THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

CSSE2002/7023
SVN in IntelliJ Guide
Version 1.0.2

Written by Emily Bennett and Brae Webb

1 What is SVN?

Subversion (SVN) is a source version control system aimed at simplifying the software development process. Source version control is used to track the changes that are made to source files in a project.

While source version control is an important topic to cover, there is a significant amount of theory behind it that goes beyond what is required for this course. As such, we will be focusing on the practical aspects of SVN, rather than what's happening behind the scenes.

1.1 Why do we use version control?

There are many benefits of learning to use version control, some of which include:

- The ability to work on your code anywhere (as your repository can simply be checked out on a new machine).
- The ability to go back to a working copy if something breaks (this is the most efficient method of backing up your code).
- Learning industry tools and techniques.

1.2 What is a repo?

In SVN, your work is contained within something called a repository, or repo for short. It is effectively the same as having a directory/folder, except it is not stored on your computer. This repo is where you will store your files, and track changes to those files. At UQ, we provide a server which hosts this "directory/folder", allowing you to keep your files safe.

1.3 How do I install it?

1.3.1 Linux

For a Linux installation, you will find instructions on the offical Apache website: https://subversion.apache.org/packages.html.

1.3.2 Mac

For MacOS, an older version of SVN may already be installed (try typing `svn` into a terminal — if it runs without error, then SVN is installed). You can also use Homebrew to install SVN (`brew install subversion`). Otherwise, have a look on the Apache website for installation instructions: https://subversion.apache.org/packages.html.

1.3.3 Windows

For Windows installations, any SVN client will work, however the SlikSVN command line client (https://sliksvn.com/download/), or TortoiseSVN (the client installed on lab computers — https://tortoisesvn.net/), are recommended. You may need to restart your PC after installing you SVN client in order to use it in Command Prompt (cmd) or PowerShell.

2 But how do I ...

Now that we know what SVN is, we need to learn how to use it. This guide is built so that you can follow along at home, using a repo which has been set up for you.

This guide makes use of the IntelliJ SVN interface. If you would prefer to use a command line interface, please see the `Command Line SVN` guide on Blackboard.

2.1 Checkout my repo?

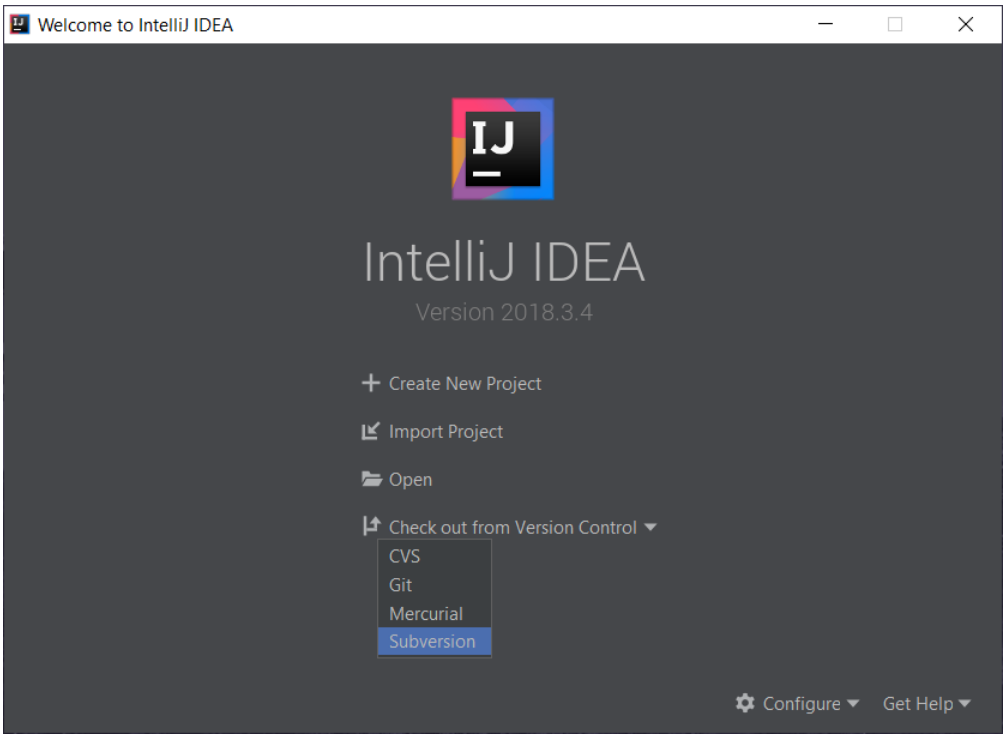
At the start of our journey, we will need a repository to work with. In CSSE2002/7023, you will generally be given a URL to your repository in the following form:

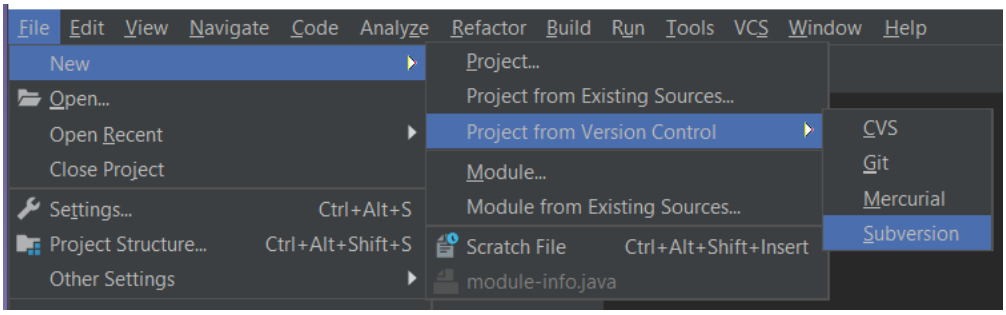
`https://source.eait.uq.edu.au/svn/csse2002-s1234567/trunk`.

Where `s1234567` is your student number. Note that the above URL containing `csse2002` should be used by both CSSE2002 and CSSE7023 students.

For the following guide we will use the `opportunity` repository that has been populated for you.

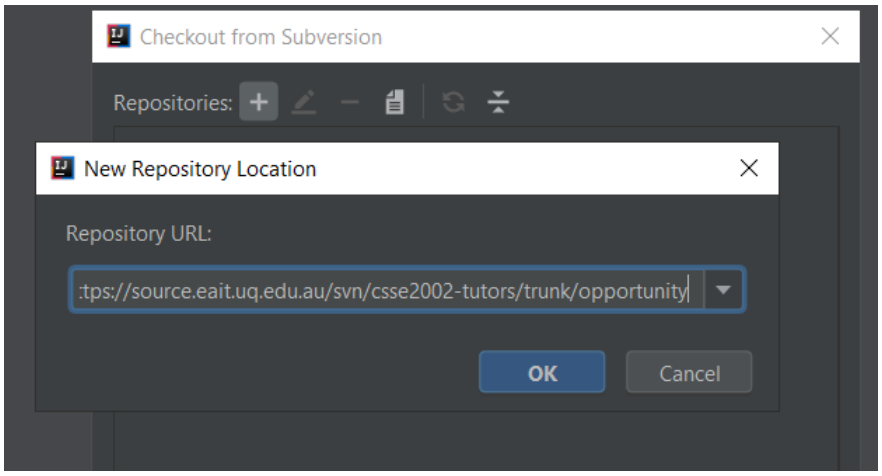
In order to access this repository in IntelliJ, we either need to select `Check out from Version Control` on the main IntelliJ launch screen, or select `File` > `New` > `Project from version control` while an existing project is open.





In either case, we will then need to select `Subversion` from the list of options. A new window will pop up with a (likely currently empty) list of repositories you have used before. Pressing the plus (`+`) button will allow you to enter a new repository location. For this guide, you should use `https://source.eait.uq.edu.au/svn/csse2002-s1234567/trunk/opportunity`, but for your assignments, use the link given to you in your task sheet.

Note: Replace s1234567 with your student number.



Once you have added the repository, select the newly-added correct entry from the list, and click `Checkout`. A new window will then pop up asking you to select a destination directory. This is the the location on your machine where the repository will be stored. If you are on a lab machine, pick a location somewhere on the `H:` drive, otherwise just pick a sensible location that you will be able to find again later.

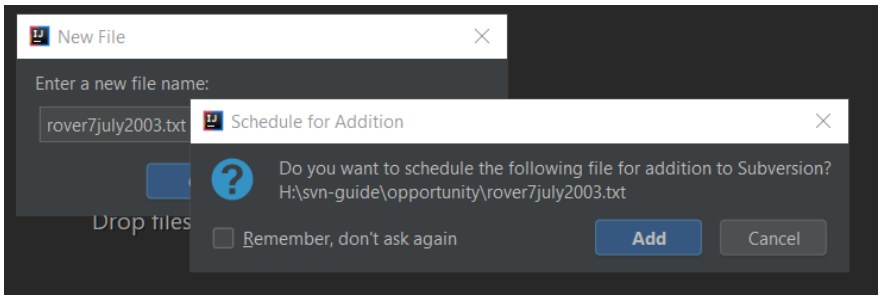
Pressing `OK` will take you to another screen with checkout options — all the default options should be fine as they are, so you can just press `OK` again. *If you get an error at this point, please refer to troubleshooting in section 5*

Leave `1.8 format` selected on the next screen, select `OK`, and then login on the following screen using your UQ username (sXXXXXXX) and password. You can optionally select to save your credentials so that you do not need to log in every time you make commits. Press `OK`, and then `Yes`, and then `Next` on the following screens, until finally pressing `Finish`, at which point your project should open in IntelliJ.

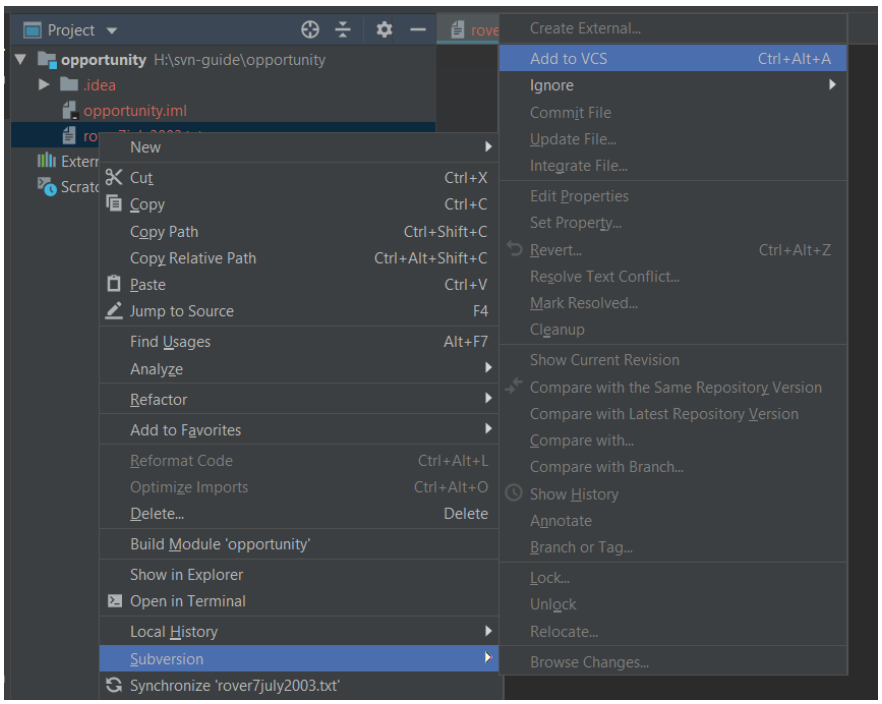
2.2 Add a file?

This section involves creating a file in the opportunity folder, and instructing SVN to track this file. We will be using text files for this guide, however the same process applies to Java source files. Right click the `opportunity` folder and select `New` > `File`, then type in `rover7july2003.txt`.

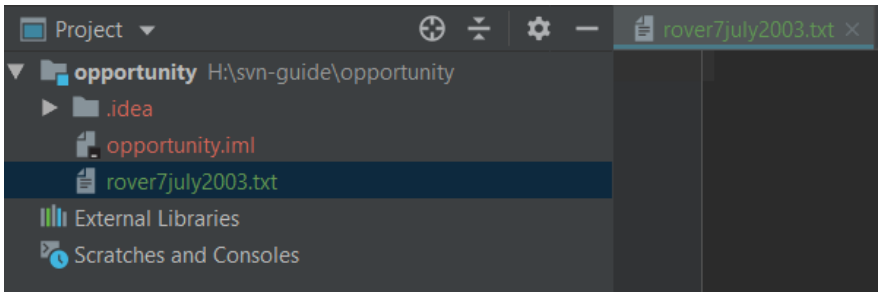
Selecting `OK` should cause a window to appear asking whether you want to add this file to be tracked by Subversion. Select `Add` here, and you will notice that the filename appears green in IntelliJ.



If, for any reason, you select `Cancel` instead of `Add`, files can still be added to your repo at a later stage. Simply right click the file, and select `Subversion` > `Add to VCS`, which should also cause the filename to turn green.



IntelliJ uses colours to indicate the status of files and directories — red indicates that files are not being tracked at all (not even in your local repository), while green indicates that files have been added for tracking. Note that files coloured green *are not present in your remote repository, and are thus not accessible by course staff for marking yet, in the context of assignments.*



You can then add the following contents to the file:

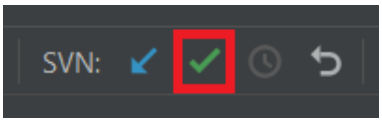
```
0 Project: Opportunity
1 Destination: Mars
```

*Note: you **should not** add or commit any IDE-specific files, such as the `.iml` file or `.idea` directory.*

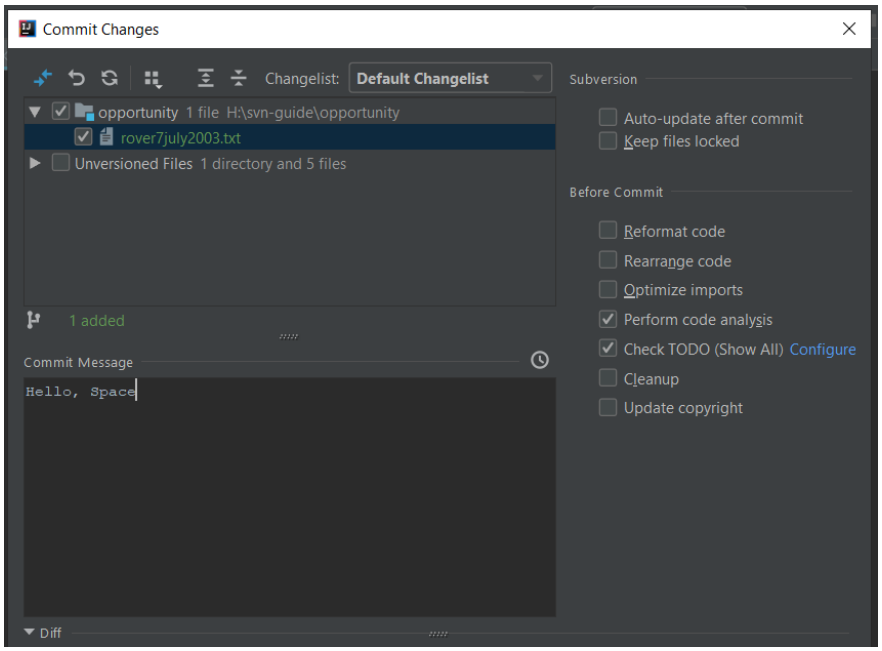
2.3 Commit a file?

Our local copy of the repo currently has the `rover7july2003.txt` file, however our remote server does not. This means that course staff are still unable to view these files (which is important to remember for submission, as anything missing from the server is effectively missing from your submission). To move move the file to the server, we need to `commit` it, along with a commit message giving a short description of the changes which have been made in this revision.

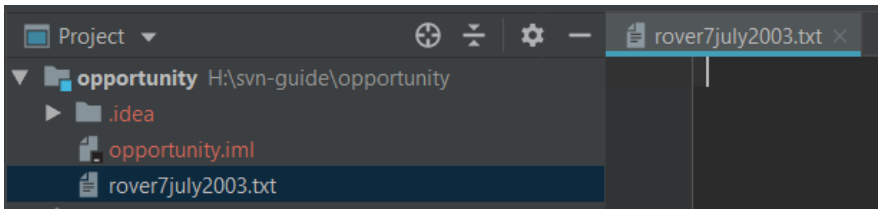
Committing a file can be done in multiple ways, including right clicking the filename and selecting `Subversion` > `Commit file...`, or simply selecting the green checkmark at the top right corner of the screen.



Either of these options should bring up the commit window. This window will allow you to choose which files you want to include, and provides space for a commit message.



Now we have commited our changes to the central repository stored at UQ's server, and the colour of the filename should have changed to the ordinary grey colour.



This shows that we have no pending changes locally — our local repository is up to date with the remote repository, and everything we can see is what the course staff can see too).

Now add a few more files (`rover25Jan2004.txt`, `rover2011.txt`, `rover12June2018.txt`), each with the following contents:

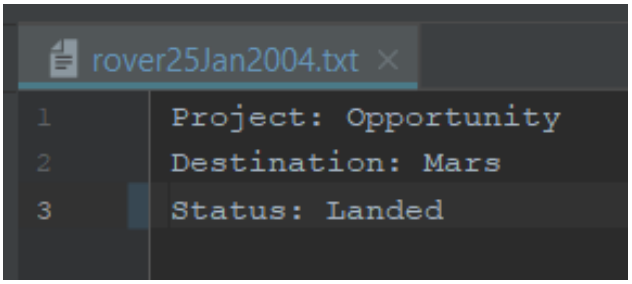
```
0 Project: Opportunity
1 Destination: Mars
2 Status: ?
```

Add them so that they are being tracked, and then commit them with the commit message "setup".

2.4 Making changes?

Adding new files is useful, but we also need to be able to record changes to files that are already being tracked. To get started, edit the `rover25Jan2004` file and the `Status:  ?` line to instead read `Status:  Landed`.

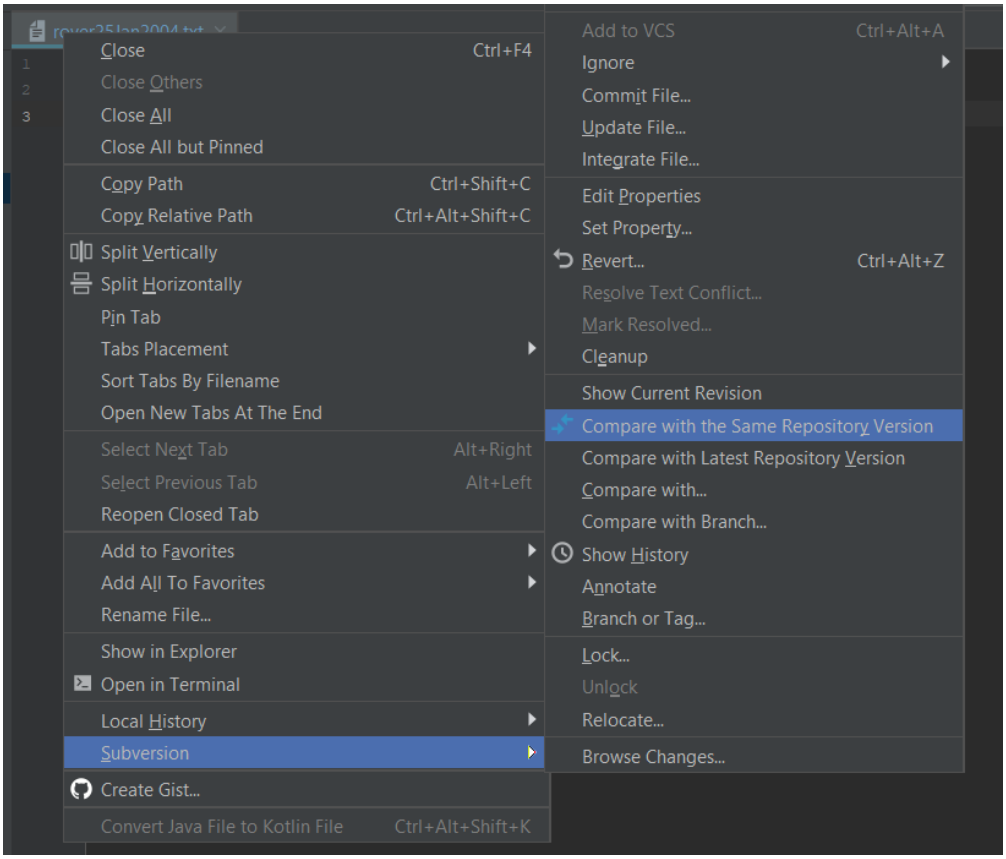
You will notice that the filename has changed to a blue colour. This is how IntelliJ indicates that there are uncommitted changes in your local repo that are not on the server.

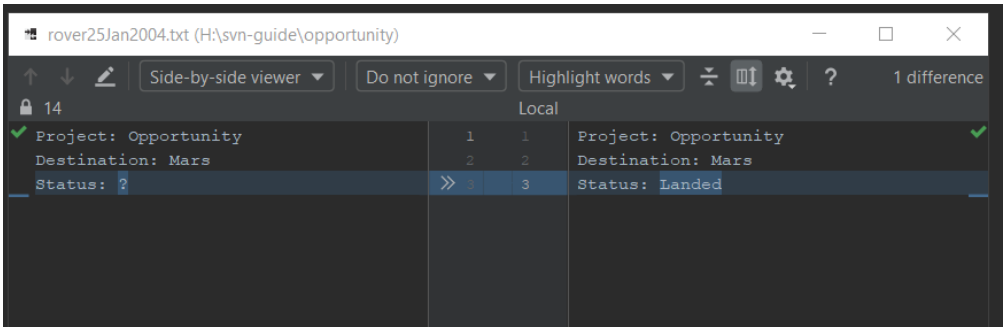


We could now commit the file, however let's first have a look at the changes we have made to ensure that we have not made any mistakes/accidental changes.

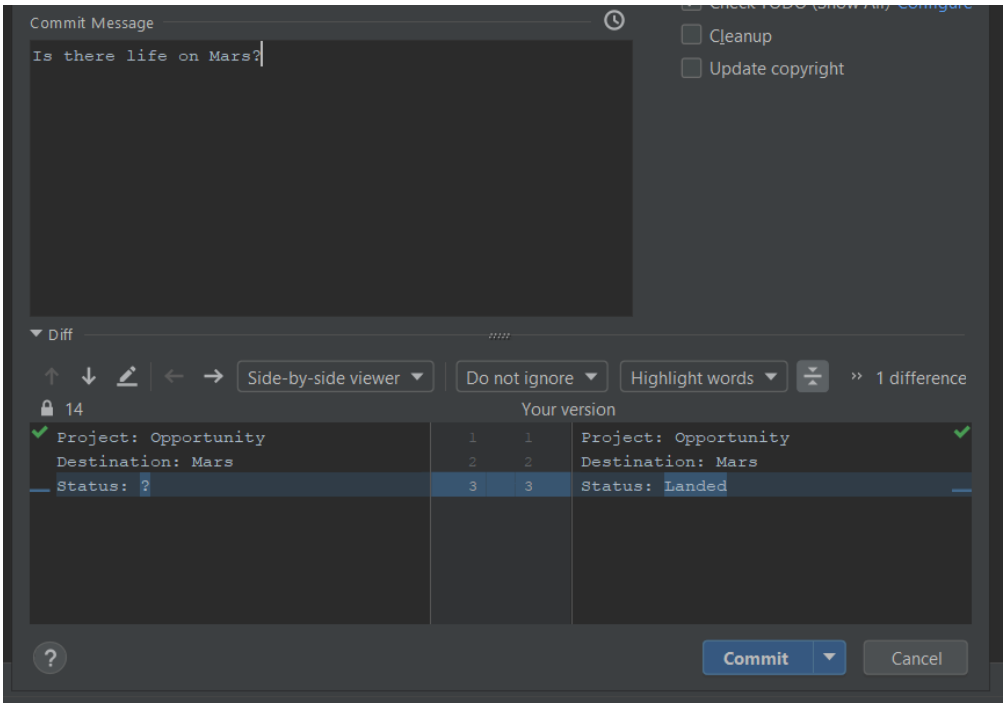
2.5 View changes to a file?

If you want to see the changes you have made to a file since the last commit, there are multiple ways to view a diff. One is to right click a file and select `Subversion > Compare with the Same Repository Version` (the option with the two blue arrows). The other way is to select the `Version Control` panel from the bottom left hand corner, and select the same icon with two blue arrows.





In addition to this, when going to commit an existing file that has had changes made to it, you will notice that the commit window now has a diff at the bottom of it, showing the differences between the previously committed version (on the left) and your local version (on the right).



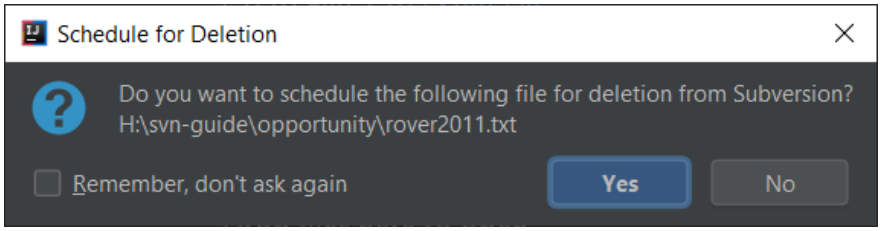
We can now commit the file, since we are happy with the changes.

2.6 Remove a file?

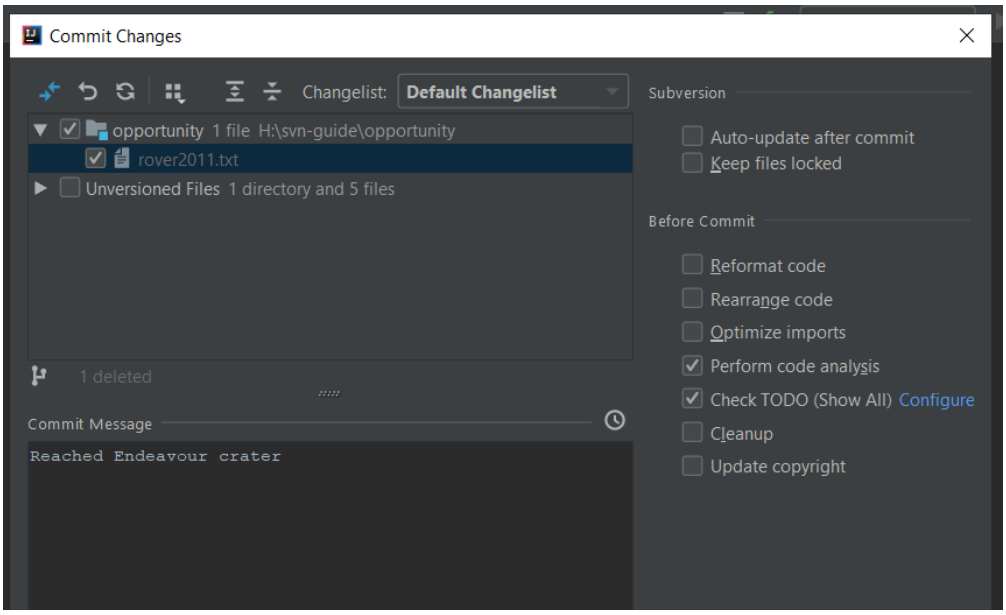
So far we have checked out a repo, added files, and commited updates to those files. Sometimes, however, we need to remove files from our repo.

Say, for example, we decide that we no longer need the `rover2011.txt` file — this requires telling SVN to stop tracking it.

We can delete the file as usual, either by right clicking it, or selecting it and clicking the delete key on the keyboard. After confirming deletion, a window should pop up asking if we want to schedule this file for deletion from Subversion.

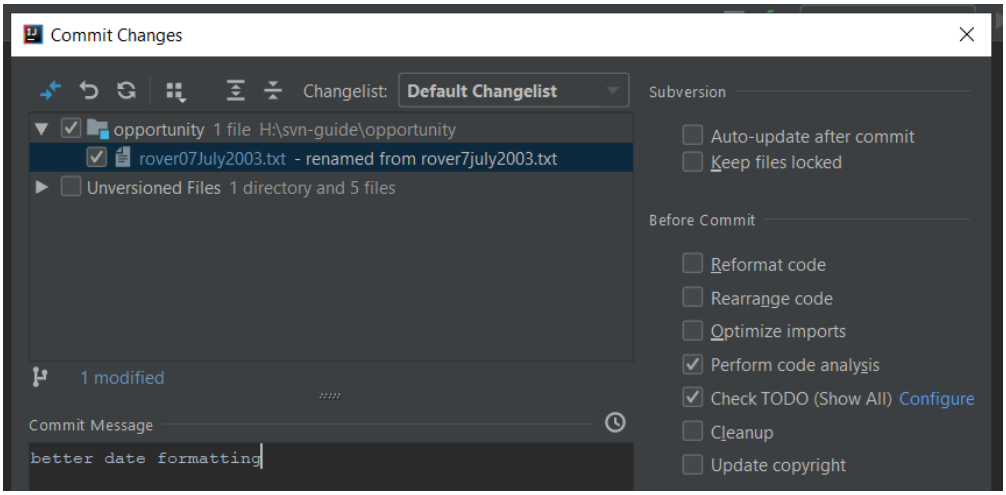


Clicking `Yes` will remove it from our project structure on the left hand side, but note that this has only occurred in our local copy — we still need to commit these changes in order for the server to reflect them as well. This means that the file we just deleted it is still visible to course staff when it comes time for marking, so be aware of this.



2.7 Rename a file?

As with deleting a file, you can rename files as you normally would in IntelliJ (right click, and then `Refactor > Rename...`). After renaming the file, you should see it turn blue, indicating uncommitted changes. Simply commit the file as you would normally after making changes, and you will note that IntelliJ's commit window informs you that the file has been renamed.



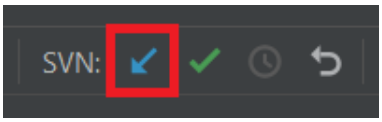
2.8 Update my repo?

Since SVN allows us to checkout a repo multiple times on multiple machines, we could end up in a scenario where two different local copies of our repo have two different states (for example, you committed changes to your assignment on your home computer, and the lab computer is now out

of date). You could, in theory, just checkout a new copy of the repo, however to avoid this, we can use `svn update`.

Note: For this demo, a change was made to another local copy of the repo, and was commited to the remote repo.

To update our repo, we can right click on the project name and select `Subversion` > `Update directory...`, or simply select the blue arrow pointing down and to the left (next to the green commit checkmark) at the top right-hand corner of the screen.

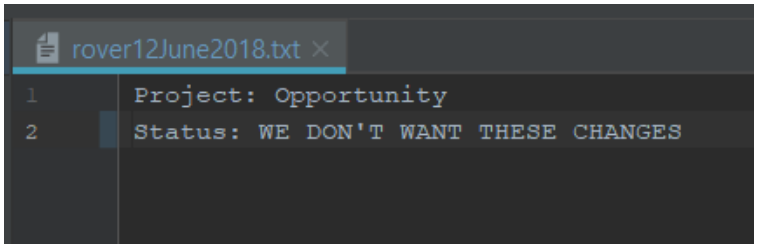


An update window will pop up, which you should be able to leave set to all the default values and select `OK`. This local copy of your repo should now be up-to-date with the most recent commit we made on any other machine.

2.9 Revert a file's changes?

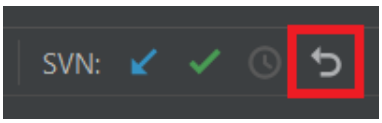
One of the benefits of version control is that it acts as a backup of our code — if at some point we realise we have been heading in the wrong direction with our code, we can revert back to a point we knew the code was working.

In order to demonstrate this, make some changes to the `rover12june2018.txt` file at random (for example, delete a line). These changes made to the local copy should be visible by viewing a diff of the file.



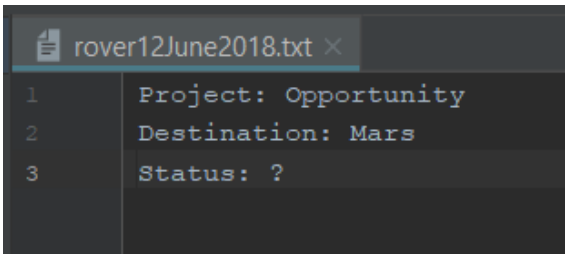
We can now use `svn revert` to restore this file to what it was before.

Reverting in IntelliJ can either be done by right clicking the file being reverted and selecting `Subversion` > `Revert...`, or by pressing the white back arrow on the top right-hand corner of the screen.



This will pop up a screen allowing you to choose which files should be reverted, and pressing `Revert` should cause the file to return back to its previous state.

The diff now shows that our local copy is up-to-date with the most recent commit.



3 A note on commits

We now know how to commit files, however an important aspect of version control is knowing when to commit. You should try to stick to the following guidelines to enhance your version control experience[1]:

1. Commit your change each time significant progress has been made — each commit should represent a step forward towards the completion of your project.
2. Commit often — if you commit after several unrelated changes have been made, going back (reverting) to the exact version of code you want can become difficult. This is because you may find yourself in a position where one version of code doesn't have the changes you want, and the following version has additional unwanted changes.
3. Don't commit broken code — committing code that is broken or doesn't work as expected can cause significant issues if you need to revert to a previous working version. It is recommended that you fix the issue before you commit changes.
4. Test before you commit — closely linked to the above point, is test your code before you commit. Often you think you've only changed a line or two then you commit it and now your code doesn't compile, or the new changes have caused bugs in unexpected places.

Commit messages are also important — you should try and make these messages as informative as possible, so you can later go back and find the correct version you are looking for.

3.1 Sanity Checks

While you should be able to tell what is and isn't in your remote repo through a web interface, if you'd like to double check this prior to submission, replace 'svn' with 'viewvc' in your repository's url.

[1] Note that information in this section is taken from UQ's Git edX course.

Example:

Repository: https://source.eait.uq.edu.au/svn/csse2002-s1234567
ViewVC: https://source.eait.uq.edu.au/viewvc/csse2002-s1234567

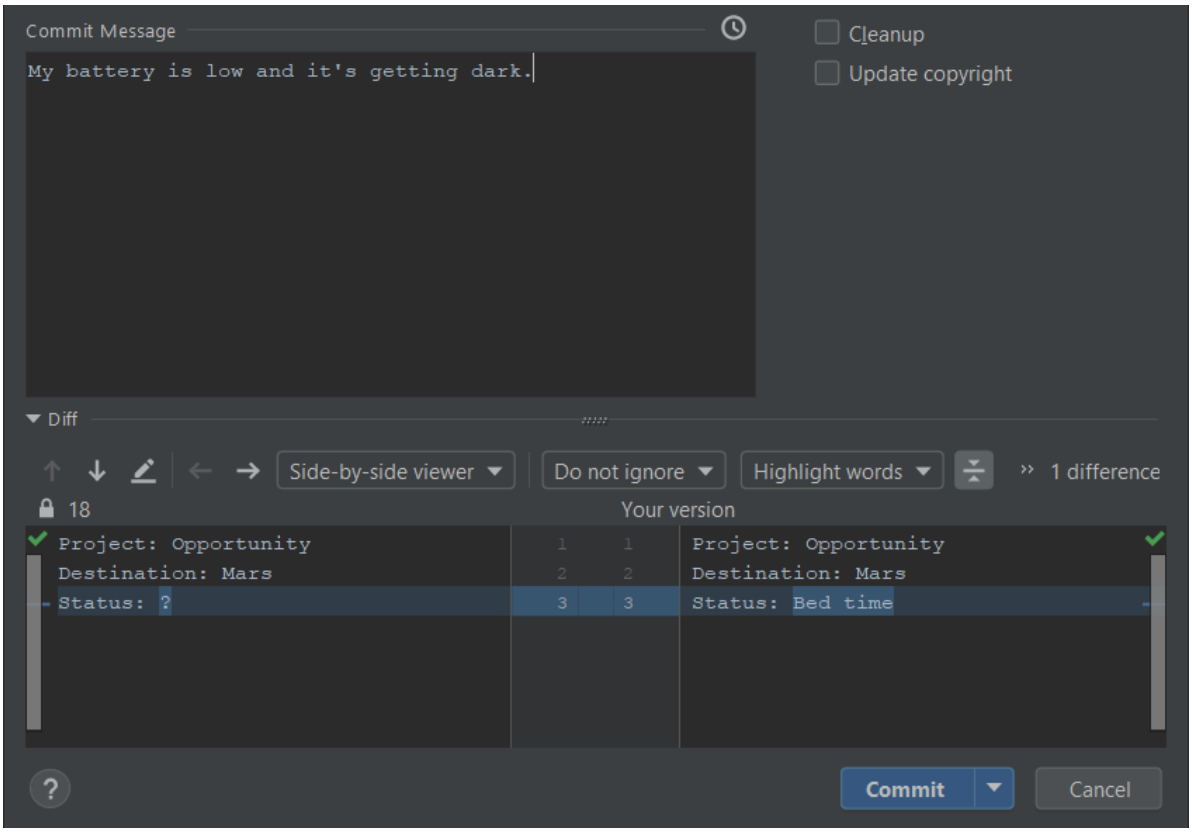
3.2 Get more help?

If you would like to learn more about SVN and the things it allows you to do, the following resources may be helpful.

- https://www.jetbrains.com/help/idea/using-subversion-integration.html
- https://subversion.apache.org/docs/
- http://svnbook.red-bean.com/
- https://xkcd.com/2111/

4 Goodbye!

Change the 'Status: ' line of the `rover12June2018.txt` file to be 'Bed time', and make one final commit.



Thanks for playing, feel free to give feedback to the course staff if your found any issues or have any thoughts about this document.

5 Troubleshooting SVN Setup

If you encounter issues whilst checking out your repository, it may be because the path to SVN is not set up in IntelliJ.

Navigate to the IntelliJ settings (using either `Configure > Settings` from the launch screen, or `File > Settings` from within the IDE), and select `Version Control > Subversion` from the menu on the left-hand side of the window.

The first text entry box should contain the filepath to where SVN is installed on your machine. If it does not have a path, or does not have the correct path, you should navigate to the correct location of the `svn.exe` file. On the lab computers, for example, the filepath should be something similar to `C:\Program Files\TortoiseSVN\bin\svn.exe`. Then press `OK`, restart the IDE, and restart the process from section 2.1.