FREE eBook

LEARNING

svn

Free unaffiliated eBook created from
Stack Overflow contributors.

#svn

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: svn

It is an unofficial and free svn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official svn.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with svn

Remarks

Apache Subversion (SVN) is a universal and centralized open source version control system. Subversion is currently a project under Apache Software Foundation (ASF) and is licensed under the Apache License, Version 2.0.

Subversion is designed to manage and control files and directories and track changes made to them; it acts as a reliable *time machine* and *management tool* for the collaboratively developed projects. It can easily answer the standard questions any version control system has to answer reliably. For example,

- How did the project/file FOO look like on 12/12/2012 ?
- What changes were introduced by USERNAME or on 20/12/2012 ?
- Who modified the particular string since the last review?
- and much-much more.

Versions

Version	What's new?	Release Date
1.9.x	Numerous usability and performance improvements	2015-08-05
1.8.x	Improved rename tracking, automatic reintegration merges, inherited versioned properties, built-in conflict resolution tool	2013-06-18
1.7.x	Complete rewrite of the working copy library, improved HTTP protocol usage	2011-10-11
1.6.x	Identifying tree conflicts, improved interactive conflict resolution, repo-relative URLs support	2009-03-20
1.5.x	Merge and branch tracking (`svn:mergeinfo`), interactive file conflict resolution, sparse checkouts, improved `svn:externals` syntax	2008-06-19
1.4.x	`svnsync` tool for repository replication, new and improved working copy library	2006-09-10
1.3.x	High-level logging of user operations on the server side, performance improvements	2005-12-30
1.2.x	Support for lock-modify-unlock model (i.e. locking), DAV autoversioning, FSFS is used by default for new repositories	2005-05-21

Version	What's new?	Release Date
1.1.x	New repository backend (FSFS), symlink versioning	2004-09-29
1.0.x	Initial public release	2004-02-23

Apache Subversion 1.9.x is currently the latest and best SVN release that is fully supported. Subversion 1.8.x is partially supported. Subversion 1.7.x and earlier versions are no longer supported.

Examples

Installation and initial setup

Install the `svn` client to start collaborating on the project that is using Subversion as its version control system.

To install Subversion, you can build it yourself from a source code release or download a binary package pre-built for your operating system. The list of sites where you can obtain a compiled Subversion client (`svn`) for various operating systems is available at the official binary packages page. If you feel like compiling the software for yourself, grab the source at the Source Code page.

With Subversion, you are not limited to using only the standard `svn` command-line client. There are some notable graphical Subversion clients for various operating systems and most of the IDEs nowadays provide robust integration with SVN right out of the box or via plugins. For the list of graphical clients, check the Wikipedia page: https://en.wikipedia.org/wiki/Comparison_of_Subversion_clients.

Right after you install the client you should be able to run it by issuing the command `svn`. You should see the following:

```
$ svn
Type 'svn help' for usage.
```

Everything is mostly ready. Now you should create a local workspace called a *working copy* which is going to be connected to the remote central *repository*. In other words, you are going to *checkout a working copy*. You are going to operate with the versioned data with the help of the working copy and can publish your changes (called *committing* in SVN) so that others working on the same project can see them and benefit from your changes. To later fetch the future changes made by others from the repository, you would *update your working copy*. These basic operations are covered in other examples.

Checking out a working copy

To begin making modifications to the project's data, you have to obtain a local copy of the versioned project. Use the command line `svn` client or your favorite SVN client (TortoiseSVN, for example). Your local copy of the project is called a *working copy* in Subversion and you get it by issuing the command `svn checkout <URL>` where `<URL>` is a repository URL. e.g.

```
$ svn checkout https://svn.example.com/svn/MyRepo/MyProject/trunk
```

Alternatively, you can use `svn co <URL>` as a shorthand in order to checkout a local copy.

As a result, you will get a working copy of the `/trunk` of a project called MyProject that resides in MyRepo repository. The working copy will be located in a directory called `trunk` on your computer relative to the directory you issued the command in.

If you wish to have a different name for your working copy you can add that as a parameter to the end of the command. e.g.

```
$ svn checkout https://svn.example.com/svn/MyRepo/MyProject/trunk MyProjectSource
```

This will create a working copy called `MyProjectSource`.

Note that instead of checking out the trunk, you could check out some branch, private shelve or a tag (assuming they already exist in the repository); you can have unlimited number of local working copies on your machine.

You could get the working copy of the whole repository MyRepo, too. But you should refrain from doing so. Generally speaking, you do **not** need to have a working copy of the whole repository for your work because your working copy can be instantly switched to another development branch / tag / whatever. Moreover, Subversion repository can contain a number of (un)related projects and it's better to have a dedicated working copy for each of them, not a single working copy for all of the projects.

Exporting the versioned data (plain download)

If you want to get the versioned project's data, but you don't need any of the version control capabilities offered by Subversion, you could run `svn export <URL>` command. Here is an example:

```
$ svn export https://svn.example.com/svn/MyRepo/MyProject/trunk
```

As a result, you will get the project's data export, but unlike with a working copy, you won't be able to run `svn` commands on it. The export is just a plain download of the data.

If some time later you'd want to convert the downloaded data to a fully-functional working copy, run `svn checkout <URL>` to the directory where you ran the export to.

Updating a working copy

You are not the only person working on the project, right? This means that your colleagues are also making modifications to the project's data. To stay up to date and to fetch the modifications

committed by others, you should run `svn update` command in your working copy. As a result, your working copy will sync with the repository and download the changes made by your colleagues.

Shorthand for `svn update` is `svn up`.

It is a rule to run `svn update` before committing your changes.

Making changes in your local working copy

The *working copy (WC)* is your *local and private workspace* that you use to interact with the central Subversion repository. You use the working copy to modify the contents of your project and fetch changes committed by others.

The working copy contains your project's data and looks and acts like a regular directory on your local file system, but with one major difference -- the working copy tracks the status and changes of files and directories within. You can think of the working copy as of a regular directory with a version-control flavor added by a hidden `.svn` metadata directory at its root.

Most of the time, you are going to perform modifications to the project's data by modifying the contents of the working copy. As soon as you are satisfied with the modifications and you've reviewed them thoroughly, you are ready to publish them to the central repository.

You can perform any actions with your project's data within the working copy, but operations that involve copying, moving, renaming and deleting must be performed using the corresponding `svn` commands:

- **Modifying existing files**. Modify the files as you usually do using your favorite text processor, graphics editor, audio editing software, IDE, etc. As soon as you save the changes to disk, Subversion will recognize them automatically.
- **Adding new files**. Put new files to the working copy and Subversion will recognize them as *unversioned*. It will not automatically start tracking the new files unless you run `svn add` command:

  ```
  svn add foo.cs
  ```

- **Moving files and directories**. Move files and directories using `svn move` command:

  ```
  svn move foo.cs bar.cs
  ```

- **Renaming files and directories**. Rename files and directories using `svn rename` command:

  ```
  svn rename foo.cs bar.cs
  ```

  NOTE: `svn rename` command is an alias of `svn move` command.

- **Copying files and directories**. Copy files and directories using `svn copy` command:

```
svn copy foo.cs bar.cs
```

- **Deleting files and directories**. Delete files and directories using `svn delete` command:

```
svn delete foo.cs
```

- **Checking the status of files and directories in the working copy**. Review your changes using `svn status` (or `svn st` for short) command:

```
svn status
```

> **IMPORTANT:** Always review your changes before committing them. This will help you to avoid committing unnecessary or irrelevant changes.

- **Reverting changes**. Revert your changes using `svn revert` command:

```
svn revert foo.c
```

- **Reverting all changes**: From the repository's root:

```
svn revert -R .
```

> **IMPORTANT:** Reverted uncommitted changes will be lost forever. You won't be able to recover the reverted changes. Use `svn revert` with caution! If you want to keep the changes but need to revert, save them in a patch. See example of how to create and apply a patch.

Committing your local changes to the repository

To publish the changes you made in your working copy, run the `svn commit` command.

> **IMPORTANT:** Review your changes before committing them! Use `svn status` and `svn diff` to review the changes. Also, make sure you are in the correct path before performing a commit. If you updated many files across various directories, you should be at the appropriate level to include all of them beneath your location.

Here is an example of the commit command:

```
svn commit -m "My Descriptive Log Message"
```

Alternatively, `svn ci` is the shorthand for `svn commit`

Note the `-m (--message)` option. Good commit messages help others understand why a commit was made. Also, on the server side it's possible to enforce non-empty messages, and even enforce that each commit message mentions an existing ticket in your bug tracking system.

Checking out a working copy at a specific revision

To get version 5394 use:

```
svn co --revision r5394 https://svn.example.com/svn/MyRepo/MyProject/trunk
```

Or the shorter version:

```
svn co -r 5394 https://svn.example.com/svn/MyRepo/MyProject/trunk
```

Or by using pegged revisions:

```
svn co https://svn.example.com/svn/MyRepo/MyProject/trunk@5394
```

If already checked out, you can use the `update` command to move a to a particular revision, by doing:

```
svn up -rXXX
```

Using a password-protected repository

A Subversion repository can be configured so that certain contents or commands are only accessible to certain users. In order to access this restricted content, you will need to specify a username and password.

Your username and password can be specified directly as part of the command:

```
$ svn checkout https://svn.example.com/MyRepo/trunk --username JoeUser --password topsecret
```

Unfortunately, this causes your password to appear in plaintext on the console. To avoid this possible security problem, specify a username but not a password. Doing this will cause a password prompt to appear, allowing you to enter your password without exposing it:

```
$ svn checkout https://svn.example.com/MyRepo/trunk --username JoeUser
Password for 'JoeUser':
```

Providing no authentication information at all causes Subversion to prompt you for both the username and password:

```
$ svn checkout https://svn.example.com/MyRepo/trunk
Username:  JoeUser
Password for 'JoeUser':
```

While the first method is less secure, it's frequently seen in automated scripts since it is difficult for many types of script to provide information to an interactive prompt.

Note: commands that only operate on your working copy (such as `revert` or `status`) will never require a password, only commands that require communicating with the repository server.

Creating and applying patches

A patch is a file that show the differences between two revisions or between your local repository and the last revision your repository is pointing.

To share or save a patch of your local uncommitted changes either for peer review or to apply later, do:

```
svn diff > new-feature.patch
```

To get a patch from the differences between two revisions:

```
svn diff -r NEWER_REVISION:OLDER_REVISION > feature.patch
```

To apply a patch, run:

```
svn patch new-feature.patch
```

> In order to apply the patch successfully, you must run the command from the same path where the patch was created.

Reviewing the logs

Running `svn log` will show you all the commit messages, you probably want to review only certain revisions.

- View the `n` most recent revisions:

  ```
  svn log -n
  ```

- View a specific revision:

  ```
  svn log -c rXXX
  ```

- View the paths affected:

  ```
  svn log -v -c rXXX
  ```

Revert or rollback of a file

To restore a file to the latest updated svn version, i.e. undo the local changes, you can use `revert`:

```
svn revert file
```

To restore a file to an older version (revision XXX) use `update`:

```
svn update -r XXX file
```

Warning: in both cases you will lose any local changes in the file because it will be overwritten.

To only view the older version of a file use `cat`:

```
svn cat -r XXX file
```

And to view the differences with your local version of the file:

```
svn diff -r XXX file
```

Read Getting started with svn online: https://riptutorial.com/svn/topic/638/getting-started-with-svn

Chapter 2: Administering SVN

Examples

Creating A New Repo

New repository can be created with two different options:

1. Using command line

Execute following command. It will create a directory for the repository, but parent path has to be present. i.e. in the following example, `/var/svn` should already be there, while it will create `my_repository` directory.

```
svnadmin create /var/svn/my_repository
```

If you are using TortoiseSVN, you can use GUI to create repo.

1. Open the directory where you want to create a new repository.
2. Right click on the folder and select `TortoiseSVN -> Create Repository here...`
3. A repository is then created inside the selected folder. Don't edit those files yourself! If you get any errors make sure that the folder is empty and not write protected.

Create new user

To add user, use following command

```
htpasswd /etc/subversion/passwd user_name
```

Specify `user_name` with the username you wish to add in above command. It will prompt to provide password for the user.

If you are creating very first user, you need to add `-c` switch in above command, which will create the file.

```
htpasswd -c /etc/subversion/passwd user_name
```

You can check existence of the file or list of configured users using following command `cat /etc/subversion/passwd`

You might need to execute above commands as super user.

Create user groups

Groups can be defined in `/etc/subversion/svn_access_control` file.

Create/edit the file using following command

`nano /etc/subversion/svn_access_control`

Use syntax specified as below to define groups and assign members.

```
[groups]
groupname = <list of users, comma separated>
```

e.g.

```
[groups]
myproject-dev = john, peter
myproject-support = maria, cristine
```

Above example will create two groups named `myproject-dev` and `myproject-support`. It will add users `john` and `peter` to group `myproject-dev` and users `maria` and `cristine` to group `myproject-support`.

Groups can then be used to manage repository access

Managing repository permissions

Access specifications for subversion repositories is specified `etc/subversion/svn_access_control` file

Create/edit the file using following command

`nano /etc/subversion/svn_access_control`

Use following syntax to configure access permissions for repositories to group/members

```
[Repository:<Path>]
@groupname = r/rw
User = r
```

e.g.

```
[myproject:/]
@myproject-dev = rw
@myproject-support = r
jack = r

[myproject:/branches/support]
@myproject-support = rw
patrick = r
```

Above example configuration will grant read-write access to entire `myproject` repository to users belonging to group `myproject-dev`, while read-only access is granted to users belonging to group `myproject-support` and specific user `jack`. **Note that, group names are preceded by `@`.**

Similarly, it will assign read-write access to `support` branch of `myproject` repository to all users belonging to `myproject-support` and read-only access to `patrick`.

Read Administering SVN online: https://riptutorial.com/svn/topic/7935/administering-svn

Chapter 3: Branching, shelving and tagging in Apache Subversion

Syntax

- svn copy [BRANCH-FROM-URL] [BRANCH-TO-URL] -m `<COMMIT-LOG-MESSAGE>`
- svn copy [^/PATH-TO-BRANCH-FROM] [^/PATH-TO-BRANCH-TO] -m `<COMMIT-LOG-MESSAGE>`

Remarks

As you might have noticed, we use `svn copy` command to create branches, tags and shelves (we'll skip mentioning tags and shelves in the next paragraphs). This is the same command used to copy items in your working copy and into the repository.

`svn copy` is used for branching because, branch is technically a copy of the source you copy from. However, this is not an ordinary copy are familiar with when copying files on your local file system. Branches in Subversion are so called "Cheap Copies" that are similar to symlinks. Therefore, creating a new branch takes minimal time to complete and takes practically no space in the Subversion repository. Create branches and use them for any change you want regardless of the change's size and scope.

`svn copy` can be shortened to `svn cp` as Subversion has aliases for most commands.

Examples

Creating a branch using direct URL to URL copy

Branching in Subversion is *very* simple. In the simplest form, creating a new branch requires you to run the command against the remote repository's URLs. For example, let's create a new branch out of the mainline trunk:

```
svn copy https://svn.example.com/svn/MyRepo/MyProject/trunk
https://svn.example.com/svn/MyRepo/MyProject/branches/MyNewBranch -m "Creating a new branch"
```

The new branch is ready and you can begin working with it. Check out a new working copy with the new branch or switch your existing working copy using `svn switch` command.

Creating a branch through a working copy

When you interact with the remote central repository using your private local workspace -- the working copy -- you can use repository-relative URL instead of direct URL to URL copy to create a new branch:

```
svn copy "^/MyProject/trunk" "^/MyProject/branches/MyNewBranch" -m "Creating a new branch"
```

Switching a working copy to a different branch

An existing working copy can be quickly transformed to reflect the contents of a different branch in the same repository. For example, you might have a working copy of the trunk and now need to work on a development branch. Instead of checking out a completely new working copy (which can waste a lot of time and disk space), you can use the `svn switch` command to efficiently modify your existing working copy:

```
svn switch ^/MyProject/branches/MyNewBranch
```

Your working copy will now reflect the contents of the branch instead of the trunk.

Using tags

"Tags" are a type of label that can be applied to a repository at a certain point in time. They are frequently used to give human-readable names to important milestones so that they can be easily accessed later (for example, "version-1.2").

Creating a tag is exactly the same as creating a branch:

```
svn copy -r 1234 ^/MyProject/trunk ^/MyProject/tags/version-1.2
```

In this specific case, the `-r 1234` argument was used to indicate that the tag should be created from revision 1234 of the trunk.

Subversion doesn't make any distinction between a tag and an ordinary branch. The only difference is in how you decide to use them. Traditionally, no commits are made to a tag once it has been created (to ensure that it remains an accurate "snapshot" of a past repository state). Subversion doesn't enforce any special tag-related rules by default since different people can use tags differently. A repository administrator can, however, set up access control scripts to enforce whatever rules their team has decided to use.

> In Windows, you need to use a double caret `^^`.

Deleting a branch

Just run:

```
svn delete https://svn.example.com/svn/MyRepo/MyProject/branches/MyNewBranch -m "Deleting no longer needed MyNewBranch"
```

Or, using the short URL:

```
svn delete ^/branches/MyNewBranch -m "Deleting no longer needed MyNewBranch"
```

- In Windows, you need to use ^^
- You can always bring back a deleted branch by creating it again specifying the desired revision back then when the branch was alive (a deleted branch is just a branch that is not available in the HEAD revision). For example if branch was deleted at revision 101: `svn copy https://svn.example.com/svn/MyRepo/branches/MyNewBranch@r100 https://svn.example.com/svn/MyRepo/MyProject/branches/resurrected-branch -m "Resurrected MyNewBranch from revision 100"`. See Resurrecting Deleted Items

Read Branching, shelving and tagging in Apache Subversion online:
https://riptutorial.com/svn/topic/668/branching--shelving-and-tagging-in-apache-subversion

Credits

S. No	Chapters	Contributors
1	Getting started with svn	agold, bahrep, bta, Chad Nouis, Community, Eiren Smith, opticyclic, ronnyfm, V-R, wrothe
2	Administering SVN	opticyclic, Rumit Parakhiya
3	Branching, shelving and tagging in Apache Subversion	bahrep, bta, opticyclic, ronnyfm