**How to get the current branch name in Git?**

*Ans: git branch*

*should show all the local branches of your repo. The starred branch is your current branch.*

*If you want to retrieve only the name of the branch you are on, you can do:*

*git branch | grep \\* | cut -d ' ' -f2*

**How can I delete all Git branches which have been merged?**

*UPDATE:*

*You can add other branches to exclude like master and dev if your workflow has those as a possible ancestor. Usually I branch off of a "sprint-start" tag and master, dev and qa are not ancestors.*

*To delete all local branches that are already merged into the currently checked out branch:*

*git branch --merged | egrep -v "(^\\*|master|dev)" | xargs git branch -d*

*You can see that master and dev are excluded in case they are an ancestor.*

*You can delete a merged local branch with:*

*git branch -d branchname*

*If it's not merged, use:*

*git branch -D branchname*

*To delete it from the remote in old versions of Git use:*

*git push origin :branchname*

*In more recent versions of Git use:*

*git push --delete origin branchname*

*Once you delete the branch from the remote, you can prune to get rid of remote tracking branches with:*

*git remote prune origin*

*or prune individual remote tracking branches, as the other answer suggests, with:*

*git branch -dr branchname*

*Hope this helps.*

*Share*

**Finding what branch a git commit came from**

*686*

*down vote*

*accepted*

*+50*

*While Dav is correct that the information isn't directly stored, that doesn't mean you can't ever find out. Here are a few things you can do.*

*Find branches the commit is on*

*git branch --contains <commit>*

*This will tell you all branches which have the given commit in their history. Obviously this is less useful if the commit's already been merged.*

*Search the reflogs*

*If you are working in the repository in which the commit was made, you can search the reflogs for the line for that commit. Reflogs older than 90 days are pruned by git-gc, so if the commit's too old, you won't find it. That said, you can do this:*

*git reflog show --all | grep a871742*

*to find commit a871742. The output should be something like this:*

*a871742 refs/heads/completion@{0}: commit (amend): mpc-completion: total rewrite*

*indicating that the commit was made on the branch "completion". The default output shows abbreviated commit hashes, so be sure not to search for the full hash or you won't find anything.*

*git reflog show is actually just an alias for git log -g --abbrev-commit --pretty=oneline, so if you want to fiddle with the output format to make different things available to grep for, that's your starting point!*

*If you're not working in the repository where the commit was made, the best you can do in this case is examine the reflogs and find when the commit was first introduced to your repo; with any luck, you fetched the branch it was committed to. This is a bit more complex, because you can't walk both the commit tree and reflogs simultaneously. You'd want to parse the reflog output, examining each hash to see if it contains the desired commit or not.*

*Find a subsequent merge commit*

*This is workflow-dependent, but with good workflows, commits are made on development branches which are then merged in. You could do this:*

*git log --merges <commit>..*

*to see merge commits that have the given commit as an ancestor. (If the commit was only merged once, the first one should be the merge you're after; otherwise you'll have to examine a few, I suppose.) The merge commit message should contain the branch name that was merged.*

*If you want to be able to count on doing this, you may want to use the --no-ff option to git merge to force merge commit creation even in the fast-forward case. (Don't get too eager, though, that could become obfuscating if overused.) VonC's answer to a related question helpfully elaborates on this topic.*

***Listing each branch and its last revision's date in git***

142

down vote

accepted

commandlinefu has 2 interesting propositions:

for k in `git branch | perl -pe s/^..//`; do echo -e `git show --pretty=format:"%Cgreen%ci %Cblue%cr%Creset" $k -- | head -n 1`\\t$k; done | sort -r

or:

for k in `git branch | sed s/^..//`; do echo -e `git log -1 --pretty=format:"%Cgreen%ci %Cblue%cr%Creset" $k --`\\t"$k";done | sort

That is for local branches, in a Unix syntax. Using git branch -r, you can similarly show remote branches:

for k in `git branch -r | perl -pe 's/^..(.\*?)( ->.\*)?$/\1/'`; do echo -e `git show --pretty=format:"%Cgreen%ci %Cblue%cr%Creset" $k -- | head -n 1`\\t$k; done | sort -r

**How to automatically push after committing in git?**

*112*

*down vote*

*accepted*

*First, make sure that you can push manually without providing your password. If you are pushing over HTTP or HTTPS, that will be a case of either creating a .netrc file with the login details or adding your username and password into the URL for the remote. If you're using SSH, you can either create a keypair where the private key doesn't have a password, or use ssh-agent to cache your private key.*

*Then you should create an executable (chmod +x) file in .git/hooks/post-commit that contains the following:*

*#!/bin/sh*

*git push origin master*

*... customizing that line if you want to push to a remote other than origin, or push a branch other than master. Make sure that you make that file executable.*

***How to remove local (untracked) files from the current Git working tree?***

*As per the Git Documentation git clean*

*Remove untracked files from the working tree*

*Step 1 is to show what will be deleted by using the -n option:*

*git clean -n*

*Clean Step - beware: this will delete files:*

*git clean -f*

*To remove directories, run git clean -f -d or git clean -fd*

*To remove ignored files, run git clean -f -X or git clean -fX*

*To remove ignored and non-ignored files, run git clean -f -x or git clean -fx*

*Note the case difference on the X for the two latter commands.*

*If clean.requireForce is set to "true" (the default) in your configuration, one needs to specify -f otherwise nothing will actually happen.*

***How do I delete a Git branch both locally and remotely?***

*Executive Summary*

*$ git push --delete <remote\_name> <branch\_name>*

*$ git branch -d <branch\_name>*

*Note that in most cases the remote name is origin.*

*Delete Local Branch*

*To delete the local branch use one of the following:*

*$ git branch -d branch\_name*

*$ git branch -D branch\_name*

*Note: The -d option is an alias for --delete, which only deletes the branch if it has already been fully merged in its upstream branch. You could also use -D, which is an alias for --delete --force, which deletes the branch "irrespective of its merged status." [Source: man git-branch]*

*Delete Remote Branch [Updated on 8-Sep-2017]*

*As of Git v1.7.0, you can delete a remote branch using*

*$ git push <remote\_name> --delete <branch\_name>*

*which might be easier to remember than*

*$ git push <remote\_name> :<branch\_name>*

*which was added in Git v1.5.0 "to delete a remote branch or a tag."*

*Starting on Git v2.8.0 you can also use git push with the -d option as an alias for --delete.*

*Therefore, the version of Git you have installed will dictate whether you need to use the easier or harder syntax.*