and store the above 5 file names in it

cat > .gitignore

f1

f2

f3

f4

f5

To come out of cat command press ctrl+d

4 Check the status of git

git status

This will only show .gitignore as untracked

f1-f5 are no longer accessible by git

Day 2

==============================================================================

Branching in Git

========================

This is a feature of git using which we can create separate branches for different functionalities and later merge them with the main branch also known as the master branch. This will help in creating the code in an uncluttered way

1 To see the list of local branches

git branch

2 To see the list all branches local and remote

git branch -a

3 To create a branch

git branch branch\_name

4 To move into a branch

git checkout branch\_name

5 To create a branch and also move into it

git checkout -b branch\_name

6 To merge a branch🡪 first move to branch in which you want to merge

e.g git checkout master

git merge branch\_name

7 To delete a branch that is merged

git branch -d branch\_name

This is also called as soft delete

8 To delete a branch that is not merged

git branch -D branch\_name

This is also known as hard delete

Note: Whenever a branch is created whatever is the commit history of the

parent branch will be copied into the new branch

Note: Irrespective of on which branch a file is created or modified git only

considers from which branch it is committed and the file belongs to that

committed branch only.

============================================================================

Working on the Github

=========================

This is the remote repository into which the code is uploaded and

this process is called as checkin

1 Signup for a github account

2 Signin into that account

3 Click on + on top right corner

4 Click on New repository

5 Enter some repository name

6 Select Public or Private

7 Click on Create repository

8 Go to Push an existing repository from command line and copy paste

the commands

Enter username and password of github

=========================================================

To download the code from the remote github we can use the following

2 ways

1 git clone

2 git fetch

3 git pull

=========================================================

git clone

==============

This is used to download the entire remote git repository into the

local machine. This is generally used only one time when all the team

members want a copy of the code present in the git remote server

git clone remote\_git\_hub\_url

git fetch:

==============

This will work only when the code present in the remote server is different from the code present in the local repository

git fetch will download only the modified files and it will place them on a separate branch which is called as "remote branch"

We can go to this remote branch and check if the modifications are acceptable, if so we can merge them with the master branch

1 Open github.com

2 Click on the repository that we updated

3 Select a file to be modified and click on Edit icon

4 Make some changes and click on commit changes

5 Open git bash

git fetch

7 This will download all the modified files and place them on a

remote branch

8 To see the list of all the branches

git branch -a

9 Move into the remote branch

git checkout remote\_branch\_name\_from\_step8

10 Check if the modifications are acceptable if so move to master and merge

git checkout master

git merge remote\_branch\_name\_from\_step8

git pull

==============

This will also download only the modified files but it will

merge them with the master

1 Open github.com

2 Click on the repository that we updated

3 Select a file to be modified and click on Edit icon

4 Make some changes and click on commit changes

5 Open git bash

git pull

We will be able to see the modified files directly on the master

=========================================================

Day 3

=========================================================

Git reset

===============

This is used to move between multiple versions of the code

This is of 3 types

1 Hard reset

2 Soft reset

3 Mixed reset

Hard Reset

==================

1 Create a file called file1 with the below data

one

2 Send it to stagging area and local repository

git add .

git commit -m "a"

3 Open the same file and add more data

one

two

4 Send it to stagging area and local repository

git add .

git commit -m "b"

5 Open the same file and add more data

one

two

three

6 Send it to stagging area and local repository

git add .

git commit -m "c"

7 Check the commit history

git log --oneline

8 To move to an older version example b version

git reset --hard commit\_id\_from\_step7

Soft Reset will move the git one step back i.e we will see the files as

present in the stagging areas

git reset --soft older\_commit\_id

Mixed Reset will move the git two steps back i.e we will see the files

as present in untracked/modified section

git reset --mixed\_commit\_id

If conflict occure because of one file that you edited then you have to restore changes before pulling remote change when you are pulling remote changes

**git reset file\_name**

Day 4

=========================================================

=========================================================

Git Merging: This happens based on the time stamp of the commit

=========================================================

Git Merging

===================

1 Create few commits on master

touch f1

git add .

git commit -m "a"

touch f2

git add .

git commit -m "b"

2 Check the commit history

git log --oneline

3 Create a new branch "test" and create few commits on it

git checkout -b test

touch f3

git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"

4 Check the commit history

git log --oneline

5 Move to master and create few more commits

git checkout master

touch f5

git add .

git commit -m "e"

touch f6

git add .

git commit -m "f"

6 Check the commit history

git log --oneline

7 Merge test with master

git merge test

8 Delete test branch

git branch -d test

9 Check the commit history

git log --oneline

==========================================================================

Git Rebasing or Fast forward merge

This is used when we want the commits from a branch to be projected

as the top most commits on the master branch, generally done when

the functionality created on the branch has to be projected

as the latest version of code

1 Repeat Step1-6 from the above merge scenario

2 To rebase test with master

git checkout test

git rebase master

git checkout master

git merge test

3 Check the commit history

git log --oneline

======================

Git Cherry pick

======================

This is used to selectively pick up few commits from a

branch and add them to the master branch

1 Create a commit on master

touch f1

git add .

git commit -m "a"

2 Create a new branch and create few commits on it

git checkout -b test

touch f2

git add .

git commit -m "b"

touch f3

git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"

touch f5

git add .

git commit -m "e"

3 To bring only c and d commits to master

git checkout master

git cherry-pick c\_commitid d\_commitid

=========================================================

=========================================================

Rearranging commit history

=========================

1 Create few commits on master

a-->b-->c-->d-->e-->f-->g

HEAD is pointing to g

\*note: we can not disturb first commit while rearranging commit history

2 To rearrange the top 6 commits order

git rebase -i HEAD~6

This will open the commit history in editor

Rearrange the commit history

3 Check the commit history

git log --oneline

Git Squash

=================

This is used to merge multiple commits to make the commit history

shorter

1 Below are few commits on master

a-->b-->c-->d-->e-->f-->g

HEAD is pointing to g

2 To squash f and g commits with d

git rebase -i HEAD~5

Remove the word "pick" and replace with "squash" for f and g commits

3 Check the commit history

git log --oneline

Git Amend

================

To make changes to the existing commit

i.e suppose you already have commit a , b and c. then you modified something, edited something or deleted something in files present in commit c. But you want to add these changes in commit c itself then this command comes into picture.

1 Create few commits on master

a-->b-->c

Head points to c

2 Create new file and send to stagging

touch f5

git add .

3 To add this new file to the last commit "c"

git commit --amend -m "c"

=========================================================

Day 5

=========================================================

Git Stash

==================

Stash a section of git where once files are finished

further commands of git are not accessible

1 To stash the files present in the stagging area

git stash

2 To stash the files present in stagging area and untracked section

git stash -u

3 To stash the files present in stagging area and untracked section and .gitignore

git stash -a

4 to check list of stashes

git stash list

5 to unstash files

$ git stash pop

With this only latest stash will come out

6 run 4th command and copy any stash id then run

git stash show stash-id

this command will show what exactly done in that stash

7 to unstash any particular stash\_id

git stash pop stash\_id

-----------------------------------------------------------------------

Tagging

Git Tags

==============

This is used for placing bookmarks on important commits

This is generally done to identify important commits relates

to release activities

Tags are classified into 2 types

1 Light weight Tags

2 Annited Tags

1 To create a light weight tag

git tag tag\_name

Note: Tags are by-default created by the latest commit

2 To create tag for an older commit

git tag tag\_name older\_commitid

3 To create an annoted tag

git tag -a tag\_name -m "some message"

This is also by default created for the top most commit

4 To create an annoted tag for an older commit

git tag -a tag\_name -m "some message" older\_commit\_id

git show annoted\_tag\_name

this will show tag with message included in it

5 To see the list of all the tags

git tag

6 To push the tags into the remote git repository

git push --tags

7 To delete a tag from the local machine

git tag -d tag\_name

8 To delete a tag from the github repository

git push origin :tagname

9 To see the annoted tag message and info

git show tag\_name

10 To create branch from an older commit

git checkout -b branch\_name tag\_name