Shell Test Exercise

IT 115 - Intro to Software Development & Version Control

Kit Transue

Activity

Implement in a Unix shell: a testing framework and test cases for regular expressions.

Goals:

- experience with Unix bash scripting
- practice with regular expressions and egrep
- think about testing frameworks
- communicate results with exit codes
- practice making non-happy-path tests

Requirements

Work in a bash shell. This can be:

- in Linux
  - in the default GitHub Codespace container
  - in a local Docker container (Alpine is very small and the “bash” below is actually simple sh.)
  - in WSL
- on macOS (zsh is OK)
- on Windows, in the git-bash shell provided with Git

Produce:

- for each provided task:
  - a regular expression to match the provided sample
  - one additional example you think the pattern should match
  - three samples you do *not* want the pattern to match (these should be non-trivial)
- a shell script to test your expression against your samples
  - should return zero if pattern works as expected
  - should return non-zero if there is a problem with pattern or testcase

Tasks:

Implement egrep detectors for the following three patterns:

- lines beginning with “+” or “-”

- any lines from “git log –decorate=short” that have a tag or ref associated with them (lines 1 and 19 of included git-log.txt)
- lines representing new ssh sessions in auth.log (use “egrep -i session auth.log” to find a starting point)

Scripting

$? is a shell variable that is set after every command returns. Examine what the return value is for some commands:

- echo hello > welcome.txt; echo $?
- echo ‘see ya!’ > bye.txt; echo $?
- egrep hello welcome.txt; echo $?
- egrep -v hello welcome.txt; echo $?
- egrep hello bye.txt; echo $?
- egrep -v hello bye.txt; echo $?
- cat not-an-existing-file.txt; echo $?
- cat welcome.txt bye.txt > both.txt; echo $?
- egrep -v hello both.txt; echo $?

Make a testsuite

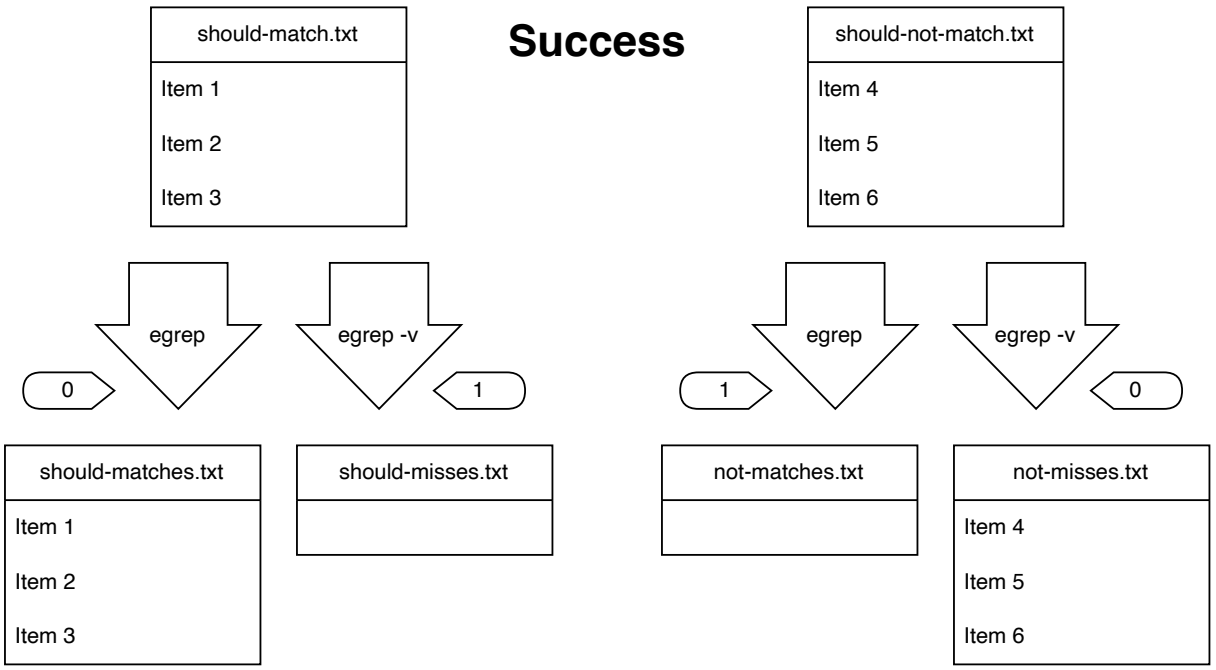
Turn your regular expressions and postive/negative testcases into a testsuite.

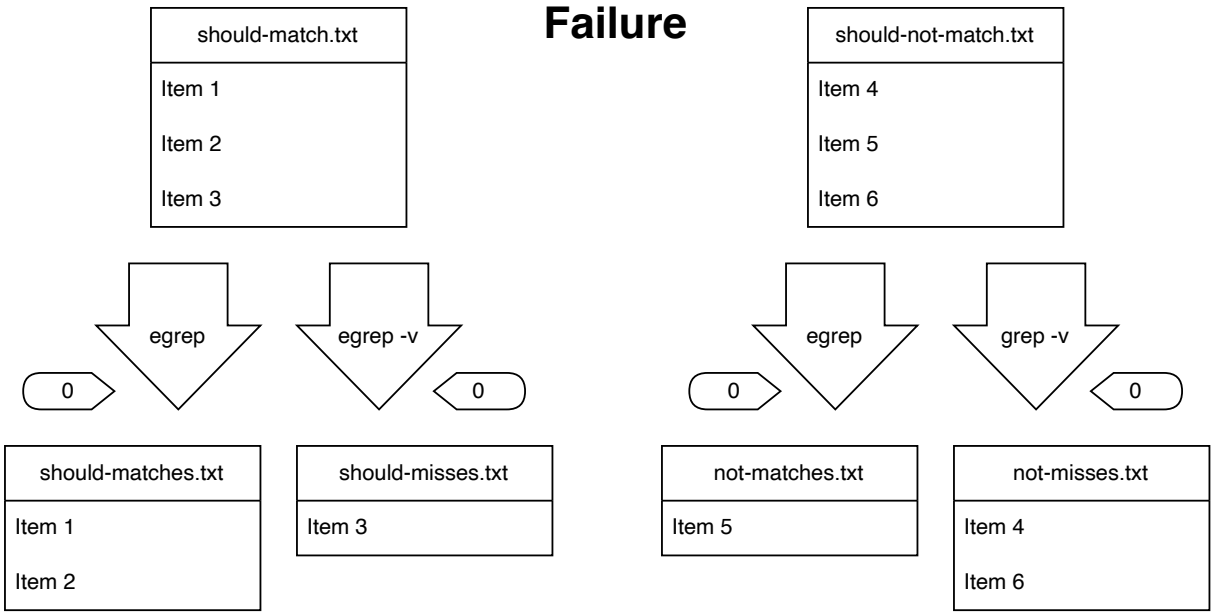
You testsuite should:

- exercise each of your regular expressions
- make sure all positive tests succeed
- make sure no negative tests match
- return 0 or success
- return non-zero on failure

Logic

This chart shows the egrep exit codes for egrep and “egrep -v” run against sample data.





What exit codes differ between good and bad output? Use those to build your test framework.

Writing in Shell

Create a text file with a ".sh" extension.

Write your code in the file.

Run the code by:

“bash filename.sh”

optional: make executable

- add “#!/usr/bin/env bash” as the first line of the file
- chmod a+x the filename
- to run: “./filename.sh”

Shell Conditionals

$? can be used in the shell’s “if” construct. “0” means “true” and execution moves to the “then” clause; all other values are interpreted as “false” and execution moves to the optional “else” clause:

```
failed=0  # variable to track our planned exit status

# run "cmd" (can pass arguments), and check exit code
if cmd
then  # cmd status was 0
    echo "cmd success"
else  # cmd status was non-zero
    echo "cmd failed"
    failed=1
fi

exit $failed
```

Shell loops

To use a for loop:

```
for x in *.txt
do
    cmd $x
done
```

Other features

Shell is a pretty full-featured programming environment; ping me if you have other things you want; I can give them to you.

The shell is sensitive to line endings; if you want to write all on one line (or mix split and single lines), use a semicolon:

```
for x in $.txt; do cmd $x; done
```