Version Control Script

Mike Jackson, The Software Sustainability Institute

Things you should do are written in bold.

Suggested dialog is in normal text.

```
Command-line excerpts and code fragments are in shaded fixed-width
font.
```

Derived from Chris Cannam's original at, https://code.soundsoftware.ac.uk/projects/easyhg/wiki/SC2012BootcampPlan, but just using the command-line.

Prerequisites

Mercurial, BitBucket.

Introduction

Do VersionControl.ppt

We'll introduce these concepts practically, using Mercurial.
We'll use the command-line.
There's a GUI tool called EasyMercurial, which you've installed, that is more visually impressive but the concepts are the same.
Besides, if you can use it at the command-line you can use it via the GUI no problem.
We have tried to introduce both tools in the boot camp but this led to some confusion...and we're teaching the concepts not the tools!
Please follow along…

Create a repository and add a file

```
mkdir myrepository
```

```
cd myrepository
```
We can turn this directory into a repository…
```
hg init
```
hg is the Mercurial command – after the atomic element name.
```
xemacs fishstew.txt
```
Ask for some ingredients and save.
```
hg status fishstew.txt
```
It's a "?" as it's not been added so the repository knows nothing about it.
```
hg add fishstew.txt
hg status fishstew.txt
```
Note the status now says "A" as it's been scheduled for addition but is not yet in the repository.
```
hg commit -m " Recipe with ingredients only" fishstew.txt
abort: no username supplied (see "hg help config")
```
We need to give a username and e-mail so hg can record who is making these changes. So, edit a .hgrc file in your home directory
```
xemacs /home/mjj/.hgrc
```
I'll add my username and e-mail.
```
[ui]
username = Boot Camp <bootcamp@gmail.com>
```
Now try again. Note that the "-m" is a commit message. This is very important as it says why you made the change. This is the one part of version control you have to do yourself! And make it meaningful so you and your colleagues will know what you've done and why. Nothing is more redundant than a "made a change" message – and, yes, I'm guilty of those.
```
hg commit -m "Recipe with ingredients only" fishstew.txt
```
And success. If we check the status now…
```
hg status fishstew.txt
```
…we see nothing as we've now commited it.
If we run the status on the directory…
```
hg status
```
We may see one for "~" files if we're using Emacs.
We can define files that Mercurial can ignore…
```
xemacs  .hgignore
```
Add
```
syntax: glob

*~
```
Save.
And we can add and commit it.
```
hg add .hgignore
hg commit -m "Ignore editor scratch files" .hgignore
```
If we check the status now, the temporary files won't show up…
```
hg status
```
We can add any files to the status e.g. .class files or .o files, any binary that can be regenerated from the source code, for example.

Log, status, difference and revert

We can see the history, or log, of changes…

```
hg log fishstew.txt
```

Now, let’s add a method and save our recipe again.

Add method and save.

And if we now check the status…

```
hg status fishstew.txt
```

“M” means modified, we’ve made a local change to our working copy. So let’s commit…

```
hg commit -m "Added method" fishstew.txt
```

And check the log…

```
hg log fishstew.txt
```

Now add one ingredient and remove one…give me one to add and one to remove…

Get ingredients, update and save.

```
hg diff fishstew.txt
```

diff shows us the difference between the file in the repository and the one in our working copy.

“-“ denotes a removed line and “+” an added line.

Suppose we’re unhappy with that and want to abandon our changes…we can do…

```
hg revert fishstew.txt
```

…and throw away the changes to our working copy. Note how it creates a copy of our changed version just in case we need it.

Now, let’s look at the log again…

```
hg log fishstew.txt
```

We can retrieve our original version as follows…

```
hg update 0
more fishstew.txt
```

Note the method is missing! But we can go back to the current version, since the repository records the complete history of changes…

```
hg update 0
more fishstew.txt
```

Or we could just do “hg update”.

Now let’s leave aside our fish stew, so exit the editor, and create another file e.g. for an omlette.

```
xemacs omlette.txt
hg add omlette.txt
hg commit –m “Omlette recipe” omlette.txt
```

We can bounce back to our initial state…

```
hg update 0
ls
```

And omlette.txt isn’t there. We can then go to version 1…

```
hg update 1
ls
```

And it’s not there. And if we return to our most-up-to-date version,

```
hg update
ls
```

…it is!

These commands apply to sets of files and directories too.

We can use diff in conjunction with versions too for a single file,

```
hg diff -r 0 -r 2 fishstew.txt
```

Or a directory,

```
hg diff -r 0 -r 2
```

We can bind specific versions to journal papers or conferences or important events just by noting down the version number we used! But that's a bit opaque. So we can tag the repository, for example…

```
hg tag MY_JOURNAL_CODE
```

We can use MY_JOURNAL_CODE as a revision number and get that version. For example,

```
hg update 0
ls
hg update MY_JOURNAL_CODE
ls
```

And if we look at the history, we can see what version it corresponds to…

```
hg log
```

Now I'll just ensure I'm using my latest version…

```
hg update
```

We can now manage the complete history of changes to our files but we still have a big problem? What is it?
This is all local and if our laptop crashes then good-bye to all our code, history and all!

Working by yourself with backups

Now we could just manually copy our entire repository using a secure copy, or dump it into DropBox but version control systems support remote access and use too.
So you should all have a BitBucket account…if not you'll just have to play catch up!
So, browse to https://bitbucket.org/ and log in.
Click Create a repository.
Enter Software Carpentry Boot Camp.
Check Access level: This is a private repository.
Select Repository type: Mercurial
Click Create repository.
Now Click Make changes and push
You'll see a "push" URL so copy that into your window and hit return

```
hg push https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

Enter your BitBucket username and password
Now, click Source and you'll see your files are on BitBucket.
Click Commits and you'll see your complete commit history. You've just pushed your entire repository to BitBucket.
It is now backed up. And BitBucket back-up their servers too!
We may get a warning like

```
warning: bitbucket.org certificate with fingerprint
24:9c:45:8b:9c:aa:ba:55:4e:01:6d:58:ff:e4:28:7d:2a:14:ae:3b not
verified (check hostfingerprints or web.cacerts config setting)
```

This is because from version 1.7.3 Mercurial warns us about SSL problems. Either we can use an --insecure flag at the command-line or edit our .hgrc file and add:

```
[hostfingerprints]
bitbucket.org =
24:9c:45:8b:9c:aa:ba:55:4e:01:6d:58:ff:e4:28:7d:2a:14:ae:3b
```

Basically just the fingerprint in the warning.

[From https://bitbucket.org/site/master/issue/2780/getting-warning-while-using-https-and-ssh]

Let’s make another change to our recipe. Give me another one for the fish…

Make the change.

```
hg commit -m "Added anchovies" fishstew.txt
```

And another, for the omlette…

Make the change.

```
hg commit -m "Added mushroom" omlette.txt
```

Now, let’s push our changes…

```
hg push https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

And now in our browser click Source then Commits and we should see our new changes are there.

Now let us suppose something awful happens…

```
cd ..
rm -rf myrepository/
```

Argh! We’ve lost our repository!

But we now have a complete copy on BitBucket so we can get, or clone, that…

```
hg clone https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

And if we change directory…

```
cd software-carpentry-boot-camp/
ls
```

We can see our files there! Phew!

Now let’s clone another copy of our repository, elsewhere in our file system…

```
cd ..
hg clone https://mikej888@bitbucket.org/mikej888/software-carpentry-
boot-camp mycopy
cd mycopy
```

Edit omlette.txt and add peppers.

```
hg commit -m "Added peppers" zonk.txt
hg push https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

And now in our browser click Source then Commits and we should see our new changes are there.

And now let’s change back to our original repository.

```
cd ../software-carpentry-boot-camp
```

So we know our BitBucket repository has changed. But how would we find this out, for example if a colleague had made a change. We can do…

```
hg incoming https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

This shows us the changes that have been made. And we can then get, or pull these,

```
hg pull https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

The changes are now in the history but not the working copy…

```
hg history
```

And we can see what was added e.g.

```
hg diff -r 4 -r 5
```

So now we update…

```
hg update
```

So, we can push changes to a remote repository, pull changes into a local repository or clone a remote repository. And now we have everything we need to work with others!

Working with colleagues

So, please could I have a volunteer.
Give me your username and I'll share my BitBucket repository with you.
Give the volunteer write access to your repository.
In BitBucket, click on the cog, and click Access Management.
Enter the volunteer's username, select Write and click Add.
Now, if my volunteer could now clone the repository from BitBucket.
This is the command that they will use:

```
hg clone https://volunteer@bitbucket.org/mikej888/software-carpentry-boot-camp mycopy
cd mycopy
```

The first username is their username, so they can log into BitBucket, the second is mine, since they're using my repository.
Now, please make a change to fishstew.txt and commit it.
Now, I'll pull these changes down…

```
hg pull https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

You'll notice there are none since my volunteer's changes are still local to them.
Now please push your changes.

```
hg push https://volunteer@bitbucket.org/mikej888/software-carpentry-boot-camp
```

Now, I'll pull these changes down…

```
hg pull https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

And I now have them.
The changes are now in the history but not the working copy…

```
hg history
```

So, what I now need to do is

```
hg update
```

Now, please could you change a specific line. And I'll also change that same line.
Now you commit and push your changes.
I'll commit my changes.
And now, when I push my changes…

```
hg push https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

I get a warning about "push creates new remote heads" or "Push failed". This is because the repository has changed since I last received a version from it.

I need to pull to get the most up to date version.

```
hg pull https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

I can see their changes.

```
hg history
```

So, what I now need to do is

```
hg merge
```

As we changed the same lines I get a conflict! If we look at the file we can see these marked
I can now either
Accept my change and ignore my volunteer's.
Accept my volunteers and ignore mine.
Or edit the file and resolve it manually.

```
xemacs fishstew.txt
```

I now need to mark it resolved and commit it,

```
hg resolve -m
```

Note that "-m" here is "mark resolved" and not "message".
I now commit the merged version.

```
hg commit -m "Merged P1 and P2s changes."
```

Then push the new version.

```
hg push https://mikej888@bitbucket.org/mikej888/software-carpentry-boot-camp
```

And we can run

```
hg annotate fishstew.txt
```

to see what changes were made by who and when.
If you look at Commits on BitBucket it shows a tree of the changes.
So, when to share? Whenever you have completed a unit of work.
Ensure units of work are small enough to be reviewed carefully by a peer within an hour to encourage and contribute to code reviews.

Practical

Now, pick a partner.
One of you give your BitBucket username to the other. The other, give your partner write access to your repository.
Both of you clone the shared repository.
Now both edit the same file and commit the changes.
Now both try and push.
One of you will get a conflict so you'll need to pull, merge, resolve, commit and push.

Summary

Version control gives you...

- Back-ups
- Support for collaborative working

- A complete history of the provenance of your code, configuration files, documents etc.
- The ability to get the exact code you used to produce the exact data you used to produce the exact graph in a specific conference or journal paper, or thesis.

So, when to share? Whenever you have completed a unit of work.
Ensure units of work are small enough to be reviewed carefully by a peer within an hour to encourage and contribute to code reviews.

It is the single most important tool we teach over these two days!