Unix and Computer Science Skills Tutorial
Workbook 4:
Version Control

October 15, 2015

This workbook is a tutorial on how to use version control tools to keep track of your programs.

As with the other workbooks I recommend that you take the time to work through the examples to get a feeling for how things work. Unlike the other workbooks this one provides a general overview of version control but then points you at outside resources to provide you with the particulars.

This workbooks assumes that you are comfortable doing basic tasks on the command line to the extent covered in Workbook 1.

1 What is Version Control?

Often enough when programming you'll do something and discover that you wish you hadn't. Sometimes you'll discover that you've made a mistake that's hard enough to undo that it's easier to go back to how things were before you started. Sometimes you'll discover that adding a new feature to your program breaks things that used to work. Sometimes you'll discover that the thing you added wasn't the thing you needed at all.

Version control let's you handle this. When using version control, you keep copies of your program at various points (usually where it compiles and works as expected). These versions allow you to go back to how your code looked at any point in time. You can do version control manually by copying your files into a separate directory (possibly one named with a time stamp), but this requires you to do a lot of work. Fortunately there are a solutions in the from of Version Control Systems.

Version Control Systems allow you to keep track changes on your work by allowing you to *check-in* your *working copy* to a *repository*. Then when you need a particular version There have been a lot of different version control systems created by various organizations for various purposes over the years. You can also create a *branch* of your project where you can try things out before *merging* the branch back in if the changes were a good idea or abandon the branch if the changes weren't.

Version Control Systems also make it easier to work with a group. Different members of the group are able to check out from a main repository and make changes in their own copies before checking them back into the repository. Problems can arise, however, when two people make changes that *conflict*. For example, two developers might add each add a new function to the same class. When they check in their code they will each have different versions of that class and someone (usually whoever checks in second) will have to sort out the different versions into a whole again (and most version control software has tools to help with this). Usually this is easy, although sometimes there can be complications. So while version control systems make it easier to work together they don't eliminate the need for communication and planning.

One very important thing to remember is that *Version control is not the same as backing things up*[Wolever]. Version control let's you keep versions of your code through time, but backing up makes sure that you're able to recover from something going wrong. For one thing you need to ensure that you have backups *of* your version control repository. If you're using a professionally maintained remote version control server (for example the department offers both `forge` for CVS and `gitlab` for Git) then those are usually routinely backed up. If you are running your own you will have to make sure you back your data up yourself. You should always have a system in place to make sure that you can get all of your data back if something goes wrong.

> One other important thing to mention is that version control is not just for programming. Version control is very useful for keeping track of your configuration files. It's also useful in graphic design, or creative writing or any other project where keeping versions of your work will be useful. I've used version control to keep all of my work on this tutorial organized.

2 Learning Modern Version Control (By which we mean Git)

There are a lot of version control systems, however we are going to focus on Git which is currently the most popular version control system. It solves a lot of the problems that used to plague version control systems and is very flexible and fairly user friendly. Because it is so widely used it is also really well documented.

2.1 TryGit

I think that the easiest way to learn most of the concepts covered by this tutorial is to practice and fortunetly GitHUB (a service that hosts Git repositories) offers an excellent interactive tutorial at `https://try.github.io/levels/1/challenges/1`.

2.1.1 Questions

1. Complete the GitHUB tryGit tutorial.

2.2 Try Git Yourself

The full documentation for Git is located at `https://git-scm.com/docs` but this tends to be a little heavy for starting out. As with many other technologies it's worth finding a cheat sheet that you like. I recommend one by Roger Dudler here: `http://rogerdudler.github.io/git-guide/`.

2.2.1 Questions

2. Create a new directory in your home directory structure create a git repository in it. Once you've done that:

   - Create two new files in that directory.
   - Add and commit those files to your repository.
   - Make a change to one of your files and add and commit that change.
   - Create a second branch in your repository and switch to it.
   - Change the other files and commit it in your new branch.
   - Switch branches to see what the differences in states are between your two branches.
   - Merge your two branches into one again.

2.3 GitLab

There are lots of different places where you can store remote versions of your repositories. The department provides one of it's own using a software suite called GitLab. You can access it at `gitlab.cpsc.ucalgary.ca`. You can log in using your CPSC credentials.

2.3.1 Questions

3. Set up your profile on the CPSC GitLab server, including adding an SSH key.

4. Add GitLab to your repository as a remote remote repository and push your repository to the GitLab server.

5. Add someone else in the tutorial to your project in GitLab. Experiment with working together by editing some text files. Experiment with merging by creating different changes to the same file.

2.4 GitHub

GitHub hosts a wide variety of projects, many of which are worth contributing too or cloning for related purposes. Take a few minutes to explore GitHub (possibly using its explore feature).

2.4.1 Questions

6. Find two interesting projects on GitHub. One should be a programming project in Python or Java, the second can be anything that strikes your interest. Checkout local copies of these projects to a directory in your home directory and explore them.

References

[Wolever] **Using git for Backup is Asking for Pain**, Blog Post, David Wolever, December 8, 2009, `http://blog.codekills.net/2009/12/08/using-git-for-backup-is-asking-for-pain/`