GIT: From Beginner To Fearless

GIT Cherry Picking Activity:
Get a specific commit into your history from a chain of commits

Brian Gorman, Author/Instructor/Trainer

Introduction

Imagine you are developing away on a feature branch when you discover a simple bug in some existing code. One strategy you might take is to commit your changes, go back to the master, create a new branch, and then checkout the new branch and make the fix, merge, pull down to local, merge into feature branch and then continue. This is a great way to solve the problem, but creates a few merge commits and also is a lot of extra jumping around (albeit very safe!).

Another scenario exists where you might be developing and you get something completed, and then you keep developing, and you then want to merge part of what you had done in order to get it tested, or just to get the code out the door. As above, there are many ways you can go about working with your codebase to do this.

Yet another scenario is that developer A and B accidentally were working on the same feature. Developer B's solution is farther along and ready to move forward, but there is one part of Developer A's branch that really makes sense to merge into the source for Developer B to use (and just throw away the rest of developer A's work – sorry dev A). Once again, there are many ways to accomplish this.

However, in all of these scenarios what we really want is just part of a feature branch's commit chain to be merged into master. This is where cherry-picking can be a great tool to use. Be advised that cherry-picking is one of the more complicated activities in GIT, but with more complication generally comes more power, right?

Let's dive in and find out!

Step 1: Make sure we have a working repository that is up to date

a) Start with any repo, make sure you have the latest in master, and create a feature branch.

First clone the repo if it doesn't exist:

[git clone <link> <folder>]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder
$ git clone https://github.com/majorguidancesolutions/SimpleActivityRepo.git Che
rryPickingActivity
Cloning into 'CherryPickingActivity'...
remote: Counting objects: 51, done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 51 (delta 28), reused 39 (delta 16), pack-reused 0
Unpacking objects: 100% (51/51), done.
```

If you didn't clone, make sure master is up to date

[**git checkout master**]

[**git fetch origin**]

[**git pull origin master**]

[git checkout –b feature-branch]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git fetch origin

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git pull origin master
From https://github.com/majorguidancesolutions/SimpleActivityRepo
 * branch            master     -> FETCH_HEAD
Already up-to-date.

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-b
ranch)
$
```

b) Make 3 commits that are easily identified. Understand that if we take commit 2, commit 1 will be included by default and only commit 3 would be left out, so make your changes accordingly, and clear.

[code info.txt] //leave it open

[git commit -am "CherryPickingActivity – commit #1"]

[make changes in info.txt]

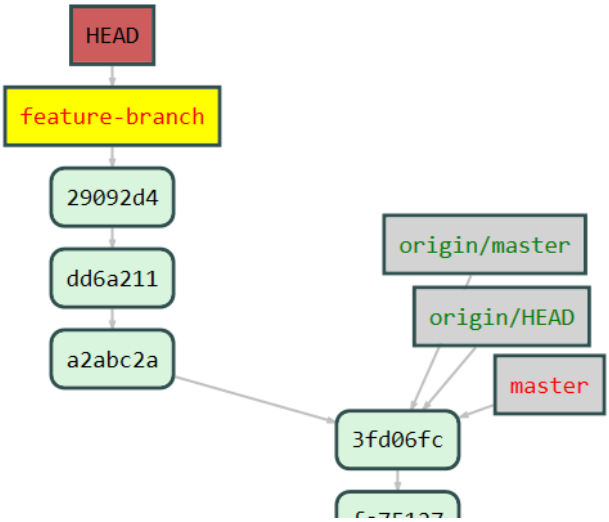
[git commit -am "CherryPickingActivity – commit #2"]

[make changes in info.txt]

[git commit -am "CherryPickingActivity – commit #3"]

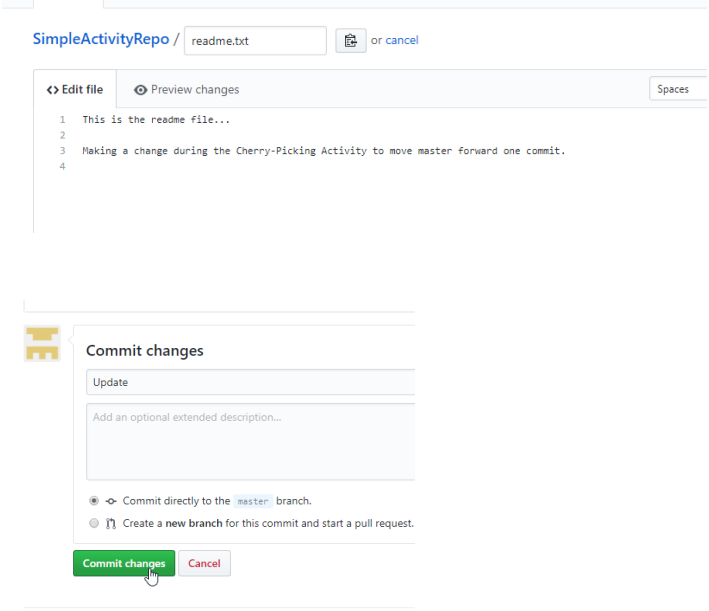
Notes

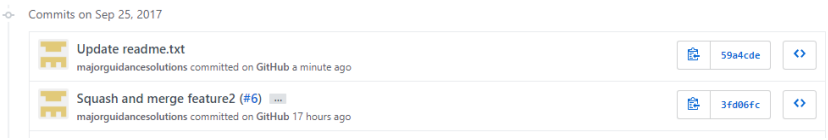
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (featur
ranch)
$ git log --oneline
29092d4 (HEAD -> feature-branch) CherryPickingActivity - commit #3
dd6a211 CherryPickingActivity - commit #2
a2abc2a CherryPickingActivity - commit #1
3fd06fc (origin/master, origin/HEAD, master) Squash and merge feature2 (#6)
```



Step 2: Make one or more commits on the master at GitHub

a) Go out to GitHub and create a new commit to move the master forward one commit. [Don't create a conflict unless you want to have practice resolving conflicts] You can make the commit directly on master to save some time.





Step 3: Get master up to date, create a branch to pick to

a) Get the latest changes into master

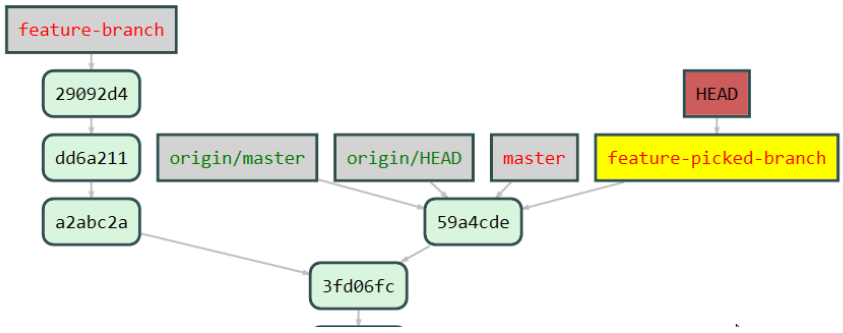
[git checkout master]
[git fetch origin]
[git pull origin master]

```
ranch)
$ git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git fetch origin
git
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git pull origin master
From https://github.com/majorguidancesolutions/SimpleActivityRepo
 * branch            master     -> FETCH_HEAD
Updating 3fd06fc..59a4cde
Fast-forward
 readme.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

b) Checkout a new branch for cherry-picking into:

[git checkout <feature-branch>]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git checkout -b feature-picked-branch
Switched to a new branch 'feature-picked-branch'

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch)
```



Step 4: Perform the cherry-pick operation

Before we begin, I'd like to point out that if we want all of the commits we could easily do a simple rebase with [rebase master], which would bring all three commits over. If the final commit is easy to undo, it might be easier to rebase, and then revert the last commit. However, this activity is only a simple change, whereas in real life the changes probably wouldn't be that easy to mess with, and the cherry-pick is probably the safer and more trustworthy approach.

a) Begin the cherry-pick with a call to the commit (and by default its parents if any) that we want to pick. Here we will take the second commit, id dd6a211 in our history

[git cherry-pick dd6a211]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch)
$ git cherry-pick dd6a21
error: could not apply dd6a211... CherryPickingActivity - commit #2
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch|CHERRY-PICKING)
$
```

As expected, we have to resolve some merge conflicts. Here is where we'll need to tell GIT what to keep on the cherry-pick. Note that the terminal tells us that we are 'CHERRY-PICKING'. If for some reason we want out, we could simply abort. Do that now:

[git cherry-pick --abort]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch|CHERRY-PICKING)
$ git cherry-pick --abort

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch)
$
```

Nothing is done, and we go back to where we were. Of course we actually want to pick, so hit the up arrow a couple of times and re-run the command to cherry-pick dd6a21. Also, before doing that, if you want to validate, you could run a [git log --oneline] to see no extra commits have been added.

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity
icked-branch|CHERRY-PICKING)
$ git cherry-pick --abort

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity
icked-branch)
$ git cherry-pick dd6a21
error: could not apply dd6a211... CherryPickingActivity - commit #2
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

b) Resolve the merge conflicts

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/Cher
icked-branch|CHERRY-PICKING)
$ git mergetool
Merging:
info.txt

Normal merge conflict for 'info.txt':
  {local}: modified file
  {remote}: modified file
```

```
Squash and Merge commit #9
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
=======

Cherry Pick merge #1
Cherry Pick merge #2
>>>>>>> dd6a211... CherryPickingActivity - commit #2 (Incoming Change)
```

Here you can see simple commits I had done for the first two commits. The commit #3 is not shown because I'm not picking it [commit 29092d4]. This is exactly what I want. Of course in real life it will be more complex to hit all the changes, but that is ok as well, because we'll know what we should keep and what we should not keep.

I'll hit [Accept Incoming Change].

Then save and exit the mergetool

[git status]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (f
icked-branch|CHERRY-PICKING)
$ git status
On branch feature-picked-branch
You are currently cherry-picking commit dd6a211.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

Changes to be committed:

        modified:   info.txt
```

Good thing I read that! I was about to commit! Instead, I'll run 'git cherry-pick –continue' as the terminal suggests:

[git cherry-pick --continue]

//which just gives me a chance to edit the commit message ☺

```
Selection  View  Go  Debug  Tasks  Help
COMMIT_EDITMSG
1   CherryPickingActivity - commit #2
2
3   # Conflicts:
4   #   info.txt
5   #
6   # It looks like you may be committing a cherry-pick.
7   # If this is not correct, please remove the file
8   #   .git/CHERRY_PICK_HEAD
9   # and try again.
10
11
12  # Please enter the commit message for your changes. Lines starting
13  # with '#' will be ignored, and an empty message aborts the commit.
14  #
15  # Date:      Mon Sep 25 18:03:35 2017 -0500
16  #
17  # On branch feature-picked-branch
18  # You are currently cherry-picking commit dd6a211.
19  #
20  # Changes to be committed:
21  #   modified:   info.txt
22  #
23
```

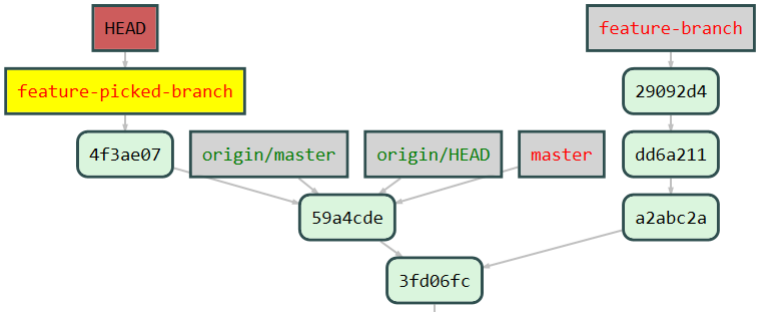
So I'll take this message, save, and close the editor

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity
icked-branch|CHERRY-PICKING)
$ git cherry-pick --continue

[feature-picked-branch 4f3ae07] CherryPickingActivity - commit #2
 Date: Mon Sep 25 18:03:35 2017 -0500
 1 file changed, 3 insertions(+)

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity
icked-branch)
```

And the history:
[git log --oneline]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (featu
icked-branch)
$ git log --oneline
4f3ae07 (HEAD -> feature-picked-branch) CherryPickingActivity - commit #2
59a4cde (origin/master, origin/HEAD, master) Update readme.txt
3fd06fc Squash and merge feature2 (#6)
fa75127 Merge pull request #5 from majorguidancesolutions/SquashAndMergeFeat
```

Sweet, it squashed my two commits into one. That's awesome. But now I need to clear out that other branch, so I need to be SURE that I don't want what's in commit 3 before doing that. I should also make sure I merge my branch to master and get that all squared away so that I have my expected changes before cleaning up.

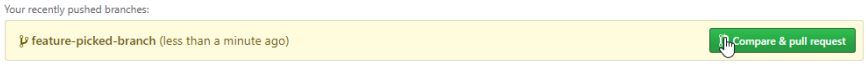


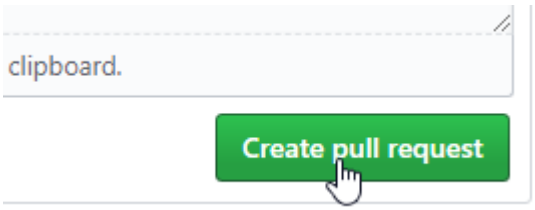
c) Push the feature picked branch to GitHub, do a pull request, merge to master, then update master locally with the history from GitHub

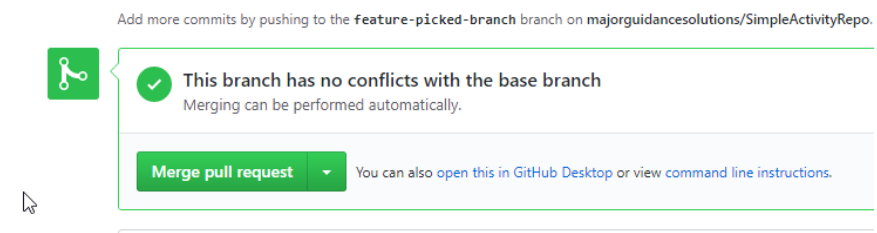
[git push -u origin <your-branch-name>]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch)
$ git push -u origin feature-picked-branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 339 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/majorguidancesolutions/SimpleActivityRepo.git
 * [new branch]      feature-picked-branch -> feature-picked-branch
Branch feature-picked-branch set up to track remote branch feature-picked-branch
 from origin.
```

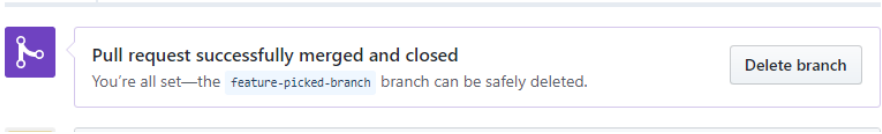
Then at GitHub:







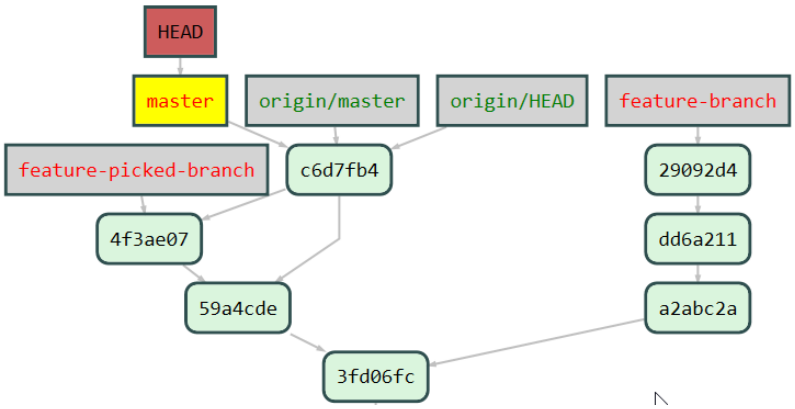
You can either do a regular merge, a squash and merge, or a merge with rebase here, it doesn't matter – as long as we delete our branch after for all but the regular merge commit.
I'm going to do a regular merge pull request as shown.



I'm also going to delete the branch.

Back at local, get master up to date
[git checkout master]
[git fetch origin]
[git pull origin master]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (feature-p
icked-branch)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git fetch origin
From https://github.com/majorguidancesolutions/SimpleActivityRepo
 - [deleted]         (none)     -> origin/feature-picked-branch
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 1 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
   59a4cde..c6d7fb4  master     -> origin/master

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git pull origin master
From https://github.com/majorguidancesolutions/SimpleActivityRepo
 * branch            master     -> FETCH_HEAD
Updating 59a4cde..c6d7fb4
Fast-forward
 info.txt | 3 +++
 1 file changed, 3 insertions(+)
```



d) Make sure changes we want are on master from feature branch

//if not using gitvis, would need to look into the reflog
[git reflog]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git reflog
c6d7fb4 (HEAD -> master, origin/master, origin/HEAD) HEAD@{0}: pull origin maste
r: Fast-forward
59a4cde HEAD@{1}: checkout: moving from feature-picked-branch to master
4f3ae07 (feature-picked-branch) HEAD@{2}: commit (cherry-pick): CherryPickingAct
ivity - commit #2
59a4cde HEAD@{3}: reset: moving to 59a4cdef45c63f7b5c3a0679f6035a793f3f8ac9
59a4cde HEAD@{4}: checkout: moving from master to feature-picked-branch
59a4cde HEAD@{5}: pull origin master: Fast-forward
3fd06fc HEAD@{6}: checkout: moving from feature-branch to master
29092d4 (feature-branch) HEAD@{7}: commit: CherryPickingActivity - commit #3
dd6a211 HEAD@{8}: commit: CherryPickingActivity - commit #2
a2abc2a HEAD@{9}: commit: CherryPickingActivity - commit #1
3fd06fc HEAD@{10}: checkout: moving from master to feature-branch
3fd06fc HEAD@{11}: checkout: moving from master to master
3fd06fc HEAD@{12}: clone: from https://github.com/majorguidancesolutions/SimpleA
ctivityRepo.git
```

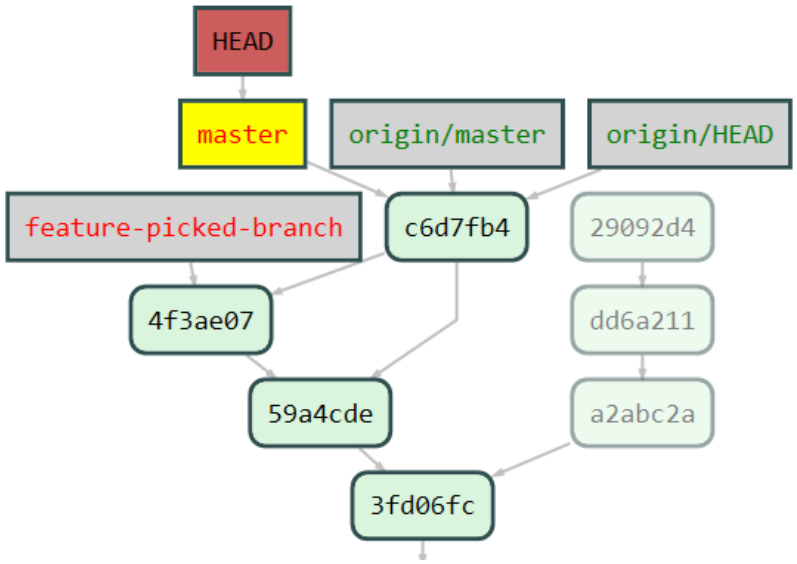
Note the three original commit ids at HEAD@{9}, HEAD@{8}, and HEAD@{7}. Since I'm using GitVis, it's easy to see them on the overall diagram from the previous page [same ids as shown on the reflog].

If the cherry-pick and merge was successful, master, commit c6d7fb4, will have everything from commit a2abc2a and dd6a211, but not 29092d4. Of course if I changed anything during the pick it may have a few differences [or whitespace differences]. That being said, we should be pretty solid for this activity.

Also note-even if we had deleted the branch and had these commit ids, we could still do the comparison:

In fact, let's do it! NO FEAR! [we can get them back if something goes wrong!] [git branch –D <your-branch-name>] //have to use –D to force it! One commit lost forever [right?]!

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity
$ git branch -D feature-branch
Deleted branch feature-branch (was 29092d4).
```



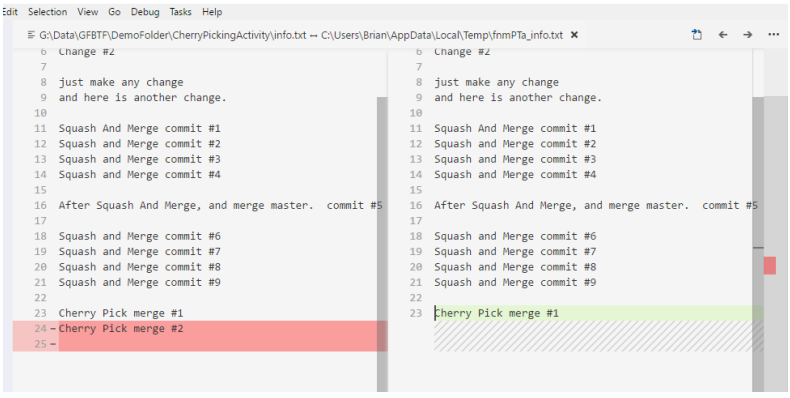
They are still there in cache!

[git reflog]

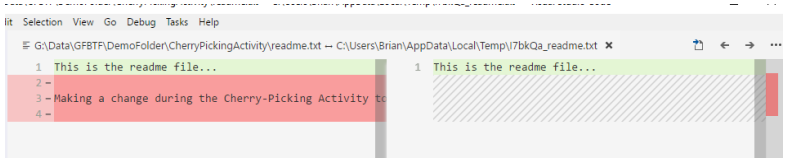
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git reflog
c6d7fb4 (HEAD -> master, origin/master, origin/HEAD) HEAD@{0}: pull origin maste
r: Fast-forward
59a4cde HEAD@{1}: checkout: moving from feature-picked-branch to master
4f3ae07 (feature-picked-branch) HEAD@{2}: commit (cherry-pick): CherryPickingAct
ivity - commit #2
59a4cde HEAD@{3}: reset: moving to 59a4cdef45c63f7b5c3a0679f6035a793f3f8ac9
59a4cde HEAD@{4}: checkout: moving from master to feature-picked-branch
59a4cde HEAD@{5}: pull origin master: Fast-forward
3fd06fc HEAD@{6}: checkout: moving from feature-branch to master
29092d4 HEAD@{7}: commit: CherryPickingActivity - commit #3
dd6a211 HEAD@{8}: commit: CherryPickingActivity - commit #2
a2abc2a HEAD@{9}: commit: CherryPickingActivity - commit #1
3fd06fc HEAD@{10}: checkout: moving from master to feature-branch
3fd06fc HEAD@{11}: checkout: moving from master to master
3fd06fc HEAD@{12}: clone: from https://github.com/majorguidancesolutions/SimpleA
ctivityRepo.git
```

And in reflog too! If I need 29092d4 back, I can just check it out, create a branch and go with it!

[git difftool c6d7fb4 a2abc2a]

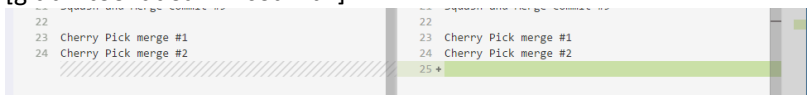


Wait, what? How come there is a difference? Because this was the first of TWO commits, and so it looks like there is a difference from this perspective [so be careful!].

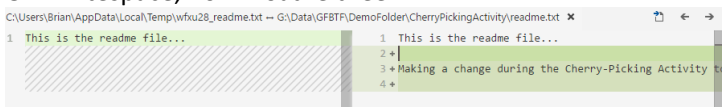


Ah crap – I forgot I made a change at master too. Of course that is different. Let's check the second commit – which should only have the change at master, and let's flip the order. Green is better than red, right?

[git difftool dd6a211 c6d7fb4]

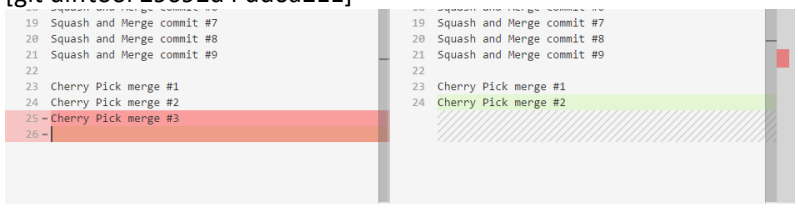


Oh whitespace, how I loathe thee.



Looks good.

Let's validate that commit 3 is not in there..
[git difftool 29092d4 dd6a211]



And it's not. That's great!

e) Clean up the unreachables.

Now that we know the commits that are unreachable don't matter to us anymore, let's use the reflog expire and garbage collector to clean them up.
[git reflog expire --expire-unreachable=now --all]
[git gc --prune=now]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git reflog expire --expire-unreachable=now --all

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git gc --prune=now
Counting objects: 58, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (56/56), done.
Writing objects: 100% (58/58), done.
Total 58 (delta 27), reused 0 (delta 0)
```

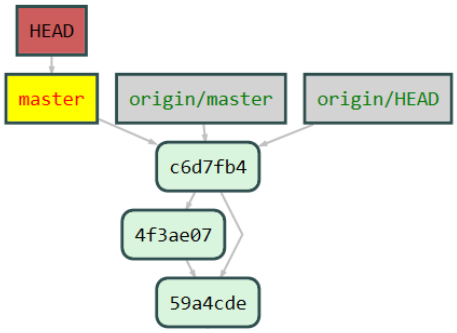
f) Clean up the branch that is behind master now and not at origin anymore, then show the final state of the repo after this activity

[git branch -d feature-picked-branch]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git branch -d feature-picked-branch
Deleted branch feature-picked-branch (was 4f3ae07).
```

[git reflog]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/CherryPickingActivity (master)
$ git reflog
c6d7fb4 (HEAD -> master, origin/master, origin/HEAD) HEAD@{0}: pull origin maste
r: Fast-forward
59a4cde HEAD@{1}: checkout: moving from feature-picked-branch to master
4f3ae07 HEAD@{2}: commit (cherry-pick): CherryPickingActivity - commit #2
59a4cde HEAD@{3}: reset: moving to 59a4cdef45c63f7b5c3a0679f6035a793f3f8ac9
59a4cde HEAD@{4}: checkout: moving from master to feature-picked-branch
59a4cde HEAD@{5}: pull origin master: Fast-forward
3fd06fc HEAD@{6}: checkout: moving from master to feature-branch
3fd06fc HEAD@{7}: checkout: moving from master to master
3fd06fc HEAD@{8}: clone: from https://github.com/majorguidancesolutions/SimpleAc
tivityRepo.git
```



This concludes our cherry-picking activity.

Closing Thoughts

Cherry picking can be scary, so hopefully this activity has removed a lot (if not all) of that fear for you. What we've seen is that we can get the changes from a chain of commits and leave part of the changes out by not including one or more of the commits.

Cherry picking does give us an all-or-nothing operation on the commit, so if you're looking to get just part of a commit, you'd have to use something else, or do the pick and be careful during merge resolution as to what is included.

Because of the nature of the pick, we also got to see a few more things about the reflog and how we can get to 'unreachable' commits even after they don't have a direct reference. We also used the reflog to expire the unreachables and the garbage collector to get rid of those commits.

In the end, this powerful tool allowed us to easily merge the parts of our changes that we wanted while leaving the others behind.

Take a few minutes to make some notes about the various commands we've learned about in this activity, and practice using them.

Notes