

Git merging by example

UPDATE (28 Apr 2008): So apparently this article made it to the front page of delicious, reddit, and hacker news. It's always the ones I least expect :) Anyway, I've updated the branch names to be less confusing, and I've added an explanation of why I say `git ci` instead of `git commit`; I have the following section in my `.gitconfig` file:

```
[alias]
  st = status
  di = diff
  co = checkout
  ci = commit
  br = branch
  sta = stash
```

This lets you type `git [the left side]` and allows git to interpret it as `git [the right side]`. A very useful feature.

We now bring you the actual article:

I've always liked git's merging algorithm, so I thought I'd show an example where it works exactly like I'd expect.

Let's get started by creating a new git repository and project:

```
$ mkdir git-test
$ cd git-test
$ git init
```

Then we'll create a file to play with, `test.pl`:

```
#!/usr/bin/env perl

print "Hello, world.\n";

__END__
```

And commit that:

```
$ git add test.pl
$ git ci -m 'initial import'
Created initial commit 219c5b3: initial import
1 files changed, 6 insertions(+), 0 deletions(-)
create mode 100644 test.pl
```

At this point, we want to create a branch so we can refactor this mess without ruining the working code.

```
$ git co -b refactor
Switched to a new branch "refactor"
```

Now that we're on the `refactor` branch, let's sequester the `print` statement into a subroutine.

```
#!/usr/bin/env perl

say_hello();

sub say_hello {
    print "Hello, world.\n";
}

__END__
```

And commit:

```
$ git ci -a -m 'factor print into a subroutine';
Created commit 518dec1: factor print into a subroutine
1 files changed, 5 insertions(+), 1 deletions(-)
```

I have an idea for a new UI feature, so I'm going to make a branch here, but not switch to it, since I want to add another feature on this `refactor` branch first.

```
$ git branch new-ui
```

While we're still on `refactor`, let's get rid of that ugly `\n`:

```
#!/usr/bin/env perl
use 5.010;

say_hello();

sub say_hello {
    say "Hello, world.";
}

__END__
```

(Note to the non-perl-users; we added `use 5.010` to pull in the new `say` feature. Unfortunately the program now depends on perl 5.10 instead of perl 5.anything.)

Much cleaner. Commit.

```
$ git ci -a -m 'use say instead of print'
Created commit 518dec1: use say instead of print
1 files changed, 2 insertions(+), 1 deletions(-)
```

OK, with that braindump saved, let's go over to the `new-ui` branch:

```
$ git co new-ui
Switched to branch "new-ui"
```

Now let's add that super cool feature, namely printing "Hello, world" three times instead of just once. You think the boss will go for this change before converting all the servers to Perl 5.10, so you add it on this branch that doesn't require 5.10.

```
#!/usr/bin/env perl

say_hello();
say_hello();
say_hello();

sub say_hello {
    print "Hello, world.\n";
}

__END__
```

Let's commit that.

```
$ git ci -a -m 'say hello three times'
Created commit affad78: say hello three times
1 files changed, 2 insertions(+), 0 deletions(-)
```

The unfortunate part is that management hasn't approved that UI change, and they haven't let you upgrade your production server to 5.10 yet. So you switch back to `master` to work on a task that needs to be done immediately -- adding some documentation.

```
$ git co master
Switched to branch "master"
```

And then edit the file:

```
#!/usr/bin/env perl

print "Hello, world.\n";

__END__

=head1 NAME

test.pl - say hello to the world

=head1 SYNOPSIS

perl test.pl
```

And commit:

```
$ git ci -a -m 'add docs'
Created commit 80d2bba: add docs
1 files changed, 7 insertions(+), 0 deletions(-)
```

With all that productivity, you feel you've earned a sugary cup of coffee, so you head over to Caribou, buy one, and come back. You check your e-mail and find that the UI team loves your "say hello 3 times" change. So, let's merge that into [master](#):

```
$ git pull . new-ui
Auto-merged test.pl
Merge made by recursive.
test.pl | 8 ++++++-
1 files changed, 7 insertions(+), 1 deletions(-)
```

Your file looks like this now:

```
#!/usr/bin/env perl
```

```
say_hello();  
say_hello();  
say_hello();
```

```
sub say_hello {  
    print "Hello, world.\n";  
}
```

```
__END__
```

```
=head1 NAME
```

```
test.pl - say hello to the world
```

```
=head1 SYNOPSIS
```

```
perl test.pl
```

If you do a git log, you'll see that git adds each change you merged in into the history:

```
commit a6d16af596b2d122f4348ded85ca14a74b6adaae  
Merge: 80d2bba... affad78...  
Author: Jonathan Rockway <jon@jrock.us>  
Date: Sun Apr 27 05:35:16 2008 -0500
```

```
Merge branch 'new-ui'
```

```
commit 80d2bba051c525257aa4930b362dbe01d6c280fe  
Author: Jonathan Rockway <jon@jrock.us>  
Date: Sun Apr 27 05:34:03 2008 -0500
```

```
add docs
```

```
commit affad78861d53900199860e60b44fb5c500791f5  
Author: Jonathan Rockway <jon@jrock.us>  
Date: Sun Apr 27 05:28:02 2008 -0500
```

```
say hello three times
```

```
commit 518dec18167d36f6f7813b3affc0e76ad9baf2d9  
Author: Jonathan Rockway <jon@jrock.us>  
Date: Sun Apr 27 05:20:52 2008 -0500
```

```
factor print into a subroutine
```

```
commit 219c5b3a580c0d5d4453e118b7a9c40efb6cd13b  
Author: Jonathan Rockway <jon@jrock.us>  
Date: Sun Apr 27 05:15:39 2008 -0500
```

```
initial import
```

Now you've found out that every copy of Perl 5.8 has been destroyed (blame the sunspots), and you'll have to upgrade to 5.10. Because of that, you can merge in your 5.10 branch. Lucky break! One thing to lose hair over, though, is that the `new-ui` branch that you just merged in has some commits in common with the `refactor` branch you're about to merge in. Will git try to apply that change twice? (No.)

Let's try it:

```
$ git pull . refactor
Auto-merged test.pl
Merge made by recursive.
 test.pl | 3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)
```

As expected, it worked perfectly. Here's the final file:

```
#!/usr/bin/env perl
use 5.010;

say_hello();
say_hello();
say_hello();

sub say_hello {
    say "Hello, world.";
}

__END__

=head1 NAME

test.pl - say hello to the world

=head1 SYNOPSIS

perl test.pl
```

If you look at the log, you'll see that git knows exactly what changes were included by the merge:

```
commit 1d4a7814c29678d8ce69d7e241dcb21c4e5d1b88
Merge: a6d16af... d35db62...
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:44:20 2008 -0500
```

Merge branch 'refactor'

```
commit a6d16af596b2d122f4348ded85ca14a74b6adaae
Merge: 80d2bba... affad78...
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:35:16 2008 -0500
```

Merge branch 'new-ui'

```
commit 80d2bba051c525257aa4930b362dbe01d6c280fe
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:34:03 2008 -0500
```

add docs

```
commit affad78861d53900199860e60b44fb5c500791f5
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:28:02 2008 -0500
```

say hello three times

```
commit d35db62f2a628687db8ab2e5759cae35cffb5ab0
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:23:45 2008 -0500
```

use say instead of print

```
commit 518dec18167d36f6f7813b3affc0e76ad9baf2d9
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:20:52 2008 -0500
```

factor print into a subroutine

```
commit 219c5b3a580c0d5d4453e118b7a9c40efb6cd13b
Author: Jonathan Rockway <jon@jrock.us>
Date: Sun Apr 27 05:15:39 2008 -0500
```

initial import

Even though you've merged these branches into trunk, you can still work on them:

```
$ git co new-ui
$ git rebase master
```

`rebase` will merge `master` into the current branch, but it works by deleting your commits, bringing in `master`'s commits, and then replaying yours on top. This avoids the useless "merge branch 'foo'" commit, but at the cost of being unable to share your changes with another repository (since the commit ids will change; commit ids are dependant on history, and rebase rewrites history).

The usual use case for `rebase` is when you're working on a feature that takes a few days to write and want to bring in upstream every day. Instead of polluting your feature branch with

merges, you can rebase and nobody will ever know that your changes weren't originally made against today's upstream branch. (If there are conflicts, git will allow you to resolve them for each of your patches, and it will remember how you resolved them in case the same thing comes up when you rebase again tomorrow. See `git rerere --help` for details.)

Anyway, after the rebase, our `new-ui` branch is now up to date with respect to master. If you made some changes here, you could merge them into `master` without issue.

Let's do one more example; we'll see how to merge two branches at once. We'll switch back to master, and make a `doc-refactor` branch, since your app really needs more docs.

```
$ git co master
Switched to branch "master"
$ git co -b doc-refactor
Switched to a new branch "doc-refactor"
```

Here, we'll add some POD to the end of the file:

```
__END__

=head1 NAME

test.pl - say hello to the world

=head1 SYNOPSIS

    perl test.pl

=head1 HISTORY

As of version 0.01, C<test.pl> now prints "Hello, world." three
times, for maximum enjoyment.
```

This isn't quite perfect yet, but let's commit it.

```
$ git ci -a -m 'explain the 3x feature'
Created commit 80920ac: explain the 3x feature
1 files changed, 5 insertions(+), 0 deletions(-)
```

Meanwhile, a critical bug was just discovered -- a user paused too long while reading the comma in "Hello, world.", so you need to remove it. We'll branch off master, since this complicated fix may take a few days of testing to fully implement:

```
$ git branch comma-bug-fix master
$ git co comma-bug-fix
# fix it
$ git ci -a -m 'fix the comma bug'
Created commit 053413e: fix the comma bug
1 files changed, 1 insertions(+), 1 deletions(-)
```

That was easier than expected. Since the bugfix was pretty simple and your docs are done, you're ready to share both of those changes with your coworkers by merging them into `master`:

```
$ git co master
$ git pull . comma-bug-fix doc-refactor
Trying simple merge with 053413e7fc2935cfe54de74178f493b7965081b3
Trying simple merge with 80920ac16ad72d8714b4e767a09e1f8ce974fe5a
Simple merge did not work, trying automatic merge.
Auto-merging test.pl
Merge made by octopus.
test.pl | 7 ++++++-
1 files changed, 6 insertions(+), 1 deletions(-)
```

I *love* the "Merge made by octopus" message; I am seriously going to have a t-shirt made that says that. I like it when sea creatures help maintain my code.

Finally, if you don't need those branches anymore, you can blow them away:

```
$ git branch -d refactor new-ui comma-bug-fix doc-refactor
Deleted branch refactor.
Deleted branch new-ui.
Deleted branch comma-bug-fix.
Deleted branch doc-refactor.
```

Git will only delete branches that are completely merged into the current branch, so you don't need to worry about losing data.

In conclusion, git makes non-linear development easy. It's smart, so merges Just Work; worrying about branches is not something you need to do anymore. Additionally, if you are the visual type, try running `gitk --all`. It will draw an interactive graph of all your merge and branch points, so you can see where your branches are at-a-glance. It's awesome.

Last modified: 2008-4-28 (月) at 6:01 pm



Re: Git merging by example

Written by Anonymous Coward (0) on 2008-4-29 (火) at 12:28 pm

test

 [[Link](#) | [XML](#) | [YAML](#) | [Reply...](#)]

 Re: Git merging by example

Written by Anonymous Coward (0) on 2008-4-27 (日) at 11:42 am

Thanks,

That just about explains everything that I knew I was missing.

[[Link](#) | [XML](#) | [YAML](#) | [Reply...](#)]

 Re: Git merging by example

Written by Anonymous Coward (0) on 2008-4-27 (日) at 9:04 pm

You are my new hero, i can't even imagine doing all that with svn. I'm really scared every time that i have to merge/branch stuff, it's so painful and boring...(the git rebase command rocks!, how many times do i ask myself why the hell there isn't something like that?).

Thanks for this post!, i'm going to install git right now :)

[[Link](#) | [XML](#) | [YAML](#) | [Reply...](#)]

 Re: Git merging by example

Written by Anonymous Coward (0) on 2008-4-27 (日) at 7:37 pm

Cheers for that, rebase suddenly clicked ^_^

[[Link](#) | [XML](#) | [YAML](#) | [Reply...](#)]

 Re: Git merging by example

Written by Anonymous Coward (0) on 2008-4-29 (火) at 8:15 pm

Instead of

```
git pull . branch-name
```

you can do

```
git merge branch-name
```

[[Link](#) | [XML](#) | [YAML](#) | [Reply...](#)]