<https://www.youtube.com/watch?v=xyXPFRFfXo4> : the basic working in 10 mins

The flow:

1. Add the file that you have changed by:- **git add first.txt** // or you can do is **git add .** //to add all files to staging phase
2. Then commit the files , like a checksum… by :- **git commit**
3. Push it to the github.com by statement : **git push** or **git push origin master**

// here everywhere where the name of the file contains a white space(i.e. here it means space)

//then we need to put the name of file in double inverted commas …

Master comes when git is tracking it

Cd <path> : for going into some directory

Ls : to show files right there

Git status : for showing status of files you are currently working with, i.e which are done after the last commit

Git commit : type of a save stage like in games we have

Git push / git push origin master : pushing the changes done by us to the online repo.

Git add <file-name> : to add the file which is untracked by git at the moment which is shown as untracked when done git status command

Git pull : retrieving the changes other have done and pushed online

Git checkout <file-name > : actually converse of add command, i.e. the untracked file would be tracked with its previously committed state

Git log : log of commits

Git reflog : technically the same but with the # code

The most commonly used git commands are:

add Add file contents to the index

bisect Find by binary search the change that introduced a bug

branch List, create, or delete branches

checkout Checkout a branch or paths to the working tree

clone Clone a repository into a new directory

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

fetch Download objects and refs from another repository

grep Print lines matching a pattern

init Create an empty Git repository or reinitialize an existing one

log Show commit logs, i.e. on which date which changes where commited

merge Join two or more development histories together

mv Move or rename a file, a directory, or a symlink

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

rebase Forward-port local commits to the updated upstream head

reset Reset current HEAD to the specified state , i.e remove it from staging or remove it from added

rm Remove files from the working tree and from the index

show Show various types of objects

status Show the working tree status

tag Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

Git show HEAD for showing the top commit

Git reset HEAD <fileName> // for getting fie back from staged phase to upstaged phase

//To retrieve to some last commited version:-

Git checkout <Commit code, only first few digits can also work , no need to type the complete one> -- <file name>

Git branch <branch name> // to create a new branch

Ex: git branch testBranch

//here this will not redirect us to the branched branch, it would just create one

//To go to the selected branch:-

Git checkout testBranch

//now you can do changes in here and then save it in the branch

Git merge <branch name>

Git vs mercurial:-

In git the branches are just like stickey notes which points the nodes(i.e. which are the actual commit states) and they don’t actually form in the commit node.

Whereas in mercurial the branch name forms the part of the commit

When we need to create a patch the it could be re learned by this following video:

<https://www.youtube.com/watch?v=QtXj9tt-RUE>

it basically is what we can have only one patch of our commit to any other branch without having all the other commits all together

kind of like cherry picking

when ever I get an thing like local repo is ahead of remote by <some X> commits…check my stack overflow profile I have a question which has been asked by me and also has answers for the same

git merge : what it does is merges your branches and also keeps your commit history safe. That’s an advantage… but when the branch which is to be merged is quite big and has quite big of a history …it is recommended to not use it since the tree wont be very much readable.

In cases like that what we can do is git rebase…what it actually does that it takes up your complete branch which was to be merged to say master… and commits all its commits in master one after another so as we get one linear tree structure of it. But there is a drawback to it , that is it looses the author ship…all the commits would be under the name of the person who is having the control of master branch.