1) diff <name of code 1> <name of code 2> : show the difference between two codes

5c5,6 --> line 5 at code 1 changed with line 5,6 at code 2 (c means changed)

11a13,15 --> line 11,13,15 added.

2) To show better the differences, we can use the unified version: diff -u <name of code1> <name of code 2>

In this version, - signs means deleted line and + signs means added

3) wdiff, meld, KDiff3, and vimdiff are other tools.

4) diff -u <name of code1> <name of code 2> > <change.diff>

5) patch command takes a file generated by diff and applies the changes to the original file: patch <output file name> < <filename.diff>

6) Files are usually organized in repositories, containing seprate software projects or just group all related code.

7) a commit is a collection of edits which has been submitted to the version control system (VCS) for safe keeping.

8) To show git who we are, we should use git config command and specify our user.name and user.email: git config --global user.email "<email\_address>" user.name "<your name>"

--global: means that we use this name and email for all repositories.

9) git init --> start a repository from the scracth --- git clone --> start a repository that already exists somewhere else.

After creating a repository with git init --> we can use ls -la to find it in the directory.

Also ls -l .git/ command help to see the files in the created Git repository (Git directory) --> stores the changes and the change history. The area outside the git directory is the working tree. The working tree is the current version of your project (workbench).

10) To make git track our files we use git add command: git add <file name> --> we add our file to the staging area (index): means a file mainted by GIt that contains all of the information about what files and changes are going to go into your next commit

11) Getting some information with: git status

changes to be commited: .... (they are in the staging area)

12) To get it committed into the .git direcotry, we run: git commit --> it opens a text editor to write a commit message.

13) any Git project will consist of three sections. The Git directory, the working tree, and the staging area.

The Git directory contains the history of all the files and changes.

The working tree contains the current state of the project, including any changes that we've made.

the staging area contains the changes that have been marked to be included in the next commit

SO each track file can be in one the three main states: modifed, staged or commited.

14) git commit -m '<your message>'

15) git config -l: show git repository information.

16) When we create a file in a repository (using git init or git clone [copy]), the git does not track the changes into file until we add it (git add )

17) The good git commit message should be like this:

50 charactres tell the summary

72 charactres tell the detailed

18) git init or git clone --> git config --> git add --> git commit

19) git log: shows the histroy infromation about all commits.

20) git commit -a: a shortcut to sage any changes to tracked files and commit them in one step. By using this we skip staging area, means we can't add any other changes before creating the commit.

21) git commit a- -m " <commit message>": all of the steps can be done in one command. This shortcut is good fod small changes.

22) git log -p: show the patch that was created. To see the details changes in each commit.

git log --graph: seeing the commits as a graph

git log -- oneline: see one line per commit.

23) git show <git commit id>: show the changes for specific commit.

24) git log --stat: show some stats about the changes in the commit.

25) git diff: helps us to track the difference before commiting.

If we want to see specific file diff rather than all file in repositroy: git diff <name of file>

26) git add -p: show the difference before adding to staging area.

git add \*: This command will end yp adding any change done in working tree to the staging area.

27) git rm <filename> : remove the file from the git repository

After that we need to commit it.

28) git mv : to move files between directories.

29) git cheat sheet: https://training.github.com/downloads/github-git-cheat-sheet.pdf

30) git checkout <filename> : can change a file back to its earlier committed state by using the git checkout (it reverts changes to modified files before they are staged)

git checkout: use to check out the latest snapshot for both files and for branches.

git checkout <name of branch>: swicth to the specified branch.

git checkout -b < name of branch>: creating a new branch and switch to it.

31) git reset : remove from the staging area.

git reset -p: used for specific changes.

32) git commit --amend: allow us to modify and add changes to the most recent commit. Avoid amending commits that have already been made public because it it will overwrite the previous commit.

33) git revert: creat a new commit, opposite of the changes in the bad commit.

git revert <commit ID>: This can be used for rollback a commit that wasn't most recent one.

34) Each git commit has an ID which is created by SHA1 alglorithm using information related to the commit. It also brings c onsistency.

35) Branch: a pointer to a particular commit. It represents an independent line of development in a project.

The deafult branch that git creates for you when a new repository is initialized is called MASTER. The master branch is commonly used to represent the known good state of a project. So, it is better to commit the last code in there and use another branch for testing.

36) git branch: show all of the branches in your repositroy, "\*" the asterisk show which branch we are.

git branch <branch name>: creating a new branch

git branch -d <branch name>: delete the branch

37) This demonstrates that when we switch branches in git, the working directory and commit history will be changed to reflect the snapshot of our project in that branch. When we check out a new branch and commit on it, those changes will be added to the history of that branch.

38) Merging: The term that Git uses for combining branched data and history together.

git merge <branch name>: merge the branch to the master branch

git merge --abort: If there are merge conflicts (meaning files are incompatible), --abort can be used to abort the merge action.

39) If you dont wan't set up a Git sever yurself and host your repositories, you can use online service like GitHub.

GitHub is a web-based Git repository hosting service. Other examples are GitLab or BitBucket

40) git clone <URL address>: copy the repository in your local computer.

git clone <URL address> <directory\_name>: copy the repo in specified directory

41) git push: send changes to that remote repository (update the remote repository). [commits from local to remote]

42) git pull: get changes from the remote repository. [fetch the newest updates from a remote repo]

43) To avoid having enter the password each time we push or pull. We can do these ways:

1) create an SSH key pair and sotre the public key in our profile so that GitHub recognizes our computer.

2) Use credential helper, which caches a password for a limited time

git config --global credential.helper cache

44) For synchronization of all codes, we should consider these stages when we are doing collaborative project.

modify --> stage --> commit |local changes in local repo| --> fetch |new changes from remote repo |--> merge | if necessary| --> push)

45) Git set up the remote repo with the default origin name when we call a git clone.

git remote -v: in the directory of the repo show the configuration ( the fetch and push URL can be different in some cases)

git remote show <origin or other name>: a complete information about the remote origin

git remote update: get the contents of a remote branch w/o auto merging.

46) git branch -r: looking at the remote branches that our Git repo is currently tracking.

To modify the branch contents: we pull any new changes to our local branch, then merge them with our changes and push our changes to the repo

47) Git does'nt keep remote and locan branches in sync automatically

git fetch: This command copies the commits done in the remote repository to the remote branches

the difference between git fetch and git pull is git fecth only fetches remote updates but git pull fetches + merges

the difference betwenn git fetch and git remote update: git fetch download but the other one show

git merge origin/master: merge the changes of the master branch of the remote repo into our local branch

48) git push -u origin <name>: adding branch to a remote repository, having branch reduces the problem in merging.

49) git rebase < followed by the branch name want to set as the new base>: change the base commit that's used for our branch

keeping history linear helps with debugging especially when we are trying to identify which commit first introduced a problem in project

50) Forking: a way of creating a copy of the given repo so that it belongs to our user. our user will be able to push changes to the forked copy, even when we can't push changes to the other repo -- there is a button in github for forking the selected repo in our repo.

51) pull request: after changing and modifying the selected project, we can pull request to notify the developers of that project to apply our changes.

52) git remote -v: verify that you have already setup a remote for the upstream repository, and an origin