Configuring the Submitter Program – Professor's Instructions

for the

Core Programming Courses

School of Information and Communications Technology

written by

Fardad Soleimanloo

September 11 2016

Preface

The submitter application helps students check and submit their assignment solutions in a unified way in a Linux environment by automating compilation, program output verification and submission, and thereby removing some of the marking tasks from the professor's responsibility.

This application replaces the traditional assignment submission process in console programming courses at Seneca. The traditional submission process can be summarized as follows.

First the assignment description was published for the students, possibly with some initial code to help them start their work.

The students would follow the instructions in the assignment and complete its tasks. They would do this either on Matrix (the Linux platform in Seneca) or locally on a computer and then upload their solution to Matrix. After this, they were asked to somehow capture the output of their program by redirecting the output to a text file or by recording the execution process using a typescript command.

The student would upload or attach the source code or the typescript to an email for their professor to mark. There was no guarantee that the source code attached was actually the code that generated the output, that the compilation process was genuinely successful or that the typescript was not modified manually to match the expected output.

Under this system, it was the student’s responsibility to ensure that the output is EXACTLY as expected with no immediate feedback. The professor would verify this at marking time and any errors would cost the student a significant portion of the total possible mark. The student would learn their mistake, days and sometimes weeks, after submitting the assignment and receiving their mark.

The submitter application described in this note notifies the students of any errors in the compilation or execution phases enabling them to correct their solutions before submitting them to the professor. The application also ensures that the output submitted is that produced by the source code submitted.

Summary

The submitter program is a C++ program written to execute Linux commands based on a configuration file created separately for each assignment. The template for this file is created in the proper directory by the installation script.

You install the submitter program once in your home directory on the Linux platform that students will use to submit their assignments. You then create a configuration file for each and every assignment. This file tells the Submitter program how to process each student's solution before emailing it to your account.

Each student submits their solution by running the submitter program located in your home directory.

> **For example if Professor John Doe wants his students to use the Submitter application for Workshop 2 and the configuration file he created for this is ws2.cfg, his students should issue this command to submit their Workshop 2 on matrix from their working directory:**
>
> **`$ ~john.doe/submit ws2 <ENTER>`**

Installation Instructions

First the professor needs to install the program; this is how it is done:

Login to your account on Matrix (if your account is not activated, email helpdesk@senecacollege.ca requesting to activate your account).

In your home directory, enter the following command:

```
$ git clone https://github.com/fardad/Submitter.git <ENTER>
```

This will create a directory called "**`Submitter`**" and download the installation script and all of the source files.

Change the current directory to the "**`Submitter`**" directory:

```
$ cd Submitter <ENTER>
```

Give execute permission to the installation script:

```
$ chmod 700 install <ENTER>
```

Run the installation script:

```
$ install <ENTER>
```

This will prompt you with the warning that access to your root directory will be modified to `755`. (Make sure that the sub-directories of your home directory have the proper permissions so that they are not accessible if you don't wish them to be.) The install script will then compile the source code, create a sub-directory called **`submitter_files`** in your home directory, and copy the result (the submit command) to your root with proper permissions set. This sub-directory will include the submission configuration template named **`template.cfg`**. For each assignment, you will need to modify a separate copy of this template and save it in the **`submitter_files`** sub-directory.

> *The **`submitter_files`** sub-directory is the place where the assignment configuration files are saved as well as the unit tests along with the expected output of each assignment, if needed. This is explained in more detail below.*

Once the installation script finishes executing, the Submitter application is ready to configure for a particular assignment.

Configuration Instructions

Let us assume that we are preparing Workshop 5 and that we want our students to use the following command to compile their code:

```
$ g++ -Wall -std=c++0x -o w5exe w5.cpp w5main.cpp <ENTER>
```

> *Since we are on a Linux system I rather call the executable "w5exe" and not "w5.exe".*

The values shown in red below are values that we define in our configuration for this workshop.

Our first decision is the name of the configuration file. Let us say that we want the student to enter the following when they submit their assignment:

```
$ ~john.doe/submit ws5 <ENTER>
```

In this case, the name of configuration file in "**`submitter_files`**" directory should be: **`ws5.cfg`**.

Then, we make a copy of **`template.cfg`** and save it in **`submitter_files`**

```
$ cp submitter_files/template.cfg submitter_files/ws5.cfg <ENTER>
```

We open **`ws5.cfg`** and provide the following information:

1- Administrative information:

   a. Course code: Let's say `OOP244`.

   b. Professors user id: **`john.doe`**

   c. Email to send the submissions to: **`john.doe@senecacollege.ca`**

   d. Email of TA for feedback: **`teacher.assist@senecacollege.ca`** (If applicable)

   e. Title of the workshop: "**`Workshop 5`**"

2- The files that must be present before the student can start the submission process: **`w5.cpp`**, **`w5.h`**, **`data.txt`** and **`reflect.txt`**

3- Does this assignment need to be compiled: **`yes`**

   a. The compile command without the source file names:
      `g++ -Wall -std=c++0x -o w5exe`

b. The name of the executable that compile command generates read: `w5exe`

c. The name of the file for unit testing if the assignment is going to be tested using a unit test or a main program that is handed out to the students. If this is the case the file for the unit test is saved in **`submitter_files`** to prevent a student from tempering with it: **`w5main.cpp`**.

d. The files that participate in the compilation command line: **`w5.cpp, w5main.cpp`**

e. Are warnings allowed in the compilation: **`yes`** or **`no`**

f. The name of the file in which the compiler messages are captured: **`errw5.txt`**

4- Is there a program to be executed for testing:

a. Should a program be executed for testing: **`yes`** or **`no`**

b. If so, you need to do the assignment yourself, test it, run it, capture the output and save it in a file of expected output. Let's say the name of this file is **`w5output.txt`**.

c. The method of capturing the output – `redirect`ing the output to a text file or creating a type`script`. Let us choose type`script` in this example: **`script`**

d. The name of the file that captures the execution result of the student's submission: **`output.txt`**

e. The comparison range (from line , to line , inclusive) , let's say (`5, 34`). This means that only the output lines will compared from line 5 to line 34, inclusive.

f. Should the student's output be compared to the expected output: **`yes`** or **`no`**

g. The name of the file with the expected output: **`w5output.txt`**

5- The names of the files that the student needs to submit. Let's say that these files are: **`w5.cpp`**, **`w5.h`**, **`data.txt, errw5.txt, reflect.txt`**.

Copying the values listed above into configuration file **`ws5.cfg`** produces:

```
-- name of this file must be the same as the name you would like the students
-- to use at submission command;
--    if you like the submission command to be:
--    $ submit workshop2
--    then for this submission you should copy this config file to workshop2.cfg
--    and modify its values base on your needs.


-- format for configuration values:
-- value_name|value1, value2, value3,...

--Subject Code
  subject_code|OOP244
```

```
--Professor Linux userid
  userid|john.doe

--Professor email
--you can add more emails to send a copy of submission to TAs
  prof_email|john.doe@senecacollege.ca, teacher.assistant@senecacollege.ca

--Assignment name
  assessment_name|Workshop 5


--files that must exist for a successful submission (or compile)
  assess_files | w5.cpp,w5.h,data.txt,reflect.txt

--File names to be copied to student account from professor's
--submitter files direcotry for testing (tester programs and etc)
--comment out the line if you don't need copying
  copy_files|w5main.cpp


--compile code (yes, no)
  compile|yes

--compiler command
--if you want the program to be executed then this compile command should
--an exactutable with the same name as exe_name value (see next value)
  compile_command|g++ -Wall -std=c++0x -o w5exe


--executable name
  exe_name|w5exe

--files to compile
--this will be ignored if compile value is no
  compile_files|w5.cpp,w5main.cpp

--error capture file name
--this file will hold the warnings and error resulted by compiling the code
  err_file|errw5.txt

--Allow warnings in compilation (yes, no)
  allow_warning|no

--execute (yes, no)
  exectue|yes

--type of output (script, redirect):
--this will be ignored if check_output is no or execute is no)
--   "script", for interactive programs, typescript will be invoked
--   "redirect" or any other value, dump ouput to file for non-interactive programs

   output_type|script



--output file name
--the output of the execution or the typescript will be dumped in this
  output_file|output.txt

--compare the output to professor's; (yes, no)
--this will be ignored if execute value is no
--anything but "yes" will be considered as no value
--in this case the file for the outpu comparison should be copied to submitter_files dir.
  check_output|yes

--comparison range, line numbers
```

```
-- values: from, to ; integer numbers
--to specify the range of comparison (in this example from line 16 to line 31, inclusive)
  comp_range|5, 34

--filename for the original output in submitter_files dir.
--this will be ignored if check_output is no
  correct_output|w5output.txt

--files to be emailed as submission. (if any of them is missing assingment can not be sumbitted)
  submit_files |w5.cpp, w5.h, data.txt, reflect.txt, errw5.txt
```

For the configuration to be complete, the “submitter_files” sub-directory must contain the three files named above: ws5.cfg, w5output.txt and w5main.cpp

Student Submission Instructions

To submit their solution, a student enters the following command:

```
$ ~john.doe/submit ws5 <ENTER>
```

The submitter program will

- check and make sure that all the files needed are present
- compile the code with the provided tester main in professor’s account
- prompt the students to get ready for demonstrating execution of the assignment
- run the executable
- if done, compare the output generated with the expected output in professor’s account
- If everything goes ok, attach all the files needed and email them to the professor and the TA.

If at any point, anything goes wrong, the submitter program will display a clear message to the student that identifies their mistake and prompt them to retry.