My Git Notes

August 24, 2022

Contents

1 Branches

branches are separate isolate workspace to prevent making main line of work unstable. unlike subversion which copy entire working directory git just creating a new pointer to creating a new branch. so master branch is just a pointer to the last commit on the main branch. HEAD pointer used for pointing to the current branch.

git branch branch-name	create new branch
git branch	list of branches
git status	find out the current branch
git switch branch-name	switch the branch-name
git branch -m old-name new-name	renaming branch
git log –oneline –all	see commit across all branches
git branch -d branch-name	deleting branch
git branch -D branch-name	force deleting branch
git switch -C branch-name	create and change branch
git branch –merged	list of merged branches
git branch –no-merged	list of no merged branches

if didn't merge the branch deleting may occur a error

1.1 Comparing Branches

git log master..bugfix	show all commits in the bugfix
git diff master..bugfix	show all the changes
git diff bugfix	show differences between current branch and bugfix branch
git diff [–name-only –name-status] bugfix	show differences between current branch and bugfix branch

1.2 Stash

if we have some changes in some branch and didn't commit yet changes could get lost by changing branch because when changing branch git reset our working directory to the last snapshot that saved on the latest commit on the target branch.
stashing something means store it on the safe place.
by default new untrack files are not included in the stash
each stash has a unique identifier.

git stash push -m 'massage'	create new stash
git stash push -am 'massage'	to contain new untrack files
git stash list	list of stash
git stash show stash-id	show the changes
git stash apply stash-id	apply the stash
git stash drop stash-id	delete the stash
git stash clear	remove all the stash

1.3 Merging

bring up changes from a branch to another branch. use –graph option with git command its a better representation of git branches and diverge.

1.3.1 Fast Forward Merge

when the main line of work didn't have diverge and we have a direct linear path between two branches. disabling fast forward merging may have some benefits (reverting the commits are easier)

git merge branch-name	merge branch-name to the current branch
git merge –no-ff branch-name	merge branch-name to the current branch with no fast forward option
git config –global ff no	disable fast forward global
git config ff no	disable fast forward on the current repository

1.3.2 3-Way Merge

when the main line of work have diverge git look the last snapshot of two branches and make merge commit and merge the changes.

1.3.3 Conflict

happens when the same line of code changes in the different branches, or delete the file in another branch or add the same file with different contents in two branches
with the git status command see the file(s) that cause the problem
open files and solve the problem (REMEMBER DO NOT ADD NEW LINE OF CODE)
add the file to the staging area then use git commit message. kdiff and p4merge are cross platform graphical tools to resolving the conflict.
git config –global merge.tool p4merge
git config –global mergetool.p4merge.path "path/to/the/p4merge"
git merge –abort back to the state before we stated the merge

1.3.4 Undoing Faulty Merge

two ways:

rewriting the history(deleting the merge commit) both the master and the head point to the last commit which is the merge commit we should move both of them to point to the commit before creating branch by using git reset command, the merge commit determine garbage by the git and one in a while remove by the git
git reset –hard HEAD∼1

reset command options

soft effect on last snapshot

mixed effect on last snapshot and staging area

hard effect on last snapshot, staging area and working directory

reverting the merge the last commit have two parents one in the master branch(1) and one in the bugfix branch(2) so we should use the revert command like this git revert -m 1 HEAD

the first approach is ok if this history is local in our local repository but its not the best practice if we are working on the remote repository in this situation we should use the second approach we creating a new commit and cancel all changes on this commit.

1.3.5 Squash Merging

if we don't have good commits on the branches we combines all the changes in the branch and then merge it to the master branch so we have clean linear history path this is what we call squash merging this new commit not a merge commit because it doesn't have two parent commit. we should use for small and short lives branches like bug fixes of features can implemented in a few hours or a day.
git merge –squash branch-name(add all changes to the staging area ready to commit)
git commit
it's very IMPORTANT TO DELETE THE BRANCH unlike it may occur conflict in the future because git shows it on the unmerged branches so delete it with -D option

1.3.6 Rebasing

an other technique to bringing changes from a branch to another
rebasing mean change the base commit of a branch e.g we have bug fix branch which master v1.1 is its base commit of the branch and the master branch current version is v.1.2 so we have to use 3 way merge method or change the base commit (rebase) of the current branch to the master v.1.2 and we can use fast forward merge its may sounds a cool idea but rebasing rewrites the history and only on local repository we should use it.

Why rebasing is rewrites the history?
what the git actually do is that, git create new commits like the commits on the branch with the base we want then move the branch pointer to point the new commits, git do not change the commit because commits on git are immutable

git switch target/branch
git rebase master(tell git to rebase the branch to the last commit of the master)
if rebasing occur conflict we should resolving the conflict like before then use

the below command
git rebase –continue
git rebase –skip(skip the current commit and move on the next commit)
git rebase –abort(abort the rebase and take us back to previous state before rebasing)
git config –global mergetool.keepbackup false

1.3.7 Chary Picking

think of a situation that we're working on a branch and we have interesting commit whit some changes and we want to add this commit to the master but we're not ready to merge the branch so we chary pick the particular topic and apply it on the top of the master
on master: git charypick commit-id
if there's a conflict resolve it then commit the changes.

1.3.8 Picking Files From Another Branch

like chary picking but interested on a single file not whole commit
git restore –source=branch-name –file-name(– for file name is optional and if git doesn't recognize the file name)