**Autotest Creation – CS135 Fall 2011**

By: Yen-Ting(Mark) Chen

ID: y274chen

Term: Fall 2011

[1. Course Account Test Related Structure](#T1) 2

 [1.1 Handin Folder](#T11)

 [1.2 Marking Folder](#T12)

 [1.2.1 In Folder](#T121)

 [1.2.2 Provided Folder](#T122)

[2. test.pt/0 Set Up](#T2) 4

 [2.1 General Configuration](#T21)

 [2.2 In Folder](#T22)

 [2.2.1 Stepper](#T221)

 [2.2.2 General](#T222)

 [2.3 Provided Folder](#T23)

 [2.3.1 Disallow](#T231)

 [2.3.2 Error Checking](#T232)

 [2.3.3 Structural Definition](#T233)

[3. RST](#T3) 9

 [3.1 rst](#T31)

 [3.2 distrst](#T32)

[4. Running Autotest](#T4) 10

 [I. Preliminary Steps](#T41)

 [II. Checkout – Test - Commit](#T42)

 [III. Markus](#T43)

 [IV. Notice Related to Marking](#T44)

**0. Introduction**

This guide is created in Fall 2011. It may be not applicable in the future term. Sorry if grammatical error or unclear explanation exists and may be too orally worded.

This guide discuss about autotest creation in detail and application specific. For more general information (like theoretically how the command work), go to search in twiki. The Goal of This Guide is **to guide through creating autotest in detail.**

Although hyperlinks exist, for the first time user, read it section by section is recommended.

Some notations used:

${Anum}=01

${studentid}=${sid}=y274chen

**1. Course Account Test Related Structure**

In crouse account, students assignment are in

 ~/handin/a${Anum}

 ~/handin/a01

and test cases are in

 ~/marking/a${Anum}

 ~/marking/a01

When using rst (the command to autotest), these 2 folders are automatically corresponded according to a${Anum}(a01)

[top](#_top)

**1.1 Handin Folder**

In handin/a${Anum}, each students will have a folder ${studentid} contain his/her submission. The folder is created automatically when a student submits files via Markus.

So, assume a01 has 2 files a1q1.rkt and a1q2.rkt. If a student y274chen submits the assignment on Markus,

in handin, it will have:

~/handin/a01/y274chen/a1q1.rkt

~/handin/a01/y274chen/a1q2.rkt

Because some student may have submitted incorrect named files or random files, the autotest has a filter to filter out the incorrect submission when testing.

So in handin/a01, **we will** **[create a file ".subfiles",](#subfile2)** contains the following:

 a1q1.{rkt,ss,scm}

 a1q2.{rkt,ss,scm}

since Drracket can also have ss and scm, so we have {rkt,ss,scm} to accept.(I think it can just be .rkt)

[top](#_top)

**1.2 Marking Folder**

In the making folder, we have a${Anum} folders. In each folder, contains the test cases/configuration for the test cases.

In each assignment we have 2 folders:

 **test.pt** folder is for public testing, must have the same name.

 **test.0** folder is for auto testing, the .0 is arbitrary, but stick to 0 for consistency with other terms.

The configurations are usually all the **same** for every assignment(both pt/0). Copying the existed configuration is recommended.

So the only thing we need to care is the following 2 folders, in and provided in pt/0.

 [top](#_top)

**1.2.1 In Folder**

Assume for an autotest for a01,(continue from above)

we have 3 cases for q1. 4 cases for q2. Then, the file system in in folder will be :

“

in/1 ;for test set1

 /case1 ;folder name can be more meaningful

 /test.ss ;1 test case only

 /case2

 /test.ss

 /case3

 /test.ss

 /options.ss ;options for set1

in/2 ;for test set2

 /case1

 /test.ss

 /case2

 /test.ss

 /case3

 /test.ss

 /case4

 /options.ss

in/options.ss ;total options apply to assignment (usually language type and timeout)

“

When rst is being run, the cases in the in folder(with options applied) will be used.

**1.2.2 Provided Folder**

Sometimes, some functions and/or some constants are provided to students or to be used by autotesting.

We can use provided folder to contains those functions and constants.

There are some restrictions and may be frustrating when getting the designated goal to sort out.

[top](#_top)

**2. test.pt****/0 Set Up**

There are some configurations about the test.pt/0 folders. Usually the configuration does not need to be changed for every assignment. And mostly test cases are the only change you need to make every time.

[top](#_top)

**2.1 General Configuration**

The configuration options can be found in twiki, but this setting was used in Fall 2011.

**/answer**

 - leave it empty, not sure what it does, but just create it and leave it there.

**/computeMarks**

 -leave it as it is, it's a bash script

"

#!/bin/sh

exec /u/isg/bittersuite3/computeMarks

"

**/config.ss**

 -same as computeMarks

"

(print-by-question true)

(nroff-mark-scheme true)

"

**/mark-submit**

 -empty, leave it there

**/runTests**

 - leave it as it is, it's a bash script

"

#!/bin/bash

/u/isg/bittersuite3/runTests

"

**/in/0**

 -this folder is dealing with: 1. Incorrect submission filter, 2. Filter the submission without a00 done. Just copy paste from the previous term/assignment.

 -**this folder** [**will need ".subfiles"**](#subfile1) created in handin folder.

**/provided/a0\_require**

 -For filtering a00 completion. Using

~/course/publictest/${term}/a00completedrun.log

(${year}1${term-in-month} : year = 2011, term=Fall, term-in-month=9(September) =>1119)

[top](#_top)

**2.2 In Folder**

So, this is where the test cases are contained.

In in/options.ss, you probably need to put:

"

(language lang/option)

(timeout time(s) )

"

**language**: options are - scheme/beginner scheme/intermediate ... etc look up in twiki

**timeout**: time in second (prevent some students throw in infinite looping program.

for options in each set question

in/${Qnum}/options.ss

"

(loadcode "studentfile.rkt")

(modules "providedfile1.rkt" "providedfile2.rkt" .... "providedfilen.rkt")

"

**loadcode**: load students file in handin to test

**modules**: require the file in provided folder, so test and student file can use the (provide ...) func/const in provided file

modules is not necessary to be declared here. You can use modules in each test cases, and it will override the modules in options.ss

Example in [2.3 Provided Folder](#T23) section.

I guess, any **re-declaration will clear the previous declaration**.

[top](#_top)

**2.2.1 Stepper**

Some assignments use stepper questions and it is different from general test cases

For simplest solution, copy from the previous term/assignment!

So once you copy it, modify the file "test.exe" in in/1/1/test.exe

The test loads ~/course/stepping/${studentid}, so you can have a reference to see how the test works.

You need to **change 3 places:**

1.

"

# The number on the next line is the output of:

# date +%s -d 'Tue Oct 11 21:00:00 EDT 2011'

my $duetime = 1318381200;

"

Execute the command with corresponded assignment due time, and copy the result number (remind EDT, PST exchange)

2.

"

 next unless $F[5] =~ /(value|err) c/;

 next unless $F[3] =~ /^assignment4q1([a-c])$/;

"

Change 4q to ${Anum}q (eg, 5q for assignment 5) and ([a-c]) [indicate the number of questions] to (eg, 6 questions) ([a-f])

3.

"

print THREE (($#plist + 1) \* 100), "/3\n";

"

Change 3 to whatever the number of questions is.

Use the same test for both public and autotest.

[top](#_top)

**2.2.2 General**

In test.ss for each test cases:

 “

(desc "description send to student, this will be seen in public/autotest")

(value case-value) ;how many marks the question worth, usually 1 which is default and don't need to set it

(modules ...) ; as above, it will override the modules in options.ss

(result expr1) ;rst will match expr1=expr2 otherwise the test case fail ,expr as scheme expression

(expected expr2) ; this is the main part of the test case, you can treat it as check-expect

“

There are more options in twiki, but I found these are sufficient enough to create all test cases.

There are some details to notice:

**1. Inexact number**

Bittersuite is not good at inexact number produce by scheme.

So (< (abs (- expr1 expr2)) tolerance-range) true

is usually used. (like check-within)

**2. Error checking**

Sometimes the question asks for producing certain error message.

So, check next section ([2.3.2 Error Checking](#T232)), if we have a function called "errorcheck"

(result (errorcheck expr1))

(expected "error string")

**3. Disallow functions**

Sometimes the question disallowed some functions use (like reverse).

As above ([2.3.1 Disallow](#T231)), put the disallow file in provided and modules the file

Then when test is running, the test fails if disallowed function is used

[top](#_top)

**2.3 Provided Folder**

The provided file is same as provide/require in scheme where your provided file explicitly provide functions and constants, and student program uses the provided file via modules in the suite (they can also explicitly use require)

For File1 to use function “f1” defined in File2:

1.

Regular scheme

File1 File2

(require “File2”) <- (provide f1)

2.

Testing with explicitly require

File1 provided/File2

(require “File2”) (provide f1)

 <-

(modules “File2”)

;This case, the testing case must have modules

3.

Testing without require

File1 provided/File2

 (provide f1)

 <-

 (modules “File2”)

Self-defined(not the one already in twiki) provided file need to have

#lang racket

in the provided file

You can use provided file for

1. constants definition (such as tree that is used a lot)

2. functions definition

 - functions use to compare with the result of student function

 - functions use to generate value for test cases

 - functions that have special purpose (disallow/error)

[top](#_top)

**2.3.1 Disallow**

To disallow function use, use the following code, and put to provided folder.

Then modules the file in the designated test cases/options.ss

"

#lang scheme

(provide

**(rename-out [d-append append])**

(rename-out [d-reverse reverse]))

(define-syntax disallow

 (syntax-rules ()

 ((~ func)

 (lambda args

 (error (format "Disallowed function ~a called" (quote func)))))))

**(define d-append (disallow append))**

(define d-reverse (disallow reverse))

"

In this case append and reverse are disallow. You can add as many disallow as you want, as long as, it comes in pair, above and below.

[top](#_top)

**2.3.2 Error Checking**

Having the following code in provided folder and modules in the test case.

"

#lang scheme/base

(provide errorcheck)

(define-syntax errorcheck

 (syntax-rules ()

 ((\_ expr)

 (with-handlers ((exn:fail? (lambda (exn) (exn-message exn))))

 expr

 "No exception raised"))))

"

So, if the result of the student function is (error "Invalid")

Your test case will be:

(result (errorcheck (student-function ...)))

(expected "Invalid")

[top](#_top)

**2.3.3 Structural Definition**

There are 3 cases for the structual definition to be handled

1. **Struct-define** **in student program.**

- provided file cannot use the struct-define created functions

 -> cannot have constant/function definition related to the structure in provided file

 - it says function not defined, when the provided file is compiled before start testing.

2. **struct-define in provided file**

- assignment needs to provide the file with struct-define in it

 -> students need to explicitly require the file in the assignment (may lead to chaos since students are ...)

3. **Pass in the struct-define functions into provide file**

For example

student file:

(struct-define pos (x y))

provided file:

(define (calc-length make-f f-x f-y f? pos)

(cond

 [(f? pos) (sqrt (+ (sqr(f-x f)) (sqr(f-y f))))]

 [else false]))

Test case:

(result (calc-length make-pos pos-x pos-y pos? (make-pos 3 4))) ; pass in the function so calc-length in provided file has the functions to use

(expected 5)

Unfortunetly, it is only usable after the function can be passed as variable.(after intermediate or intermediate-lambda)

Discuss with the professor to see which solution is to be used.

[top](#_top)

**3. RST**

After,

**1. a${Anum} folder and .subfile is prepared**

**2. test.0/pt folder is configured**

**3. test cases are ready**

You can run testing.

Public test is automatically run when a student makes a submission and send the result to the student. But you still can run manually to test the test cases.

There are 2 way to run testing: rst and distrst

rst is for running **few or single submission** -> for testing the test cases correctness or remark

distrst is for running **all submission** after due date

Test the detail out yourself

[top](#_top)

**3.1 rst**

rst -s ${sid} a${Anum} pt/0 ${runid}

eg:

rst -s y274chen a01 pt test1

-s : a single student

${sid}: student id

a${Anum}: assignment number, can be arbitrary

pt/0: public/auto test suit, can be arbitrary

${runid}: an indication of the result

So the rst will take students file at

 ~/handin/a${Anum}/${sid}

 ~/handin/a01/y274chen

and test cases at

 ~/marking/a${Anum}/test.pt/0

 ~/marking/a01/test.pt

to run and produces result in

 ~/marking/a${Anum}/test.pt/0.${runid}/${sid}

 ~/marking/a01/test.pt.test1/y274chen

[top](#_top)

**3.2 distrst**

distrst -t t -s '\*' a${Anum} pt/0 ${runid}

eg:

distrst -t t -s '\*' a01 0 finalize

-t t: just use it

-s '\*': run single student on every student

This is the same as above but just running on every student in the handin folder.

The advantage of using distrst is that, distrst distributes the testing to different school servers. So multiple assignments are tested at the same time to save time.

The result is again in

 ~/marking/a${Anum}/test.pt/0.${runid}

 ~/marking/a01/test.0.finalize

[top](#_top)

**4. Running Autotest**

So after due date of the assignment, do the following step to finish the autotesting.

**I. prelimanray steps**

 1. [**Download CSV**](#CSV) file that contains submitted class list of the assignment from **MARKUS**

 2. **Create a${Anum}autotest** in ~/handin

 3. **Make .subfiles** in ~/handin/a${Anum}autotest (same as in~/handin/a${Anum} )

 4. In ~/marking, create a symlink **`ln -s a${Anum} a${Anum}autotest`**

Why doing this?

1. CSV file is for future use, so we can checkout the assignment submitted just before due time to anothre folder which is -

2. a${Anum}autotest. Because Markus will update the handin folder when a student make a submission - even after due time.

So we want to separate the legit submission from the original folder and use it for testing.

Then we are going to test a${Anum}autotest, but in marking folder, we only have a${Anum}. 4. So, we use symlink to link

a${Anum}autotest to a${Anum}. Doing anything to a{Anum}autotest is same as to a${Anum}

Download CSV : log in Markus -> assignment -> groups -> download cvs.

[top](#_top)

**II. Checkout - Test – Commit**

**1. Checkout**

1.1. Using python script created by Nick Lee - [CheckoutandCommit.py](https://www.cs.uwaterloo.ca/twiki/view/ISG/MarkUsScripts), **configure the python script**. (It is documented, follow the instruction - CSV file is used here)

1.2. Execute **`python CheckoutandCommit.py checkout runcmds`** (use viewcmds to just see it)

 - the latest submission before due time is now in ~/handin/a{Anum}autotest

1.3 **Check** the result of checkout (you can do it when still checking out)

Take approximately 15~30min depends on server condition.

**2. Test**

2.1 **Run distrst** with a${Anum}autotest

 - `distrst –t t –s ‘\*’ a01autotest 0 finalize` ; for assignment 1, finalize is arbitrary.

2.2 **Check** the test result of autotest, halt and fix and re-run if the result goes out wrong (has bug in test case). (Do it ASAP)

- 33 cases for 600 students takes 1hr.

**3. Commit**

3.1 **Configure** [CreateGradedAssignment.sh](https://www.cs.uwaterloo.ca/twiki/view/ISG/MarkUsScripts) (created by Nick Lee, and documented, so follow the instruction, it is now python script)

3.2 **Run** the script, so in each student folder in ~/marking/a${Anum}autotest GRADEDASSIGNMENT.ss is created

 (which contain the result of all test and files, which is for marking purpose and need to be commit to Markus)

3.3 **Check** the result

3.4 Execute **`python CheckoutandCommit.py commit runcmds`** which commit the GRADEDASSIGNMENT.ss to Markus

 (configuration is same as when you checkout)

3.5 **Check** on Markus

[top](#_top)

**III. Markus**

1. Change the deadline to just **after finish commit but before now**

 --|---------|----------------|----

 after change now

 commit to

2. **Collect all submission** (work like CheckoutandCommit.py, collect the one just before deadline.

3. Change the deadline **back to original**

4. There might be some **glitch with late submission** (rarely, haven't seen one, as long as the collected submission has GRADEDASSIGNMENT.ss from Markus),

 so check the log of submission. type

**ssh markus@linux.student.cs.uwaterloo.ca "grep -e 'timeout.\*a${Anum}' /u/markus/bin/logs/markus\_cs135\_w-post-commit.log"**

(change w to f for fall s for spring)

5. **Check** if late submission has any **error**.

[top](#_top)

**IV. Notice Related To Marking**

1. **Only mark GRADEDASSIGNMENT.ss** since other file **may be late submission**

2. If a student hands in **late**, his/her GRADEDASSIGNMENT.ss **will not appear** (so if you get email about missing GRADEDASSIGNMENT.ss, check the log to see if it is a late submission or not first.)

[top](#_top)

**Hope you have a fun time being a TUTOR!!!**