Git – Cherry Pick

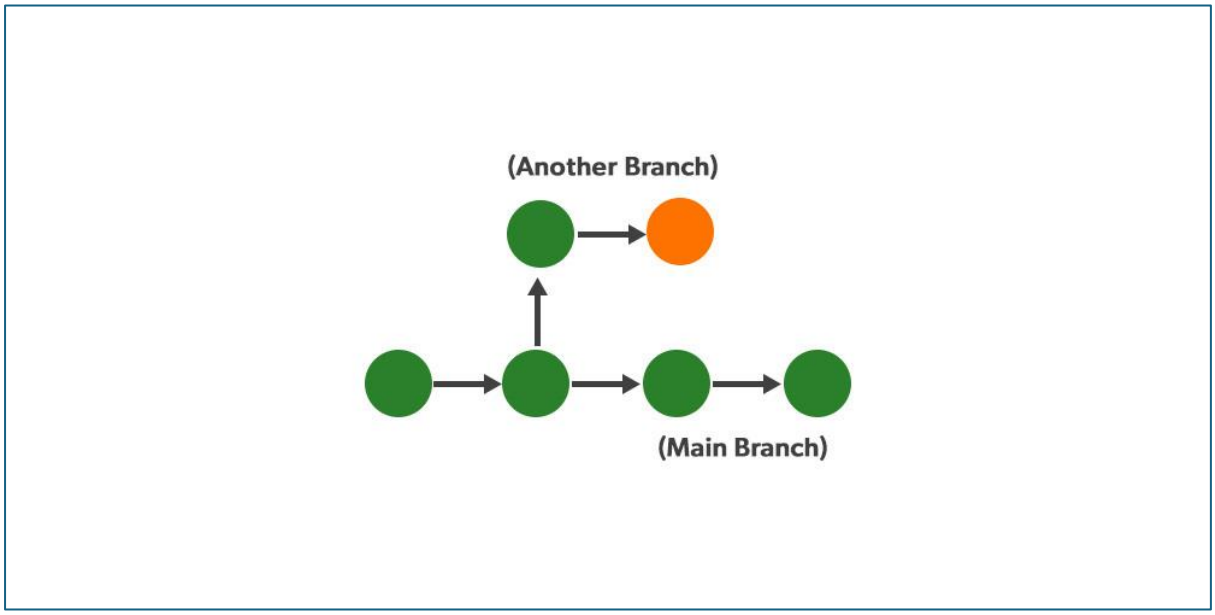
git **cherry-pick** in git means choosing a commit from one branch and applying it to another branch. This is in contrast with other ways such as **merge** and **rebases** which normally apply many commits into another branch.

git cherry-pick is just like **rebasing,** 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.

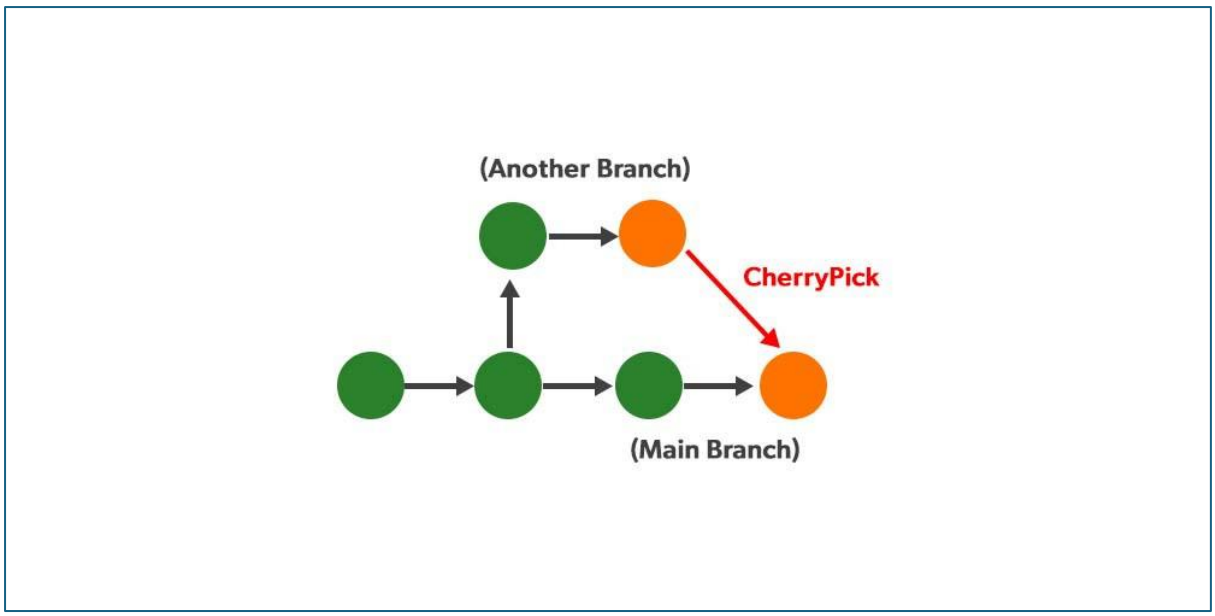
When to use cherry-pick?

Suppose a developer fails to recognize which branch he is currently on, and by mistake, he commits to another branch instead of committing to the main branch. Now to fix it, he has to first run **git show,** then save the commit, check out the main branch, apply a 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:



Before Cherry Pick



After Cherry Pick

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.*

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** and creating a new project named **sample** and initializing the repo using the **git init** command.

```
MINGW64:/c/Users/GAURANG/git-sample

GAURANG@EDITH MINGW64 ~
$ mkdir git-sample

GAURANG@EDITH MINGW64 ~
$ cd git-sample

GAURANG@EDITH MINGW64 ~/git-sample
$ git init
Initialized empty Git repository in C:/Users/GAURANG/git-sample/.git/

GAURANG@EDITH MINGW64 ~/git-sample (master)
$
```

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.*

```
GAURANG@EDITH MINGW64 ~/git-sample (master)
$ vi index.txt

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git add .

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git commit -m "Successfully done first commit"
[master (root-commit) 1ce2f1e] Successfully done first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.txt
```

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

```
GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git commit -m "Successfully done first commit"
[master (root-commit) 1ce2f1e] Successfully done first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.txt

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git log --oneline
1ce2f1e (HEAD -> master) Successfully done first commit
```

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.*

```
GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git branch 1

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git branch 2

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git branch
  1
  2
* master

GAURANG@EDITH MINGW64 ~/git-sample (master)
$ git checkout '2'
Switched to branch '2'
```

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.

```
GAURANG@EDITH MINGW64 ~/git-sample (2)
$ vi feature.txt

GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git add .

GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git commit -m "Feature added in 2nd branch"
[2 18957d8] Feature added in 2nd branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature.txt
```

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

```
GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git log --oneline
18957d8 (HEAD -> 2) Feature added in 2nd branch
1ce2f1e (master, 1) Successfully done first commit
```

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.

```
GAURANG@EDITH MINGW64 ~/git-sample (2)
$ vi fix.txt

GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git add .

GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git commit -m "fixed the issue"
[2 a0d8a15] fixed the issue
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 fix.txt
```

Checking our final commits:

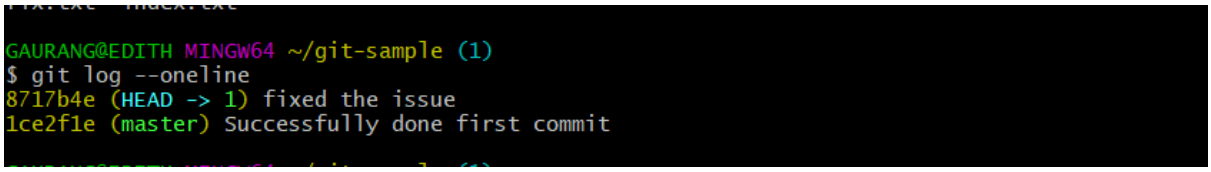
```
GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git log --oneline
a0d8a15 (HEAD -> 2) fixed the issue
18957d8 Feature added in 2nd branch
1ce2f1e (master, 1) Successfully done first commit
```

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.

```
GAURANG@EDITH MINGW64 ~/git-sample (2)
$ git checkout 1
Switched to branch '1'

GAURANG@EDITH MINGW64 ~/git-sample (1)
$ ls
index.txt

GAURANG@EDITH MINGW64 ~/git-sample (1)
$ git cherry-pick a0d8a15
[1 8717b4e] fixed the issue
 Date: Sat Feb 26 02:55:20 2022 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 fix.txt

GAURANG@EDITH MINGW64 ~/git-sample (1)
$ ls
fix.txt  index.txt
```

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.



Some important Use Cases of Cherry-pick

The following are some common applications of Cherry-Pick:

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.