Bash: Individual Assignment

Bas Terwijn

1 Introduction

In the individual assignment we will write Bash scripts to change and analyze the Java code in git repository:

- `https://bitbucket.org/bterwijn/knapsack`

For this assignment you will be mostly working in directory "./individual/".

2 Find and Replace

Write script **"find_name.sh"** that prints the file name and line number of each occurrence of a name in all Java source files in the repository. The script should be usable as

```
./find_name.sh <name>
```

where "<name>" is any name without white spaces. Also write script **"rename_name.sh"** to find an old name in all Java source files in this repository and replace it with a new name.

```
./rename_name.sh <name-old> <name-new>
```

Also write script **"rename_class.sh"** to rename a Java class in the repository. If you rename a class in Java you will also have to rename the file the class is in, therefore rename the file as well.

```
./rename_class.sh <class-old> <class-new>
```

3 Keyboard shortcuts

To work a lot faster in the Bash shell and to make it more enjoyable use keyboard shortcuts. Take some time now to practice using them and see which you like:

shortcut key	***effect***
Tab	auto-completes filename, press again when stuck for options
middle-mouse-button	copy selected text
Ctrl-p or Cursor Up	previous command in command history
Ctrl-n or Cursor Down	next command in command history
Ctrl-r <type>	search in command history
Alt-.	last word of previous command in command history
Ctrl-a	move cursor to beginning of line
Ctrl-e	move cursor to end of line
Ctrl-f	move cursor forward one character
Alt-f	move cursor forward one word
Ctrl-b	move cursor backwards one character
Alt-b	move cursor backwards one word
Ctrl-xx	jump between current position and beginning of line
Ctrl-d	delete one character
Alt-d	delete one word
Ctrl-h	backspace one character
Ctrl-w	backspace one word
Ctrl-k	delete all to the right of cursor
Ctrl-u	backspace all to the left of cursor
Ctrl-/	undo last edit
Ctrl-l	clear screen

Most shortcuts are the same as for the "emacs" text editor. For the languages in the following weeks of this course emacs can help you with proper syntax highlighting and indentation (based on file extension) which makes writing code more pleasant. I strongly recommend using an editor that understands the particular language you write in (not necessarily emacs but it is an option).

4 Class diagram

In file "example_class_diagram.dot" you find an example class diagram:

```
digraph D {

  A -> B                    # for inheritance
  C -> B [arrowhead=diamond] # for composition
  B -> D [arrowhead=dot]     # for import

  subgraph cluster_X { label = "package X"
    A [shape=box]
    B [shape=box]
  }

  subgraph cluster_Y { label = "package Y"
    C [shape=box]
    D [shape=box]
  }

}
```

Install program "graphviz" and generate a pdf file based on this file with:

```
dot -Tpdf example\_class\_diagram.dot > \
          example\_class\_diagram.pdf
```

The class diagram in the pdf shows different relations between class A, B, C and D.

classes	***relation***
A	extends B
B	the file class B is in imports class D
C	has one or more objects of type D as member variables (composition)
A,B	are in the same package
C,D	are in the same package

Now write the following scripts to generate similar diagrams for the classes in the Java source files in this repository:

script	***diagram that shows***
generate_diagram_classes.sh	just all classes
generate_diagram_extends.sh	all extends relations
generate_diagram_import.sh	all import relations
generate_diagram_composition.sh	all composition relations
generate_diagram_package.sh	all package relations
generate_diagram_all.sh	all relations above

Script "generate_diagrams.sh" should then afterwards generate all ".dot" files for each diagrams. Use regular expressions to parse the Java sources. Script "generate_diagram_classes.sh" is already given as an example. For other useful regular expressions examples see "example_perl_regex.sh".

Parsing of source code with a complex syntax like the Java language is in general very difficult. Your scripts only have to work for the sources in the Knapsack repository that for example have no nested classes. Classes not defined in this repository don't have to show up in the diagrams but you are allowed to do so.

5 Makefile

Add to the **"Makefile"** in the root directory so that with:

- **"make diagrams_dot"** each dot file for each diagram gets generated
- **"make diagrams"** first all dot files and then all pdfs for each present dot file gets generated, use a makefile pattern
- **"make clean"** also all the files generated by your scripts (dot,pdf, and possibly other temporary files) in "./individual" get removed (be very careful not to remove your own scripts by accident!)

6 PATH

To run a particular script from inside an arbitrary directory, and without a path prefix (for example "./") you need to:

- Add the global path of the script to your "PATH" environment variable.
- Set the executable bit of each script using "chmod +x <script>.
- Add the "#!/bin/bash" shebang line as the first line of the scripts to define how the script is to be executed.

Experiment with this by creating directory "$HOME/bin" and creating a script in that directory. Then add line "export PATH=$PATH:$HOME/bin" to your shell start-up file (file "$HOME/.bashrc" for Ubuntu) so that "PATH" will get set every time you start a new shell. Start a new shell/terminal and check with "echo $PATH" and try to execute your script. After you've gotten it to works "$HOME/bin" is your new place to collect all useful scripts you come across.

7 Submission

Submit your solutions on Canvas after cleaning up the repository (make clean) and zipping it as a whole to knapsack.zip. Double check that all your solution files are included.