When: 2/21 - 3/18 MWF @ 8am
Where: 216 Olin Hall
Drop deadline: 2/28, one week into the course
Office hours: Wednesdays 12-1pm
Contact info:
- Office: 4139 Upson Hall
- Email: hussam at cs.cornell.edu

Course: http://www.cs.cornell.edu/courses/cs2044
CMS: http://cms.csuglab.cornell.edu/
Class is 1 S/U credit hour
No exams, just 3 assignments
You must complete all 3 assignments to pass this course
Course Goals

- Gain experience with Unix-like environments
- First hand experience writing scripts to automate everyday tasks

Topics include:
Bash scripting, Regular Expressions, AWK, Make, Python ...etc
If you’re registered for this course, you have an account with the CS undergraduate lab (csuglab). http://www.csuglab.cornell.edu/

You can SSH into the machines in the lab for remote access. Instructions are available at the csuglab webpage. For Windows machines, use PuTTY to connect.

To convert from csh to bash on startup, add the following to your ∼/.login file:

```
if ( -f /bin/bash ) exec /bin/bash --login
```
If you have a Windows machine, there are other options to access a Linux environment:

- **Cygwin**: a Linux-like environment for Windows (http://www.cygwin.com/)
- **Any Linux Live CD** (http://www.livecdlist.com/)

If you have Mac OS X, it already runs on BSD.
Let’s dive in!
A quick review
General syntax to invoke Unix commands

```plaintext
command -a -b -c input1 input2
```

Where:
- `command` is the command to invoke
- `-a -b -c ..` are flags to modify command behavior
- `input1 input2 ..` are input files to command
Moving around

- `pwd` - prints current directory
- `cd dir` - go to `dir`
- `cd ..` - go to parent directory
- `cd ~` - go to home directory
File manipulation

- `cp file1 file2` - copy file1 to file2
- `mv file1 path` - move file1 to path
- `rm file1` - remove file1
- `cat file1 file2` - concatenate file1 and file2 and print to screen
- `mkdir dir1` - create directory dir1
- `ls dir1` - list the content of directory dir1
Each application is given an:
- **Input stream**: receives input, usually from user via keyboard
- **Output stream**: usually prints to screen
- **Error stream**: log error messages, by default to screen

Can redirect streams to do interesting things:
- `cmd > file` - redirects `cmd` output, writes to `file`
- `cmd >> file` - redirects `cmd` output, appends to `file`
- `cmd < file` - redirect `cmd` input stream to read from `file`

**Examples**

```
$ echo "Welcome to CS2044" > input
$ tr '[:lower:]' '[:upper:]' < input
```
We can take a program’s output stream and feed it as an input stream.

This is a very powerful idea!

`cmd1 | cmd2` executes `cmd1` and sends its output to `cmd2`.

**Example**

```
$ echo "How many words are in this sentence" | wc -w
7
```
What is a shell script?

- A script is a text file that a special program (called an interpreter) can read and execute.
- `bash` is not only a shell, but it is also an interpreter
- A simple `bash` script is just a text file with a list of commands to execute one by one

Benefits of scripting:

- Automate common tasks
- Usually smaller, cleaner, and easier code relative to C, Java, C# ..etc
Create a file with name `HelloWorld.sh` and the following content:

```
#!/bin/bash

echo "Hello, my name is $USER.
Greetings to the world from my $SHELL shell
on $HOSTNAME which is a machine of type
$MACHTYPE."
```

To be able to run this script, make it executable:

```
chmod +x HelloWorld.sh
```

Finally, run the script:

```
./HelloWorld.sh
```