

# CS> System User Manual

Jeb Brooks

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Basic Use</b>	<b>6</b>
2.1	General Use . . . . .	6
2.1.1	Logging in/Password Recovery . . . . .	6
2.1.2	The Main Navigation Bar . . . . .	6
2.2	Student Use . . . . .	8
2.2.1	Submitting Homework . . . . .	8
2.2.2	Viewing Grades . . . . .	9
2.3	Grader Use . . . . .	10
2.3.1	Grading Submitted Homework . . . . .	10
2.4	Instructor Use . . . . .	14
2.5	Wiki Use . . . . .	14
<b>3</b>	<b>Administration</b>	<b>15</b>
3.1	Server Structure . . . . .	15
3.2	Set-up . . . . .	16
3.2.1	MongoDB Set-up . . . . .	16
3.2.2	File-system Set-up . . . . .	17
3.2.3	RabbitMQ Set-up . . . . .	18
3.2.4	Create config.py . . . . .	18
3.2.5	Flask Front-end Set-up . . . . .	19
3.2.6	Celery Worker Back-end Set-up . . . . .	20
3.2.7	Bootstrap the Administrator Account . . . . .	20
3.3	Managing Courses . . . . .	21
3.3.1	Creating Courses . . . . .	21
3.3.2	Adding Instructors . . . . .	21
3.3.3	Deactivating Courses . . . . .	21
3.4	Creating User Accounts . . . . .	22

3.4.1	Creating via web . . . . .	22
3.4.2	Creating via command line . . . . .	22
<b>4</b>	<b>Development</b>	<b>24</b>
<b>5</b>	<b>Command Line Reference</b>	<b>25</b>
5.1	Setting up the Environment . . . . .	25
5.2	Using the provided scripts . . . . .	26
5.2.1	Adding batches of users . . . . .	26

# List of Figures

2.1	The header of the main page before logging in. . . . .	6
2.2	The login page. Reset password is at the bottom. . . . .	7
2.3	The main navigation bar . . . . .	7
2.4	A list of problems for a course. Problems are grouped by <b>Assignment Group</b> . . . . .	8
2.5	The page for submitting files for a problem . . . . .	9
2.6	The My Grades page . . . . .	10
2.7	A list of all currently assigned problems in CS Test and the current grading status of the problems . . . . .	10
2.8	The current status of grading for all submissions in this problem	11
2.9	The top of the grading page . . . . .	12
2.10	The comments and files section of the grading page . . . . .	13
3.1	Possible server configuration for <b>CS&gt;</b> . . . . .	16
3.2	Red: The user Dropdown, Blue: The admin dashboard link .	21
3.3	The admin users panel . . . . .	23

# Chapter 1

## Introduction

The **CS>** system is an all in one submission system and grading platform designed primarily for use with CS coursework. It is designed to support basic grading functionality and auto-grading capabilities. Additionally it is designed to be extended to support new testing systems through a simple plug-in system which can hot-load new plugins on a live instance.

This document will attempt to get you familiar with the **CS>** system and guide you through using and maintaining its various features. **CS>** provides the following major features:

- Logical assignment/problem grouping
- Fully featured grade-book system. Supports adding groups and columns to the grade-book which are not based on student submitted work (e.g.. Participation Points or Quizzes)
- Markdown enabled grader comment system.
- Automatic grading based on unit tests (modifiable via plug-ins)
- Automatic late grade calculation (modifiable via plug-ins)
- Built-in wiki support for problem descriptions and notes for graders. The wiki also supports various permissions levels.
- Support for command line control of the system.
- Support for listing currently active tutors and locations so students can get help on problem sets.

This document will attempt to guide you through the use of these features. Because not all features are useful to every class of user this document is divided into three chapters. The [Basic Use](#) chapter should be enough for instructors, tutors, and students. The [Administration](#) chapter will cover topics required to set-up and maintain the **CS>** system. Finally, the [Development](#) chapter will cover topics required to develop new plug-ins and modify the core functionality of the system.

## Chapter 2

# Basic Use

This chapter will attempt to cover all of the basic use cases for the various user situations. In addition it will describe the built in wiki system which ships with the **CS>** system.

### 2.1 General Use

This section covers general account management.

#### 2.1.1 Logging in/Password Recovery

Most of the features for the site are restricted to registered users. The login button is located in the top right corner of the page, see Figure 2.1.

If you forget your password (and the system has been configured to send emails, see Section 3.2.4) you may recover your password on the login page, shown in Figure 2.2.

#### 2.1.2 The Main Navigation Bar

Most navigation will be done through the use of the main navigation bar at the top of the screen, see Figure 2.3. The navigation bar's content will change based on the permissions of your account. Figure 2.3 shows an account that occupies all three roles, student, grader, and instructor.

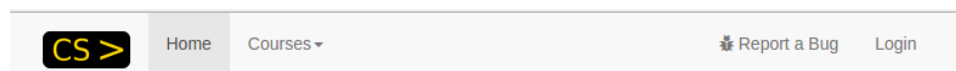


Figure 2.1: The header of the main page before logging in.

## Sign In

**Username**

**Password**

☐ **Remember Me**

Sign In

Forgot your password?

Reset Password

Figure 2.2: The login page. Reset password is at the bottom.

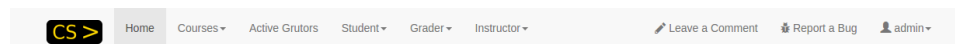


Figure 2.3: The main navigation bar

From right to left the links on the navigation bar are:

**Home** Takes you to the main page.

**Courses** Presents a drop-down menu which can take you to the homepage for active courses in the system.

**Active Grutors** Takes you to a page which shows where the current active grutors are for courses you are involved with (enrolled, grading, or teaching).

**Student** This only appears if you are a student in a course and will present a drop-down that can take you to the problem submission page for your course or a summary of your grades.

**Grader** This only appears for graders and instructors of a course. It presents a drop-down for choosing which course you want to grade assignments for.



**Instructor** This only appears for instructors. It presents a drop-down for which course you want to edit settings for.

**Leave a Comment** Takes you to a feedback page. Feedback provided through this page is sent to the admin.

**Report a Bug** Takes you to the github issues page for the **CS>** system.

**<username>** Presents a drop-down for account settings and logging out.

## 2.2 Student Use

The primary actions students will take in **CS>** is submitting homework assignments and checking grades.

### 2.2.1 Submitting Homework

To submit homework start by clicking the *Student* link on the navigation bar and selecting the course which you want to submit work to. This will bring you to a page like in figure 2.4.

Assignment Group	Problem	Submit	Status	Is Late	Due Date	Maximum Points
Week 1	Problem 0	<a href="#">Submit</a>	Unsubmitted		May 6, 2015 11:59 PM	20.00
Week 0	Problem 0	<a href="#">Submit</a>	Graded <a href="#">View</a>		May 6, 2015 11:59 PM	5.00
	Problem 1	<a href="#">Submit</a>	Unsubmitted		May 6, 2015 11:59 PM	10.00

Figure 2.4: A list of problems for a course. Problems are grouped by **Assignment Group**

Problems are grouped by **Assignment Group** which may refer to single weeks or any other logical grouping of multiple problems. Assignment groups are displayed in reverse order so that the most recently assigned group is at the top of the list and thus easier to find.

Once you have located the problem you wish to submit click the blue *Submit* button. This will direct you to a page like in figure 2.5.

On the submit page there is information about what files are expected, a place for you to choose the files you are submitting and a drop-down to select an optional partner.

Required Files For Submission:

Required Files For Auto-Grading:

CS Test/Week 1/Problem 0

Files

No file chosen

Partner

Figure 2.5: The page for submitting files for a problem

**Required Files For Submission** These files must be present in the set of files you select or else the system will reject the submission.

**Required Files For Auto-Grading** If any of these files are not present the submission will be accepted but the auto-grader will not run.

**Files** Will open a file selection window. To submit multiple files you may either select multiple files in the window, or submit a zip file. If you need to preserve directory structure submit a zip file.

**Partner** A drop-down of all the students in the course for selecting a partner. If you select a partner your partner does not have to submit the solution as well.

### 2.2.2 Viewing Grades

There are two ways to view grades. First on the problem list page, see figure 2.4, you can click the *view* button to see the grade and grader comments for an individual problem.

If you want an overview of all of your grades you can go to the *My Grades* page which is under the student drop-down menu. The My Grades page will display something like in figure 2.6.

#### Test Semester/CS Test

	Week 0		Week 1	Total
	Problem 0	Problem 1	Problem 0	
Max Score	5.00	10.00	20.00	35.00
Your Scores	0.00	0.00	0.00	0.00

Figure 2.6: The My Grades page


## 2.3 Grader Use

Graders can assign grades in one of two ways, they can grade submitted assignments, or they can enter external grades into the grade-book. This section will cover both of these options.

### 2.3.1 Grading Submitted Homework

First to grade a student's submitted work click the *Grader* drop-down from the navigation bar (see Figure 2.3) select the course you wish to grade. This will bring up a list of all the problems currently assigned as seen in Figure 2.7.

#### CS Test

 Grade Book

#### Assignments

Assignment Group	Problem	Due Date	Submissions	Status	Grade
Week 1	Problem 0	May 6, 2015 11:59 PM	0	Done	<a href="#">Grade</a>
Week 0	Problem 0	May 6, 2015 11:59 PM	1	Done	<a href="#">Grade</a>
	Problem 1	May 6, 2015 11:59 PM	1	Unfinished (1-0)	<a href="#">Grade</a>


Figure 2.7: A list of all currently assigned problems in CS Test and the current grading status of the problems

On this page click the *Grade* button for the problem you wish to grade. This will bring up a list of all of the students in the class with the status of their submission, see Figure 2.8.

There are two ways to claim a submission to grade. First, you can click the *Random Grade* button. This button will atomically select a currently

## Submissions for: CS Test/Week 0/Problem 1

Not Graded	In Progress	Done Grading
1	0	0

 Random Grade

Username	Submission Status	Is Late	Submission Time	Scored Points	Grade Submission	Graded By
admin	Submitted (Auto-graded, Waiting for Grader)		May 6, 2015 4:39 PM	0.00/10.00	<a href="#">Grade</a>	None
jdoe	No Submission		N/A	0.00/10.00	<a href="#">Grade</a>	N/A

Figure 2.8: The current status of grading for all submissions in this problem

ungraded assignment for you. Second, you can click the *grade* button for the student you wish to grade. Once you have selected a problem to grade you will see a page like in Figures 2.9 and 2.10.

Submission for: CS Test/Week 0/Problem 1

User: admin

### Submission Properties

Submission Number	1 ▾
Submission Time	May 6, 2015 4:39 PM
Partner	None
Submission Status	Grading in progress
Toggle Late	<button>Toggle</button>
Grade Notes	None
Regrade Submission (Removes comments and scores for this student)	<button>Regrade</button>
<div><button>Cancel Grade</button> <button>Finish Grading (save all)</button></div>	

### Rubric


Section	Score	Max Points
Points	<input type="text" value="0.0"/>	10.00
<div> Save Grades</div>		

Figure 2.9: The top of the grading page

Grader Comments
[Save](#)

No Comments

Grader Comments (Preview)

Auto-Grader Comments
[Save](#)

Auto-Grader Comments (Preview)

No tests provided. Testing complete.

### Student Files


Filename	View	Download
 admin_header.png	<a href="#">View</a>	<a href="#">Download</a>

Figure 2.10: The comments and files section of the grading page

## **2.4 Instructor Use**

Foo

## **2.5 Wiki Use**

# Chapter 3

## Administration

### 3.1 Server Structure

The **CS>** system is designed to be modular and expandable as the number of users of the system grows. To accommodate this each of the five major components are designed to be operated as a separate server. The five components are:

1. Python-Flask Front-end Server
2. Python-Celery Grading Worker
3. File-system for submission storage (pick your favorite one)
4. MongoDB for storing grades
5. RabbitMQ for passing grading requests to workers

In the most basic configuration, which is most useful for testing, all of the required components will be on one machine. As the requirement for redundancy and scalability increases components may be moved to separate servers and replicated. Figure 3.1 shows one possible configuration of the servers which provides redundancy on both the front and back end. It is important to note that a load-balancer is required to support multiple front-ends.

**Note:** Currently even though MongoDB and rabbitMQ support distributed options the system has never been tested using that configuration. This is a feature which may be researched in the future if it becomes needed.



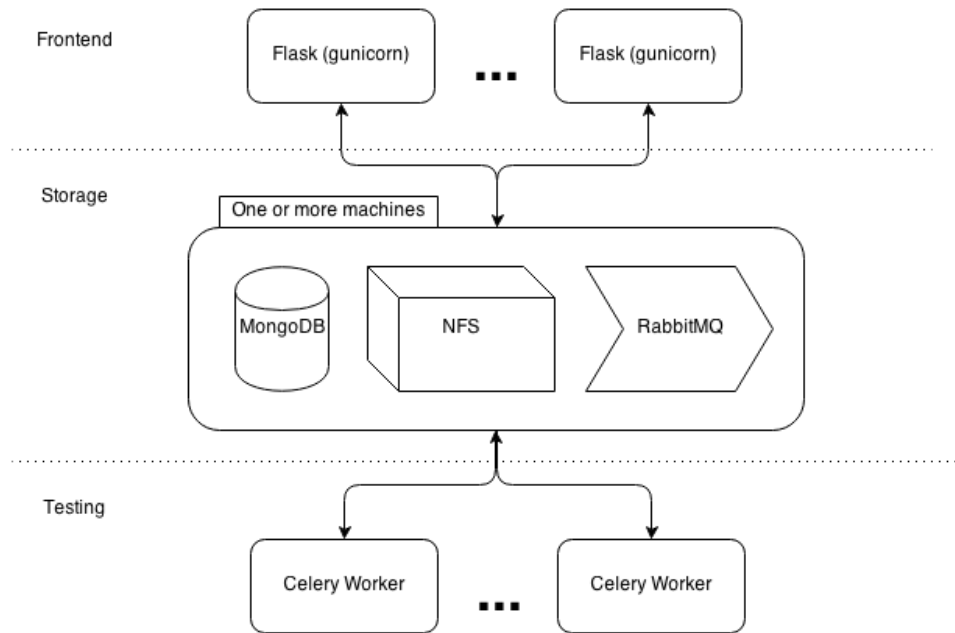


Figure 3.1: Possible server configuration for **CS>**

## 3.2 Set-up

Once you have decided on the basic configuration for your servers setting up the requisite systems requires a little work. For each server type I will provide basic set-up instructions. These instructions are listed in order to prevent dependency issues.

### 3.2.1 MongoDB Set-up

To set-up a brand new MongoDB instance you can use the instructions below. If you already have a running version of MongoDB you can adapt the instructions or request help from your database administrator.

1. Get the latest version of MongoDB from [here](#).
2. Follow [these](#) instructions to create an administrator account.
3. Edit `/etc/mongodb.conf` so that `bind_ip` is commented and uncomment `auth = true`. Additionally it is recommended to change the port that the server listens on to prevent attacks.

4. Reload *mongod* and log into the *mongo* shell with  
\$ `mongo -u {username} -p --authenticationDatabase admin`
5. Type `> use submissionsite` to create the *submissionsite* database
6. Add a user for this database by typing

```
> db.createUser(  
  { user: "grader",  
    pwd: "{your password}",  
    roles: [{role: "dbOwner", db: "submissionsite"}]  
  }  
)
```

7. Finally `> quit()` the mongo shell.

### 3.2.2 File-system Set-up

Setting up the file-system is pretty easy. There are two major requirements for the file-system, one, it must be accessible from all machines in the cluster, and two it must have a minimal required directory structure.

Because exporting a directory is complicated and other better resources exist on the internet I leave figuring out how to export your directory as an exercise to the reader.

Once you decide how to export your chosen storage directory you must give it some basic structure. This structure is as follows:

- submissions
- photos
- plugins
  - autograder
  - latework

Once these paths exist in your storage directory you are good to go.

### 3.2.3 RabbitMQ Set-up

RabbitMQ is a distributed queue which handles sending auto-grading requests to the various celery workers. Setting up this part of the system can be very easy but depending on your security requirements you may want to enforce more stringent permissions. To set-up RabbitMQ do the following:

1. Install *rabbitmq-server*.
2. Run  

```
$ sudo rabbitmqctl add_user {username} {password}
```
3. Run  

```
$ sudo rabbitmqctl set_permissions -p / {username} ".*" ".*" ".*"
```

RabbitMQ is now in a very basic workable configuration.

### 3.2.4 Create config.py

The `config.py` file handles all of the connection configuration for the system. Inside `config.py` there are several sections that need to be modified using configuration information from the steps above.

1. Give your own `SECRET_KEY`
2. Configure MongoDB connection properties

```
MONGODB_SETTINGS = {  
    'DB': 'submissionsite',  
    'username': '{dbuser}',  
    'password': '{dbpassword}',  
    'host': '{db_ip}',  
    'port': '{db_port}'  
}
```

3. Configure the Celery broker URL  

```
CELERY_BROKER_URL="amqp://{rabbit_user}:{rabbit_pwd}@{rabbit_ip}"
```

4. Set the where the file-system will be mounted for the front/back-end servers

```
STORAGE_HOME="{path_to_mount_point}"
```

**Note:** If you are storing the data locally you still have to set this flag but instead want to set `STORAGE_MOUNTED=False`. This disables

a check to make sure that the directory is mounted before performing writes.

5. Set up the system email (This is used for sending password reset requests)

```
SYSTEM_EMAIL_ADDRESS = "{server_email}"
SMTP_SERVER = "{smtp_server}"
```

This configuration file should be passed to all front/back-end servers.

**Note:** There is an additional setting `GRADER_USER` which is currently set to `None`. This setting is supposed to set which user to switch to as the auto-grader but it is currently non-functional.

### 3.2.5 Flask Front-end Set-up

The Flask based Front-end of the system serves dynamically generated web-pages to the users. To set-up the flask front-end do the following:

1. Ensure that `python-dev` (or its equivalent package) is installed on your platform.
2. Check out the repository
3. `$ cd <your_repository>`
4. Copy `config.py` to the repository
5. Create a virtual environment using `virtualenv <venv_name>`
6. **Temporary Fix** The current version of mongoengine does not function with the latest version of pymongo. To solve this install version 2.8 `$ <venv_name>/bin/pip install pymongo==2.8`
7. Install the following packages with pip  
`$ <venv_name>/bin/pip install flask flask-script flask-bootstrap flask-login flask-markdown mongoengine flask-mongoengine WTForms python-dateutil celery psutil python-magic bleach gunicorn gevent flower`
8. Launch the front-end processes  
`$ <venv_name>/bin/celery flower -A app:celery`  
`$ <venv_name>/bin/gunicorn -w 4 -k gevent -b 0.0.0.0:80 app:app`

**Note:** You may have to modify some permissions to allow *gunicorn* to bind to port 80. Additionally if you are using a load balancer you may want to bind it to some other higher port.

### 3.2.6 Celery Worker Back-end Set-up

Setup for the Celery Worker is almost identical to the Flask front-end but requires fewer packages. The instructions are displayed below:

1. Ensure that `python-dev` (or its equivalent package) is installed on your platform.
2. Check out the repository
3. `$ cd <your_repository>`
4. Copy `config.py` to the repository
5. Create a virtual environment using `virtualenv <venv_name>`
6. **Temporary Fix** The current version of mongoengine does not function with the latest version of pymongo. To solve this install version 2.8 `$ <venv_name>/bin/pip install pymongo==2.8`
7. Install the following packages with pip  
`$ <venv_name>/bin/pip install flask flask-script flask-bootstrap flask-login flask-markdown mongoengine flask-mongoengine WTForms python-dateutil celery psutil python-magic bleach`
8. Launch the worker processes  
`$ <venv_name>/bin/celery worker -A app:celery`

### 3.2.7 Bootstrap the Administrator Account

All of the previous sections have gotten a working version of the system but currently there are no accounts in the database. To begin creating accounts there must be an administrator account.

The easiest way to create the `admin` account is to run

```
$ <venv_name>/bin/python run.py
```

then kill this process. `run.py` launches the debug server which also checks for an administrator account if no account is found it creates it. You can find the password for the default `admin` account in `run.py`.

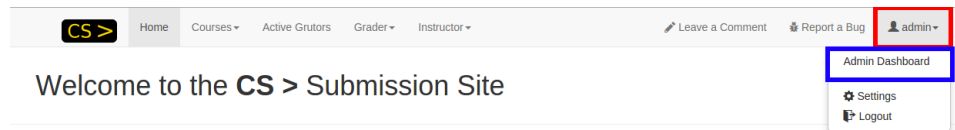


Figure 3.2: Red: The user Dropdown, Blue: The admin dashboard link

## 3.3 Managing Courses

### 3.3.1 Creating Courses

The administrator is required to create all of the courses for the system. When a course is created all administrators are invisibly added as instructors. Any other instructors must be explicitly added by an administrator.

To add a new course to the system first navigate to the *Admin Dashboard* (See Figure 3.2). Once in the Admin Dashboard click the link for the *Courses* panel. Fill in the required information and click the *Create Course* button.

### 3.3.2 Adding Instructors

Once a course has been created instructors (other than the current administrators) must be added to the course. This requires the following actions:

1. Navigate to the course settings page. If you just created the course find it in the *Active Courses* list and click *Administer*. If you are on the main page click the *Instructor* drop-down and click on the course link.
2. Scroll to the bottom of the course settings page.
3. Under the instructors heading enter the user-name of the desired instructor. (If the instructor does not have an account yet see Section 3.4). Click the plus button.
4. That user is now an instructor.

### 3.3.3 Deactivating Courses

When a course is finished it is desired to prevent that course from being modified. Additionally we want currently activated courses to be prioritized when displaying lists of courses to users. To accomplish this **CS>** allows the administrator to **Deactivate** a course. When a course is deactivated

it no longer accepts submissions and is moved from the main drop down menus into a separate page of old courses.

To deactivate a course first navigate to the Admin Dashboard and then to the Courses panel. Once on the courses panel find the desired course and click the *Deactivate* button.

**Note:** Some of the archival pages are still works in progress, but the framework is there to accomplish these features. If you run into problems with deactivated courses please submit a bug report.

## 3.4 Creating User Accounts

One of the primary roles of the `admin` account is to create user accounts. There are two methods for creating accounts for users.

### 3.4.1 Creating via web

Web creation is the easiest way to create a small number of accounts. The steps are as follows:

1. Navigate to the *Admin Dashboard* by clicking the user drop-down menu and then *Admin Dashboard*. Figure 3.2 shows the locations of the links.
2. Once in the *Admin Dashboard* navigate to the user panel using the top bar. (See Figure 3.3)
3. Fill in the information fields and click **Create User**. (See Figure 3.3)

### 3.4.2 Creating via command line

Creating new users via the command line is generally one of the easiest ways to add large batches of users. There are already several scripts (Located in `app/scripts` in the repository) which demonstrate how to add users from various files (CSV files with different column layouts). Below are general instructions for adding a user via command line:

**Note:** To correctly use the mongo interface for the system in python you must export `PYTHONPATH` as the root directory of the repository.

```
$ export PYTHONPATH=<path_to_repository>
$ <venv_name>/bin/python
```

Grader Admin Home Statistics Courses Users

Add a User

Username  
eg. jdoe

First Name  
eg. Jamie

Last Name  
eg. Doe

Email  
eg. jdoe@example.com

Password  
eg. asdf (defaults to asdf)

Create User

Figure 3.3: The admin users panel

```
...python start message...
>>> from app.structures.models.user import *
>>> user = User()
>>> user.username = "<username>"
>>> user.firstName = "<firstName>"
>>> user.lastName = "<lastName>"
>>> user.email = "<email>"
>>> user.setPassword("<password>")
>>> user.save()
```

If we then want to add the user to a course we want to continue by doing

```
>>> from app.structures.models.course import *
>>> c = Course.objects.get(semester="<semester>", name="<name>")
>>> #One or more of these
>>> user.courseStudent.append(c)
>>> user.courseGrutor.append(c)
>>> user.courseInstructor.append(c)
>>> user.save()
```



## Chapter 4

# Development

Foo

## Chapter 5

# Command Line Reference

Some functionality is too infrequently used or too cumbersome to provide through the web interface. Including:

- Assigning all students a comment and a score for a given problem.
- Changing the password of a student.
- Adding a user to a large number of courses at once.
- Making a user an administrator.
- Creating a large number of user accounts from a file.

Other functionality simply hasn't been made available yet in the web interface and will be added later. Including:

- Marking all submission grading status as "Done".
- Sending all submissions through the auto-grader again.

This section will demonstrate how to use the scripts provided to perform these functions as well as other useful tips for the command line.

### 5.1 Setting up the Environment

Because of the way the python import system works, before you can run any scripts or do work on the command line you must first export the environment variable PYTHONPATH to the location of the root of your repository. One easy way to do this is:

```
$ cd <your_repository>
$ export PYTHONPATH='pwd'
```

## 5.2 Using the provided scripts

The repository contains several useful scripts in the directory `app/scripts`.

### 5.2.1 Adding batches of users

The three scripts `addUsersFromList`, `addUsersFromSakai`, `addGrutorsFromList` all provide examples for how to read CSV style files and creating user accounts from them.

Theses scripts were designed for the data format used by the Harvey Mudd College Registrar and so may not be immediately useful at other schools, but they will provide good reference.