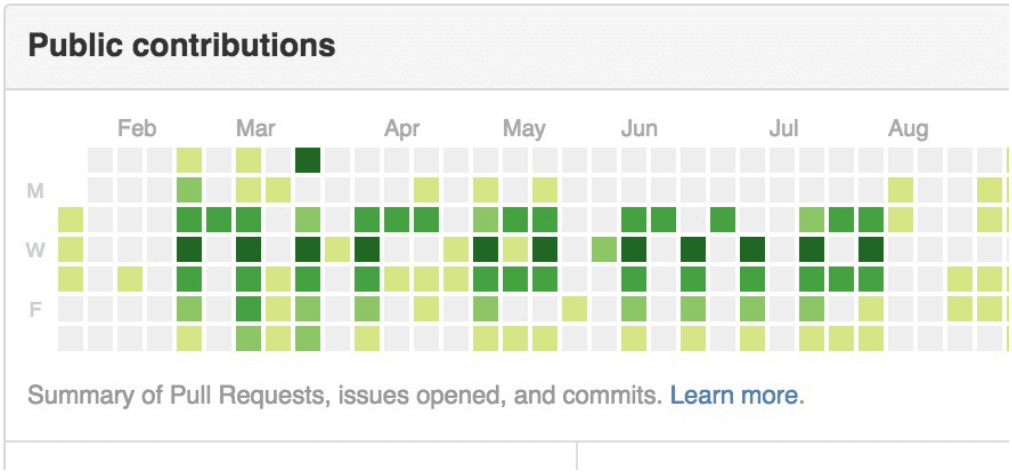
School of Computing and Information Systems
The University of Melbourne
SWEN900XX
Software Project (Semester 2, 2021)

Dr Eduardo Oliveira

Recommendations on the use of GitHub (for your own benefit)

Let's start doing the right thing since the very start of this subject.



To work well and to be able to show case your experience with the industry, we listed a few things for you to keep in mind:

- Keep your GitHub professional. Treat this as a real-world project. Add a photo to your profile (no avatars);
- Keep your repo's README.md updated
    - What is that repository about?
    - Is there a link to a live demo for that project?
    - Highlight the contents of your project (summary)
    - Setup and Configuration details
    - Changelog
        - keep track of the new changes performed after every release
    - License details

        - The text of a license is usually stored in the LICENSE (or LICENSE.txt, LICENSE.md) file in the root of the project.
- Commit often
    - The smaller the changes, the easier it is to integrate the code later. Also, if everyone in the team is committing regularly, everyone is more likely to have the most up to date code
    - Each commit should be a single conceptual unit of change. Even if two changes happen in the same file, if they relate to two different bugs for example, they should go in different commits. Conversely, if changes in two separate files are related to the same bug, they should go together in the same commit
- Commit completed work
    - Don't commit half-arsed code. Make sure you test it before committing. At least the code should compile.
- Write good commit messages;
    - Use the imperative mood in the subject line
    - A properly formed Git commit subject line should always be able to complete the following sentence:
    - **If applied, this commit will** *your subject line here* (this should be your commit message. You do not need to write 'If applied…')
      For example:
    - **If applied, this commit will** *refactor subsystem X for readability*
    - **If applied, this commit will** *update getting started documentation*
    - **If applied, this commit will** *remove deprecated methods*
- Branch before you build
      The naming convention simply adds prefixes to branch names, so that branches of the same type get the same prefix
    - feature/username/task_name
    - bugfix/username/task_name
    - test/username/task_name
      This lets you search for branches in many git commands, like this:
    - git branch --list "feature/*"
    - git log --graph --oneline --decorate --branches="feature/*"
    - git --branches="feature/*"

- Structure your folders
    - A typical top-level directory layout[1]

```
.
├── build                    # Compiled files (alternatively `dist`)
├── docs                     # Documentation files
├── src                      # Source files (alternatively `lib` or `app`)
├── test                     # Automated tests
├── tools                    # Tools and utilities
├── LICENSE
└── README.md
```

Use short lowercase names at least for the top-level files and folders except LICENSE, README.md.

The actual source files of a software project are usually stored inside the src folder. Alternatively, you can put them into the lib (if you're developing a library), or into the app folder (if your application's source files are not supposed to be compiled).

Automated tests are usually placed into the **test** or, less commonly, into the **spec** or **tests** folder.

```
.
├── ...
├── test                     # Test files
│   ├── benchmarks           # Load and stress tests
│   ├── integration          # End-to-end, integration tests
│   └── unit                 # Unit tests
└── ...
```

Often it is beneficial to include some reference data into the project, such as Rich Text Format (RTF) documentation, which is usually stored into the **docs** or, less commonly, into the **doc** folder.

```
.
├── ...
├── docs                     # Documentation files (alternatively `doc`)
│   ├── TOC.md               # Table of contents
│   ├── faq.md               # Frequently asked questions
│   ├── misc.md              # Miscellaneous information
│   ├── usage.md             # Getting started guide
│   └── ...                  # etc.
└── ...
```

Same is applied to tools.

- Deploy it
    - Have your project ready to show! One of the worst possible things is to show up to an interview and offer to show your super cool website and then have to do a bunch of debugging or configuration in order to launch it (true story).

[1] Source for folder structure conventions: https://github.com/kriasoft/Folder-Structure-Conventions

Deploy your project and have it publicly available via URL (keep the URL simple and easy to type for non-tech people).

- Release
    - At the end of each sprint, generate a RELEASE TAG from your repository (master branch). Example: SWEN900XX_2021_Sprint<1>_<team name>
    - This tag includes ALL the branches that were developed and completed during that Sprint;
    - This is the tag that should be DEPLOYED and presented to the client (while development continues in the repository);