

Due: Thursday, February 16, 2017, 11:59:59 Midnight

Introduction

Today's lab is designed to give you a basic introduction to the Git version control system as well as the web hosting service BitBucket.

Lab Objectives

- Learn the concepts behind version control systems (VCS)
- Create a Git repository
- Learn how to commit files to your repository
- Connect your repository to an online hosting service

Prior to Lab

- Feel free to read the Git tutorial found here - <https://www.atlassian.com/git/tutorials>

Instructions

Version control systems are used throughout computing to make developing programs and projects more straightforward. Git is one popular example of a VCS which we will use today. Version control systems like Git are incredibly powerful tools with many nuances; today we will just scratch the surface of all the things a VCS can do.

Background

In a VCS, you create **repositories** which house your project. These repositories track changes in files and allow you to keep a living history of your project. These repositories also allow for multiple developers to work on the same code, and for code to be kept up to date on multiple machines. We'll now work through a step by step example of creating and maintaining a repository.

Lab Activity

Step 1 – Create a Repository

We'll begin by navigating to your course directory and creating a Lab5 folder.

```
cd cpssc1021 && mkdir Lab5 && cd Lab5
```

Note the “&&” operator in the command above. This allows you to chain Linux shell commands together as long as they all complete successfully. Now that we have our new directory, we'll tell Git to make it into a **repository**:

```
git init && ls -a
```

This will create a Git repository in the current directory and then list the contents of the directory. Notice the “.git/” folder that was created! Your Lab5 folder is now ready to use as a repository.

Step 2 – Basic Git setup

Now we need to do a little housekeeping before we start to play with our new repository. Run these commands to configure your username and email, as well as your preferred text editor: Note that there are two dashes in front of “global” with no space between them.

```
git config - -global user.name "Your Name"
```

```
git config - -global user.email "Your Email"
```

```
git config - -global core.editor vim
```

Step 3 – Add a file to your repository

Now download the **clock.cpp** program from Canvas. Move the file into your Lab5 repository and then compile the program with the following command:

```
g++ clock.cpp -o clock
```

This program is a simple CPP program that prints the current date and time to the screen, updating it every second. The most basic function of a Git repository is to track your files as you code. Every time you make changes to a file, you can save a snapshot of your work which you can revert back to at a later date. You can check which files are being tracked with the command:

```
git status
```

```
nick@nick-VirtualBox:~/cpsc1021/Lab5$ ls
clock  clock.cpp
nick@nick-VirtualBox:~/cpsc1021/Lab5$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        clock
        clock.cpp

nothing added to commit but untracked files present (use "git add" to track)
nick@nick-VirtualBox:~/cpsc1021/Lab5$
```

So our **clock.cpp** is currently not being tracked. Let's **add** it to the repository:

```
git add clock.cpp
```

```
nick@nick-VirtualBox:~/cpssc1021/Lab5$ git add clock.cpp
nick@nick-VirtualBox:~/cpssc1021/Lab5$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   clock.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    clock

nick@nick-VirtualBox:~/cpssc1021/Lab5$
```

Great! So now Git knows about our file. The next step is to **commit**, which is the step where we are saving a snapshot of our repository:

git commit

```
First commit!
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   clock.cpp
#
# Untracked files:
#   clock
#
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                1,14      All
```

Notice that your text editor was automatically opened! Every time you **commit**, you need to leave a short message describing what changes you made or what work you did. Once you've done that, we officially have a working Git repository!

Step 4 – Edit the clock program

The clock program is great, but it looks like the timestamp is missing some information. Specifically, we want it to print out the day of the week in the first <?> and the year in the second <?>. The function that turns the raw time into a formatted string is called **strftime()** and the description of how it works can be found here:

<http://man7.org/linux/man-pages/man3/strftime.3.html>

Check out the link and edit the program with the appropriate string format. The line you need to edit is **clock.cpp:20**. Once you’ve edited the program and confirmed that it works, we are going to add our changes and commit again:

```
git add clock.cpp
```

```
git commit -m “Fixed the time!”
```

Here we used the **-m** flag on commit to give a change message without having to open up a text editor. For short messages, this is a nice shortcut.

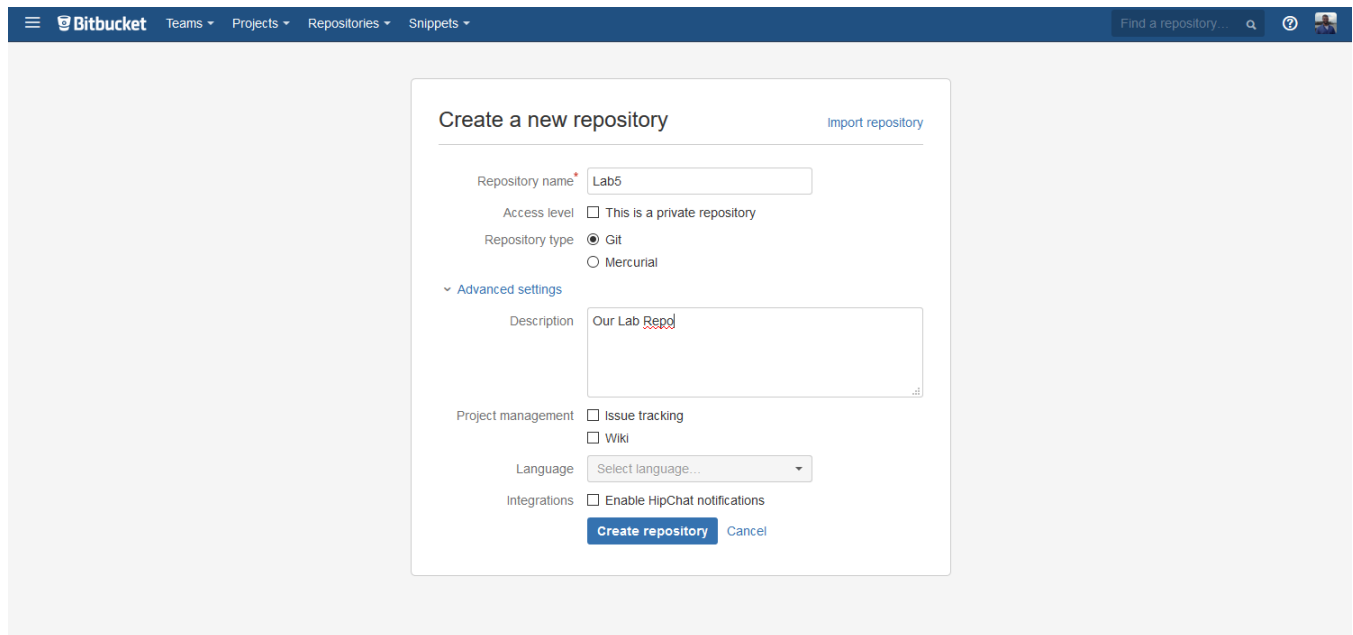
So now we have a working Git repo! Any time you make a change, **add** then **commit** your changes.

Step 5 – Hosting our repository on BitBucket

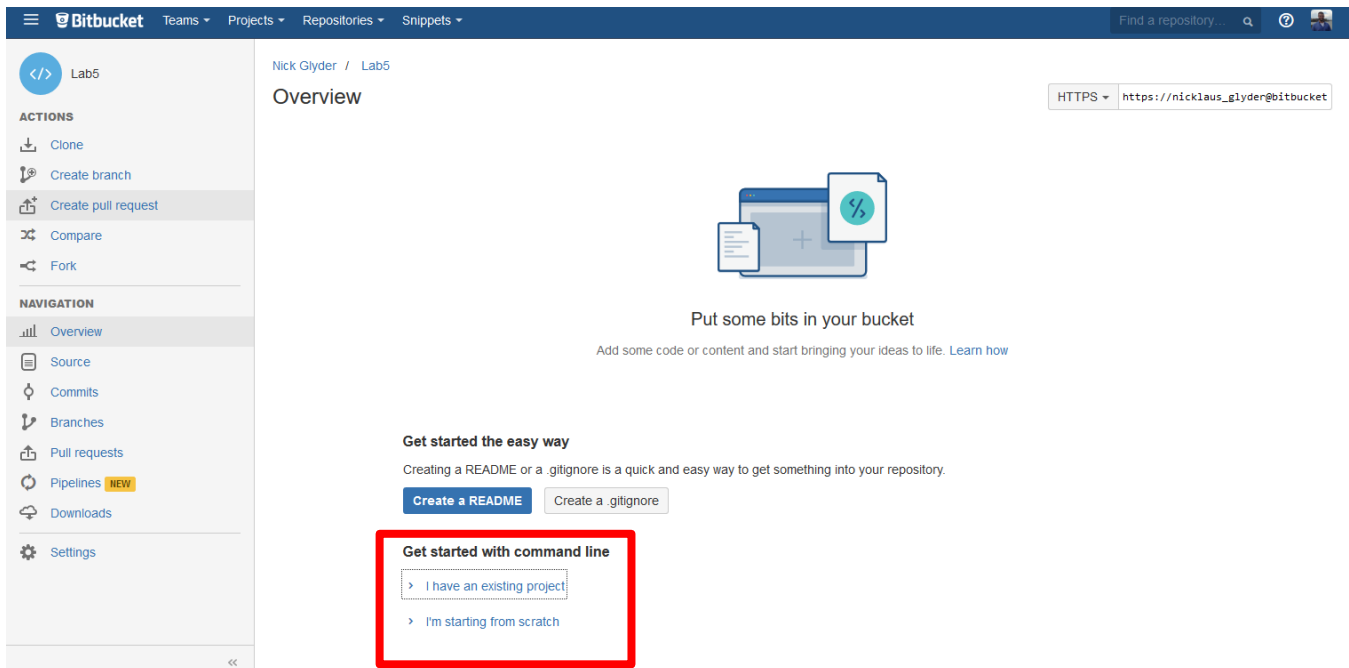
Go to www.bitbucket.org and create a free account. BitBucket allows you to host your repositories online instead of simply on one machine. This will allow you to **check-out** your repositories from any machine that has Git installed. This hosting is what allows teams of developers to work on the same code, or for you to share coding projects with other people (perspective employers?!).

For example, here is my BitBucket: https://bitbucket.org/nicklaus_glyder/

What we are going to do is **push** our repository out to BitBucket so that it is tracked online. Aside from the benefits of web hosting, BitBucket has lots of great tools for viewing your commit history, reverting back to previous versions of your repository, etc. To start, click on “Repositories” at the top of the screen and select “Create New Repository”.

The image shows a screenshot of the BitBucket web interface. At the top, there is a navigation bar with the BitBucket logo and links for Teams, Projects, Repositories, and Snippets. A search bar and user profile icon are on the right. The main content area displays a 'Create a new repository' form. The form has a title 'Create a new repository' and a link for 'Import repository'. It includes fields for 'Repository name' (filled with 'Lab5'), 'Access level' (checkbox for 'This is a private repository'), 'Repository type' (radio buttons for 'Git' and 'Mercurial', with 'Git' selected), and an 'Advanced settings' section. The 'Advanced settings' section contains a 'Description' field (filled with 'Our Lab Repo'), 'Project management' options (checkboxes for 'Issue tracking' and 'Wiki'), a 'Language' dropdown menu (showing 'Select language...'), and an 'Integrations' section with a checkbox for 'Enable HipChat notifications'. At the bottom of the form are two buttons: 'Create repository' and 'Cancel'.

Fill out the form and click “Create Repository”.



Click on “I have an existing project” and run the first command it gives you. It should look something like:

```
git remote add origin <your repo>
```

This is telling Git that there is a **remote** location where your repository originates from. Pretty self-explanatory! Our last step is to **push** our repository out to BitBucket. The command should be:

```
git push -u origin master
```

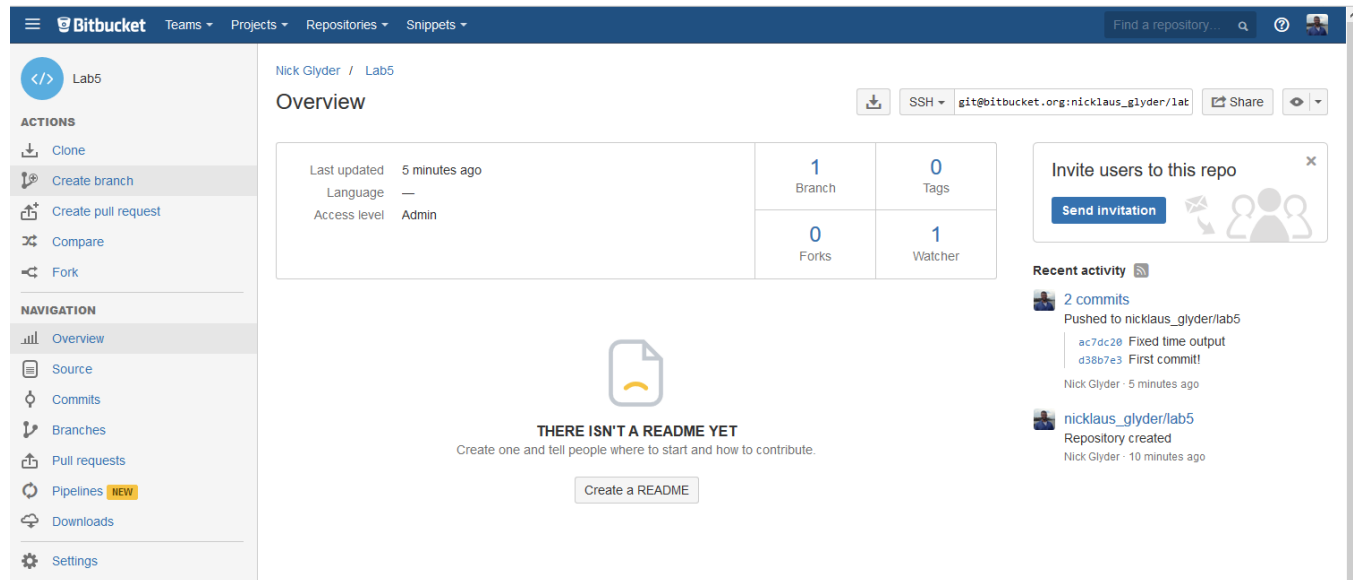
This command tells Git to send your repository out to the **origin** (BitBucket remote we just added) and to set it as the **master**. Really, it just means we are making our BitBucket repository the authority on our repository instead of the lab machine we started on.

```
nick@nick-VirtualBox:~/cpsc1021/Lab5$ git push --set-upstream origin master
Password for 'https://nicklaus_glyder@bitbucket.org':
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 819 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
To https://nicklaus_glyder@bitbucket.org/nicklaus_glyder/lab5.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
nick@nick-VirtualBox:~/cpsc1021/Lab5$
```

And that’s it! Now if you check BitBucket you’ll be able to view your repository online. Check out the “Commits” and “Source” tabs!

Step 6 – Adding a README and Pulling

You'll notice on the "Overview" tab there is a "Create a README" button. Go ahead and click that button and write a short README file, saving when you're done.



Now your repo has a README file, but this file only exists out on BitBucket. To make sure the changes are reflected on your lab machine, run the command:

```
git pull
```

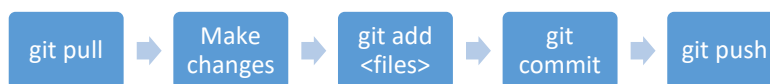
You'll see that Git **pulls** down the changes we made out on BitBucket. And with that, our tutorial is finished.

Cloning – Not like Dolly the sheep

One thing we didn't cover was how to get your repository on multiple computers! It's really simple once we are up and running on BitBucket. On the left menu bar, there is a **clone** button that will give you the exact Git command to copy your repository down to another computer.

Summary

So in summary, once you have a repository set up on BitBucket, the work flow you'll use is as follows:



Before we start working, we **pull** from the repository to make sure we are up to date. Then we work on our code. Once we are ready to save the changes, we **add** any files we changed, **commit** the changes, and **push** to send our repository out to BitBucket (or GitHub, or wherever you choose to host your repo).

Merging – A Word of Warning

Git is extremely powerful, and we barely scratched the surface of all the features. If you are interested, BitBucket has great documentation to help you learn some of the more advanced features. Branching, shelving, and multi-developer projects are used all the time in professional settings. Knowing about these features looks great on a resume!

However, you might have realized that it seems to be possible for two+ people to be working on the same repo... and not be in sync with each other! Not only could this lead to you working on code that is outdated, it is certainly possible for multiple developers to make changes to **the same file** and then try to commit their changes! Which version of the file should we use?

The short answer is; we need to **merge** the conflicting commits together somehow. There are many tools that accomplish this that are outside of the scope of this lab, and the topic of merging is a fairly complicated one. For now, just make sure you **pull** down your changes if you are working on multiple computers, and get ready to read a little documentation if you run into issues 😊

Submission Instructions

Please raise your hand when you are done and your lab TA will check out your work!