A brief Subversion readme for CS 148

Stu Black

Fri Sep 25 14:49:31 2009

Contents

1 Introduction **2**

2 Cheat sheet **2**
2.1 Concepts . . . 2
2.2 Authentication . . . 3
2.3 The repository structure . . . 3
2.4 Running Subversion commands . . . 4
2.5 Quickstart . . . 4
2.5.1 Browsing the repository . . . 4
2.5.2 Checking out . . . 4
2.5.3 Updating . . . 5
2.5.4 Examining the state of your working copy . . . 5
2.5.5 Manipulating files in your working copy . . . 5
2.5.6 Committing your changes . . . 6
2.5.7 Resolving conflicts . . . 6
2.5.8 Merging from branches . . . 6
2.5.9 Last notes . . . 7

3 For advanced users **7**

1 Introduction

Version control systems are a class of software that keeps track of revisions to files as you make them. One popular open source version control system is Subversion[1]. For this semester of CS 148, we are requiring you to use Subversion to submit the code for your finished assignments.

From `man svn`:

> Subversion is a version control system, which allows you to keep old versions of files and directories (usually source code), keep a log of who, when, and why changes occurred, etc., like CVS, RCS or SCCS. Subversion keeps a single copy of the master sources. This copy is called the source "repository"; it contains all the information to permit extracting previous versions of those files at any time.

In addition to helping you keep track of the revisions that you make for your source code, Subversion facilitates collaboration on your source code. Anyone in your project group can check out code, modify it, and commit their changes to your common repository. You can then update the code you're working on with their changes without having to trade whole files back and forth. Additionally, your code can be checked out directly to one of the class robots, allow for greater portability when switching between robots necessitates.

We hope that those of you who have not used a version control system before will learn how to use Subversion to guide your group's workflow, while those of you who have will be able to start working with Subversion right away. There is a very comprehensive book freely available at `http://svnbook.red-bean.com/` which gives extensive documentation on Subversion. There are numerous quick introductions to Subversion on the Web. What follows is a cheat sheet to help you get started.

2 Cheat sheet

2.1 Concepts

Subversion may be thought of as a file versioning system that sits on top of your filesystem. With it, you can take a set of files, apply changes to them, commit the new changes, and continue working. At any point in time, you can revisit your files as they were at an earlier revision. What follows is a brief glossary of terms.

Repository

> a central location where your project, in all its version-controlled glory, resides.

[1] http://subversion.tigris.org/

Working copy

a file tree you have checked out from the repository.

Revision

every time you commit code to your repository, a new revision is created and given a number. You can refer to the repository as it was at a specific commit using that commit's revision number.

Trunk

a file tree that contains your main body of code.

Tags

a tag is a special name that you give to some snapshot of code in the repository. In Subversion, this is a copy of some file tree that you give a special name in the top-level directory `tags`.

Branches

a branch is a section of material that is being worked on in separately from other code in order to isolate one set of changes from others. In Subversion, this is a copy of some file tree that you have given a special name in the top-level directory `branches`.

2.2 Authentication

We will give each student a unique username/password pair that provides access to your group's section of the repository. There may be many like it, but this one is yours. The usual rules about not sharing passwords with friends apply. If you inadvertently grant another group access to your group's work and they copy off of it, you are liable.

When accessing the repository for the first time, you will be prompted for a username/password pair. To specify a particular username to `svn`, use the `-username` argument. We have configured Subversion on the EeePCs not to cache your usernames or passwords, but be on the lookout that you do not accidentally store them on elsewhere (such as on a shared laptop).

2.3 The repository structure

We have set up Subversion repositories for each of your groups. They are located on the CS network at `svn://foxwood/<your group name>`. At their top level, they have the directories `trunk`, `tags`, and `branches`. `trunk` should be used for whatever you're currently working on, while `tags` and `branches` hold tags and branches of your work, respectively.

We have already put some skeleton code into the branches `asgn1_skel` and `asgn3_skel`. You should use it to get started on these assignments. See "2.5.7," below, for instructions on how to do this.

2.4 Running Subversion commands

The command-line Subversion client is `svn`. There are stand-alone GUI applications (like RapidSVN[2], Subcommander[3], or TortoiseSVN[4]) and plugins for most popular open-source IDEs, but this guide will describe the command-line interface.

Invocations of the client use this syntax:

```
svn <action> [arguments]
```

For any action, you can run `svn help <action>` to get usage for that action.

2.5 Quickstart

2.5.1 Browsing the repository

You can examine the contents of the repository before checking anything out. Try the commands:

```
svn ls svn://foxwood/groupname/ svn ls svn://foxwood/groupname/trunk/
svn ls svn://foxwood/groupname/branches/ svn ls svn://foxwood/groupname/tags/
```

`svn` has analogues to the Unix `ls`, `cat`, `mv`, `rm`, and `cp` commands that can all work directly on your group's repository. As you start out, you should see using `svn ls` that you have a few branches in `branches` and nothing else in any of the other top-level directories. We have prepared these directories for you. As you commit material to your repository, you may examine and manipulate it using these commands.

2.5.2 Checking out

To get started, you should check out a new working copy. You only need to do this once, unless for some reason you wish to have multiple working copies of your code. To check out your trunk directory, run:

```
svn checkout svn://foxwood/groupname/trunk <dirname>
```

When you are in the robotics lab, you can check code out directly on the robots using the same command. If you have another computer on the AIBO network (such as a laptop of your own), you can use foxwood's IP address, `10.100.0.201`, and use the command:

```
svn checkout svn://10.100.0.201/groupname/trunk <dirname>
```

One way or another, you should end up with a new directory with the name `<dirname>` where you ran the command. Go ahead and treat this directory as if it were any other and develop your project in it. Future example commands with `svn` will assume that you are running it from within the working copy you just checked out.

[2] http://rapidsvn.tigris.org/
[3] http://subcommander.tigris.org/
[4] http://tortoisesvn.tigris.org/

2.5.3 Updating

Your day should begin with updating your working copy with the latest material from the repository. You should also update your working copy whenever one of your group members commits some of their work. To update your working copy, run:

```
svn update
```

This will download any changes that your groupmembers have made and then apply them to your working copy. If there are conflicts between changes you've made in your working copy and changes that have been committed to the repository, you will need to 2.5.6.

If you want, you can "update" to an older version of your code (in order to look at what state the code was in last week, for example). This is done with the `-r` parameter:

```
svn update -r <revision>
```

See `svn help update` for information about what `<revision>` might be. If you run `svn update` to look at old code, be sure to `svn update` again later to bring yourself back up to date. It is no fun to make changes to old code and then have to integrate them with new code before you can commit them.

2.5.4 Examining the state of your working copy

To see what changes have been made to your working copy relative to what files in the repository, run:

```
svn status
```

See `svn help status` for information about what the output says. You should run this before you try to commit code, in order to see what changes you've really made.

If you want to see the specific changes that you've made, you can use `svn diff`. If you just run `svn diff` from a directory, it will print the changes made to all files rooted in that directory. If you run it with a specific file argument, it will display the changes made to that file.

2.5.5 Manipulating files in your working copy

The `svn cp`, `svn mv`, and `svn rm` commands mentioned above may be used directly on your repository, and they should also be used to manipulate files in your working copy. If you want to rename something, the command:

```
svn mv <file1> <file2>
```

will rename the file in a manner that Subversion is aware of it and will track it for you when you want to commit this change.

2.5.6 Committing your changes

Eventually, you will reach a point at which you want to commit your work (this is also called "checking in"). Good examples of such junctures are whenever you are going to make large changes to files, you finish some new functionality, or before you go home for the day.

Always, *always* run `svn update` before a commit. It will save you a lot of headaches. Don't commit code based on something that is out of date! You could end up overwriting someone else's work.

Once you make sure that you're committing work based on your group's latest material, you need to mark the new files you want to commit as "added" and then commit them to the repository.

To mark files as added, use `svn add`:

```
svn add <file1> [file2] ...
```

You can see the status of files with `svn status`.

Once you have added the new files you want to have in your commit, run:

```
svn commit
```

An editor will pop up asking you to write a message describing the commit. Write something informative, maybe even detailing what semantic changes you made to each file, save your message, and then exit the editor. Your commit should be complete! Tell your groupmembers that you've made a commit and that they should update their working copies.

2.5.7 Resolving conflicts

It does happen that, while you are working on one section of your project, someone else will commit something that overlaps with what you're working on. You may resolve the conflict immediately using an interactive prompt, or will need to look at the files with conflicting sections, edit them to your satisfaction, and then mark them with:

```
svn resolved <file1> [file2] ...
```

The section of the Subversion book on resolving such conflicts[5] gives details on this process.

2.5.8 Merging from branches

We've given you some skeleton code for assignments 1 and 3 to get you started on them. Before beginning work on either of these assignments, you should pull its respective skeleton code into your trunk. You may then proceed to work in your trunk.

[5] http://svnbook.red-bean.com/en/1.5/svn.tour.cycle.html#svn.tour.cycle.resolve

Conceptually, what you need to do is merge the code in the branch onto whatever is currently in your trunk. This is very easy for assignment 1, as you are starting with an empty trunk. Once you have checked out your trunk (with `svn checkout`), the command to do this is:

```
svn merge svn://foxwood/groupname/branches/asgn1_skel
```

You then must commit the merged changes with `svn add` and `svn commit`, as explained in "2.5.5."

See `svn merge help` for further explanation of this command. Once again, the section of the Subversion book on branching and merging[6] gives a comprehensive treatment of the subject.

For assignment 3, the merge command is much the same, but you're merging in code from the assignment 3 skeleton branch:

```
svn merge svn://foxwood/groupname/branches/asgn3_skel
```

If you have a conflict because code in the assignment 3 skeleton would overwrite some of yours, you can resolve it as explained in "2.5.6."

ROS skeletons There are alternate skeleton branches for groups who are using ROS. If you are in one of these groups, you should merge from a ROS skeleton branch. The ROS skeleton branches have the same names as the regular branches, except they have `_ros` inserted in their names before `_skel`. For assignment 1, then, the merge command you want is:

```
svn merge svn://foxwood/groupname/branches/asgn1_ros_skel
```

2.5.9 Last notes

If you're wondering about how to do something, run `svn help <action>` or just `svn help` and read it. Some other commands you might find helpful are `svn revert` (undo changes to a file and reset it to a just-checked-out state) and `svn log` (view a history of commits and their messages). Plain old `svn help` lists all of the Subversion commands.

Temporary files like `cmake.install` (generated by CMake) or object files (like executables you generate) shouldn't generally be committed, as they will be changed arbitrarily and you can regenerate them whenever you want.

3 For advanced users

If you and your group are comfortable with some other version control system (git, darcs, hg, or something else), you're free to use it to coordinate your work. Our handin policy does require you to submit your finished material to the appropriate tag in your group's Subversion repository, however, so you must at least do that.

[6] http://svnbook.red-bean.com/en/1.5/svn.branchmerge.html