*To Configure global values across Git.*

Git config –help → gives the man or usage of git config and supported levels.

git config --global user.name Ganesh → To configure the user ame at global level of git.

Git config –global –unset user.name → This will unset/remove the value of user.name globally

git config --global user.email emailid -> → To configure the email id at global level of git.

git config --global push.default simple → To configure push behaviour globally in git.

*To list the global values*

git config --list

*To initialize the bare repository:*

git init --bare

*To initialize the git repository alone:*

git init

*To clone git from central repo/git hub:*

git clone central.git userworkspace

touch file1 file2

*To move the files from working directory to Staging Directory:*

git add .

git add file1

*To rollback the files from staging Dir to Working Directory:*

git rm --cache file1

git reset HEAD <file>..." to unstage

*To commit/move files from staging dr to repository inside workspace:*

git commit -m "commit message"

git commit -m "commit message" file1 → if we need to commit only specific file.

“git commit—amend”. By running this command, you can fix the broken commit message in the editor.

git status → To check the status of workspace/repo

To move the files that are committed from repo to staging area:

git reset –soft commitid/HEAD~1 → to move the latest commit from repository to staging.

git reset –mixed commitid/HEAD~1 → to move the latest commit from repository to workspace.

git reset commitid

commit changes.

**Git Log commands:**

git log

git log –oneline , git log –oneline -2 , git log –oneline –grep “file”

git show b5252dc2e1c36abb95a984027f2c28115b338619(commit id)

*To ignore/exclude few files from pushing or commit use below:*

.gitignore → this is a hidden file which holds file types which need to ignore.

Note:- In .gitignore file you can enter extension of files that shouldn’t committed .

**Log Format**

If we need to redirect the complete git log with more log information try below:

**Option 1:** One line summary w/ Graph

***git log --pretty=format:'%h : %s' --graph > log.log***

Results in:

\* 2d3acf9 : ignore errors from SIGCHLD on trap

\* 5e3ee11 : Merge branch 'master' of git://github.com/dustin/grit

|\

| \* 420eac9 : Added a method for getting the current branch.

\* | 30e367c : timeout code and tests

\* | 5a09431 : add timeout protection to grit

\* | e1193f8 : support for heads with slashes in them

|/

\* d6016bc : require time for xmlschema

**Option 2:** One line summary w/o Graph

***git log --pretty=format:'%h mor***

Results in:

a6b444f was Scott Chacon, 5 days ago, message: dammit, this is the second time this has re

49d77f7 was Scott Chacon, 8 days ago, message: modified index to create refs/heads if it i

9764edd was Hans Engel, 11 days ago, message: Add diff-lcs dependency

e1ba1e3 was Hans Engel, 11 days ago, message: Add dependency for Open4

0f87b4d was Scott Chacon, 12 days ago, message: merged recent changes

Note:- For more place holders of log files check <https://git-scm.com/docs/pretty-formats>

*About Git Branch:*

Uses of branch is , with a same code but for different releases user can use the different

code parallely like android and i phone.

It means a single repository contains a branch to hold the information. By default every

workspace contains a repository and every repository contains a default branch called “master”.

So , to work on multiple branches we can use git branches concept.

git branch → Displays list of branches that are available under a workspace.

Git branch child-branch → this creates a branch called child-branch.

Git checkout child-branch → this allows user to switch from master branch to child-branch

git branch -a -r –merged → This gives the branches list that are merged

git branch -a -r –no-merged → This gives the non merged branches list

git branch -d branchname → Deletes the branch

Note:- We can able to add n no of branches.

*To push changes directly to github use below:*

git push --set-upstream https://github.com/clouddevopspractice/Practice1.git childbranch

*If you like to merge child-branch* and *master branch:*

git merge source branch destination branch → This command is to merge source branch with destination branch.

**“Rebasing” is an alternative to merging in git.**

**The syntax used for rebase is “git rebase [new-commit] “**

Git push -u --all → This will push all branches to central/git hub repository from your local repository.

**Cherry Pick:**

git cherry-pick commitid → it will reset to that specific commit id.( git cherry-pick will merge the specific commit id ).

Note:- git doesn’t check for merge conflicts

**Stashing:**

Git stash will take the backup of all files before you commit changes , and it undo the changes so that you can work on it.

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy.

Note:- Usually git stash wont include files that are in working directory for stash, to achieve this use below command.

Git stash   
***git stash -u or git stash –include-untracked*** this tells git stash to also stash untracked files.

Saved working directory and index state WIP on master: a01fa5c Merge branch 'master' of https://github.com/clouddevopspractice/Practice1

HEAD is now at a01fa5c Merge branch 'master' of <https://github.com/clouddevopspractice/Practice1>

git stash list → This will displays the backup information that taken using git stash.

stash@{0}: WIP on master: a01fa5c Merge branch 'master' of https://github.com/

stash@ 0 – > Here 0 is called as index no.

stash@{1}: on master: a01fa5c Merge branch 'master' of <https://github.com/cl>

Note :- To restore the data from back up ( stash) use below process.

Git stash apply stash@{1}

OR

git stash pop → This will reapply previously stashed files.

git stash clear → To clear the stash entries/backups taken by stash.

Git stash drop → When you are done with the stashed item or want to remove it from the list, run the git ‘stash drop’ command. It will remove the last added stash item by default, and it can also remove a specific item if you include as an argument.

Note:- <https://www.atlassian.com/git/tutorials/saving-changes/git-stash>

**Reverting changes:**

git reset file → Revert uncommitted changes.

git reset –soft / commitid → Move only to head pointer

git reset –mixed → move the head pointer & reset the the staging area(default)

git reset –hard → move the head pointer & resets staging area & working tree to the new HEAD. This will make files to first stage before you modified in working tree.

Git revert commitid → To revert/undo the change to the commit we performed.

Git checkout commitid → To enter to the commit id.

It is only for showing the content of that commit id , here you cant modify anything as its nor reset.

Note:- It works like this

in this state we will be in ***“detached head state”***

“Git checkout branchname@@commitid”

“Git checkout branchname@@latestcommitid”

When you checkout to a specific commit id , and check git log you wont be able to see earlier commitids because the reference of head is pointing to old commit ids.

Git rm <file1> → To delete the file

***Removing Un Tracked Files:***

Git clean -n (-n is for dry run) – This will show all the files that are untracked files which are in working directory and the option -n wont delete anything.

git clean -f (force delete) – This will forcefully cleans all the untracked files.

git checkout file → Discarding changes in working directory.

**Tags:**

Git has the option to tag a commit in the repository history so that you find it easier at a later point in time.

Git tag -a <pattern> -m “comment” <commit id>

*contents of the tag →* git show pattern

*Display list of tags →* git tag

*push the tags* → git push –tags

*Delete a tag* → git tag -d <tag>

**Authorisation :**

**Triggers /Hooks:**

> Its an action that allows an user for proceeding a change.

Ex:- If user wants to push then for push we can enable pre hook or script so that the action in script will execute and then push will happen.

Git lab or garit are the tools that we can automate these triggers and automation in git.

Note:- As its distributed GIT is used more often and speed.

GIT HUB is your central repository to store and clone globally.

Also , we need to check how to integrate GIT with our company LDAP Servers for access part.

Other Points:

Default installed location of git using yum update installer is /usr/bin/git

### **push an existing repository from the command line to remote repository or github repository**

git remote add origin https://github.com/clouddevopspractice/MyPractice.git

git push -u origin master

### **Version Controls in market other than GIT:**

### Sub Version , Mercurial , clearcase & TFS Project

### Other Important links:

<https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config>

**Interview Questions:**

[**https://career.guru99.com/top-40-interview-questions-on-git/**](https://career.guru99.com/top-40-interview-questions-on-git/)

**Mention some of the best graphical GIT client for LINUX?**

Some of the best GIT client for LINUX is

a)Git Cola

b)Git-g

c)Smart git

d)Giggle

e)Git GUI

f)qGit