Chapter 12

Why Branching Is Great

Wouldn’t it be great if your code could exist in a parallel universe, where you could make changes to it in one version without impacting another version? Well, with branching, you can.

SVN, Git, and Mercurial all have ways you can branch your code. In this chapter I’ll show you what a branch is, when you can use branches, and how to create them in SVN, Git, and Mercurial. I’ll finish the chapter by stepping through a worked example of creating a branch, committing code to it, then merging it back into the main line of development.

What is a branch?

In a version control system a branch is code that diverges from the main line of development. If you think of it in terms of a tree, the trunk would represent the main line of development (as it does in SVN), with the branches diverging from the trunk at various points.

I mentioned a parallel universe right at the start of this chapter; when you create a branch you can think of this as causing a split in your codebase. From that point onwards, your code can exist in two parallel futures, completely independent from one another.

When you create a branch, it inherits its history from the main line of development up until the point it was created. Going forward it has its own commit history, and almost behaves like its own repository.

In a moment I’ll show you how to create a branch in SVN, Git, and Mercurial. Before I do that, let’s have a quick look at two of the main uses of branches in software development.
When can I use branches?

Although you aren’t restricted in any way when it comes to using branches, the two main ways they are employed are as release branches and feature branches.

Release branches

*Release branches* are used to “freeze” the code base on a project prior to a release. When a team is working on a project, part of the team will form the release team. They’ll create and work from a release branch, fixing bugs, polishing the product, and working with the testing team, to get it ready for release. Meanwhile, the rest of the team can continue working on the main line of development, perhaps with features planned for the next release.

Once released, a project’s release branches will be used for ongoing maintenance against that particular release until it is no longer supported. For example, if a customer reports a problem against Version 1.0 of your product, you’d check out the release-1.0 branch, fix the problem, and commit your changes to the release-1.0 branch. At the same time, another customer could have reported a problem with Version 1.1. You could then check out the release-1.1 branch, fix the problem, and commit your changes. If it turns out that either of the bugs reported still exists in the main code base, these fixes could then be merged back into your main line of development.

Feature branches

*Feature branches* are used whenever you need to work on something that could lead to your main branch—trunk, master, or default—becoming unstable. Unlike release branches, they are only temporary and only kept for as long as it takes to implement the feature. Features might include testing out a new database abstraction library, trying out a new version of a third-party API, or rewriting a search function using a new or different algorithm. Either way, you wouldn’t want your experimental code ending up in your main code base until you were sure it was stable.

Feature branches, being only temporary in nature, tend to get deleted once they have been merged back into the main code base.

**TOPIC BRANCHES**

Related to feature branches are topic branches. They are essentially the same as feature branches, as they are created to implement a specific feature or bug fix, and form an integral part of the workflow for developers using Git and Mercurial.

The master or default branch would represent the stable form of the code. Whenever a developer starts work on a user story, new feature, or bug fix, they’d create a topic branch named according to whatever it is they’re about to work on, hack away on that branch until done, committing their work to it regularly as they go, then merge their branch back into master or default.