Hack Remotely

Logging Onto The College Server

We are going to do all of our hacking on a **remote computer** installed somewhere on campus by logging into the remote computer from our **local** computer (laptop). This will allow us to avoid the hassle of installing on our laptops a code editor, a compiler, and a runtime environment to run the programs that we've written. More importantly, it will provide us a common system on which to learn.

To log into the remote computer, we are going to use the applications named **GitBash** (for Windows users) and **Terminal** (for MacBook users). From here on out, if I ask you to run/launch/start your **terminal** application I will be referring to GitBash and Terminal.

Ok, start up your terminal application.

Terminal applications run a program called a **shell**. There are many different types of shell programs. A commonly used shell is called **bash**. bash stands for *Bourne Again Shell*. It was written by Jason Bourne. No it wasn't. It was written by Brian Fox as a replacement for the Bourne shell written by Stephan Bourne. Hence the play on words, Bourne (born) again.

When a shell is running, it displays a **system prompt**. The appearance of the system prompt depends on the shell that you are using, but it usually ends with a symbol like $ or #.

We can enter **commands** at the shell's system prompt to run programs. When we press the *enter/return* key, the commands are executed (aka run) on the computer. In this class, the commands that we will run will include the *name of a program* followed by *data that we want to pass to the program*.

The **ssh** program is used to log into a remote computer. ssh stands for *secure shell*. We can only log into a remote computer if we know the remote computer's *name* (aka **host name**) or IP (internet protocol) address, and have access to a *user account* on the remote computer.

When we run ssh it will create an **encrypted connection** to the *remote computer* and provide us with a new *shell* and *command prompt* for the remote computer that we are logged onto.

When we run commands on the *remote computer's command prompt*, the commands are executed on the remote machine - not on our laptops.

To summarize, we use a **Terminal** application that runs a **shell** (e.g. bash) which gives us a **command prompt** to run commands. We'll run the **ssh** program from the command prompt to log onto a **remote computer,** and once connected will be given a new shell and command prompt for the remote machine, from which we can execute commands on the remote machine.

Terminal > shell (bash) > command prompt > ssh → shell > command prompt > commands
local computer (laptop) → remote computer

A **server** is a computer that *serves* data to the computers that connect to it. For example, a web server is a server that serves web pages to the computers (typically using a browser) that connect to it.

Here on campus, we have a server named **cs.bridgewater.edu**, that runs the **Linux** operating system. The Linux operating system was originally created by Linux Torvolds and is modeled after the UNIX operating system. It runs very similar to MacOS, and not at all similar to Windows. The Android operating system was build from a modified Linux **kernel**.

Each of us has an account on cs.bridgewater.edu. The username of your account is the same as your Bridgewater College username. Your BC username is the *part* of your email address *before* the @eagles.bridgewater.edu. For example, my BC username is *rmcgregor*.

To log into the cs.bridgewater.edu server we do the following.

1. In your terminal application, type the following command, *omitting* the $ and *replacing username* with your Bridgewater College username. Then press enter/return.

```
$ ssh username@cs.bridgewater.edu
```

2. You should see a prompt which asks you if you trust the connection. Type *yes* and press enter.

3. You will then be asked for your password. **Note that when you enter your password the cursor will not move - it appears as if you are not typing, but you are.** Enter your Bridgewater College password and press enter.

If successful, you will be connected to cs.bridgewater.edu and you should see a new system prompt. Now, all the commands you enter on the command prompt are executed on cs.bridgewater.edu.

Hack away!

Set Up Git on cs.bridgewater.edu

Earlier you installed Git on your laptops. Git is *already* installed on cs.bridgewater.edu.

Before we use Git on cs.bridgewater.edu, we need to configure it.

Scott Chacon and Ben Straub have written a book titled Pro Git which discusses how to use Git. The book is released under the Creative Commons Attribution Non Commericial Share Alike 3.0 license. Below are instructions from chapter 1.6 Getting Started – First-Time Git Setup.

While logged onto cs.bridgewater.edu, follow the instructions below to inform Git of your identity, set a default branch name, and check your settings.

Your Identity

We can use Git to track changes to the code we write by issuing *git commit* commands (more on that later). In order to record who made changes we need to inform Git of our real name and email address.

Run the following commands replacing **John Doe** with your real name (keep the quotes), and replacing **johndoe@example.com** with your BC email address.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Your default branch name

Git allows multiple people to work on different *branches* of the same project. By default, when Git creates a project repository it creates a default branch named *master.* Enter the following command to change the default branch name to *main*.

```
$ git config --global init.defaultBranch main
```

Checking Your Settings

Enter the following command to view your current configuration settings.

```
$ git config --list
```