Home
PUBLIC
Stack Overflow
Tags
Users
Jobs
TEAMS What's this?
Free 30 Day Trial

How to selectively merge or pick changes from another branch in Git?

Ask Question

Asked 11 years, 1 month ago Active 10 months ago Viewed 714k times

1408

I'm using git on a new project that has two parallel -- but currently experimental -- development branches:

- `master` : import of existing codebase plus a few mods that I'm generally sure of
- `exp1` : experimental branch #1
- `exp2` : experimental branch #2

`exp1` and `exp2` represent two very different architectural approaches. Until I get further along I have no way of knowing which one (if either) will work. As I make progress in one branch I sometimes have edits that would be useful in the other branch and would like to merge just those.

What is the best way to merge selective changes from one development branch to another while leaving behind everything else?

Approaches I've considered:

1. `git merge --no-commit` followed by manual unstaging of a large number of edits that I don't want to make common between the branches.
2. Manual copying of common files into a temp directory followed by `git checkout` to move to the other branch and then more manual copying out of the temp directory into the working tree.
3. A variation on the above. Abandon the `exp` branches for now and use two additional local repositories for experimentation. This makes the manual copying of files much more straightforward.

All three of these approaches seem tedious and error-prone. I'm hoping there is a better approach; something akin to a filter path parameter that would make `git-merge` more selective.

git git-merge git-cherry-pick git-patch

share improve this question

edited Apr 30 '18 at 22:35 vas 733 • 6 • 17

asked Jan 16 '09 at 4:55 David Joyner 18.4k • 4 • 23 • 33

4 If changes in your experimental branches are well organized in separate commits, it's better to think in terms of merging selective *commits* instead of selective files. Most of the answers below assume this is the case. – akaihola Feb 13 '10 at 12:05

2 Wouldn't a combination of `git merge -s ours --no-commit` followed by some `git read-tree` be a good solution for this? See stackoverflow.com/questions/1214906/... – VonC Mar 4 '11 at 7:27

34 A more recent question has a one-line, well-written answer: stackoverflow.com/questions/10784523/... – brahn Aug 15 '13 at 0:36

Checkout this blog for merging specific files only jasonrudolph.com/blog/2009/02/25/... – Ashutosh Chamoli Jan 28 '19 at 15:13

add a comment

25 Answers

active oldest votes

458

You use the cherry-pick command to get individual commits from one branch.

If the change(s) you want are not in individual commits, then use the method shown here to split the commit into individual commits. Roughly speaking, you use `git rebase -i` to get the original commit to edit, then `git reset HEAD^` to selectively revert changes, then `git commit` to commit that bit as a new commit in the history.

There is another nice method here in Red Hat Magazine, where they use `git add --patch` or possibly `git add --interactive` which allows you to add just parts of a hunk, if you want to split different changes to an individual file (search in that page for "split").

Having split the changes, you can now cherry-pick just the ones you want.

share improve this answer

edited May 1 '18 at 1:40 vas 733 • 6 • 17

answered Jan 16 '09 at 6:01 1800 INFORMATION 110k • 26 • 143 • 227

14 From my understanding, this is needlessly more convoluted than the higher-voted answer. – Alexander Bird Feb 3 '12 at 2:24

53 This is technically the correct answer, the correct answer does appear "convoluted". --- The higher voted answer is just a quick and dirty "does the trick" answer, which for most people is all they are about (: – Jacob Mar 9 '12 at 7:46

3 @akaihola: HEAD^ is correct. See man git-rev-parse: A suffix ^ to a revision parameter means the first parent of that commit object. The prefix ^ notation is used to exclude commits reachable from a commit. – Tyler Rick Feb 13 '13 at 19:49

13 I just wanted to share another approach which seems the cleanest and less convoluted of them all: jasonrudolph.com/blog/2009/02/25/... total simplicity and awesome – superuseroi Sep 15 '14 at 0:03

12 Confused by the debate over which approach is 'correct'? Consider the difference between files & commits (see

Blog
Requirements volatility is the core problem of software engineering
The Overflow #11: Efficiently lazy
Featured on Meta
The company's commitment to rebuilding the relationship with you, our community
What is the mission of Meta, as a community?

Related

4675 How do I discard unstaged changes in Git?
6682 How to remove local (untracked) files from the current Git working tree
4674 How to resolve merge conflicts in Git
8749 How do I undo 'git add' before commit?
20625 How do I undo the most recent local commits in Git?
6548 How do I check out a remote Git branch?
16611 How do I delete a Git branch locally and remotely?
3810 Undo a Git merge that hasn't been pushed yet
7518 How do I revert a Git repository to a previous commit?
8202 How do I rename a local Git branch?

Hot Network Questions

Not even one of me does anyone want
What would be the consequence of a traffic light system where the users with green light pay the users with red light?
Purchase a vehicle from a decade of savings in cash for $10,000 and was told form 8300 sent to IRS
How do I collect on a judgement against a debtor who only deals in cash?
Why does pitch increase when you blow harder into a whistle?
Can the direction of electron flow in a voltaic cell be reversed?
Best place on Earth to fake being on a distant planet?
Framing shots before purchasing cameras?
Jigsaw possible?
If plants and trees *want* carbon dioxide, why is car traffic something bad?
Can frequent light bicycle commuting severely damage pants?
Toggle Shift Key
How to make this "friendly war" plausible?
What is the history of the character "耻"?
How does Flixbus get between London to mainland Europe?
What was the point of horse armour?
Recipe Doesn't Work In US
Can Bladesingers combine: Bladesong + Double-Bladed Scimitar + Revenant Blade feat?
Distance Concentration
What is the evidence for 'billions of neutrinos pass

Remarks at bottom). OP wants to merge FILES & does not mention COMMITS. The higher-voted answer is specific to files; the accepted answer uses cherry-pick, which is specific to commits. Cherry-pick may be key for merging commits selectively, but it can be very painful for moving files from one branch to another. Though commits are the heart of git's strength, don't forget files still have a role! – Kay V Aug 26 '16 at 5:16

show 7 more comments

955

I had the exact same problem as mentioned by you above. But I found this clearer in explaining the answer.

Summary:

- Checkout the path(s) from the branch you want to merge,

```
$ git checkout source_branch -- <paths>...
```

  Hint: It also works without `--` like seen in the linked post.

- or to selectively merge hunks

```
$ git checkout -p source_branch -- <paths>...
```

  Alternatively, use reset and then add with the option `-p`,

```
$ git reset <paths>...
$ git add -p <paths>...
```

- Finally commit

```
$ git commit -m "'Merge' these changes"
```

share improve this answer edited Dec 13 '18 at 14:27 community wiki 11 revs, 9 users 33% Chris Steinbach

9 Bart J's linked article is the best approach. Clear, simple, one command. It's the one I'm about to use. – Pistos Oct 7 '09 at 2:07

253 This isn't a real merge. You're picking changes by file instead of by commit, and you'll lose any existing commit information (author, message). Granted, this is good if you want to merge all changes in some files and it's ok that you must re-do all commits. But if files contain both changes to merge and others to discard, one of the methods provided in other answers will serve you better. – akaihola Feb 13 '10 at 12:00

10 @mykhal and others: this stages the files in the index automatically, so you if you checked out `foo.c` do `git reset HEAD foo.c` to unstage that file and you can then diff it. I found this out after trying it and coming back here to look for an answer to this – michiakig Feb 15 '11 at 4:09

12 to see the changes you could also use: `git diff --cached` – OderWat Jun 21 '11 at 6:16

9 According to this answer `git checkout -p <revision> -- <path>` will be the same as issuing the first three commands you described – 7hi4g0 Jan 22 '15 at 22:32

show 16 more comments

323

To selectively merge files from one branch into another branch, run

```
git merge --no-ff --no-commit branchX
```

where `branchX` is the branch you want to merge from into the current branch.

The `--no-commit` option will stage the files that have been merged by Git without actually committing them. This will give you the opportunity to modify the merged files however you want to and then commit them yourself.

Depending on how you want to merge files, there are four cases:

1) You want a true merge.

In this case, you accept the merged files the way Git merged them automatically and then commit them.

2) There are some files you don't want to merge.

For example, you want to retain the version in the current branch and ignore the version in the branch you are merging from.

To select the version in the current branch, run:

```
git checkout HEAD file1
```

This will retrieve the version of `file1` in the current branch and overwrite the `file1` automerged by Git.

3) If you want the version in branchX (and not a true merge).

Run:

```
git checkout branchX file1
```

```
git checkout branchX file1
```

This will retrieve the version of `file1` in `branchX` and overwrite `file1` auto-merged by Git.

4) The last case is if you want to select only specific merges in `file1`.

In this case, you can edit the modified `file1` directly, update it to whatever you'd want the version of `file1` to become, and then commit.

If Git cannot merge a file automatically, it will report the file as "*unmerged*" and produce a copy where you will need to resolve the conflicts manually.

To explain further with an example, let's say you want to merge `branchX` into the current branch:

```
git merge --no-ff --no-commit branchX
```

You then run the `git status` command to view the status of modified files.

For example:

```
git status

# On branch master
# Changes to be committed:
#
#       modified:   file1
#       modified:   file2
#       modified:   file3
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      file4
#
```

Where `file1`, `file2`, and `file3` are the files git have successfully auto-merged.

What this means is that changes in the `master` and `branchX` for all those three files have been combined together without any conflicts.

You can inspect how the merge was done by running the `git diff --cached`;

```
git diff --cached file1
git diff --cached file2
git diff --cached file3
```

If you find some merge undesirable then you can

1. edit the file directly
2. save
3. `git commit`

If you don't want to merge `file1` and want to retain the version in the current branch

Run

```
git checkout HEAD file1
```

If you don't want to merge `file2` and only want the version in `branchX`

Run

```
git checkout branchX file2
```

If you want `file3` to be merged automatically, don't do anything.

Git has already merged it at this point.

`file4` above is a failed merge by Git. This means there are changes in both branches that occur on the same line. This is where you will need to resolve the conflicts manually. You can discard the merged done by editing the file directly or running the checkout command for the version in the branch you want `file4` to become.

Finally, don't forget to `git commit`.

share improve this answer

edited Jul 28 '17 at 9:52

JamesThomasMoon1979
2,429 ● 1 ● 22 ● 34

answered Sep 3 '11 at 8:48

alvinabad
3,467 ● 1 ● 14 ● 6

10 Careful though: If the `git merge --no-commit branchX` is just a fast-forward, the pointer will be updated,

and the --no-commit is thus silently ignored – cti Mar 12 '12 at 16:14

15 @cfi What about adding `--no-ff` to prevent that behavior? – Eduardo Costa Aug 16 '13 at 13:31

5 I definitely suggest updating this answer with Eduardo's "--no-ff" option. I read through the whole thing (which was otherwise great) only to have my merge get fast-forwarded. – Funktr0n May 2 '14 at 15:44

7 This solution gives the best results and flexibility. – Thiago Macedo Jul 24 '14 at 5:13

20 Unlike the answer with the most votes, this solution preserved my merge history, which is important to me as I weave partial commits back and forth across branches. I didn't try all of the other suggested solutions, so perhaps some of them do this as well. – ws_e_c421 Oct 19 '14 at 12:54

show 5 more comments

105

I don't like the above approaches. Using cherry-pick is great for picking a single change, but it is a pain if you want to bring in all the changes except for some bad ones. Here is my approach.

There is no `--interactive` argument you can pass to git merge.

Here is the alternative:

You have some changes in branch 'feature' and you want to bring some but not all of them over to 'master' in a not sloppy way (i.e. you don't want to cherry pick and commit each one)

```
git checkout feature
git checkout -b temp
git rebase -i master

# Above will drop you in an editor and pick the changes you want ala:
pick 7266df7 First change
pick 1b3f7df Another change
pick 5bbf56f Last change

# Rebase b44c147..5bbf56f onto b44c147
#
# Commands:
# pick = use commit
# edit = use commit, but stop for amending
# squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#

git checkout master
git pull . temp
git branch -d temp
```

So just wrap that in a shell script, change master into $to and change feature into $from and you are good to go:

```
#!/bin/bash
# git-interactive-merge
from=$1
to=$2
git checkout $from
git checkout -b ${from}_tmp
git rebase -i $to
# Above will drop you in an editor and pick the changes you want
git checkout $to
git pull . ${from}_tmp
git branch -d ${from}_tmp
```

share improve this answer

edited Feb 22 '19 at 22:54 — jthill — 34.6k ● 3 ● 51 ● 94

answered May 21 '09 at 3:00 — nosatalian — 5,471 ● 4 ● 16 ● 15

I fixed up the formatting - this is a pretty nice method, if you want to do a selection of commits – 1800 INFORMATION Sep 4 '09 at 10:57

I'm using this technique now and it seems to have worked really well. – dylanfm Feb 10 '10 at 11:24

4 You might want to change `git rebase -i $to` to `git rebase -i $to || $SHELL`, so that the user can call `git --skip` etc, as necessary if the rebase fails. Also worth chaining the lines together with `&&` instead of newlines. – sircolinton Feb 24 '11 at 18:10

2 Unfortunately, it appears the the link in the answer is dead. – ThomasW Apr 6 '12 at 2:08

Not only is the link dead, but it has a WOT poor reputation warning. Therefore I removed it. – Jean-François Corbett Jan 10 '13 at 7:49

add a comment

91

There is another way do go:

```
git checkout -p
```

It is a mix between `git checkout` and `git add -p` and might quite be exactly what you are looking for:

```
    -p, --patch
        Interactively select hunks in the difference between the <tree-ish>
        (or the index, if unspecified) and the working tree. The chosen
        hunks are then applied in reverse to the working tree (and if a
```

```
<tree-ish> was specified, the index).

This means that you can use git checkout -p to selectively discard
edits from your current working tree. See the “Interactive Mode”
section of git-add(1) to learn how to operate the --patch mode.
```

share improve this answer

answered Aug 28 '12 at 18:47

Chronial
49.1k ● 11 ● 67 ● 76

10 This is by far the easiest, simplest method, as long as you only have a manageable number of changes to merge in. I hope more people will notice this answer and upvote it. Example: git checkout --patch exp1 file_to_merge – Tyler Rick Feb 13 '13 at 19:56

1 Similar answer posted on this question: stackoverflow.com/a/11593308/47185 – Tyler Rick Feb 13 '13 at 20:00

Oh, I didn't know checkout had a patch! I did checkout/reset/add -p instead. – Daniel C. Sobral Feb 26 '13 at 14:04

2 Truly the most simplest method. git checkout -p featurebranch filename. And the best thing is when the command is run, it gives you a y/n/e/?/...etc. option to decide how to merge the file. I tried with e and I could even edit the patch before applying ... How cool is it. A true one liner for merging selective files from other branches. – infoclogged Mar 2 '16 at 15:33

add a comment

53

While some of these answers are pretty good, I feel like none actually answered OP's original constraint: selecting particular files from particular branches. This solution does that, but may be tedious if there are many files.

Lets say you have the `master`, `exp1`, and `exp2` branches. You want to merge one file from each of the experimental branches into master. I would do something like this:

```
git checkout master
git checkout exp1 path/to/file_a
git checkout exp2 path/to/file_b

# save these files as a stash
git stash
# merge stash with master
git merge stash
```

This will give you in-file diffs for each of the files you want. Nothing more. Nothing less. It's useful you have radically different file changes between versions--in my case, changing an app from Rails 2 to Rails 3.

EDIT: this will merge files, but does a smart merge. I wasn't able to figure out how to use this method to get in-file diff information (maybe it still will for extreme differences. Annoying small things like whitespace get merged back in unless you use the `-s recursive -X ignore-all-space` option)

share improve this answer

edited Aug 25 '11 at 2:31

answered Aug 25 '11 at 1:31

Eric Hu
17k ● 8 ● 46 ● 65

4 Also note: you can do multiple files from a given branch all inline, eg `git checkout exp1 path/to/file_a path/to/file_x` – EMiller Oct 4 '11 at 22:37

2 This is beautiful. I did `git checkout feature <path>/*` to get groups of files. – isherwood Jan 30 '14 at 18:43

This works ok, but added two extra commit objects. Not a huge deal but a bit messy – MightyPork Sep 20 '15 at 13:42

@MightyPork you're right. Unfortunately, since I wrote this so long ago, I'm no longer sure why the "git stash" and "git merge stash" steps are in there instead of a "git commit". – Eric Hu Sep 22 '15 at 20:47

2 Oh that's clear, I think. This way it merges the one file, not necessarily overwriting previous changes on the target branch. – MightyPork Sep 22 '15 at 22:23

show **2** more comments

47

1800 INFORMATION's answer is completely correct. As a git noob, though, "use git cherry-pick" wasn't enough for me to figure this out without a bit more digging on the internet so I thought I'd post a more detailed guide in case anyone else is in a similar boat.

My use case was wanting to selectively pull changes from someone else's github branch into my own. If you already have a local branch with the changes you only need to do steps 2 and 5-7.

1. Create (if not created) a local branch with the changes you want to bring in.

   `$ git branch mybranch <base branch>`

2. Switch into it.

   `$ git checkout mybranch`

3. Pull down the changes you want from the other person's account. If you haven't already you'll want to add them as a remote.

   `$ git remote add repos-w-changes <git url>`

4. Pull down everything from their branch.

   `$ git pull repos-w-changes branch-i-want`

5. View the commit logs to see which changes you want:

```
$ git log
```

6. Switch back to the branch you want to pull the changes into.

```
$ git checkout originalbranch
```

7. Cherry pick your commits, one by one, with the hashes.

```
$ git cherry-pick -x hash-of-commit
```

Hat tip: http://www.sourcemage.org/Git_Guide

share improve this answer

edited Sep 21 '12 at 8:30
Paweł Prażak
2,649 • 1 • 22 • 37

answered May 7 '09 at 11:38
Cory
15.6k • 15 • 78 • 75

3 Tip: first use the `git cherry` command (see manual first) to identify commits you haven't yet merged. – akaihola Feb 13 '10 at 12:09

This works .. 1. created a new branch 2.created some files/ made some changes 3. commit 4. checkout the master branch 5. Run git cherry-pick -x hash-of-commit and resolve merge conflicts are you are good to go. – RamPrasadBismil May 12 '16 at 18:50

Your link is not working anymore. Can you update it please? – creep3007 Apr 16 '18 at 7:00

add a comment

42

Here is how you can replace `Myclass.java` file in `master` branch with `Myclass.java` in `feature1` branch. It will work even if `Myclass.java` doesn't exist on `master`.

```
git checkout master
git checkout feature1 Myclass.java
```

Note this will overwrite - not merge - and ignore local changes in the master branch rather.

share improve this answer

edited Oct 19 '15 at 2:24
mahemoff
33.4k • 25 • 117 • 186

answered Jul 29 '13 at 8:37
maestr0
3,892 • 3 • 22 • 26

6 This won't merge. It'll just overwrite the changes on master with the changes from the feature1 branch. – Skunkwaffle Sep 6 '13 at 20:59

3 Perfectly, I was looking for this kind of merge where `theirs` overwrite `ours` => +1 Cheers 😉 – olibre Sep 25 '13 at 9:19

1 Some times all you want to do is replace the whole file, so this is what I wanted, but you need to make sure you want to lose all the changes you made to this file. – MagicLAMP Sep 10 '14 at 1:30

1 Cleanest solution, given that the OP specifically wanted to replace the entire file with the equivalent on another branch: `2. Manual copying of common files into a temp directory followed by ...copying out of the temp directory into the working tree.` – Brent Faust Sep 25 '14 at 22:21

add a comment

29

The simple way, to actually *merge* specific files from two branches, not just replace specific files with ones from another branch.

Step one: Diff the branches

```
git diff branch_b > my_patch_file.patch
```

Creates a patch file of the difference between the current branch and branch_b

Step two: Apply the patch on files matching a pattern

```
git apply -p1 --include=pattern/matching/the/path/to/file/or/folder my_patch_file.patch
```

useful notes on the options

You can use `*` as a wildcard in the include pattern.

Slashes don't need to be escaped.

Also, you could use --exclude instead and apply it to everything except the files matching the pattern, or reverse the patch with -R

The -p1 option is a holdover from the *unix patch command and the fact that the patch file's contents prepend each file name with `a/` or `b/` (or more depending on how the patch file was generated) which you need to strip so that it can figure out the real file to the path to the file the patch needs to be applied to.

Check out the man page for git-apply for more options.

Step three: there is no step three

Obviously you'd want to commit your changes, but who's to say you don't have some other related tweaks you want to do before making your commit.

share improve this answer

edited Jan 20 '16 at 3:14

answered Feb 27 '12 at 22:42
masukomi
7,330 • 8 • 34 • 42

1 This was very useful where current_branch had lots of "additional" changes which needed to be preserved. Got the diff of only changes brought in by branch_b as: git diff HEAD...branch_b (yes--three periods does the magic trick). – Saad Malik Aug 16 '15 at 20:08

@masukomi, On step 2, shouldn't you add the patch-file created in step 1 as an argument? – Spiralis Jan 20 '16 at 0:23

For me, all changes are rejected. Any idea why? – LinusGeffarth Jan 31 at 23:49

initial thought @LinusGeffarth is that maybe you got the branches backwards when making the patch? will followup outside of SO to see if we can figure it out. – masukomi Feb 5 at 17:12

add a comment

23

Here's how you can get history to follow just a couple files from another branch with a minimum of fuss, even if a more "simple" merge would have brought over a lot more changes that you don't want.

First, you'll take the unusual step of declaring in advance that what you're about to commit is a merge, without git doing anything at all to the files in your working directory:

```
git merge --no-ff --no-commit -s ours branchname1
```

. . . where "branchname" is whatever you claim to be merging from. If you were to commit right away, it would make no changes but it would still show ancestry from the other branch. You can add more branches/tags/etc. to the command line if you need to, as well. At this point though, there are no changes to commit, so get the files from the other revisions, next.

```
git checkout branchname1 -- file1 file2 etc
```

If you were merging from more than one other branch, repeat as needed.

```
git checkout branchname2 -- file3 file4 etc
```

Now the files from the other branch are in the index, ready to be committed, with history.

```
git commit
```

and you'll have a lot of explaining to do in that commit message.

Please note though, in case it wasn't clear, that this is messed up thing to do. It is not in the spirit of what a "branch" is for, and cherry-pick is a more honest way to do what you'd be doing, here. If you wanted to do another "merge" for other files on the same branch that you didn't bring over last time, it will stop you with an "already up to date" message. It's a symptom of not branching when we should have, in the "from" branch should be more than one different branch.

share improve this answer

edited Oct 17 '12 at 17:49

community wiki
2 revs
jejese

3 Your first command (`git merge --no-ff --no-commit -s outs branchname1`) is exactly what I was looking for! Thanks! – RobM Nov 23 '12 at 18:39

1 With multiple branches, required history, need to merge single file(s) and have to change the content of the file before pushing, this seems to be a decent alternative. For example dev => master, but you want to change the a host definition or similar before pushing to master. – timss May 11 '14 at 12:33

add a comment

15

I know I am a little late but this is my workflow for merging selective files.

```
#make a new branch ( this will be temporary)
git checkout -b newbranch
# grab the changes
git merge --no-commit  featurebranch
# unstage those changes
git reset HEAD
(you can now see the files from the merge are unstaged)
# now you can chose which files are to be merged.
git add -p
# remember to "git add" any new files you wish to keep
git commit
```

share improve this answer

answered Feb 18 '10 at 4:28

Felix
840 ● 8 ● 7

I used a slight variation on this. Instead of merging I cherry-picked. It does the job. The only downside to this approach is you lose the reference to the original commit hash. – Matt Florence Jun 24 '11 at 2:49

add a comment

15

Easiest way is to set your repo to the branch you want to merge with then run,

```
git checkout [branch with file] [path to file you would like to merge]
```

If

If you run

```
git status
```

you will see the file already staged...

Then run

```
git commit -m "Merge changes on '[branch]' to [file]"
```

Simple.

share improve this answer

edited Oct 29 '13 at 21:17
flyx
16.5k ● 5 ● 50 ● 77

answered Oct 29 '13 at 20:54
dsieczko
335 ● 2 ● 11

3 This almost the best answer I found. please see jasonrudolph.com/blog/2009/02/25/... So clear, concise and it just works! – superuseroi Sep 15 '14 at 0:02

1 that will completely replace the files content from the source branch rather than merging – Amare Sep 13 '18 at 0:04

I was just about to answer like this, I thought I invent new things that haven't been answered yet! But this is the **the most simple way** to do it. This should be on top! – Irfandy Jip Dec 11 '19 at 10:10

add a comment

15

I found this post to contain the simplest answer. Merely do:

```
$ #git checkout <branch from which you want files> <file paths>
```

Example:

```
$ #pulling .gitignore file from branchB into current branch
$ git checkout branchB .gitignore
```

See the post for more info.

share improve this answer

answered Nov 4 '13 at 10:51
Stunner
10.3k ● 10 ● 71 ● 131

3 This doesn't really merge, it overwrites the file on current branch. – Igor Ralic Feb 12 '14 at 17:13

1 @igrali That's a useful comment, but compared to the difficulty of the "proper" ways to do this, this is a good workaround. One just has to be very careful. – owensmartin Aug 15 '14 at 23:10

add a comment

12

It's strange that git still does not have such a convenient tool "out of the box". I use it heavily when update some old version branch (which still has a lot of software users) by **just some** bugfixes from the current version branch. In this case it is often needed to quickly get **just some** lines of code from the file in trunk, ignoring a lot of other changes (that are not supposed to go into the old version)... And of course **interactive three-way** merge is needed in this case, `git checkout --patch <branch> <file path>` is not usable for this selective merge purpose.

You can do it easily:

Just add this line to `[alias]` section in your global `.gitconfig` or local `.git/config` file:

```
[alias]
    mergetool-file = "!sh -c 'git show $1:$2 > $2.theirs; git show $(git merge-base $1 $(git
```

It implies you use Beyond Compare. Just change to software of your choice if needed. Or you can change it to three-way auto-merge if you don't need the interactive selective merging:

```
[alias]
    mergetool-file = "!sh -c 'git show $1:$2 > $2.theirs; git show $(git merge-base $1 $(git
```

Then use like this:

```
git mergetool-file <source branch> <file path>
```

This will give you the true selective **tree-way** merge opportunity of just any file in other branch.

share improve this answer

answered Jul 3 '14 at 4:29
JimStar
121 ● 1 ● 4

add a comment

It is not exactly what you were looking for, but it was useful to me:

```
git checkout -p <branch> -- <paths> ...
```

It is a mix of some answers.

share improve this answer — edited Feb 9 '16 at 1:14 — answered Feb 9 '16 at 1:02

Felipe
14.1k ● 9 ● 57 ● 85

This is indeed usefull and could be added to what is for me the best answer, the @alvinabad's answer. When doing: `git checkout HEAD file1` to keep the current version and unmerge the file `file1`, one can use `-p` option to select *part* of the file to be merged. thanks for the trick! – Simon C. Feb 14 '18 at 14:15

add a comment

I would do a

```
git diff commit1..commit2 filepattern | git-apply --index && git commit
```

This way you can limit the range of commits for a filepattern from a branch.

Stolen from: http://www.gelato.unsw.edu.au/archives/git/0701/37964.html

share improve this answer — edited Jan 26 '11 at 10:55 — answered Jan 26 '11 at 10:50

lumpidu
589 ● 5 ● 7

In some situations, this could be very handy. However, if the changes are in a different branch, you can just checkout from the tip of that branch, as in Bart J's answer above. – cdunn2001 Jun 22 '11 at 8:26

Who is Bart Js? – Black Dec 13 '18 at 14:00

add a comment

I had the exact same problem as mentioned by you above. But I found this git blog clearer in explaining the answer.

Command from the above link:

```
#You are in the branch you want to merge to
git checkout <branch_you_want_to_merge_from> <file_paths...>
```

share improve this answer — answered Jun 19 '12 at 10:15

Susheel Javadi
2,486 ● 3 ● 28 ● 31

did you test this? i am sure the files will be replaced from <branch_you_want_to_merge_from> rather than being merged – Amare Sep 13 '18 at 0:02

add a comment

I like the 'git-interactive-merge' answer, above, but there's one easier. Let git do this for you using a rebase combination of interactive and onto:

```
     A---C1---o---C2---o---o feature
    /
----o---o---o---o master
```

So the case is you want C1 and C2 from 'feature' branch (branch point 'A'), but none of the rest for now.

```
# git branch temp feature
# git checkout master
# git rebase -i --onto HEAD A temp
```

Which, as above, drops you in to the interactive editor where you select the 'pick' lines for C1 and C2 (as above). Save and quit, and then it will proceed with the rebase and give you branch 'temp' and also HEAD at master + C1 + C2:

```
     A---C1---o---C2---o---o feature
    /
----o---o---o---o-master--C1---C2 [HEAD, temp]
```

Then you can just update master to HEAD and delete the temp branch and you're good to go:

```
# git branch -f master HEAD
# git branch -d temp
```

share improve this answer — answered Dec 9 '11 at 19:46

Wade
3,172 ● 1 ● 18 ● 25

add a comment

What about `git reset --soft branch` ? I'm surprised that no one have mentioned it yet.

For me, it's the easiest way to selectively pick the changes from another branch, since, this command puts in my working tree, all the diff changes and I can easily pick or revert which one I need. In this way, I have full-control over the committed files.

7

share improve this answer

answered Apr 1 '19 at 22:08

GarryOne
708 ● 7 ● 14

add a comment

I know this question is old and there are many other answers, but I wrote my own script called 'pmerge' to partially merge directories. It's a work in progress and I'm still learning both git and bash scripting.

6

This command uses `git merge --no-commit` and then unapplies changes that don't match the path provided.

Usage: `git pmerge branch path`
Example: `git merge develop src/`

I haven't tested it extensively. The working directory should be free of any uncommitted changes and untracked files.

```
#!/bin/bash

E_BADARGS=65

if [ $# -ne 2 ]
then
    echo "Usage: `basename $0` branch path"
    exit $E_BADARGS
fi

git merge $1 --no-commit
IFS=$'\n'
# list of changes due to merge | replace nulls w newlines | strip lines to just filenames | er
for f in $(git status --porcelain -z -uno | tr '\000' '\n' | sed -e 's/^[[:graph:]][[:space:]]
    [[ $f == $2* ]] && continue
    if git reset $f >/dev/null 2>&1; then
        # reset failed... file was previously unversioned
        echo Deleting $f
        rm $f
    else
        echo Reverting $f
        git checkout -- $f >/dev/null 2>&1
    fi
done
unset IFS
```

share improve this answer

edited May 27 '11 at 2:02

answered May 27 '11 at 1:33

Andy
4,538 ● 2 ● 20 ● 36

add a comment

You can use `read-tree` to read or merge given remote tree into the current index, for example:

4

```
git remote add foo git@example.com/foo.git
git fetch foo
git read-tree --prefix=my-folder/ -u foo/master:trunk/their-folder
```

To perform merge, use `-m` instead.

See also: How do I merge a sub directory in git?

share improve this answer

edited May 23 '17 at 12:02

Community ♦
1 ● 1

answered Feb 25 '16 at 18:25

kenorb
92.2k ● 40 ● 491 ● 528

add a comment

A simple approach for selective merging/committing by file:

4

```
git checkout dstBranch
git merge srcBranch
// make changes, including resolving conflicts to single files
git add singleFile1 singleFile2
git commit -m "message specific to a few files"
git reset --hard # blow away uncommitted changes
```

share improve this answer

answered Nov 22 '17 at 17:25

Dave C
686 ● 7 ● 11

add a comment

If you don't have too many files that have changed, this will leave you with no extra commits.

3

1. Duplicate branch temporarily
`$ git checkout -b temp_branch`

2. Reset to last wanted commit
`$ git reset --hard HEAD~n` , where `n` is the number of commits you need to go back

3. Checkout each file from original branch
`$ git checkout origin/original_branch filename.ext`

Now you can commit and force push (to overwrite remote), if needed.

share improve this answer

answered Oct 16 '18 at 18:15

JBaczuk
8,826 ● 7 ● 34 ● 59

add a comment

2

When only a few files have changed between the current commits of the two branches, I manually merge the changes by going through the different files.

`git difftoll <branch-1>..<branch-2>`

share improve this answer

answered Oct 7 '17 at 19:49

raratiru
4,492 ● 2 ● 38 ● 75

add a comment

2

If you only need to merge a particular directory and leave everything else intact and yet preserve history, you could possibly try this... create a new `target-branch` off of the `master` before you experiment.

The steps below assume you have two branches `target-branch` and `source-branch` , and the directory `dir-to-merge` that you want to merge is in the `source-branch` . Also assume you have other directories like `dir-to-retain` in the target that you don't want to change and retain history. Also, assumes there are merge conflicts in the `dir-to-merge` .

```
git checkout target-branch
git merge --no-ff --no-commit -X theirs source-branch
# the option "-X theirs", will pick theirs when there is a conflict.
# the options "--no--ff --no-commit" prevent a commit after a merge, and give you an opportun

# the above, would have messed up the other directories that you want to retain.
# so you need to reset them for every directory that you want to retain.
git reset HEAD dir-to-retain
# verify everything and commit.
```

share improve this answer

answered Nov 6 '18 at 16:35

code4kix
2,227 ● 2 ● 18 ● 31

add a comment

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.

Not the answer you're looking for? Browse other questions tagged `git` `git-merge` `git-cherry-pick` `git-patch` or ask your own question.