EX.NO: 9

DATE:

VERSION CONTROL USING SVN

AIM:

Version Control System setup and usage using **SVN**.

Procedure:

In this lab you will learn about *version control*, which is the process of managing multiple versions of the files and directories in a software project. For example, if you develop a program over a period of several days, you may want to store a version of each source code file at the end of each day. Or if you decide to make major changes, you may save a copy of your program before you start any changes.

Version control is also critical when multiple developers are working together on a single project. In particular, version control makes it easier for two people to make edits to the same set of files without overwriting each other's changes.

Today you will learn the basics of SVN, an open source version control tool that is available on machines in the CS department network.

Creating a SVN repository

SVN stores versions of files in a central *repository*. This is a directory that is separate from the directory where you write and test your source code.

Create a repository in your home directory with the command `svnadmin create`:

```
$ mkdir /home/username/svn
```

$ svnadmin create --fs-type fsfs /home/username/svn

Here `/home/username/svn` is the pathname of the new repository. Option `--fs-type` specifies the type of file system to be used by SVN; you must use the value `fsfs` to save a repository on a shared network, as is the case with department home directories.

You can see that your new repository directory already contains files and subdirectories:

```
$ ls /home/username/svn
```

```
README.txt  conf  dav  db  format  hooks  locks
```

You should not directly modify any of these files or subdirectories; they are only to be used by SVN.

Adding data to your SVN repository

Next you will add a project to your SVN repository, but first you must create a temporary directory that will reflect how the project will be organized.

Create the directory `/tmp/lab8_username` on your machine:

```
$ mkdir /tmp/lab8_username
```

Also create these subdirectories:

```
$ mkdir /tmp/lab8_username/trunk
$ mkdir /tmp/lab8_username/branches
$ mkdir /tmp/lab8_username/tags
```

`trunk` is the directory that you will use to store and edit files that you will upload to the repository. We will explain directories `branches` and `tags` later on.

In this lab we will start by making a repository from the unmodified mypoint files from the last lab. Copy these files into the trunk directory:

```
$ cd /tmp/lab8_username/trunk
$ cp /stage/classes/archive/2011/summer/50101-1/lab/lab7/src/mypoint/*
.
$ ls
distance.c mypoint.c mypoint.h  simple.c
```

Next you will add the source files into your SVN repository. (We only want to import source code files, so if you have compiled the above program then remove the binary files now.)

Call `svn import` to add `lab8` and all its contents to the repository. (Here we refer to `import` as a *subcommand* of `svn`):

```
$ svn import /tmp/lab8_username
file:///home/username/svn/cspp50101/lab8 -m"Initial import"
Adding         /tmp/lab8_wax/trunk
Adding         /tmp/lab8_wax/trunk/mypoint.c
Adding         /tmp/lab8_wax/trunk/mypoint.h
Adding         /tmp/lab8_wax/trunk/simple.c
Adding         /tmp/lab8_wax/trunk/distance.c
Adding         /tmp/lab8_wax/branches
Adding         /tmp/lab8_wax/tags

Committed revision 1.
```

Let's look at the arguments in this call to `svn import`:

- /tmp/lab8_username is the directory that will be added to the repository, along with its contents.
- file:///home/username/svn/cspp50101/lab8 indicates that /tmp/lab8_username and should be added as the *virtual* directory /home/username/svn/cspp50101/lab8. We say this directory is virtual because the directory /home/username/svn/ does not actually contain a subdirectory called cspp50101/lab8 that you can examine with ls and cd. Instead, data is stored in a database format, and you must use SVN tools as an interface to examine the repository.
- -m is used to associate a log message with this import. If you do not give a message then a vi editor will open for you to enter one.

SVN stores multiple *revisions* of your repository, where each revision is a snapshot of the repository at a certain point in time. A new, empty repository is at revision 0, and as the above output shows, revision 1 is created when you first import data.

View the contents of your repository by calling svn ls:

```
$ svn ls file:///home/username/svn/cspp50101/lab8
branches/
tags/
trunk/
$ svn ls file:///home/username/svn/cspp50101/lab8/trunk
distance.c
mypoint.c
mypoint.h
simple.c
```

At this point you can remove the directory /tmp/lab8_username because its contents are now stored in your SVN repository.

```
$cd
$rm -rf /tmp/lab8_username
```

Checking out a repository

Next you will learn how to access and update the files that you have just imported. Call svn checkout to retrieve the repository data from the virtual directory that you just created:

```
$ cd /home/username/cspp50101
$ svn checkout file:///home/username/svn/cspp50101/lab8
A    lab8/trunk
A    lab8/trunk/distance.c
A    lab8/trunk/mypoint.c
A    lab8/trunk/mypoint.h
A    lab8/trunk/simple.c
A    lab8/branches
A    lab8/tags
Checked out revision 1.
```

You can see that all the files for the Lab 8 project are now found in directory `/home/username/cspp50101/lab8/trunk`:

```
$ cd lab8/trunk
$ ls
distance.c  mypoint.c  mypoint.h simple.c
```

Editing source files and reviewing edits

You are now ready to edit the Lab 8 source code. First, add the changes you made to distance.c, mypoint.h, and mypoint.c in Lab 7. (You can just copy these files into the current directory). Also copy the Makefile you made in the last lab into the current directory. Build the executable distance by calling `make` to check that your updates are correct.

After you have distance.c and mypoint.c, you can use the `svn status` command to compare the current directory with the repository:

```
$ svn status
?      distance
?      Makefile
M      mypoint.c
M      mypoint.h
M      distance.c
```

The '?' tells us that, as we expect, files Makefile and distance are not under version control. On the second line, 'M' tells us that source code files mypoint.c, mypoint.h, and distance.c have been modified since you checked it out from the repository. File simple.c is not listed since it has not yet been updated.

```
$ svn diff mypoint.h
Index: mypoint.h
===================================================================
--- mypoint.h       (revision 1)
+++ mypoint.h       (working copy)
@@ -8,6 +8,11 @@
 #define MYPOINT_H

 /* *** Add structure declaration here *** */
+struct mypoint
+{
+  double x;
+  double y;
+};

 /* Returns the distance between points (x1,y1) and (x2,y2) */
 double get_distance( struct mypoint * p_p1, struct mypoint * p_p2);
```

Here the lines marked with '+' mark which lines are found in the updated file, and '-' shows the same line in the repository version.

Committing changes

At this point you have updated your working copy of mypoint.c, but the copy in the repository is unchanged. To update the repository you need to commit your changes by calling `svn commit`:

```
$ svn commit -m"Adding changes for lab 7"
Sending        trunk/distance.c
Sending        trunk/mypoint.c
Sending        trunk/mypoint.h
Transmitting file data .
Committed revision 2.
```

The last line tells us that we are at revision 2 of this repository. You can also type `svn info` to see revision number information:

```
$ svn info
Repository Root: file:///home/username/cspp50101/svn
Repository UUID: 1d5eca6d-b426-4f00-922e-c42863db29fe
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: username
Last Changed Rev: 2
Last Changed Date: 2009-03-07 12:16:30 -0600 (Fri, 07 Mar 2009)
```

After committing your changes, re-check the status of the current directory:

```
$ svn status
?      distance
?      Makefile
```

Note that source code files no longer shows up as modified, i.e., with the 'M' status code, since the repository and working copies are now identical.

It is possible to recover previous revisions of a file that is under version control: You can use the command `svn update` to restore your working copy to the latest version in the repository:

```
$ rm mypoint.c  ## delete source file
$ svn update
Restored 'mypoint.c'
At revision 2.
$ ls *.c
mypoint.c distance.c simple.c
```

You can also checkout earlier revisions of a file with the `-r` option:

```
$ rm mypoint.c  ## delete source file
$ svn update mypoint.c -r 1
U    mypoint.c
```

```
Updated to revision 1.
```

Here the 'U' indicates that your current version of the file was updated to match the revision that you requested.

You can also call `svn update` without a file name to update an entire directory:

```
$ rm *.c
$ svn update
Restored 'mypoint.c'
Restored 'distance.c'
Updated to revision 2.
```

Adding files to the repository

After you have created the original repository you may still add and delete files and directories by calling the commands `svn add` or `svn delete`. As before, you must call `svn commit` to apply your local changes to the repository.

You should have already added a Makefile to your lab8 directory. Add this Makefile to the repository by calling `svn add`:

```
$ svn add Makefile
A         Makefile
```

Here the status code 'A' indicates that Makefile will be added on the next commit.

```
$ svn commit -m"Adding Makefile"
Adding         trunk/Makefile
Transmitting file data .
Committed revision 3.
```

You can now call `svn update` and `svn ls` to see that your changes are in the repository:

```
$ svn update
At revision 3.
$ svn ls
Makefile
distance.c
mypoint.c
mypoint.h
simple.c
```

Tags and branches

We mentioned above that you should use the directory `trunk` in your repository to store and edit files. As your project develops, you may find that you would like to take a "snapshot" of your project at a particular revision. For example, you may want to tag a version after you have completed a major revision. You can do this using by storing this version in the `tags` directory that you created earlier.

Next you will tag the current version of your project, which now contains a Makefile. To do this you will make a virtual copy of your latest revision, using the command `svn copy`:

```
$ cd ..
$ pwd
/home/username/cspp50101/lab8
$ ls
branches tags trunk
$ svn copy trunk tags/uses-make
A          tags/uses-make
```

Next commit your changes. Note that you should pass the argument `tags` to the `svn commit` command, to indicate that you only want to commit changes in this directory:

```
$ svn commit tags -m "Tagging version where Makefile added"
Adding tags/uses-make

Committed revision 4.
$ svn ls tags/uses-make
Makefile
distance.c
mypoint.c
mypoint.h
simple.c
```

While it appears that we have copied all the files in the trunk directory, in reality SVN has only tagged a particular revision in its internal database. At any time you can check out a copy of this new directory:

```
$ cd /tmp
$ svn checkout file:///home/username/svn/cspp50101/lab8/tags/uses-make
A     tags/uses-make/Makefile
A     tags/uses-make/mypoint.c
A     tags/uses-make/mypoint.h
A     tags/uses-make/distance.c
A     tags/uses-make/simple.c
$ ls uses-make
Makefile mypoint.c  mypoint.h  distance.c simple.c
```

The `branch` subdirectory is used when you want to make two separate versions, or "branches" of your program, and would like to continue to work on both branches independent of each other. We will not cover branches in this lab.

Accessing a repository remotely

In this lab you created a repository in your home directory that can be accessed from any machine in the department network. Home directories have quotas, however, and this may cause problems for larger projects where the SVN database uses up disk space.

One alternative is to place the repository on another machine and access it remotely. This can be done by using the `svn+ssh` schema with SVN commands, as shown below (You do not need to run this step yourself):

```
$ svn ls svn+ssh://someone@hostname.cs.uchicago.edu/data/svn/trunk
usernames@hostname.cs.uchicago.edu's password:
foo.c
foo.h
Makefile
```

Remote access also allows users on different machines to access the same SVN repository and work together on a single project.

If you would like to set up a repository for collaborative work on future projects, then you should email techstaff@cs.uchicago.edu for assistance.

Your work for exercises 1 and 2 should be completed in directory `/home/username/cspp50101/lab8/trunk/` where you initially added your updates for the mypoint module.

`typedef` and type constructors

In file mypoint.h we created a new compound data type called mypoint, representing a 2-dimensional point. In our code we refer to this type as `struct mypoint`. We declare variables of this new type as:

```
  struct mypoint p1, p2;
```

Alternately, we can rename this new data type using typedef:

```
typedef struct mypoint {
  double x;
  double y;
} MYPOINT;
```

MYPOINT is a new type name that is equivalent to `struct mypoint`. A declaration of two type point variables becomes:

```
  MYPOINT p1, p2;
```

Result:

Thus the version control for handling different software version was successfully completed using SVN.