CITS4407 Open Source Tools and Scripting
Semester 1, 2021
Week 10 workshop – Build automation

Before starting this workshop, make sure you've reviewed the recommended reading for weeks 1–9, and completed the lab sheets for weeks 2–8.

0. Required software

For this lab, you will need to ensure the `make` package is installed on Ubuntu. (You should know how to install Ubuntu packages from previous labs.)

You also will need to download Pandoc from the webpage at https://github.com/jgm/pandoc/releases/tag/2.13 and place it on your PATH. (You should know how to download Pandoc and amend your PATH from last week's lab.)

1. Makefiles

Make is a build automation tool, which uses a configuration file called a Makefile. A Makefile contains rules consisting of a *target*, *dependencies*, and a *recipe*, describing what to build, and how to build it. Make uses the modification time of prerequisites to re-build targets only when needed.

Sometimes targets are *files* that need to be created; but sometimes they are best thought of as naming a set of tasks we would like done.

Building software artifacts using Makefiles has several advantages:

- Make is widely available
- Make is a standard build automation tool, which many software developers are familiar with. Using `make` instead of a less standard tool makes it easier for other people to build your project.
- Makefiles are written in plain text that is easily human-readable and -editable (at least when Makefiles are kept simple).
- Make does not place any restrictions on what tools or compilers your project uses.

2. Creating a Makefile

Clone the repository at https://github.com/cits4407/workshop08.git, and `cd` into it. (You should know how to do this from previous labs.)

Inside, you will find:

- an empty Makefile
- a data file, `data/ENROLMENTS-2017`.

In this lab, we will create a report written in Markdown using data contained in the data file, then convert it to HTML using Pandoc.

Edit the Makefile using your preferred editor. Add the following rule:

```
geng5505-students: data/ENROLMENTS-2017
    ./create_listing data/ENROLMENTS-2017 > geng5505-students
```

Note that on the second line of this rule, the first character **MUST** be a *tab* character (not a space). You can check whether it is by running the following:

```
$ cat --show-tabs Makefile
```

If you have correctly typed the rule, it should show up with the tab marked as "`^I`":

```
geng5505-students: data/ENROLMENTS-2017
^I./create_listing data/ENROLMENTS-2017
```

Try running "`make geng5505-students`". You should see that

(a) Make reports that an error occurred – the command `./create_listing` is not found, since we have not created it yet.
(b) An empty file `geng5505-students` is created (since that's where we redirected output to).

Try running the same command again; you should see `make` claiming that "`geng5505-students` is up to date" – can you explain why?

This is undesirable behaviour – when something goes wrong with the `create_listing` command, we don't want the target to be created.

Delete the empty `geng5505-students` file, and edit the Makefile by adding the following (either before or after our existing rule, but at the start of the file is conventional):

```
.DELETE_ON_ERROR:
```

Try running `make geng5505-students` again; this time, you should see that

(a) Make reports that an error occurred – the command `./create_listing` is not found, since we have still not created it.
(b) ***No*** empty file `geng5505-students` is left behind – `make` deletes it.

`DELETE_ON_ERROR` is a special target used by GNU Make; when it is added to a Makefile, `make` alters its normal behaviour.

3. Preparing data

You will need to create a script, `./create_listing`, which:

- Takes an argument on the command-line – an input file to read.
- Outputs (on standard output) all lines in the file containing the string `GENG5505-1`.

You should be familiar with how to do this based on previous workshops on creating scripts.

Once your script is ready, try running `make geng5505-students` again – if you have written the script correctly, the file `geng5505-students` should be created, containing the data we want.

Make special variables

The rule

```
geng5505-students: data/ENROLMENTS-2017
    ./create_listing data/ENROLMENTS-2017 > geng5505-students
```

could be made simpler – it needlessly repeats the name of the target, `geng5505-students`. Based on the material on `make` contained in the lecture slides and the lecture readings, how would you amend the Makefile to avoid this? (Avoiding unnecessary repetition in source code and data is sometimes called the "DRY" Principle – "Don't Repeat Yourself". The opposite of "DRY" is "WET" – "Write Everything Twice" or "We Enjoy Typing".)

(Another question to consider in your own time: what about the "`data/ENROLMENTS-2017`" file – can we avoid mentioning it twice?)

4. Creating a report

Add a new rule to your Makefile:

```
geng5505-students.md: geng5505-students
    ./create_report > geng5505-students.md
```

Your task is to write a script, `create_report`, which will produce a report written in Markdown – the `geng5505-students.md` file.

It should contain:

- A header, "GENG5505 Report"
- A summary line: "No. of GENG5505 students: n" (where n is the number of lines in our `geng5505-students` file).
- A *bulleted list* of all the student numbers (but *not* the GENG5505 code).

5. Self-study exercise

Try this in your own time (or in the lab, if you have time available).

Create a new rule which has as its target a file `output/geng5505-students.html`, and as dependency the file `geng5505-students.md`. The *recipe* in the rule should invoke Pandoc, to create HTML output from the Markdown file.

Lastly, try *generalizing* this rule.

Rather than having `geng5505-students.md` specifically as a dependency and `output/geng5505-students.html` as a target, amend your rule so it will work for *any* `.md` file in the current directory, and create a corresponding HTML file in the `output` directory.