

# Git Branching Tutorial with Real-World Examples



Bahadır Mezgil

Jan 20 · 9 min read



Git is the most popular version control system (VCS) in the world for keeping track of text-based file changes. It also provides a distributed collaborative work environment. In other words, it is suitable for teamwork.

One of the most important features of Git like many VCS is the Branching feature which enables us to open a new channel for a specific set of changes like a bug fix, new feature implementation or an experiment without interfering with the mainstream or parallel changes. But unlike others in git, the branching is a very lightweight operation.

In this article, I will show you the usages of Git Branching with real-world examples including how to create a branch, update it, delete it, switch to another branch, and merge them locally and remotely.

First, let me give you some information about the two commonly used git branches for software projects.

## Commonly Used Software Project Branches

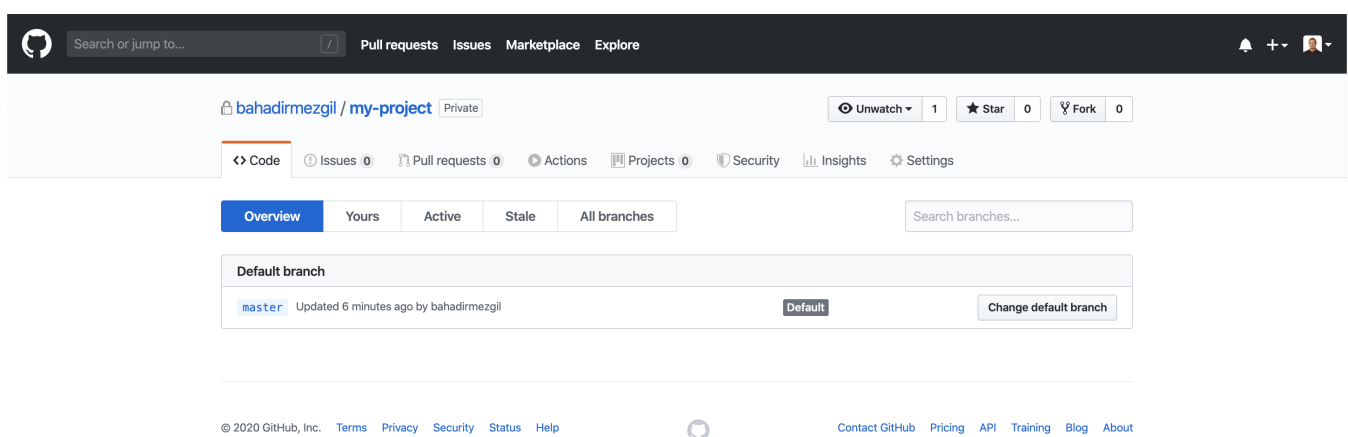
Most of the software projects have two mainstream branches as **master** and **dev**. The master branch is used for ready for production changes and the dev branch is used for testing purposes. When new changes are tested on the dev branch, eventually it will be merged into the master branch and finally, we will take a deployment on the master branch. Of course, these branch names and numbers can be changed but these are the most common ones.

Like many developers, we should work neither on the master branch nor on the dev branch directly. For every tiny or massive change in a software project, we should first create a new branch from the dev branch by naming the new branch related to our new change.

Now I can show you how git branching is used in software developers' daily life with real examples.

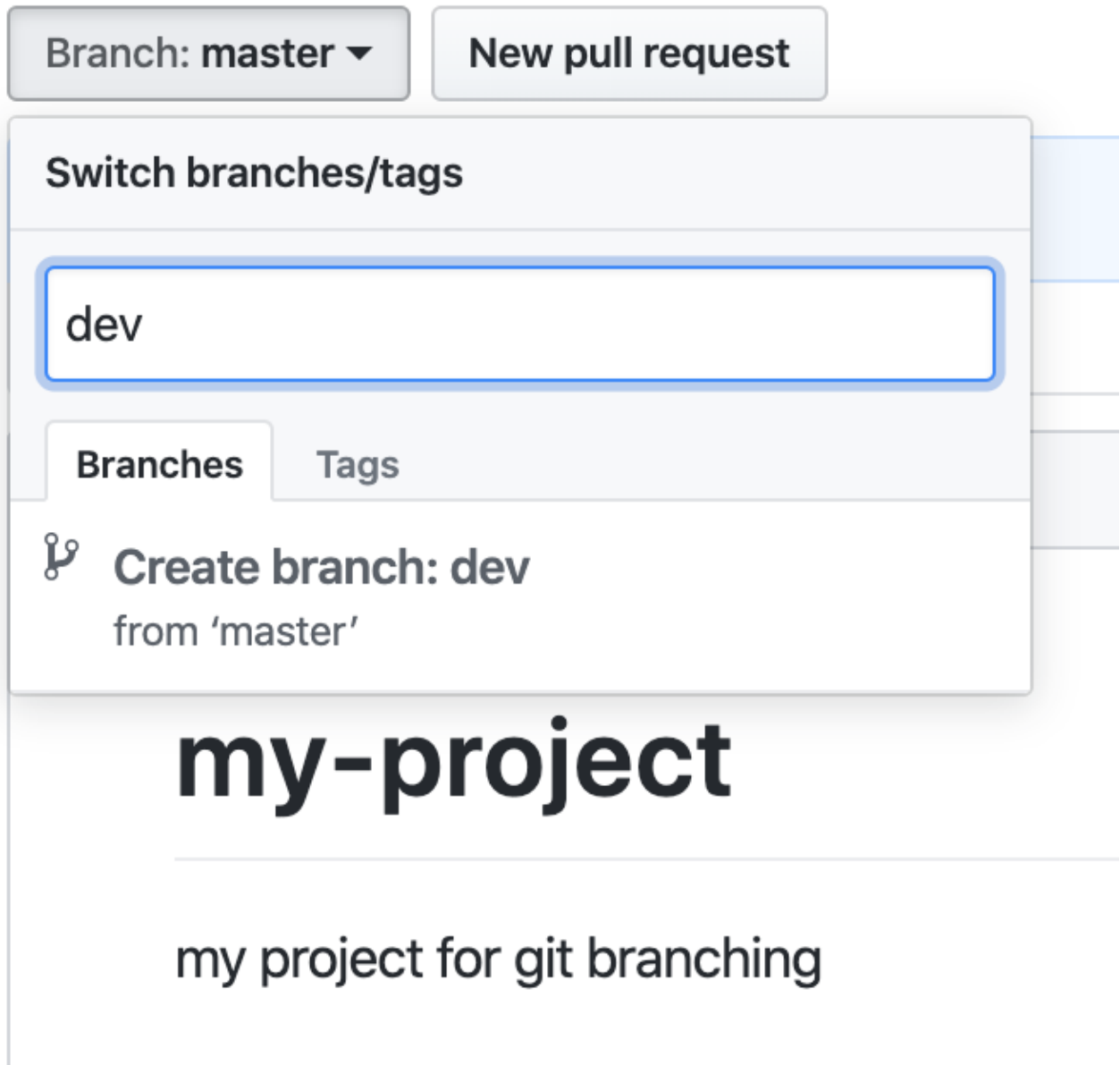
## Setting Remote and Local Branches

Let me assume that I already have a GitHub repository. Go to the homepage of that repository and click the “1 branch” section to see what is going on.



As you can see I have only the “master” branch which is created by default. Also, it is the default branch that indicates that repository homepage content is loaded as the master branch is selected.

I go back to the homepage, and I click the master branch dropdown. As I said before most software projects have common branches as **master** and **dev**. By typing “dev” and clicking the appeared “Create branch: dev” section, and I can create the dev branch on my remote repository.



# my-project

my project for git branching

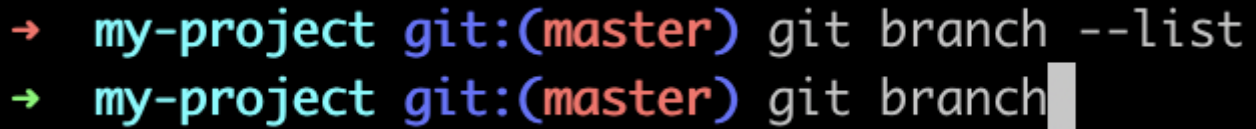
## Create a Branch on Local Repository

But what about my local repository. I want to check my local git repo after adding the dev branch to the remote repository with another common git command to list my local branches:

```
$git branch
```

or the same command

```
$git branch --list
```



```
→ my-project git:(master) git branch --list  
→ my-project git:(master) git branch
```

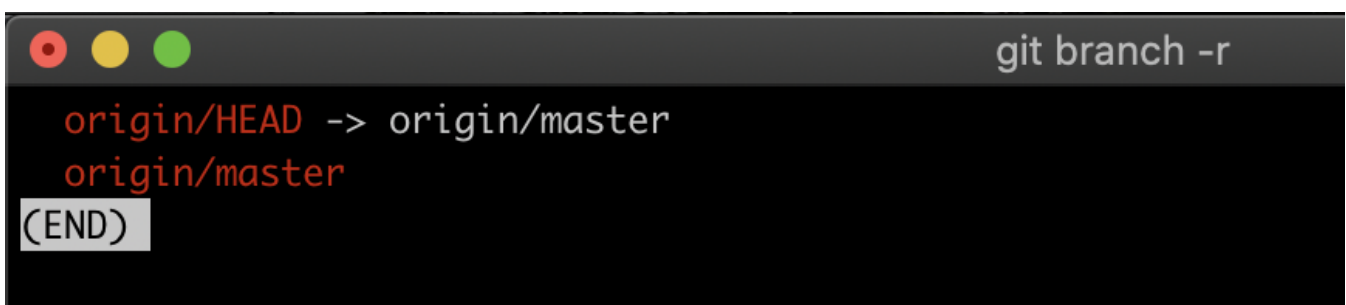
This command outputs your local branches in a new window of the terminal.



If you want to exit from that branch list page, just hit the q key from your keyboard.

But I realize that I couldn't see the dev branch. If you want to see your remote repository branches, we can use the same git command with an option.

```
$git branch -r
```



## Fetching Changes from Remote Repository

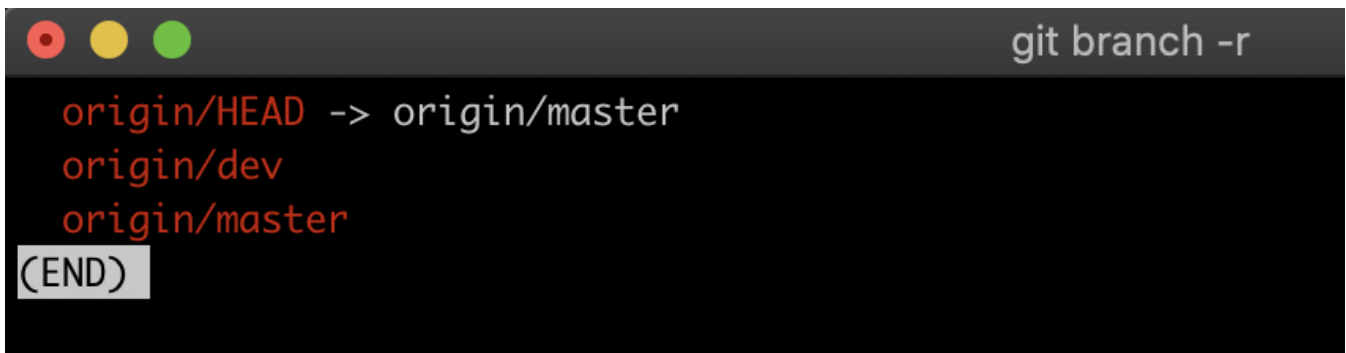
But the remote dev branch is not shown here either. Ok so let me try to check out to that branch with another common git command. Why?

Because first I need to fetch all up to date changes from my remote repo with **git fetch** command:

```
$git fetch --all
```

A terminal window with a dark background. The title bar shows three colored circles (red, yellow, green) and the text 'bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project'. The terminal content shows the command '→ my-project git:(master) git fetch --all' followed by the output: 'Fetching origin', 'From https://github.com/bahadirmezgil/my-project', '\* [new branch] dev -> origin/dev', and '→ my-project git:(master) ' with a cursor.

Now I can see the dev branch among other up to date remote branches because by **git fetch** we retrieved the latest remote metadata without any copy operation.

A terminal window with a dark background. The title bar shows three colored circles (red, yellow, green) and the text 'git branch -r'. The terminal content shows the output of 'git branch -r': 'origin/HEAD -> origin/master', 'origin/dev', 'origin/master', and '(END)'.

But in order to retrieve and copy all the remote changes, we should use the **git pull** command.

```
$git pull origin dev
```

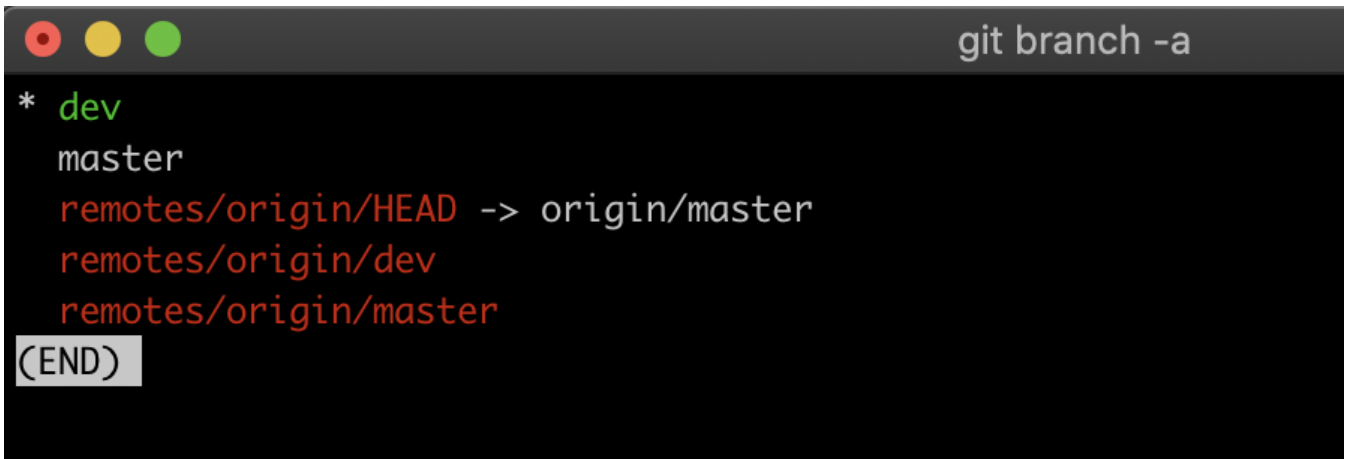
---

*In git as a convention, **origin** keyword indicates the remote repository*

---

So We have updated our branch information. To see both remote and local branches, we can use another **git branch** command option:

```
$git branch -a
```



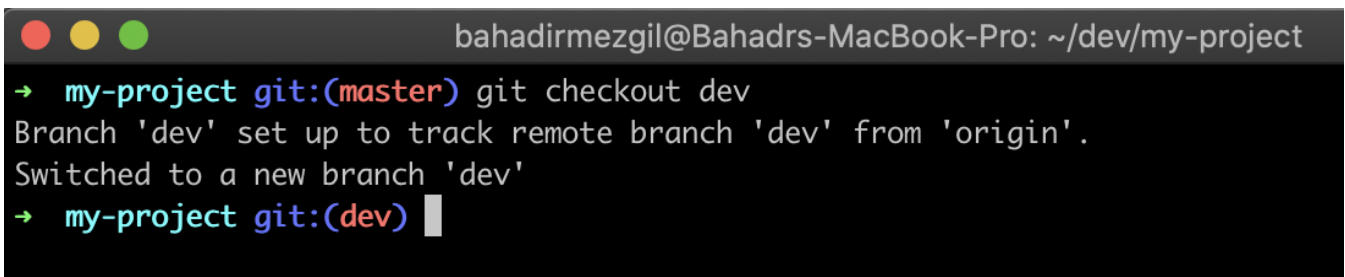
```
git branch -a

* dev
master
remotes/origin/HEAD -> origin/master
remotes/origin/dev
remotes/origin/master
(END)
```

## Switching Branches

Now we are ready for development and just before starting, we need to change our current branch from master to dev by using a very popular git command on our local repository.

```
$git checkout dev
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(master) git checkout dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
Switched to a new branch 'dev'
→ my-project git:(dev)
```

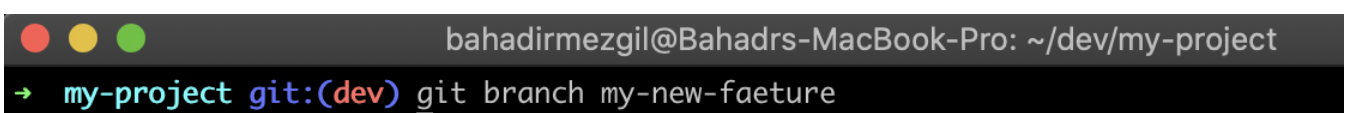
The dev branch which will be our main development branch. All development operations will be originated from that branch.

## Branch CRUD Operations

### Creating Branch

Let me create a new branch from the dev branch with a very common git command:

```
$git branch my-new-faecture
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(dev) git branch my-new-faecture
```

```
→ my-project git:(dev)
```

Let me check my local branches again.

A terminal window titled "git branch" with standard macOS window controls (red, yellow, green buttons). The output of the command is: 

```
* dev
master
my-new-faecture
(END)
```

## Updating Branch Name

But wait a minute! I realized that I made a mistake while naming my new branch. No problem because we have already a git command for that.

```
$git branch -m my-bug-fix
```

If you are in a different branch, you can still rename any of your branches without checking out:

```
$git branch -m my-new-faecture my-new-feature
```

Now I can check my correctly named branch.

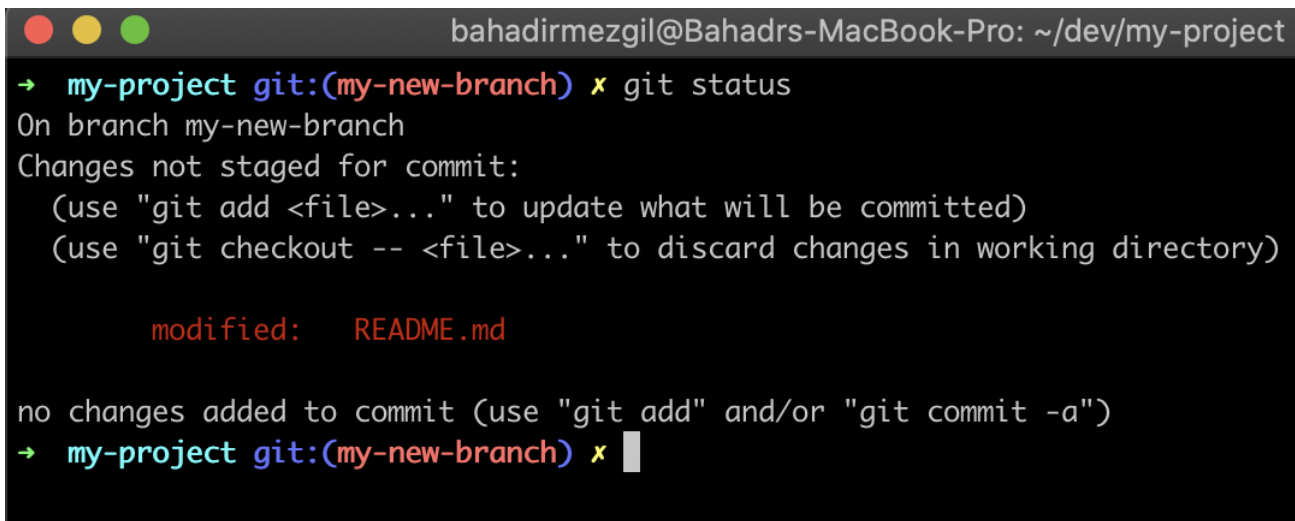
A terminal window titled "git branch" with standard macOS window controls. The output of the command is: 

```
* dev
master
my-new-feature
(END)
```

## Committing Changes to Branch

Let's make some gibberish changes on README.md file then check our local repository with **git status** command.

```
$git status
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(my-new-branch) ✕ git status
On branch my-new-branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
→ my-project git:(my-new-branch) ✕
```

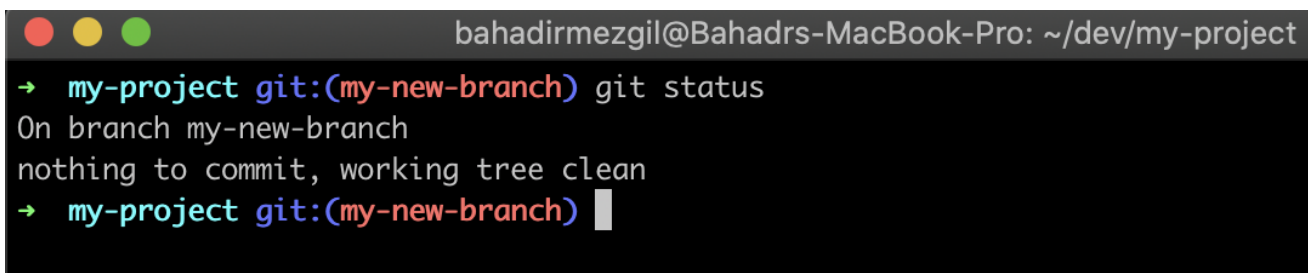
Let me add this modified file to the staging area and make my first commit.

```
$git add README.md
$git commit -m "my first commit"
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(my-new-branch) ✕ git add README.md
→ my-project git:(my-new-branch) ✕ git commit -m "my first commit"
[my-new-branch 7596d51] my first commit
1 file changed, 1 insertion(+)
→ my-project git:(my-new-branch)
```

Let me check my git repository again with the **git status** command.



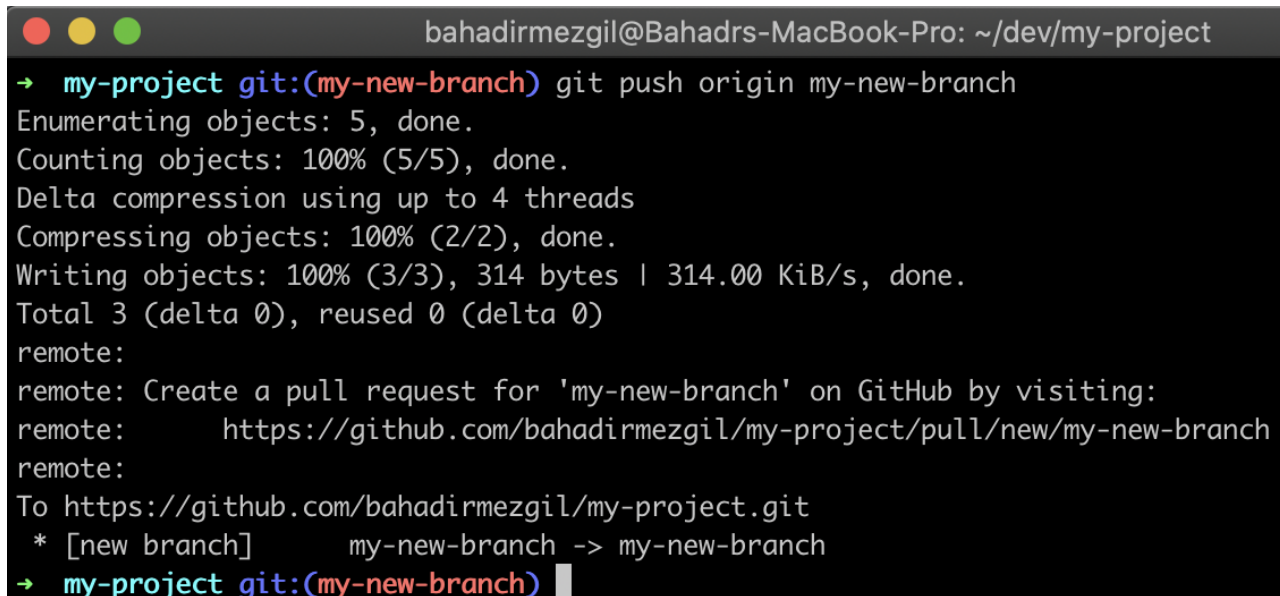
```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(my-new-branch) git status
On branch my-new-branch
nothing to commit, working tree clean
→ my-project git:(my-new-branch)
```

## Sending All Changes to Remote



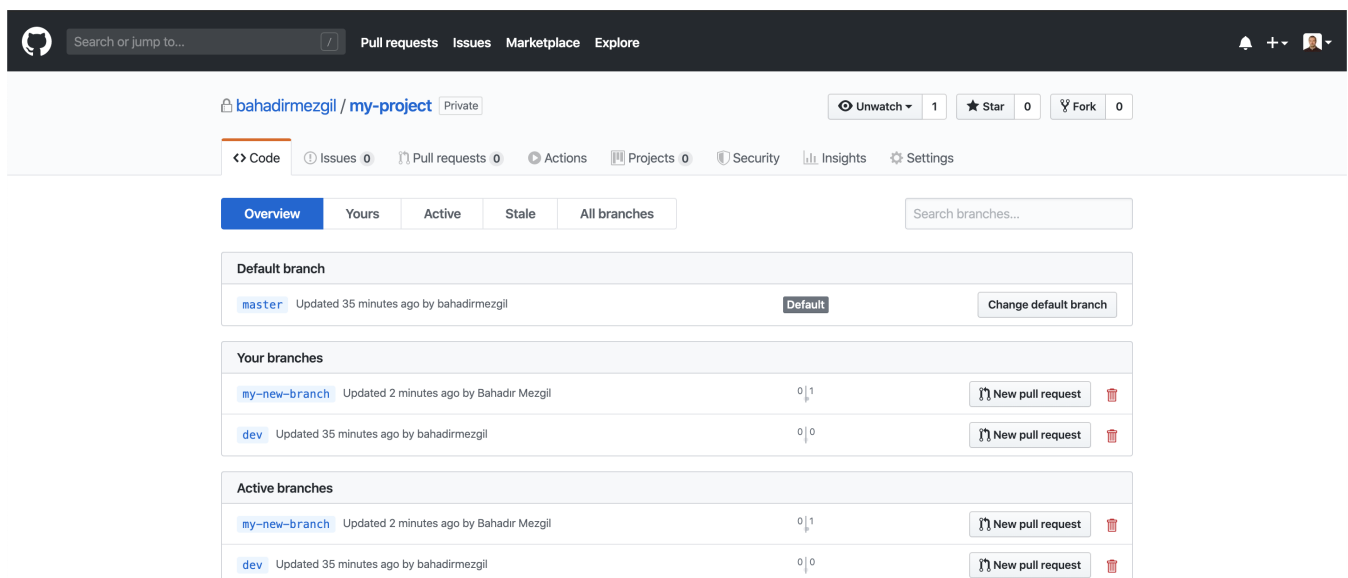
Now I am ready to push my first commit with my-new-branch. Just like the **pull** command, this time we will use the **push**.

```
$git push origin my-new-branch
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(my-new-branch) git push origin my-new-branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'my-new-branch' on GitHub by visiting:
remote:   https://github.com/bahadirmezgil/my-project/pull/new/my-new-branch
remote:
To https://github.com/bahadirmezgil/my-project.git
 * [new branch]      my-new-branch -> my-new-branch
→ my-project git:(my-new-branch)
```

After pushing our commit with our new branch, we can check the remote repository from the branches section. You could see the new branch.



The screenshot shows the GitHub interface for the repository 'bahadirmezgil / my-project'. The 'Branches' tab is selected, displaying a list of branches. The 'my-new-branch' is listed as an active branch, updated 2 minutes ago by Bahadır Mezgil. It has 0 commits and 1 pull request. The 'dev' branch is also listed, updated 35 minutes ago by bahadirmezgil, with 0 commits and 0 pull requests. The 'master' branch is the default branch, updated 35 minutes ago by bahadirmezgil.

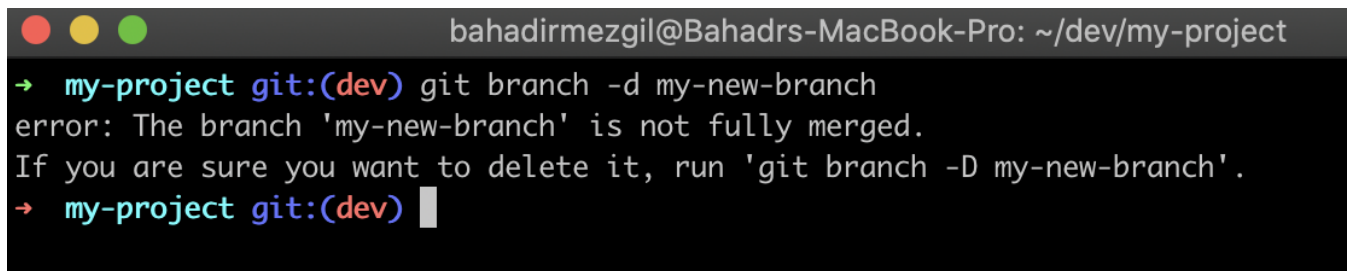
Branch	Updated	By	Commits	Pull Requests
my-new-branch	Updated 2 minutes ago	Bahadır Mezgil	0   1	New pull request
dev	Updated 35 minutes ago	bahadirmezgil	0   0	New pull request
master	Updated 35 minutes ago	bahadirmezgil	0   0	New pull request

## Deleting Branch

But what if I really made a mistake by creating that branch and I really ashamed of that. I need to get rid of that branch.

First, I need to go to the dev branch (Actually It can be any branches other than you want to delete). Then I can safely delete my unwanted local branch:

```
$git checkout dev
$git branch -d my-new-branch
```

A terminal window with a dark background and light-colored text. The title bar shows the user 'bahadirmezgil' on a 'Bahadrs-MacBook-Pro' in the directory '~/dev/my-project'. The prompt is 'my-project git:(dev)'. The user enters 'git branch -d my-new-branch'. The terminal outputs an error: 'error: The branch 'my-new-branch' is not fully merged. If you are sure you want to delete it, run 'git branch -D my-new-branch'.'. The prompt returns to 'my-project git:(dev)' with a cursor.

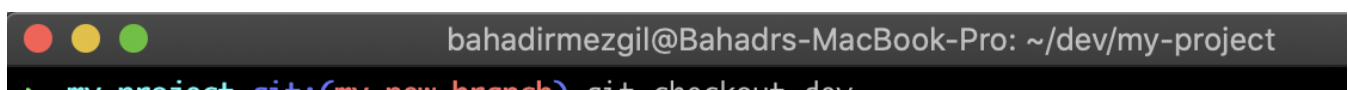
The delete operation is failed. Because deleting a local branch with **git branch -d** command is a safe operation. If your branch has unpushed or unmerged commits it will not be deleted safely. If you still insist on deleting that branch, you can use the **git branch** with the **-D** option.

```
$git branch -D my-new-branch
```

A terminal window with a dark background and light-colored text. The title bar shows the user 'bahadirmezgil' on a 'Bahadrs-MacBook-Pro' in the directory '~/dev/my-project'. The prompt is 'my-project git:(dev)'. The user enters 'git branch -D my-new-branch'. The terminal outputs 'Deleted branch my-new-branch (was 7596d51)'. The prompt returns to 'my-project git:(dev)' with a cursor.

What about deleting a remote branch? This time instead of using the **git branch** command, we should push this delete request with the “delete” option.

```
$git push origin --delete my-new-branch
```

A terminal window with a dark background and light-colored text. The title bar shows the user 'bahadirmezgil' on a 'Bahadrs-MacBook-Pro' in the directory '~/dev/my-project'. The prompt is 'my-project git:(my-new-branch)'. The user enters 'git checkout dev'.

```
my-project git:(my-new-branch) git checkout dev
Switched to branch 'dev'
Your branch is up to date with 'origin/dev'.
→ my-project git:(dev) git push origin --delete my-new-branch
To https://github.com/bahadirmezgil/my-project.git
- [deleted]          my-new-branch
→ my-project git:(dev) █
```

Finally, let me check all local and remote branches.

```
git branch -a
* dev
master
remotes/origin/HEAD -> origin/master
remotes/origin/dev
remotes/origin/master
(END)
```

As you can see my new branch is cleaned from all local and remote repositories.

## New Feature Implementation with Branching

Now let's be serious and create a branch like a professional for my new feature implementation.

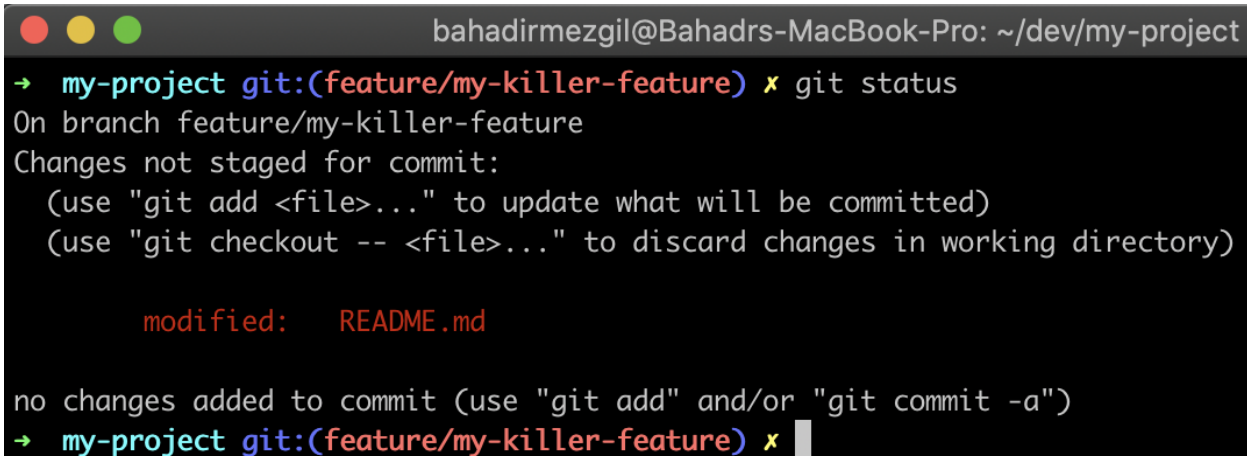
```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(dev) git checkout -b feature/my-killer-feature
Switched to a new branch 'feature/my-killer-feature'
→ my-project git:(feature/my-killer-feature) █
```

Let me make a few changes in my repository.

```
README.md — my-project
M+ README.md ×
M+ README.md > [abc] # my-project
    You, a few seconds ago | 2 authors (Bahadir Mezgil and others)
1  # my-project
2  my project for git branching
3  my killer feature    You, a few seconds ago • Uncommitted changes
```

After making some changes or to be sure about the local git repository status, we can check with the **git status** command.

```
$git status
```

A terminal window with a dark background and light text. The title bar shows the user 'bahadirmezgil' on a 'Bahadrs-MacBook-Pro' in the directory '~/dev/my-project'. The prompt is 'my-project git:(feature/my-killer-feature)'. The command 'git status' has been executed. The output shows the current branch, instructions for staging changes, and a list of modified files (README.md).

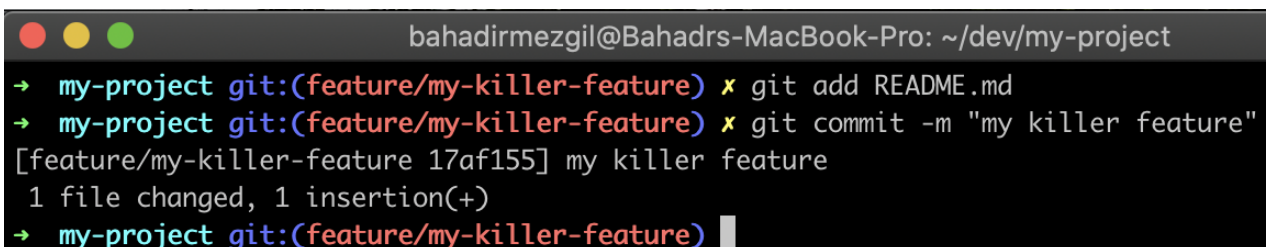
```
→ my-project git:(feature/my-killer-feature) ✕ git status
On branch feature/my-killer-feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
→ my-project git:(feature/my-killer-feature) ✕
```

I see my modified file as README.md. Let me commit the current local repository change by first adding the modified file to the staging area, second writing a commit message about the change.

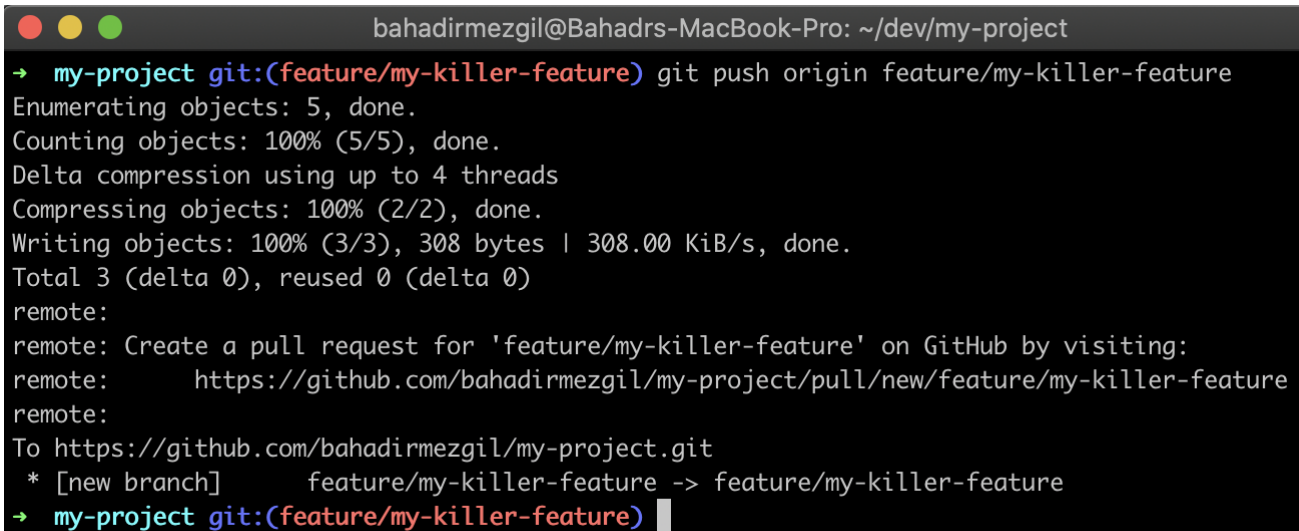
```
$git add README.md
$git commit -m "my killer feature"
```

A terminal window showing the execution of 'git add' and 'git commit' commands. The title bar is the same as the previous screenshot. The prompt is 'my-project git:(feature/my-killer-feature)'. The first command 'git add README.md' is executed. The second command 'git commit -m "my killer feature"' is executed, resulting in a commit hash '17af155'.

```
→ my-project git:(feature/my-killer-feature) ✕ git add README.md
→ my-project git:(feature/my-killer-feature) ✕ git commit -m "my killer feature"
[feature/my-killer-feature 17af155] my killer feature
1 file changed, 1 insertion(+)
→ my-project git:(feature/my-killer-feature) ✕
```

Now we are ready to send our commit to the remote repository by git push command which will also send our branch information.

```
$git push origin feature/my-killer-feature
```



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/dev/my-project
→ my-project git:(feature/my-killer-feature) git push origin feature/my-killer-feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'feature/my-killer-feature' on GitHub by visiting:
remote:   https://github.com/bahadirmezgil/my-project/pull/new/feature/my-killer-feature
remote:
To https://github.com/bahadirmezgil/my-project.git
 * [new branch]      feature/my-killer-feature -> feature/my-killer-feature
→ my-project git:(feature/my-killer-feature)
```

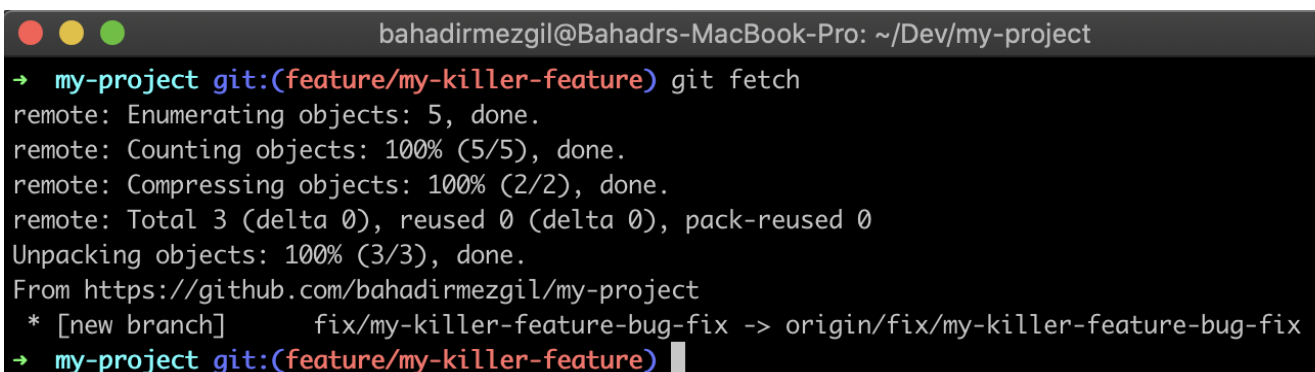
So I am ready to make a pull request to the dev branch from *feature/my-killer-feature* branch. But I went to sleep and next morning my coworker texted me as “Hey Bahadır, I find a bug on your new feature implementation. If you are ok about that I will fix it.” Of course, as every sleepy person, I texted back with a “yes”.

Then my friend first retrieved my latest change by **git fetch** and **git pull** commands in order to check out to our feature branch then he made the changes and pushed them in a new branch called *fix/my-killer-feature-bug-fix*.

After a good sleep, We need to take these changes into our branch right? Thanks to git we have a very common git command which is **git merge**.

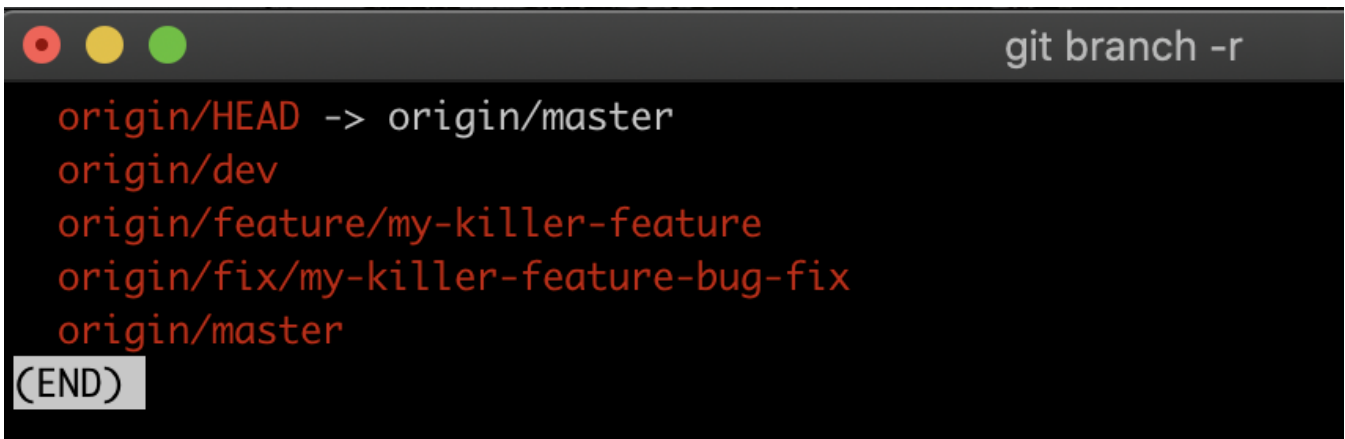
## Merging Branches

First I should fetch up to date information about the remote repository.



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/Dev/my-project
→ my-project git:(feature/my-killer-feature) git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/bahadirmezgil/my-project
 * [new branch]      fix/my-killer-feature-bug-fix -> origin/fix/my-killer-feature-bug-fix
→ my-project git:(feature/my-killer-feature)
```

Then I can check the remote branches to find the branch my friend is worked on.



```
git branch -r
origin/HEAD -> origin/master
origin/dev
origin/feature/my-killer-feature
origin/fix/my-killer-feature-bug-fix
origin/master
(END)
```

Now I know the remote branch name to merge into my feature branch with git merge command. Before that just be sure you are in the right branch.

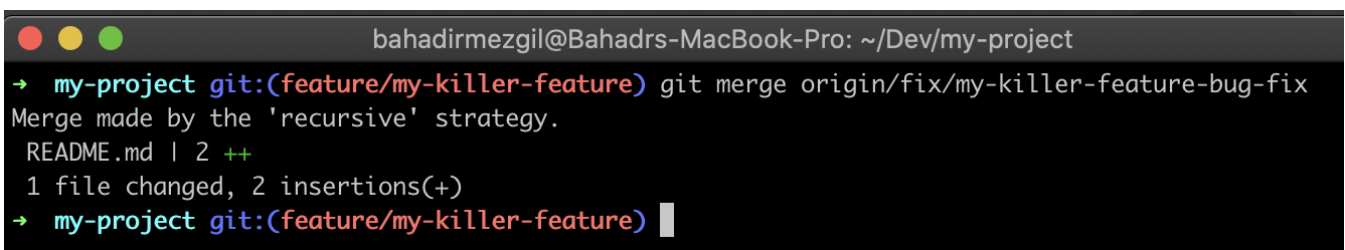
```
$git merge origin/fix/my-killer-feature-bug-fix
```

Then the default editor will be opened to show you about the changes. Exit from the vi editor by typing “:q”.



```
git merge origin/fix/my-killer-feature-bug-fix
Merge remote-tracking branch 'origin/fix/my-killer-feature-bug-fix' into feature/my-killer-feature
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

Then you would see a summary of the made changes in the current branch.



```
bahadirmezgil@Bahadrs-MacBook-Pro: ~/Dev/my-project
→ my-project git:(feature/my-killer-feature) git merge origin/fix/my-killer-feature-bug-fix
Merge made by the 'recursive' strategy.
 README.md | 2 ++
 1 file changed, 2 insertions(+)
→ my-project git:(feature/my-killer-feature)
```

## Conclusion

I hope you learn or strengthen your knowledge about how to use git branching on a daily base by creating or deleting branches locally or remotely, making changes with

branches, merging them.

**If you want to learn more about programming, you can visit our website [www.codingdocs.com](http://www.codingdocs.com) and subscribe to our newsletter. If you have any questions, please don't hesitate to contact me.**

See you in new articles.

[Git](#)[Github](#)[Software Development](#)[Software Engineering](#)[Programming](#)[About](#)[Help](#)[Legal](#)