15-210: Git Handout System

Last Revised: August 31, 2011

1 Introduction

We will handout homework assignments in this class through a read-only `git` repository. This will let us smoothly distribute starter code to you, including the evolutions of the libraries we'll be using for this class. Since `git` supports fully distributed version control, this will also let you take advantage of version control to manage the iterations of your solution.

`git` is available on the UNIX timeshares and for modern operating systems, so you can choose to work on your local machine or remotely on `unix.andrew.cmu.edu`. You should feel free to work on your assignments locally, but we strongly suggest you work remotely on `unix.andrew.cmu.edu`. We will grade your assignments with the version of SML installed there, and we know that the version of `git` installed there works.

`git` has a huge user base, so it's easy to find out information about it. We're only using a small subset of the features it offers, so a lot of that information will not be relevant. That said, the following are good places to learn more about both `git` and version control in general.

- `http://en.wikipedia.org/wiki/Version_control`
- `http://en.wikipedia.org/wiki/Git_(software)`
- `http://git-scm.com/`
- `http://git-scm.com/documentation`

2 Using Our `git` Repository

2.1 Initial Check Out

To check out the repository initially, change to the directory that you'd like to have your work in and run

```
git clone http://www.cs.cmu.edu/~15210/resources/dist.git 15210
```

This will create a subdirectory called `15210`. You can replace `15210` with a directory name of your choosing.

2.2 Updates

To update your copy of the repository, change into the `15210` directory created by running `git clone` and run

```
git pull
```

This command is how you will receive new assignments, solutions to previous assignments, and possible corrections to existing assignments.

2.3 Local Commits

You can make local commits by running[1]

```
git commit -a
```

The `commit` command preserves the state your working copy, so you could come back to it later if need arises. Therefore, you should make a commit whenever the collection of files you're working on reaches a state that you think you might want to get back to later. Since you'll be working in a strictly local repository, there are no other developers to complain if you commit code that doesn't compile, so: commit early, commit often.

`git` will ask you for a commit message each time you commit. It's tempting to leave this blank or trivialize it, but you should avoid the urge. Meaningful commit messages are invaluable when you're going through previous revisions looking for a working bit of code that you lost.

When you create new files, `git` doesn't know about them. If you want them under version control, you need to add them manually by running

```
git add file.sml
```

In general, the updates you pull for each new assignment will have empty files with appropriate names; these are already under version control and you won't need to add them again.

It is important to note that since this will be a strictly local repository, if you delete it or your computer crashes *you will lose all your work*. There is no server that you're pushing your commits to. This is not a substitute for conventional backup.

2.4 Assignment Submission

You will *not* be using `git` to submit your solutions. Each assignment will have a list of files that you should submit to your handin directory on AFS. The path to that directory is, generally

`/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn<num>/`

[1]The option `-a` tells `git` to commit changes in all the files that have been modified or deleted. Without this option, `git` expects a list of files to commit.

Each assignment will be paired with a check script that will make sure you submitted the right number of files with correct names. These check scripts will be

`/afs/andrew.cmu.edu/course/15/210/bin/checks/<num>.sh`

These are *not* grading scripts and do not check that your submission compiles. For example, if you submit correctly named files that contain syntactically incorrect code, you would not receive credit just because the check script ran cleanly.