Version Controlling
=====================
This is the process of maintianing multiple versions of the code
All the team members uplaod their code(check in) into the remote
version controlling system.The VCS accepts the code uplaods from
multiple team members and integrates it so that when the other
team members download the code they will be able to see the entire
work donw by the team

VCS's also preserve older and later versions of the code so that
at any time we can switch between which ever version we want

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

===

VCS's are categorised into 2 types
1 Centralised version controlling
2 Distributed version controlling

Centralised Version controlling

Here we have a remote server(code repository) into which all the team
members check in the code and all the features of version controlling
are implemented in this remote server

 Distributed version controlling

Here we have a local repository installed on every team members machines
where version controlling happens at the level of individual team members
form where it is uploaded into a remote server where version cotrolling
happens for the entire team

===
Setting up git on Windows
----
1 Download git from
  https://git-scm.com/downloads

2 Install it

3 Open gitbash and execute the git commands

===
Setting up git in ubuntu linux servers
----
1 Update the apt repository
  sudo apt-get update

2 Install git

```
  sudo apt-get install -y git

------------------------------------------------------------------------
Configuring user and email globally for all users on a system
git config --global user.name "sai krishna"
git config --global user.email "intelliqittrainings@gmail.com"

------------------------------------------------------------------------
On the local machine git uses three sections
1 Working directory
2 Stagging Area
3 Local repository

Working directory is the location where all the code is created
Initially all the files present here are called as untracked files

Stagging area is the location where file indexing happens and it
is the buffer area of git and the files are called as indexed files

Local repository is where version controlling happens and the files
are called as commited files


========================================================================
Day 2
========================================================================
Branching in Git
========================
This is a feature of git using which we can create seperate branches
for different functionalites 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
  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 create 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 form which branch it is commited and the file belongs to that
commited 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 Singup 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

==========
Downloading the code from the remote github
==========
This can be done in three ways
git clone
git fetch
git pull

==========
git clone
==========
This will download all the code from the remote repository
into the local repository and it is generally used only once
when all the team members want a copy of the same code

Syntax: git clone remote_git_repo_url

git fetch
==========
This will download only the modified files but it will place
them on a seperate branch called as "remote branch",we can go into
this remote branch check if the modificatios are accpetable and then
merge it with the main branch

1 Open the github
2 Go to the repository that we uploaded
3 Select a file and edit it--->Click on commit changes
4 Open git bash
5 git fetch
6 To see the name of remote branch
  git branch -a
7 To switch into this branch
  git checkout branch_name_from_step6
8 View the modified file
  cat filename
9 If these modifications are ok then merge with main branch
  git checkout main
  git merge branch_name_from_step6

==========
git pull
==========
This will download only the modified files and merge them with
our local branches
1 Open the github
2 Go to the repository that we uploaded
3 Select a file and edit it--->Click on commit changes
4 Open git bash
5 git pull
  We can see the modified files on the main branch

==========
Git Merge
==========
Merging always happens bases on the time stamps of the commits

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 git commit history
  git log --oneline

3 Create a test branch 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 Go back 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

9 Merge test with master
  git merge test

10 Check the commit history
   git log --oneline

================================================================
Git rebase
==================
This is called as fastforward merge where the commits coming from a
branch are projected as the top most commits on master branch

1 Implement step1-6 from above 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 Cherrypicking
=========================
This is used to selectivey pick up certain commits and add them to the
master branch

1 On master create few commits
  a--->b

2 Create a test branch and create few commits
  git checkout -b test
  a--->b--->c--->d--->e--->f--->g

3 To bring only c and e commits to master
  git checkout master
  git cherry-pick c_commitid e_commitid



========================================================================
Git reset
==============
This is a command of git using which we can toggle between multiple
versions of git and access whichever version we want

Reset can be done in 3 ways
1 Hard reset
2 soft reset
3 Mixed reset

In hard reset HEAD simply points to an older commit adn we can see the
data as present at the time of that older commit

1 Create few commits on master
  a-->b--->c

2 To jump to b commit from c
  git reset --hard b_commit_id




========================================================================
Git reset
==============
This is a command of git using which we can toggle between multiple
versions of git and access whichever version we want
```

Reset can be done in 3 ways
1 Hard reset
2 soft reset
3 Mixed reset

In hard reset HEAD simply points to an older commit adn we can see the
data as present at the time of that older commit

1 Create few commits on master
  a-->b--->c

2 To jump to b commit from c
  git reset --hard b_commit_id

Soft reset will also move the head to an older commit but
we will see the condition of the git repository as just one
step prior to the c commit ie the files will be seen in the
stagging area

git reset --soft b_commitid

Mixed reset also moves the head to an older commit
but we will see the condition of git as 2 steps prior to the c commit
ie the files will be present in the untracked/modified section

git reset --mixed b_commitid

==========

==========

Git stashing

==========

Stash is a section of git into which once the files are pushed
git cannot access them

To stash all the files present in the stagging area
git stash

To stash all files present in stagging area and untracked section
git stash -u

To stash all files present in stagging area,untracked section and .gitignore
git stash -a

To see the list of stases
git stash list

To unstash a latest stash
git stash pop

To unstash an older stash
git stash pop stash@{stashno}
==========

==========
Git sqaush
==========
This is the process of merging multiple commits and making
it look like a single commit.This can be done using the git rebase
command

1 Create a commit history
  a --> b --> c --> d --> e --> f
  HEAD is pointing to f commit

Note: a commit is called as the "initial commit" and it cannot be
   squashed

In the above scenario we can sqaush only a max of 5 commits

2 To squash
  git rebase -i HEAD~5
  This will open the top 5 commits in vi editor
  For which ever commits we want to perform a squash opration
  remove the word "pick" and replace it with "squash"

3 Check the commit history
  git log --online

==========
Git rebase can also rearrange the commit history order

1 Create a commit history
  a --> b --> c --> d --> e --> f
  HEAD is pointing ti f commit

2 To rearrange the commit history order
  git rebase -i HEAD~5
  Reaarange the commits in whatever order that we want

3 Check the commit history now
  git log --oneline

==========