PHAS3459
Scientific programming using object-oriented languages

Module 1: Version Control

Ben Waugh & Simon Jolly

What Is The Point Of Version Control?

- Most of you will – sadly – have lost data at some point in the past…
  - Disks crash.
  - Networks fail.
  - Documents become corrupted.
- The solution: regular backups! Make sure you always have more than one copy of **ALL** your important data.
  - But backups only store a single copy of your most recent document. They should also be more difficult to access to make them more difficult to destroy!
  - Also, when many people are working on the same file, having a way of tracking incremental changes becomes very important.
- The simple answer: create a new file every time you make significant changes:
  - WorkingDocument_v1.doc
  - WorkingDocument_v2.doc
  - WorkingDocument_v3.doc
  - …

What Is The Point Of Version Control?

- Now you have a (simple) method of keeping track of all changes! And you can also “undo” changes that you don’t like by going back to an earlier version of the file.
- You also have a significant amount of additional “book-keeping” to do:
  - All changes are tracked “by hand”, particularly if the files are just simple text files without additional tools for tracking changes, such as Word.
  - Disk space quickly gets eaten up as files become larger.
  - Filing becomes more difficult if you store all old versions of the file in the same directory.
- Wouldn’t it be great if there was a more sophisticated way of doing this…?

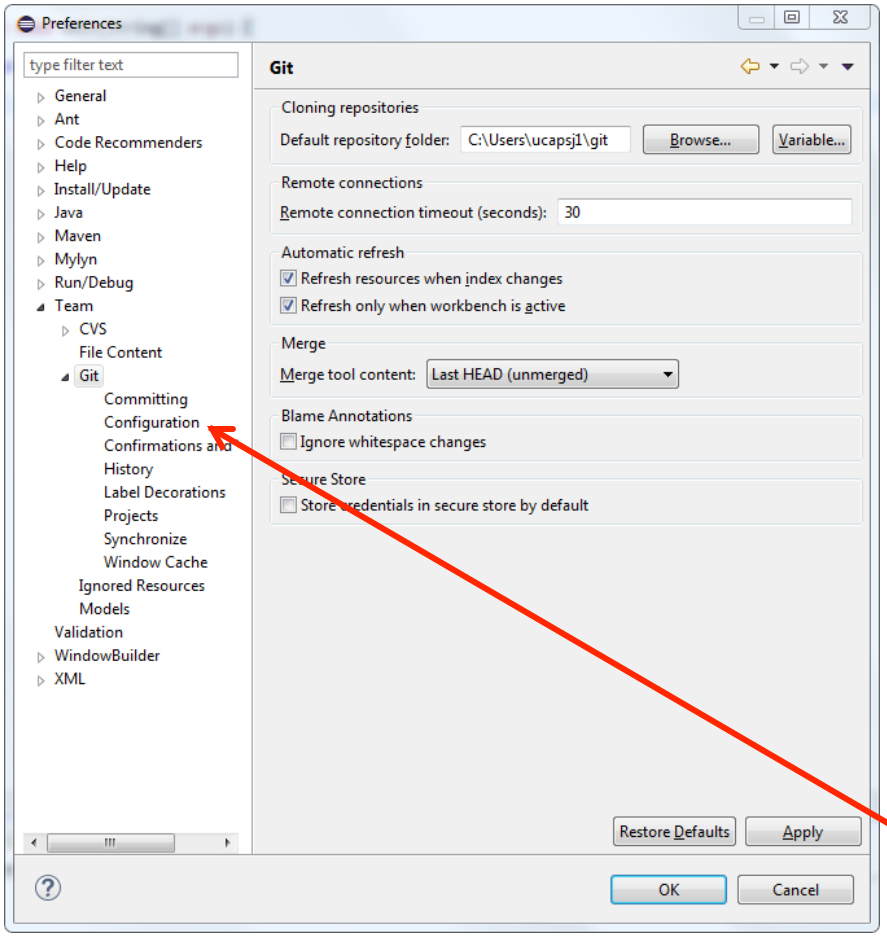
Enter Version Control Systems

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- A Version Control System (VCS) allows you to:
  - revert files back to a previous state;
  - revert the entire project back to a previous state;
  - compare changes over time;
  - see who last modified something that might be causing a problem (in the case of projects with more than one editor);
  - who introduced an issue and when, etc.
- The VCS is essentially a simple form of database that records all your files and the incremental changes to them.
- There are a number of different modern VCS's, the most popular being Git and Mercurial (Hg).
- We will be using Git, since Eclipse includes its own built-in version (EGit).
- The course coordinators use a Git repository to keep track of the course notes…

Some Git Terminology

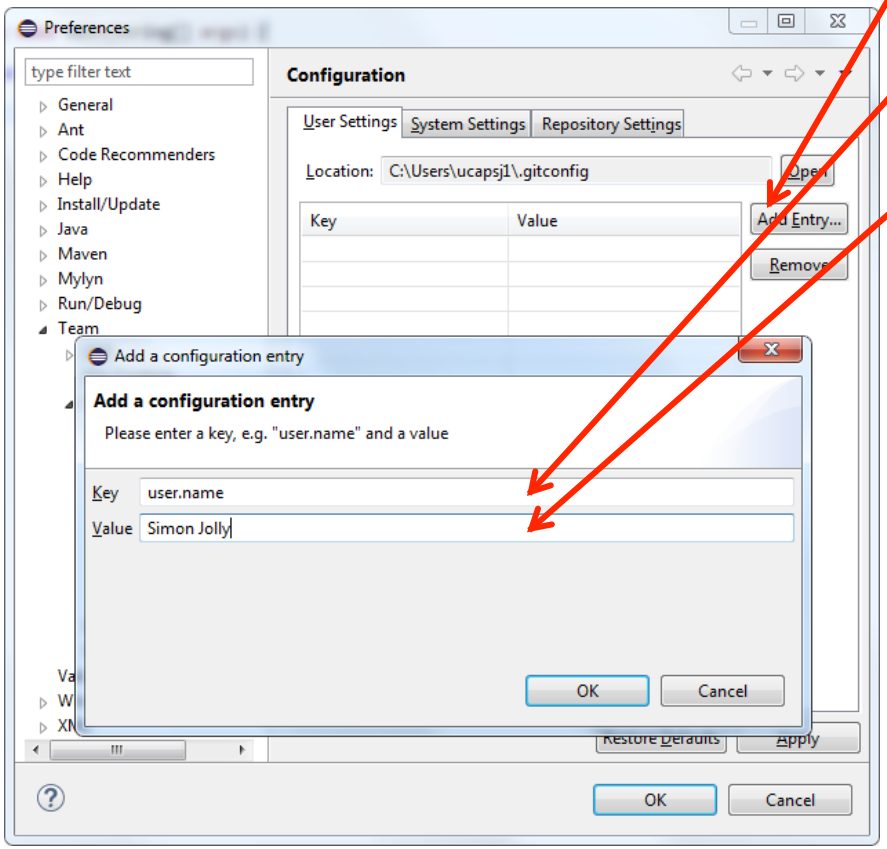
- **Repository.** The database of all your files and their incrementally recorded changes is called the **Git Repository**. This is normally stored alongside the original files in a ".git" directory (you'll be able to choose this location later).
- **Commit.** The term "commit" is used in two ways:
  - Saving a snapshot of your files (or a subset of them) in their current state is referred to as **committing** them to the repository.
  - Each one of these saved snapshots is called a **commit**. A commit, or "revision", is an individual change to a file (or set of files). It's like saving a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by whom.
  - Commits usually contain a commit message which is a brief description of what changes were made.
- **History.** This is the list or "timeline" of all the commits you've made. You can freely restore, or **"Check Out"** any of the commits from your Git history. This will restore the files to the state they were in when you made that commit.
- **HEAD.** Git's way of referring to the currently checked out commit. If you use some tool to visualise your Git timeline, the commit you have checked out will be indicated by HEAD.

Setting Up EGit: Git Within Eclipse



- Most Git settings and options are accessed within the “Team” menu: this is because Git is normally used to allow several people to work on the same set of files simultaneously.
- Open the Preferences window and select “Team → Git”.
- Select “Configuration”.

EGit Configuration

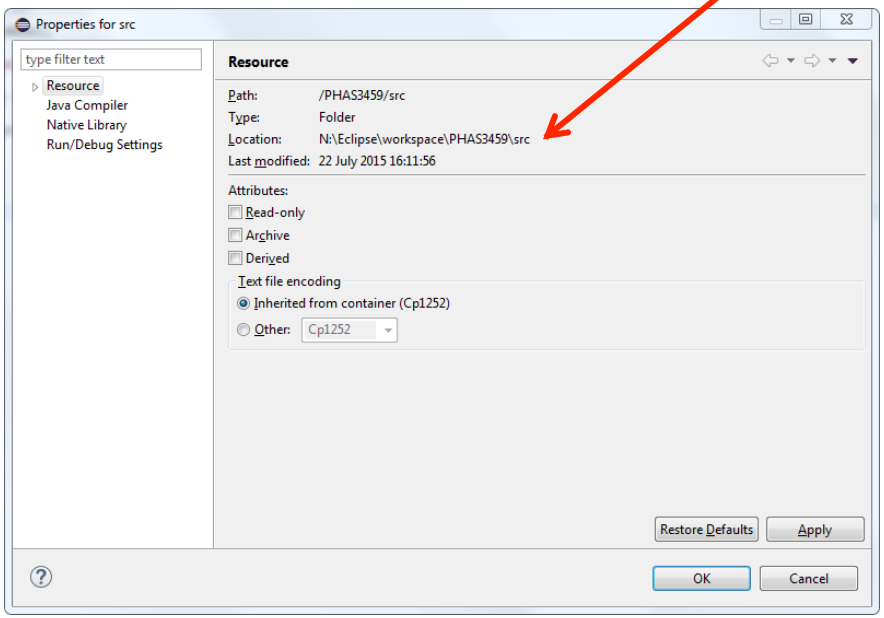


- Select “Add Entry” in the top right hand corner.
- Under “Key”, enter “user.name”.
- Under “Value”, enter your name.
- Select “OK”.
- Repeat this process with a Key of “user.email” and enter your UCL email address under Value.
- Your name and email are used to uniquely identify each commit you make if you share your repository with other people.

Existing Files

- Check the location of your existing source files.
- Right-click “src” under “PHAS3459” and click “Properties”…
- Alternatively you can select “PHAS3459 → src” and click File → Properties…

Source File Location

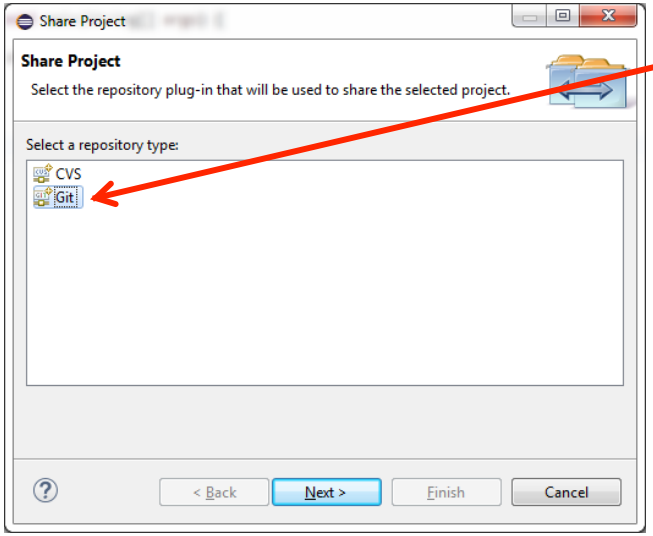


- The location of the Java source files is given next to “Location” under the “Resource” menu.
- Make a note of this! This is where your Java source files are stored (and should be somewhere like N:\Eclipse\workspace\PHAS3459\src).
- You will store your EGit repository alongside your workspace, not inside it.

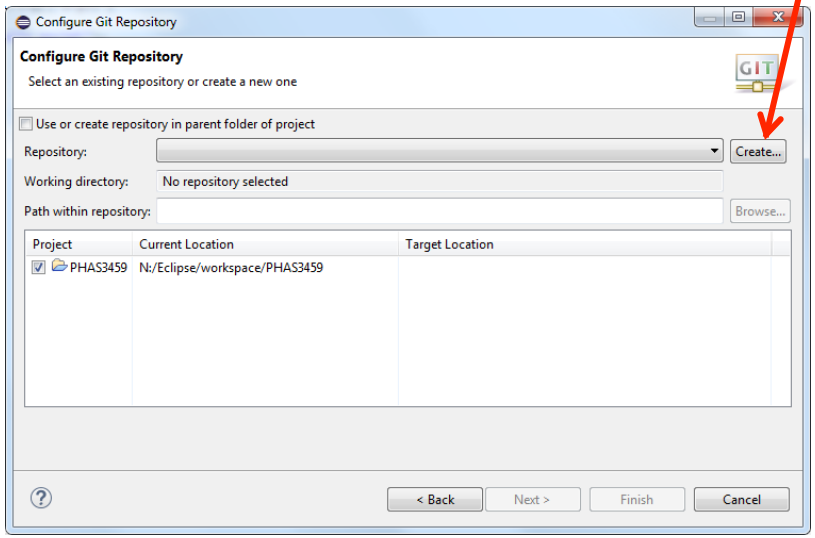
Preparing For Version Control

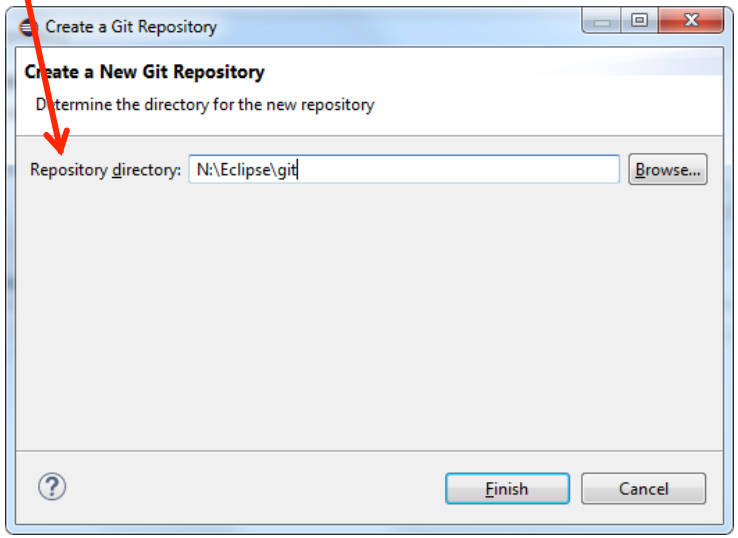
- Version Control in Eclipse is almost all found under the various "Team" menus.
- Right-click "PHAS3459" and select "Team → Share Project".

Setting The Repository Location

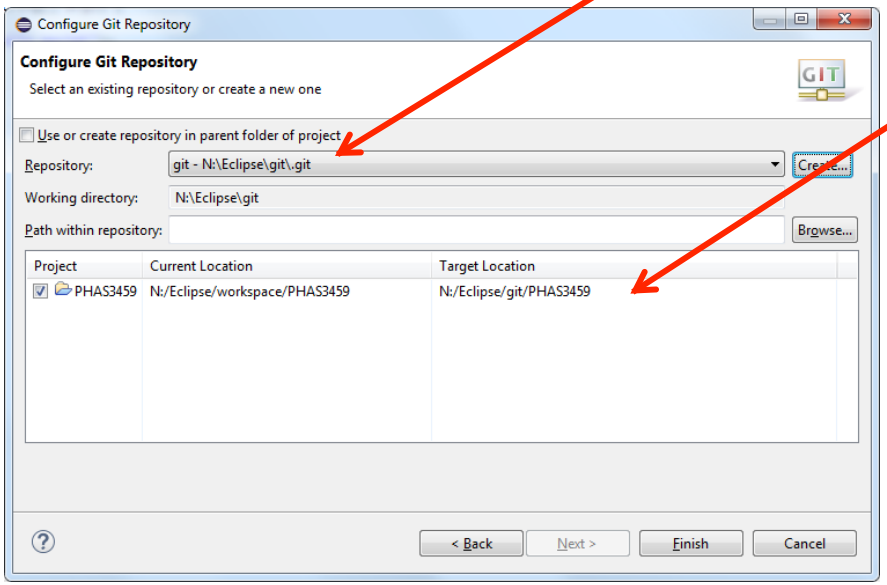


- If the repository selection window appears, select “Git”.
- In the configuration window, click “Create…” to create a new directory for the Git repository.
- Enter in the directory to store the new Git repository: if this doesn’t exist Eclipse will create it for you.
- We would suggest putting it next to your workspace, so if your workspace is stored in N:\Eclipse\workspace, choose N:\Eclipse\git.



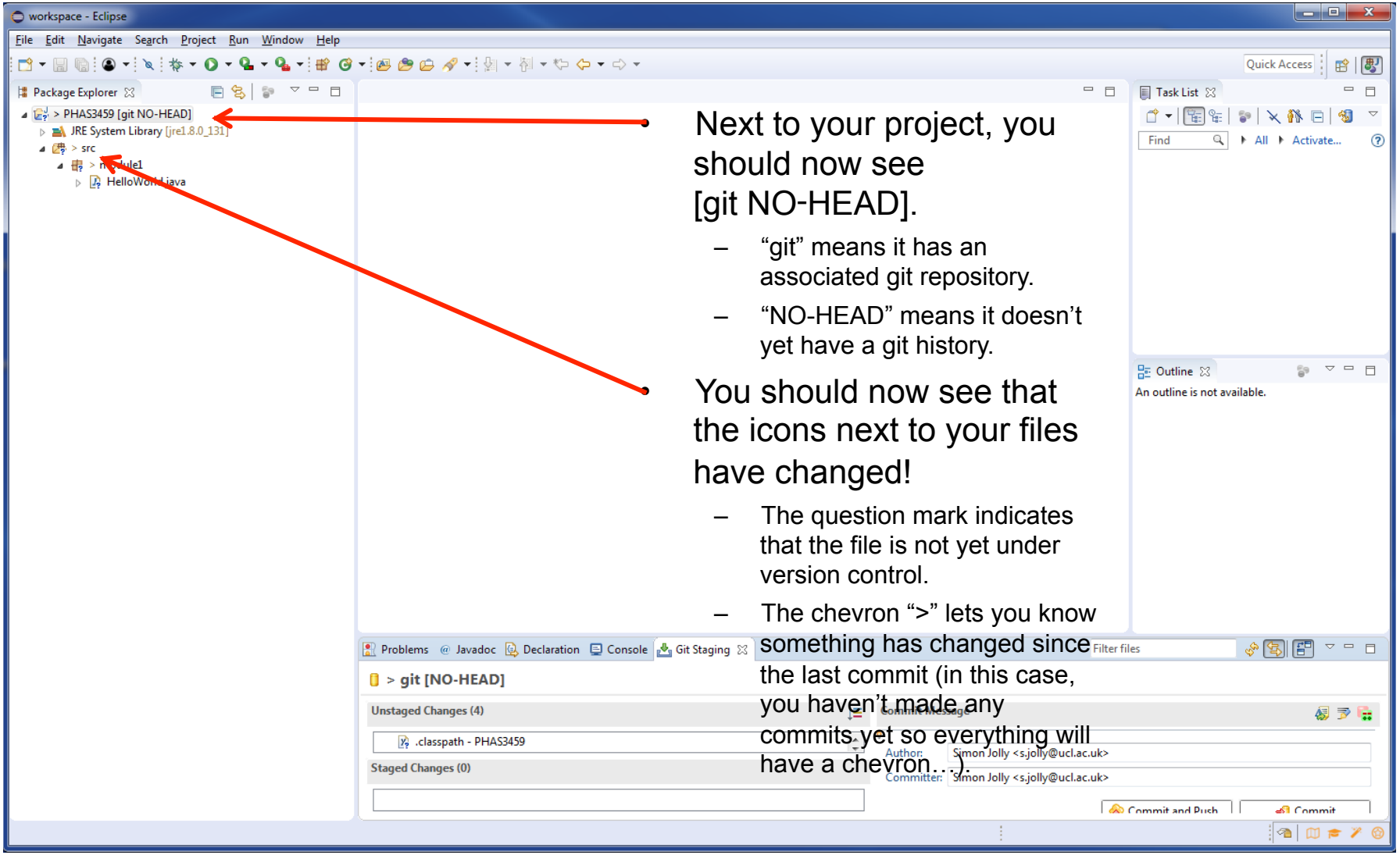


Creating The Repository



- The repository field should now show you the directory that contains the repository.
- The repository itself is stored in the ".git" sub-folder.
- Note that once the repository is created, your source files will be moved from "Current Location" to "Target Location".
- Click "Finish" to create the repository.
- Remember the Target Location you chose for your Git repository: **this is where your source files will be stored from now on**.

Eclipse Under Version Control



Next to your project, you should now see [git NO-HEAD].

- "git" means it has an associated git repository.
- "NO-HEAD" means it doesn't yet have a git history.

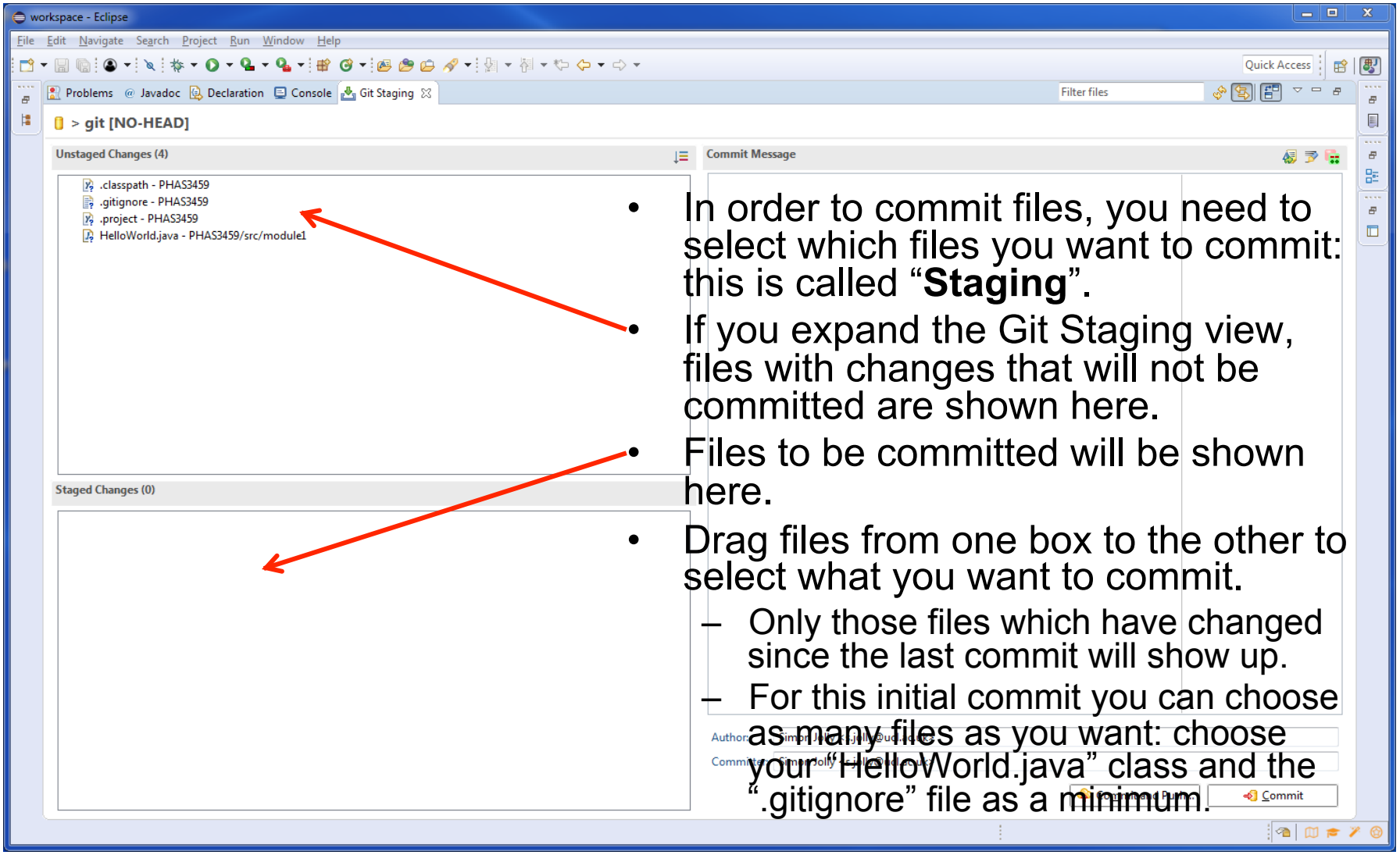
You should now see that the icons next to your files have changed!

- The question mark indicates that the file is not yet under version control.
- The chevron ">" lets you know something has changed since the last commit (in this case, you haven't made any commits yet so everything will have a chevron…).

The First Commit

- Now we need to choose which files to track with Git.
- We then need to commit those files to the repository.
- Both steps are accomplished in one by right-clicking on the project and selecting “Team → Commit”.

Staging Files



- In order to commit files, you need to select which files you want to commit: this is called "**Staging**".
- If you expand the Git Staging view, files with changes that will not be committed are shown here.
- Files to be committed will be shown here.
- Drag files from one box to the other to select what you want to commit.
  - Only those files which have changed since the last commit will show up.
  - For this initial commit you can choose as many files as you want: choose your "HelloWorld.java" class and the ".gitignore" file as a minimum.

Committing Files

workspace - Eclipse

File Edit Navigate Search Project Run Window Help

Quick Access

Problems Javadoc Declaration Console Git Staging

Filter files

> git [NO-HEAD]

Unstaged Changes (0)

Staged Changes (4)

- .classpath - PHAS3459
- .gitignore - PHAS3459
- .project - PHAS3459
- HelloWorld.java - PHAS3459/src/module1

Commit Message

```
First version of HelloWorld class.
```

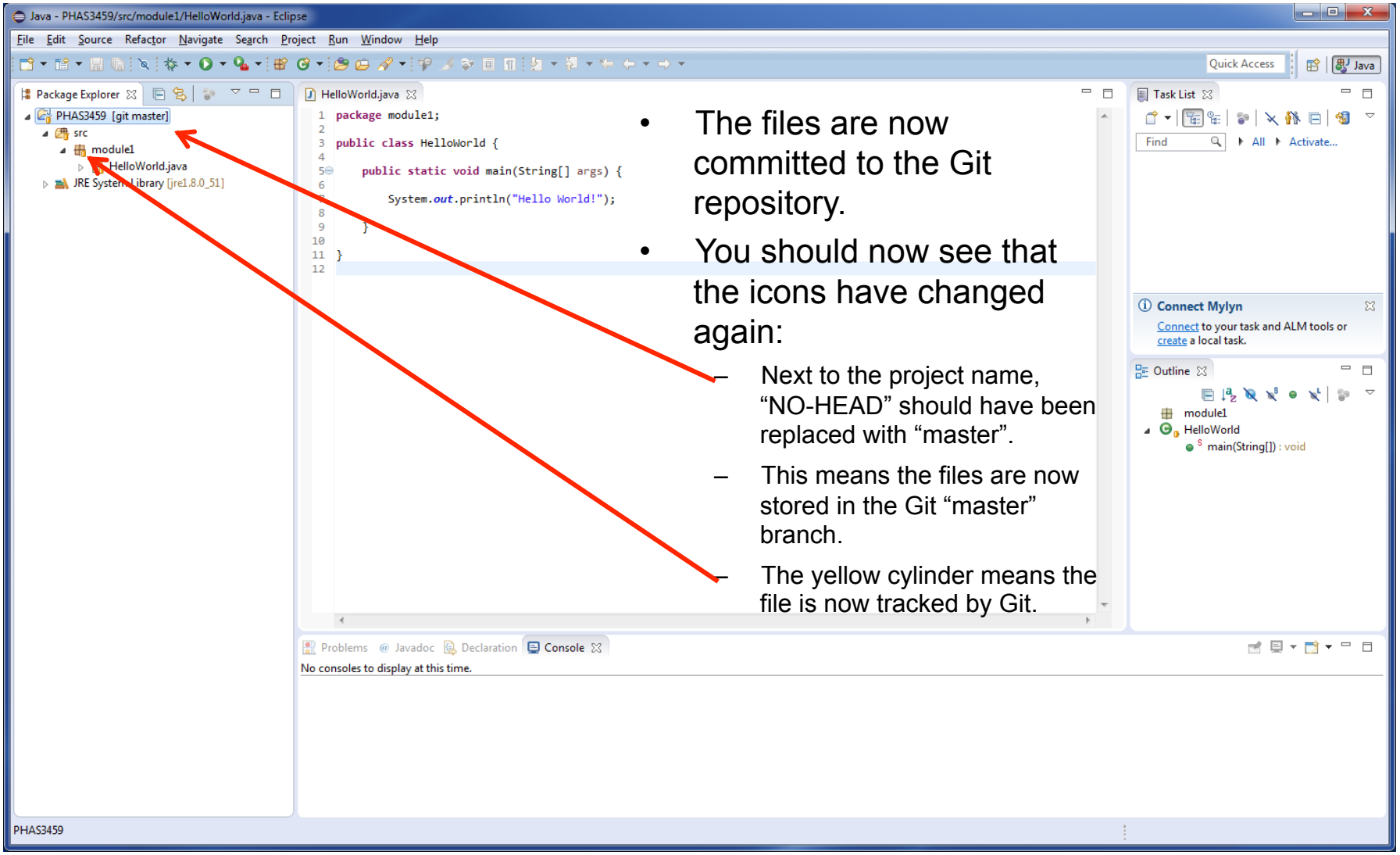
- Every time you make a commit, you include a commit message that describes briefly what you're committing.
- Enter this text into the "Commit message" box.
- Once you're happy with your file selection and your message, click "Commit".

Author: Simon Jolly <s.jolly@ucl.ac.uk>

Committer: Simon Jolly <s.jolly@ucl.ac.uk>

Commit and Push... Commit

First Commit Complete



- The files are now committed to the Git repository.
- You should now see that the icons have changed again:
  - Next to the project name, “NO-HEAD” should have been replaced with “master”.
  - This means the files are now stored in the Git “master” branch.
  - The yellow cylinder means the file is now tracked by Git.

Creating A Second Java Class

- If you haven't already, create a new class called "Simple":
  - Right-click on the "module1" package and select "New → Class".
  - Give the class the name "Simple" and select the "public static void main (String args[])" check box.
  - Click "Finish".
- Copy the text shown below into this new class and save it.

```
package module1;

import java.util.Date;

public class Simple {

  public static void main(String[] args) {

    System.out.println("Program starting");
    Date myDate = new Date();
    System.out.println(myDate);
    System.out.println("Program finished");

  }

}
```

“Simple” Ready To Be Committed

Java - PHAS3459/src/module1/Simple.java - Eclipse

Package Explorer
- PHAS3459 [git master]
  - src
    - module1
      - HelloWorld.java
      - Simple.java
  - JRE System Library [jre1.8.0_51]

HelloWorld.java | Simple.java

```
1  package module1;
2
3  import java.util.Date;
4
5  public class Simple {
6
7      public static void main(String[] args) {
8
9          System.out.println("Program starting");
10         Date myDate = new Date();
11         System.out.println(myDate);
12         System.out.println("Program finished");
13
14     }
15
16 }
17
```

Console

```
<terminated> Simple [Java Application] C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (24 Jul 2015, 17:07:38)
Program starting
Fri Jul 24 17:07:39 BST 2015
Program finished
```

Task List

Connect Mylyn

Connect to your task and ALM tools or create a local task.

Outline
- module1
  - Simple
    - main(String[]) : void

You should now see that the chevrons have reappeared:

- Not next to the “HelloWorld” class: this is already under version control and unchanged…
- …but next to the Project and Package, indicating something they contain has changed since the last commit.
- “Simple” should also have shown up with a question mark to indicate it hasn’t yet been committed to Git.

Commit “Simple” To Git (1)

workspace - PHAS3459/src/module1/Simple.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems Javadoc Declaration Console Git Staging

> git [master]

Unstaged Changes (1)

Simple.java - PHAS3459/src/module1

Staged Changes (0)

Commit Message

- Right-click the Project and select “Team → Commit”.
- The “Simple.java” class should now appear under “Unstaged Changes”:
  - Note that this is the only file that shows up since it is the only difference from the last commit.

Author: Simon Jolly <s.jolly@ucl.ac.uk>

Committer: Simon Jolly <s.jolly@ucl.ac.uk>

Commit and Push... Commit

Commit “Simple” To Git

workspace - PHAS3459/src/module1/Simple.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems Javadoc Declaration Console Git Staging

> git [master]

Unstaged Changes (0)

Staged Changes (1)

Simple.java - PHAS3459/src/module1

Commit Message

Added Simple class to display date and time.

Author: Simon Jolly <s.jolly@ucl.ac.uk>

Committer: Simon Jolly <s.jolly@ucl.ac.uk>

Commit and Push... Commit

- “Simple.java” is now ready to be committed.
- Enter in a suitable commit message.
- Click “Commit”.

Second Commit Completed

Java - PHAS3459/src/module1/Simple.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- PHAS3459 [git master]
  - src
    - module1
      - HelloWorld.java
      - Simple.java
  - JRE System Library [jre1.8.0_51]

HelloWorld.java | Simple.java

```
1  package module1;
2
3  import java.util.Date;
4
5  public class Simple {
6
7      public static void main(String[] args) {
8
9          System.out.println("Program starting");
10         Date myDate = new Date();
11         System.out.println(myDate);
12         System.out.println("Program finished");
13
14     }
15
16 }
17
```

Task List

Find All Activate...

Connect Mylyn

Connect to your task and ALM tools or create a local task.

Outline

- module1
  - Simple
    - main(String[]) : void

Problems | Javadoc | Declaration | Console

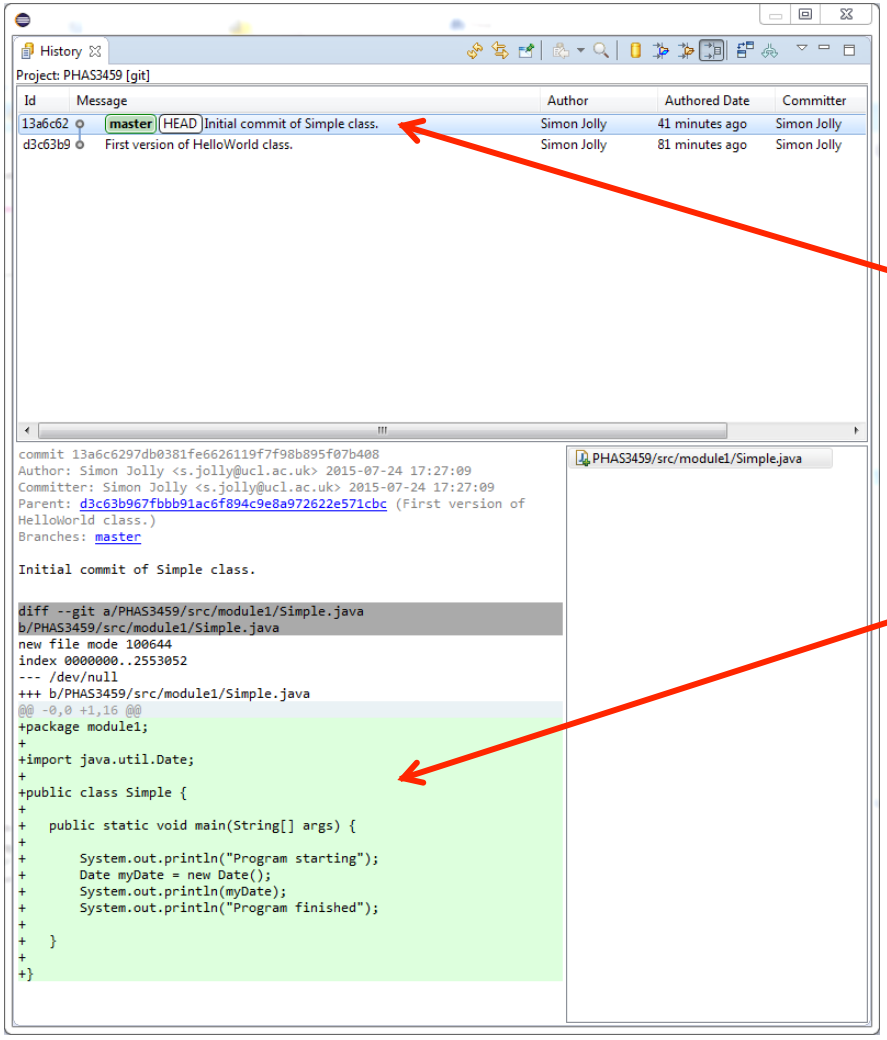
<terminated> Simple [Java Application] C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (24 Jul 2015, 17:07:38)

```
Program starting
Fri Jul 24 17:07:39 BST 2015
Program finished
```

module1.Simple.java - PHAS3459/src

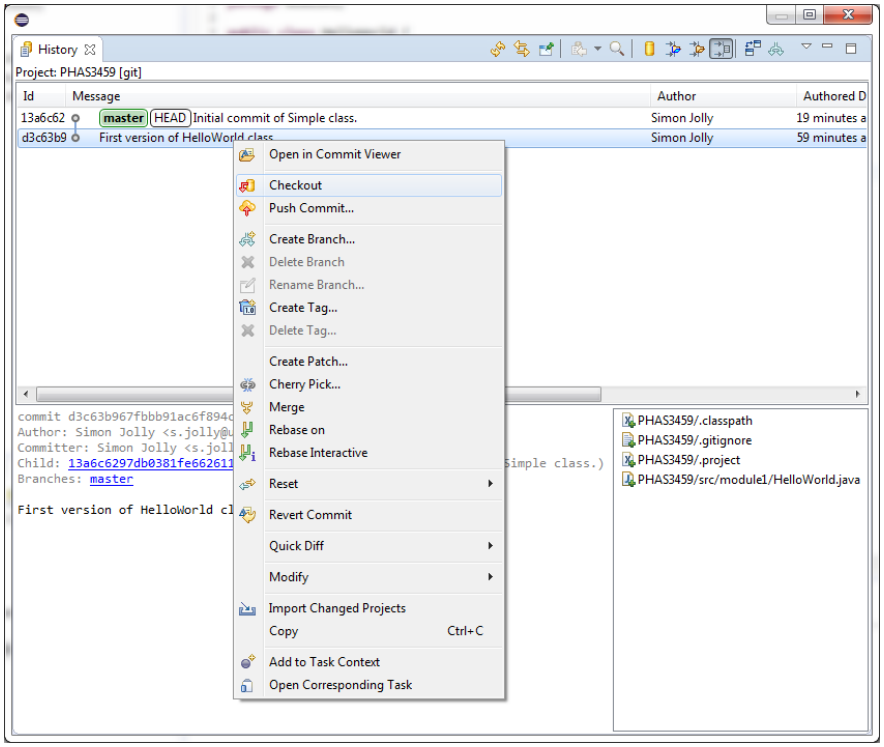
- Both classes should now show up with yellow cylinders.
- The chevrons should have disappeared.
- This indicates all is well…

Viewing Your History



- To view your Git history, right-click your Project and select "Team → Show in History".
- The Git history will show the timeline of your two commits.
- You can view more information about each commit – such as the commit message and the files committed – by selecting the appropriate commit in the History viewer.
- Clicking on an individual file in the bottom right will show all the lines that have been added (green) or removed (red) since the last commit in the bottom left-hand window (in Git's language, this is called a **diff**).

Checking Out A Previous Commit



- Say you've made a mistake in one of your files and want to go back to an earlier version: how do you do this?
  - If you're lucky, repeatedly clicking "Undo" will revert everything you've done…
  - A better way is to check out one of your previous commits.
- Open the History viewer, right-click the first commit you made and select "Checkout".
- You should get a warning message about a "detached HEAD state" telling you not to make changes: click "OK".
- Your "Simple" class should have disappeared!
- The data isn't lost: it's all stored in the Git repository.
- To get back to your most recent commit, right-click the Project and select "Team → Switch To → master".

Summary

- Version Control provides a method of tracking changes to your files over time and being able to revert to older versions of your files non-destructively.
- Some important things to note:
  - Always make your commit messages legible and explanatory (much like commenting your code…).
    - Too little information ("first commit", "another commit" etc.) makes the message pointless and means you have no idea what state your files will be in if you try to revert back to them.
    - Too much information makes it difficult to wade through to find out what's been changed.
  - You don't have to have your files in pristine condition when you commit! They don't even have to be working (it is helpful though): Git doesn't care.
- **We will be using Version Control as part of the assessment on each module.** When we mark your work, we will also check your Git History to make sure you have a commit corresponding to your submitted work.
- To find out more about Git and EGit, the best places to start are:
  - Pro Git Book: https://git-scm.com/book/
  - EGit User Guide: http://wiki.eclipse.org/EGit/User_Guide