Extracted from:

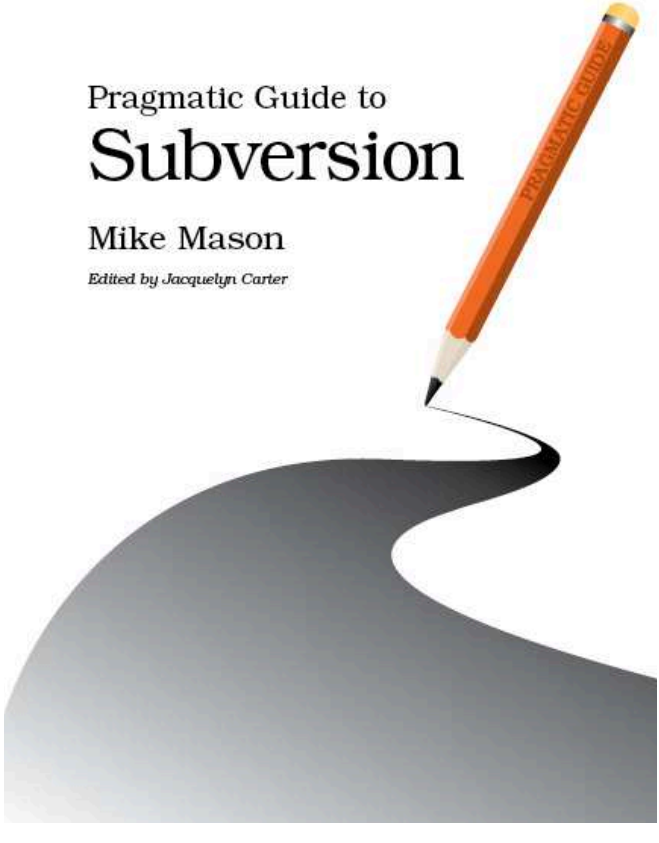
Pragmatic Guide to Subversion

The Pragmatic Programmers

Pragmatic Guide to Subversion

Mike Mason

Edited by Jacquelyn Carter



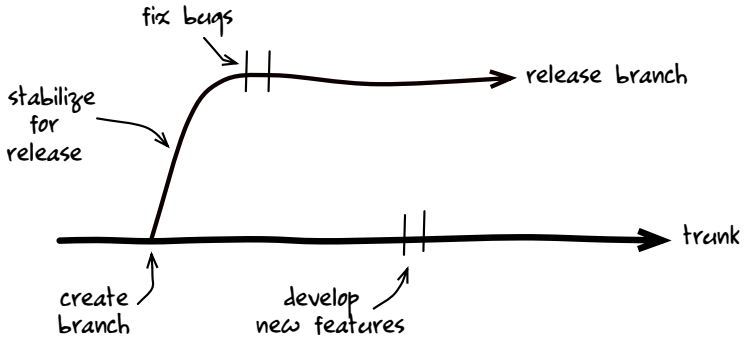
Part V

Branching, Merging, and Tagging

Real-world software projects are rarely straightforward and easy. The team must develop the software, stabilize it ready to be released into production, and support it once it's in production. We've shown how a team can use Subversion to collaborate during development; this chapter will focus on how a team can release and support their software.

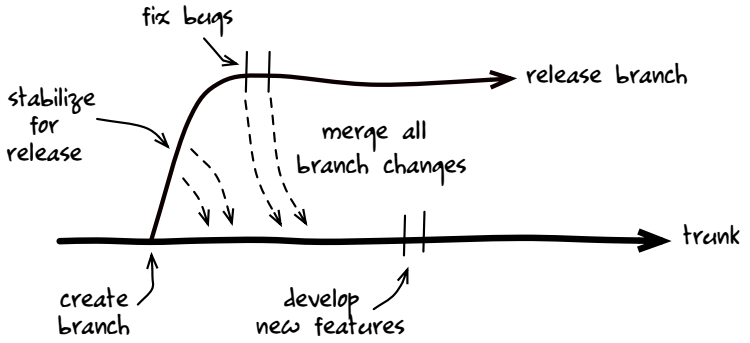
Usually when a team is preparing to release their software, they want to focus on quality. The team might decide to fix bugs and improve performance rather than adding new features. Generally, though, the team will want to continue some forward momentum. Maybe the team will split, with some developers working on stabilizing the code for release and everyone else developing as normal.

These two activities, stabilization and adding new features, generally cannot be done in the same code base. It's very likely that the new features will add instability to the software, which is exactly what we don't want when we're trying to put a release into production. The solution is to *branch* the code. Branching splits off a new line of development where stabilization and bug fixing can be done, while new features can continue to be added on the trunk. The following diagram shows the branch and the trunk visually:

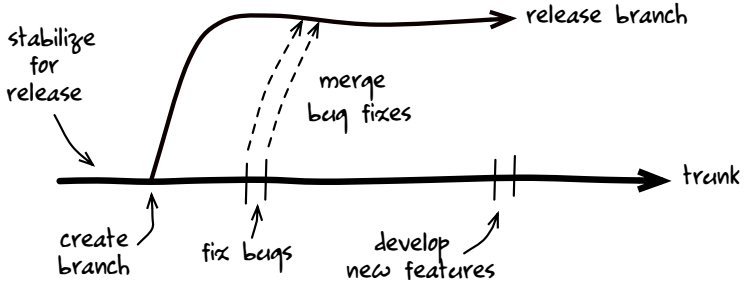


The first step is to create a branch. Branches are identified with a name and are stored in the branches directory within the Subversion repository. A branch starts out as an exact copy of the trunk but can be modified independently. One team can work on the branch, fixing bugs and stabilizing the code. Another team can work on the trunk, adding new features. The two teams will never accidentally surprise each other because they are working on different branches.

When working on a release branch, there are usually some bug fixes or other improvements that we'd like to include in the trunk. Rather than making a fix on the branch and then reimplementing the fix on the trunk, Subversion allows us to *merge* the change from the branch to the trunk. Subversion can do this because a branch is much more than a simple copy of the files. Subversion remembers the origin of the files on the trunk and the branch, and it uses their shared ancestry to make merges easier and more automatic. The dotted lines on the following diagram show changes being merged from the release branch to the trunk:

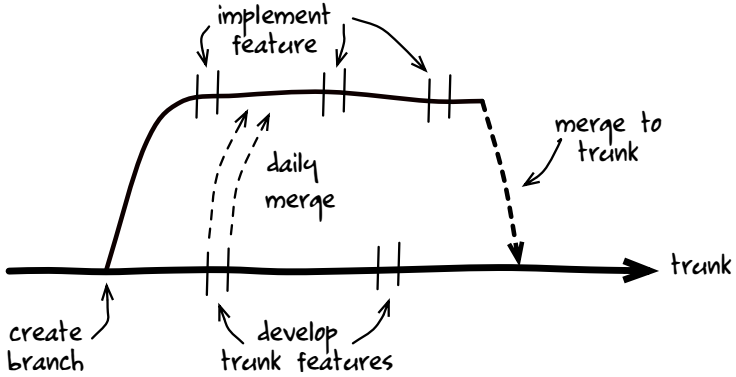


The diagram shows a fairly standard release branch strategy but does require a lot of merging because every change made on the branch needs to be merged back to the trunk. It's usually best to reduce the number of merges in your branching strategy because this reduces effort and the potential for a "forgotten" merge. We can change the branching strategy to reduce merging as follows. Stabilize for release *before* creating the branch, and then fix any bugs on the trunk and merge them to the branch. The branch diagram looks like this:



You might already be familiar with branching and merging and have your own strategy. This is fine; just make sure everyone on the team understands where they need to make fixes and where they should merge. If you're not careful and disciplined, it's possible to "lose" a change. For example, if you fixed a bug on the release branch but forgot to merge the fix to the trunk, the trunk still has that bug. The QA team might see the bug in a later release and call it a *regression* since from their perspective it was fixed once already.

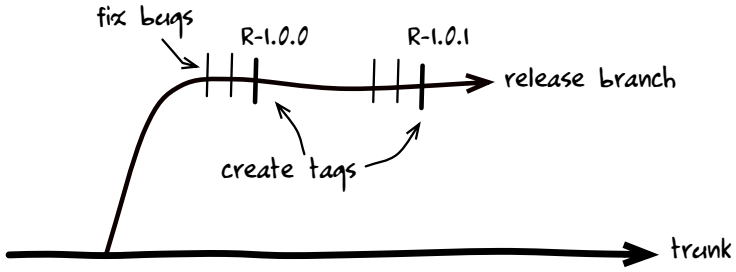
Another strategy worth mentioning is called *feature branching*. A team might use this when a new feature will take a long time or cause some instability in the code base. Instead of developing the feature on the trunk, the team can create a branch specifically for the feature. The rest of development continues on the trunk as normal. The feature branch should be updated with trunk changes frequently—usually daily—to keep the feature branch "close" to the trunk. This merge from the trunk to the feature branch is known as *rebasing*. Once the feature is finished, merge the feature branch back to the trunk.



When a customer calls up with a problem, it's important to know exactly what code they are running so you can diagnose and fix the problem. You should only ship software to a customer from a release branch, but since the code on the branch can change over time, you need a better way to uniquely identify a release. With Subversion, you can *tag* the code that was used to build a release, giving it a number such as 2.0.3. To create a tag, you will copy your release branch into a named directory within the repository tags directory. Usually the tag name is also compiled into the software

as a version number. Once you know what version your customer is running, you can check out the tagged code and get exactly the code that is running in production.

The following diagram shows a release branch with two tags, R-1.0.0 and R-1.0.1:



Covered in this part:

- Task 24, *Creating a Branch*, on page 80 shows how to create a release branch.
- If you have a trunk working copy and want to quickly switch to a branch, follow the instructions in Task 25, *Switching to a Branch*, on page 82.
- Task 26, *Merging Changes from Trunk to Branch*, on page 84 covers the process for merging changes, such as bug fixes, to a branch.
- Repeated merging, usually used to keep the trunk and a branch in sync, is discussed in Task 27, *Using Change Tracking*, on page 86.
- Tagging is explained in detail in Task 28, *Tagging a Release*, on page 88.

Let's start by creating a release branch.

24 Creating a Branch

Subversion branches are copies of the trunk and are stored in the branches directory inside the repository. The branches directory sits alongside the trunk directory, as we saw in Task 5, *Creating an Empty Project*, on page 26. This directory organization is a Subversion convention—nothing forces you to organize your repository in this way, but if you stick to the convention, it makes it easier for people to work with your project.

To create a branch, use the Subversion copy command to copy the trunk to a new location. You should always use repository URLs when creating a branch. You *can* copy the file revisions in a working copy to create a branch, but using a repository URL is much faster. It's also safer because if your working copy contains mixed revisions (not all the files in a working copy have to be at the same revision), Subversion will faithfully copy the mixed revisions to the branch, which usually isn't what you want to do.

Branches can be named using any characters that Subversion allows in a directory name, including spaces and characters with accents (although we suggest sticking to alphanumerics). Use a naming scheme that makes it easy to identify branches. Here we're using "RB" to indicate a release branch, followed by the version number of the branch. You could also organize your branches into different directories, such as releases/1.0.

Once you have created your branch, you can check out a working copy of the code. Make sure that you use a working copy directory name that makes it easy to identify the branch. In our example, we already have an mbench directory for the trunk working copy, so we check out into an mbench-1.0 directory for the 1.0 release branch.

- Create a release branch.

```
prompt> svn copy -m "Create 1.0 release branch" \
             http://svn.mycompany.com/mbench/trunk \
             http://svn.mycompany.com/mbench/branches/RB_1.0
```

- Check out the branch to a new working copy.

```
prompt> cd ~/work
prompt> svn checkout  \
            http://svn.mycompany.com/mbench/branches/RB_1.0  \
            mbench-1.0
```

- Create a branch using Tortoise.

  Using Windows Explorer, right-click the base directory for your working copy, and choose TortoiseSVN > Branch/tag....

  Edit the To URL setting, replacing trunk with branches/RB_1.0, and click OK.

  Enter a log message, and click OK to create the branch.

- Create a branch using Cornerstone.

  Select your repository from the repository source list, and then navigate to the trunk directory for your project.

  Drag the trunk directory to the branches directory while holding down the Option key. Your mouse pointer will indicate you are about to make a copy with a green + icon.

  Give the branch a name, click Copy, and then enter a log message. Click Continue to create the branch.

Related Tasks

- Task 7, *Checking Out a Working Copy*, on page 34
- Task 25, *Switching to a Branch*, on the following page
- Task 26, *Merging Changes from Trunk to Branch*, on page 84

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Pragmatic Guide to Subversion
http://pragprog.com/titles/pg_svn
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/pg_svn.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)