

## Lab 02: Part 1, In Class Using a Subversion Repository

### Instructions

Exercises 2.0 – 2.3 are to be completed in class before you leave. For this and all future labs, all of your work will be done and submitted through your own student repository on the nisl class server. If you have not done already, you need to check out a working copy of your personal repository before you can begin this lab.

Use the in-class lab time not only to find the answers to the questions and complete the given tasks, but to practice your skills in using subversion repositories from the unix command line and/or from a subversion client like kdesvn or rapidsvn. The first portion of the lab is to be completed in class so that you may ask the instructor for help and suggestions for any problems you may be experiencing learning about subversion and version control systems.

### Exercise 2.0: Preliminaries

Check out a working copy of your students repository if you haven't already done so. Your student repository will have a url designation of the form:

<http://nisl.tamu-commerce.edu/repo/csci553/username>

Where you need to substitute your own nisl account username for the last portion above.

In your working copy, create and add to your repository a directory called *lab02* (make sure you name it exactly as shown, make sure you not only create the directory, but you run the appropriate command to add the directory to be under version control.

### Exercise 2.1: Moving Files

In order to keep things clean in your repositories, we are going to also at this time create a directory to hold your class project work for the class. If you are up to date on all of your work, you should already have a file called *project.txt* in the top-level of your repository. Create and add a directory to your repositories top-level called *project*, the same as you did to create and add the *lab02* directory in step 2.0. Use the subversion rename (also referred to as move or mv) command to move your *project.txt* file into your new project directory. If you previously had called your file something other than *project.txt*, now is a good time fix this to the correct name.

**WARNING:** Be careful, there is a difference between using the bash mv command to move the file, and using the subversion mv command to move the file. To get this part right you must use subversions rename/move to move your *project.txt* file into your new *project* directory.

**Exercise 2.2: Status and Committing Changes**

At this point you should have done several actions in your repository. From the root of your working copy, perform the following command:

```
[harry@nis1 harryrepo]$ svn status
A      lab02
D      project.txt
A      project
A +    project/project.txt
```

If the result of your status does not look exactly the same as this then you may have done something incorrect. Ask for help from the instructor. If you have done things correctly, then this output indicates that the directories *lab02* and *project* have been added to the repository, the *project.txt* has been deleted, and (re)added into the *project* directory. The + sign indicates that this addition was actually the result of moving an existing file. At this point please perform the command: `svn help status`, and read a bit about what the various indicators mean in the subversion status output.

In order to do the next part of the lab, you need to commit your current changes to your repository. Please do a commit of your repository from the root of your working copy.

**WARNING:** In order to do this step correctly, you must commit all changes in a single commit operation. If your commit does not indicate that you are checkpointing the 4 changes listed above, then you have done something wrong. I can tell from the log of your repository whether or not you successfully commit all changes in a single operation.

**WARNING:** Also, I will now and in the future be taking off points if you don't provide meaningful log messages for your commits. You should get in the habit of describing what has happened in the log message for the commit you are performing. Provide a meaningful message of about a sentence or 2 describing what you just did and why.

**Exercise 2.3: Questions about Subversion**

Create and add a file called *lab02-part1.txt* in the *lab02* directory of your repository. Provide answers to the following questions in this file. When you are done, make sure you commit the file to the repository.

- 1) Change to your project directory and perform the command: `svn log project.txt`. Does the result surprise you? What would have been different about the log if you had done a `svn rm` and `svn add` of the *project.txt* file by hand (instead of using `svn rename` to move the file into your project directory)?
- 2) You've seen an example of using the `svn help` system to get information about subversion command line operations. Use `svn help` to determine which subversion command you would use to view all of the properties of a file. Which subversion command does this? What flag would you use to see the properties for a particular revision of a file?
- 3) As shown in the class lectures, `svn merge` is the correct way to undo changes to a file in order to get back to a previous version. If the current revision number of a file is 600, and you want to go back to revision 550, how would you invoke merge in order to do this?
- 4) Why is the `revert` command not the correct thing to do to undo changes back to revision 550? What does `revert` actually do?

**WARNING:** You are not done with this portion until you commit your answers to the repository. Have you committed your answers to the repository yet?

## Lab 02: Part 2, Outside Assignment Using a Subversion Repository

### Instructions

Exercises 2.4 and higher are to be completed in your own time outside of class, but are to be finished no later than the beginning of the next class period (Tuesday, February 5<sup>th</sup>).

### Exercise 2.4: Create Working Copies for Multiple Developers

In this part of the lab, you'll simulate the actions of two people editing a single file. To do that, you'll need to check out multiple working copies of your repository. This is not an unusual situation, even for a single developer. For example you could have a working copy on your laptop computer, and on your desktop at home.

However, we are going to simulate two separate developers working on the repository simultaneously, all in your home directory. It is perfectly fine to have multiple working copies of the same repository on the same machine, though it would normally be unusual to do this. We will call the two developers by the usernames *harry* and *ron*.

Change to your home directory and check out multiple working copies of your repository. Designate one working copy to be the one that the user *harry* is developing with, and the other one will be for *ron*. Do the following:

```
$ cd
$ svn co http://nisl.tamu-commer.edu/repo/csci553/yourusername harryrepo
$ svn co http://nisl.tamu-commer.edu/repo/csci553/yourusername ronrepo
```

I suggest you use the directory names for the working copies as indicated (and remember to use your actual nisl account username for *yourusername* in the `co` commands).

### Exercise 2.5: Harry creates a File

We will simulate the user *harry* creating and adding a file to the repository. Change to the *lab02* directory of *harry*'s working copy. Instead of creating a file to work with from scratch, I'll give you a file to use. Copy the file *lab02-spells.txt* from the *classfiles* directory to *harry*'s *lab02* working copy:

```
$ cp /home/csci553/classfiles/lab02-spells.txt .
```

Now have *harry* add the file *lab02-spells.txt* to the repository, and commit the file. Make sure you give a meaningful log message. Also for this and all future log messages for lab 2, indicate which user it is that is performing the action. Prepend this log message with "User *harry*:" (and likewise when *ron* is committing things, prepend with "User *ron*:").

### Exercise 2.6: Ron updates to current revision

At this point *harry* has modified the repository by adding a new file. *Ron*'s working copy is out of date. Use the update command as user *ron* in *ron*'s working copy in order to update his working copy to the most recent revision. If you do this correctly, you should see that the new file *lab02-spells.txt* will be added to *ron*'s working copy.

### Exercise 2.7: Non-conflicting Changes

We will now show what happens if the users *ron* and *harry* simultaneously make changes to the same file, but the changes do not conflict. As user *ron*, edit the *lab02-spells.txt* file. Change the title of the first spell (Title: Confundo Confundus Charm) to something simpler, for example call it "The Confusion Charm". DO NOT YET COMMIT YOUR CHANGES.

Now, change over to user harry's working copy and edit the *lab02-spells.txt* file this time as user harry. Have user harry modify the title of the second spell (Title: Expelliarmus, about ½ way down the file) to something simpler, for example rename it “The Wand Disarming Spell”.

At this point, users harry and ron have made simultaneous changes to the same *lab02-spells.txt* file in the repository. However the changes are non-conflicting. Subversion, and all modern version control systems, are smart enough to be able to merge non-conflicting changes such as these for the developers automatically.

Still as user harry, go ahead now and have harry commit his changes to the repository (Make sure you indicate in the log message that this is User harry:, and that he has simplified the name of the Expelloramous spell).

Now switch back over and become user ron. Go to rons lab02 working copy directory. If you do an `svn status`, what does subversion show as the status of the file? The file is shown as being modified, but no conflict has been noted. Go ahead and have ron commit his work, again remember to indicate that it is user ron that is doing the commit, and what he has done. What happens now? Did you expect this?

You should have gotten an error message like the following when ron tried to commit his now out of date file to the repository:

```
svn: Commit failed (details follow):
svn: Your file or directory 'lab02-spells.txt' is probably out-of-date
```

If you see a message like this it means you first need to bring your working copy up to date before you can commit something new. So update your repository and do a status:

```
$ svn update
G    lab02-spells.txt
Updated to revision 601.
$ svn status
M    lab02-spells.txt
```

Notice two things when you did this update/status as user ron. The file, *lab02-spells.txt*, was updated and a status code of G was displayed when it updated. Also, after the update, the file is still showing a M status, meaning it has been modified locally. The G actually means that a successful merge occurred when this file was updated from the repository. If you look at the contents of *lab02-spells.txt*, you will now see that it contains not only ron's changes to the title of the first spell, but also harry's changes that he committed have also been merged. Now as user ron if you go ahead and try to commit the file again, the commit should succeed this time.

At this point user ron has successfully merged and committed a newly changed revision of the file. Change back to user harry, and have harry do an update of his working copy, so that now both user's harry and ron will both have the same updated working copy of the repository.

### Exercise 2.8: Making Conflicting Changes and Resolving

Now we will simulate what happens when two users make a simultaneous change to a file that conflict with one another. As user harry, change the description of the first spell to read: “Causes the victim to become befuddled and confused.” (notice harry has simply switched the order of the words confused and befuddled in the description). Again DO NOT YET COMMIT YOUR CHANGE.

Now switch over to user ron's repository and edit the spells. Have ron change the description of the first spell to read "Causes the victim to become confused and muddled."

Now still as user ron, go ahead and have ron commit his changes to the file (don't forget to log the user who is making the changes, and describe what he did).

Change back to user harry and try and have harry commit his changes. You should again experience the same failure on the commit. So go ahead and have user harry update his repository. This time on the update what status code did you receive? What does the status code mean? If you do an svn status, you should now see that, unlike the previous example, the status of the spells file is listed as 'C'. This means that the file has a local change that conflicts with the current version in the repository (a conflicted change that could not be resolved automatically by subversion).

Do a directory listing of harry's lab02 working copy. You should notice a couple of new files that look something like this:

```
$ ls -al
total 32K
drwxrwxr-x 3 harry harry 4.0K Jan 30 14:31 .
drwxrwxr-x 5 harry harry 4.0K Jan 30 14:30 ..
-rw-rw-r-- 1 harry harry 2.4K Jan 30 14:31 lab02-spells.txt
-rw-rw-r-- 1 harry harry 2.3K Jan 30 14:31 lab02-spells.txt.mine
-rw-rw-r-- 1 harry harry 2.3K Jan 30 14:31 lab02-spells.txt.r603
-rw-rw-r-- 1 harry harry 2.3K Jan 30 14:31 lab02-spells.txt.r605
-rw-rw-r-- 1 harry harry 89 Jan 30 13:49 rollbacktest.txt
drwxrwxr-x 6 harry harry 4.0K Jan 30 14:31 .svn
```

Your revision numbers will differ, but you should have 4 similar *lab02-spells.txt* files. The file *lab02-spells.txt.mine* contains harry's changes (what harry was trying to check in before he discovered his copy of the file was out of date and in conflict). The file with the lower revision number contains the file before either set of conflicting changes were made, and the higher revision number contains the most recent version that is currently in the repository (e.g. the version that the dirty sneaky ron checked in before harry could successfully get his changes into the repository).

Edit the original file *lab02-spells.txt*. When subversion detects a conflict, it puts these conflict markers in the file to indicate where conflicts have occurred. The changes marked as "`<<<<<<< .mine`" are those that harry as made, and the changes marked with a revision number "`>>>>>.r764` for example" are those that the user ron made and checked in that were in conflict.

Go ahead and edit the file in any way you like to resolve the conflicts. Where there is an unresolvable conflict such as this, the developers must resolve the conflicts by hand. Of course your resolution should remove all of the conflict markers from the file. So make sure, before you resolve the conflict and check the file back in, that edit out the change markers.

When you have resolved the conflicts by hand, you must let subversion know that the version you have has been resolved by hand by the user. In this case, after user harry resolves the conflict, he should run the svn resolved command to reset the status of the file to be resolved:

```
$ svn resolved lab02-spells.txt
Resolved conflicted state of 'lab02-spells.txt'
$ svn status
M      lab02-spells.txt
```

After doing this, as you can see, the status of the file will be set as locally modified. The user harry should now go ahead and commit his resolved version of the spells file to the repository (don't forget the log messages).

### Exercise 2.9: Finishing Up and Checking you Work

After finishing the conflict resolution, take a look at your log file for the spells file you have been modifying. If you have completed the steps 2.4-2.8 successfully, you should see a log for the file that looks something like this:

```
$ svn log lab02-spells.txt
-----
r606 | harry | 2008-01-30 14:43:56 -0600 (Wed, 30 Jan 2008) | 3 lines
User harry: resolved conflicting descriptions of 1st spell.

-----
r604 | harry | 2008-01-30 14:29:59 -0600 (Wed, 30 Jan 2008) | 3 lines
User ron: changed description of first spell.

-----
r603 | harry | 2008-01-30 14:22:32 -0600 (Wed, 30 Jan 2008) | 3 lines
User ron: modified title of first spell to simplify.

-----
r601 | harry | 2008-01-30 14:12:19 -0600 (Wed, 30 Jan 2008) | 2 lines
User harry: Changed name of spell to "The Wand Disarm Spell"

-----
r598 | harry | 2008-01-30 13:57:48 -0600 (Wed, 30 Jan 2008) | 2 lines
User harry: Added spells to the spell book. Done for Lab 2 Exercise 2.5
-----
```

When you are satisfied that you have successfully completed the lab, you can go ahead and remove the working copies of ron and harry (use the `rm -rf` command to recursively remove directories, but be careful that you don't accidentally remove something that you don't want to.)