1 Git Setup

You will be using git for accessing all course material as well as for project submissions:

- All course material is maintained on a git repository hosted on a CS department machine. You will be mirroring this repository on your VM.

  You should access course files only via this mirror. **DO NOT DOWNLOAD MATERIAL DIRECTLY FROM THE WEB SITE**.

- You will be submitting your course projects using a private git repository on github.com.
- Project submission will be done from a clone of your github repository on your VM.

This document assumes that you have already *set up your VM*.

This document provides step-by-step directions for mirroring the course repository and setting up your github repository. You can obtain basic familiarity with `git` by looking at the copious documentation and tutorials available on the web. This presentation is recommended.

1.1 Mirror Course Git Repository on your VM

1. On your VM, create a public-private key-pair:
   ```
   $ ssh-keygen
   ```

   If you accept the defaults, you will create a private key in `~/.ssh/id_rsa` and the corresponding public key in `~/.ssh/id_rsa.pub`. The former should be retained on your VM whereas the latter may be distributed to machines to which you would like to access from your VM.

2. Copy the public member of the above key-pair to `remote.cs`:

   `$ ssh-copy-id -i ~/.ssh/id_rsa` *LOGIN*`@remote.cs.binghamton.edu`

   where *LOGIN* is your login-id on `remote.cs`.

   You will be prompted for your password on `remote.cs`.

   Verify that you can ssh from your VM to `remote.cs` without having to provide your password.

3. Create a `~/projects` top-level directory to hold all your cloned repositories:
   ```
   $ mkdir -p ~/projects
   ```

```
$ cd ~/projects
```

4. Clone the course web site:

   ```
   $ git clone LOGIN@remote.cs.binghamton.edu:~umrigar/git-repos/cs544.git
   ```

   where *LOGIN* is your login-id on `remote.cs`. All the files contained within the course web site should be copied into a `cs544` sub-directory.

   If you need to type in your `remote.cs` password, you have not set up your ssh key correctly. Please review the earlier step to see where you may have gone wrong.

5. Go into the `cs544` directory. You should then see all the course web site files.

6. Create a symlink to the `cs544` directory in your home directory:

   ```
   $ cd
   $ ln -s projects/cs544 .
   ```

   This will ensure that scripts can find the directory.

You should never be writing into this directory.

1.1.1 Automatically Tracking cs544 Changes

You should set up a cron job to update your `~/cs544` directory from the course git repository:

```
$ EDITOR=YOUR_FAVORITE_EDITOR crontab -e
```

where `YOUR_FAVORITE_EDITOR` is the command you use to start your preferred editor. It should open up showing you an initial `crontab` file. Add the following line at the end:

```
BB * * * * cd ~/cs544; git pull > /dev/null
```

where `BB` is your B-number modulo 60. Ensure that you have a newline at the end of this line. Save the file and exit your editor.

The cron job should update your repository every hour at `BB` minutes past the hour (by specifying the minutes using part of your B-number, we ensure that students do not all update at the same time, potentially causing an overload).

You should treat this directory as a **read-only directory**.

1.1.2 Manually Tracking cs544 Changes

If necessary, you can also manually update your `~/cs544` directory:

```
$ cd ~/cs544
$ git pull
```

You can get a summary of all git changes by running a Ruby script:

```
$ cd ~/cs544
$ bin/git-changes.rb .
```

This will output a summary of all git changes since the last time you ran the script (it records the time via a timestamp stored in `~/cs544/.last-login`).

If you want to see all the commits in a particular directory like `hws/hw1`:

```
$ cd ~/cs544
$ git log --oneline -- hws/hw1
dcdef92 minor hw1 changes
05fc1a1 added hw1; minor correction to prj1
$
```

Your commit id's will differ.

If you want to see the details of a particular commit, use:

```
$ git log --stat dcdef92^!
```

To see the diff's for a particular commit-id:

```
$ git show dcdef92
```

To restrict the diff's to a particular path:

```
$ git show dcdef92 -- hws/hw1/hw1.umt
```

1.2 Setting Up Your Github Repository

We will also be using git for project submissions:

- You will be submitting your course projects to a private git repository on github.com which you share with the TA.
- Project and lab submissions will be done from a clone of your github repository on your VM.

1.2.1 Quick Start

If you know what you are doing, this `tl;dr` section should largely suffice:

1. Set up a personal github account and create a `i444` or `i544` git repository, depending on the course you are registered for. Choose the github options to generate a `.gitignore` for `node` and initialize your repository with a `README`.
2. Set up the TA to have access to your repository:
   (a) Github user id `necatianil`

   As there are many similar names and some people may have multiple accounts, **make 100% sure that you have the correct github accounts** (the above *ID* must match).
3. Clone your github repository into the `~/projects` directory on your workstation. Create a symlink to your cloned repository in your home directory.
4. Use a `submit` subdirectory within the cloned directory for working on projects. If submitting late, email the TA for that project.

The following sections explain each of the above steps in detail.

1.3 Setting Up A Git Hub Repository

Each student needs to set up a **private** git repository called `i444` or `i544` (depending on the course you are registered for) on github. It will be referred to as *iX44* in the sequel. This repository needs to be shared with your CS 444/544 TA and will be used for submitting projects for grading.

1. Go to github.com.
2. Click on `Sign Up`, and complete the form. You may choose any appropriate **Username**, but it is strongly recommended that you use the user-name associated with your `binghamton.edu` email (if it is available).

Please ensure that you provide your `binghamton.edu` email address in the **Email address** field. This may make it slightly easier for the TA to associate your github account with your BU id.

3. Please complete all necessary steps to complete your registration on github after satisfying their captcha and submitting the form.

4. Set up your account appropriately. On the github page, access your account using the user icon on the top right of the page and selecting *Settings*. Select **SSH and GPG keys" from the left hand menu, specify a** New SSH key".

   On your terminal `cat ~/.ssh/id_rsa.pub` and then copy-and-paste the output of that command into the provided **Key** box. Submit the form to set up your new key.

5. To set up a project for the course, click on the green **New** button on your *github home page*.

   (a) Fill in the repository name as `i444` or `i544` exactly (this is absolutely essential).

   (b) Provide a suitable description for the repository.

   (c) Make sure that you set up your repository as **Private**.

   (d) Select `Initialize this repository with a README`.

   (e) Specify a `.gitignore` file for `node`.

   Hit the **Create repository** button.

6. Go to your repository using the link you will now find on your github homepage. Go to `Settings->Collaborators` from the left-hand side navigation. Add the TA as a collaborator:

   (a) Github user id `necatianil`

   As there are many similar names and some people may have multiple accounts, **make 100% sure that you have the correct github accounts** (the above *ID* must match).

   This will provide the TA access to your repository once the invitation to collaborate is accepted.

1.3.1 Initializing Your Repository on your Workstation

Clone your github repository into your `~/projects` directory:

```
$ mkdir -p ~/projects
$ cd ~/projects
```

```
$ git clone YOUR_GITHUB_PROJECT_URL
```

You can copy and paste `YOUR_GITHUB_PROJECT_URL` from your github project page. Click the green **Clone or download** button and then copy the **Clone with SSH** url into your clipboard by using the copy widget on the right of the url.

If you get an error when doing so, verify that you have correctly uploaded your public ssh key to github.

Create a symlink to your github project in your home directory:

```
$ cd ~
$ ln -s ~/projects/i?44 .
```

If you `cd` over to your `iX44` directory and do a `ls -a` you should see both the `README.md` and `.gitignore` files. Feel free to add names/patterns into `.gitignore` file for files which should be ignored by git. For example, if you are using emacs as your text editor, you may want to add in a line containing `*~` to tell git to ignore emacs backup files.

Make sure that all updates have been pushed over to github:

```
$ cd ~/i?44
$ git status -s #see if there are changes you want to commit
$ git commit -a -m 'SOME COMMIT MSG' #commit if necessary
$ git push #push changes to github
```

1.3.2 Working On and Submitting Projects

A common git workflow is to work on new program features in separate git branches and merge each feature branch in to the main `master` branch when the feature is complete. This course will expose you to this workflow by regarding each lab and project as a separate feature and using a separate branch for developing each lab or project.

The following lists the steps necessary to work on your first project `prj-1`. You will need to adapt those steps for subsequent projects. The git commit comments specified by the `-m` option are only examples; you may replace them with more suitable comments.

(a) Create a branch and a new directory for working on your project:

```
$ cd ~/i?44
$ git checkout -b prj1-sol #create a new branch
$ git branch -l #show branches with '*' next to current branch
                #which should be prj1-sol
```

```
$ mkdir -p submit/prj1-sol #make dir for project
```

(b) We will be using `npm` to package our projects together:

```
$ cd submit/prj1-sol
$ npm init -y #create a package.json with default values
$ git add package.json #add package.json to git staging area
$ git commit -m 'started prj1' #commit locally
$ git push -u origin prj1-sol #push branch with changes
                               #to github
```

(c) Copy over and commit start up files which have been provided for the project:

```
$ cp -pr ~/cs544/projects/prj1/prj1-sol/* .
$ cp -pr ~/cs544/projects/prj1/prj1-sol/.* .
$ git commit -a -m 'provided files'
$ git push
```

(d) If your project requires external js modules:

```
$ npm install MODULE...
$ git commit -a -m 'added dependencies'
$ git push
```

This will create a `node_modules` directory and a `package-lock.json` file. The latter file should be committed to git, but the `node_modules` directory should not be. If you have used the `.gitignore` file provided by github, that directory should be ignored by git.

(e) Work on your project. You should get into the habit of committing and pushing frequently, making it easy for you to recover from unsuccessful changes or a crashed VM. Some git commands which may prove useful:

- List local branches using `git branch -l`. The current branch will have an asterisk next to it.
- Switch the current branch to branch `B` using `git checkout B`.
- Add new files, directories and changes to the git staging area using `git add`.

```
$ git add *.js   #add all js files in current dir
$ git add .      #all all files, dirs, changes in
                 #current dir to git staging area
```

- Commit changes locally.

```
$ git commit -m 'MSG'     #commit staging area
```

```
                                    #with message MSG
$ git commit -a -m 'MSG'            #commit staging area
                                    #and all changes
```

- Push all committed local changes to remote github repository using `git push`.

(f) Once your project is complete, move it over to the master branch and submit it by pushing it to github:

```
$ git checkout master #go to master branch
$ git status -s #should show no outstanding changes;
                #otherwise add and commit till git status
                #shows no outstanding changes
$ git checkout prj1-sol #go to project branch
$ git status -s #should show no outstanding changes;
                #otherwise add and commit till git status
                #shows no outstanding changes
$ git merge master #merge any changes from master.
                   #should not cause any conflicts
                   #since all and only prj1 changes are
                   #in this branch
$ git commit -m 'merge master' #commit
$ git push                     #and push changes
$ git checkout master #back to master branch
$ git merge prj1-sol #should be a fast-foward merge
$ git commit -m 'merge prj1-sol'
$ git push  #submit project
```

You should use the github web interface to verify that the project has been submitted correctly to github.

If you are sufficiently paranoid (in general, you should be paranoid when working with computers :-( ), you will verify that it is possible to run your project using only what you submitted to github:

```
$ cd ~/tmp
$ git clone YOUR_GITHUB_PROJECT_URL
$ cd i?44/submit/prj1-sol
$ npm ci #build your project
```

You can now test your project. If everything is ok:

```
$ cd ~/tmp
$ rm -rf i?44  #rm cloned project
```

If you discover errors in your project after your initial submission, you can resubmit; the project submission time will be the time of the last submit.

(g) Once you are happy with your project submission, you can clean up the project branch:

```
$ git push -d origin prj1-sol  #rm remote branch
$ git branch -d prj1-sol       #rm local branch
```

If you submit late, please email the TA.

1.4 References

Official Git Site.

Scott Chacon, Ben Straub, *Pro Git*.

Reference Documentation

Anish Athalye, *Version Control (Git)*; Part of *MIT Missing Semester*; Highly recommended.