/open_source/ version control

Your essential CD
All the files you'll require for this tutorial can be found on this issue's CD.

Where would we be without the ability to undo our mistakes? Gavin Montague shows how your projects can benefit from an undo feature that puts Ctrl-Z to shame

Knowledge needed A little HTML and CSS

Requires Free to download Subversion tools

Project time One hour

Subversion is a free, open source tool produced by CollabNet Inc and distributed from subversion.tigris.org. It's used to manage the source for Apache, Python, Django and literally thousands of other projects, large and small. In fact, I'm using Subversion to track the changes to this very article as I write it.

You can think of Subversion as working like a file server. A nominated directory, called the repository, holds a master copy of the project. Anyone making alterations to the project must first check out a copy to work on. With their alterations complete, the changes are committed back to the repository where they become visible to other collaborators. Nothing fancy so far, but where it gets interesting is that Subversion doesn't just store the current state of the files; it records every commit along with a note describing what the change was and who made it.

If everyone can edit their working copies and commit back to the repository, what happens when the changes are in conflict? If Steve and Bill are both editing index.html, what happens when Steve commits his changes and Bill tries to commit his changes a few minutes later?

Subversion has a good answer to this: it will refuse to add any of Bill's changes to the repository until Steve's edits have been merged into his working copy. If the changes don't clash (say Steve was editing the document head and Bill was making changes to the text in the footer) Subversion will silently add Steve's changes into Bill's working copy. If any changes overlap, Subversion will ask for Bill's help in merging Steve's alterations. After bringing his working copy up to date, Bill is free to commit to the repository.

Subversion assigns each commit an incremental stamp. The first import is assigned number 1; the next set of changes make the project into revision 2, and so on. This makes it easy to check whether your working copy is up to date. If the repository is at revision 15 and your working copy is on revision 12, then three commits must have been made since you last updated. You can use the commit numbers to ask for older copies of files: 'Show me how the style sheet looked at revision 5'.

Alternatively, you can use dates to phrase the question: 'Show me how the homepage looked two weeks ago.'

This all boils down to a four-step workflow:

1. Check out a working copy from the repository.
2. Make your alterations; add, delete and edit files.
3. Merge recent changes from the repository into your working copy.
4. Commit your changes back to the repository.

We've covered a lot of ground in this section. Don't worry if it all seems a bit abstract.

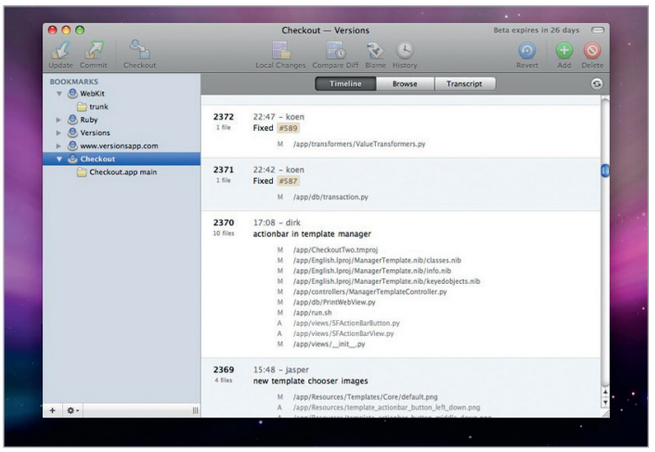
There's a glossary of the most important and commonly used terms on the facing page (top right) and we're going to run through some of the practical benefits Subversion gives us by working on a hot Web 3.0 dot-com. The files you'll need can be found on this issue's disc.

Expert tip Don't save up your commits

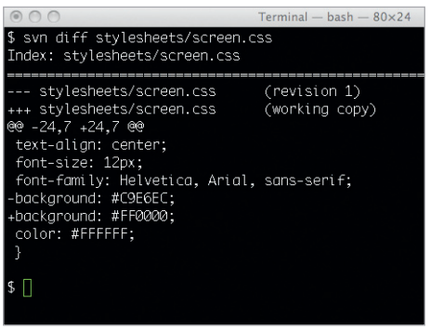
Subversion only takes snapshots of your project when you commit back to the repository. Don't work for a whole day and then bundle up all your changes in a single commit just before you go home. Commit each time you complete a discrete task and add a descriptive comment that will be meaningful at a later date. 'Updated descriptions on product pages with correct SKU code' is an example of a good commit message. 'Fixed some bugs and added new stuff' is not.

Installing Subversion

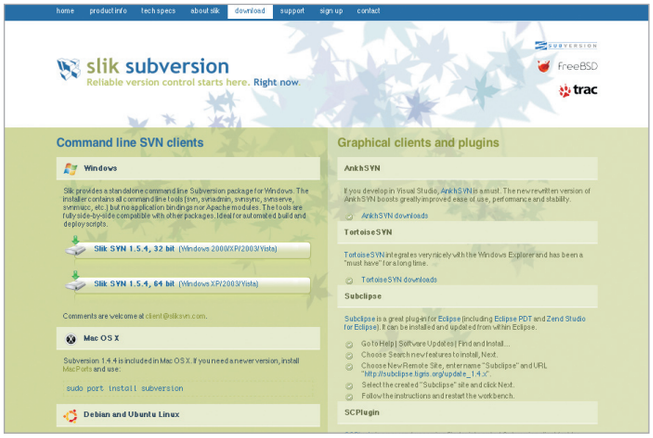
Subversion is available for Windows, OS X and Linux, so everyone will be able to play along. The bad news is that we'll be using the command line interface in this tutorial. You might not be familiar with the command line, which we'll call the 'prompt' from now on, but it's the best way of learning the underlying



Left Open source version control system Subversion enables you to keep track of changes to your project. It records every commit along with a note describing what the change was and who made it



Right You can use the diff command to get more details about the changes that have been made to a particular file



Slik service If you're using Subversion on Windows you can install the Command Line Client SVN tools from Slik (www.sliksvn.com/en/download)

concepts of version control. Once you're happy with what's happening, you can graduate to any of the graphical front-ends.

For Windows, install the Command Line Client SVN tools from Slik (www.sliksvn.com/en/download). Once it's installed, you can bring up the DOS prompt by clicking on the **Start** menu, selecting **Run** and then typing **cmd** into the **Run this program** field.

You can ask for older copies of files: 'Show me how the style sheet looked at revision 5'

Subversion has come pre-installed with OS X since 10.4. For earlier versions, Martin Ott maintains a Subversion package (homepage.mac.com/martinott/). Once installed, open the **Terminal** prompt by launching **Applications > Utilities > Terminal.app**.

For Linux, refer to the instructions for your package manager. It's likely that you'll be able to install Subversion easily.

In the code samples below, the lines of input are prefixed with a $ sign. Depending on your system, the $ sign might be replaced with your username or the path to the current directory.

Lines that don't have a dollar show the generated output. For example, to confirm Subversion is installed, type the following at the prompt:

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

Here svn is your line of input. Pressing enter will generate the output **Type 'svn help' for usage**.

Detailed help on all the commands we use below can be accessed by appending the command name after **help**. For example:

```
$ svn help checkout
```

Expert tip Graphical interfaces to svn

Although we've been using the command line in this tutorial, Subversion also supports a range of graphical front-ends that enable you to manage repositories and working copies from within a normal desktop application. On a Mac you might like to try Versions (versionsapp.com/) or svnX (www.lachoseinteractive.net/en/community/subversion/svnx/features/) while on Windows, TortoiseSVN (tortoisesvn.tigris.org/) integrates directly into your File Explorer.

Glossary

Like all software, Subversion comes with its own terminology: here's our guide

Repository The database that retains the master copy, and history, of your project.
Check out To connect to the repository and make a local working copy of the project
Working copy The local version of the project in which edits are made
Commit To write changes from your working copy back to the repository
Update Pulling changes from the repository into your working copy
Merge When changes in your working copy can be silently combined with changes in the repository
Conflict Two changes between your working copy and the repository that Subversion cannot resolve and required manual intervention.

Setting up your repository

Create a new folder on your desktop called **svn_tutorial** and move the prompt into it with the **cd** (Change Directory) command and then create an empty repository:

```
$ cd Desktop/svn_tutorial
$ svnadmin create repository
```

You'll now see the folder repository appear inside **svn_tutorial**. This is the database that will store the master copy of our project; you'll never use these files directly, but it's good to know they exist.

Copy the folder **emailr** from the disc into **svn_tutorial**. We'll now import it into the repository for Subversion to track. You'll need to give the full path to the repository folder so the command will vary between systems. On Windows, it'll look like this:

```
$ svn import emailr "file:///C:/Documents and Settings/<your user name>/
Desktop/svn_tutorial/repository"  -m "Imported the starter project into the
repository"
```

And on a Mac the command will be:

```
$ svn import emailr "file:///Users/<your user name>/Desktop/svn_tutorial/
repository"  -m "Imported the starter project into the repository"
```

Subversion will reply by listing the files added to the repository. The final portion of the command, from -m onwards, is the commit comment. We'll look at how to access these later.

Our repository is now set up and we can check out a working copy called **dev_copy**. Windows users do this with the command:

```
$ svn checkout "file:///C:/Documents and Settings/<your user name>/
Desktop/svn_tutorial/repository"  dev_copy
```

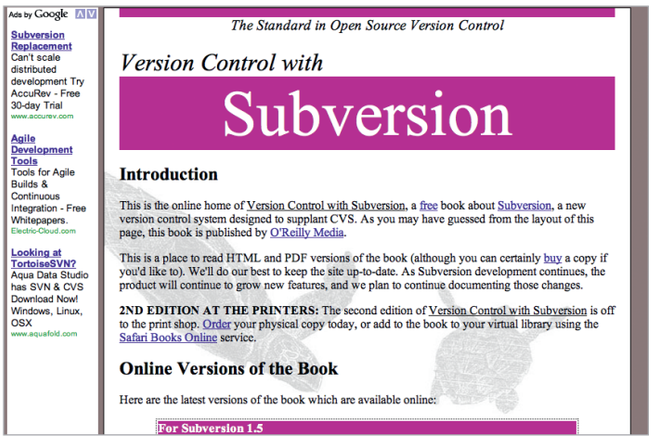
And for the Mac users:

```
$ svn checkout "file:///Users/<your user name>/Desktop/svn_tutorial/
repository"  dev_copy
```

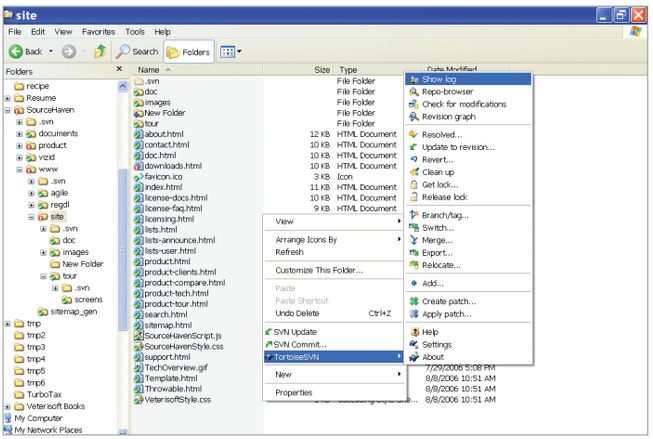
A folder called **dev_copy** will appear. We won't have to give the long path to the repository again. Our working copy will automatically find its repository. We also no longer need the original copy of **emailr**, so that can be deleted.

It's worth noting that although we're keeping our repository and our working copy on the same computer, Subversion can use a whole range of network protocols to communicate with remote repositories via ssh, http and its own svn protocol. Keeping your repository off-site can give you a higher level of security and allow for easier collaboration.

>>



Learn online If you want to find out more about using Subversion, the book *Version Control with Subversion* is available to download for free at svnbook.red-bean.com



Tortoise time Subversion supports a range of graphical front-ends, such as TortoiseSVN (tortoisesvn.tigris.org), which integrates directly into your File Explorer in Windows

>> Your first edit

Time to start using Subversion on our site: a cutting edge, paradigm-shifting Web 3.0 email tool. We'll only be dealing with the HTML and JavaScript parts in this version.

The first thing we'll do is update the background colour of the page in **screen.css** to match the new company branding – replace **#C9E6EC** with **#FF0000**.

After saving the file, move the prompt into **dev_copy with cd** and ask Subversion to look over the working copy for changes.

```
$ cd dev_copy
$ svn status.
m  stylesheets/screen.css
```

The **m** indicates that the file has been modified. We'll see other possible markers representing the state of files later. To get more details on the changes to a particular file, use the diff command:

```
$ svn diff screen.css
Index: stylesheets/screen.css
===================================================================
--- stylesheets/screen.css  (revision 1)
+++ stylesheets/screen.css  (working copy)
@@ -24,7 +24,7 @@ text-align: center;
font-size: 12px;
font-family: Helvetica, Arial, sans-serif;
-background: #c9e6ec;
+background: #FF0000;
color: #FFFFFF;
}
```

This format is called 'unified diff' and is often used by developers to exchange patches of code via email, IM and the web. There are tools for automatically applying patches, but the format is rather readable and will serve us well here. Lines beginning with a + show insertions. Lines with a - show deletions. As you can see, we've removed the line containing the old background colour and inserted a line containing the new one.

```
$ svn commit -m 'Changed the background colour of the page to match our new corporate identity'
```

The working copy sends its changes to the repository to be committed. Run **svn status** and you'll notice Subversion no longer reports that **screen.css** is modified – the changes have been written to the repository and the file is now up to date.

It's good practice to call update after a commit. This will pull in any changes made by other collaborators.

```
$ svn update
```

Adding and deleting files

Let's add a new page to our site: create **thanks.html** by copying **index.html** and altering the copy.

Once the file has been created in **dev_copy**, run **svn status**:

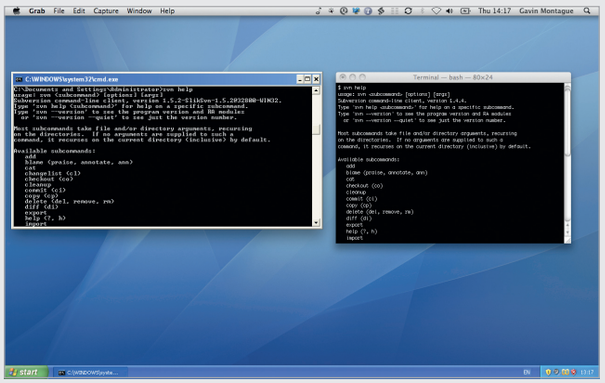
```
$ svn status
?  thanks.html
```

The question mark indicates that Subversion doesn't currently track this file. There are several cases where you wouldn't want all the files in the working copy to be added to the repository (log files, temporary file or cached data, for example) so Subversion won't track a file unless we explicitly ask:

```
$ svn add thanks.html
$ svn status
A  thanks.html
```

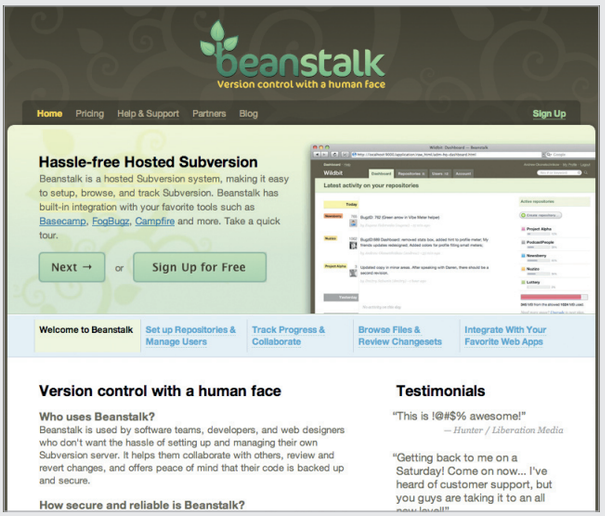
The 'A' flag shows that Subversion will now add **thanks.html** to the repository in the next commit.

Interfaces The prompt is your friend



The Windows DOS prompt (left) and OS X bash shell (right) might not be the prettiest interfaces in the world but there are a lot of powerful tools that can be used through them.

Hosted system Remote repositories



It's always a good idea to host your repositories on a server that's backed-up and properly maintained. But that can be such a hassle. The good folk at Beanstalk provide a range of hosted Subversion packages that remove all the faffing. There's a free package available that's great for taking your first steps with remote repositories.

```
$ svn commit -m 'Added the thanks page with draft copy'
```

You can edit multiple files between each commit. Add a page called **copyright.html** to **dev_copy** and then put a link to it in the footer of both **index.html** and **thanks.html**.

```
$ svn status
?  copyright.html
M  index.html
M  thanks.html
```

Next add **copyright.html** to the working copy's index and then commit all the changes.

```
$ svn add copyright.html
$ svn commit -m 'Added copyright notice to all pages'
```

Try adding a few more pages to the repository. You can then delete any of them from the repository with the command:

```
$ svn delete example.html
D  example.html
```

At the next commit, the files marked as 'D' will be removed from the repository and your working copy.

Make some more edits to the files within the site. Try adding some images to the site. You'll find Subversion handles binary files as well as text: images, PDFs, Word files, in fact almost anything can be kept in a repository.

After each set of changes, commit to the repository.

Rolling back changes

Fast-forward a few weeks. Our fictional client calls to say that they've decided to stick with the old colour scheme and could we go back to the original background? Without source control, we'd be searching our hard drives for an old copy of the project and we might find one, or we might not. Not a problem with Subversion.

After bringing our project up to date with **svn update**, inspect the log for edits to our style sheet.

```
$ svn log stylesheets/screen.css
------------------------------------------------------------
r2 | gavin | 2008-09-17 21:25:26 +0100 (Wed, 17 Sep 2008) | 1 line
changed the page background colour
------------------------------------------------------------
r1 | gavin | 2008-09-17 17:53:14 +0100 (Wed, 17 Sep 2008) | 1 line
initial import
------------------------------------------------------------
```

Depending on other changes you've made to **screen.css**, the output of this command may vary, but you'll notice that only commits that have modified this file are listed.

Should you want to see the commit log for the whole project, run the command without giving a filename.

As you can see, the log lists the time of each commit and the name of the user responsible.

We can see the background colour was changed at revision 2. Use the diff command to inspect the change in more detail:

```
svn diff -r 1:2 stylesheets/screen.css
Index: stylesheets/screen.css
===================================================================
--- stylesheets/screen.css      (revision 1)
+++ stylesheets/screen.css      (revision 2)
@@ -24,7 +24,7 @@
text-align: center;
font-size: 12px;
font-family: Helvetica, Arial, sans-serif;
-background: #C9E6EC;
+background: #FF0000;
color: #FFFFFF;
}
```

This command reads as 'Show what changed in screen.css between revisions 1 and 2'. As you can see, our original edit is returned. We could copy/paste this change back into our working copy, but we'll be smarter and have Subversion merge the change for us:

```
svn merge -r 2:1 stylesheets/screen.css
--- Reverse-merging r2 into 'stylesheets\screen.css':
U    stylesheets\screen.css
```

Your style sheet will now contain the original background colour. You can use the merge command to pull any change, or group of changes, back into your working copy; you need never lose work again.

Where to go from here

We've only covered the basics of Subversion here. We haven't looked at, for example, how it can be used across a network, to keep a secure off-site backup of your code; how it can help teams work simultaneously on the same code without fear of overwriting each other's code; or how to use branching to help develop experimental features without interfering the ongoing maintenance of your production code.

If you'd like to learn more, one of the best resources is the book *Version Control with Subversion*, published by O'Reilly but also freely available to download at svnbook.red-bean.com.

About the author

Name Gavin Montague
Site leftbrained.co.uk
Areas of expertise Ruby, PHP, front-end development and Cocoa
Which (if any) football team do you support? None, I'm waiting for fox hunting to make a comeback