**THE SCOUT PATTERN**

Here's a quick overview of the steps in this recipe:

1. Make sure you're on the right branch and that you have a clean working state.
2. Create a new branch (I often name it *test\_merge*) and switch to it.
3. Do the merge.
4. Switch to your visualizer and predict how its view will change when you refresh it.
5. Refresh your visualizer and see whether your prediction was correct.
6. Are you happy with the result?
   * If YES: Move your real branch forward to where the *test\_merge* branch is.
   * If NO: Delete the *test\_merge* branch.

I call this the Scout pattern: you're unsure what the terrain ahead is like, so you send a scouting party ahead to check it out. If they radio back that everything's okay, you'll move ahead and join them. If not, well, it was just a small scouting party and we'll tell the families that they died with honor...

**THE LONG VERSION**

You're on the ***master*** branch and you want the changes from the ***spiffy\_new\_feature***branch to be incorporated into ***master.*** You're not sure if this will be a good idea, so you want to try out the merge, but be able to abort it if things don't go smoothly.

1. **Make sure you're on the right branch and that you have a clean working state.**

Whatever visualizer you're using, figure out how it shows you where your current branch is. Or, at the command line, type **git status** and you should see something like this:

**# On branch master   
nothing to commit (working directory clean)**

1. **Create a new branch and switch to it.**

Type **git checkout -b test\_merge**. Now, if you type **git status** again, you should see a message that you're on the ***test\_merge*** branch.

1. **Do the merge.**

Type **git merge spiffy\_new\_feature**. If you're lucky, there won't be any merge conflicts you can't resolve.

**If you want to abort the merge** at this point, type **git reset --hard**.

1. **Switch to your visualizer and predict how its view will change when you refresh it.**

For example:

* + 1. After a merge, you should see a new commit.
    2. The new commit should have a message like "Merge branch 'spiffy\_new\_feature' into master".
    3. Your ***test\_merge*** branch label should have moved to this new commit, while the ***master*** and***spiffy\_new\_feature*** branch labels should still be in the same place.

1. **Refresh your visualizer and see whether your prediction was correct.**
2. **Are you happy with the result?**

**If YES:** Move the *master* branch forward to where the *test\_merge* branch is with:**git checkout master   
git merge test\_merge**

**If NO:** Drop the *test\_merge* branch with:**git checkout master   
git branch -D test\_merge**

**THE SAVEPOINT PATTERN**

Here's a quick overview of the steps in this recipe:

1. Make sure you're on the right branch and that you have a clean working state.
2. Create a new branch to use as a savepoint, but don't switch to it.
3. Do the merge.
4. Switch to your visualizer and predict how its view will change when you refresh it.
5. Refresh your visualizer and see whether your prediction was correct.
6. Are you happy with the result?

If YES: Delete the savepoint.

If NO: Reset your branch to the savepoint.

(Unless the last video game you ever played was Super Mario Brothers, it should be obvious why I call this one the Savepoint pattern.)

**THE LONG VERSION**

You're on the ***master*** branch and you want the changes from the ***spiffy\_new\_feature***branch to be incorporated into ***master.*** You're reasonably confident that you'll want to keep the changes, but you want to be able to abort it if, for example, this feature has unintended side effects.

1. **Make sure you're on the right branch and that you have a clean working state.**
2. Whatever visualizer you're using, figure out how it shows you where your current branch is. Or, at the command line, type **git status** and you should see something like this:
3. **# On branch master   
   nothing to commit (working directory clean)**
4. **Create a new branch to use as a savepoint, but don't switch to it.**

Type **git branch savepoint**. Now, if you type **git status** again, you should still see a message that you're on the ***master*** branch.

1. **Do the merge.**

Type **git merge spiffy\_new\_feature**. If you're lucky, there won't be any merge conflicts you can't resolve.

**If you want to abort the merge** at this point, just type **git reset --hard**.

1. **Switch to your visualizer and predict how its view will change when you refresh it.**

For example:

* + 1. After a merge, you should see a new commit.
    2. The new commit should have a message like "Merge branch 'spiffy\_new\_feature' into master".
    3. Your ***master*** branch label should have moved to this new commit, while the***spiffy\_new\_feature*** branch label should still be in the same place.

1. **Refresh your visualizer and see whether your prediction was correct.**
2. Are you happy with the result?

**If YES:** Delete the savepoint.**git branch -d savepoint**

**If NO:** Reset your branch to the savepoint.**git reset --hard savepoint** If you want to clean up, you can now delete the savepoint with **git branch -d savepoint**.

**BLACK BELT MERGING**

Once you're comfortable with [the Savepoint pattern](http://think-like-a-git.net/sections/testing-out-merges/the-savepoint-pattern.html), you might get tired of creating the savepoint branch, only to have to remember to delete it every time. If so, I have good news for you: you don't actually need to use a branch as a savepoint for merges.

Merge commits always wind up with a branch label pointing at them, and one of the branch's parent commits will be the commit that that branch label was just moved from.

It may take you a few tries to parse that sentence. The upshot is that the commit you started on—the one you would've marked with a savepoint branch—**will always be reachable.**

Here's the thing: **Git doesn't care what you call your branches.** Remember, branches are just a 40-byte file on disk that points to the commit's SHA-1 hash, which is what Git really uses for all of its work. The branch is just there for us puny humans to have some convenient, memorable name pointing to part of the graph.

At the end of [the Savepoint pattern](http://think-like-a-git.net/sections/testing-out-merges/the-savepoint-pattern.html), the command you type to reset your branch to the savepoint is **git reset --hard savepoint** (where "savepoint" is a branch name). If you look at the documentation that comes up when you type **git reset -h**, you'll see that the final argument it takes is called **<commit>**. Older versions of the documentation called this **<commit-ish>**, which was a convenient reminder that you could use anything that Git can turn into a SHA-1 hash.

Things Git is happy to accept in a **<commit>** argument include (but are probably not limited to):

* Branch names
* Tags
* Relative references like **HEAD^**, **HEAD^^**, or **HEAD~3**
* partial SHA-1 hashes like 8d434382 (you just have to provide enough initial digits to be unique; Git will fill in the rest for you)
* SHA-1 hashes like 8d434382d9420940be260199a8a058cf468d9037 (these are *very*easy for Git to turn into SHA-1 hashes!)

So, at the end of [the Savepoint pattern](http://think-like-a-git.net/sections/testing-out-merges/the-savepoint-pattern.html), if you wanted to back out of the merge, you could just as easily use **git log** or your visualizer to find the SHA-1 of the commit (let's say it starts with 1234abcd), and type this:

**git reset --hard 1234abcd**

...and **Git would behave exactly the same** as if you'd remembered to create a branch in the first place.

That bit about "if you'd remembered to" reminds me: one of the single most helpful bits of commentary I ever read about Git was Ryan Tomayko's [The Thing About Git](http://tomayko.com/writings/the-thing-about-git). (Yes, that's the same thing I linked to in [LSD and Chainsaws](http://think-like-a-git.net/sections/git-makes-more-sense-when-you-understand-x/example-4-lsd-and-chainsaws.html).)