CITS4407 Open Source Tools and Scripting
Semester 1, 2021
Week 9 workshop – Markdown

Before starting this workshop, make sure you've reviewed the recommended reading for weeks 1–8, and completed the lab sheets for weeks 2–7.

1. Installing software

In previous labs, we have seen that the `apt-get` command can be used for installing software packages on Ubuntu. These software packages are downloaded from *software repositories* (not the same as Git repositories) – you can look in the file `/etc/apt/sources.list` to see their addresses. You should see URLs like http://archive.ubuntu.com/ubuntu/ in the file; the `apt-get update` command queries the servers at those addresses to obtain fresh information about what packages are available, and the `apt-get install` command downloads packages and adds them to your system.

Sometimes software we would like to use isn't contained in those repositories. However, there are many other ways of installing software on Ubuntu, and we will discuss three of them.

Single-file scripts. If the software consists of a single *script* – a text file like the ones you have written for the assignment, written in a *scripting language* (like Bash or Python) – we can simply download the script, and run it by giving the full path to that script.

For instance, the `numbers2` script from the week 3 workshop can be downloaded from the web page at https://github.com/cits4407/example/blob/master/numbers2 (from the link labelled "raw" on that page).

We can download and run it with the following commands:

```
$ wget https://raw.githubusercontent.com/cits4407/example/master/
   ↪ numbers2
$ ./numbers2
```

(The `wget` command downloads files from the web; it is covered in the Shotts textbook in chapter 16, "Networking".)

Single-file binaries. Scripts consist of human-readable plain text, but programs in some languages are instead *compiled* into machine instructions which are not human-readable, often called "binaries" or "binary executables". These, too, we can download and run, but we need to ensure that the binary is designed to run on the type of machine we are using.

(For instance, Linux can run on the Raspberry Pi, a tiny credit-card sized computer often used by hobbyists; but machine instructions for the Raspberry Pi will be different from the ones used by most laptops.)

Scripting language packages. If a program consists of just one script file, then we can just download and run it, as we have seen. But a program may be large enough that it is best broken into multiple files, or might depend on other programs or libraries.

Because of this, many scripting languages have their own software repositories, which allow you to download programs and libraries written in languages like Python, Ruby, JavaScript, and Perl. Each language will have its own tool for installing packages written in that language – we will use the tool used by Python, called `pip`.

2. Writing Markdown

We will create a file written in the Markdown markup language, and show how to convert it to other formats. In a text editor, create a text file named "my-startup-plan.md". (Text files can always be called anything you like, but conventionally, Markdown files are given the extension "`.md`" or "`.markdown`".) Add the following text to the file:

```
---
title: Startup plan
author: YOUR NAME
date: 29 April 2021
fontfamily: times
---

# Introduction

Description of the plan for my new startup.

# The plan

1. Come up with great idea
2. Create business
3. Profit $$$
```

Save the file.

3. Converting to HTML and Word with Pandoc

We will now download the `pandoc` tool, which can be used for converting between a wide range of formats – including MS Word, Web pages (HTML), and PowerPoint.

You can download Pandoc from the web page at https://github.com/jgm/pandoc/releases/tag/2.13; the file you want (if you are running on a lab machine or most laptops) is called

`pandoc-2.13-linux-amd64.tar.gz`. (If you are running Ubuntu on a recent MacOS X computer, you may instead need to use `pandoc-2.13-linux-arm64.tar.gz`).

This file is compressed, and contains multiple files within it (like a Zip file), so we need to extract its contents using the `tar` program (covered in Shotts chapter 18, "Archiving and Backup"):

```
$ tar xvf pandoc-2.13-linux-amd64.tar.gz
```

You should now be able to find the `pandoc` binary by typing (for instance):

```
$ ls ./pandoc-2.13/bin/pandoc
```

And you can convert your Markdown file to HTML by typing:

```
$ ./pandoc-2.13/bin/pandoc --from markdown --to html --output=my-
   ↪ startup-plan.html  my-startup-plan.md
```

If you list the contents of the current directory, you should now see a new file pandoc has created, `my-startup-plan.html`. You can open this in a web browser (using "File / Open" from the menu); if you use WSL2 on Windows, get your lab instructor to show you how to find where the file is located.

(We can also convert to MS Word format (`.docx`), by running `./pandoc-2.13/bin/pandoc --from markdown --to docx --output=my-startup-plan.docx my-startup-plan.md`, but you will need to have MS Word, the open source LibreOffice software suite, or similar software installed to open it.)

4. Converting to PDF with Pandoc and WeasyPrint

We can also convert our Markdown file to PDF format – the lab worksheets are created in this way. In addition to Pandoc, we will need a Python library called "WeasyPrint". The lab worksheets use a typesetting system called LaTeX, but WeasyPrint is smaller and easier to install.

First install the Python language, its package management tool `pip`, and several graphics libraries:

```
$ sudo apt-get update
$ sudo apt-get install python3 python3-pip libcairo2 libpango-1.0-0
   ↪ libpangocairo-1.0-0
```

Then install the WeasyPrint package for Python:

```
$ pip3 install --user WeasyPrint
```

We can now convert from Markdown to PDF:

```
$ export PATH=~/.local/bin:$PATH
$ ./pandoc-2.13/bin/pandoc --from markdown --to html --pdf-engine=
   ↪ weasyprint --output=my-startup-plan.pdf  my-startup-plan.md
```

We need the `export` command, which adjusts our `$PATH` environment variable, because by default, `pip` does not put new programs in (for instance) `/bin` or `/usr/bin`, where many system programs are found. (In fact, you cannot add programs to those directories unless you use `sudo`, because non-`root` users don't have permissions to write files to those directories.) Instead, it puts them in our home directory, within a directory called `.local/bin`. The `export` command tells Bash to look in that directory for executable programs.

The PDF output can be customized in many ways, but we will not cover them here – see if you can find out more about how to do so.

5. Other Markdown features

See if you can find out (Google is a good place to start!) how to include the following features in your Markdown documents:

- bulleted lists
- images
- mathematical equations

Create a Markdown document that includes these, and convert it to PDF.