**Git – Cherry Pick**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 06 May, 2022

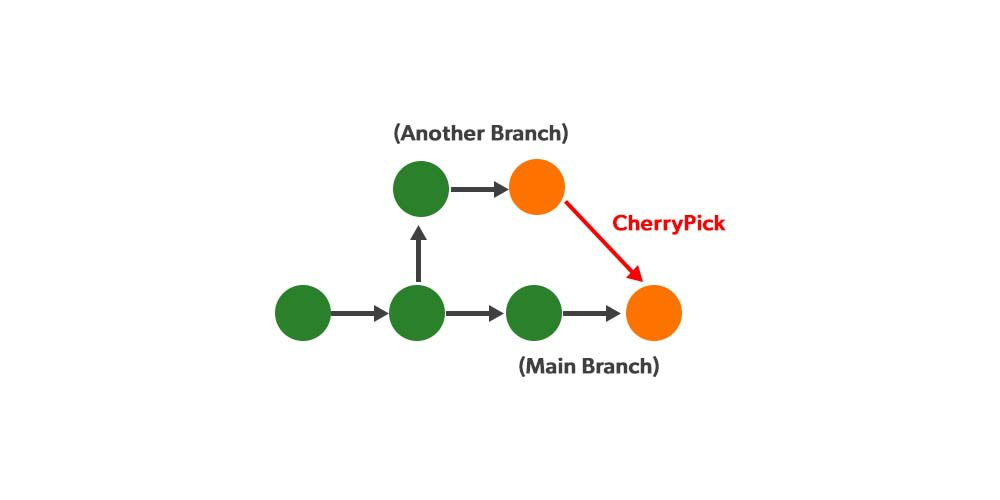
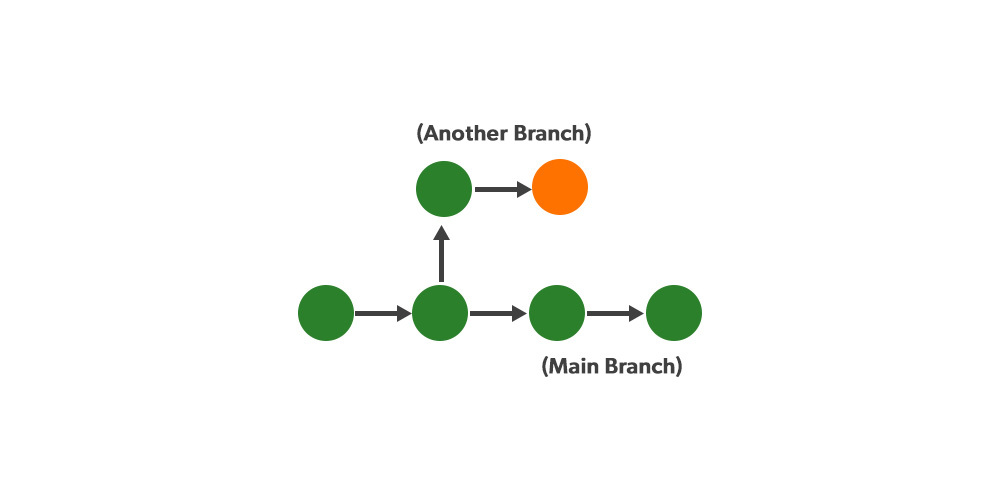
Cherry-picking in git means choosing a commit from one branch and applying it to another branch. This is in contrast with other ways such as merge and rebases which normally apply many commits into another branch.

Cherry-picking is just like [**rebasing**](https://www.geeksforgeeks.org/rebasing-of-branches-in-git/)**,**an advanced concept and also a powerful command. It is mainly used if you don’t want to merge the whole branch and you want some of the commits.

**Implementation:**

Suppose a developer fails to recognize which branch he is currently on, and by mistake, he commits to another branch instead of committing that to the main branch. Now to fix it, he has to first run **git show,**then save the commit, then check out the main branch, apply patch there and commit with the same commit message. But all this can be done automatically by using just one command i.e. **cherry-pick.**

In order to understand better refer to the below diagram as follows:



The command for Cherry-pick is as follows:

git cherry-pick<commit-hash>

**Commit hash:**A commit hash is a unique identifier that is generated by Git. Each commit has its one commit hash.

***Note:****While using this command make sure you are on the branch you want to apply the commit.*

**Some important Usecases of Cherry-pick are as follows:**

1. If you by mistake make a commit in an incorrect branch, then using cherry-pick you can apply the required changes.
2. Suppose when the same data structure is to be used by both the frontend and backend of a project. Then a developer can use cherry-pick to pick the commit and use it to his/her part of the project.
3. At the point when a bug is found it is critical to convey a fix to end clients as fast as could be expected.
4. Some of the time a component branch might go old and not get converged into the main branch and the request might get closed, but since git never loses those commits, it can be cherry-picked and it would be back.

**Disadvantages of using Cherry Pick**

Cherry-pick should not be used always as it can cause copy commits and numerous situations where cherry-picking would work, conventional merges are liked all things considered. Also, in the situation where the commits from 2 or more branches update similar lines of code with various substances and git cherry-pick one commit to the other branch, it prompts conflict as well.

**How to use cherry-pick?**

Here is the step by step explanation of the use of cherry-pick command in the below-created project stepwise shown below as follows:

**Step 1:**Opening the [**git bash**](https://www.geeksforgeeks.org/working-on-git-bash/) and creating a new project named **sample**and initializing the repo using the **git init** command.

Text

Description automatically generated

**Step 2:**Creating a *‘.txt’* file using**vi**command to the project let’s say an index file and add it to our sample project and make a commit and write a commit message before pressing the Enter.

***Note:****After running the****vi <file\_name>****command , type :wq to save and quit the file.*

Text

Description automatically generated

One can check your commit by **git log**command easily:

Text

Description automatically generated

**Step 3:**Now assume we have 2 versions, so create 2 different branches by using the **git branch** command and move to a branch, let us say 2 by using **git checkout** command.

***Note:****One can simple view all the branches by running the git branch command as shown in the below diagram.*

Text

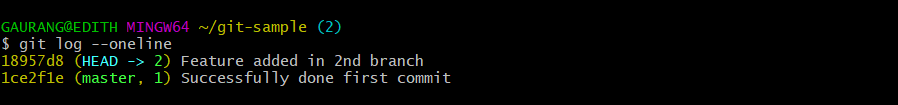
Description automatically generated

**Step 4:**Now suppose you want to work on some new feature, so creating and adding a new feature file lets say feature.txt using **vi** and **add** command respectively as shown below. Then commit your changes with a commit message.

Text

Description automatically generated

One can check your commit by git log command as shown below:



*It is clearly showing our first commit where our branch 1 is there and in branch 2 it has moved away farther ahead and we are currently working on our feature in branch 2*

**Step 5:** Now suppose we found a bug in our feature and we came to know that this same bug is also present in our 1 branch.

And now we are trying to fix some bug or issue as shown below by adding a fix.txt file let’s suppose and add it to the current branch i.e. 2 and committing the required changes.

Graphical user interface, text

Description automatically generated

*Checking our final commits:*

Text

Description automatically generated

**Step 6:**Now,we have fixed the bug in branch 2, but we also need to add this fix to our branch 1 also, but we don’t want to merge this branch 2 into our branch 1, because the work might still be going on the feature.

Thus, in this scenario, we can cherry-pick this particular commit. To do so, just copy the **hash value**highlighted in the above diagram, then move to branch 1 by using checkout, and then use the command **cherry-pick**and paste the hash we just copied.

Graphical user interface, text, application

Description automatically generated

As seen clearly from the above, then we notice that previously we have only index.txt before doing cherry-picking, but now we have the fix.txt file also in our 1st branch.

Now if we try to check **git log –oneline**, we will be able to see that commit also came in the branch 1.

A picture containing text

Description automatically generated