

Class 4 Handout

Table of Contents

Introduction.....	1
Source Control: An Absolute Must.....	2
What is Source Control?.....	2
First, think of it as storage.....	2
Second, Source Control will track your changes over time.	2
Types of Source Control & Hosting.....	2
Using BitBucket with Mercurial.....	3
How does Git and SVN differ from Mercurial?.....	10
In Closing.....	11
See Also.....	12
C++.....	12
Source Control.....	12
Source Control Hosting.....	12
Source Control Applications.....	12

Introduction

So far in this class, we have covered:

- Classes
- Functions
- If Statements
- While and For Loops
- Arrays
- File Input and Output

If you're interested in learning more about C++, here are some of the next topics you'll want to learn:

- Inheritance
- Pointers & References
- Polymorphism
- Templates
- Virtual Members and Interfaces
- Data Structures
- Design Patterns

This class will be more exploratory than tackling somewhat difficult topics like Pointers and Polymorphism. If you want to keep learning C++ after this class is over, check the last page of this handout for links to more resources.

Source Control: An Absolute Must

Source Control, otherwise known as Revision Control or Version Control, is something that all programmers must be familiar with.

Whether you're working alone or in teams, on small projects or large ones, source control is a boon.

Once you get the hang of it, it will be hard to not want to set it up before beginning a new project. I would also suggest getting used to it while you're in school, or if you plan on working on various projects!

What is Source Control?

Source Control is a type of programming tool.

First, think of it as storage.

You can store all of your source code in one central place (whether on your harddrive or on a server like BitBucket). The way this is set up, it also makes it really easy to pull the code from that server to any directory on your computer, on any computer.

This could be handy so you can keep your old projects and homework saved somewhere on a server, nice and organized, so you don't lose it, whether from a harddrive failure or just losing the code somewhere after years.

Second, Source Control will track your changes over time.

When you make changes to a file, you can view a *diff* to see what text was added or removed, you can see when new files were added, etc. This also allows you to go back to previous commits if you ever decide what you're working on isn't going to work out. You can also *branch* your repository, so you have the core working set, and a "new feature" branch that you can work on without breaking the core code, then *merge* it in later.

Types of Source Control & Hosting

There are different source control tools out there. The most common are probably SVN (Subversion), Git, Mercurial (hg), and Team Foundation Server (TFS).

There are also different source control hosting solutions out there, including many free ones. Some websites- like GitHub, SourceForge, BitBucket and Google Code- will give you free hosting for your Open Source projects. This means that your code would be public and visible to everybody, and that other people can check out your code and add on to it.

These guides are hosted on my GitHub account!

You can also get private hosting through GitHub, but it costs money. However, BitBucket will give small teams (under 5 people) *free private hosting*, so that is what we will be using.

Additionally, GitHub only allows Git, while BitBucket and the others allow other forms of Source Control. One of the huge draws of GitHub is its focus – Social Coding. Lots of people use GitHub, lots of companies use GitHub, and lots of recruiters check GitHub (There is even a job posting section to it!)

I've had projects on Google Code, but nobody ever pokes it. In my experience, Google Code is relatively dead when it comes to community, but GitHub (and SourceForge) are more active.

Anyway, let's get things set up!

Using BitBucket with Mercurial

We're going to be using BitBucket with Mercurial – BitBucket because I'm assuming you'll want privacy (and private repos are free here), and Mercurial because it's a bit harder than SVN, and a bit easier than Git.

Additionally, we'll be working from the Command Line, but there are also GUI programs you can use for SVN, Mercurial, and Git. (See: TortoiseHg, or programs like GitHub for Windows).

Register an Account with BitBucket

First, you will need to set up a BitBucket account. Head over to <https://bitbucket.org/> and register an account. You can also log in with OpenID, which means that you can use a Google, Yahoo, or other account to log in.

Create a Repository

With your private account, you can create as many Repositories as you want. You will generally create one Repository per project.

To create one, go to the Repositories dropdown menu on the top of the page, and select "Create Repository". Fill out the New Repository form:

- Repo name – Can be "CPP Class" or whatever you'd like
- Select "This is a private repository"
- Select Mercurial
- Select C++ as our language (This isn't required, but it helps)

And click "Create repository".

Also notice that we can have Issue Tracking and/or a Wiki. This would be helpful for logging bugs or a to-do list of tasks to work on for this project, as well as a Wiki for documentation.

Create a new repository

Name*

Description

Access level ☒ This is a private repository

Repository type ☐ Git
☒ Mercurial

Project management ☐ Issue tracking
☐ Wiki

Language

Then we will have a repository all set up and ready to go! The page says "Add some code", so select "I'm starting from scratch". It will then give you some command-line arguments to set up a repo on your computer:

```
$ mkdir /path/to/your/project  
$ cd /path/to/your/project  
$ hg clone https://Moosader@bitbucket.org/Moosader/cpp-class
```

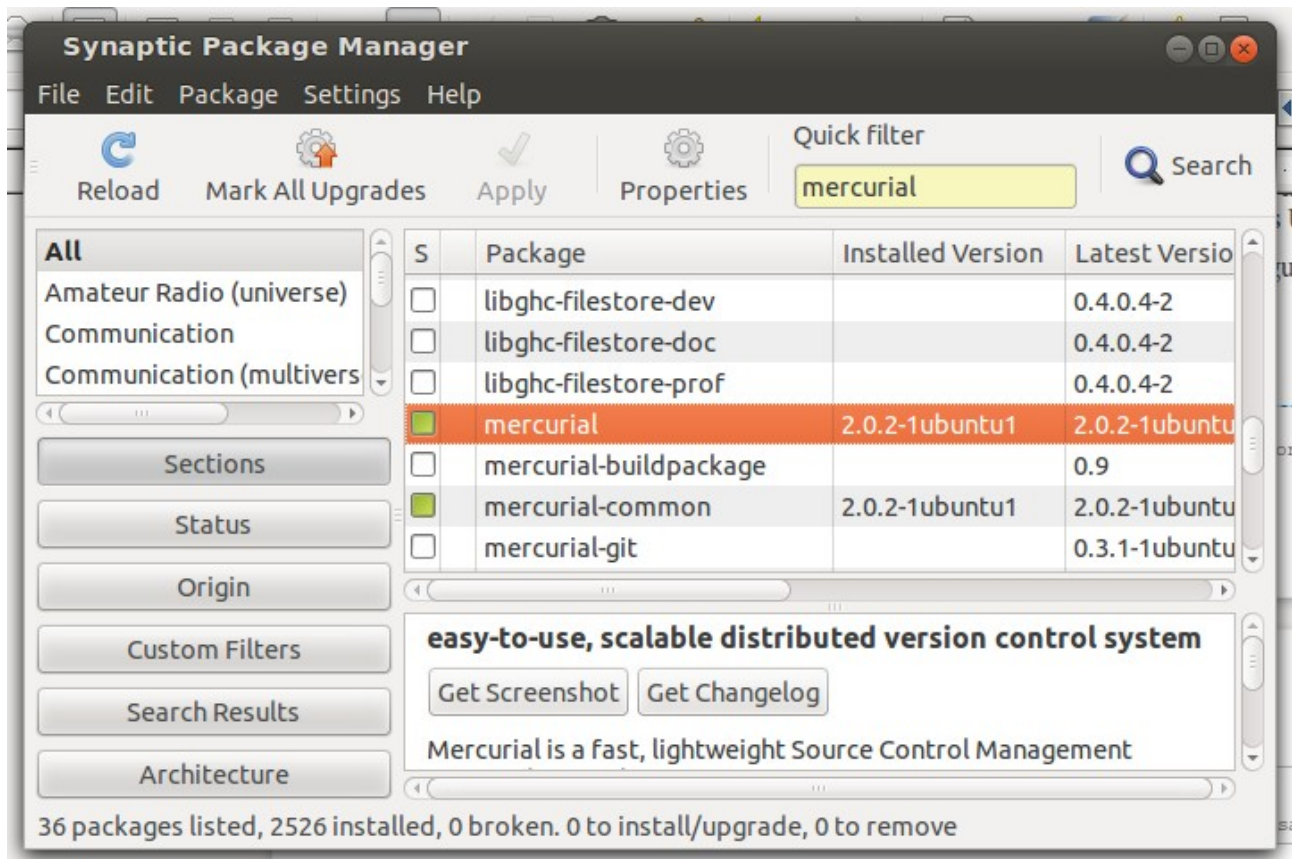
Really, you can ignore the first two lines. These are just to create a folder on your harddrive and navigate to it. The "clone" command listed is how you **clone** the repository from the Server to the Hard-drive. If you're using a GUI tool like TortoiseHg, you would need this URL.

There's also a link to "Bitbucket 101" if you need additional help after this guide.

Install Mercurial

Before we can clone our repository, though, we will need to install Mercurial.

If you're in Linux you can install it through a package manager, such as Synaptic.



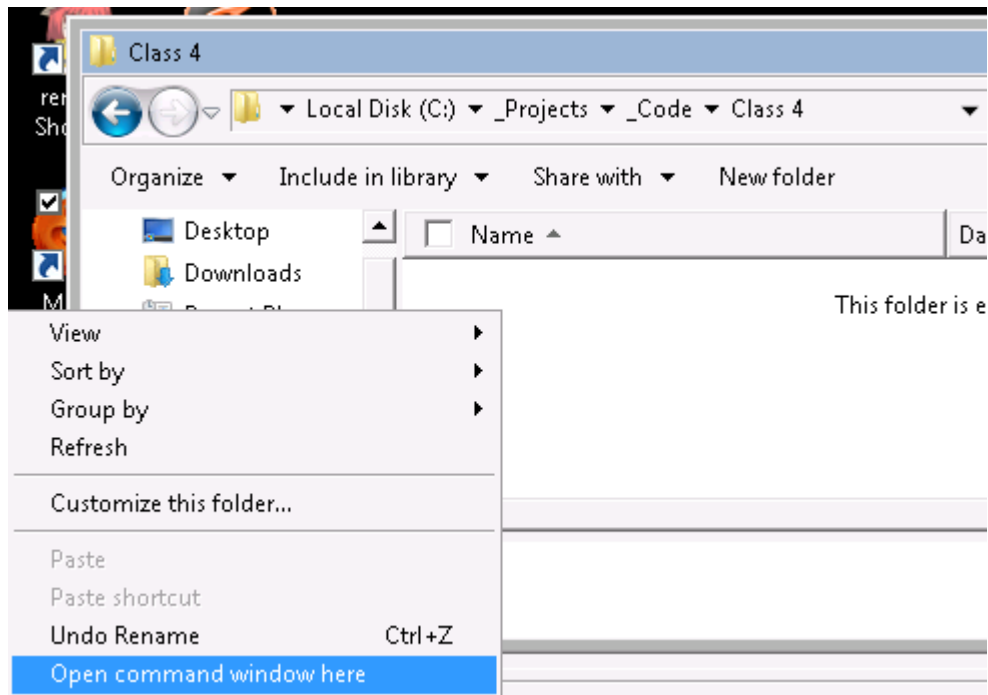
If you're running Windows, you can download Mercurial from here:

<http://mercurial.selenic.com/wiki/Download#Windows>

You can download the "TortoiseHg with Windows Explorer "shell" integration" version and install that. Now we can use Mercurial!

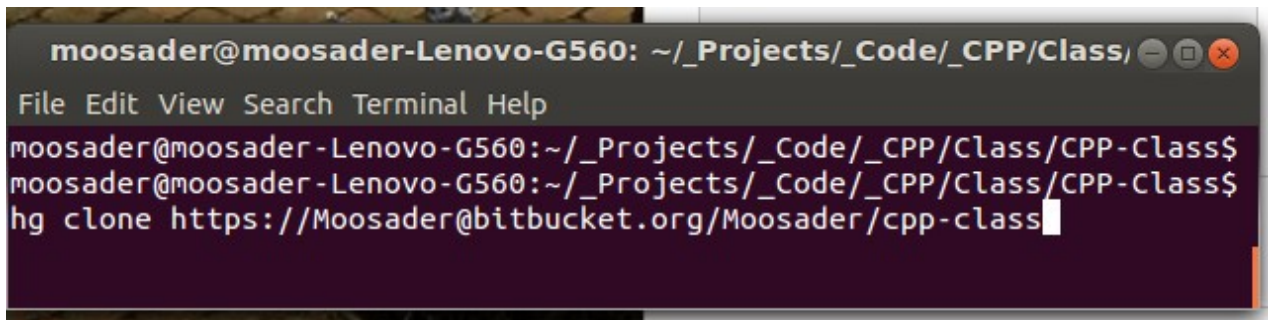
OK, create a folder on your harddrive and navigate to it. We want to open this in the Terminal or Command Prompt. If you're in Windows 7, hold down SHIFT and right-click in the folder. This will add an option to "Open command window here".

In Ubuntu, you might need to install **nautilus-open-terminal** in the package manager to do this (without shift, just right-click). Either way, open the Terminal.



So we should be in the command prompt, in the folder we're going to put your project in.

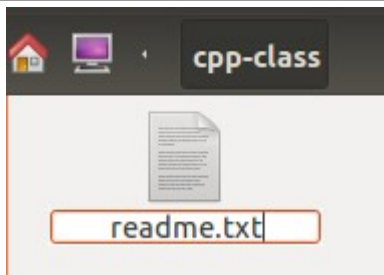
Copy the "hg clone" command that BitBucket gave us, then paste it in:



It might ask for a username and password, just use your BitBucket account to log in.

It will create our project folder. You can then go into that folder and create new files – Text files, source code files, etc.

Let's create a "Readme.txt" file, then edit it!



Open it up and add some basic text to the file, such as the project name and date.

Save the file, close it, and return to the Terminal.

To add this file to our repository, we will use **hg add**:

```
hg add readme.txt
```

This doesn't immediately add it to the repository – This is saying that this is a file we will want to track, and we will eventually commit. There are some files you might not want to add to source control, like a project file or our project's .exe file, so we only add what we need.

You can use * as a wildcard to add all files, or *.cpp to add all C++ source files. For now, we're just adding the readme.

Next, we'll commit the change:

```
hg commit
```

Abort Error

If you get an error message like "abort: no username supplied", we will have to set a username up in the config file. Type:

```
cd .hg
```

And hit enter. Then:

```
notepad hgrc
```

If you're in Linux, you might use "gedit" instead of "notepad", or a text editor available on your computer.

At the end of this file, type and save:

```
[ui]  
username = racheljmorris@gmail.com
```

Then close. Type:

```
cd ..
```

To go back to our project folder.

Try to commit again and see if it works properly.

Committing

The Commit command may open up notepad, or ask you what program to open with. **Nano** is a good choice, if you're in Linux.

Either way, once it opens up a text file, it expects you to enter a commit message, save, and close the file. In Nano, enter the file, then type "ctrl+x" to quit, and "y" to save.

Phew! OK, so really this shouldn't be that complicated if you're used to your operating system and text editors like Nano. It's a matter of:

- Adding a file (hg add)
- Committing your changes (hg commit)
- Adding a commit message (through nano or a text editor)

We can commit as many times as we want now, or undo changes to go back to the last commit. This commit is local to our own machine, though, and we have not affected the core repo on the server.

This is actually really handy, since you can commit code that *isn't quite working yet* without worrying about breaking the main project. You can commit as often as you want while you're working on something, undo commits, and work some more, and when you're finally ready to submit your changes, you will **push** the code...

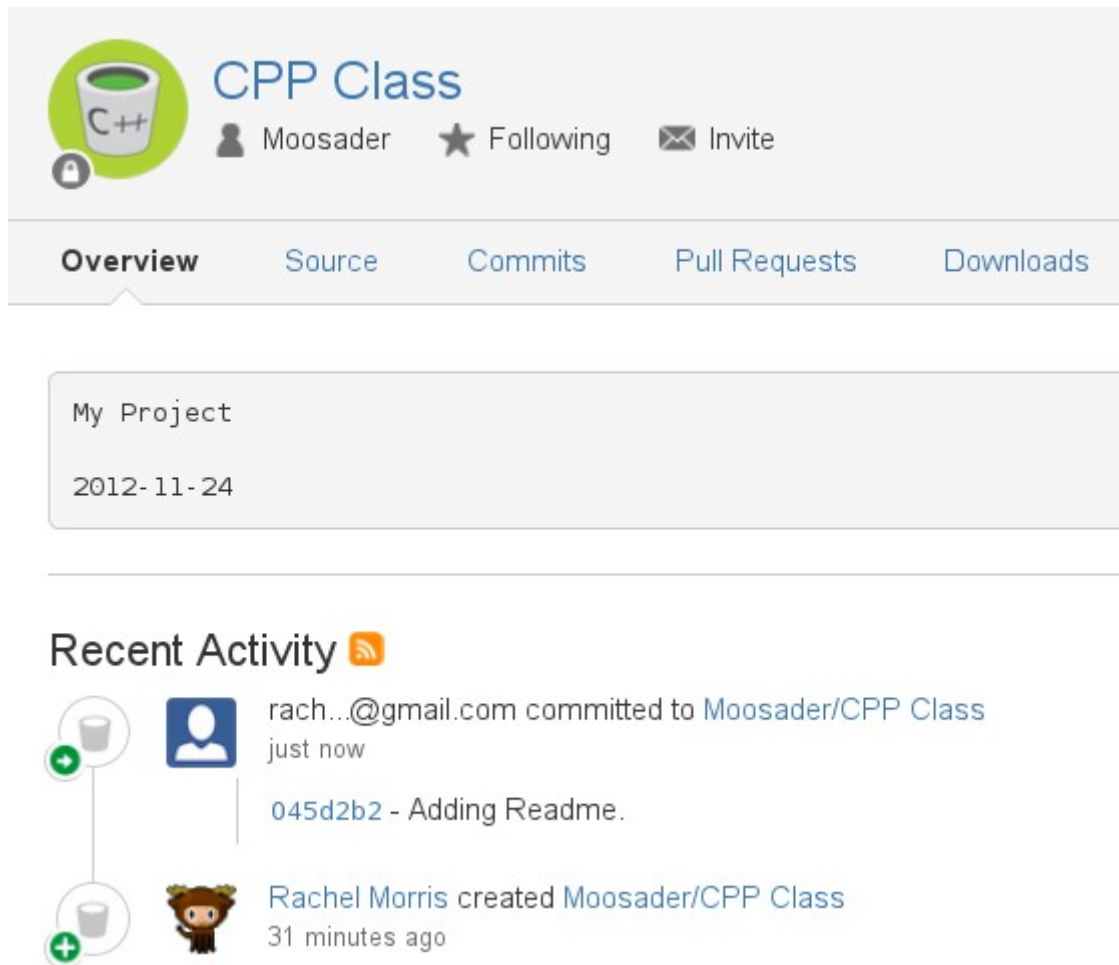
Pushing Changes

Once you've done the last commit of changes, we will need to **push** to get them to the server. We will use the command:

```
hg push
```

To do this. The terminal will output the status of the push as it goes, and might ask for the BitBucket password.

Now, if we go to our BitBucket page, we can see some new information!



The screenshot shows the GitHub interface for a repository named "CPP Class" owned by "Moosader". The repository is private, indicated by a lock icon. The user "Moosader" is following the repository. The navigation tabs are "Overview" (selected), "Source", "Commits", "Pull Requests", and "Downloads". Below the tabs, there is a section titled "My Project" with the date "2012-11-24". The "Recent Activity" section shows two events: 1. A commit by "rach...@gmail.com" to "Moosader/CPP Class" just now, with commit hash "045d2b2" and message "Adding Readme.". 2. A repository creation by "Rachel Morris" for "Moosader/CPP Class" 31 minutes ago.

Our **readme.txt** file now shows up with a grey background on the main page. This might be handy if this were a public repo and we wanted to show information about the project on the front page!

Next, we see **Recent Activity**, which shows that we just committed a new change.

Make sure to explore the different tabs in your repo page!

- **Source** – A web interface to show all the files in this repo. If you have source files in here, you can click their link to read the code from the web!
- **Commits** – Shows a list of commits that have been done in the past, who did 'em, and what the commit message was. This is why you should use detailed commit messages!
- **Downloads** – Here, you can upload an exe or zip file (or other types of files) for this project. This might be good if you want to upload different versions of this project.

hg push : hg pull, hg commit : hg update

Now, since you're working alone, there won't be much pulling or updating to do, but if you update your code from a different computer and check it in, or if someone else is working with you, then the *code on your computer will be out-of-date*.

No problem, though – This is part of the point of Source Control! Merging code from different users!

If someone has updated the code and you need to grab the changes, simply use:

```
hg pull
```

To pull down the latest changes. After that, to update the code on your machine, use:

```
hg update
```

To get things merged together.

Sometimes, you'll run into **merge conflicts**. This means that Mercurial was unable to figure out what code was changed in a specific case, and needs your help. This probably won't be an issue for you at first, but you can read more about **merging** online.

How does Git and SVN differ from Mercurial?

SVN is a bit simpler than Mercurial because you only have the **commit** and **update** operations – these check in and check out directly from the Server Repository. I dislike SVN because it doesn't give you the freedom to *commit anytime*; if you're working with others, there will be a fear of *breaking the build* because you've checked in incomplete code.

Git is a bit harder than Mercurial. Like Mercurial, you **commit, update, push, and pull**. However, **git add** acts differently than **hg add**, in that you have to add what has changed every new commit (Mercurial handles this automatically), and you have to explicitly specify which **branch** to **push** to. I think this verbosity is a bit confusing for anyone new to source control ("What are branches?" "What is this *origin master*?", "Why do I have to keep adding everything??")

Feel free to research SVN vs. Hg vs. Git on the internet, but if I were working on a project with others, I would prefer Mercurial (aka Hg) or Git to SVN (or even TFS).

I do have a GitHub at <https://www.github.com/Moosader>, so if you'd like to look at or modify my code, or share code with me, that would be a good place to do so.

That's about all I have to say about Source Control as an introduction. I really encourage you to try to use it regularly for your projects and get comfortable with it – *it really is invaluable*.

Finally, you don't **have** to use the Terminal (Command Line Interface) for source control. There are graphical programs out there to make it easier for you, and if you're using GitHub as your server, there is **GitHub for Windows** (and Mac!) to make things easier on you.

I* just prefer the Command Line because it feels more straightforward **to me.*

Since this is the last class, we are going to spend time reviewing concepts, while using source control.

Since BitBucket lets us have free private repositories for teams of 5 people and less, we are going to practice working together on simple projects.

The write-ups for these projects are in a separate text document, so please refer to those.

In Closing

Thank you for taking my class, or going through these handouts independently! If you have any comments or questions, please email me at

RachelJMorris@gmail.com

and I will get back to you whenever I can!

Thanks!

See Also...

C++

For more information about C++, check out some of these websites!

Learn C++

<http://www.learncpp.com/>

Source Control

Mercurial

<http://mercurial.selenic.com/>

Git

<http://git-scm.com/>

SVN

<http://subversion.tigris.org/>

Hg Init – Mercurial Tutorial

<http://www.hginit.com/>

Source Control Hosting

Google Code

<https://code.google.com/>

GitHub

<https://github.com/>

SourceForge

<http://sourceforge.net/>

BitBucket

<https://bitbucket.org/>

Source Control Applications

Tortoise Hg (Mercurial)

<http://tortoisehg.bitbucket.org/>

Tortoise Git

<http://code.google.com/p/tortoisegit/>

GitHub for Windows

<http://windows.github.com/>

Tortoise SVN

<http://tortoisesvn.net/>

Notice that these source control applications are hosted on **bitbucket**, **google code**, and **github!** :)