Writing Tests Using Shell Script

Apr. 29

SWPP 2021

Juneyoung Lee

Testing

- During the project, each pull request should contain tests
  - At least one test for the new feature only
  - A test for end-to-end compilation if you want
- How to write tests?
  - FileCheck, Alive2
  - For dynamic tests: use gtest. It works well! See the link below*

* https://github.com/jenny5546/swpp202001-team4/blob/master/src/test_main.cpp

Running tests, debug, ...

- The skeleton will contain Makefile
- ‘make’ will compile the skeleton, but ‘make test’ will do nothing!
- Please properly update Makefile to successfully run your tests when ‘make test’ is given!
  - A general approach: let’s write a shell script that runs all tests.

Shell script

- sh: A programming language specification
- bash, csh, zsh: implementation of sh
  - With their own extensions
- Shell script: a script that is written in sh
  - Run using interpreter
- To execute ‘a.sh’, either run as ‘bash a.sh’, or ‘./a.sh’
  - In the latter case, the file should have permission ‘executable’ (chmod +x a.sh)

Checking a shell script

- shellcheck: brew/apt install spellcheck
  - Online version: https://www.shellcheck.net/

```
In ./mybuild.sh line 9:
  -DZ3_INCLUDE_DIR=$Z3DIR/include \
                   ^----^ SC2086: Double quote to prevent globbing and word splitting.
```

- VSCode: Bash IDE, shellcheck

She-bang (#!)

- “#!/bin/bash” at the first line of a script
- “#!”: format indicator
  - Unix reads first a few bytes to identify the format of the file
  - Indicates that this is a script (not ELF executable file)
- The remaining characters indicate the interpreter to use
  - This can be /usr/bin/python , etc
- Note that # is also used as a character to begin comment

Variables

- Defining and using a variable:

```
# Define a variable X
X=“hi”
# Use the variable with prefix $
echo $X
```

- Variables can be defined from the outer space: “environment variable”
  - A user can define X from outside using `export X=hi`
- It cannot be exported to the outer space though
  - To allow it to export values, run `source a.sh`

String

- No quote, Double-quote(“”), single-quote(‘’)
  - No quote: a string without any space (e.g. apple)
  - Double-quote: can use a variable inside a string (e.g. “hello world”, “hello $name”)
  - Single-quote: cannot use a variable (e.g. ‘hello $name’: name is not expanded)
- Backticks(``): execute the command and get the result from stdout!
  - FILELIST=`ls -1 mydir`; echo $FILEIST
  - Parenthesis works as well! FILELIST=$(ls -1 mydir)

Checking whether a compiler does not timeout

```
cp bitcount1.ll input.ll
timeout 60 sf-compiler input.ll -o a.s
# Check whether $? is 124
```

- A shell script contains a list of commands
- Execution of a command finishes with a return code: the integer from its `int main()`
- If it is zero, the command finished successfully
- Otherwise, 'something' happened:
  - `timeout`: returns 124 if timeout, otherwise the exit code of sf-compiler
- It will fall-through (does not exit immediately)
  - to immediately exit the script, use 'set -e'

Checking whether a compiler does not timeout

- The space between tokens is important (marked those as underbars)
- -eq, -ne: integer compare
- ==, !=: string compare
  - Can use regular expression as well
- Prefer [[..]] over [..] *
- More operators: https://www.tldp.org/LDP/abs/html/comparison-ops.html

```
timeout 60 sf-compiler a.ll …
CODE=$?

if_[[_$CODE_-eq_124_]]; then
  echo “Timeout”
elif [[ $CODE -ne 0 ]]; then
  echo “Failure”
else
  echo good
fi
```

* https://stackoverflow.com/questions/669452/is-double-square-brackets-preferable-over-single-square-brackets-in-ba

For loop

```
for i in apple banana pear “black bean”; do
  echo $i # prints fruits per a line
done
```

- For loop iterates over items that are separated by spaces
- To iterate over a range, you can use `for i in {1..10}` that is a bash extension
  - Note that this does not allow $N to be used as lower/upper bound *
- Combined with `ls -1` or `find . -name “*.txt”`, you can iterate files inside the dir
  - Should not have spaces in there names: use `find . -name “..” -exec`

* https://stackoverflow.com/questions/169511/how-do-i-iterate-over-a-range-of-numbers-defined-by-variables-in-bash