
iPad 3G
AnswerHub
The Enterprise Q&A Platform
Collect, Manage and Share All the Answers Your Team Needs
> Q&A Software for Enterprise Knowledge Sharing
> Mobile-friendly Responsive Design
> Easy to Customize with Themes and Plugins
> Integrates with Your SSO and IT Infrastructure
> Choose Cloud-based or On-premise Hosting
Watch Video
Free Demo
Pricing

DZone Refcardz

CONTENTS INCLUDE:

Getting Started with Subversion

By: Lorna Jane Mitchell

INTRODUCTION TO SUBVERSION

Subversion is a source control tool—a way of keeping track of the changes made to any text-based document. Typically we use these types of tools for code (as the name suggests), but they're also useful for configuration files, text-based markup, and anything else where a diff is useful. Source control tools can also handle different versions of binary files such as images.

This Refcard is intended for both version control novices and experienced developers who want to use Subversion more effectively.

Subversion has been around for over 10 years, and is now a stable and well-supported technology. It is a centralized system, meaning that it has one central repository that everyone connects to. Developers do their work in their own working copies.

The repository resides on a server and should be backed up safely (more information on backing up can be found in the SvnAdmin section). Anyone wanting to work on the contents of the repository checks out a working copy onto their own machine. They work with the code locally, making changes. Then these can be committed back to the repository, making those changes available to everyone else.

It's possible to connect to a Subversion repository in a number of different ways:

- **file**: access files if you're on the same machine as your repository. This is only really useful for playing with Subversion to try it out.
- **svn**: simple protocol, suitable for secure networks (including VPN)
- **svn+ssh**: good choice if SSH is already in use, Subversion can make use of this protocol
- **http**: make the repository available over the Internet, using Apache's features for security and to add SSL support. This is the most flexible solution, catering for distributed teams and existing identity systems.

Subversion is a well-understood and established protocol, which means that there is no need to standardize on the tool used to access the repository. All modern IDEs will have support for Subversion built in or easily available as a plugin, so developers can easily integrate the tools into their existing workflow. Subversion is well supported across all operating system platforms, making it a very easy way to make code deployable to a *nix hosting platform, while developers might be using Windows or OSX. Many of these will be GUI tools, but the example commands here are shown in the simplest form using the CLI client.

INSTALLING SUBVERSION

Subversion is available for all platforms, and is quick and simple to install. It is important to make the distinction between installing the client to work with remote repositories and installing the server in order to create and host your own repositories. Many packages include both, however.

To find the right package for your system, visit the packages page on Subversion's website:

http://subversion.apache.org/packages.html

Either download the package or follow the installation instructions for your package manager as appropriate. Source code is available if you wish to compile your own copy of Subversion, but this is not necessary for the majority of installations.

TOOLS FOR WORKING WITH SUBVERSION

A great many tools are available for working with Subversion, too many to list here, but in essence you have three choices.

Developers in particular may choose to add an **extension for Subversion to their existing IDE tool**. Since Subversion is stable and mature, it is well-supported in almost all modern IDEs (and many quite ancient ones). Using Subversion in this way can be an easy way to add a new tool into an existing setup, and is usually quite an approachable way to work. Some of the extensions may have incomplete support for some Subversion commands, however.

Another option is to install an alternative **GUI tool** that you can use to manage your interactions with Subversion. A great example is TortoiseSVN (http://tortoisesvn.net/), which is a GUI tool that integrates tightly with Windows Explorer. Other, equivalent tools are available for other platforms.

Finally, you may choose to work with Subversion from the **command line**. This is a great solution when working on a server that doesn't have a GUI, or which is accessed over SSH, for example. For administrative and advanced use of Subversion, the command line can sometimes give access to more commands than the other tools have support for – and often the best way to solve any difficult Subversion problems is from the command line. Whichever tools you choose, it's likely that you'll see Subversion from the command line at some point, which is why the commands documented here use that interface.

ABOUT REVISION NUMBERS

Subversion refers to different versions of a repository as "revisions" and each one has a number. In contrast to other, newer version control systems, Subversion's revisions refer to a snapshot; the "revision" is a point-in-time status of the whole repository, rather than a changeset.

As a result, we speak of a file, directory or project being at a particular revision.The changeset is then a comparison between two revisions.

Revision numbers in Subversion are incremental, starting from zero in a new repository and increasing with every successful commit to that repository. Whenever a file is changed, all the files in the repository increment their revision number, regardless of whether that specific file was affected by the change. This means that a sequence of changes to a single file can show huge gaps in revision number history – because the intervening changes did not affect that file.

The numbering system is in contrast to other version control systems that you may have used. For example CVS gives each file its own individual version number, whereas Git gives unique identifiers to changesets rather than snapshots.

WORKING WITH THE REPOSITORY

With Subversion now installed, you are ready to start using the commands to work with a repository.

svn checkout

Create a working copy from the repository in the local directory. You can checkout just a subdirectory of the repository if you only want a single project or branch.

```
svn checkout http://svn.example.com/projects/project1 myproject
```

Checks out the "project1" subdirectory from the repository at http://svn.example.com/projects to a new directory called "myproject" within the current directory.

svn update

Bring new changes from the repository into the working copy in this directory. The command needs no arguments; it picks up all the information it needs from the metadata in the current directory.

```
svn update
```

If you see conflicts when you update, look at the svn resolve command for help.

svn status

Check the current status of your working copy, showing any files that are modified, new, or have other special statuses. You can run svn status from a subdirectory to only see changes in that directory and below it.

```
svn status
```

The list does not show files that are in sync with the repository, as there's no special information about those. The list shows a code in the left hand column, which denotes the status. The codes are:

A	Added - This new file will be included with the next commit.
C	Conflicted - This file has local and remote changes which affect the same line.
D	Deleted - This file will be deleted on the next commit.
M	Modified - This file has been changed, and the changes will be included in the next commit.
?	This file is not under version control (you can add it now, if you like)
!	This file is missing. To reinstate it, run svn update. To delete files safely, use svn rm. If this file shouldn't be under source control, consider setting an ignore on it (see the "Properties" section).
X	Represents an external resource managed by svn (svn:externals).

Running svn status is strongly recommended before updating or committing files.

svn add

Bring a new file under source control. When you create a new file and then run svn status—the file will show with a "?". To add a file called script1.php to your project so that it will be committed, use svn add:

```
svn add script1.php
```

Next time you commit to the repository, this file will be included.

svn revert

Removes the local, uncommitted changes to this file. The revert will change the file to the newest version that came from the repository.

```
svn revert script1.php
```

svn commit

Sends local changes to the repository, usually after completing a bug fix or task. Any files shown as M or A by svn status will be sent to the repository. You will be prompted to include a commit message to say what you changed and why, or you can specify this on the command line with the -m switch.

```
svn commit –m "commit all my files"
```

When you commit from a subdirectory only the changes in the current directory and below will be included.

```
cd images
svn commit –m “commit only changes in the images directory”
```

To include only specific files, list those files at the end of your command.

```
svn commit –m “commit only the CSS file” style.css
```

It is recommended to run two commands before you commit:

- svn update to synchronize with the central repository. If any of your changes conflict with changes made to the repository, you will resolve these before you commit.
- svn status to check which changes will be sent.

svn resolve

This command enables you to handle conflicts. Conflicts occur when there are multiple changes to the same line number in a file; Subversion doesn't know what to do about that and will ask you to resolve the conflict yourself. Conflicts occur when you run svn update from the main repository, or when you commit if you didn't update first.

Many clients will offer you the opportunity to resolve the conflict interactively. You will see which lines were changed locally and which were changed on the repository—you can then either pick which version should be used or edit the file as appropriate. When handling source code, many conflicts occur because two developers have added new functions to the end of the same library file – and Subversion just sees that the same lines have been changed. If you resolve interactively, then all is well. If not, you will need the svn resolve command.

When a file is conflicted, it will show the C label in svn status. It is not possible to commit from this repository until the conflict is resolved. When you have a conflict, some extra files appear in your working copy. These have extensions showing which revision they are from, and they are:

- .mine is the local version of your file from before the conflict.
- .rX (where X is a revision number) are the committed revisions from the repo. One will be the version your changes were based on, and the newer one is the current version in the repository.

The actual file with the conflict will have notation in the file showing which version is in which revision, and it will look something like this:

```
<<<<<<< .mine
$greeting = ‘hello world’;
print $greeting;
=======
$message = ‘hello world’;
echo $message;
>>>>>>> .r25
```

Either copy one of the “extra” files over the original, or edit the file with the conflicts in until it looks right. Make sure you remove all the ASCII art notation from the file! Once you are ready, you can tell Subversion that you have dealt with the conflict:

```
svn resolve script1.php
```

The extra files will then be deleted, and the svn status command will show M rather than C for this file. Once done, you can commit your changes.

RESEARCHING PROJECT HISTORY

svn blame

The blame command has two nicer-sounding aliases: “praise” and “annotate”. Whichever you use, these commands will mark up a file and show which user last changed which line and in which revision:

```
svn blame index.php
    715         lorna <html>
    715         lorna <head>
    762         eric   <link rel=”stylesheet” type=”text/css” href=”style.css” />
    715         lorna </head>
    715         lorna <body>
    584         lorna <?php
```

This example shows a snippet of a file, with the revision number of the most recent change in the first column, followed by the name of the user that made that commit, and then the contents of the line.

svn diff

Show the difference between two versions of the code. This might be showing the local changes, the difference between two revisions, or the difference between two remote paths. Here are some different examples to give you an idea:

```
svn diff
```

Shows which local changes you have made to your working copy that aren't in the repository.

```
svn diff –r 18:25
```

Show the changes made between revision 18 and revision 25

```
svn diff http://svn.example.com/projects/stuff/trunk http://svn.example.com/projects/stuff/branches/branch1
```

Show the changes between the trunk and branches/branch1 versions, both in the remote repository.

Regardless of how the diff is generated, it shows up in “diff” format, with a plus sign before any line that was added, and a minus sign before any that were removed. Changed lines show with the old version removed and the new version added. Here's an example of the output from the diff command:

```
Index: index.php
===================================================================
--- index.php       (revision 18)
+++ index.php       (revision 25)
@@ –14,7 +14,7 @@
$event_id = 951;
 $twitter_url_stem = ‘http://search.twitter.com/search.json?q=’;
–$searches = array(“PHP”, “agile”, “technology”);
+$searches = array(“PHP”, “agile”, “nodejs”);
```

This format is split into sections—first by file, then by line number chunk and length (both the old and new line numbers and section lengths are given). If you have used the diff and patch programs to apply changes in the past, you'll be familiar with this format.

svn log
The log command shows the story of changes that have been made. The output shows the revision number, the user that committed the change, and the date it was made. The commit message also shows.

```
svn log


------------------------------------------------------------------------
r23 | lorna | 2011-10-11 19:12:59 +0100 (Tue, 11 Oct 2011) | 1
line

updating the variable name
------------------------------------------------------------------------
r22 | lorna | 2011-10-11 19:05:07 +0100 (Tue, 11 Oct 2011) | 1
line

removing those folders
------------------------------------------------------------------------
r21 | lorna | 2011-10-11 19:02:46 +0100 (Tue, 11 Oct 2011) | 1
line

adding some pointless empty folders for examples
------------------------------------------------------------------------
```

Append a -v switch to also see which files were changed with each commit.

HANDLING FILES SAFELY

Subversion controls your files, which allows it to track changes, but also means we need to allow it to track filesystem operations such as moving or removing files.

svn copy
Use in place of your usual copy command to safely copy files and directories to a new location. This will also copy the history of the file/directory.

```
svn copy file1 file2
```

Copying directories manually is particularly troublesome, as the .svn folders inside them then contain inaccurate information and these incorrectly-copied directories cannot then be added to Subversion, so remember to use the copy command.

svn remove
This command is also aliased to svn delete, and it removes the file from the directory. Using svn status, the file will be listed with a D shown, and then will disappear completely on the next commit.

```
svn remove script1.php
```

Information about the deleted file remains in the repository and it can be "undeleted" as needed, by reverting this change. If you did delete your file without using this approach, you will see the exclamation mark (!) in the output of svn status to indicate that the file is obstructed. To correct the problem, run svn update and then remove the file using the svn remove command.

svn move
This command allows safe renaming of files, allowing Subversion to retain control of the files and to persist the history across the rename operation. When you rename a file, svn status will show the old file with a D and the new one with an A. Then, the change will be included with the next commit.

```
svn move script1.php better-filename.php
```

MANAGING CODE AND REPOSITORIES

svn export
Look closely at your working copy; there are directories at every level called ".svn". These are Subversion's metadata files that allow it to keep track of your files. It means, though, that every working copy contains these, and when we put code live we don't want to include them.

The export command gives you the files from your project, but without any Subversion data included. Run the command from the location you want your new export to go.

```
svn export http://svn.example.com/projects/project1 myproject
```

This command takes the code from the project1 folder in the repository, and places it (without any Subversion metadata) into the myproject folder. The export command is particularly useful as part of your deployment process, since you don't want to publish any Subversion information on your live site or include it in your build.

svn info
Gives information about the current working copy. This includes the root of the working copy, the remote URL of the repository it was checked out from, and which revision both the working copy and the revision are at.

```
svn info
```

svn relocate
Allows a different remote location to be associated with an existing working copy.

```
svn relocate http://svn.example.com/projects http://example-svn.
com/projects
```

This might seem like a rather drastic move, and it is. It is mostly used when the repository URL changes for some reason, or changes protocol.

WORKING WITH BRANCHES AND TAGS

In Subversion, a branch is just a copy of a project. By convention, each project folder contains three subdirectories: trunk, branches, and tags. "Trunk" is usually the main version of a project, and branches and tags are copies. Branches are separate versions of the code, often used to work on a specific release or feature, allowing many commits over time and collaboration before the feature or branch is ready. Tags are also copies, but (again by convention) these are not edited after they are created. Think of tags as labels, they are usually used to mark a particular milestone or release.

svn copy
The copy command is used to create branches and tags. For example, creating a new branch from the trunk of a project:

```
svn copy http://svn.example.com/projects/project1/trunk http://
svn.example.com/projects/project1/branches/new-feature
```

This will create the copy on the remote server, and this can be checked out as a working copy:

```
svn checkout http://svn.example.com/projects/project1/branches/
new-feature new-feature
```

Use this approach for creating tags as well as branches.

svn merge
This merges the difference between two revisions into your local working copy. It can be tricky, but Subversion has a feature called "merge tracking," which means that it knows which revisions have already been merged into a branch and it won't merge the same ones repeatedly.
Before merging, use svn diff and check that the difference between the two revisions looks correct – in particular that the plus and minus signs are what you expect. The svn merge command applies the diff (literally like applying a patch, if you've used patch) to your working copy, so checking the diff shows you what will happen.

```
svn merge –f 21:HEAD http://svn.example.com/projects/project1/
trunk
```

Take the time to check that everything looks right before you commit. You can also perform this on a clean working copy, only committing if the merge is successful.

It's also possible to "reverse merge"—to undo changes from a particular commit or range of commits, without undoing everything. To achieve this, supply the newest revision number first, and the older one second. Just as with any other merge, using the svn diff command first will show clearly which changes will be made.

TEAM COLLABORATION WITH SUBVERSION

The examples so far have related mostly to a single developer's view of the world, but Subversion is a powerful collaboration tool.

Subversion allows different people to work on the same system, adding each set of changes to a central copy of the repository. For teams with more than one person in them, this saves a lot of time in reconciling work from different people, and also prevents mistakes!

Using a source control system means having a single, canonical version of your project in a known place. The team is always certain which is the latest version of the code; anything not committed to the repository simply doesn't exist.

Branches allow larger teams to work and collaborate safely on larger features or different generations of a single product. It's common to branch for all new features where the work will take more than, say, half a day to complete. Using branches means learning to use the merge command, but also brings many benefits:

- Use the source control tool to commit code, making sure no work will be lost during the development of a complex feature
- Gain the advantage of using Subversion during feature development, so that even partial features can be committed at regular intervals, and changes shared with others to get the work completed
- Merge in changes from trunk to the branch, and test the new features from here, before the changes are merged into the main version of the project

Tags are another key feature; they're particularly useful for adding information to the repository when the code is deployed, so that it's always clear which version of the code is currently live.

SETTING SUBVERSION PROPERTIES

Subversion recognizes a series of properties that you can use to configure how certain files and directories should behave.

svn propset

The most common example is to set a property to ignore files:

```
svn propset svn:ignore '*.pyc' ./src/
```

This example adds an "ignore" rule to the src directory that stops files ending in .pyc from showing as new in svn status. This is very useful for generated content, cache files, or other items that don't belong under source control. It's also possible to supply multiple patterns, and to do so by supplying a file that holds these.

The general format of the command is:

```
svn propset [property name] [value] [target]
```

There are no restrictions on the properties that you can set, but there are some that are supported by Subversion and are very helpful:

- svn:ignore: unversioned files matching this pattern aren't shown by svn status
- svn:executable: file is executable (*nix systems only)
- svn:eol-style: the line ending format to use
- svn:externals: sets a reference for another repository path to be included here

When the properties are changed on a file or directory, these will appear in the output from svn status with an M in the second column along. This is to show that while the file or directory itself has not changed, the data about it has done so. As with other kinds of change, this will be included with the next commit.

Other Commands for Subversion Properties

Use svn proplist to see which properties are set on a particular file or directory:

```
svn proplist index.php
```

To check the current values of any properties that are set, the svn propget command can be used:

```
svn propget svn:ignore src
```

If a property is set but should be removed, the svn propdel command will do so:

```
svn propdel svn:ignore greeting.php
```

Just as when setting a property, deleting a property will result in a modification showing for the affected path in the output of the svn status command.

ADMINISTERING SUBVERSION WITH SVNADMIN

The svnadmin command is used for working with repositories rather than working copies, and it is used mainly for administration tasks. Run the svnadmin commands from the machine where the repository exists, and use the file path to refer to the repository.

svnadmin create

Use svnadmin create to instantiate a new, empty Subversion repository:

```
svnadmin create /path/to/new/repo
```

You can then start adding content using the usual svn commands.

svnadmin hotcopy

The hotcopy command allows you to safely duplicate a repository to a different location on the file system, ensuring that all information is stored consistently.

```
svnadmin hotcopy /path/to/repo /path/to/copy/to
```

For backup purposes, it isn't safe to simply copy the files holding the repository, so use the hotcopy command instead.

svnadmin dump

The svnadmin dump command is the export feature for Subversion. It takes the repository you specify and writes the contents of it to stdout, so you will probably want to capture its output in a file:

```
svnadmin dump /path/to/repo > repo.svndump
```

svnadmin load

The sister command to svnadmin dump, svnadmin load allows the loading of a dumped repository format into a repository. It reads from stdin, so it is usually used with a file directed into it:

```
svnadmin load /path/to/repo < repo.svndump
```

Together, the load and dump commands give us import and export functionality for Subversion.

svnadmin verify

The verify command is a spot of housekeeping; this command will fail if there are any revisions which have been corrupted. Use it at regular intervals and make sure that notifications go to the person responsible for the repository if any failure occurs.

SUBVERSION IS QUICKER WITH SHORTCODES

A great many of Subversion's commands also have shorter aliases, which are quicker to type. These are handy to know so we've compiled a list of them for you:

Shortcode	command
co	checkout
up	update
ci	commit
ann	blame / annotate
di	diff
cp	copy
rm/del	remove / delete
mv	move
sw	switch
h / ?	help

Keeping these at your fingertips will help you to streamline the way you work with Subversion.

RESOURCES AND FURTHER READING

You can find out more about Subversion on its project page: http://subversion.apache.org/

There is a fabulous, freely available, online reference book for Subversion called the Red Bean Book. You can find it here: http://svnbook.red-bean.com/ The book is also available in print, as "Version Control with Subversion" from O'Reilly.

Another excellent book is "Pragmatic Version Control Using Subversion", which has a hands-on, tutorial style.

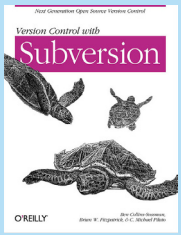
A single-page cheat sheet, listing the commands, is available from Added Bytes: http://www.addedbytes.com/cheat-sheets/subversion-cheat-sheet/ There are also providers of online repository hosting that may be useful. Here are a few:

- Beanstalk: http://beanstalkapp.com/
- Unfuddle: http://unfuddle.com/
- Code Spaces: http://www.codespaces.com/

ABOUT THE AUTHOR

Lorna Jane Mitchell is a PHP developer, blogger, trainer and evangelist from Leeds in the UK. She is active with phpwomen.org and her local user group PHP North West, and writes for a variety of outlets, including her own blog at lornajane.net. She is an active member of the PHP and open source communities and contributes to the joind.in event feedback project. When she's not at her computer, Lorna enjoys yarn craft, hobby electronics, and her home renovation project.

RECOMMENDED BOOK



One of the greatest frustrations in most software projects is version control: the art of managing changes to information. Today's increasingly fast pace of software development--as programmers make small changes to software one day only to undo them the next--has only heightened the problem; consecutive work on code or single-programmer software is a rare sight these days.

Buy Here

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-80-6
ISBN-10: 1-936502-80-1
50795
9 781936 502806
$7.95

Version 1.0