Extracted from:

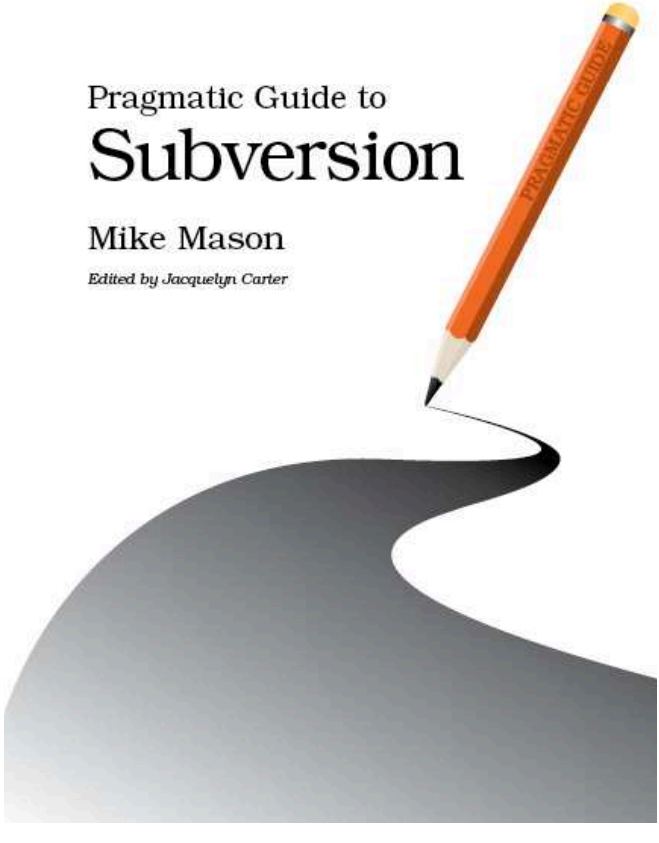
Pragmatic Guide to Subversion

The Pragmatic Programmers

Pragmatic Guide to Subversion

Mike Mason

Edited by Jacquelyn Carter



Part II

Working with Subversion

Now that we have Subversion installed and a repository set up, we can begin doing useful work. While Subversion is designed to enable team collaboration, you can also use it individually. If you are programming or authoring on your own, the diff, history, and commit tools can help you organize your work. You can also use Subversion as a kind of safe "backup" copy for your files.

In this section, you'll see how to get a working copy of the files from Subversion. After you make changes to your files, you commit the changes to the repository. We'll also cover renaming and moving files and directories.

In this section and for the rest of the book, we'll be using an example project called *mBench*. mBench is a simple benchmarking tool for MongoDB, one of a new generation of nonrelational databases (sometimes known as "NoSQL" databases). Mongo is a document-oriented database with very different performance and reliability characteristics than a traditional SQL database. Before using Mongo in production, we'd like to make sure it meets our needs for speed and reliability. mBench will contain Java code to exercise Mongo, documentation detailing things we find out about how to best use Mongo, and other artifacts such as performance testing results.

The complete source code for mBench is available as part of the code download for this book. The code, errata, and forums are available from the Pragmatic Bookshelf at http://pragprog.com/titles/pg_svn. For all the tasks involving mBench, we assume that it's available from a repository at http://svn.mycompany.com/mbench. This URL isn't real, so you should substitute your own repository URL when working with the examples.

Covered in this part:

- To edit files stored in Subversion, you need to first get a copy of the files on your machine. Task 7, *Checking Out a Working Copy*, on page 34 shows you how.
- Task 8, *Seeing What You've Changed*, on page 36 covers how Subversion shows the edits you have made to files. Task 9, *Seeing What You've Changed Using Tortoise*, on page 38 and Task 10, *Seeing What You've Changed Using Cornerstone*, on page 40 show you how to examine changes using GUI tools.

- Once you have made your changes, you need to commit them to the repository. I describe the commit process in Task 11, *Committing Changes*, on page 42.
- Task 12, *Adding Files and Directories*, on page 44 covers how to add new items to your repository, while Task 13, *Removing Files and Directories*, on page 46 shows how to remove items.
- Renaming and moving items is covered in Task 14, *Moving and Renaming Files and Directories*, on page 48.
- If you make a local change that you'd like to undo, follow the instructions in Task 15, *Reverting Working Copy Changes*, on page 50.
- Subversion allows you to ignore temporary files; Task 16, *Ignoring Files*, on page 52 shows you how.

Let's get started by checking out a fresh working copy.

7 Checking Out a Working Copy

You need to *check out* the files from your repository before you can access them on your computer. The files will be stored in a local directory called a *working copy*. Your Subversion client will talk to the server and copy the latest files onto your computer ready to edit. You will be asked for a username and password if your repository is secured.

Subversion projects are usually stored in a trunk directory on the server; this will be the case if you have followed the instructions in this book for creating projects. You need to specify this trunk directory as part of the repository URL from which you are checking out. If you forget, you'll end up checking out everything in the repository—branches, tags, everything—and this could end up being much more than you really wanted. After specifying the project's trunk directory, we tell Subversion that we want to check out into a local directory called mbench. If you don't include this information, Subversion will check out into a folder called trunk, which probably isn't what you want and might be quite confusing if you're working on more than one project!

After Subversion has copied files from the repository to your computer, they'll be saved in a directory known as a working copy. Inside a working copy, Subversion remembers where all the files came from, knows what revision they were when they came from the server, and can detect any changes you make to the files. Subversion uses hidden .svn directories to track all its bookkeeping information. Don't alter, delete, or otherwise mess with these directories because you could corrupt your working copy.

You need to check out from Subversion only the first time you want to work with a particular project. Once you have a working copy, you can always *update* it to get the latest files. We describe the update process in more detail in Task 17, *Updating to the Latest Revision*, on page 58.

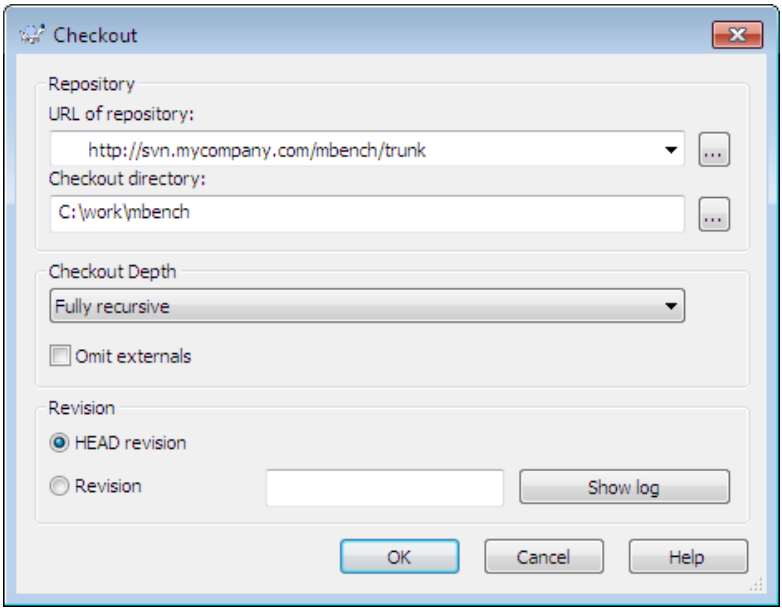
After you check out a working copy using Cornerstone, it will automatically add the working copy as an entry in your working copy sources list. You'll need to use this entry when manipulating your working copy for the rest of the tasks in this book.

- Check out into a local working copy.

```
prompt> cd ~/work
prompt> svn checkout http://svn.mycompany.com/mbench/trunk mbench
```

- Check out using Tortoise.

Using Windows Explorer, navigate to your C:\Work directory. Right-click inside the directory, and choose SVN Checkout.... Then fill in the dialog box as follows:



- Check out using Cornerstone.

Select your repository in the repository source list, and then use the repository browser to navigate to the trunk directory. Ctrl+click “trunk,” and choose Check Out Working Copy.... Specify where you’d like to save the new working copy, and click Check Out.

Related Tasks

- Task 5, *Creating an Empty Project*, on page 26
- Task 36, *Using Third-Party Subversion Hosting*, on page 112

8 Seeing What You've Changed

Subversion keeps track of files in your working copy and will detect any changes you make. You can see which files or directories you've changed and even see exactly what has changed within each file. As a programmer, it's easy to get confused about what changes you have made once you're halfway through a feature, and Subversion can help keep you on track.

The status command tells Subversion to scan all the files in your working copy to see whether they've changed. Subversion uses some tricks to make this process faster, but it still has to look at each file to see whether it has changed. The scan might take a while if you have lots of files in your working copy. If you know that you've been working in only one particular directory, you can ask Subversion to tell you the status of that directory instead of the whole project, which should speed things up.

Subversion marks each file with a letter indicating its status:

A Added in your working copy

C Conflicted, because of an update or merge

D Deleted in your working copy

G Merged with changes from the repository

I Ignored in your working copy

M Modified in your working copy

R Replaced in your working copy

? Not under version control

! Missing from your working copy (removed by non-svn command) or incomplete

The diff command asks Subversion to print out a "unified diff" for each of the files you have changed. This displays the difference between the file as you checked it out and the file after you made your changes. A plus indicates that you have added some new text to the file; a minus means you've removed some text. If you have changed several lines, you might see a group of pluses and minuses indicating that a whole "block" of your file has changed.

The command-line client isn't great at showing diffs—not surprising when a text interface is all it has to work with. The graphical Subversion clients do a much better job displaying diffs, as we'll see in the next tasks.

- See which files have changed in your working copy.

```
mbench> svn status
M       .idea/workspace.xml
M       src/mbench.java
```

- See how your files have changed.

```
prompt> svn diff src/
Index: src/mbench.java
===================================================================
--- src/mbench.java     (revision 6)
+++ src/mbench.java     (working copy)
@@ -1,10 +1,13 @@
 public class mbench {
-    public static int main(String[] args) {
+    public static void main(String[] args) {
         if(args.length != 3) {
             usage();
-            return -1;
+            return;
         }
-        return 0;
+
+        String dbHost = args[0];
+        long docCount = Long.parseLong(args[1]);
+        long runTime = Integer.parseInt(args[2]);
     }

     private static void usage() {
```

Related Tasks

- Task 21, *Viewing the Log*, on page 68
- Task 9, *Seeing What You've Changed Using Tortoise*, on the next page
- Task 10, *Seeing What You've Changed Using Cornerstone*, on page 40
- Task 15, *Reverting Working Copy Changes*, on page 50

9 Seeing What You've Changed Using Tortoise

The Tortoise Check for modifications... command can be used anywhere inside a working copy and will show any changes you have made. Similar to the command-line client, it will do its work faster if you know you're only interested in changes in a particular directory. Each file listed is color coded as well as having its state listed in the "text status" column.

There are a number of checkboxes that control how Tortoise shows changes:

Show unversioned files
: Shows any new files that haven't yet been added to Subversion. It's a good idea to keep this option selected so you don't forget to add new files when committing to the repository.

Show unmodified files
: Useful only on really small projects, this will show files that *haven't* been modified.

Show ignored files
: Not usually very useful, this will show any files that Subversion is currently ignoring. For more information on ignored files, see Task 16, *Ignoring Files*, on page 52.

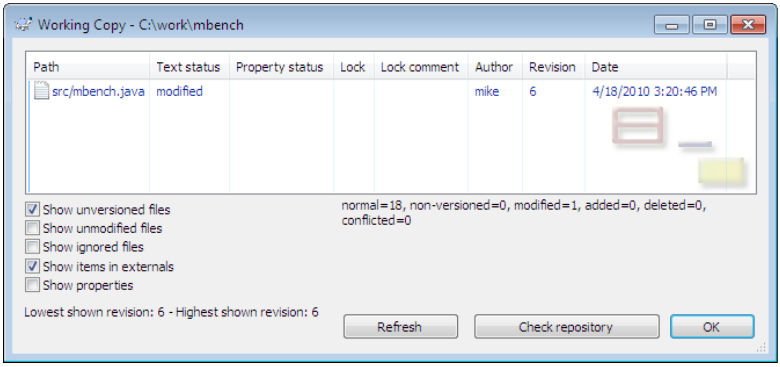
Show items in externals
: If you are using Subversion *externals* to pull items from another repository URL into your working copy, this option will show any changes to those files. See Task 44, *Using Externals*, on page 130 for more information on externals. You should usually keep this option selected.

Show properties
: Shows changes to file properties. You can usually keep this option deselected unless you're specifically changing file properties.

You can right-click changed files to see a context-sensitive menu for each item. You can view the recent history, undo your changes, open the file, and so on. If you double-click a changed file or right-click and choose "Compare with base," Tortoise will pop up a graphical diff window showing the changes. By default Tortoise uses TortoiseMerge to show diffs, but you can configure it to use your favorite diff tool.

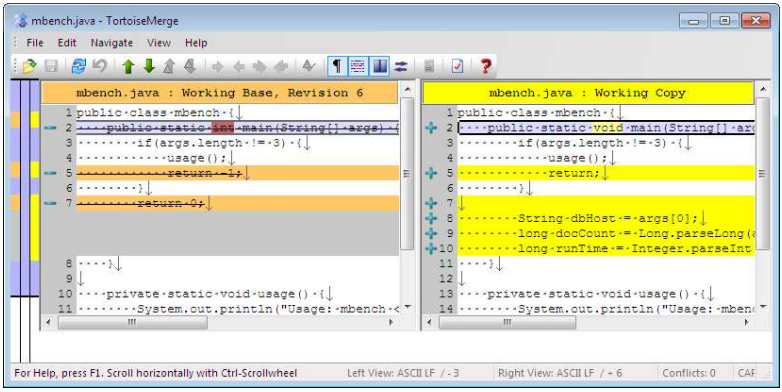
- See which files have changed in your working copy.

  Right-click the base directory of your working copy, and choose TortoiseSVN > Check for modifications.... Tortoise will show a window like this:



- See how your files have changed.

  In the Tortoise modified files window, double-click a file that you've changed. Tortoise will pop open a diff window showing the changes to that file.



Related Tasks

- Task 21, *Viewing the Log*, on page 68
- Task 15, *Reverting Working Copy Changes*, on page 50

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)