Programming Project Progress Report

Sean Canavan Nick LeClair Steven! Lorenzen

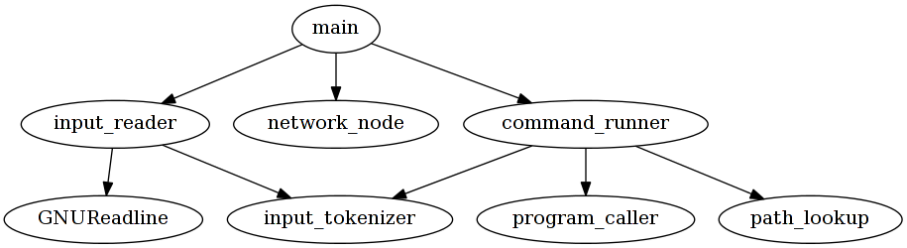
April 5, 2012

1 Modules in sh142

Our implementation of sh142 will include multiple modules broken up by task responsibility.

Our main modules deal with input reading, program execution, and networking. Each of those main modules has submodules for more finely grained implementation details.

The figure below details our dependency graph.



2 Main Module APIs

2.1 input_reader

- `char* readline(char*)` - A thin wrapper around GNU Readline.

2.2 input_tokenizer

- `char*[] tokenize(char*)` - Break the input up into individual tokens separated by whitespace. Quoted bodies are kept together with quoting stripped.

2.3 command_runner

- `void invoke_command(char*[])` - Invoke the tokenized command as a sh142 command. This is the stage where input will be parsed before being delegated elsewhere.. for-loop expansion and PATH and DATA lookup will also be substituted here.

2.4 path_lookup

- `char* resolve_data(char*)` - Test for presence of filename in the specified DATA path. If, present return the absolute path to the file otherwise NULL.
- `char* resolve_executable(char*)` - Test for presence of filename in the specified PATH variable. If present return the absolute path to the file otherwise NULL.

2.5 program_caller

- `int runcmd(char* cmd, char* argv[])` - Run the command with the given arguments return the exit code of the command.
- `int bgruncmd(char* cmd, char* argv[])` - Run the command with the given arguments in the background, retyrn the pid of the created job.

2.6 network_node

TBD

3 Work Plan

The work will primarily be broken up by module as given below:

- input_reader - COMPLETED
- complete spec tests - All by April 7
- program_caller - Steven! by April 14
- path_lookup - Sean by April 14
- input_tokenizer - Nick by April 14
- command_runner - Sean and Steven! by April 21
- main sh142 - Nick and Steven! by April 21
- network_node - Nick and Sean by April 28
- report - All by May 8

4 Test Plan

Testing will be carried out at every stage of development. Prior to designing the API, a test harness for complete automated interaction testing was set up using the Cucumber tool. As we build the API we will be using the ctest unit-test framework to perform isolated tests on individual C functions.

A sample of cucumber output:

```
Feature: PATH and DATA manipulation

  Scenario: Altering the PATH of the running shell.  # features/path_manipulation.feature:2
    Given a directory named "mybin"                  # aruba-0.4.11/lib/aruba/cucumber.rb:11
    And a file named "mybin/myprg" with:             # aruba-0.4.11/lib/aruba/cucumber.rb:15
      """
      echo my program
      """
    When I run `sh142` interactively                 # aruba-0.4.11/lib/aruba/cucumber.rb:74
    And I type "PATH=./mybin:$PATH"                  # aruba-0.4.11/lib/aruba/cucumber.rb:78
      Broken pipe (Errno::EPIPE)
      features/path_manipulation.feature:9:in `And I type "PATH=./mybin:$PATH"'
    And I type "myprg"                               # aruba-0.4.11/lib/aruba/cucumber.rb:78
    Then the stdout should contain "my program"      # aruba-0.4.11/lib/aruba/cucumber.rb:159
```