## CS 7600 Virtual Machine

You can do your work on any Linux machine; however this is the "official" one for this class – i.e. if there's something wrong with it then I'm on the hook to fix it, while if it's your own machine...

You should be able to run the virtual machine image under any of the following platforms:

- VirtualBox on Windows, Linux, and Mac OS X (free from virtualbox.org)
- VMware Player on Windows and Linux
- VMware Workstation on Windows (and Linux?)
- VMware Fusion on Mac OS X

You can probably use it with KVM on Linux and Parallels on Windows/Mac as well, but you're on your own.

To install the virtual machine download the file
http://pjd-1.ccs.neu.edu/files/CS7600-f16-Ubuntu32.ova

Virtualbox instructions:

1. before you do anything else you will have to go to Preferences/Network and click '+' to add a host-only network.
2. Use the "Import" function in VirtualBox or VMware to install the .ova file.
3. Click the "settings" button, and when the settings window comes up, click "OK"
(for some reason this step seems to be needed, even though it doesn't seem to do anything)
4. Start the VM, log in as user 'student', password 'student'.
Note that it will ask you if you want to upgrade to Ubuntu 16.something – click "no".
5. Create a new account with the same user ID as your CCIS account.
- click the weird Ubuntu logo at the very top left of the screen
- select "Settings" and "Users and Groups"
- click "+" to add a new user, set the user name, user id, and password
- now click "Manage groups", and add the new account to the "admin" group.

Note that if you get an error message on starting the VM ("Failed to open a session for the virtual machine CS5600-f16-Ubuntu32") then it means you forgot step 3.

This has been tested on VirtualBox 5.0.26 on Mac OS X 10.10.5. Please let me know if you have difficulties on other versions or operating systems.

## Git Version Control for CS 7600

For homeworks in this class you will be using the Git distributed version control system. Git works with ***repositories*** and ***local copies***. A repository is a central copy of the files, while a local copy is just what it sounds like - a copy of the repository on your machine that you can edit and work with.

### Theory of Operation

You ***clone*** a repository to make a local copy; you then synchronize the copy with the repository by ***pushing*** changes back, and ***pulling*** changes that have been made in other local copies (by yourself or your lab partner). Your local copy has revision control features of its own - you ***add*** files to put them under Git control, and ***commit*** changes to files. Only committed changes can be pushed to the repository.

## Specifics for CS 7600

All repositories for CS 7600 will be on the CCIS GitHub server, github.ccs.neu.edu, which you access with your CCIS username and password. This runs GitHub Enterprise, so if you are familiar with using GitHub.com you should have few problems with it.

## Git Commands

```
git clone https//myid@github.ccs.neu.edu:myid-hw1
```

- make a local copy of your hw1 repository.

```
git add file
```

- add file to the repository

```
git commit -m 'message' -a
```

- commit changes in the local copy. Git forces you to add a commit message like this. (if you don't supply a commit message it will start an editor and ask you for one)

```
git push
```

- copy local commits to the repository

```
git pull
```

- get changes from the repository to your local copy.

```
git status
```

- list locally changed files, etc.

If you've used subversion (SVN) before, this is mostly the same, except that there's an extra local step - rather than just committing changes to the repository, you commit them locally and then push the most recently committed change to the repository. It doesn't make much of a difference if you're working on your own, but if you're working on a team it means that you can commit frequently (so you can retrieve an earlier copy if you mess up) but only push changes to the repository when you're willing to let your teammate see it.

## Git Example

Here is the sequence of operations if Alice and Bob work together, with a repository starting out with 2 files, "file1" and "file2". Note that in real life things will probably be much more complex. For instance Alice may work on two different machines, and so she would clone the repository on each machine, and then push her changes back to the repository when she's done working on one machine, and pull changes when she sits down at the other machine.

| Alice | Alice's local copy | Repository | Bob's local copy | Bob |
|---|---|---|---|---|
| | | file1, file2 | | |
| `git clone ...:team-1-hw1` | | . | | |

| | | | | |
|---|---|---|---|---|
| | file1, file2 | . | | `git clone ...:team-1-hw1` |
| | . | . | file1, file2 | |
| <create file3> | . | . | . | |
| `git add file3` | . | . | . | |
| `git commit -m 'add file3' -a` | file1, file2, file3 | . . | . . | |
| `git push` | . . | file1, file2, file3 | . . | |
| | . . | . . | file1, file2, file3 | `git pull` |
| | . . | . . | file1, *file2'*, file3 | Edit file2 (into f*ile2'*) |
| | . | . | . | `git commit -m 'edit file2' -a` |
| | . . | file1, *file2'*, file3 | . . | `git push` |
| `git pull` | file1, *file2'*, file3 | . . | . . | |

At this point if Alice and Bob are finished, they should double-check that all their changes are pushed and all their tests work, and they're done. The instructor can do a 'git clone' of their repository, giving access to the latest copy as well as all the intermediate versions.