Goals

- Setup your class account.
- Learn `git` and setup your Bitbucket private repository.
- Build more intuition for working with binary.
- Get to know your classmates!

Reading

- P&H (4th or 5th): 2.4

Policies and Partners

You are **REQUIRED** to have a partner for lab checkoffs. This will reduce the number of check-offs we have to perform (allowing us to answer more of your questions) as well as give you someone to discuss class material with. **BOTH** partners will need to be present at check-off to receive credit and both partners will be asked to participate during the check-off. Try your best to find someone in your lab section with similar work habits as yourself.

How Checkoffs Work

You'll notice that at the end of (almost) every exercise, there is a section labelled "Check-off." The items in this section are what you must successfully demonstrate to your TA in order to receive credit for completing the lab. Once you and your partner finish **ALL of the exercises**, you should put your names AND logins on the checkoff list on the board, and a TA or Lab Assistant will come and check you off.

Labs in CS61C are graded out of 2 points. Labs are due for full points by the next lab session (which is 1 week after the lab was assigned). If they're another week late, then you get half credit. Any later than that and it's 0 points. You can always ask for help on the lab, but you can only asked to be checked off once. If you asked to be checked off and you don't pass the checkoff you'll get 0 points.

Exercises

In today's lab only, you should do exercises zero through two alone. You will need a partner for exercises three through five.

Exercise 0: Account and Environment Setup

To obtain your CS61C login, go to https://acropolis.cs.berkeley.edu/~account/webacct/ and login using your Calnet ID. Once you login, create a new account for CS61C. This should give you a username and a temporary password. Now, you can login to your instructional account by running the command `ssh cs61c-xxx@hiveYY.cs.berkeley.edu` on your laptop (where `xxx` is your 61c login, and YY is any number betwwen 1 and 29), and entering in your temporary password. Congratulations! You are now remotely accessing the hiveYY computer located in Soda 330. You can also login directly onto one of the lab computers with that username and password.

In order to change your password from the temporary one, while still logged into your instructional account (i.e. in the same terminal window that you ran `ssh cs61c-xxx@hiveYY.cs.berkeley.edu`), and enter `ssh cs61c-xxx@update.cs.berkeley.edu` and follow the prompts.

Now you're ready to start the lab!

Checkoff:

- Run `check-register` and show the results to your TA.

Exercise 1: Bitbucket Account Setup

Please read the following instructions carefully before proceeding. Almost all issues students run into during this lab can be prevented by carefully following the steps provided. Even if you have experience with `git` from previous 61 series classes, the process we use to set up your accounts may be different in this class.

This semester, we will be requiring that you use `git`, a distributed version control system. Version control systems are better tools for sharing code than emailing files, using flash drives, or even other file sharing mechanisms like Dropbox.

We'll be using Bitbucket to host private repositories in which you'll store your code. If the previous sentence means nothing to you, don't be alarmed! We'll walk through the process shortly. But first, you'll need to create a Bitbucket account if you don't already have one.

Why Bitbucket? Bitbucket allows accounts to have unlimited private repos. GitHub limits this to 5 per (student) user. Since many of you have already used those 5, we decided to use Bitbucket. If you have expereince using Github.com in previous classes, don't worry: Bitbucket interacts with `git` in the same way as Github.com, it is just a different website for hosting the remote (online) repositories.

Setting up Bitbucket and creating a Bitbucket repository

Navigate to bitbucket.org

- If you don’t have a bitbucket account, create a bitbucket account.
- Create a new **empty**, ***private*** repository called: `lab0_with_git`
- Navigate to your Bitbucket repo settings page
    - https://bitbucket.org/<bitbucket-login>/lab0_with_git
    - Settings → User and group access. Under "Users", you should see yourself listed as "owner"
    - Give 'cs61c-staff' admin access to your repo.
    - Leave this page open to show your TA at the end of the lab
- **Do not give access to anyone else to your repo.**

Setting up `git`

Now that we have created our repo, lets configure `git` so that it knows who you are. *While logged into your instructional account (from part 0),* Run the commands listed below, replacing `YOUR NAME` with your first and last name (inside quotes) and `YOUR EMAIL ADDRESS` with the email address you used to register for your Bitbucket account:

```
$ git config --global user.name "YOUR NAME"
$ git config --global user.email "YOUR EMAIL ADDRESS"
```

Checkoff:

- Show your TA the Access Management page for your private repo on Bitbucket

Exercise 2: `git`, Remotes, and the Hive machines

First, some quick defintions.

- A **remote** is the website host or server that will store your class code REMOTELY instead of only having your code LOCALLY on your own computer. You can think of it as a file store like Google Drive/Dropbox but armed with the powers of git.
- A **branch** is its own sequence of different changes to your code. You can think of branches as different versions of your code, that at one point were the same before branching from each other after one or multiple changes. These are very useful for always having a working/safe version of your code.

Throughout this class, you will regularly work with three different computers that may very well have three different versions of your code. These three are your local machine (your personal computer), one of the hive machines (while logged into your instructional account), and a remote (your Bitbucket repositories). For the least pain throughout the semester, it's essential that you understand the difference between these three and how you can share code between them.

1. Your local machine. Just your good ol' personal computer--nothing new here!
2. The hive machines. We'll be using a lot of different software and libraries throughout this class that might require different versions than the ones on your local machine (e.g. python2.7 versus python3). Therefore, you'll need to log into your instructional account on a hive machine (hiveYY.cs.berkeley.edu) so that you can run your assignment code in an environment with all of the correct software/library versions.
3. The Bitbucket remote. Your Bitbucket account serves a lot of purposes, but 2 of the most important are as 1) a way to share your code using git 2) a backup of your code, so that if something ever happens to your local machine, you can recover your code instead of having to start over! Conceptually, you can also think of the Bitbucket remote as another machine that only stores your project code (and doesn't do much else). You will push changes to Bitbucket (i.e. updating the files on Bitbucket) and also pull changes from Bitbucket (i.e. updating files on your local machine)

If you aren't already logged in, SSH onto the hive machine as in part 0. Next, obtain the files for this section.

Obtaining Lab Files

Copy the contents of ~cs61c/labs/00 to a suitable location in the home directory of your instructional account as follows.

```
$ mkdir labs (if ~/labs doesn't exist)
$ cp -r ~cs61c/labs/00/ ~/labs/00
```

You should be able to see the lab files if you navigate to ~/labs/00

To copy the lab files to your local machine, you have two options. 1) You can use `git` and Bitbucket, which you'll learn in the next section, and allow you to have version control over the changes you make to your lab code. 2) Using the **scp** command, which requires less steps but does not give you version control. Read below on how to use **scp** for future labs, but for this lab, we'll be using `git`. If you already know how to use **scp**, feel free to ignore the following.

Scp follows the same semantics as the **cp** command used above (**cp -r [host] [destination]**, -r being a flag to copy recursively all directories and files). Thus, one way to use **sc**p to copy files to your local machine is to navigate to a place on your local machine that you want to store your lab files, and run **scp -r cs61c-XXX@hiveXX.cs.berkeley.edu:~/labs/00 ./** (don't forget the "./" at the end for the destination argument). *You must run this command from your local machine; it will not work if you run this on the instructional machines.* If you then type ls, your lab folder should appear on your local machine.

Pushing to and pulling from Bitbucket

Now let's push some code to the Bitbucket "lab0_with_git" repo that you made earlier! Run the following list of commands.

```
git init # initalizes git to start tracking all changes within this repo (i.e directory and its subdirectories)
git remote add origin https://bitbucket.org/mybitbucketusername/lab0_with_git.git # Adds your Bitbucket as a remote to backup you
git status
```

After the last command, you should see some output in your terminal about untracked files.

Now let's commit your changes (i.e. all of these newly tracked files). Remember the following sequence of commands, you'll be using them regularly to commit changes to your code.

```
git add -A # stages all modified files for committing
git status # you should now see that files are staged
git commit -m "Commit message" # you can enter whatever you'd like for the message
git branch # you should see that only the master branch exists, and you're currently on it
git push origin master # this pushes your code to Bitbucket! You should now see it on Bitbucket.
```

Git's version control is built around commits, or checkpoints in development of different versions/stages of your code. To explain the above steps a little further:

- `git add [filename]` will tell git that you've made changes to that file and would like its changes to be saved in the next commit (AKA staging).
- `git commit -m "commit message"` officially saves the changes that you just added, and makes a snapshot of the current contents of all of the files in the repo. You'll now always have the option to revert your code to what it was at this commit.
- `git push origin master` pushes the current contents of your code on the branch "master" to the remote "origin" (remember that we added your Bitbucket repo for "lab0_with_git" as a remote named "origin"). This will create a branch on your remote repository (you should see a branches button

on the Bitbucket page for your repo) if it doesn't exist already, and save a copy of your recent commit on the remote repository.

When it comes to git, if you're ever unsure of something, but just want to make sure you have a saved copy of the current contents of your code, just run `git add` and then `git commit` .

Now that we've got your code stored on Bitbucket, let's get your code on your local machine and actually start modifying it. Open a terminal window on your local machine, and navigate to the directory where you'd like to store your labs on your local machines (e.g. "~/cs61c/labs"). Now run the following command.

```
git clone https://bitbucket.org/mybitbucketusername/lab0_with_git.git
```

Just like that, the lab0 files you had on your instructional account and on Bitbucket are now also on your local machine! Now, let's make some changes. Within the `lab0_with_git` directory, open the `hello.sh` file. Change `student_name` from "Oski" to your own name, and change line 14 from `ls` to `sl`. Then, try running the file with `./hello.sh`. You should see some greetings printed out, and a message that the `sl` command was not found. Not so interesting, huh? This is one of the situations where the hive machines have different things installed than on your local machine. Let's use git to get your newly changed code onto a hive machine, so that you can run `hello.sh` as expected! Run the following commands:

```
git add hello.sh
git commit -m "Write your own commit message"
git push origin master
```

Now open up the terminal window where you were logged onto a hive machine (back from part 0). Navigate to the directory where you originally copied the lab files, "~/labs/00", and then run:

```
git pull origin master
```

You may be presented with a new screen asking you to write a commit message. This is likely the built in text editor Vim. To start editing your message, first type `i`, write your commit message, and then type `:wq` to save your message and exit the editor.

This command fetches any new changes/updates to the branch "master" that have been stored on the remote "origin", and then merges them onto the current branch of your non-remote copy of your code (in this case on your instructional account). So just like that, the name and "sl" changes that you made earlier on your laptop, are now also present on your instructional account. Now, again run `./hello.sh`. Cool train, eh?

One last git command that you'll find useful is `git log` . Run this command, and you'll see a history of all the commits ever made to your code (on the current branch), including the time and who made the commit.

Checkoff:

- Run `git log` and show your TA the output.

Exercise 3: Working on Projects

This semester, you'll be working on projects individually and you may work on both the hive machine and your local machine. But how can you smoothly change code in the same files without having to delete and copy files back and forth? There is where Git will come to the rescue! In this part, you'll learn the process you will use for every project for obtaining the starter files for the project and then also working on the code on both machines. For the rest of this part, you will work on both your laptop and your instructional account. It will be easiest to open up multiple tabs in your terminal.

Setup: Getting the Project Files

First, let's make a Bitbucket repo to store your project code. Follow the steps from part 1 again in order to create a new Bitbucket repo named "lab0-xxx", where xxx is your 61c login.

Now navigate to the directory on the hive machine where you'd like to store your project code (e.g. "~/cs61c/projects" or "~/projects"), and run the following commands:

```
git clone https://bitbucket.org/mybitbucketusername/lab0-xxx.git
cd lab0-xxx
git remote add lab0-starter https://github.com/61c-teach/sp18-lab0.git
git fetch lab0-starter
git merge lab0-starter/master -m "Getting lab0 skeleton code"
git push origin master
```

Now switch to your local machine, set up a lab0-xxx on your local machine and pull the lab files.

```
git clone https://bitbucket.org/mybitbucketusername/lab0-xxx.git
cd lab0-xxx
git pull origin master
```

You should now have the starter code (`foo.txt` with content "`World Hello`") on both machines.

Now, let's pretend you are working on the project on your local machine. Open up `foo.txt` on your local machine, change the contents to "Hello World", and push your changes to your Bitbucket repo:

```
git add foo.txt
git commit -m "Change to Hello World"
git push origin master
```

You should be able to push to your Bitbucket repo successfully. At this time, foo.txt should have "Hello World" on your local machine, but "World Hello" on the hive machine.

Now, pretend that you are switching to work on your code on the hive machine, but forgot to pull the new changes. Switch to the hive machine and open up `foo.txt`. Change its content to your name and try to commit and push your changes to Bitbucket.

```
git add foo.txt
git commit -m "Change to my Name"
git push origin master
```

This time, your push command errors out because your Bitbucket repo contains work that you do not have on your hive machine repository:

```
To https://bitbucket.org/mybitbucketusername/lab0-xxx.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://bitbucket.org/mybitbucketusername/lab0-xxx.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

To fix this, let's pull and then re-push.

```
  git pull origin master
```

You should get "`Auto-merging fails`". What does it mean?

Merge Conflicts: When things goes wrong

In the previous section, you worked on the out-dated directory on the hive machine and tried to push the new changes without pulling the most recent code. When you try to pull code that conflicts with the code you are working on, this situation is called a **merge conflict** and poses a problem because `git` doesn't know which modifications it should accept for the conflicting line(s). Let's see how to resolve this.

If you open up `foo.txt`, you'll notice that some arrows and other numbers have been added by `git` to signify a merge conflict. The top half (before the "========") is the changes you made on hive machine, and the bottom half is the code from the remote Bitbucket repo (The change you pushed from your local machine). Resolve the conflict by deleting the ">>>>", "======", and the commit numbers (next to the arrows), and leaving the correct text (your name). You now need to re-add the once-conflicted file and commit it:

```
git add foo.txt
git commit -m "Resolved merge conflict"
git push origin master
git log
```

This time push should succeed. Take a look at the git log output and make sure you understand where each log comes from.

Finally, switch back to your local machine and pull the changes you just made:

```
0: git pull origin master
```

You should see your name in foo.txt on your local machine.

Checkoff:

- Go to the Bitbucket page for your repository and click on "Commits". Show your TA the list of commits.
- Explain how to resolve a merge conflict to your TA.

Exercise 4: Binary Alphabet

Let's take 4-bit numbers. If we stack five 4-bit numbers on top of each other in binary, we can make patterns and pictures! To help visualize this, you can think of 0's and 1's as white and black instead. For example, what does the following bit pattern look like?

```
0 1 1 0     □■■□
1 0 0 1     ■□□■
1 1 1 1 --> ■■■■
1 0 0 1     ■□□■
1 0 0 1     ■□□■
```

Checkoff:

- What five decimal digits produce the pattern above? What five hexadecimal digits?
- What letter is drawn with 1,1,9,9,6? 0xF8F88?
- What is the hexadecimal representation you would use to spell the letter 'B'? 'N' (you probably won't want to use 5 hex digits for this one)?

Exercise 5: 1,000 $1 Bills

I hand you a thousand $1 bills and ten envelopes. Your job is to find a way to put various numbers of dollar bills in those ten envelopes so that no matter what amount of money I ask you for (between $1-1000), you can simply hand me some combination of envelopes and always be assured of giving me the correct amount of cash.

Checkoff:

- Explain to your TA how to distribute the dollar bills in the ten envelopes.