********** GIT ************

Git Index:
==========

Git is a Source Code Management(SCM) / Version Control system(VCS) / Distributed version control system

--> Types of SCM/VCS tools
      * Git,
      * CVS,
      * Perforce,
      * clearcase,
      * svn

git:- it is a process of tracking and controlling/maintaining changes of a software product

--> Download git from https://git-scm.com/
    which git --> path of git where it installed
    git --version -->to verify
    git help config

source installation method:
----
yum groupinstall "development tools" -y
yum install gettext-devel openssl-devel perl-CPAN perl-devel zlib-devel curl-devel
go to official git website to download source file(tarball) of git -->>
wget https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.19.2.tar.gz
extract tarball-->> tar -xvzf git-2.19.2.tar.gz
remove tarball(git-2.18.0.tar.gz)
1. configure --->> ./configure --prefix=/usr/local/git
2. compile code --->> make
3. installation --->> make install
4. we have to export the path to run git from anywhere
      >> echo "export PATH=/usr/local/git/bin:$PATH" >> /etc/profile
      >> source /etc/profile

git features compare to other tools:-

1. Speed
2. Support for non-linear development(thounds of parallel branches)
3. distributed version control system
4. Able to handle large projects efficiently

Why Git is Distributed Version Control system:-

1. Speed
2. Network Issues
3. user collabration
4. security reasons user to user sharing is not recommanded.

Git Terminology:-
==========
1. client/server
2. workspace: - space where the client comm with server
3. Repository : - it is a container/Directory. for each product/project we've one repository in the server.
4. Branch :- parallel development
5. Checkin:- Once the changes done, storing back to the server to maintain perminentaly
6 Checkout:- Taking petmission from server to do modifications.
7. Revision/Version id/commit id : - git will track all the info of changes done by a user like when,which repo,who,which file, what changes)

it stores in the form of snapshots i.e diff b/n existing file data and currently modified file data.

commit id/check in :- with 40 char long

in workspace we've 3 stages before commit id get generated

--> working dir:- where you working with files physically

--> Staging Area:- it helps to create commit id(it is virtual area , here workspace don't know which one is existing and modified data)

--> Repository:- where stores data (commited file)

git config:-
=========

```
git config --global user.name "vinodh machi"
git config --global user.email "vinodhk070@gmail.com
git config --global push.default simple
git config --list
```

local : By default, git config will write to a local level if no configuration option is passed. .git/config

global : Global level configuration is user-specific, meaning it is applied to an operating system user.

Global configuration values are stored in C:\Users\vinodh\.gitconfig

Ex:- One of the most common use cases for git config is configuring which editor Git should use.

editor = VIM

Ex:- Merging Tools

git config --global merge.tool kdiff3
system : System-level configuration is applied across an entire machine. This covers all users on an operating system and all repos.
system configuration values are stored in C:\Program Files\Git\mingw64\etc\gitconfig

git config --edit/--list --local
git config --edit --global
git config --edit --system

creating Repositorys:-
1. bare repo: is the centrailized repository which is used to store and share the changes.
2. non-bare repo: user (or) workspace (or) local repository whih is used to modify the changes.

mkdir central.git
git init --bare
git clone central.git <source> vinuspace <dest>
create file in workspace
move to staging area : git add java
move from staging repo : git commit -m "first chickin"

-->git log : list of all the commits will display
--> set of changes together with single version id: git add . (or) java, oracle...etc

if you don't have the same content then-->Modified/Untracked files
if you have same content in the file then --> Unmodified

--> git push <source> <dest>
--> git push enter : to push the changes to repository for sharing to other users
--> 2nd user access the repo and created new file and pushed it to central repo and now
1user want to update things what user2 changed. then use
-->git pull

Git ignoring:
==========
--> .gitignore : is a conf file to ignore the unwanted runtime files like jar, war, log....
1.class (Full name)
*.class (pattern matching)

Log viewing:
==========
--> git log -3
--> git log --oneline --grep "workspace"
--> git log --grep "stringmasg" --oneline
--> git shortlog
--> git log --stat
git show --stat 23rt459bf
git show --name-status 23rt45bf

```
git show --name-only abc
git log hello.txt

Branching:-
===========
--> parallel development
--> storing of files in a repo is in the form of branches
       ex:  android               windows
               file1                   file1
               file2                   file2

       here the futures are same only slight diff i.e os
--> master is the default branch whenever we create repository/workspace
--> at any point of time you can work with  only one branch at a time
--> git branch : list of branches avilable
      git branch branch_name
      git checkout new_branch : switch to other branch
      git checkout -b new_branch_name : creating new and switching to new
branch
--> when we create a new branch based on the master the same fiels and
commit ids will come to new branch also.
# merging the 2 branches:-
--> git merge <sour> <dest>
--> git branch -d <branch_name> : to delete branch, use 'D' for forcely
remove without fully merged.
--> git branch --merged:- lists the branches that have been merged into
the current branch
--> git branch -no-merged:- lists the branches that have not been merged
--> git rebase :Alternate to merge

Note:- when we create,edit,delete in workspace, this shows same in all
branches until you commit permanently.

--> bare repo  you will not have working dir, so i.e you not able to see

Merging:-
=========
--> Master(Target)<---------feature(source)

checkout command:-
==================
1. git checkout <BRANCH>:<COMMITID> --> specific commit('detached HEAD')
2. git checkout branch_name --> to switch one branch to another
3. git checkout -b <new-branch-name> --> creating and directly switching
to new branch
4. git checkout -- file1 file2 --> to dicard the changes in working
directory(when only files deleted,edited from work dir)

Stashing commited changes :-
============================

stashing will do 2 things
      1. whenever you modifing the changes it will take backup.
```

2. and it revert back with orginal position of the file where you started.
git stash --> create stash, remove changes from working directory(when files create and edit)
git stash list--> list all stash available for the repository
git stash apply stash@{0} -->my 1st idea is good (it only apply it not remove)
git stash pop --> it will apply and remove the last stash array
git stash clear --> remove all backup entrys

Note:- Here no need to move changes,files..etc to staging or commiting

Removing files:
==========
rm
git rm filename
git clean -n(dry run means which are the file eligible to remove, very care full before running this)
note: removes untracked files
git clean -f(removes the untracked file instead of adding to .gitignore conf file)

Removing files in working Dir:-
==========

git rm :- will remove the file from the index and working directory (only index if you used --cached) -
so that the deletion is staged for next commit.
When using git rm, the removal will part of your next commit. So if you want to push the change you should use git rm
rm:-However, if you do end up using rm instead of git rm. You can skip the git add and directly commit the changes using: git commit -a

Undo/Reverting Changes after adding to staging area :-
==========
i added a file into staging area, now i want to modify it before going to commit it?
Syntax: git reset HEAD <file>
sol is :	1. git reset (mixed is the default option)
		2. git reset --soft --> revert back / undo the changes from staging area
		3. git reset --hard --> to remove changes from all the 3 areas
					@ working directory
					@ staging Area
					@ Repository(once file came to staging area, it creats temperary commit id in repo)
		4. git reset --soft --> remove changes only in repo temp id (HEAD)
HEAD--> once we add file into staging area it create one temp id in the repo as a reference, this is the
	latest commit id which we working.

Undo/Reverting Changes after commiting:-
==========

1. git revert 256ed01(1st 7char) --> it revert only the content not the complete entry(commit id) and create a new commit id and refering to previous.

HEAD:-
======
HEAD is the reference to the most recent commit in the current branch(In most of the cases).
HEAD Doesn't point to most recent commit when we go into DETACHED HEAD State.
git show HEAD --> to see head commit
git difftool HEAD HEAD~1

Tags:-
----
to identify/reference for a commitid to quick access/ creating specific point in history for our repository

create tag for particular commits:
git tag <tag-name> <reference of commit>

git tag <name-of-tag> --> light weight tag
git tag -a <tag-name> -m "comment" <commitid> --> annotated tag
git tag -n9 --> to see tag messgaes
git tag --> to list tags
git show --stat <tag_name>
git push origin v7.2 ---> pushing to remote(github)
git push origin --tags ---> pushing all tags at once to remote
git push --tags
Note:- when you working in folder in which github repository is cloned)

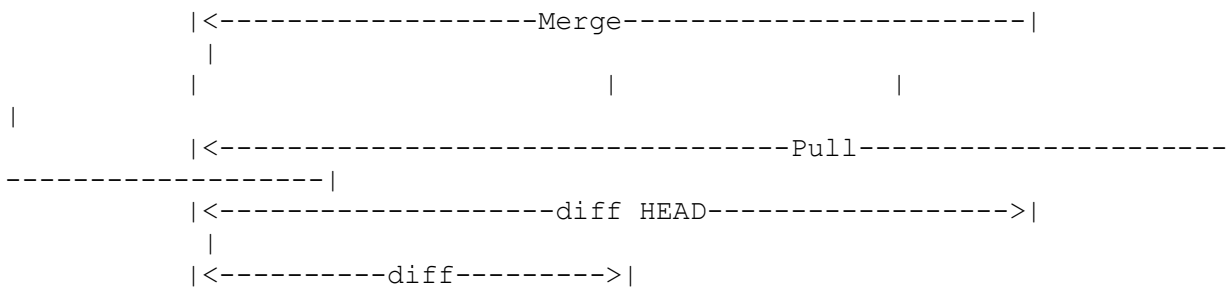
Deleting tags from local repository:
git tag -d <tag_name>
git tag -delete <tag-name>
git tag -d v7.2 vinumaa v7.3 --> deleting multiple tags at once in local repository

Deleting tags from remote repository:
git push origin -d v7.2
git push origin -delete v7.2
git push origin :v7.2
git push origin -d v7.2 vinumaa v7.3 --> deleting multiple tags at once in remote repositor

we cannot checkout tags in git but, we can create a branch from tag and checkout the branch
git checkout -b <branch_name> <tag_name>(give already created tag names)

Diff:-
======
used to compare Changes,branches,and commits.

```
git diff --> diff b/n version in the working dir and version in the
staging/index area
git diff HEAD --> diff b/n version in working dir and commiting dir
git diff --cached --> diff b/n staging and commit versions

git diff --stat 2b7889f4..08j86gj9
git diff master..new

Rewriting Commit Messages:-
===========================
--> git commit --amend : for latest commit
--> git rebase -i(interactive) HEAD~2 : for other commits
note: use reword option, while r=editing commit message
--> git ls-files : for latet commit in current branch to list all files
--> git show --name-only <commit id> : specific file to list
--> git commit --amend --no-edit : # Edit hello.py and main.py git add
hello.py git commit
                          # Realize you forgot to add the changes from
main.py git add main.py git commit --amend --no-edit

 p, pick = use commit
 r, reword = use commit, but edit the commit message
 e, edit = use commit, but stop for amending
 s, squash = use commit, but meld into previous commit
 f, fixup = like "squash", but discard this commit's log message
 x, exec = run command (the rest of the line) using shell
 d, Drop = remove commit

rebase --continue
rebase --abort
rebase --skip

Fetch and pull:-
----------------
git fetch origin: i.e all the repos in the github will come to local
repository,
git merge origin/master : to integrate the commitids

workflow:-
=========
           |           Add          |    Commit             |
            |
           |------------------------>|---------------------->|
            |
           |-----------------Commit -a ---------------------->|   git push
|
           |                         |                |-------------------
---------->|
      Working Dir           Index       Local Repository       Remote
Repository
      (work space)              (Staging)               (HEAD)
            |
           |                         |                |<----------git
fetch----------|
```



Note:- the above commit -a option will only work for already known files to staging area(i.e old files)

reflog:-
========
it shows when we commit,checkout branch, reset....etc
git reflog
git reflog --relative-date
git reflog show -all (or) <branch_name> for particular branch reflogs
git reflog show/HEAD@{5}

** Basically "git blame <file_name>" is used to show what revision and author last modified each line of a file.
   It's like checking the history of the development of a file.

Bisect:-
----
first it will bisect/divide the given range into 2 parts and start executing
      1. range between culprit/bug is there
      2. pass Script file
           git bisect file_name.rb
           exit status 0(success) 1(failure)

commit 10 < fail
commit 9
commit 8
commit 7
commit 6 < fail
----
commit 5 < fail
commit 4 < START<<----
commit 3 < pass
commit 2
commit 1

Hooks:-
----
hooks are programs which are automatically trigger at certail points in the git execution cycle.

GitHub:- (ORS)
========

@ is a website to upload repositories online
@ Provides backup
@ Provides visual interface to repo
@ makes collaboration easier

* Through browser you can create, or upload files into github
* git clone https://github.com/vinodhk070/Machi.git
* change some data and add, commit
* git push (or) git push origin master

```
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/vinodhk070/hello.git
git push -u origin master
```

```
git remote add origin https://github.com/vinodhk070/gitsample.git
git push -u origin master
```