Introduction to

Using Mercurial Software Repositories

by Derek Schuurman

Source code management systems are used to manage the revision history of a software project. A source code management system has a databases that stores the changes that are made to files in a project. It can be used as a tool to allow different programmers to collaborate on a software project by allowing them to blend and merge the various changes that are made.

Although there are many source code management systems in use today, the one described here is an open-source program called ***mercurial***. Mercurial is friendly and relatively easy to use for those who are new to software repositories.

Step 1: Verify your mercurial installation

To test your installation of Mercurial, type:

```
hg
```

if the command prompt returns with a list of **`hg`** commands that are available, then the program is installed. To get a complete list of commands, type:

```
hg help
```

Note that mercurial is invoked using the **`hg`** command, which also happens to be the chemical symbol for mercury on the periodic chart.

Step 2: Setup a mercurial configuration file

Before you start using mercurial, you should setup a local configuration file to tell it who you are. In Linux in your local home folder, create a file called **`.hgrc`** with the following contents:

```
[ui]
username=Firstname Lastname <username@youraddress.com>
merge=internal:merge
```

In the **`username`** line, substitute your own name and email address. You are now ready to make your own software repository!

Step 3: Create a new software repository

A repository is a database that contains the files in a project. But we must first indicate the files that we want in the repository and those that we are not interested in saving. Typically, we are interested in saving all the source code files. However, we are not usually interested in saving all the intermediate object files and binary files since these can always be recreated from the source files.

To initialize a new project folder, issue the following command:

```
hg init project
```

This creates a new folder named **`project`** for your work. It also creates a new **`.hg`** folder in the project folder. You could also enter an existing project folder and issue the **`init`** command within the project folder.

Next, enter the project folder as follows:

```
cd project
```

Within your project folder, use a standard text editor to create a new text file called **`.hgignore`** which will contain

information about the files that should be *excluded* from the repository. A sample **.hgignore** file is show below:

```
syntax: glob
*.class
*.o
```

This file indicates that **.class** files (for Java) and **.o** object files (for C/C++) should be excluded from the repository. There is no point storing more files than necessary.

You can now start to add and edit files within the project folder. As you do, you can check the status of this files with respect to the repository as follows:

```
hg status
```

This shows the current status of the repository. Initially, all the files in the project folder will be listed with a **?** in front of them indicating that they are unmanaged. To add them to your mercurial repository, type the following:

```
hg add
```

This adds all the files in the current folder except the ones indicated by the **.hgignore** file. Running the status command again will show that files are now part of the repository by placing an **A** in front of each of the filenames. To remove a file from the repository, use the following command:

```
hg remove <filename>
```

where **<filename>** is the name of the file you want to remove from the repository.

Mercurial also has a convenient combination command, **hg addremove**, that adds untracked files and marks missing files as removed.

Step 4: Committing changes to the repository

Once you are done editing your changes, you can decide to "commit" the current changes to the repository. This can be accomplished by typing:

```
hg commit
```

This command will automatically open a text editor where you can enter a meaningful summary to document the changes that were made. Alternatively, you can supply a message directly on the command line as follows:

```
hg commit -m "message"
```

Now run the status command again:

```
hg status
```

This time, no file status are displayed because all the files have been committed to the repository.

We could now delete a file as follows:

```
rm <filename>
```

If this is a file that was part of the repository, it can now be easily restored as follows:

```
hg revert <filename>
```

You can also edit **<filename>** and make changes. To see the differences between a file and its copy in the repository, issue the following command:

```
hg diff
```

Subsequent updates can be committed to the repository as time goes on. You can get a complete history of past commits that were performed by typing:

```
hg log
```

This command prints a brief summary of each change that was committed including the date, time and name of the user who performed the commit along with any message that was saved with the commit.

If you just committed a change that you now regret, and you haven't done any other commits since, you can use a **`rollback`** to undo the last commit as follows:

```
hg rollback
```

Step 5: Working with a remote repository

You can copy a mercurial repository to or from a remote server over the network using SSH. For example, to make a copy of a remote project to a local project folder type:

```
hg clone ssh://username@server.domain/source_project destination
```

where **`username`** is your user name on the server, and the **`source_project`** is the location of the remote project you want to clone, and **`destination`** is the local project folder where the project files will be created. You will be prompted for your password on the server. The step of entering a password can be eliminated if you setup SSH keys.

To bring in changes from a remote repository to an existing local repository, use the **pull** command as follows:

```
hg pull ssh://username@server.domain/path/project
```

Note that this updates the repository database, but not does not (by default) touch the files in the working project directory. Instead, use the **`update`** command to bring the working files in your local "sandbox" up to the latest version:

```
hg update
```

You can look at a previous version of the project using the update command as follows:

```
hg update -r n
```

where ***n*** is the version number you wish to restore (note that the version numbers start with version 0).

One thing to note is that **`hg update`** can fail if there are uncommitted changes. If you want to ignore the uncommitted changes and update anyways, use the **`-C`** option to "clobber" your changes:

```
hg update -C
```

To "push" your changes to an existing remote repository, use the **`push`** command:

```
hg push ssh://username@server.domain/path/project_folder
```

Note that the "push" command updates the remote repository database, but does not update the actual working files in the project directory since other people may be working on them while the update is in progress. To update the working files in the remote project directory, issue an update on the remote server after completing the push operation.

This simple introduction should help you get started with small personal and team projects. Once you are more comfortable with the basics, check out other tutorials on the web to learn more.

Drill Exercises

1. Create a simple Java program that prints "Hello World". Initialized and commit the code into a local mercurial repository (note: be sure to ignore the **`.class`** file and just save the **`.java`** source files).
2. Make some minor changes to your program and then use the "revert" command to confirm you can return to the original version of the program.
3. Add code to display another message and then commit the changes to the repository.
4. Copy the repository by cloning the repository to a folder on a remote server.
5. Pull the code from the remote server to a new local folder and ensure that all the project files are copied and are up-to-date.