COMP3111: Software Engineering

Extending the base code and teamwork at GitHub

Learning Outcomes

- To extend the base code
- To learn branching and pull requests at GitHub

Glossary lists:
The following concepts are important for this lab.

Concepts	
repo	Repository, refer to a place that store files. There are local repos and remote repos (e.g. Github, Bitbucket).
commit	Take a snapshot of your current file system. You can rewind your commit like what you have done in Lab 1.
push	To upload your commit to your remote repo.
pull	To download from your remote repo to your local repo.
branch	A series of commits separated from main (master) trunk. Very useful when developing a new feature.
merge	Be used to combine two branches.
checkout	Be used to switch between branches
clone	To download a new copy from remote repo to your local machine, and also set up a local repo.
origin	The name of the default remote repo
master	The name of the default branch
Operations on Github	
pull request (PR)	Request the repo owner to adopt a change made by the requester.
fork	To duplicate a remote repo on GitHub and gain full access right on it. **We are not using fork in our project.**

Supervised Lab Exercises

Exercise 1: Create your team repo

In this exercise you are asked to setup a team private repository that contains the project skeleton code. Then, clone the project to your local machine and try to compile it. You might work with either Eclipse or command line interfaces.

This repository is shared by the team. Therefore, only one member of your team is required to do this task. To facilitate the lab work, please grant your teammate the

ability to approve a Pull-Request. For any reason if you are unable to create a private repo, please do the lab with public repo. But you need to recreate a private repo after the lab.

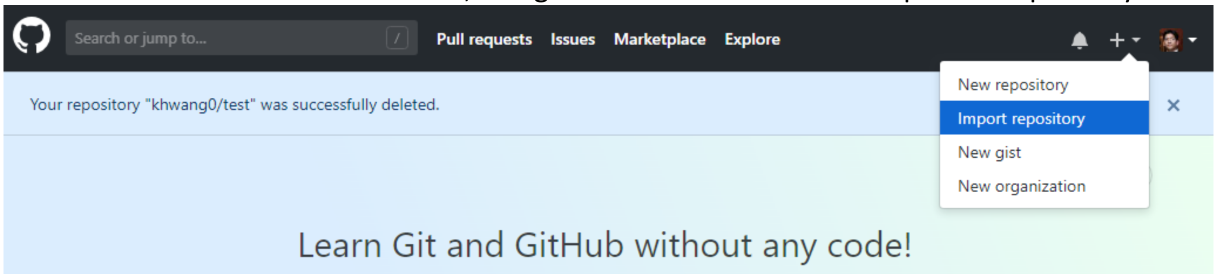
Step 1.1: Initial Software Team Checklist

Before we start, we assume the followings:

- A team of 3 student is formed
- Each student should have applied a free GitHub account
- Your team has a private repository
- Your team has invited the TA account **comp3111ta (not khwang0)** as a collaborator.
- DO NOT COMMIT OR CREATE README.MD. (I hope you know why)

Step 1.2: Create a private repo by import

With the GitHub Education account, navigate on Github and create a private repository.



Your repository "khwang0/test" was successfully deleted.

Learn Git and GitHub without any code!

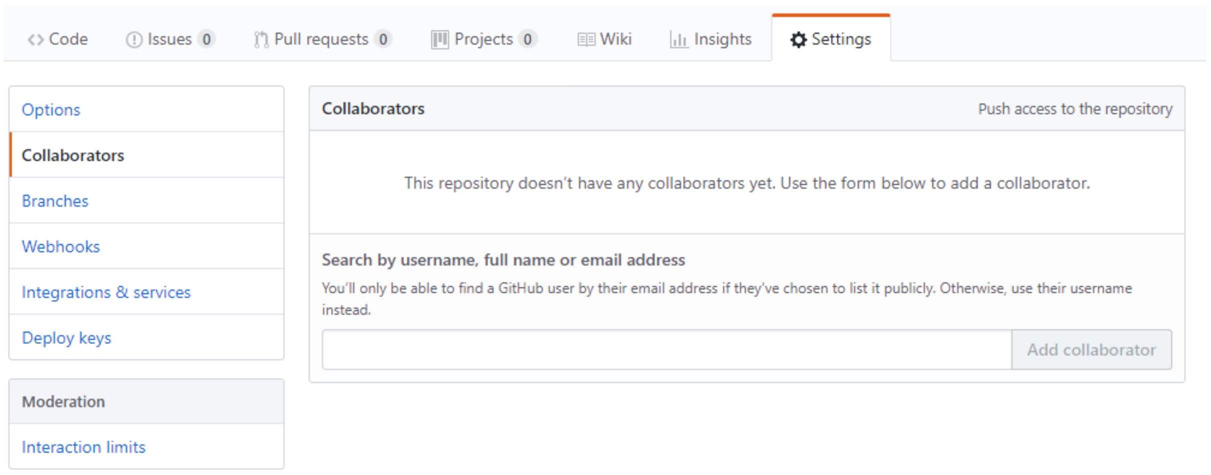
Type the project GitHub URL into the *Your old repository's clone URL*. Make sure you have selected private. After a while, you should have create a new private repo that contain all the project code.

Step 1.2 Alternative. For some reasons if it does not work, you can:

1. Make an empty folder in your computer. Use command line to navigate to that folder.
2. Use the command shown in lab3 (you can find it on Piazza) to create a git repo and add your github repo as a remote repo.
3. Download the project from khwang0/2019F-COMP3111 as zip.
4. Unzip everything into the local folder you have created above.
5. Git add everything (**git add** *) and commit it and push it.

Step 1.3: Add collaborators

On your private repo page select Setting > Collaborator and add your teammates and **comp3111ta**



Collaborators

Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

Step 1.4: Clone the project to your Eclipse

- Select File > Import...
- Select Git > Projects
- Select Clone URI and then click Next >
- Use the URL of the private repository
- Click Next > buttons a few times. Accept all default settings
- Click Finish at the end

After importing this GitHub repo, the Eclipse project explorer should be displayed as follows:

Step 1.4 Alternative: For some reasons it might not work in certain version of Eclipse.

1. Clone the project from your repo by "git clone <your_url>" (skip this if you have done Step 1.2A)
2. Select File > Import ...
3. Select General > Project from Folder or Archive
4. Click Directory > Select the folder that you have clone the project > Click ok to import
5. A project shall be added to your eclipse. Right click the project > Configure > Add Gradle Nature

Step 1.5: Use Gradle Run to compile and execute it. You shall see a sample program running.

At this point you might experience that there is a lot of errors displayed in your Eclipse IDE but the code runs. It is because that Eclipse does not recognize the library. Try:

- Right click the file build.gradle > Gradle > Refresh Gradle Project;
- If it does not work, try include the SDK into your build path manually.

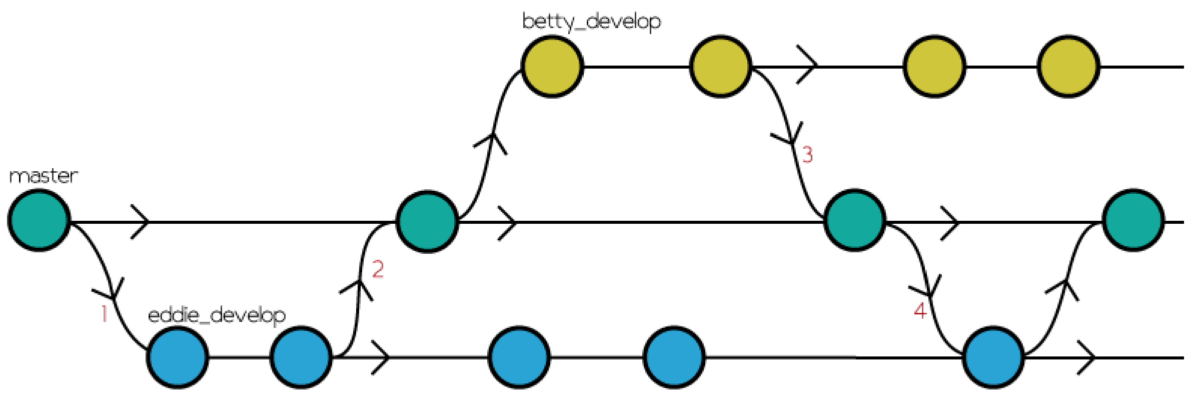
Note: You may use this Github to manage your project.

Exercise 2: Git Branching

In this exercise you are required to create a local branch and update Controller.java. Review your change and push it to GitHub as a separated branch.

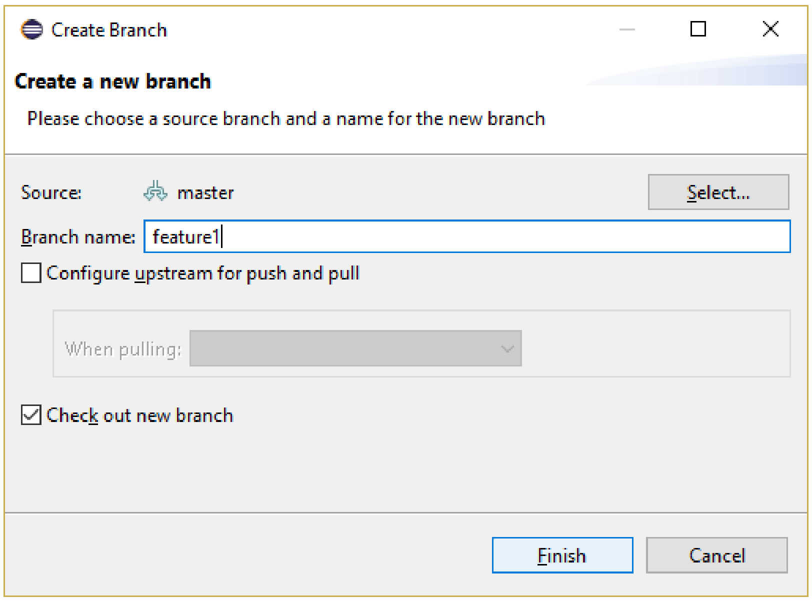
By default, there is a default branch (called **master** branch). Usually, the master branch should be a stable release of the project. In some projects (your project as well), pushing new code to the master branch is a bad idea.

A developer should first split from a master branch (or other appropriate development branches). After that, the developer should focus on developing the features. S/he should also test the implemented features thoroughly.



Step 2.1: Creating a new branch (e.g. feature1)

- Right-click the project folder
- Select Team > Switch To > New Branch…
- Make sure "Check out new branch" is checked
- Click "Finish"



Create Branch

Create a new branch

Please choose a source branch and a name for the new branch

Source: master

Branch name: feature1

☐ Configure upstream for push and pull

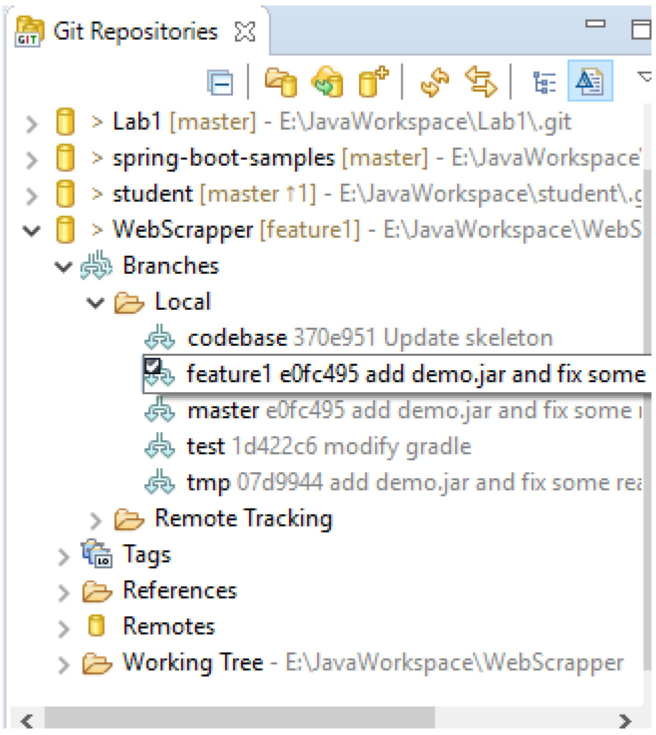
When pulling:

☑ Check out new branch

Note: do not repeat branch name with your teammates.

Step 2.2: Switching to different branches

In Eclipse, “Git Repositories” Window is the best tool to review and switch different branches. You can double-click any local branch to checkout and switch. If it is not visible, choose Window > Show View > Other to re-open this window

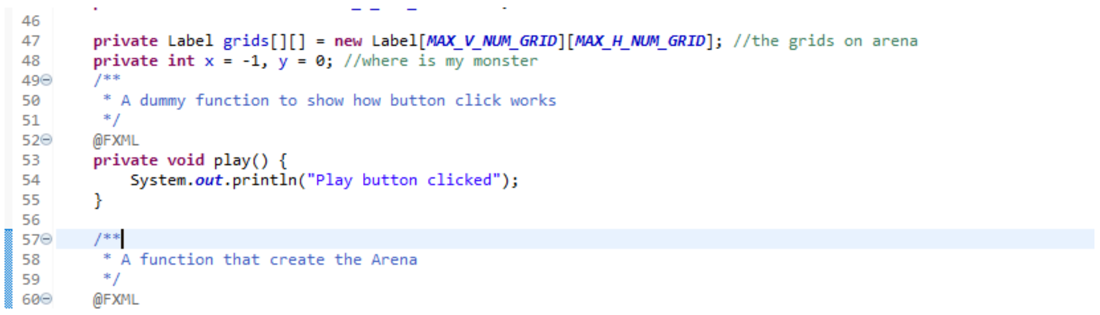


Git Repositories

```
> Lab1 [master] - E:\JavaWorkspace\Lab1\.git
> spring-boot-samples [master] - E:\JavaWorkspace
> student [master ↑1] - E:\JavaWorkspace\student\.g
> WebScrapper [feature1] - E:\JavaWorkspace\WebS
    Branches
      Local
        codebase 370e951 Update skeleton
        feature1 e0fc495 add demo.jar and fix some
        master e0fc495 add demo.jar and fix some i
        test 1d422c6 modify gradle
        tmp 07d9944 add demo.jar and fix some rea
      Remote Tracking
    Tags
    References
    Remotes
    Working Tree - E:\JavaWorkspace\WebScrapper
```

Step 2.3: Make some change

- Ensure that you are now working in a development branch (e.g. feature1)
- Double-click MyController.java and insert the following code into the function play()

- 

```
private Label grids[][] = new Label[MAX_V_NUM_GRID][MAX_H_NUM_GRID]; //the grids on arena
private int x = -1, y = 0; //where is my monster
/**
 * A dummy function to show how button click works
 */
@FXML
private void play() {
    System.out.println("Play button clicked");
}

/**
 * A function that create the Arena
 */
@FXML
```

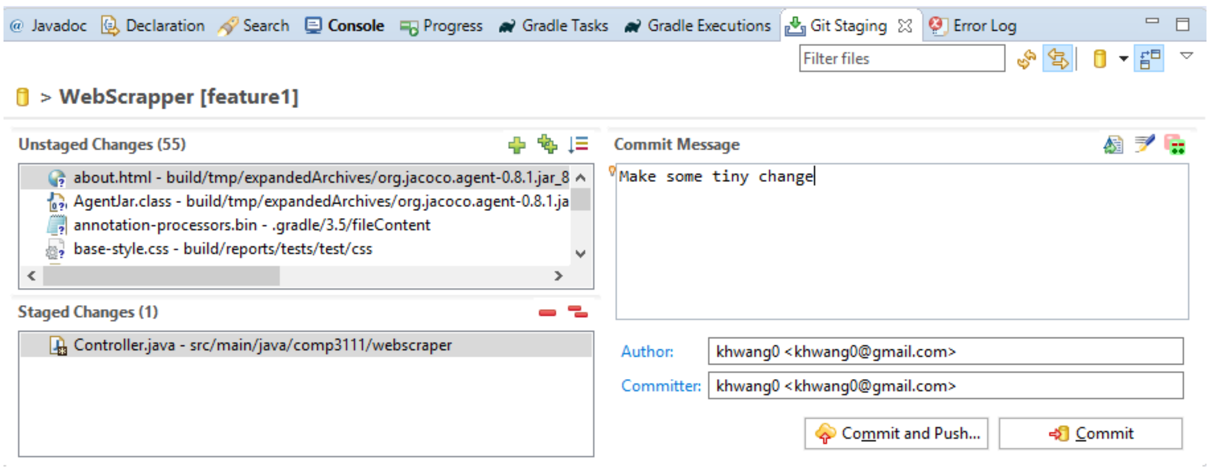
```
    Label newLabel = new Label();
        newLabel.setLayoutX(GRID_WIDTH / 4);
        newLabel.setLayoutY(GRID_WIDTH / 4);
        newLabel.setMinWidth(GRID_WIDTH / 2);
        newLabel.setMaxWidth(GRID_WIDTH / 2);
        newLabel.setMinHeight(GRID_WIDTH / 2);
        newLabel.setMaxHeight(GRID_WIDTH / 2);
        newLabel.setStyle("-fx-border-color: black;");
        newLabel.setText("*");
        newLabel.setBackground(new Background(new BackgroundFill(Color.YELLOW,
CornerRadii.EMPTY, Insets.EMPTY)));
        paneArena.getChildren().addAll(newLabel);
```

- Try running the program before and after the changes

Step 2.4: Commit changes to the development branch

In Eclipse, “Git Staging” Window is the best tool to commit changes to an active branch. If it is not visible, choose Window > Show View > Other to re-open this window

- Ensure that you are working in the development branch
- Add some files to it.
- Type in an appropriate commit message
- Click “**Commit**” .   **Not push**



about.html - build/tmp/expandedArchives/org.jacoco.agent-0.8.1.jar_8
AgentJar.class - build/tmp/expandedArchives/org.jacoco.agent-0.8.1.ja
annotation-processors.bin - .gradle/3.5/fileContent
base-style.css - build/reports/tests/test/css

Controller.java - src/main/java/comp3111/webscraper

```
Make some tiny change
```

Author: khwang0 <khwang0@gmail.com>
Committer: khwang0 <khwang0@gmail.com>

(The filename does not really match with yours)

Step 2.5: Keep working on the development branch

You should keep working on your development branch. The changes won't affect other branches. There are many advanced features (e.g. restore the branch from a previous commit, rebasing, renaming branch, merging, etc.). Eclipse IDE should support most of these advanced features in "Git Repositories" or "Git Staging" window. In this lab you might try changing another label.

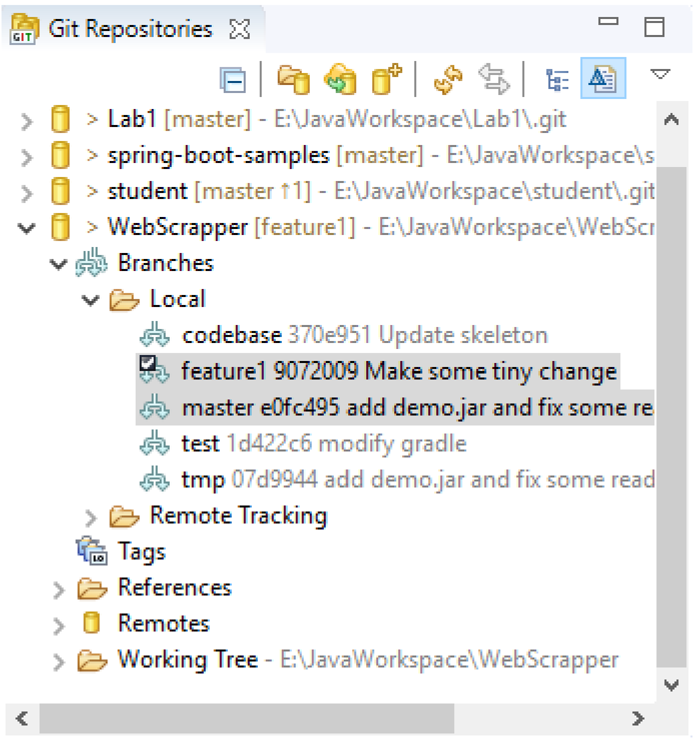
Step 2.6: Pull changes from GitHub

Before pushing the changes to GitHub, we should pull, check, and merge changes from GitHub. Ideally, there won't be any conflict. If conflicts happen, the developers need to resolve them.

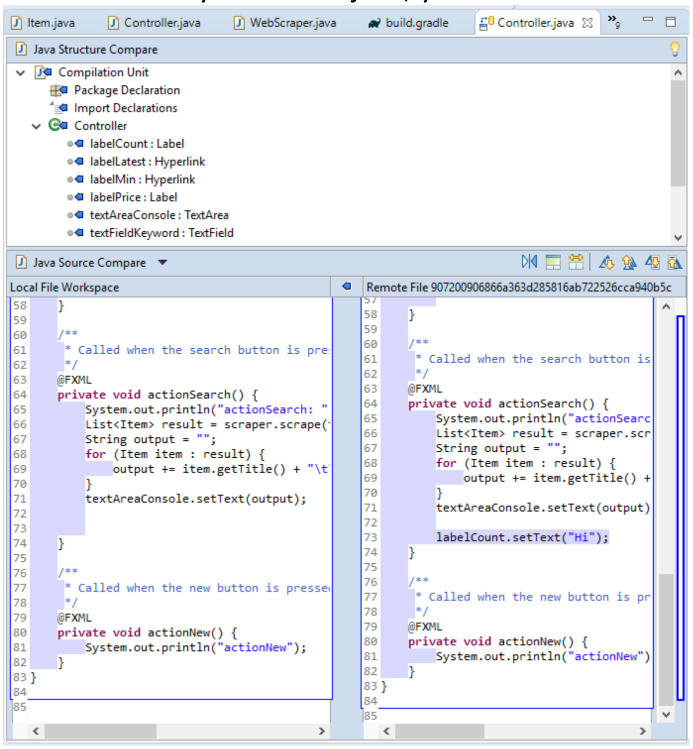
- Right-click the repository icon in "Git Repositories" window
- Select "Pull"
    - There may be an error message. The reason is that in the current step, we don't have the development branch (i.e. feature1) in GitHub.

Step 2.7: Review the difference between the master branch and development branch

- In "Git Repositories" window, press "SHIFT" key to select 2 branches
- 2 items selected should be shown in the status below


Git Repositories
Lab1 [master] - E:\JavaWorkspace\Lab1\.git
spring-boot-samples [master] - E:\JavaWorkspace\s
student [master ↑1] - E:\JavaWorkspace\student\.git
WebScrapper [feature1] - E:\JavaWorkspace\WebScr
Branches
Local
codebase 370e951 Update skeleton
feature1 9072009 Make some tiny change
master e0fc495 add demo.jar and fix some re
test 1d422c6 modify gradle
tmp 07d9944 add demo.jar and fix some read
Remote Tracking
Tags
References
Remotes
Working Tree - E:\JavaWorkspace\WebScrapper

- Right click, and select “Synchronize with each other” A “Synchronize” perspective will be open.
- Double-click MyController.java, you can review the difference between 2 branches



```
    }

    /**
     * Called when the search button is pre
     */
    @FXML
    private void actionSearch() {
        System.out.println("actionSearch: "
        List<Item> result = scraper.scrape(
        String output = "";
        for (Item item : result) {
            output += item.getTitle() + "\t
        }
        textAreaConsole.setText(output);


    }

    /**
     * Called when the new button is presse
     */
    @FXML
    private void actionNew() {
        System.out.println("actionNew");
    }
}
```

```
    }

    /**
     * Called when the search button is
     */
    @FXML
    private void actionSearch() {
        System.out.println("actionSearc
        List<Item> result = scraper.scr
        String output = "";
        for (Item item : result) {
            output += item.getTitle() +
        }
        textAreaConsole.setText(output)

        labelCount.setText("Hi");
    }

    /**
     * Called when the new button is pr
     */
    @FXML
    private void actionNew() {
        System.out.println("actionNew")
    }
}
```

(The file content does not really match with yours)

- After that, switch back to Java perspective
  - Window > Perspective > Close perspective

Step 2.8: Push the development branch to GitHub

- Select the development branch (i.e. dev-peter) in “Git Repositories”
- Right-click, choose “Push Branch…”
- Accept all default settings
- Go to the GitHub repository via web browser, there should be 2 branches at GitHub:

Try the following with command lines if things does not work out.

```
#assume your branch name is feature1
git checkout –b feature1
#edit your Controller.java using some editors
#assume you are in Windows, run the program
.\gradlew run
#or in macOS
./gradlew.sh run
#commit
git commit –am “some tiny change”
#pull
git pull origin master
#compare
git diff feature1 master
#push
git push origin feature1
```

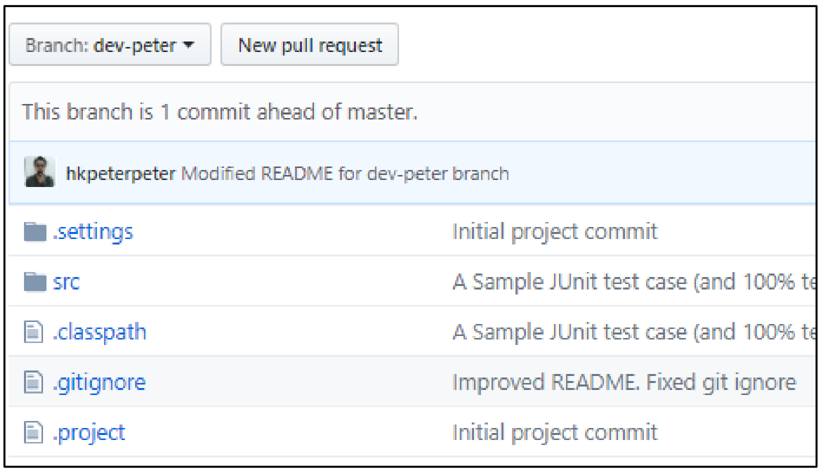
Exercise 3: Pull Request at GitHub

Pull request is a systematic way to merge changes from one branch (e.g. dev-peter) to another branch (e.g. master). Before merging the changes, a GitHub pull request provides a web interface to let developers discuss and review the changes

With some advanced settings in GitHub, several operations can be applied (e.g. continuous integration) as well to automatically rebuild and run unit tests. Students who are interested can explore relevant tutorials to add these features.

Step 3.1: Switch to the development branch at GitHub

- From the "Branch" combo box, select the development branch (i.e. dev-peter)
- You should see that README is changed in this branch



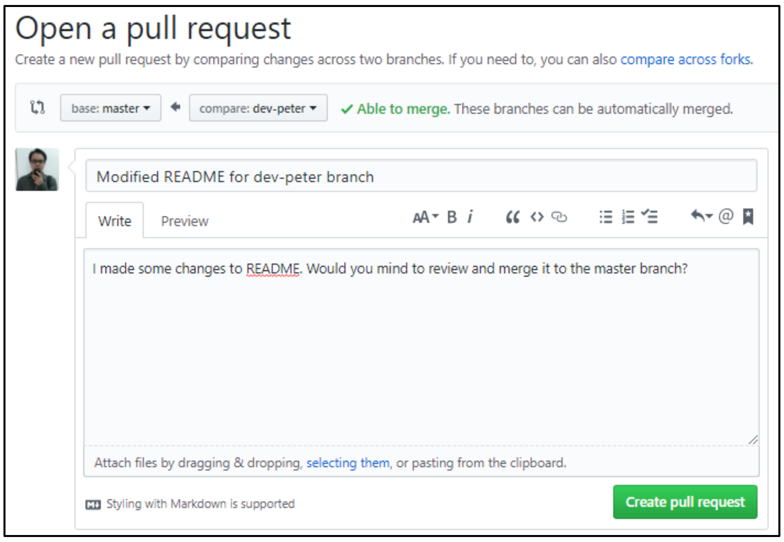
This branch is 1 commit ahead of master.

hkpeterpeter Modified README for dev-peter branch

.settings	Initial project commit
src	A Sample JUnit test case (and 100% te
.classpath	A Sample JUnit test case (and 100% te
.gitignore	Improved README. Fixed git ignore
.project	Initial project commit

Step 3.2: Creating a new pull request

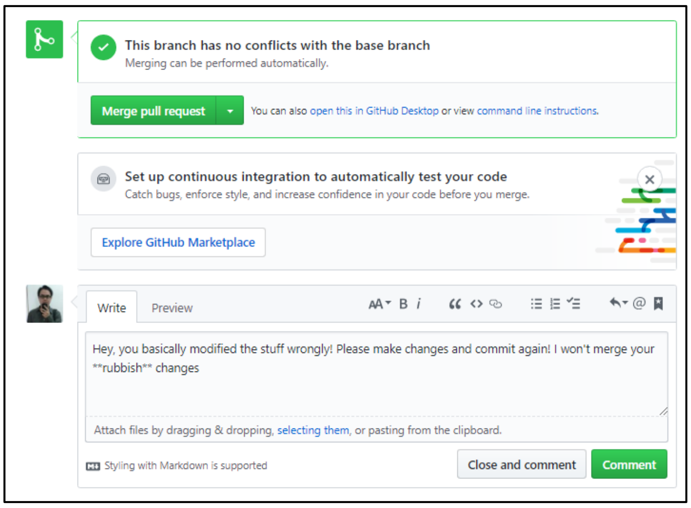
- Click "New pull request". Type in a message and click "Create pull request"
- You can assign reviewers (if your project has more than one contributors) on the right-hand side

- 
Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.
base: master ← compare: dev-peter ✓ Able to merge. These branches can be automatically merged.
Modified README for dev-peter branch

I made some changes to README. Would you mind to review and merge it to the master branch?
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Styling with Markdown is supported

Step 3.3: Keep an online discussion before merging

- The project leader **SHOULD** review the codes before pressing the "Merge pull request" button
- If the code quality is not good, the project leader **CAN** provide useful comment to help the developers make changes
- **DON'T** need to close this request. Just ask the developer to modify and re-commit

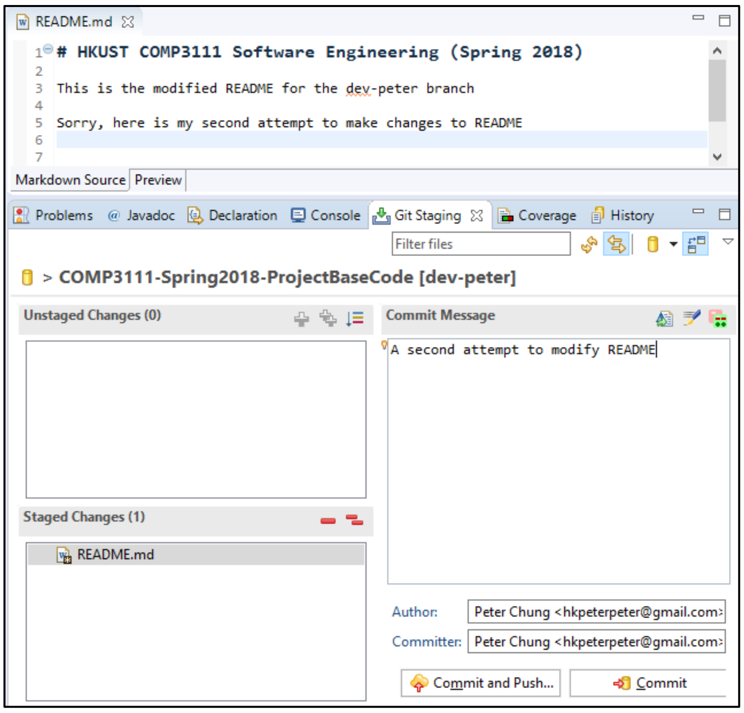

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.
Set up continuous integration to automatically test your code
Catch bugs, enforce style, and increase confidence in your code before you merge.

Hey, you basically modified the stuff wrongly! Please make changes and commit again! I won't merge your **rubbish** changes
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Styling with Markdown is supported

Step 3.4: Modifying the development branch and commit

- Developer can keep modifying and commit the development branch
- To save the number of button clicks, you can choose "Commit and Push…" to the remote branch directly



```
# HKUST COMP3111 Software Engineering (Spring 2018)

This is the modified README for the dev-peter branch

Sorry, here is my second attempt to make changes to README
```

COMP3111-Spring2018-ProjectBaseCode [dev-peter]

Unstaged Changes (0)

Staged Changes (1): README.md

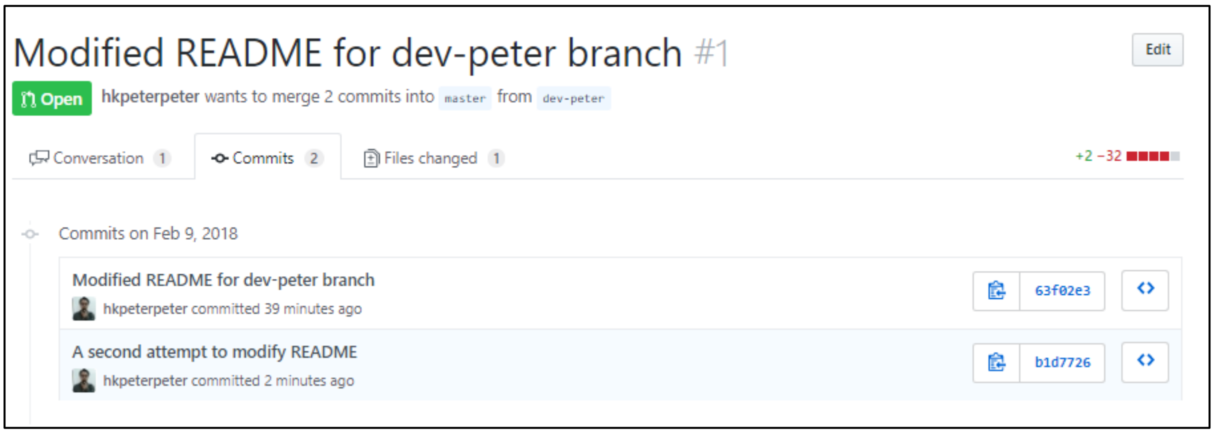
Commit Message: A second attempt to modify README

Author: Peter Chung <hkpeterpeter@gmail.com>

Committer: Peter Chung <hkpeterpeter@gmail.com>

Step 3.5: Further review and comment

- In this project, we recommend to have the team leader as the ONLY person to merge changes to the master branch. However, for the lab work, please allow everyone (except the TA) to have a temporary power to approve a pull request.



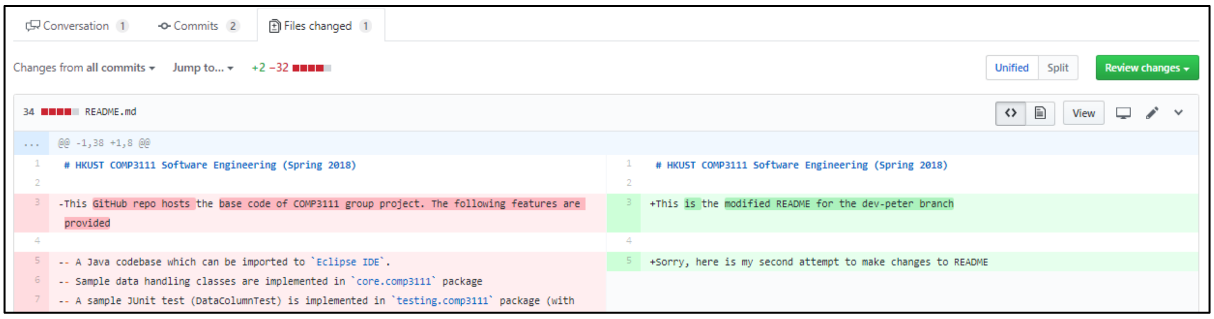
Modified README for dev-peter branch #1

Open hkpeterpeter wants to merge 2 commits into `master` from `dev-peter`

Commits on Feb 9, 2018

Modified README for dev-peter branch
hkpeterpeter committed 39 minutes ago — 63f02e3

A second attempt to modify README
hkpeterpeter committed 2 minutes ago — b1d7726



Changes from all commits ▾ Jump to... ▾ +2 −32

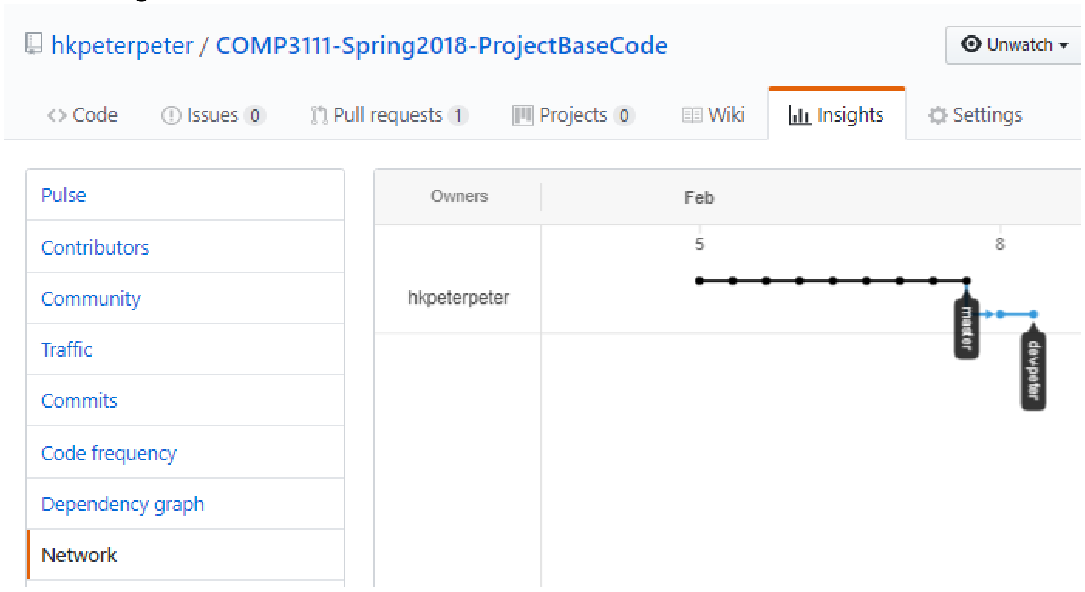
34 README.md

```
@@ -1,38 +1,8 @@
 # HKUST COMP3111 Software Engineering (Spring 2018)      | # HKUST COMP3111 Software Engineering (Spring 2018)

-This GitHub repo hosts the base code of COMP3111 group   | +This is the modified README for the dev-peter branch
 project. The following features are provided             |

-- A Java codebase which can be imported to `Eclipse IDE`. | +Sorry, here is my second attempt to make changes to README
-- Sample data handling classes are implemented in `core.comp3111` package
-- A sample JUnit test (DataColumnTest) is implemented in `testing.comp3111` package (with
```

Step 3.6: Review the branches at GitHub

GitHub provides a nice visualization tool to help you understand the commit history of different branches (available to pro account only)

- Select Insights > Network



Lab Activity

- Create another label in different location and different color. If your teammate has already done it, change the color and the location. If you have started your project already, you don't need to do the above one. Just show us part of your work.
- Create a branch and commit the change.
- Push the change to Github.
- Make a pull request and comment it as "Trivial PR" (That is, this PR does not count towards the project PR).
- Approve the pull request.
- Try to *synchronize* your local repository with Github again.

Assessment

- Show us you can compile and run the program in your laptop with the changes you made.
- Show your team github (has to be private) to us such that there are four people in it (including three members of your team and the TA account comp3111ta).
- Show us your PR approval.

Note:
You need to do some survey on how to approve a pull request and how to synchronize things from Github to your local repository.

Some Git Learning Tips – other than the Lab 3 slide.

Topics	**Resources**
Concept	What is git – Atlassian
	Video: Git Basics Episode 2 – What is Git?
	Web: Video: Git Basics Episode 3 – Get Going with Git
Basic	Web: Git Basics
	Interactive Tutorial – by Code School
	Interactive Tutorial – by Atlassian
	Web: git for beginners
Branching	Video: Branching – by Codemy School
	Video: Merge – by Codemy School
	Interactive Tutorial – by learngitbranching.js.org
	Web: Using Branches – by Atlassian
	Interactive Tutorial: Branching – by Atlassian
Cheatsheet	Atlassian's Cheatsheet
	GitHub's Cheatsheet