Computer Science 507
Software Engineering
The College of Saint Rose
Spring 2015

Lab 5: Source Code Control

Due: 6:00 PM, Monday, March 9, 2015

In this week's lab exercises, we will be experimenting with some of today's more popular source code control systems: Subversion and Git.

You should work in groups of size 2 to 5 for these exercises. Ideal groups would combine members who have some experience with version control who can guide the team through the basics, and others for whom this is new. If you don't know the basics, don't let your team get ahead of you before you understand what is going on!

Getting Set Up

Create a document where you will record your answers to the lecture assignment and lab questions. If you use plain text, call it "`lab5.txt`". If it's a Word document, you can call it whatever you'd like, but when you submit, be sure you convert it to a PDF document "`lab5.pdf`" before you submit it.

Version Control Basics

Read about version control in Section 25.5 of Sommerville, and the Version Control Basics section of the free online book, Version Control with Subversion.

? Question 1:

Even if working independently on a software project, describe briefly why a source code control system would be helpful to manage development. Consider both multiple development environments (*e.g.*, a portable computer as well as an office desktop computer, both used in development) and the need for revision history. (3 points)

Subversion

Read the sections Version Control the Subversion Way and Chapter 2, Basic Usage (except the "Dealing with Structural Conflicts" section) in Version Control with Subversion.

I have created a shared subversion repository for all of us to try out on `mogul.strose.edu`. It was created with the command:

```
svnadmin create /home/cs507/s15/svn-lab/svn
```

I then added the typical directory structure to include "`branches`", "`tags`", and "`trunk`" directories in the root of the repository, using these commands:

```
mkdir -p /tmp/svn/{branches,tags,trunk}
svn import /tmp/svn file:///home/cs507/s15/svn-lab/svn -m "initial import"
```

Once these commands were executed (and I set permissions so all of us in the course have permissions to write to the repository), the repository was ready to be used. So let's use it.

First, you (all group members) will need to set your permissions mask, so that when you commit to our group repository, the permissions will remain "group writeable".

Please edit the file `~/.bashrc` on `mogul.strose.edu` so that it contains the command

```
umask 002
```

This will mean files that you create, will, by default, include group write permissions. Combined with the fact that I have set the `svn-lab` directory to have "set group-id" permissions, this should allow all of us to share access to the repository.

Once you have edited the file, log out and log back in to make sure it has taken effect.

You can then "check out" a copy of our repository.

To avoid potential conflicts (which we will reintroduce intentionally later on), we will take turns working through this section of the lab. Please do not proceed until your group is cleared to do so by possessing the physical "edit token" we will have in class.

Create an empty directory where you would like to do this, and issue one of the following commands. If on `mogul.strose.edu`, you can check out with the `file` protocol.

```
svn checkout file:///home/cs507/s15/svn-lab/svn/trunk svn
```

If on a different machine, you will need to use the `svn+ssh` protocol.

```
svn checkout svn+ssh://you@mogul.strose.edu/home/cs507/s15/svn-lab/svn/trunk
```

In either case, you should now have a directory called `svn` which contains the contents of the "trunk" of the repository.

? Question 2:
What is in that directory when you first checked it out? (1 point)

You should see at least a file called `CLASSFILE` in your working copy directory.

? Question 3:
Add a paragraph for yourself or your group to `CLASSFILE` and commit your change with an appropriate commit message. Paste the output you get when you run your command. (2 points)

For example, the following will commit your version of `CLASSFILE` with a good commit message.

```
svn commit -m "CLASSFILE addition for Alice, Dilbert, and Carol" CLASSFILE
```

This is the end of the part of the lab where we need to take turns. If your group is working on this part, please announce when you get to this point so the next group can get started.

Now, create another file in our class repository. It should be a program in any programming language you wish that prints out the names of your group members, and anything else you wish.

Question 4:
What does the `svn status` command tell you about your file after you created it? (1 point)

After you create your file, you still need to tell `svn` about it. You will want to `add` it to the repository with `svn add`.

Question 5:
What does the `svn status` command tell you about your file after you add it? (1 point)

It is still not committed. `svn commit` will be needed for that.

Question 6:
What does the `svn status` command tell you about your file after you commit it? (1 point)

Next, using a different group member's account, check out a new working copy of the repository (or update it if you had already checked it out previously). We'll call this "copy B", and the one you were working with previously "copy A".

Question 7:
List the commands you used to do the above. (2 points)

In each copy of the repository, make a change to the program you added. To avoid conflicts, make sure each copy is edited in a different place in the code.

Commit the file from copy A to the repository, then attempt to commit the file from copy B.

Question 8:
What output do you get when you attempt the second commit? (1 point)

Assuming the intent is to get both changes into the repository, we need to update copy B after committing from copy A other, then commit from copy B, and update copy A.

Question 9:
List the commands you used to achieve this. (2 points)

Git and GitHub

If you are not familiar with Git, read quickly through Learn GitHub's Introduction to Git.

If you do not already have a GitHub account, create one and log yourself in.

Examining a sample GitHub repository

In this part of the lab, you will look at a previously-constructed repository on GitHub named "Bootstrap".

Load `https://github.com/twbs/bootstrap` in your favorite browser. We will examine several aspects of its contents.

First, notice the three buttons at the upper right of the page: "Watch", "Star", and "Fork".

Question 10:
Investigate and briefly describe the purpose and functionality of each. (3 points)

In the main table of project files on the page, find the `README.md` file, and click on it to examine its contents.

Question 11:
Based on the contents of this file, briefly summarize the purpose of the project. (1 point)

Return to the main Bootstrap page and click the "commits" link. Then scroll down to find the link (under "Commits on Feb 10, 2015") for "Merge pull request #15779 from twbs/deps", and follow that link. Note: be sure to click on the "Merge pull request" part of the entry, not the number.

Question 12:
How many changed files are involved? How many additions and deletions were committed? (1 point)

Question 13:
What color coding is used to indicate additions and deletions to the files. (1 point)

Return to the main Bootstrap page and click the "Pull Requests" link. You should see a tabbed area with entries for "Issues", "Pull requests", "Labels", and "Milestones". Browse through some of the entries under each category.

Question 14:
Briefly describe the purpose of issues and pull requests, and how they differ. (3 points)

Click on one of the links to see the details of one of the Issues on the "Issues" category. Then click on the "Pulse" tab off to the right (shown as a small icon that looks like a heart monitor display).

Question 15:
Describe the information shown under "Pulse". (1 point)

Creating your own GitHub Repository

Next, your group will create its own repository on GitHub. Here are the steps:

1. Make make sure each member of the group has and knows his or her GitHub login credentials.
2. One group member should be in charge of creating the repository, and that person should log into GitHub with his or her credentials.
3. Click on the "+" icon next to your username, and choose "New Repository".
   - Give your repository a meaningful name such as "`cs507s15lab5`" and include a description.
   - Be sure to choose the "Public" option, and check the box for "Initialize this repository with a README."
4. Add your teammates as Collaborators
   - Click the "Settings" button on the right.
   - Select "Collaborators" on the left menu.
   - Add each teammate as a collaborator.
   - Back in "Settings", check the "Restrict editing to Collaborators only".
5. Create a Java program `Hello.java` in your repository, which prints your name when it is run.
   - In the main view of your GitHub repository, you can click the "+" after the name of your repository above the file list.
   - Enter `Hello.java` for the file name at the top.
   - Type in your Java code in the main editor window, and commit it to the repository with the "Commit new file" button at the bottom.

Question 16:
Give the URL of your repository. Creation of the repository is worth 10 points.

Using your Group Repository

Finally, other team members will access and alter the repository. Each team member other than the one whose account created the repository should perform these steps. It is best if one team member at a time works through these steps.

You can do these either by installing the GitHub desktop application for Windows or Mac, or by using the command-line from `mogul.strose.edu`. These instructions apply to mogul.

1. Log into mogul.
2. Create a new directory into which you can clone your group's repository and `cd` to that directory.
3. Clone the repository with a command such as

   ```
   git clone https://github.com/USERNAME/REPONAME.git
   ```

   where you would replace `USERNAME` with the GitHub username of the team member who created the repository and `REPONAME` with the repository name.

   This should result in a directory with the team's files.
4. To be certain your clone of the repository is up to date with the master on GitHub, issue a `pull` command from inside the cloned repository:

   ```
   git pull
   ```

5. Edit the `Hello.java` file to add a line or two (could be code, could be comments). Use Emacs or your favorite editor.

   Question 17:
   (Only one group member needs to answer this.) Issue the command `git status`. What does this print? (1 point)

6. Now issue the command:

   ```
   git add Hello.java
   ```

   Question 18:
   (Only one group member needs to answer this.) Issue the command `git status` again. What does this print now? (1 point)

7. Your change to `Hello.java` is ready to commit to your cloned repository. Issue the command:

   ```
   git commit
   ```

and add an appropriate message in the text file that comes up. These messages are important to remind yourself and tell others about what you just did.

> **Question 19:**
> (Only one group member needs to answer this.) Issue the command `git status` again. What does this print now? (1 point)

You have now committed the change to your cloned repository, but it is not back in the master on GitHub yet.

8. Let's push our change back to the master, with the command:

```
git push
```

You will be prompted for your GitHub username and password. Enter your own, not that of the team member whose account was used to create the master repository.

> **Question 20:**
> (Only one group member needs to answer this.) Issue the command `git status` again. What does this print now? (1 point)

So your change should now be back in the master copy of the repository. You can verify this back on GitHub's web site for your repository.

> **Question 21:**
> A successful update to your `Hello.java` by each member of your group is worth a total of 12 points.

There is much more `git` can do, but hopefully this gives you the basics of a typical workflow.

Submitting

Before 6:00 PM, Monday, March 9, 2015, submit your lab for grading. Package up all required files into an appropriate archive format (`.tar.gz`, `.zip`, and `.7z` are acceptable) and upload a copy of the using Submission Box at `http://sb.teresco.org` under assignment "Lab5".

Grading

This assignment is worth 50 points, which are distributed as follows:

Feature	Value	Score
Lab Question 1 (why version control?)	3	
Lab Question 2 (`svn checkout` contents)	1	
Lab Question 3 (`CLASSFILE` commit output)	2	
Lab Question 4 (`svn status` after create)	1	
Lab Question 5 (`svn status` after add)	1	
Lab Question 6 (`svn status` after commit)	1	
Lab Question 7 (new checkout commands)	2	
Lab Question 8 (commit attempt output)	1	
Lab Question 9 (commit complete output)	2	
Lab Question 10 (watch, star, fork descriptions)	3	
Lab Question 11 (Bootstrap purpose)	1	
Lab Question 12 (changes/additions/deletions)	1	
Lab Question 13 (color coding)	1	
Lab Question 14 (issues vs. pull requests)	3	
Lab Question 15 (Pulse description)	1	
Lab Question 16 (URL and successful creation)	10	
Lab Question 17 (`git status` after edit)	1	
Lab Question 18 (`git status` after add)	1	
Lab Question 19 (`git status` after commit)	1	
Lab Question 20 (`git status` after push)	1	
Lab Question 21 (Update `Hello.java` using `git`)	12	
Total	50	