

# Version control

Version control is a powerful tool for many kinds of work done over a period of time, including writing papers and theses as well as writing code. This session gives an introduction to a version control system called subversion.

Some reasons why using version control for writing programs is a good idea are:

- efficient development practices. It is very easy to try things out and experiment as undoing any unwanted changes is a single command (rather than undoing all changes by hand).
- reproducibility. Version control makes it possible to easily roll-back to previous versions of the program and so re-run calculations with exactly the same program originally used. Of course, results should not change as a program evolves (assuming that the program was correct in the first place). If a bug does enter the codebase then version control also makes it easy to find out what results are affected.
- makes it easy to collaborate with colleagues and work on different computers by communicating via the version control system.

There are two types of version control systems: centralised version control systems (CVCSs) and the (more recent) distributed version control systems (DVCSs). A CVCS does all communication via a central server which stores the repository whereas a DVCS is completely distributed and each user has a full copy of the repository. RCS, CVS and subversion all follow the CVCS model. (RCS and CVS are both very old: don't use unless you absolutely have to!) Recently many DVCSs have been written, including monotone, mercurial, bazaar and git. There are advantages and disadvantages to both approaches. We will use subversion throughout this course as it's widely used and is easier to learn than a DVCS.

## subversion

Some terminology used in subversion:

### repository

The repository is located on the server and stores all the files and directories under source code management as well as their histories. Users read from and write to the repository.

### working copy

The working copy is a local copy of the repository. It does not contain the full history of the repository. Changes are made in the working copy before being sent back (*committed*) to the repository.

### revision

A revision is a snapshot of what the repository looked like in time. Subversion labels the revisions by an integer, which increases as a new revision is added to the repository. A new revision is created each time a set of changes are committed to the repository. Revisions in subversion are global: files all have the same revision id, so committing a change to one file will cause the revision id of all files in the repository to increase. In this way the exact state of the repository can be obtained from one number.

### HEAD

HEAD is the name of the latest revision in the repository. It can be used in subversion rather than the latest revision number.

### BASE

BASE is the name of the revision upon which the working copy is based. (BASE can be different from HEAD if another user committed to the repository after the most recent update to the working copy.)

### trunk

The trunk is the directory in the repository where the main development of a project occurs. A typical layout of a repository is:

```
project
  /trunk
  /branches
    /branch1
    /branch2
```

The project files are stored in the trunk. The branches are used in some situations for convenience: we shall not need to use branches in this course but they are discussed in a little more detail later on.

If subversion gives an error message about not being able to find an external editor then you need to set this in the subversion config file (likely on OSX, less likely under linux). To do this open the subversion configuration file (`~/subversion/config`) and find the line which start `"# editor-cmd ="`. Change the line to `"editor-cmd = vim"` (or the editor of your choice).

## Commands

All subversion comands start with `svn` followed by a subcommand. Whilst `svn` has a brief man page, the main source of help is provided as a subcommand:

```
svn help          # General help. Lists all subcommands.
svn help checkout # Help on the checkout subcommand.
```

There are many subcommands. Fortunately only a few are needed on a regular basis. Some options to the subcommands are useful--please see the help pages for more details.

### **svn checkout**

Check out a copy of the repository from a server. You need to specify the URL of the repository. You can also (optionally) specify the directory you want to check out into:

```
svn checkout URL directory_name
```

### **svn update**

Update the working copy: get new commits from the server.

### **svn commit**

Commit changes in the working copy to the server. Requires a log message to be provided, which details the changes made. The log message can be provided using the `-m` option:

```
svn commit -m "Changes made are:...."
```

If a log message isn't provided on the command line then subversion opens up a text editor for you to enter the log message. Enter the log message and save and quit to make the commit.

### **svn diff**

Compare changes between two different versions of the repository. By default `svn diff` compares the uncommitted changes to the BASE of the repository (i.e. the state of the repository before you started making changes). The format of a diff output is not always the easiest to read. A graphical diff program, whilst slower, is often clearer. One graphical diff program is `meld`. `svn diff` will use `meld` rather than print to the terminal if the `diff-cmd` is set to `meld` in the subversion config file.

### **svn status**

Show the status of the working copy: what files have been changed, added, removed or are unknown to subversion. There are shown by the symbols M, A, D and ? respectively next to the relevant filename . (Other statuses are also used, but these are the most common.)

### **svn log**

Show the log messages.

### **svn revert**

Revert the working copy to the BASE version. This undoes all your changes. It can be used to revert changes in specific files (specified on the command line) or all changes in the directory:

```
svn revert filename1 filename2 # revert changes to files filename1 and filename2
svn revert * # revert all changes in the current directory.
```

Subversion can only manage the files it knows about, so you need to tell it about files you want it to manage (or stop managing).

### **svn add**

Add files to the repository.

```
svn add file_to_be_added
```

### **svn remove**

Delete files to the repository.

```
svn remove file_to_be_removed
```

This also deletes the files from the working copy. Analogous to the normal *rm* command.

### **svn move**

Move a file, e.g. to rename the file or to move it to a different directory within the repository. Analogous to the normal *mv* command.

An advanced topic (which we won't go into here) is that of branching and merging. Earlier we discussed a project having a "trunk" i.e. the main development path for the project. A branch is used for a line of development that proceeds independently of the trunk (but was identical to the trunk at some point in the past): for instance to perform a major restructuring of a program that might involve stopping the program from working correctly for some time whilst allowing for bugs to be fixed and colleagues to continue working on the trunk. Once the development work in the branch has been completed it can be merged back into the main trunk of the repository.

## **Local setup**

The CMTH/TYC group hosts a subversion server. You all have a repository setup on the server. The address of your individual repository is <https://tyc-svn.cmth.ph.ic.ac.uk/users/username> where username is your college username. For this session you all have read and write access to each others' repositories. Afterwards the only people with access to your repository will be yourself and those involved in assessing the MTMCC course.

The same repository will be used throughout the course instead of having one repository for each project. This makes the server administration substantially easier but has the downside that the revision id is a repository property rather than a project property.

To create a place on the server for a new project use the *svn mkdir* command, e.g.:

```
svn mkdir https://tyc-svn.cmth.ph.ic.ac.uk/users/username/project_name/trunk
```

The project can now be checked out and files added to it as usual:

```
svn checkout https://tyc-svn.cmth.ph.ic.ac.uk/users/username/project_name/trunk project_name
cd !$
touch f1 f2 f3
svn add f{1,2,3}
svn commit
```

Screenshots from such a series of commands follow. You must replace `jspencer` with your own username.

1. The first access of the repository requires authentication, following which your password is cached (possibly in plain text) and so does not need to be entered again:

```
Terminal - jss43@pauling:~  
File Edit View Terminal Go Help  
jss43@pauling:~$ svn ls https://tyc-svn.cmth.ph.ic.ac.uk/users/jspencer  
Authentication realm: <https://tyc-svn.cmth.ph.ic.ac.uk:443> Subversion Repository  
Password for 'jss43':  
Authentication realm: <https://tyc-svn.cmth.ph.ic.ac.uk:443> Subversion Repository  
Username: jspencer  
Password for 'jspencer':
```

## 2. Running

```
svn mkdir https://tyc-svn.cmth.ph.ic.ac.uk/users/jspencer/svn_example
```

causes an editor (e.g. vim) to be launched. The commit message detailing the change needs to be entered:

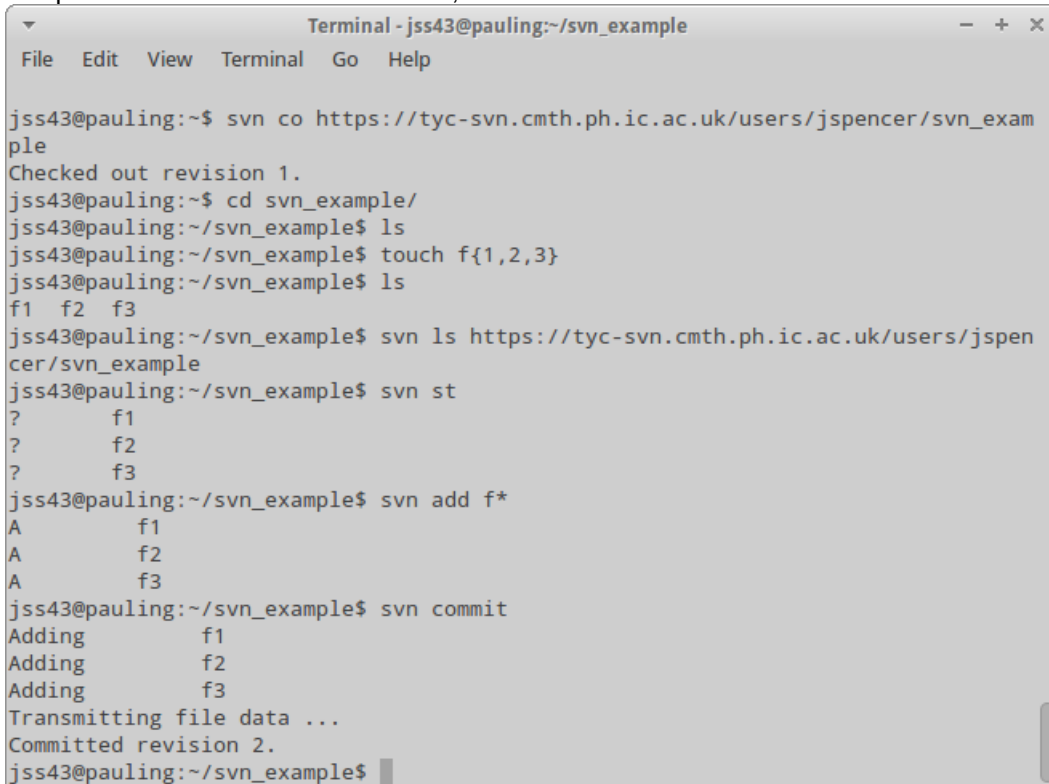
A screenshot of a terminal window titled "Terminal - svn-commit.3.tmp + (~) - VIM". The menu bar includes File, Edit, View, Terminal, Go, and Help. The main text area contains the following content:  

```
Create a directory for experimenting with subversion.  
--This line, and those below, will be ignored--  
  
A    https://tyc-svn.cmth.ph.ic.ac.uk/users/jspencer/svn_example  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

  
At the bottom left, it says "3 fewer lines". At the bottom right, there are two status indicators: "2.1" and "All".

Once the commit message is saved and vim is exited, svn creates the directory in the repository on the server and creates a commit associated with that action.

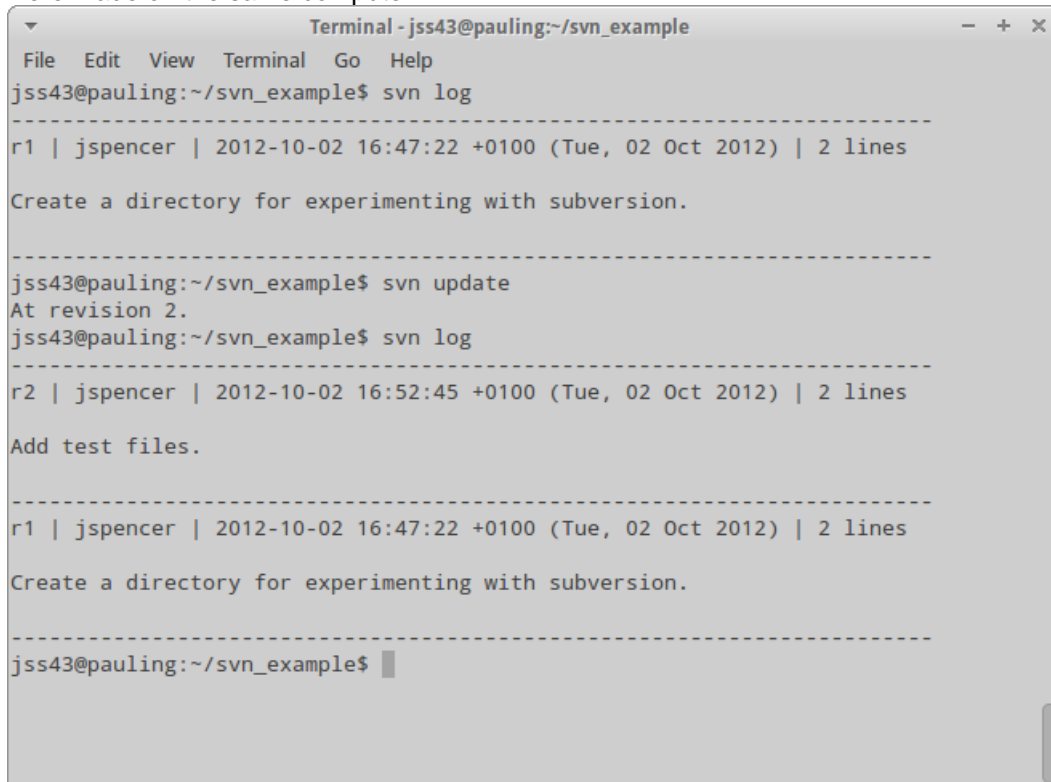
3. Once the directory has been created on the server, it can be checked out (downloaded) onto your computer and files can then be added, deleted and modified:

A terminal window titled "Terminal - jss43@pauling:~/svn\_example" with a menu bar (File, Edit, View, Terminal, Go, Help). The terminal shows the following commands and output:

```
jss43@pauling:~$ svn co https://tyc-svn.cmth.ph.ic.ac.uk/users/jspencer/svn_example
Checked out revision 1.
jss43@pauling:~$ cd svn_example/
jss43@pauling:~/svn_example$ ls
jss43@pauling:~/svn_example$ touch f{1,2,3}
jss43@pauling:~/svn_example$ ls
f1 f2 f3
jss43@pauling:~/svn_example$ svn ls https://tyc-svn.cmth.ph.ic.ac.uk/users/jspencer/svn_example
jss43@pauling:~/svn_example$ svn st
?      f1
?      f2
?      f3
jss43@pauling:~/svn_example$ svn add f*
A      f1
A      f2
A      f3
jss43@pauling:~/svn_example$ svn commit
Adding      f1
Adding      f2
Adding      f3
Transmitting file data ...
Committed revision 2.
jss43@pauling:~/svn_example$
```

Note that changes are only sent to the server when they are **committed** to the repository.

4. The repository log shows a list of the changes made. Note that *svn update* is required to get information about commits made since *svn checkout* or *svn update* was run, even if those commits were made on the same computer.

A terminal window titled "Terminal - jss43@pauling:~/svn\_example" with standard window controls. The terminal shows the following sequence of commands and output:

```
jss43@pauling:~/svn_example$ svn log
-----
r1 | jspencer | 2012-10-02 16:47:22 +0100 (Tue, 02 Oct 2012) | 2 lines

Create a directory for experimenting with subversion.

-----
jss43@pauling:~/svn_example$ svn update
At revision 2.
jss43@pauling:~/svn_example$ svn log
-----
r2 | jspencer | 2012-10-02 16:52:45 +0100 (Tue, 02 Oct 2012) | 2 lines

Add test files.

-----
r1 | jspencer | 2012-10-02 16:47:22 +0100 (Tue, 02 Oct 2012) | 2 lines

Create a directory for experimenting with subversion.

-----
jss43@pauling:~/svn_example$
```

## Resources

The subversion project has written an excellent book on using svn (published by O'Reilly) which is updated regularly and is free online: <http://svnbook.red-bean.com/>.

## Tasks

1. Create a directory in your svn repository.
2. Create and add files to your svn repository.
3. Perform the following tasks (and any others you can think of):
  - a. Make and commit changes.
  - b. Make changes and inspect them using the diff command.
  - c. Revert your changes.
  - d. Compare the current version to an earlier version.
  - e. Inspect the commit log.
4. Checkout your repository into another directory. Make a commit from this repository and update the first directory.
5. [optional] In pairs: check out each other's repository. What happens if you both edit the same file in different places and then commit? What about if the edits are to the same part of the file? Find out how to resolve this.