LOCAL

- cat .git/HEAD to check the head pointer
- cat .git/config
- see the change clearly: `git diff --color-words`
- for `commit -a` : be careful for new file and deleted file not include with modify (short cut)
- when commit , try put all conceptual context together. different concept do different commit so easy to keep track
- Undoing working directory files:
  - scenario where mistakenly to delete something . Now want to put it back where it was:
    - `git checkout - - index.html` (make sure use - - to differentiate with check out branch when it may has the same name)
- Undoing staged area:
  - Scenario where you add bunch of file but one or some you do not want include it when committing
  - `git reset HEAD resources.html`

- Amending information to the same file from last commit:
  - Real world scenario where you just committed your file.Then deciding to change some in that file or edit the commit message. By doing amend, the whole changed content will be added to the last commit
  - git commit --amend -m “messageContent”
  - we can only able to do the recent commit where the head points to
- Retrieving the old version:
  - Real world : you made changes, but somehow you want to put back the old version.(you do not want to show your mistake you made.
  - git checkout <first10SHAofThepreviousofRecent Commit> - - fileName (this put fileName back to old version in staging area) and re commit it .Then we will have the old version. But without committing

at this point and un stage ==> git checkout – – fileName (from working area) then nothing happen

- Revert a commit:
  - scenario like retrieving the old version,but this revert will revert the change and commit without waiting for you do a commit
  - git revert <ShaNumberOfthelatestOne>
- Using Reset to undo commit (**be cautious**)
  - Take control the head pointer by set it where you want to be then start record from there. it will overwrite anything after recording from that point:
  - Reset Soft: git reset –-soft e4ec8bd88b1f898d (the previous immediate):
    - What happen by doing this:move the head pointer to the previous one. In this method we can only move the

    - head pointer nothing changed. NOTHING CHANGED PUT FILE BACK TO STAGED AREA
- Reset mixed: git reset --mixed e4ec8bd88b1f898d (previous immediate).
    - What happen: nothing changed, just move the head pointer to the previous one . Different from soft is put the file back all the way to working area for modification not staged area
- reset hard: git reset ——hard e4ec8bd88b1f898d (previous immediate):
    - What happen :head pointer move to the previous one. And the method put it back previous status and automatically and discard the recent commit;YOU CAN GO BACK IF YOU STILL KEEP TRACK THE COMMIT HASH NUMBER
    - THE DIFFERENCES HARD VS.SOFT:soft and mixed give you chance to change your

self by git diff to see the changes and make your own decision whether keep the same commit (make sure to keep track the commit hash to go back using the same method). The HARD just move the pointer and reverse the context to before got changed

- Remove untracked files:
  - That means you just add couple files into your local repo. now still sitting in untracked file:
    - git clean -f

BRANCHES:

- git branch
- git branch branchName
- git checkout branchName
- when creating branch will carry the entire context from master to branches.
- git checkout -b short_title (create and check out at the same time)
- Note: when one file get modified, you will get warning when trying to switch branches. so there are three options:

    - going back to commit the file
    - checkout the file off the working directory (git checkout – – fileName)
    - Stash them for later
- Compare Branches:
    - git diff branch1..branch2
- Rename branch:
    - git branch –m branch1 branch2 (m mean move)
    - git branch –d branchName (deletion)
        - Note: must get out the branch you want to delete in order to delete it
        - if you just commited on the delete branch, when you try to delete it Git will make sure by make you use capitalize letter D instead of d
    -

MERGE Branch:

- Note:To merge you have to be on receiver side

- git merge mergeBranchName
- after merging, git diff master..mergeBranchName (to see if any more differences)

Merge Conflict:

- Happen when you merge something that system get confuse with the same content do not know which one should keep:
- 3 ways to solve conflict:
  - abort: git merge – – abort
  - fix manually
  - using tool
- ways to avoid:
  - merge often (easy to catch)
  - after merging, we still able to keep the same branch to work on and merge again over and over
  - keep track change from master by merging master to branch sometimes so their content can be synced.so should not cause conflict down the road

Stash:

- Scenario where we change some thing

but not ready to commit yet but want to switch branch the system would not let you switch without commit. So Stash helps us to save it and come back later

- git stash save “specific message”

View Stash:

- git stash list . you should keep track this stash@{0} for later call
- git show stash@{0}

Retrieve Stash:

- git stash pop stash@{numberYouWant} (take out completely)
- git stash apply stash@{}(take out still keep the copy)

Delete Stash:

- git stash drop stash@{number}

REMOTE

- Adding remote:
  - git remote add origin httpCopy
- Create remote branch:
  - git push –u origin master (whatever branch you want to push)
  - cat .git/config (to check)
- Pulling project from git

    - git clone httpCopy
    - if somehow does not have branch tracking we can :
        - git branch – – set-upstream non_tracking origin/ nontracking
    - with tracking branch setup (– when first time push we always use push –u), any time we push, all we have to do is git push should be taking care of without having to declare where to push ….
    - master vs. origin/master
- Fetching: a process when we collaborate with others, after all the clone , add remote … done. Every day we come to work, we should do FETCHING . Why ?????? Because during period time the remote github could get update by pushing code from others that your local repository DO NOT know about that till you fetch
- after fetching, we can do : git log – –oneline –5 origin/master
- guidelines for fetching:

- fetch before work
- fetch before you push
- fetch often

Process to do :

- git fetch
- git branch –a, git branch –r
- compare : git diff origin/master..master (to check the differences)
- Then do : git merge origin/master
- Then do git log – –oneline –5 origin/master . now all the change there
- Git pull = git fetch + git merge.RECOMMENDATION : fetch and merge for rookie
- fetch – merge– before push

Deleting remote branch :

- same with push branch to remote but :git push origin :non–tracking
- git push origin – –delete non_tracking

Collaboration:

- go to github page :click admin and add that person user name to it so

both can work together

- for public or company work you need a permission to merge to their code:
  - Fork is an option:
    - before folk checking network and issue tab to see what is going on
    - then fork to your account and work till you are comfortable with
    - then click pull request button so they can look at you code and merge it to the main one
- Collaboration work flow:
- MY Part:
  - git checkout master
  - git fetch (to see what change when the last time we checked in)
  - git merge origin/master
  - git checkout –b feeback_form (create branch with yourProjectName). working on feedback.html

    - git add feedback.html
    - git commit –m “Add customer feedback form”
    - now I want to push out there for my coworker look at it
    - git fetch (before push i should do git fetch one more time to see any change)
    - Now ok to do : git push –u origin feeback_form (branch : feedback_form)
    - call your coworker: let her know that branch you just push up there so see can look
- CoWorker part:
    - git checkout master
    - git fetch
    - git branch –a (will show her all the branches)
    - git merge origin/master
    - git checkout –b feedback_form origin/feeback_form
    - To see what i did see do git log
    - or she can do git show shaNUmber (to look at particular). She may make some change and commit

    it
  - git commit –am “ Adding something”
  - git fetch
  - git push
- My part again:
  - git fetch
  - git log –p feedback_form..origin/feeback_form
  - after looking all the feedback . looking good
  - git merge origin/feedback_form
  - git check out master
  - git fetch
  - git merge origin/master
  - git push