Softwares of GIT :-

1) Git 2) Notepad++ 3) P4Merge

① git config --global user.name "Your Name"
git config --global user.email "Your Email"

② notepad++ ~/.bash_profile

↓

alias app = 'notepad++ -multiInst -nosession'

③ git config --global core.editor
"notepad++ -multiInst -nosession" } → Default editor

④ git config --global -e

P4Merge :- It is an visual compare & merge tool
Command line merging is annoying.

⑤ git config --global diff.tool p4merge

⑥ git config --global ~~diff.tool~~. difftool.p4merge.path
"C:\Program Files/Perforce/p4merge.exe"

⑦ git config --global difftool.prompt false

⑧ git config --global merge.tool p4merge

⑨ git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

⑩ git config --global mergetool.prompt false.

Git Basics :-

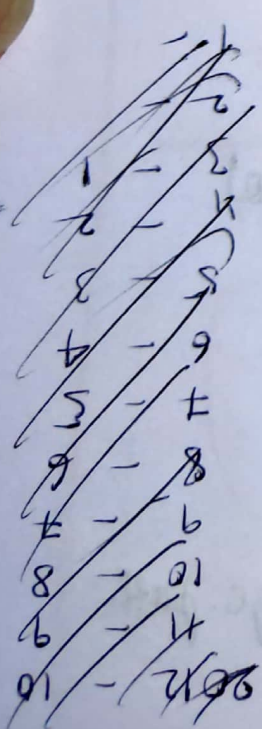
1) ~~Creating a repo~~ Git Init :- Make a directory for git projects.

i) Use git init inside the folder

git init <name>

Navigate to that to start with master branch

2) Type dir command to view all files related to git.



Local Git states:-

1) Working directory :- All files & folders of app
2) Staging area :- Used to prepare for next commit
3) Repository (.git) :- Saved changes to git repo
4) Remote repo :- Saved to github site

First commit:-

1) Git status:- To check status of commits.

Create a Readme for git project by text editor

Git status says its untracked.

2) Git Add:- git add Readme.md.
3) Git commit:- git commit -m "Msg"

~~Repository:-~~

Commit history using Log & show:-

"git log" is used to get the history of commits.

"git show" is used to get the history and difference between the commits.

Express commits:-

"git ls-files" is used to get the files that are being tracked by git.

"git commit -a -m "Msg"" → Adding modified files/ untracked files with a Message.

Recovering from mistakes:-

If we have staged a file & we want to revert back changes, then use "git reset HEAD <filename>" → Back to untracked state.

If we want to go to the previous best state, then use:-

git checkout -- <FILENAME>

Using Aliases:-

"git log --oneline --graph --decorate --all"

We get the good information of git history. We can shorten above one using aliases.

"git config --global alias.<aliasname> "command"."

Eg:-
"git config --global alias.hist "git log --oneline --graph --decorate --all"

We can get the config details using:-

git config --global --list

We can execute the alias using:-

git <alias-name>

Renaming & deleting files:-

Rename:- git mv <oldname> <newname> → To staged

Deletion:- git rm <filename> → To staged.

To complete above two methods, we need to commit

Managing files outside git :-

1) Rename a file in git repo
2) Create a new file
3) If you perform git status, it will understand above rename as delete & modified. and create to ~~master~~ untrack
4) To make clarifications, use git add -A

Excluding unwanted files :-

Adding the files to .gitignore, you can include even wild card representation, (*.log)

comparing differences:-

"git diff <Hash code for commit1> <Hashcode for commit2>"
We can use HEAD for latest commit.

"git difftool <HASHCODE> <HASHCODE2>" → This will launch the configured difftool.

Branching & Merging :-

Branches are timelines of commits

Merges:- 1) Fast forward Merge :- ~~As~~ Simple changes, commits from other branches are merged with master

2) Automatic :- Non conflicting changes, preserves both branches

3) Manual :- Automatic merges not possible, conflicts found.

Special Markers :-

Also called pointers, Eg :- HEAD pointers pointing to latest commit.

Branching :-

git branch → To list branches

git checkout -b <branchname> → Creating branch & switch to it.

git diff <branch1> <branch2>

(master) git merge <branch2> → Merge two branches

~~If HEAD and fin~~ uses type of merge & merges.

git branch -d <branch> → Deleting branch

Conflict resolution :-

We make the same linenumber changes & git gets confused about merging the lines, in that case we can use our p4merge tool to select the correct statement and hit Save to resolve the conflict.

Tagging:-

Putting labels on commits.

git tag <tag>

git tag -d <tag> → Delete

More better way to organise the git logs:-

git tag -a v1.0 -m "Release 1.0".

We can see the tags using:- git show <tag>

Saving Work in Progress with Stashing:-

Stashing is ability to put current working directory on WIP on branch and then updating ignore to exclude merge files.

"git stash"

"git stash list" → To get list of WIP.

"git stash pop" → Adding stashes to Readme & then remove stashes from list.

Time travel using Reset & Reflog :-

Rolling back to previous commits using reset

"git reset <HASH of commit> -- soft"

Soft :- Preserves staging area only changes commit ID of the HEAD pointer

Mixed :- Rolling back to provided commit & keeping the files in untracked mode.

Hard :- Complete rollback to provided commit

Reflog shows us all the actions we have performed.

"git reflog"

Linking our Repo from github :-

git remote add origin <git'repo url>

We get two URL's one for fetch & another for push.

Pushing :-

git push -u origin master --tags

Verifying Changes on Github:-

Goto github & Refresh to get the commits.

SSH Auth:-

Setting up private key for authentication.

```
mkdir .ssh
cd .ssh
ssh-keygen -t rsa -C "<emailID>"
```

ls -al to get list of all files, it should contain:- id_rsa id_rsa.pub

Open .pub file to view the public key.

Goto Github > Settings > SSH Keys and add the public key.

To connect using the ssh then:-

```
ssh -T git@github.com.
```

Github Repository:-

We can have a default setting for our new repo, select the type of gitignore file & then licensing file.

Cloning Repo :-

We are cloning with SSH, so use the SSH URL from clone link and then type "git clone <SSH-URL>" <folder name>

Seeding the Repo with some content :-

Goto initializr.com & download some bootstrap codes. We will copy all those into our repo directory like this :-

1) cp -R <Sourcepath> . (dot to indicate current path)

2) Add & commit the files to repo.

Default behaviour of push :-

git config --global push.default simple

there are two types :- simple & matching.

Simple uses the repo used in "git pull" whereas Matching uses the repo ~~used in~~ that matches the repo being pushed. (same remote & local branches)

Fetch & Pull :-

When we make some changes & commit at github level, then we need to pull / fetch to make remote & local in sync.

fetch → fetching the changes from github
pull → fetching + merging with local repo

Repo features & settings:-

We need to change our remote fetch & push URL's after renaming our repo in github. By this command:-

git remote set-url origin <SSH-URL>

git remote show origin → To get detailed info.

Click on + on the right side of repo to add new files.

Commit Details:-

Goto commits section to view the details. We can add comments inline inside the commit history.

Branching on Github:-

Click on the "branch:" button to get list of branches & search for branches. We can also create new ones by searching for name that doesn't exists.

Branching on local Git:-

"git checkout -b <branchname>"

"git push -u origin <branchname>"

Merging with master:- (on Github)

Pull & merge button besides sub-branch will do it. We need to send the pull request to the owner of master, if he accepts then the merge will happen. (under pull requests)

After merging, we can delete the other branch if required.

Merging with Master:- (Locally)

Change back to master branch using "git checkout master".

Perform "git pull" to stay in sync with github.

"git merge <branch-name" to merge to master.

"git push" to keep changes synced with github

Now as changes are integrated to master, we can delete this branch from github.

we still have the references locally, so delete the branch in local using "git branch -d <branch name>"

"git fetch -p" (pruning removed branches)

① Create a new branch in github
② Fetch it in local by using git fetch
③ Goto that branch using git checkout <branch name>
④ Make some changes in new branch & commit changes.
⑤ git push to add the changes to github new branch

Deleting All References locally :-

① Make a change to ReadMe file in new branch on github.
② We can pull the changes here by checking out to master branch locally & then typing "git pull -all" to get all the changes.
③ Merge changes with master by being in master branch & then typing "git merge <branchname>"
④ "Git push" to sync changes.
⑤ Git branch -d <branchname> to delete new branch
⑥ We still have it in github, we can delete that branch locally by — "git push origin : <branchname>"

Pull with Rebase:-

① Edit Readme on Github.

② Make changes to index.html & commit it locally.

③ Git fetch to get the contents from github.

Now we see that branches have been diverged.

④ Type "git pull --rebase" and our head will get rewinded before index.html was edited.

⑤ Type "git hist" an alias equivalent to getting log with its options, we see our head, origin pointing to commit that contains changes done in ①

⑥ Objective is to make local changes ahead of remote.

⑦ git push will make the resolution to conflicts.

Github Graphs:-

Really we can have graph by typing → git log --oneline --graph

In github, goto Graphs → Network to get the same representation

Changing default branch:-

1. We can do it under settings in github.
2. This will change the way, pull push & forks are done
3. When pull request is done, the destination changes.
4. When cloned. it will use the new default branch.

Dealing with conflict while pulling:-

1. Make changes to Readme in Github & commit.
2. Make changes to Readme locally & commit.
3. This results in conflict.
4. We can use our mergetool to fix this (f4 merge)
5. Make changes to Readme on downside bar & Save.
6. Now add & commit the Readme, that should fix it.

Tags & Releases:-

1. git log --online
2. git tag → To list the tags present in repo.
3. git tag <tagname> <reference>

Usually reference is branchname, then it applies to last commit of the specified branch.

④ git hist alias will list the graphical notation of commits and our tag will be present on the topmost commit.

⑤ We can also add the annotated tags for specific commit by:- "git tag -a <tagname> -m <Message> <commitID>"

⑥ git show <tagname> to get the commit details.

Local tags to Github:-

① We can get the tags in Github by clicking on Releases tab.

② git push origin <tagname> to push that tag to Github.

③ git push --tags → To push all tags to Github.

Deleting Tags on Github:-

① Go to Releases → Tags → TagName → Delete

② To delete from local repo, first fetch changes from Github

(i) git fetch -p

(ii) git tag -d <tagname> to delete the tag.

③ We can delete the tag locally & sync with github to delete the tag on remote repo also :-

(i) git tag -d <tagname>

(ii) git push origin : <tagname to delete>

Updating tags :-

① Add and commit the changes to Readme file locally, that will create a commit ahead of remote repo.

② git tag -f <tagname> <commitID> to update the tag.

③ Push the commits to github, if we push the updated tag using git push origin <tagname> then github rejects saying already a tag exists.

④ So we use git push --force origin <tagname>

Starting Release on Github :-

① We can add release notes by going to Releases → tags → Add Release notes.

② Diff b/w tag & Releases is, we get to see the Release notes in Releases section & only tags info under tags.

③ We can create a release along with new tag by clicking on "Draft a new Release" button under Releases tab.

④ git pull to get the new tag to be pulled locally.

Comparing Pull Requests from Github:-

① Pull requests → Create pull request

② Select the source & destination branch to check the changes & compare them below using the split view.

Comparing Commits:-

① We can go to commits tab & select a commit to get the difference of current & previous commits.

② Or go to pull requests & select source branch & destination commit ID.

Comparing Tags:-

① Go to pull requests page and search for tags in dest and source to get the differences.

Advanced comparison:-

① In the dest and source search for branches & if we want to compare last three days changes then we can write

<branchname>@{3days}

<branchname>@{YYYY-MM-DD}

Social Coding :-

Sharing the code using Fork & accepting other changes using pull requests.

① Forking the repo using Fork button on top of repo.

② It is similar to cloning other's repo.

③ Clone the forked repo in local repo using git clone command.

git clone <SSH-URL>

④ Create a new branch "git checkout -b <branch"

⑤ Add a readme file and commit the file.

⑥ Push the repo to github using "git push -u origin <new branch>"

⑦ Now you will be able to see the pushed new branch.

⑧ Change the branch to new branch in github & then Create a pull-request, that will send a pull request to original owner from whom the repo was forked.

⑨ We can also add new commits & push to same branch.

⑩ The owner reviews the changes and merges pull request if interested or closes the pull request.

Github Graphs:-

① Pulse section is used to get the stats of the issues, pull requests, commits for specified period.

② Graph section deals with contributors, traffic graph, commits, Additions/deletions, Punch Card, Network & members.

Synchronising the forks:-

In the dashboard, you find repos you contribute to, click of any one & select the cloning option from SSH & then clone the repo locally using:- ~~"git clone <SSH-URL>~~

① git remote add upstream <SSH-URL>

② checkout to required branch & then git pull to get the changes :- git pull upstream master.

③ Now as we have all changes, type "git push origin master"

④ we can also remove not required branch then git branch -d <branchname>

⑩ git fetch -p to fetch and sync with github.

Collaborators :-

The contributers that add the changes to our code. We can add them by going to settings > collaborators > Search by name > Add ~~cont~~ collaborator.

Github Issues :-

① Click on repo on Github, If you don't see Issues option go to settings > Features & turn on the issues.

② Labels are used to categorise issues → Bug, duplicate, help wanted, Question etc.

③ We can edit Labelnames & colours. & we can add new labels.

④ Milestones are something related to something happening at particular time. Click on Create Milestone, type the name & description, select the date and create it.

⑤ Click on create Issue, Add label, description, Milestone and Assignee if required and click submit new issue.

⑥ Pull requests are also counted as Issues in Github.

⑦ After doing necessary actions, go to Issue and navigate down

and write the resolution & either comment or close the issue.

⑧ Assume a new issue is raised to exclude the MAC files separately, we can fix this locally like this :-

(i) Navigate to repo & do git pull to sync.

(ii) Open .gitignore and add the MAC files in it.

(iii) Write a commit like this "git commit -am "Added MAC files, close # <Issue Num>"

(iv) git push to push to github.

(v) This will add the comment & close the issue.

⑨ We can associate commit with Issues, open a commit and write "Associating # <Issue Num>" & comment it.

Github Gists :-

It is an simple way of sharing code, entire file or snip.

① In dashboard, click Gists

② We can add a code snippet for doing something, add description such as "JAVA HOME ENV" and add the name of file with extension and the code snippet in the textbox below & select Public Gists.

③ In local, we need to clone the Gist & by taking the clone URL by:-

git clone <SSH URL>

④ Sharing of Gists is done by sharing URLs of gists to the users

⑤ Gists can be deleted using Delete button and the cloned gists must be removed from our local repo.

Github Organisations:-

Grouping related repos.

① On dashboard, click on your name's dropdown to select Organisation, if not present then add new organisation.
(or) select the + symbol.

② Click on repo & go to settings & select Transfer and enter the organisation name to be transferred.

③ We can create the copy of repo in organisation to our personal account by clicking on Fork & selecting our personal account.

④ To make changes back to local, copy the SSH URL of repo that was forked from organisation & add the remote to local

(i) git remote add v <SSH URL>
<name>

(ii) git remote -v to get names of remotes.

(iii) git fetch upstream → To get from organisation repo

(5) To deal with different branches of remote,

(i) Add a new branch to our organisation repo.

(ii) New branch under personal account.
Add

(iii) git fetch --all to get two ~~different~~ same branches but the branch is added due to same names.
We cannot access the branch.

(iv) We need to switch to any one of these branches using :- git checkout -b shared origin/shared

(v) git pull to sync up.

(6) We can create a team and add new members to it.

(7) Under team's repo tab, we can add repo & provide the permissions

(8) We can move members from one team to another, create new team, add user & remove him from old team.