COMP2129 Assignment 1

Due: Week 5: Friday 12th of April, 11:59PM

This assignment has been written both to test your knowledge of **bash** shell commands and unix utilities, and also to help you extend that knowledge. Unless otherwise stated, all questions can be solved with tools mentioned in lectures, however you may need to refer to the relevant **man** pages to discover how to use the tools. You are welcome to discuss your approaches to solving the questions with other students and on Piazza, however the work you submit must be your own (as with all university assessments). [1] You may help each other learn to use **bash** but do not share your code with other students.

Task 1 (2 marks)

Write a **bash** script called **init.sh** that you could use to initialise a directory structure for your university subjects at the start of a semester. The script should take a single argument (the name of a subject), and create a directory with that name in the working directory, and 13 subfolders called **week1**, **week2**, and so on.

For example, when run as follows:

```
$ ./init.sh comp2129
```

Your script should create a folder called **comp2129** with empty folders for weeks 1 through to 13.

You should use the **test** command to check first if the folder exists, and if it does, print the message:

```
There is already a folder for that subject
```

If there is a file there that is not a directory, your script should behave as follows:

```
$./init.sh info3404
There is already a file called info3404
```

Task 2 (2 marks)

Imagine for this task you are working with somebody else's code and you're having trouble remembering the name of a particular function you need, or what file it is in. You can remember a few things about it: it is in a `.c` file, and it starts with `print`, and then there is a four letter word starting with

[1] http://sydney.edu.au/engineering/it/current_students/undergrad/policies/academic_honesty.shtml

a capital letter. Write a script called **`grep_printer.sh`** which will list possible candidates for this function that appear in any C source files in the current directory.

Note: writing a bash script or regular expression to exactly match valid C function definitions and *only* valid C function definitions would be quite a challenge, and beyond the scope of this course. There are better tools for that kind of task. However, is quite handy in a variety of circumstances to be able to quickly type up an expression to search a code base for a certain pattern, and that is the skill we want you to practice here.

For this question, if you are thinking about tricky cases like when function names occur inside strings then you can stop and read about "worse is better": `http://en.wikipedia.org/wiki/Worse_is_better`.

Task 3 (2 marks)

Later in this course you will learn about different tools for *version control*. Version control is important for tracking changes to a software project by storing the project and the history of the project in a *repository*. One such tool for version control is called Mercurial. For the purposes of this question, a mercurial repository is defined as a directory that contains a **`.hg`** directory. So, in the example in Figure 1, the name and location of the mercurial repository is **`/home/georgina/projects/colours/`**.

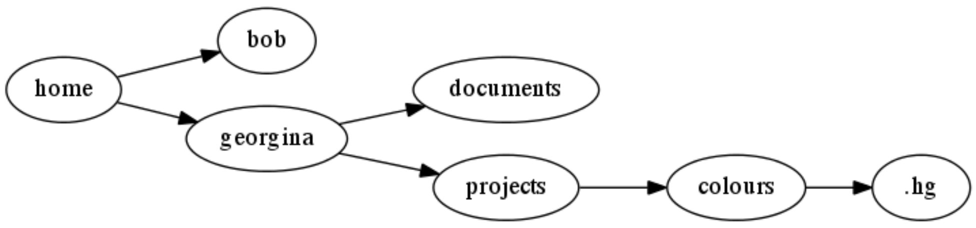

Figure 1: An example directory structure

Write a script called **`find_repos.sh`** that will recursively search the current directory and print out the name and location of any mercurial repositories, as defined above. For example for the directory structure above, if your script was run from the folder **`/home/georgina`** it should output:

```
./projects/colours/
```

You do not need to worry about cases where there might be multiple directories called **`.hg`** inside each other.

Hint: `sed`

You may find the command **`sed`** helpful for this question. **`sed`** is a *stream editor*, that is it can be used to filter streams of text on a line by line basis, replacing a given pattern with a designated string. It's basically like a kind of super awesome multi-purpose "find and replace" that will work on the output of your **`bash`** scripts . For example , let's say you are building a one-liner that will take the words in **`/usr/share/dict/words`** that contain the string `ping` and replace all instances of `ing` with `ed`, but only when the `ing` appears at the end of a word. Well, then you could do the following:

```
$ grep "ping" /usr/share/dict/words | sed 's/ing$/ed/'
```

and you would get output like:

```
anticreeped
antidumped
antishipped
... and so on ...
zipped
zippingly
```

Task 4 (2 marks)

You may have noticed that in Assignment 0 we provided you with a `Makefile` that would allow you to run some sample tests on your code. For this task, we would like you to write a script that uses **`diff`** and **`basename`** to do something similar.

Your script should be called **`run_tests.sh`** and should take a single argument on the commandline (the name of the program to be run) and rely on there being a folder called **`tests`** in the current directory. Inside the **`tests`** folder, each test is represented as a pair of files: **`testname.in`** and **`testname.out`** (where **`testname`** is the name of the test). The program to be run is considered to pass the test if given the input of **`testname.in`**, the output is exactly **`testname.out`**. For each test, your program should output `Testing: testname`, where `testname` is the name of the test, and then run that test. Your program should print nothing if the test passes, and for each test that failed it should print a one line message stating that it failed, and then continue to the next test. For example, if there were three tests (i.e. six files) in the **`tests`** directory, then your program might run like this:

```
Testing: sample1
Testing: sample2
Files - and tests/sample2.out differ
Testing: sample3
```

Hint: basename

The program **`basename`** was not covered in lectures or tutorials: please refer to the **`man`** page for help with this question.

Task 5 (2 mark)

When working with multiple versions of a project, sometimes it is useful to include statistics about when and where a program was compiled in the actual binary. For example, if you log on to **`ucpu0`** and open the Python interpreter, you will see this:

```
Python 2.7 (r27:82500, Sep 16 2010, 18:02:00)
[GCC 4.5.1 20100907 (Red Hat 4.5.1-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

This is useful because it tells us when it was compiled, what version of GCC was used, and other useful statistics. For this task, write a script called **`version.sh`** that when run compiles a C binary called **`version`** that will print out the following information:

```
$ ./version.sh
$ ./version
Compiled on Fri 15 Mar 2013 16:09 EST
```

Note that the program **`version`** should print the time it was compiled, not the time when it is run. You may find a *here document* useful for this question.