



# Edge Subversion Guide



# Contents

<b>1</b>	<b>Version Control</b>	<b>9</b>
1.1	What is a Version/Revision Control Tool? . . . . .	9
1.2	Why use a Version/Revision Control Tool? . . . . .	9
1.3	What is the Immediate Benefit for Developers? . . . . .	9
1.4	What is Subversion? . . . . .	10
1.5	How does it work? . . . . .	10
<b>2</b>	<b>Subversion for Edge</b>	<b>13</b>
2.1	Getting Started . . . . .	13
2.1.1	Creating a Repository . . . . .	13
2.1.2	Creating a Working-copy Root-directory . . . . .	14
2.2	The most Commonly used Subversion Subcommands . . . . .	14
2.2.1	Getting Help . . . . .	14
2.2.2	Examining and Comparing Working-copy with the Repository . . . . .	16
2.2.3	Undoing changes to the working-copy . . . . .	16
2.2.4	Updating Working-copy with Changes Committed by Other Developers . . . . .	17
2.2.5	Sending Modified Files back from Working-copy into the Repository . . . . .	17
2.3	Other Useful Subcommands . . . . .	18
2.3.1	Commands for Moving, Removing, and Adding files . . . . .	18
2.3.2	Adding the Keywords to a new File . . . . .	18
2.3.3	Looking Inside the Repository . . . . .	18
2.3.4	Exporting Source Code . . . . .	19
2.4	Manually Merging a Conflict . . . . .	19
2.5	Private Configuration File . . . . .	19
2.6	More Help . . . . .	19
<b>3</b>	<b>Some examples</b>	<b>21</b>



## Abstract

This document is part of Edge, a parallel CFD code developed by FOI, see [Eliasson \(2001\)](#), [Eliasson \(2002\)](#), and [Edge homepage](#).

The concept of the version/revision control tool Subversion is introduced. The text is intended for Edge developers who are beginning to use Subversion.

After reading this short introduction the reader knows how to use Subversion in general and Subversion for Edge in particular. The reader will also know where to search for more detailed information.

The first chapter introduces the concept of version/revision control. The second chapter describes how to use the Subversion tool. The third chapter is a detailed description of how to migrate to Subversion.



# 1 Version Control

---

## 1.1 What is a Version/Revision Control Tool?

A version/revision control tool helps keeping track of changes to the source code and other files for a software project with many developers.

---

## 1.2 Why use a Version/Revision Control Tool?

A version control tool saves time for developers and the project coordinator. It simplifies cooperation and communication between developers. It is considered an essential tool for maintaining good quality because it is a prerequisite for configuration management, appropriate testing, bug tracking and other important parts of modern software development.

---

## 1.3 What is the Immediate Benefit for Developers?

- When a bug is fixed or a new feature added by another developer you can merge it into your private, modified source code with a single command `svn update`. In 98% of the cases there is no need to manually merge any files.
- You can get any old version of Edge at any time, this is good for comparing different versions when suspecting a bug.
- You can examine the log to see who is responsible for a change, what was changed and when it was done.
- Subversion can automatically update some special fields like *last modified by* and *last modification date* in the file headers so you don't need to think about this. See [Collins-Sussman \*et al.\*](#) for more details.

---

## 1.4 What is Subversion?

Subversion is a modern tool for version/revision control. From the user's point of view it works very similar to the de facto standard tool CVS. However, Subversion has several essential improvements.

---

## 1.5 How does it work?

Subversion works by keeping all sources including all history in a database on one server computer. This database is called the repository and contains all versions of all files stored in an efficient manner.

Each developer has his/her own private working-copy of the files. Each working-copy directory contains a special directory named '.svn' where some administrative files are stored. The developer decides himself when and how to synchronize his working-copy with the repository.

The project responsible may choose to give read-only access or write-access to the repository for individual developers. Write-access may be restricted to some subdirectories individually for each developer (of course each developer always has full read/write access to his/her working copy). In this way developers responsible for different parts of the code can work independently and efficiently.

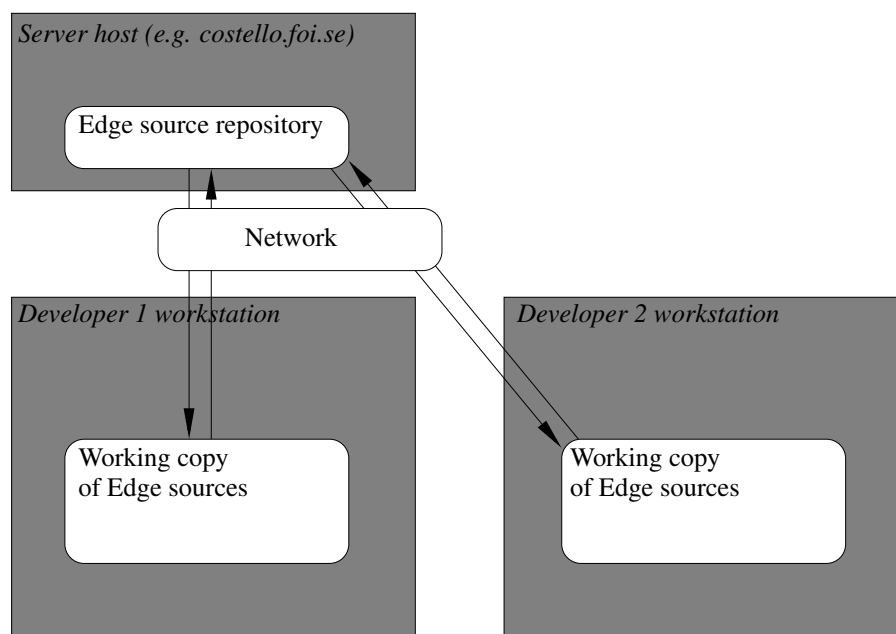


Figure 1. Subversion server and two clients.

Developers with write-access should be careful so that they do not commit faulty files into the repository. At minimum, they must check that their code compiles without errors.

A more detailed view of a working-copy and its relation to the repository is shown in figure 2. The repository may contain several copies/versions of the source code organized in a directory tree.

A working-copy directory contains visible working-copies of the files and a hidden directory named `.svn` where administrative files used by Subversion are kept. The `.svn` directories contain information on what repository the working-copy belongs to. It also contains a copy of the repository files (sometimes referred to as *the mirror*).

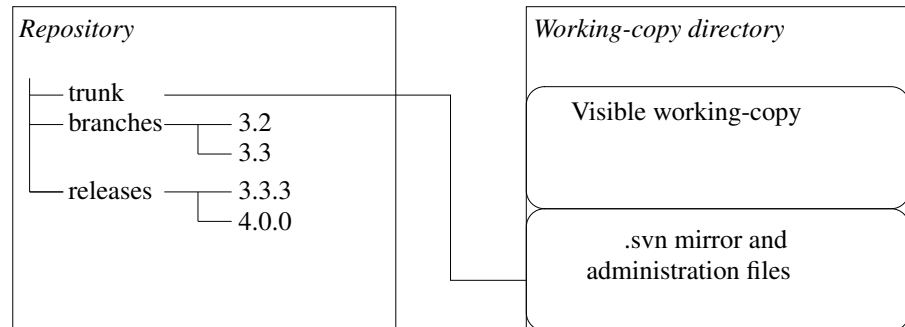


Figure 2. Subversion repository and working-copy.



## 2 Subversion for Edge

---

### 2.1 Getting Started

#### 2.1.1 Creating a Repository

Only a repository administrator needs to create repositories. Since the Edge repository already exists (at URL <https://costello/svn/edge/>) you can skip this section. However, it is instructive to see how easily a local Subversion repository can be created.

To create a new repository use `svnadmin`. The following line creates a new empty repository at `/extra/svn` on your local disk<sup>1</sup> You may create one repository per project, or use one repository for all projects. This is up to the repository administrator. See the Subversion manual for arguments.

```
> svnadmin create /extra/svn/
```

By convention, the following directories should be created in the repository (see 3):

- **trunk** contains the latest development version of the source code.
- **releases** contains all release-versions of the source code in subdirectories. These directories should only be checked in to
- **branches** contains other branches of the source code in subdirectories, for instance bug-fix branches for the release-versions.

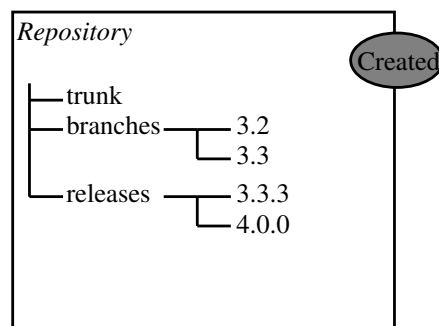


Figure 3. Creating a Subversion repository using `svnadmin create`.

One way of creating the initial repository directory structure is as follows:

---

<sup>1</sup>You should only create repositories on a local disk, not on an NFS-mounted disk. NFS-disks can cause problems, see the Subversion manual if you want to know why.

```
> mkdir tmp tmp/trunk tmp/releases tmp/branches
> svn import tmp file:///extra/svn/ -m Creating initial directories
> rm -rf tmp
```

To copy your files into the repository (assuming they are in **/work/my-files**) use:

```
> svn import ~/work/my-files file:///extra/svn/my-project/trunk -m 'Initial import'
```

The string *Initial report* is the log message. You may replace it with something else that describes the operation.

## 2.1.2 Creating a Working-copy Root-directory

The following command creates a new working-directory 'edge' and puts the latest source code from the Edge repository there:

```
> svn checkout https://costello/svn/edge/trunk edge
```

The `checkout` subcommand is only used when you want to create a new private working directory from scratch, you probably don't need to do this very often.

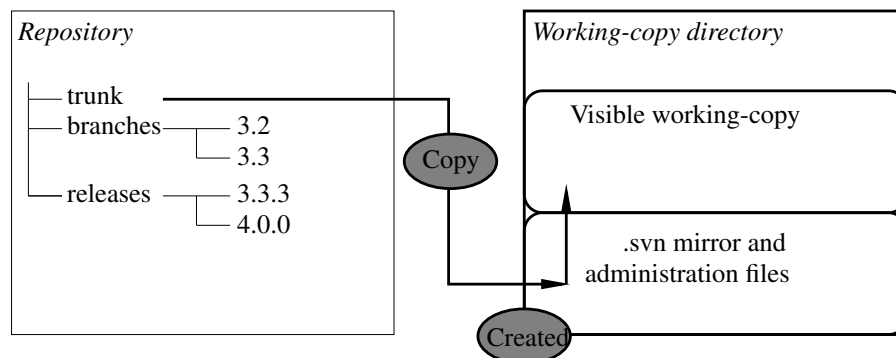


Figure 4. Creating a Subversion working-copy directory with the subcommand `checkout`.

Now you can edit the files in the **edge** directory. Changes in this directory do not affect other developers. Just don't touch the **.svn** directories, these contain administrative files that Subversion uses to keep track of your files.

---

## 2.2 The most Commonly used Subversion Subcommands

### 2.2.1 Getting Help

If you want a list of all Subversion subcommands, use `help`:

```
> svn help
usage: svn <subcommand> [options] [args]
Type "svn help <subcommand>" for help on a specific subcommand.
```

Most subcommands take file and/or directory arguments, recursing on the directories. If no arguments are supplied to such a command, it will recurse on the current directory (inclusive) by default.

Available subcommands:

```

add
blame (praise, annotate, ann)
cat
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
diff (di)
export
help (?, h)
import
info
list (ls)
log
merge
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolved
revert
status (stat, st)
switch (sw)
update (up)

```

To get more detailed help on a specific subcommand use `help cmd` (where `cmd` is the subcommand you want help for). This lists all available flags for that particular command. For example:

```

> svn help checkout
checkout (co): Check out a working copy from a repository.
usage: checkout URL... [PATH]

```

Note: If `PATH` is omitted, the basename of the URL will be used as the destination. If multiple URLs are given each will be checked out into a sub-directory of `PATH`, with the name of the sub-directory being the basename of the URL.

Valid options:

```

-r [--revision] arg      : ARG (some commands also take ARG1:ARG2 range)
                          A revision argument can be one of:
                          NUMBER      revision number
                          "{" DATE "}" revision at start of the date
                          "HEAD"      latest in repository
                          "BASE"      base rev of item's working copy
                          "COMMITTED" last commit at or before BASE
                          "PREV"      revision just before COMMITTED
-q [--quiet]             : print as little as possible
-N [--non-recursive]    : operate on single directory only
--username arg          : specify a username ARG
--password arg          : specify a password ARG
--no-auth-cache         : do not cache authentication tokens
--non-interactive       : do no interactive prompting
--config-dir arg       : read user configuration files from directory ARG

```

## 2.2.2 Examining and Comparing Working-copy with the Repository

The most commonly used subcommand reports what files you have modified in your working directory among other things:

```
> svn status
```

The `status` subcommand and many other subcommands works recursively on directories (by default the current working directory). If you want the status of a named directory or a single file you add the name of that file or directory. For instance:

```
> svn status mydirectory/myfile.f
```

There is an important extra flag to the status subcommand: `-u`. The `-u` flag is used to display pending updates from the repository. See 2.2.4. E.g.

```
> svn status -u -v mydirectory/myfile.f
```

To view the history of a file or directory, use the `log` subcommand. This shows who changed the file, when they did it and the log message they provided to describe their changes.

```
> svn log
```

The `diff` subcommand examines the difference between your working copy and the repository more in detail. The output is similar to the output of the Unix command `diff`. E.g.

```
> svn diff somefortranfile.f
```

## 2.2.3 Undoing changes to the working-copy

If you want to undo your changes to some file or directory in your working-copy there is a convenient way. Use `revert`:

```
> svn revert myfile.f
```

After the `revert` subcommand has finished the working-copy has the same state as when you did `update` or `checkout` the last time.

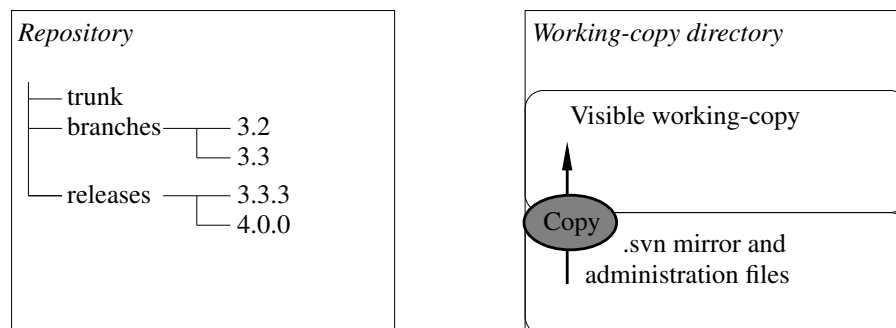


Figure 5. Subcommand `revert` copies the mirror to the working-copy.

The `revert` subcommand may also be applied to several files at once, or an entire directory. Because the `revert` subcommand reverts to the mirror-copy kept in the `.svn` directories it works even when there is no network connection to the repository, such as when working with a laptop.

## 2.2.4 Updating Working-copy with Changes Committed by Other Developers

The `update` subcommand updates your working-copy and the `.svn` directories with the latest changes to the repository. All files in your current working directory and below are updated. If you have also changed the contents of your working-copy the repository changes are merged with your changes.

```
> svn update
```

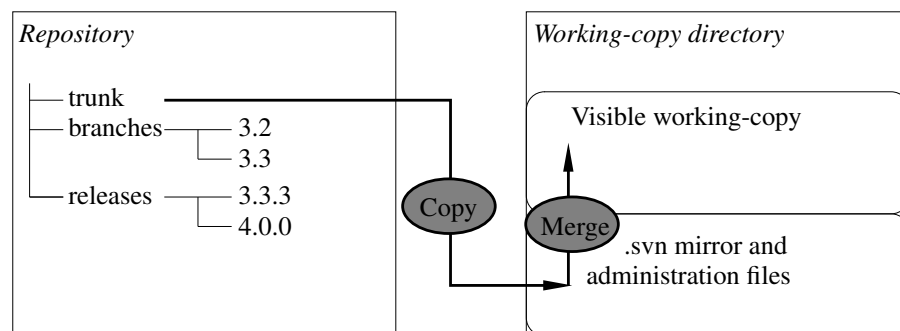


Figure 6. Subcommand `update` updates the mirror with the repository changes, and merges any changes with the to the working-copy.

Again, you may update a single file or directory by naming it:

```
> svn update mydirectory/myfile.f
```

Note that you can run `status -u` to get a list of all updates available in the repository. This is a good way to predict what update will do.

The `update` subcommand is also used when you want to retrieve an old version of some file(s) from the repository. E.g. to get the repository version of `myfile.f` as it was on the 6th of August 2006 you write:

```
> svn update -r '{20060806}' mydirectory/myfile.f
```

In some cases `update` will fail to merge changes to the repository with your local changes. If `update` or `status` lists a file with a 'C' in front this means that you need to merge the reported file(s) manually. See 2.4 for more information on how to do this.

## 2.2.5 Sending Modified Files back from Working-copy into the Repository

The `commit` subcommand updates the repository with your changes.

```
> svn commit -m 'New boundary condition sloppy-wall added'
```

This commits all files in the current directory and below recursively. You may also commit a single file or directory by naming it:

```
> svn commit mydirectory/somefile.f -m 'Fixed memory leak problem'
```

Note that `commit` will safely fail you don't have write-access to all corresponding files in the repository.

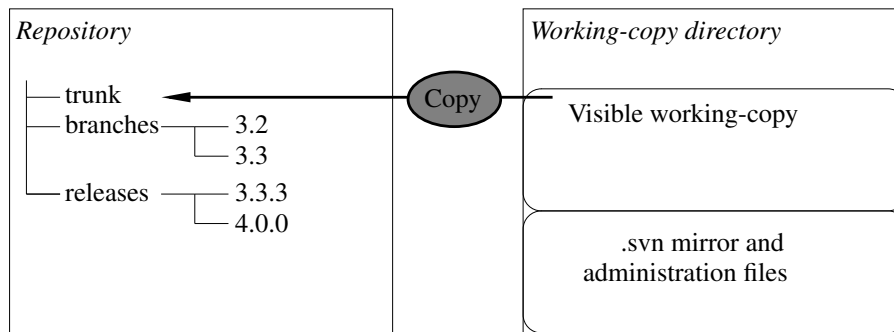


Figure 7. Subcommand `commit` copies your modified working-copy to the repository.

## 2.3 Other Useful Subcommands

### 2.3.1 Commands for Moving, Removing, and Adding files

If you add or delete files from your working copy you need to explicitly tell Subversion about this. The subcommands `add`, `move`, `copy` and `delete` are used for this. See `svn help xxx` for help on these subcommands. The `rename` subcommand is an alias for `move`. As usual, the repository is not changed when you apply these commands on your working-copy. When you do `svn commit` the repository changed.

### 2.3.2 Adding the Keywords to a new File

Subversion can substitute certain information directly into the files. This is done by putting keywords inside the file. When you add a new Edge file for version control, put the following three lines in the beginning of the file:

```
$LastChangedDate: 2007-03-14 10:13:16 +0100 (Wed, 14 Mar 2007) $
$LastChangedBy: crm $
$LastChangedRevision: 1069 $
```

You must add tell Subversion which keywords to look for. This is done by the following command:

```
> svn propset svn:keywords 'LastChangedBy LastChangedDate LastChangedRevision' filename
```

### 2.3.3 Looking Inside the Repository

Because the Subversion repository is stored in a database file on a remote server machine you cannot look at the repository files using Unix `ls` command. Instead you must use the Subversion subcommand `ls`. Note that many subversion subcommands that take a working-copy directory as argument may instead take a repository URL as argument. E.g. `svn copy`.

```
> svn ls https://costello/svn/edge/releases
...
/3.3.2
/3.3.3
/4.0.0
/4.1.0
```

### 2.3.4 Exporting Source Code

When you send the source code to a Edge-user you may want to create a tar-archive. If you create such an archive from a Subversion working-copy the archive will be unnecessarily large because it also contains many administrative files stored in special directories named **.svn**. To export your source code from a Subversion working-copy into a source code tree without the **.svn** directories there is a convenient command:

```
> svn export mydirectory edge-export
```

This creates a new directory **edge-export** with your source code in it<sup>2</sup>.

---

## 2.4 Manually Merging a Conflict

When `svn status` reports a file or directory with a 'C' in front of the filename this means that there is a conflict that Subversion cannot resolve. This could happen if you and some other developer have simultaneously changed the same lines in the same file. When the other developer commits his changes and you update your working-copy the conflict is discovered by Subversion and you must resolve it.

To resolve a conflict: simply edit the conflicting file in an editor and then save it.

To help you, Subversion has written both your modifications and the modifications from the repository into the file. In addition, Subversion has saved your file and the repository's file with in your working-copy directory (**.mine** and **.Rxxx** respectively).

---

## 2.5 Private Configuration File

After running `svn` for the first time you will have a directory in your home directory named **.subversion**. In this directory the file **.subversion/config** can be edited to customize Subversion behavior. The file is well commented and hopefully quite self-explaining. See [Collins-Sussman \*et al.\*](#) for more help.

---

## 2.6 More Help

The entire Subversion book/manual can be found in [Collins-Sussman \*et al.\*](#).

---

<sup>2</sup>The only thing `export` does is to copy all files recursively from your working-copy (or from a repository URL) into a new directory, omitting all **.svn** directories. You could of course quite easily do this yourself using Unix-shell commands, the `export` subversion command exists only for convenience.



## 3 Some examples

Some common examples of Subversion subcommands<sup>3</sup>.

Create a new working-copy root-directory.

```
svn checkout https://costello/svn/edge/trunk edge
```

List your modifications to the working-copy.

```
svn status somedirectory
```

List modifications to the repository and predict what 'update' would do.

```
svn status -u somedirectory
```

Update your working-copy with modifications to the repository.

```
svn update somefile.f90
```

Update the repository with modifications to your working-copy of a file.

```
svn commit -m 'Your log-message here' somefile.f90
```

Get an old version of a file from the repository.

```
svn update -r '{20060806}' somefile.f90
```

Undo modifications to the working-copy.

```
svn revert somefile.f90
```

List history of a file.

```
svn log somefile.f90
```

List details of your modifications to a working-copy file.

```
svn diff somefile.f90
```

List differences between your working-copy and an old repository version of a file.

```
svn diff -r '{20060806}' somefile.f90
```

List differences between two old repository versions of a file.

<sup>3</sup>Omitting **somefile.f** or **somedirectory** in these examples defaults to the current working directory.

```
svn diff -r '{20060806}:{20060730}' somefile.f90
```

**List files in the repository.**

```
svn ls https://costello/svn/edge/releases
```

**Rename a file/directory.**

```
svn mv somefile.f90 newname.f90
```

**Put a new file/directory under version control.**

```
svn add somefile.f90
```

**Remove a file/directory.**

```
svn delete somefile.f90
```

**Export a source code tree for distribution to a user/customer.**

```
svn export https://costello/svn/edge/releases/4.1.0 4.1.0
```

**HELP!!!**

```
svn help
```

## References

COLLINS-SUSSMAN, B., FITZPATRICK, B. W. & PILATO, C. M. Version Control with Subversion. <http://svnbook.red-bean.com>.

EDGE HOMEPAGE <http://www.edge.foi.se>.

ELIASSON, P. 2001 EDGE, a Navier–Stokes Solver for Unstructured Grids. Scientific Report FOI-R--0298--SE. Computational Aerodynamics Department, Aeronautics Division, FOI.

ELIASSON, P. 2002 EDGE, a Navier–Stokes solver for unstructured grids. In *Proceedings of Finite Volumes for Complex Applications III*, pp. 527–534.





FOI is an assignment-based authority under the Ministry of Defence. The core activities are research, method and technology development, as well as studies for the use of defence and security. The organization employs around 1350 people of whom around 950 are researchers. This makes FOI the largest research institute in Sweden. FOI provides its customers with leading expertise in a large number of fields such as security-policy studies and analyses in defence and security, assessment of different types of threats, systems for control and management of crises, protection against and management of hazardous substances, IT-security and the potential of new sensors.



FOI  
Swedish Defence Research Agency  
SE-164 90 STOCKHOLM

Tel: + 46 8 555 03000  
Fax: + 46 8 555 03100

[www.foi.se](http://www.foi.se)