
Java Tick 1*

Introduction

In order to gain a star in the mark sheet you must complete this exercise. Completing the exercise does not gain you any credit in the examination. To get a star you need to produce two files

`answers.txt` a text file with answers to all the questions shown in the question boxes;

`make.sh` a *shell script* which conforms to the specification described below.

To submit your work you should construct a jar file called `crsid-lstar.jar` where `crsid` is your username. The jar file should contain the two files `answers.txt` and `make.sh` inside the directory `uk/ac/cam/crsid/ticklstar`. For example, if your username is `arb33` the jar test routine should output the following:

```
crsid@machine:~> jar tf arb33-lstar.jar
uk/ac/cam/arb33/ticklstar/answers.txt
uk/ac/cam/arb33/ticklstar/make.sh
```

To submit your work, email the jar file as an attachment to `ticks1a-java@cl.cam.ac.uk`. There are no automated tests associated with this Tick but you should receive confirmation of your submission.

Further Bash

In Workbook 1 you explored a few basic commands. Most Linux systems offer a much larger range of commands and programs. In this exercise you will explore some of these and write your own *script* which can be called from the command line. You will find the on-line Bash manual¹ and the Advanced Bash-Scripting Guide² useful since they provide more extensive documentation than `man`.

More commands

1. What do each of the following commands do? Given an example alongside your description.

```
chmod lpr pushd ln tail wc find xargs cut scp
```

2. What do each of the following Bash operators do? Given an example together your description.

```
; < >> &&
```

It can take some time and effort to write a complex expression in Bash and consequently you may wish to keep a record of what you've written for future reference. The best way to do this is to write a script which you can not only read but also execute directly, avoiding the need to type in the command a second time; instead you need only type in the name of the script file. For example, if you wish to save the command

```
crsid@machine:~> echo "Hello, world"
```

then you can create a file called `hello.sh` with the following contents:

¹<http://www.gnu.org/software/bash/manual/bashref.html>

²<http://tldp.org/LDP/abs/html/>

```
#!/bin/bash
echo "Hello, world"
```

The first line of this file contains the special phrase `#!/bin/bash` which tells the operating system to use the program `/bin/bash` to execute the contents of the file; since the operator `#` in Bash turns the remainder of the line into a comment (try `#echo "no output"` in a shell) then this line is ignored, but the commands written into the rest of the file are, broadly speaking, executed as if you had typed them in at the prompt. You can execute the script `hello.sh` by making the file *executable* using the `chmod` command:

```
crsid@machine:~> chmod u+x hello.sh
crsid@machine:~> ./hello.sh
Hello, world
crsid@machine:~>
```

A more advanced script

To complete this Tick you will need to read and learn more about Bash. The introduction section of the Advanced Bash-Scripting Guide mentioned earlier is a good place to start, but there are many other good tutorials on the web. Whichever tutorial you use, to successfully gain this Tick you will probably find the following concepts in Bash helpful: creating and using variables, conditional and loop constructs, positional and special parameters, and shell parameter expansion. (Don't worry if you don't know what these terms mean yet, you probably won't! Use them as a way to steer your initial reading.) The Lecturer and Demonstrator will be able to offer some assistance if you get stuck.

To gain Tick 1* you must produce a Bash script called `make.sh` which conforms to the following specification. Your script should inspect the first argument passed to it when executed and use this value to determine what action to take. For example, if a user types in

```
crsid@machine:~> ./make.sh build
```

the script should perform the "build" action. Your script should support the following actions:

- `show` Print out a list of all files with the extension `.java` found in the current directory or any sub-directory of the current directory.
- `clean` Delete all files which end with the extension `.class` which can be found in the current directory or any sub-directory of the current directory.
- `build` Use the `javac` compiler to create a new `.class` file for *only* those files whose equivalent `.java` file exists *and* the `.java` file has a more recent modification time as recorded by the filesystem. For example, if there exists a file `uk/ac/cam/arb33/Test.class` which was created at 10:42 today, and there exists a file `uk/ac/cam/arb33/Test.java` which was modified more recently (e.g. 10:46), then the Java compiler should be invoked to build a new copy of `Test.class`. Your program should recompile all such `.java` files in the current directory and any sub-directories unless the compilation of a Java source file fails, in which case, after printing out the error message from the compiler, the script should terminate and not attempt to compile any further files.

If no command is given to `make.sh`, the program should perform the "show" action. If the command is not recognised, then the script should print an error message and terminate.

Note: You will discover in future weeks that `javac` supports some of the "build" features described above since `javac` checks for dependencies between Java sources and rebuilds any dependent class files. For example, if you invoke `javac Test1.java` and `Test1.java` uses data or methods provided by `Test2.java` then `javac` will recompile `Test2.java` if the timestamp of `Test2.class` is older than that of `Test2.java`. (A good submission for Tick 1* might take this into account, but this isn't required.)