CITS4407 Open Source Tools and Scripting
Semester 1, 2021
Assignment 1

Details

Version: 0.1
Date: 2021-04-07
Please check the CITS4407 website or the Git repository at https://github.com/cits4407/assignment1 to ensure that you have the latest version.

The goal of this assignment is to assess your understanding of concepts covered in lectures and practised in lab/workshops.

- The assignment contributes **20%** towards your final mark this semester, and is to be completed as individual work. It is marked out of 20.
- The deadline for this assignment is **23:59 pm, Sunday 25th April**.
- The assignment is to be done individually.
- The submission procedure will be announced on the Help4407 forum and in lectures, and published in the next version of this assignment spec.
- You are expected to have read and understood the University Guidelines on Academic Conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this assignment, but the work you submit must be the result of your own effort.
- You must submit your assignment before the submission deadline above. There are Penalties for Late Submission (click the link for details).

Background

You are employed by Minotoro, a company that contracts to build hedge mazes, labyrinths and Halls of Mirrors for customers. You work as a Maze Quality Control Officer.

Your manager, Stephen Dedalus, has provided you with a Bash script, `maze_gen`, which generates plans for mazes. The mazes are made up of 3-by-3 *cells* with either adjoining walls, or corridors between them.

Running `maze_gen` ***width height*** will generate a maze plan which is *width* cells wide and *height* cells high. For instance, the following maze was generated by running `maze_gen 7 5`:

```
###############
              #
# # # ####### #
# # #       # #
# # ### #######
# #   #       #
# # ##### ### #
# #     #   # #
# ####### ### #
#       #   #
###############
```

If the height and width of a maze, in terms of cells, are h and w, then the printed plan will have $w \times 2 + 1$ columns and $h \times 2 + 1$ rows.

The maze plans Minotoro produces should satisfy the following requirements (we assume maze plans are always oriented with north at the top):

Maze plan rules

1. The maze plans should contain only space characters, and the hash (“`#`”) character.
2. The top-left cell should have an entry to the maze in the western wall.
3. The bottom-right cell should have an exit from the maze in the eastern wall.
4. There should be no other entries or exits from the maze.
5. The maze should always have the height and width (in terms of cells) that the client has requested.
6. There should always be a route through the maze from the entry to the exit.
7. There should always be a route from any cell to the exit.
8. The maze should contain no loops – for instance, the following maze is a badly-formed one –

   ```
   ###########
        .....#
   # ###.###.#
   #   #.....
   ###########
   ```

   – since it contains a loop (marked with full stops).

It is also possible to use the `maze_gen` script to generate some mazes which are known to be nearly always badly-formed – they breach one or more of the above rules. It is possible a well-formed maze might be constructed by accident, but this rarely seems to happen for mazes of any reasonable size (e.g. more than about 5 cells in height).

Calling the `maze_gen` program as follows, with the “`-b`” flag –

```
$ maze_gen -b 10 6
```

– will generate a (probably) badly-formed maze.

You will be required to write several Bash *quality control scripts*, which are designed to test whether mazes are well-formed or not, according to the above rules.

Note that the `maze_gen` script is only likely to work with Bash version 4.3 or greater (you can check what version of Bash you are using by running "`bash --version`"), and when standard Linux utilities installed. We will ensure everyone has access to a standard Ubuntu 20.04 environment, with the correct version of Bash and all required utilities, for testing their programs on.

Tasks

You should clone the repository at https://github.com/cits4407/assignment1, which contains a copy of the `maze_gen` script, as well as some tools we will use for testing your programs.

It also contains files containing "stubs" for the scripts you will need to write – these are script files which currently do very little, and which you will need to fill in code for. These are called "`quality-check-script-01.sh`", "`quality-check-script-02.sh`", and so on.

The scripts all should read from standard input, and output their results to standard output. In general, they should just print the word "`yes`" if the maze they are given passes the rules they are designed to check, and the word "`no`" if it does not.

You should be able to try out one of your scripts by typing (for instance):

```
$ maze_gen 7 5 | quality-check-script-01.sh
```

If any script is invoked differently, we describe it below.

The scripts you will need to write are as follows:

1. **`quality-check-script-01.sh`**: Check that maze plans supplied on standard input contain only space characters and hash characters ("`#`") – **6 marks**.
2. **`quality-check-script-02.sh`**: Check that maze plans supplied on standard input have the correct number of rows and column – **4 marks**.

   This script can be invoked as (for example):

   ```
   $ maze_gen 7 5 | quality-check-script-02.sh 7 5
   ```

3. **`quality-check-script-03.sh`**: Check that maze plans supplied on standard input have an entry and exit as specified by rules 2 and 3 – **3 marks**.
4. **`quality-check-script-04.sh`**: Check that maze plans supplied on standard input have no other entries or exits besides the ones specified by rules 2 and 3 (this is rule 4) – **2 marks**.

5. `quality-check-script-05.sh`: Check that maze plans supplied on standard input always have a route from start to finish, as required by rule 6 – **2 marks**.

Make sure you **do not re-name** the quality control scripts. Part of the marking of the assignment is automated, and relies on the scripts having correct names.

Scripts 1–4 should only require commands and Bash features we have used in lectures or lab/workshops. Script 5 might require you to do some research into other Bash features.

Automated tests

You do not have to make use of them, but if you want to see what sort of tests we might run on your scripts, you can `cd` into the directory `tests` of the repository and execute `./run_tests.sh`.

The script should run on any Ubuntu 20.04 environment that has the expected tools installed. It runs the tests and shows how many pass or fail (as well as how long the tests took to run).

Don't be worried that some of the tests show up as failing! That is to be expected – you haven't written your scripts yet, so we *expect* at least some of them to fail. (And most of the ones that pass must be passing only by accident.)

As you work through the assignment, hopefully more of these tests should pass.

The tests are not written in Bash, but in a language called Perl, which is often more convenient than Bash for writing tests in. There is no need for you to understand how the tests work, but you are welcome to take a look if you are interested.

After assignments are submitted, we will run these as well as other, more stringent, tests to see how well your scripts perform on potentially unusual cases.

Extension tasks

Up to 2 marks are available for *extension tasks*, which are awarded at the discretion of the marker. If you lose marks on some tasks, these marks can increase your total (up to a maximum of 20). In general, 1 mark is awarded for an interesting and/or useful task, and 2 marks for an excellently implemented and very challenging task.

If you would like to attempt an extension task, make sure you

- check with a facilitator or the unit coordinator that it is appropriate
- make a directory called "`extension`" to put your extension script in
- call your extension script `extension.sh`
- add a script called `demo.sh` which shows how to run your extension script, as well as a README file describing what it does.

Environment

All submitted programs should run on an Ubuntu 20.04 Linux distribution with packages up to date as at April 2021.

A standard **reference environment** is published at https://hub.docker.com/r/adstewart/cits4407-2021-env – submitted programs should assume the packages installed are exactly the ones in version 0.1.3 of that environment.

We will provide more details shortly about how you can access this standard environment.

Assessment

In addition to the 17 marks awarded for the scripts, 3 marks are awarded for how clear and concise your code is.

Tips

Recall from lectures that the `shellcheck` command can help you spot errors in your program – it is a good idea to run it over your code.

Invoking `declare -p` *`some_var`* will produce nicely printed output of a variable, even if the variable is a Bash array or associative arrays (not covered in class, but you might find them useful for the final script).

Invoking `declare -f -p` *`some_function`* will do the same for Bash functions.

It is a good idea to start work on the assignment *early*, run your code often, and make sure you commit changes to your Git repository periodically.

Make sure you do ***not*** store your code in a *public* GitHub repository – that breaches the University's rules on Academic Conduct.

Good luck!