COMP210 - Lab 5

Spring 2014

For your first project you'll be implementing some basic Shell functionality along with a basic file system class hierarchy.

Problem

We want a basic *File System Shell* object that provides some standard linux-like functionality for a simple file system. The shell object maintains two variables: *pwd* and *root*. The later keeps a consistent reference to the root of the file system while the former keeps a reference to the current working directory. The shell should provide the following functionality:

- *pwd*: print working directory. This should print the *absolute path* of *pwd*.
- *cd*: changing the current working directory. Like in linux, we can use .. as a shortcut to parent, . as a shortcut to here, and we can access other directories via absolute or relative paths.
- *du*: disk usage reporting. Using this functionality should report[1] the total size of the file system, the size of the system from *pwd* down, and the percentage of the complete system accounted for by the *pwd* subtree.
- *ls*: list directory contents. Like in linux, calling this with no arguments should list content of *pwd* where calling it with an absolute or relative path to a directory in the system should print the contents of that directory.
- *touch*. Used on an exiting file, sets the date modified to the current date and time. If the file its called on doesn't exist, then it creates that file.
- *rm*: remove. Deletes a file[2].
- *rmr*: remove recursive. Delete a directory and all its contents.

[1] print

[2] file only

Any command that works with a file or directory should work with either absolute or relative paths. Your file system shell can and should be *mutable*, but try to work with a purely functional file system.[3].

[3] notice how this allows lots of "easy" unit testing...

Project Homework

Due Friday 2/21

Your pre-project homework is simple. Come up with a nice, robust file system from which you can draw test cases. You can document your file system any way you'd like so long as it is clear and specific. **Turn in a copy of your example file system at the start of class.**

Project

Due Monday 3/3

Complete all the functionality listed above. Code should be well documented, exhibit good coding style, and include a complete set of jUnit tests. Non-functional methods[4] should also include a jUnit test which simply calls the method one or more times such that you can observe the result. We'll discuss design issues a lot in class, so your code should exhibit good design as well[5] **Submit your project as *proj1* using *handin*.**

[4] things that print

[5] or at least consistent with how we're doing things in class