CITS5501 Software Testing and Quality Assurance
Github Primer for Software Quality Assurance

Unit coordinator: Rachel Cardell-Oliver

Introduction

Review

In the previous lecture we explored what makes a software project high (or low) quality and some ways to characterise SWQ.

Professional software engineers use version control systems to help manage software development and quality.

This workshop is a hands-on primer / revision session on using Github for project management, and particularly for managing software quality assurance.

Github and SQA

Professional software engineers use **version control** systems to help manage software development and quality.

The most widely used system (although not necessarily the best) is Github. So we will be using Github in the project for this unit.

Most of you will have used Github (to some extent) before.

Using Github effectively, especially with groups, requires care to ensure quality and avoid conflicts.

Github Training

You need your own **GitHub account** for this workshop and the project.

Go to https://docs.github.com/en/get-started/start-your-journey/creating-an-account-on-github for set up instructions if required.

This workshop is based on the Github Hello World tutorial.

You can find many other useful guides online.

See the recommendations at the end of this lecture.

Email me at cits5501-pmc@uwa.edu.au if you come across other good guides.

Github in a nutshell

1. Repositories and Commits

Repositories

A **repository** is a folder that contains related items, such as files, images, videos, or even other folders.

A repository usually groups together items that belong to the same "project" or thing you're working on.

Github is best for managing **code files** so you may want to store other artifacts elsewhere.

Every repository should include a **README** file giving a brief description of the purpose of the project and how to use and run it.

Repository Settings

Repositories for your university work should be **PRIVATE** to you, group members (for group projects) and your facilitator or UC (for marking, as required).

Make sure you select the PRIVATE option when you set up your repository.

Making your repository PUBLIC means your code is visible to other students. This is **academic misconduct on your part**.

Local and Remote Repositories

A *remote repository* is hosted on a remote location and is shared among multiple team members.

The repository you create for your team in https://github.com/ will be your remote repository.

A **local repository** is hosted on a local machine for an individual user.

You make changes and test your code locally and then commit to the remote repository branch or make a pull request to commit to the main branch when you are ready.

Beginners are advised to use Github Desktop to manage your local repository, but you may prefer to use the command line interface. See Using Git from the Command line

Commits

On GitHub, saved changes are called **commits**. Commit copies local changes you have made to project files onto the repository in GitHub (that is the online groundtruth version of your repository).

Each commit has an associated **commit message**, which is a description explaining why a particular change was made.

Commit messages capture the history of your changes so that other contributors can understand what you've done and why. It is important to take the trouble to write meaningful commit messages or your team may waste much time later trying to work out what was done and why.

2. Issues and Branches

Branching

Branching lets you have different versions of a repository at one time.

By default, your repository has one branch named main that is considered to be the definitive branch. The code on main should always be runnable, checked, and correct.

You can create additional branches from main in your repository. These branches are used for programming work: implementing new functionality and fixing bugs.

Commit changes (only) to your branch until everything is working and you are ready to merge your updates to the main branch.

Issues

Issues are used to define tasks to be carried out for your software project. You should create issues in your repository to plan, discuss, and track work. Tasks can include reporting bugs, planning work, collecting feedback and tracking ideas.

Issues can have **subtasks** to break down the work and **labels** to categorise the task (eg bug fix vs new functionality). You can create templates (a sort of checklist) for defining issues.

Issues can be **assigned** to particular team member(s).

Software development work for a issues should be done in its own branch (see above).

Discuss and review issues before you start work to minimise the risk of merge conflicts (see below).

See Quickstart for GitHub Issues

3. Pull requests and Merging

Pull Request

Pull requests are the heart of collaboration on GitHub.

When you open a **pull request**, you're proposing your changes and requesting that someone review and pull in your contribution and **merge** them into their branch.

Remember that the main branch is considered to be the definitive branch. The code on main should always be runnable, checked, and correct.

When you start collaborating with others, a pull request is the time you'd ask for their **review**. This allows your collaborators to comment on, or propose changes to, your pull request before you merge the changes into the main branch.

A pull request SHOULD generate discussion and usually some changes to the code to improve its quality.

For your project you should explicitly define standards to be checked

Merging (and Merge Conflicts)

In this final step, you will **merge** your branch changes into the main branch.

It is the responsibility of the author and reviewers to ensure this is done carefully or problems can be introduced which are much harder to fix later.

Sometimes, a pull request may introduce changes to code that conflict with the existing code on main. This is caused a **merge conflict** and can be very painful! You need to resolve the conflicts by making changes to the code and discussion with your collaborators.

Plan your tasks and branches to minimise the risk of merge conflicts.

Once you and your collaborators have checked that code from a github branch select *Merge Pull Request* and finally *Confirm merge*.

In professional settings, you can delete a branch once the changes

Further reading

Github Training

A step by step guide to the key concepts is available: Github Hello World Step by Step. Make sure you work through this tutorial by yourself to consolidate your knowledge of today's topics.

Beginners are advised to use Github Desktop for the local repository, but you may prefer to use the command line interface. See Using Git from the Command line

The Basics of GitHub includes a useful list of resources for further training.

ACS Guides

ACS: Excellent introductory courses by Kishan Iyer Git and GitHub: Introduction Git and GitHub: Working with Git Repositories Git and GitHub: Using GitHub for Source Code Management

Join the Australian Computer Society (ACS) for free to access these (and many more) training resources.
Go to http://www.acs.org.au/join-acs.html select **Student Supported ICT Membership** This is free for UWA students. See also the posters around the Department.