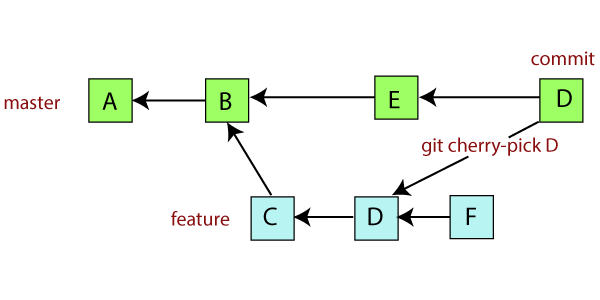
**Git Cherry-pick**

Cherry-picking in Git stands for applying some commit from one branch into another branch. In case you made a mistake and committed a change into the wrong branch, but do not want to merge the whole branch. You can revert the commit and apply it on another branch.



The main motive of a cherry-pick is to apply the changes introduced by some existing commit. A cherry-pick looks at a previous commit in the repository history and update the changes that were part of that last commit to the current working tree. The definition is straight forward, yet it is more complicated when someone tries to cherry-pick a commit, or even cherry-pick from another branch.

Cherry-pick is a useful tool, but always it is not a good option. It can cause duplicate commits and some other scenarios where other merges are preferred instead of cherry-picking. It is a useful tool for a few situations. It is in contrast with different ways such as **merge** and **rebase** command. Merge and rebase can usually apply many commits in another branch.

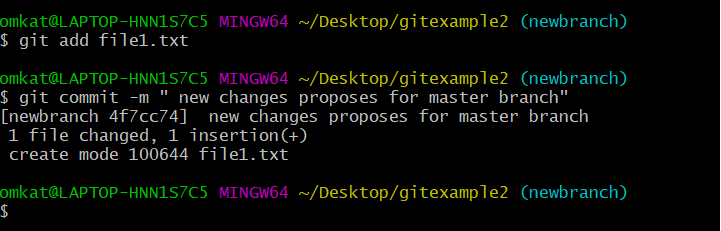
**Why Cherry-Pick**

Suppose you are working with a team of developers on a medium to large-sized project. Some changes proposed by another team member and you want to apply some of them to your main project, not all. Since managing the changes between several Git branches can become a complex task, and you don't want to merge a whole branch into another branch. You only need to pick one or two specific commits. To pick some changes into your main project branch from other branches is called cherry-picking.

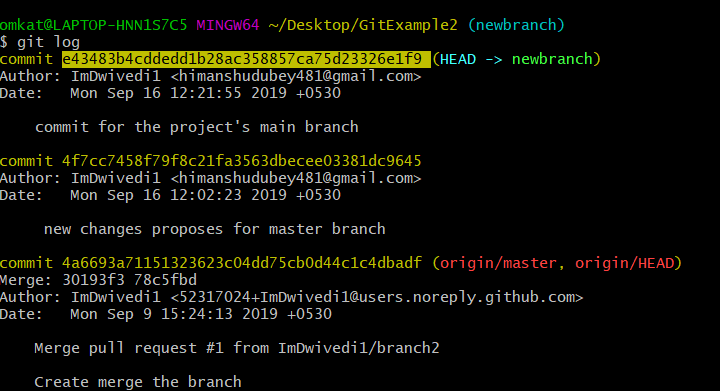
**Some scenarios in which you can cherry-pick:**

**Scenerio1: Accidently make a commit in a wrong branch.**

Git cherry-pick is helpful to apply the changes that are accidentally made in the wrong branch. Suppose I want to make a commit in the master branch, but by mistake, we made it in any other branch. See the below commit.



In the above example, I want to make a commit for the master branch, but accidentally I made it in the new branch. To make all the changes of the new branch into the master branch, we will use the git pull, but for this particular commit, we will use git cherry-pick command. See the below output:

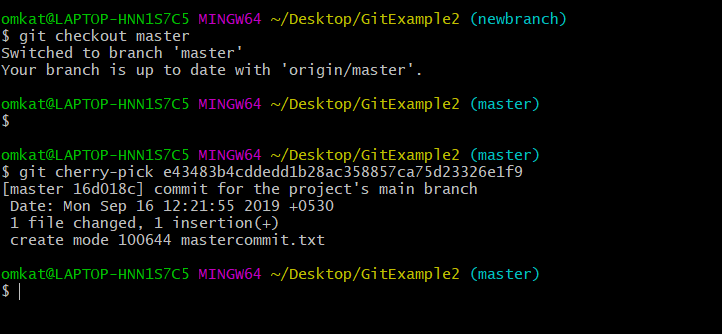


In the given output, I have used the git log command to check the commit history. Copy the particular commit id that you want to make on the master branch. Now switch to master branch and cherry-pick it there. See the below output:

**Syntax:**

1. $ git cherry-pick <commit id>

**Output:**



From the given output, you can see that I have pasted the commit id with git cherry-pick command and made that commit into my master branch. You can check it by git log command.

**Scenario2: Made the changes proposes by another team member.**

Another use of cherry-picking is to make the changes proposed by another team member. Suppose one of my team members made any changes in the main project and suggests it for the main project. You can cheery-pick it after review.

**Usage of cherry-pick**

* It is a handy tool for **team collaboration**.
* It is necessary in case of **bug fixing** because bugs are fixed and tested in the development branch with their commits.
* It is mostly used in **undoing changes and restoring lost commits**.
* You can **avoid useless conflicts** by using git cherry-pick instead of other options.
* It is a useful tool when a full branch merge is not possible due to incompatible versions in the various branches.
* The git cherry-pick is used to access the changes introduced to a sub-branch, without changing the branch.