Week 6 Lab

Design and Implementation

Objectives

In this week's lab, you will:

- learn how to deal with changes made on your remote Git repository, and to merge them with your local repository
- implement a solution that you have designed
- use concepts from Design By Contract to ensure that code meets specifications

UML Class Diagram

You must draw a UML class diagram showing the system that you plan to implement *by the end of the lab*. Note that this system will incorporate work you have done in multiple tasks, so **read the entire lab sheet** carefully.

This diagram should be quick and simple. It only needs to show the classes you plan to create and the relationships (*associations in UML*) between them. You don't need to show fields or methods. You are strongly encouraged to draw this diagram by hand — you don't need to use a UML diagramming tool.

You must get your design in UML approved by your demonstrator before you start coding. Both the design and the UML syntax must be complete and correct.

Task 1. Learn about dealing with changes on your remote Git repository

Do the exercise in the "Dealing with changes on the remote Git repository" section at the end of this document. Get your demonstrator to check that you have completed it sucessfully before attempting the following tasks.

Note

This week you have complete freedom to design your own solution for the tasks below. For each task, you must:

- Design and implement classes to implement the specified requirements.
- Use the concepts of preconditions and/or postconditions from *Design by Contract* (see Week 6 lectures) to ensure that your code meets the specifications.
- Add code to `Lair` to verify that your code works as expected before proceeding to the next task.
- Commit all changed and/or added files, and push to the Git repository on the Monash GitLab server that you started using in the Week 5 lab.

Task 2. Items

Our Lair Management System will not be complete until it can keep track of items as well as minions.

Items have the following properties:

- Items in the system may be stored in a warehouse, or may be assigned to a *properly-staffed* `LairLocation`. If they are assigned to a `LairLocation`, they must not require any skills that the `LairLocation` doesn't supply.
- Items have descriptions, and the description of a `LairLocation` should include the description of all items that have been placed there.
- Items are either equipment or traps:
  - Traps require more than one skill in order to be used: for example, a shark tank requires `SCUBA` and `OPTICS`.
  - Traps can be enabled and disabled. If they are disabled, their description should have the string `(disabled)` appended to it when it is displayed.
  - Traps cannot be moved after they have been placed in a `LairLocation`, but equipment can.
  - Equipment requires at most one skill in order to be used: for example, a mind-control ray might require `PSYCHOLOGY`.

Extend your system so that, after all minions have been created, users can do the following:

- Add new items to the warehouse
- Move items from the warehouse to compatible `LairLocations` (printing a sensible error message if this is not possible)
- Move items from one `LairLocation` to another (again, printing a sensible error message if this cannot be done)
- Enable and disable traps that have been placed in `LairLocations`

You may add new LairLocations to your system if you find that this helps you with testing. You are *not* required to make the warehouse be a `LairLocation`; some supervillains prefer to store their unused items off-site.

Task 3. Training and detraining

Extend your system so that `Minions` can have skills added or removed[1] via the user interface.

When a `Minion` has a skill added, the rest of the system should be unaffected.

If a `Minion` has a skill *removed*, and it is in a team that has been assigned to a `LairLocation`, then a check must be run to see if the team still meets the location's skill requirements. If those requirements are no longer met, then the team must be removed from the location along with any equipment it was using, and any traps that were placed there must be disabled.

Getting marked

Once you have completed these tasks, call your lab demonstrator to be marked. Your demonstrator will ask you to do a "walkthrough" of your program, in which you explain what each part does. Note that to receive full marks, **it is not sufficient merely to produce correct output**. Your solution must implement all structures specified above *exactly* — as software engineers we must follow specifications

[1]Normally, once a student has been trained in a skill, they stay trained. Unfortunately, minion training tends to lead to head trauma or permanent disability, so our analysts have identified a need for skills to be removable.

precisely. If you have not done so, your demonstrator will ask you to fix your program so that it does (if you want to get full marks).

If you do not get this finished by the end of this week's lab, you can complete it during the following week and get it marked at the start of your next lab — but make sure that your demonstrator has at least given you feedback on your design before you leave, so that you don't waste time implementing a poor or incorrect design. That will be the last chance to get marks for the exercise. You cannot get marked for an exercise more than one week after the lab for which it was set.

Marking

This lab is worth 2% of your final mark for FIT2099. Marks will be granted as follows:

- 0 marks if you do not attempt the tasks, or if you could only complete them once given the solution
- 1 mark if some tasks are incomplete, if the design or the implementation is flawed, or if you needed to see a significant part of the solution to complete them
- 2 marks if you were able to complete all tasks independently, with good design and correct implementation

Dealing with changes on the remote Git repository

When using a version control system such as Git, it is frequently the case that there is more than one person working on the project. This will be the case for your assignments in this unit. If someone else pushes changes to the remote repository (e.g. your Monash GitLab repository), it will be out of sync with your local repository. In this exercise we will see how to deal with this.

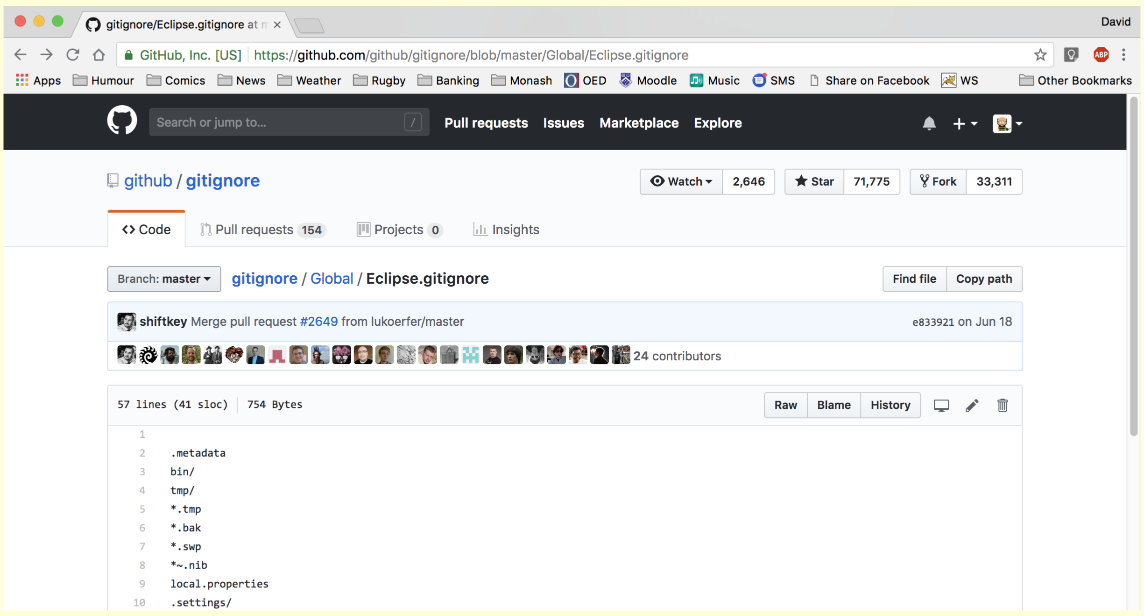
Make sure you have committed all your your Week 5 code, and pushed it to your Monash GitLab repository, before starting the exercise.

Someone adds a file not present in your local repository

If a member of your team is working on a different part of the project from you, it is likely that they will create new files, and push them to the repository. We will simulate a similar situation by adding a file to the repository via the Monash GitLab web interface, rather than from Eclipse.

First we need a file to add. It might as well be a useful one. There is a project on GitHub that collects useful `.gitignore` files (see `https://git-scm.com/docs/gitignore`) for various languages and IDEs. Go to `https://github.com/github/gitignore/blob/master/Global/Eclipse.gitignore` to get the file for Eclipse.

(Have a look around the project if you are using a different IDE. If you just want something generic for Java, go to `https://github.com/github/gitignore/blob/master/Java.gitignore`.)


gitignore/Eclipse.gitignore at m ×
David
GitHub, Inc. [US] https://github.com/github/gitignore/blob/master/Global/Eclipse.gitignore
Apps Humour Comics News Weather Rugby Banking Monash OED Moodle Music SMS Share on Facebook WS Other Bookmarks
Search or jump to...
Pull requests Issues Marketplace Explore
github / gitignore
Watch 2,646 Star 71,775 Fork 33,311
Code Pull requests 154 Projects 0 Insights
Branch: master
Find file Copy path
Raw Blame History

gitignore / Global / Eclipse.gitignore

shiftkey Merge pull request #2649 from lukoerfer/master e833921 on Jun 18

24 contributors

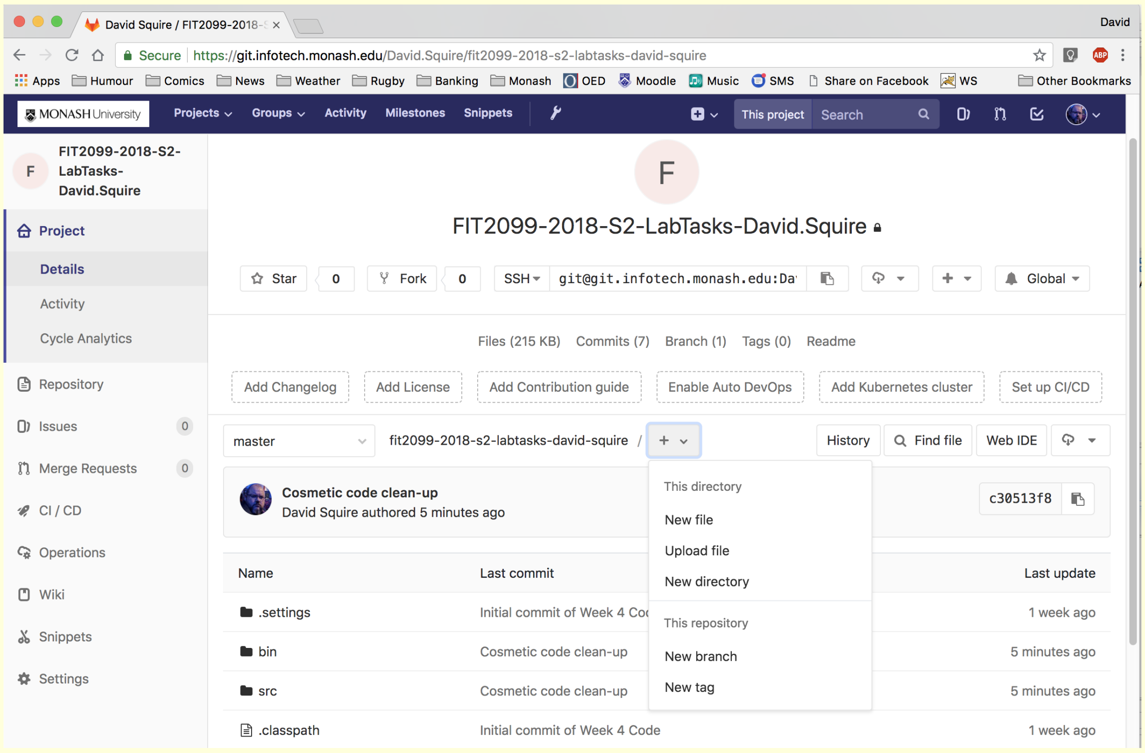
57 lines (41 sloc) 754 Bytes

```

.metadata
bin/
tmp/
*.tmp
*.bak
*.swp
*~.nib
local.properties
.settings/
```

Click on the Raw button, then right-click on in the browser window and select Save as.... Save the file to your Desktop with the filename `gitignore`. (We will later need to rename this `.gitignore`, but this is less hassle than getting your computer to show you files starting with `.` on the Desktop.)

Now go to your Monash GitLab repository in your web browser. Click on the + button, and select Upload file from the menu.


David Squire / FIT2099-2018-
David
Secure https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks-david-squire
Apps Humour Comics News Weather Rugby Banking Monash OED Moodle Music SMS Share on Facebook WS Other Bookmarks
MONASH University Projects Groups Activity Milestones Snippets This project Search
FIT2099-2018-S2-LabTasks-David.Squire
Project
Details
Activity
Cycle Analytics
Repository
Issues 0
Merge Requests 0
CI / CD
Operations
Wiki
Snippets
Settings
FIT2099-2018-S2-LabTasks-David.Squire
Star 0 Fork 0 SSH git@git.infotech.monash.edu:Da Global
Files (215 KB) Commits (7) Branch (1) Tags (0) Readme
Add Changelog Add License Add Contribution guide Enable Auto DevOps Add Kubernetes cluster Set up CI/CD
master fit2099-2018-s2-labtasks-david-squire / +
History Find file Web IDE
Cosmetic code clean-up
David Squire authored 5 minutes ago
c30513f8
This directory
New file
Upload file
New directory
This repository
New branch
New tag

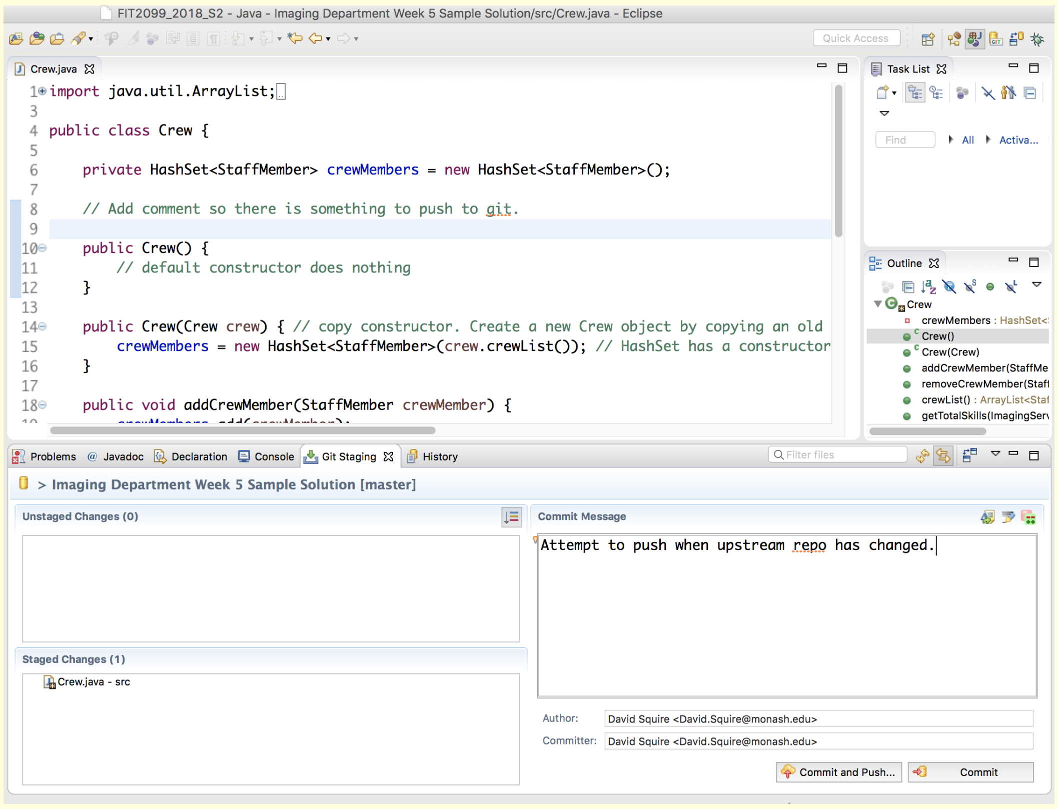
Name	Last commit	Last update
.settings	Initial commit of Week 4 Co	1 week ago
bin	Cosmetic code clean-up	5 minutes ago
src	Cosmetic code clean-up	5 minutes ago
.classpath	Initial commit of Week 4 Code	1 week ago

Upload the file `gitignore` from your Desktop.

By doing this, you have made a change on the Monash GitLab repository that does not exist in the local repository you created via Eclipse.

This is the same situation that would occur if another team member had pushed new files to your shared remote Git repository.

Now go into Eclipse. Make a change to one of your files (e.g. add a comment) and then attempt to do a Commit and Push... as you did last week.


FIT2099_2018_S2 - Java - Imaging Department Week 5 Sample Solution/src/Crew.java - Eclipse

```java
import java.util.ArrayList;

public class Crew {

    private HashSet<StaffMember> crewMembers = new HashSet<StaffMember>();

    // Add comment so there is something to push to git.

    public Crew() {
        // default constructor does nothing
    }

    public Crew(Crew crew) { // copy constructor. Create a new Crew object by copying an old
        crewMembers = new HashSet<StaffMember>(crew.crewList()); // HashSet has a constructor
    }

    public void addCrewMember(StaffMember crewMember) {
```

> Imaging Department Week 5 Sample Solution [master]

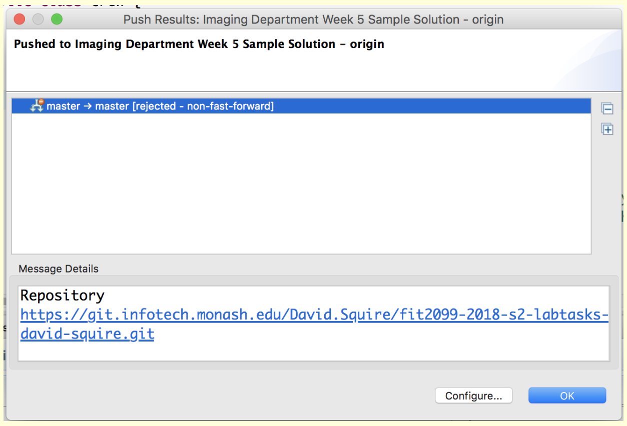
Unstaged Changes (0)

Staged Changes (1)
Crew.java - src

Commit Message
Attempt to push when upstream repo has changed.

Author: David Squire <David.Squire@monash.edu>
Committer: David Squire <David.Squire@monash.edu>

You will get a dialog box informing you that your push has been rejected:


Push Results: Imaging Department Week 5 Sample Solution - origin

Pushed to Imaging Department Week 5 Sample Solution – origin

master → master [rejected - non-fast-forward]

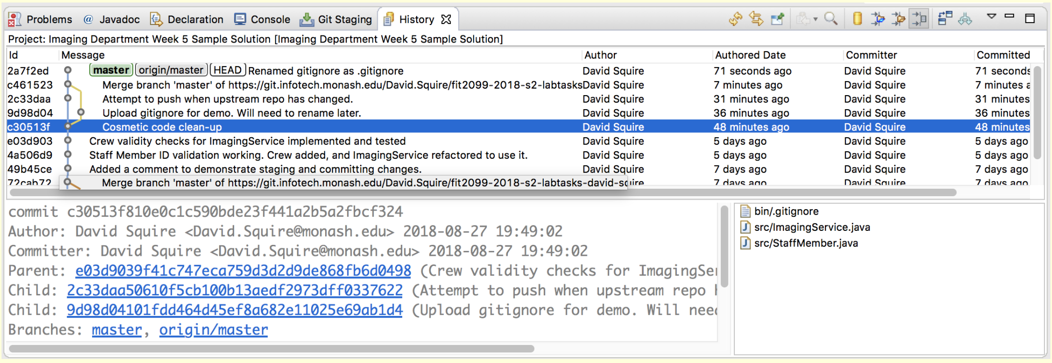
Message Details

Repository
https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks-david-squire.git

The error message is `rejected - non-fast-forward`. This is admittedly not the most easy message to understand. Basically, it means that there are changes on the remote repository that would be lost if you simply pushed the state of your repository.

There are multiple ways of dealing with this. Perhaps the simplest is to fetch the changes on the remote repository and merge them with yours first, and then do your push. You can do this in one command using Pull. Do this, as you did in the Git exercise last week.

You will see that the file `gitignore` has appeared in your Eclipse project. Right-click on it and rename it `.gitignore`. Do a Commit and Push.... This time it should work. The Git history on Eclipse should look something like this, showing the merge that occurred:



Project: Imaging Department Week 5 Sample Solution [Imaging Department Week 5 Sample Solution]

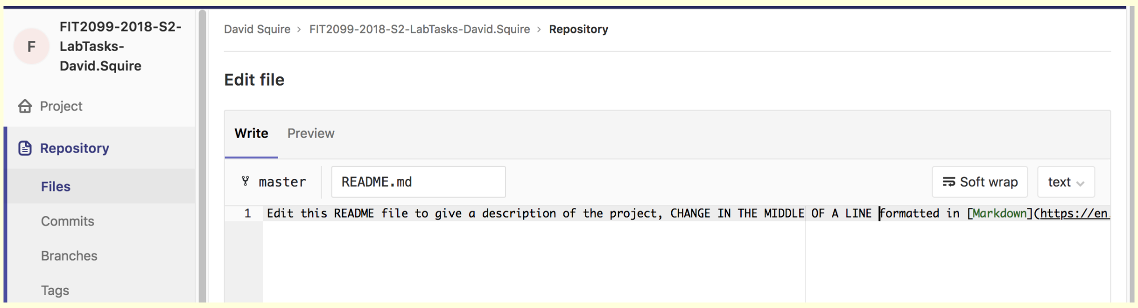
Id	Message	Author	Authored Date	Committer	Committed
2a7f2ed	master origin/master HEAD Renamed gitignore as .gitignore	David Squire	71 seconds ago	David Squire	71 seconds
c461523	Merge branch 'master' of https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks	David Squire	7 minutes ago	David Squire	7 minutes a
2c33daa	Attempt to push when upstream repo has changed.	David Squire	31 minutes ago	David Squire	31 minutes
9d98d04	Upload gitignore for demo. Will need to rename later.	David Squire	36 minutes ago	David Squire	36 minutes
c30513f	Cosmetic code clean-up	David Squire	48 minutes ago	David Squire	48 minutes
e03d903	Crew validity checks for ImagingService implemented and tested	David Squire	5 days ago	David Squire	5 days ago
4a506d9	Staff Member ID validation working. Crew added, and ImagingService refactored to use it.	David Squire	5 days ago	David Squire	5 days ago
49b45ce	Added a comment to demonstrate staging and committing changes.	David Squire	7 days ago	David Squire	7 days ago
72cab72	Merge branch 'master' of https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks-david-squire	David Squire	7 days ago	David Squire	7 days ago

```
commit c30513f810e0c1c590bde23f441a2b5a2fbcf324
Author: David Squire <David.Squire@monash.edu> 2018-08-27 19:49:02
Committer: David Squire <David.Squire@monash.edu> 2018-08-27 19:49:02
Parent: e03d9039f41c747eca759d3d2d9de868fb6d0498 (Crew validity checks for ImagingSe
Child: 2c33daa50610f5cb100b13aedf2973dff0337622 (Attempt to push when upstream repo h
Child: 9d98d04101fdd464d45ef8a682e11025e69ab1d4 (Upload gitignore for demo. Will nee
Branches: master, origin/master
```

bin/.gitignore
src/ImagingService.java
src/StaffMember.java

Someone changes a file that was already in your repository

Now go back to your Monash Gitlab repository in your browser. Click on `README.md`, and then click on the Edit button. Make a change in the middle of an existing line, like this:


F
FIT2099-2018-S2-LabTasks-David.Squire
Project
Repository
Files
Commits
Branches
Tags
David Squire › FIT2099-2018-S2-LabTasks-David.Squire › Repository
Edit file
Write
Preview
master
README.md
Soft wrap
text

```
Edit this README file to give a description of the project, CHANGE IN THE MIDDLE OF A LINE formatted in [Markdown](https://en
```

Click on the green Commit Changes button.

Now go back to Eclipse, and make a change to your code (e.g. remove the comment you added earlier).

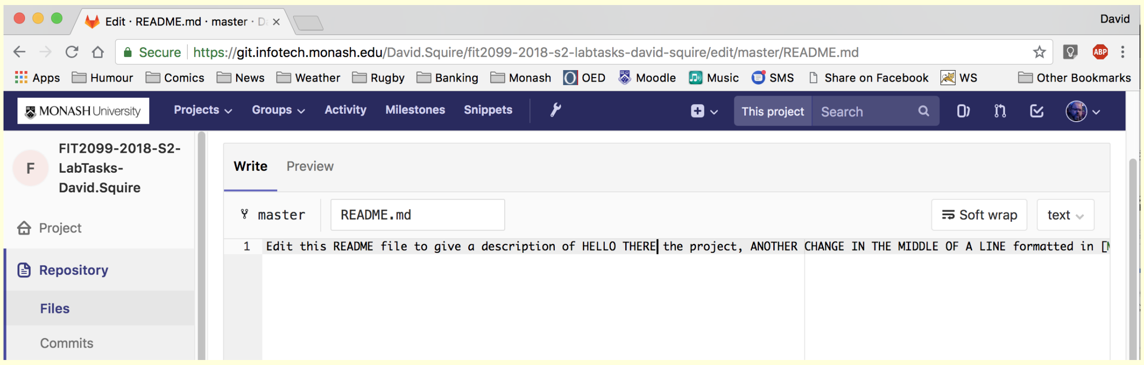
Try doing a Commit and Push.... It will be rejected again, as it was earlier.

Try doing a Pull. It should work.

Open the file `README.md` in Eclipse. You will see that the pull has caused the changes made on the remote repository to be merged into the local copy.

Remove the change you made to the file in the Eclipse editor. Save it, and do a commit (but not a push).

Go back to the Monash Gitlab repository in your browser, and edit `README.md` again, making more changes on that same line. Click on the green Commit Changes button.

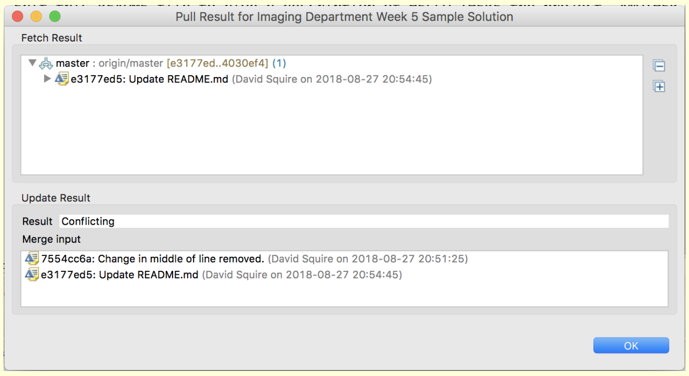

Edit · README.md · master · D
Secure https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks-david-squire/edit/master/README.md
FIT2099-2018-S2-LabTasks-David.Squire
Project
Repository
Files
Commits
Write
Preview
master
README.md
Soft wrap
text

```
Edit this README file to give a description of HELLO THERE the project, ANOTHER CHANGE IN THE MIDDLE OF A LINE formatted in [M
```

We now have different changes in the same place in the same file committed to each repository. Go back to Eclipse.

Try doing a Commit and Push.... It will be rejected again.

Try doing a Pull. It will *also* be rejected,with Result: Conflicting:


Pull Result for Imaging Department Week 5 Sample Solution

Fetch Result

master : origin/master [e3177ed..4030ef4] (1)
e3177ed5: Update README.md (David Squire on 2018-08-27 20:54:45)

Update Result

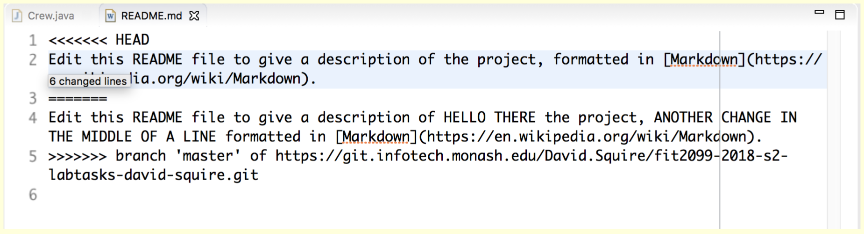
Result Conflicting

Merge input

7554cc6a: Change in middle of line removed. (David Squire on 2018-08-27 20:51:25)
e3177ed5: Update README.md (David Squire on 2018-08-27 20:54:45)

We have now created conflicting changes that Git can't merge automatically.

Close the dialog and look at README.md in Eclipse:

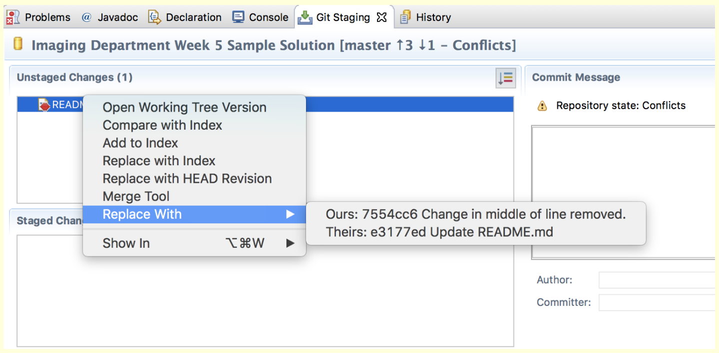


```
<<<<<<< HEAD
Edit this README file to give a description of the project, formatted in [Markdown](https://
...lia.org/wiki/Markdown).
=======
Edit this README file to give a description of HELLO THERE the project, ANOTHER CHANGE IN
THE MIDDLE OF A LINE formatted in [Markdown](https://en.wikipedia.org/wiki/Markdown).
>>>>>>> branch 'master' of https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-
labtasks-david-squire.git
```

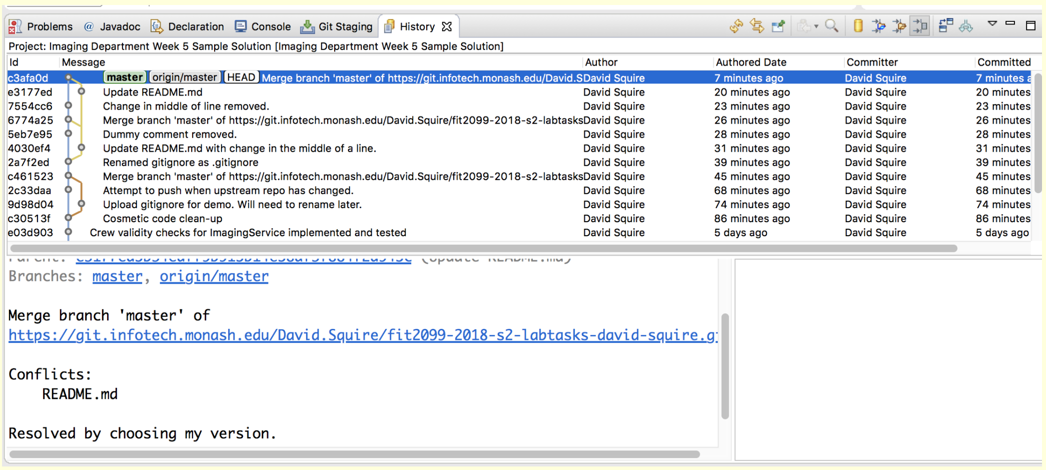
Git has inserted conflict-resolution markers, and the text from the conflicting versions. (see `https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging` for more information).

You can now do various things to resolve the conflct. For example, you can edit the file to resolve the conflict manually, use various tools to edit the file, or choose to use one of the versions from before the pull attempt as the "correct" version.

Go to the Git Staging tab in Eclipse, and right-click on README.md (notice the red conflict decoration in its icon) to see some of your options.


Problems
Javadoc
Declaration
Console
Git Staging
History
Imaging Department Week 5 Sample Solution [master ↑3 ↓1 – Conflicts]
Unstaged Changes (1)
README
Staged Changes
Open Working Tree Version
Compare with Index
Add to Index
Replace with Index
Replace with HEAD Revision
Merge Tool
Replace With
Show In ⌥⌘W
Ours: 7554cc6 Change in middle of line removed.
Theirs: e3177ed Update README.md
Commit Message
Repository state: Conflicts
Author:
Committer:

I chose to resolve the conflict by choosing Ours — thus discarding the changes that had been made earlier on the remote repository. After a Commit and Push..., my history now looks like:



Project: Imaging Department Week 5 Sample Solution [Imaging Department Week 5 Sample Solution]

Id	Message	Author	Authored Date	Committer	Committed
c3afa0d	master origin/master HEAD Merge branch 'master' of https://git.infotech.monash.edu/David.S	David Squire	7 minutes ago	David Squire	7 minutes a
e3177ed	Update README.md	David Squire	20 minutes ago	David Squire	20 minutes
7554cc6	Change in middle of line removed.	David Squire	23 minutes ago	David Squire	23 minutes
6774a25	Merge branch 'master' of https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks	David Squire	26 minutes ago	David Squire	26 minutes
5eb7e95	Dummy comment removed.	David Squire	28 minutes ago	David Squire	28 minutes
4030ef4	Update README.md with change in the middle of a line.	David Squire	31 minutes ago	David Squire	31 minutes
2a7f2ed	Renamed gitignore as .gitignore	David Squire	39 minutes ago	David Squire	39 minutes
c461523	Merge branch 'master' of https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks	David Squire	45 minutes ago	David Squire	45 minutes
2c33daa	Attempt to push when upstream repo has changed.	David Squire	68 minutes ago	David Squire	68 minutes
9d98d04	Upload gitignore for demo. Will need to rename later.	David Squire	74 minutes ago	David Squire	74 minutes
c30513f	Cosmetic code clean-up	David Squire	86 minutes ago	David Squire	86 minutes
e03d903	Crew validity checks for ImagingService implemented and tested	David Squire	5 days ago	David Squire	5 days ago

```
Branches: master, origin/master

Merge branch 'master' of
https://git.infotech.monash.edu/David.Squire/fit2099-2018-s2-labtasks-david-squire.g

Conflicts:
    README.md

Resolved by choosing my version.
```

Feel free to explore other conflict resolution options.

The Bottom Line

Git is very good at merging changes automatically, but it is not magic. It is possible to create conflicts it can't resolve.

Using a version control system doesn't mean that you don't need to think about who is working on what. Communication with your team members is still vital — particularly if you are working on the same branch.

Git provides a mechanism to facilitate team members working in parallel: branches. We do not require you to use branches in this unit, but feel free to do so. You can read about popular Git workflows at:

- GitHub Flow `https://guides.github.com/introduction/flow/`
- GitLab Flow `https://docs.gitlab.com/ee/workflow/gitlab_flow.html`
- Gitflow `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`