PULL REQUESTS

to checkout a pull request ———> git fetch origin pull/$NUMBER/head:pr-$NUMBER
git checkout pr-$NUMBER

BRANCHES

to rename any (local) branch ———> git branch -m <oldname> <newname>
show all remote branches (names) ----------> git branch -r
information about remote and local relationships ----------> git remote show origin
see the last commit on each local branch ----------> git branch -v
which branches are already merged into the branch you're on —> git branch --merged
branches that contain work you haven't yet merged in -----> git branch --no-merged
to do a fast forward when branch u are on is behind master --> git merge --ff-only origin/master
push the local branch <branchname> to remote repo and make the local one to track the remote one --------> git push -u origin <branchname>
delete branch
remote ———> git push origin --delete <branchName> OR git push origin :<branchName>
local ———> git branch -d <branchName>
to create a branch that tracks a remote branch ------> git checkout -b refac origin/Refactoring (Branch refac set up to track remote branch Refactoring from origin. Switched to a new branch 'refac')
reset local branch ***staging*** to latest snapshot at remote branch ***staging*** ———>

1. git fetch origin
2. git reset --hard origin/staging

find recently deleted branches ———> git reflog
save changes in current workspace without committing them, switch to any other branch, do the work, come back to original branch and resume from where you left
save changes in current workspace ———> git stash
apply most recent stash to the local branch ———> git stash apply or git stash apply stash@{0}
apply one of the older stashes (third most recent one in this example) ———> git stash apply stash@{2}

TAGS

checkout a tag ----> git checkout -b version2 v2.0.0
create a tag ---> git tag -a v1.4 -m 'my version 1.4'
check/show tag ----> git show v1.4
push tag to remote: git push origin <tag_name>
to make a tag point to a particular commit after the tag has been created ----> git tag -a v1.2 9fceb02
to add a folder to gitignore : open the file named 'exclude' in the directory root\.git\info. add the name of the folder at the end - e.g. to get Git to ignore any changes to the directory named '.idea' or any file inside it, just write .idea/ at the end of (but at start of a new line in) the exclude file.

VIEWING CHANGES, COMPARING and MERGING

check what has changed after a pull ----> git diff @{1}
check what has changed in a single file after a pull ———> git diff @{1} <filename>
view a commit at github website when u just have the shortened hash of the commit ——>

1. first get the full commit hash by ***git log -p -1 <shortened_commit_hash>***
2. open any commit in the branch for which a full commit hash has was just found in and replace the last part of the url with the hash from pre step.

see changes made in a file and by whom and when ——> navigate to the repo and then to the file on github and then press the "Blame" button in the upper-right part of the panel
check **branches that contain a commit** ———>
from local branches: git branch --contains <commit_hash>

from remote branches: git branch -r --contains <commit_hash>

FILE OPERATION

Reverting changes to a single file:

- from the file's (dirty) state in working directory to its state at the most recent commit -------> git checkout -- file_path
- from the file's state in the last commit to its state in another commit in history ———> git checkout **d4b87a3** file_path (will revive its state in commit d4b87a3).

stage deleted files ------> git rm $(git ls-files —deleted)
delete/remove an untracked file ——> git clean -f file_path
check diff of file with its snapshot in last commit of current branch, when the file has not yet been ***add***ed to staging area ————> git diff -- filename.ext
but if the file has been added but not yet committed ———> git diff --cached -- filename.ext

COMMITING

change commit msg for last commit on a local branch ———> git commit --amend -m "New commit message"
revert to previous commit at local branch as well as at remote branch ———>

1. ***git revert*** (for local)
2. ***git push origin <branch_name>*** (for remote)

revert an old merge commit ———>

- ***git revert*** <merge_commit#> -m <parent_to_keep*>

  parent_to_keep** is the parent that is kept. the other parent of the **merge_commit** is reverted. e.g. if we want to revert the merge commit with hash ***2fa52cf1, we can check its parents using the command ***git show 2fa52cf1***. This reveals the following info: commit 2fa52cf1db99b1293a309b213738959593f4676e
  Merge: **d4b87a3 53953b3**
  Now, if we wanted to revert the parent commit **d4b87a3** and keep **53953b3** we would do it with ***git revert 2fa52cf1 -m 2***. However, if we wanted to revert **53953b3** and keep **d4b87a3** we would accomplish that by ***git revert 2fa52cf1 -m 1***. so, **parent_to_keep** tells git which parent from the ones shown in the commit log of the **merge_commit** to keep.

find next/child commit of a know commit ———> git log --reverse --ancestry-path <known_commit_hash>^..

CONFLICT RESOLUTION

The ======= sign marks the point that divides the implementations of the contentious code chunk among the two branches. The part above the ======= sign represents the code chunk as in the branch currently in focus, master in this case, while the part after the ======= sign represents the code chunk as in the branch that is to be merged.

QUERYING

View commit history with most recent on top ———> git for-each-ref --sort=-committerdate refs/heads/