Electronic Submission

In CS 182, we use an electronic submission system to handle programming assignments and most written work. The submission system handles getting the assignments, sharing the code between members of a programming team, and submitting "working" versions of the code for testing or grading.

Note that CS 182 is using a new electronic submission system—it is not the same as the submit system you used when you took CS 70. Although this system was used successfully with CS 70 last semester, you should be on the lookout for bugs. If you experience problems, please let us know by sending mail to cs182help@cs.hmc.edu.

Setting Your Path

The CS 182 submission commands are found in the CS 182 programs directory, which you will need to add to your path. If you use `tcsh` as your shell, add the following command to the end of your `.login`:

```
set path = (/cs/cs182/bin $path)
```

If you have changed your shell to `bash`, add the following command to the end of your `.bashrc`:

```
export PATH=/cs/cs182/bin:$PATH
```

Note that the CS 182 directory *must* be placed at the *front* of your path (as it is above), not at the end.

Getting Files for the First Time

When you start a new assignment, there will be some files provided to you. The `cs182checkout` command gets these files and sets up the directory for an assignment. The syntax is

`cs182checkout` *assignment-name*

For example, if the first assignment was called `hw1`, you would type

```
mkdir -p ~/cs182
cd ~/cs182
cs182checkout hw1
```

Submitting Your Files

You will be editing a "working copy" of your files, but the definitive repository for your CS 182 files will be a master copy kept in the grader account area, where it can be

accessed by the course staff. When you are working as part of a group, everyone in the group will access the same shared repository. You will not be editing your master copy directly—instead you edit your working copy and then, whenever you have made a significant change (e.g., adding a new function or fixing a bug in an existing one), you should update the master copy using the `cs182submit` command. Your instructor and graders will use the shared master copy when we help you debug your assignment, and will also examine it to determine whether you are making progress.

To update the master copy from your working copy, you should execute

```
cs182submit
```

Optionally, `cs182submit` can take a list of filenames specifying which files to act on. You may wish to use this form if you only want to update one of the files in the master repository and *ignore* changes made to other files in your current working copy.

When it is given no arguments, `cs182submit` only checks in files from the working directory that also exist in your master copy. Thus, if you create a new file, that file will *not* be added to the master repository unless you explicitly check it in by name. In other words, by default, all files that didn't come with the assignment are assumed to be scratch files that you don't want to share. (In the output from `cs70submit`, filenames preceded by the letter `T` are submitted, whereas filenames preceded with a question mark (`?`) are *not* submitted.

For example, if you create a new file called `mytests.cpp` in your working copy of the assignment and you consider it to be part of your assignment, you should add it to your master copy by running

```
cs182checkin mytests.cpp
cs182submit
```

to add that file to your master repository and check it in. (Note that you could also type `cs70submit mytests.cpp`, but that form of `cs182submit` is the form that explicitly specifies files to check in; thus that command would only check the single file `mytests.cpp` into the master repository before marking the currently checked-in versions of everything in your repository as a submission.)

Once a file such as `mytests.cpp` has been added to your master repository, you can just type

```
cs182submit
```

from then on to check in the files in your assignment, including `mytests.cpp`.

You should run `cs182submit` whenever you've reached a major milestone. It is wise to submit your code each time it seems to be "more-or-less working" even if it doesn't do everything it is supposed to (in case subsequent changes break something). In general, if your code compiles and doesn't crash immediately when you run it, it's probably worth submitting.

Some assignments support a special argument to `cs182submit` of the form

```
cs182submit -t
```

which submits your code and also runs tests on your submission and emails you the results.

Resynchronizing with the Master Copy

If you are working with a partner, and they have edited their working copy of the files, and then checked their changes into your (shared) master repository using `cs182submit`, you can update your working copy to include your partner's changes by running

```
cs182resync
```

The resync process works *even if* you have also edited the same files, provided that you have not been editing the same lines. If you have both edited the same part of the file, the resync will produce a file that includes both parts, and you will have to merge the changes by hand.

If you get an error from `cs182submit` or `cs182checkin` telling you that you should run `cvs update`, it is a sign that you need to run `cs182resync` because your partner has checked in changes.

Checking in without Submitting

Sometimes you may want to check in your files *without* it counting as a "submission" (e.g., because your submitted code *works*, whereas your latest modifications broke something).

To check in without submitting, run

```
cs182checkin
```

This command updates the master repository, but the modified files do not count as a submission.

Revisiting History

The submit system keeps a copy of *every* submitted version of your files. If you want to recover an old version of your assignment, you can use the command

`cs182checkout -O` *hours assignment-name*

This command will check out an earlier version of your assignment, from *hours* hours ago, into a directory named *assignment-name*`.old`. To avoid confusion, this directory is *not* a CS 124 working directory—you cannot submit from it. Instead you should copy any files you need from the old version into your working directory.

Working Away from turing

We assume that you will be working on turing, either directly (on an NCD terminal) or over an `ssh` connection. It is possible to undertake the homework on your own machine(s), but you *must* keep your files on turing in sync with those on your machine. I will examine your files during the week of the assignment to see how much progress you are making—if you have not checked in any versions, I will have to assume that you have done no work. If you are using Mac OS X or Linux, the command

`rsync -aC -e ssh` *source destination*

may be a useful way to keep your files in sync (see the `rsync` manpage). For example, if you are `jsmith` on turing, you might run

```
rsync -aC -e ssh jsmith@turing.cs.hmc.edu:cs182/hw1 ~/mycoursework/
```

on your personal machine to download files from turing, and then upload them after an editing session with

```
rsync -aC -e ssh ~/mycoursework/hw1 jsmith@turing.cs.hmc.edu:cs182/
```

After uploading them, you would also need to log on to turing and run `cs182submit` (or `cs182checkin`).

Peeking at the Internals

The CS 182 submission system is just a wrapper around CVS, a source-code control system widely used in team projects. If you are interested in the internal CVS commands used by the submission system, you can run any of its commands with `-v` as the first argument to print the CVS commands it executes. You will not be able to run these commands by hand, and you should not execute any CVS commands directly.

You are not allowed to attempt to subvert the CS 182 submission system directly using CVS commands, or in any other way.