# an introduction to mercurial
# MCNet Computing School 2016

Chris Pollard

University of Glasgow, MCNet

2016 05 16

# Version control

There are a number of reasons why source control is important for modern code development:

- need backup copy (code is never lost)
- need reproducibility of former results
- need to document changes to the code base
- need to be able to collaborate
- need to develop features that may not work out!

Version control systems (VCSs) help with each of these by tracking changes in your code over time.

# VCSs

Several VCSs are available on the market: mercurial (hg), git, svn, (dropbox), etc. Why did we choose hg for today?

- it's a modern VCS
- it's distributed
- fairly intuitive to use
- widely supported
- dropbox does not really know how to handle merges.

Note that if you're not working with text (e.g. code), then hg, git, and svn may not be what you want!

# Before we start

Let's try to do this interactively (if you have hg installed).

Make a new folder and write a few lines of text into a file (poetry, prose, code, whatever)

To initialize the repository, move into the folder and issue the "hg init" command.

```
$ hg init
```

# Adding a file

Let's add your new file to the repo:

```
$ hg add myPoem.txt
```

myPoem.txt is now being tracked by hg! We still need to commit this to the repository:

```
$ hg commit -m 'adding my poem'
```

# First changes

So now you've started tracking your file. Go make some changes to it.

One very powerful feature of VCSs is the ability to show you the difference between your current copy of the code and another copy from another point in the past.

'hg diff' is your friend!

```
$ hg diff
```

# First changeset

Ok we have a tracked file and have made some changes. Let's commit the changes.

```
$ hg commit -m 'poem is now better.'
```

The 'hg log' command lets you see the history of your repository. Try it.

```
$ hg log
```

# Step back

So what is going on here?

- hg stores changesets (labeled by hash in the log)
- each changeset also has a *local* decimal revision number
- each changeset is (essentially) a snapshot of the repository at a point in time with zero, one, or two parent changesets.
- user, timestamp, commit message are stored with each changeset.
- the *tip* of the repository is always the most recent changeset; it is local!

In general, diff can be used to show the difference between any two changesets!

# Branching

Sometimes we want to try something that we're not sure will work out or we want to work on a feature that will temporarily break everything.

In this case it's often useful to make a *branch*

```
$ hg branches
$ hg branch feature_x
$ hg commit -m 'creating feature_x'
$ hg branches
```

Our working copy is no longer on the default branch!

# Playing with branches

Make some more edits to your text file and then commit them (with appropriate message), then look at the log again.

```
$ hg log
```

The *tip* of the repository should no longer point to the default branch, but to your new one. Switch between branch heads using the checkout command.

```
$ hg checkout default
```

# Merging

You can now make parallel edits in both of your branches—just use checkout to move between them and commit appropriately.

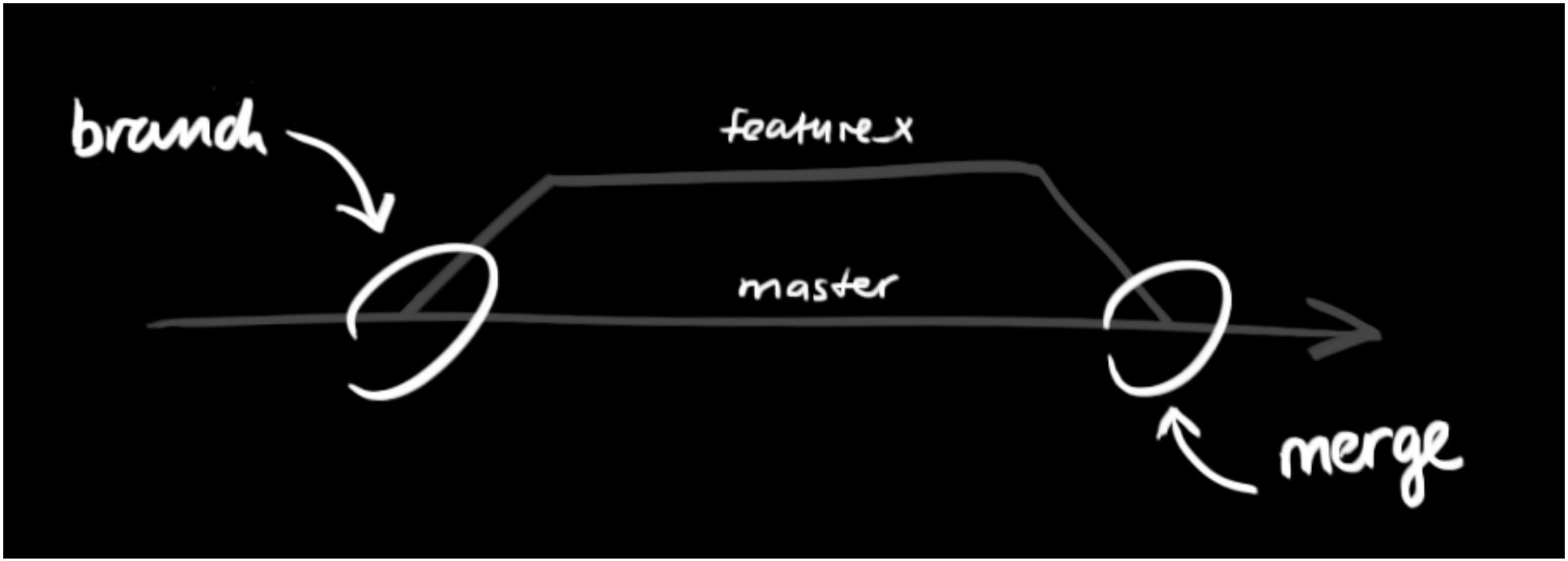
Usually we want to *merge* them back together eventually, so think carefully about what changes you make in each branch.

Let's merge our changes back into the default branch and see what happens.

```
$ hg checkout default
$ hg merge feature_x
$ hg commit -m 'merging in feature_x'
$ hg log
```

# What have we done?!

Here is what we've done:




(except master $\rightarrow$ default)

# Cloning

Often hg is used as a collaborative tool, so we need to be able to *clone* a repository that someone else is working on

```
$ hg clone PATH/TO/REMOTE/REPO PATH/TO/LOCAL/REPO
```

Once we've cloned a repository, we can pull in new remote changesets:

```
$ hg pull
$ hg update
```

Don't forget to *update* any branch you're working on!! 'Update' can also be used to set the current revision number; it just defaults to the head of a branch.

# Pushing

If you make local changes that you want to share with your friends, the 'push' command is your friend.

```
$ hg push
```

If you've created a new branch that you want to share, don't forget to pass the '--new-branch' option. When in doubt, always use the built-in help!

```
$ hg help push
```

# Tags

It's often useful to assign a label to a particular changeset, for instance if you perform an analysis that will go into a publication or want to release your code publicly.

For this, we often use tags:

```
$ hg tag PoetryBookRelease
```

Tags are extremely useful because *they don't move*! Branch heads are always changing, but tags do not. This helps maintain reproducibility.

# A few more useful commands

Before we end, I wanted to point out two more useful commands

The ‘summary’ command in practice gives you a lot of useful information about your working copy

```
$ hg summary
```

The ‘blame’ command gives you line-by-line information on when code last changed and by whom.

```
$ hg blame myPoem.txt
```