# Intro to Graduate Research at AFIT with Dr. Nykl

Scott Nykl

**Abstract**
How do I begin my research so I am efficient, effective, and productive with respect to my thesis and publications? What tools are available that can increase my productivity and ability to collaborate? This document aims to set forth expectations & advice and tips for successfully getting through your graduate studies. Disclaimer: This is the first iteration of this document, so it is in rough shape, but over time and with student feed back, I hope to incorporate applicable advice for my new students.

## 1. Expectations & Advice

Graduate school is a more self-directed environment than undergrad. A student's primary responsibilities include exploring a research area (related work / literature reviews), converging on a specific topic (what questions, and conducting and evaluating research on that topic. The advisor serves as a mentor and offers guidance throughout the process. If you ever feel "stuck" or lost, setup an appointment and speak with me. With that being said, several activities can greatly alleviate any anxiety that may be related to beginning a new research endeavor:

1. Read, read, and read papers! Create your own database of papers you have read. Keep their bibliographic information, the PDF, and a quick synopsis of the paper's important points within your own GIT repo. Once you find *one* good paper, search through its references and acquire / read / review those papers. Continue this process until the web of papers begins to connect back onto itself. You will begin to understand the context from which each paper was written, how that paper contributed to the state of the art, and how subsequent work built on top and further extended the envelope. This is how scientific progress is made - each paper (each researcher) slightly perturbs the *state-of-the-art* envelope expanding man kind's knowledge. Also, keeping a synopsis of these papers and their bibliographic information will *greatly* enhance your ability to quickly and thoroughly write Chapter 2 of your thesis (the related work chapter).

2. Survey papers are especially powerful for quickly acquires a broad view of the research domain. Survey papers typically have the word *survey* in their title and are sometimes longer than a normal paper. Do not be dismayed by the length, think of it as a deep well of domain-specific knowledge waiting to quench your mind's thirst (yeah, these puns will probably get worse with time). For a master's degree, a solid comprehension of 2 or 3 survey papers plus a few more narrow papers may be sufficient to hone in on a specific thesis topic that becomes your prospectus.

3. In case you were not aware, `http://scholar.google.com` is a power tool for finding papers, citation information, related papers, referenced papers, and reference counts of papers. In general, if a paper has been cited many times, it is more likely to be a seminal (highly important) paper and is likey worth a read.

## 2. Tips for Success

### 2.1 Read Papers

See Sec. 1, Item 1 - Read, read, and read papers.

### 2.2 GIT

GIT is an artifact versioning system that can be used to track textual (source code), as well as binary information. We will use this to store and synchronize our research, papers, source code, and other information. It will allow us to seamlessly have all of our data accessible to all of us at any time on any machine. It also offers a powerful log/versioning system that shows each change made by any user throughout the lifetime of any files and/or directories. It also supports more advanced features including branching and parallel development.

GIT is one of the most common distributed versioning systems in existence and is a very practical skill set to have acquired when working on any project involving software development or research. Other version control systems you may have heard of include Subversion, CVS, Mercurial, SourceSafe, Bazaar, etc. GitHub is a popular website that uses GIT to host software projects - our interface will be nearly identical to GitHub.

GIT itself is a command line utility that one executes from a console (DOS window or linux shell); however, many GUI-based programs exist that greatly simplify

this process. For Windows, a program called TortiseGIT is a very nice tool (which I use). OS X and linux have many alternatives as well.

To familiarize yourself with GIT, YouTube has a many excellent tutorials. One set of tutorials that does a good job introducing GIT to a complete novice (why we use it and how we use it) is located below: `https://www.youtube.com/watch?v=m3X5yPnubxc`

This is an introductory video that introduces one to GIT, the concepts behind it, and how to use it effectively. This is in contrast to many brief tutorials and paragraph-length articles which simply list a set of steps to perform one specific task (be wary of random websites telling you how to execute GIT commands, some of them may cause great harm). As you may have guessed, understanding the fundamentals versus robo-typing commands will save you (and those who work with you) a significant headache for all but the most trivial cases. For a decent, yet condensed intro to git for those familiar with version control, here's a 20 min clip that will do so (`https://www.youtube.com/watch?v=Y9XZQO1n_7c`); however, some important concepts and motivations behind why GIT is the way it is ought to be gleaned from the aforementioned video.

A quick YouTube search will reveal hundreds of hours of videos explaining GIT; so perhaps watching a quick video when time permits a few times per week is sufficient. It is still very early, so there is plenty of time to ease into using GIT. This tool is one of those things that many projects just "expect" you to know (many at AFRL use GIT) - especially those involving software development. It is also a skill that may set you apart and create demand for your abilities in the future.

I would recommend that each of you create your own GIT repo and try storing your course work for each of your classes in it. Add, commit, update/modify, and recommit each digital homework assignment, programming task, report, etc. Learn how to manage your own files in your own repo with the Git-Bash and practice dealing with merges, branches, and conflicts inside your own test repositories. Practicing this on your own will let you experience both sides of a merge conflict *before* encountering them on a larger repo used by many people. I've seen holy wars waged when someone accidently overwrites someone else's code during an ill-performed merge operation.

### 2.2.1 GIT on an AFIT Machine

If you'd like to use GIT on your local AFIT machine, these instructions should help you out. No admin rights, no network trickery. Feel free to share these instructions with anyone else who may benefit from them. The reason we need to perform these extra maneuvers at AFIT is because the network utilizes a proxy server for all http / https traffic. This means all clients must send their http/https traffic to a single server. That server inspects the request and chooses to send it to the destination on behalf of the client, if approved. The destination server then sends the reponse to the proxy server, which subsequentally returns the response to the client. The AFIT/SC network only allows proxy traffic – without using the proxy, no http / https traffic is allow into or out of the network.

I've managed to setup GIT to work properly on the local AFIT machines and authenticate with the proxy server. One can clone, push, pull to gitHub or gitLab repos (or other http / https served GIT repos) – below are simple instructions for you to do the same (roughly 5 minutes of time). This does NOT require installing any software, NOR does it require admin rights. It also properly authenticates with AFIT's proxy and does NOT bypass any network authentication; therefore, it travels through the same network path that your Internet Explorer traffic does – so any blocked domains are still blocked and your traffic is still logged by the network admins. The GIT traffic uses http or https as the underlying protocol. Both command line and GUIs can be used on your AFIT machine (see bottom section for GUI info).

1. Goto L:\NYKL\PortableGit-1.9.5-preview20141217.zip and copy the archive to your local hard drive. Extracting it to C:\temp\git, is a great place since you have read/write access within C:\temp\. I have chosen the aforementioned GIT version carefully because newer versions do not appear to correctly authenticate with the AFIT Proxy server. I'd recommend trying this archive first before attempting to use updated versions. If I find a new version that I can configure to properly authenticate with AFIT's proxy server, I will update this information.

2. After extracting the archive to your local machine, you'll have the necessary executables to open a beautiful bash shell. This can be opened by double clicking on, C:\Temp\git\git-bash.bat as shown below in Fig. 1. One may place a shortcut to this `.bat` file in a more convenient location.

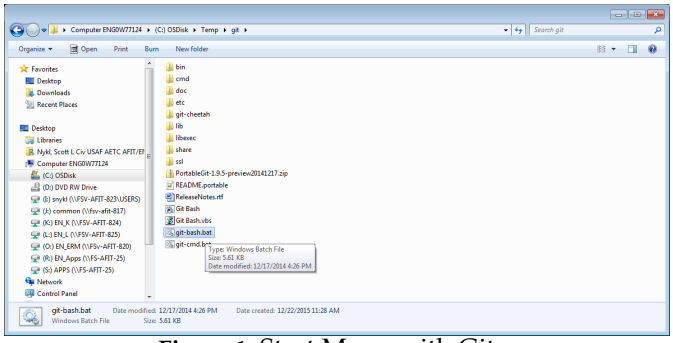

**Figure 1.** Start Menu with Git

3. You will see something like the following:

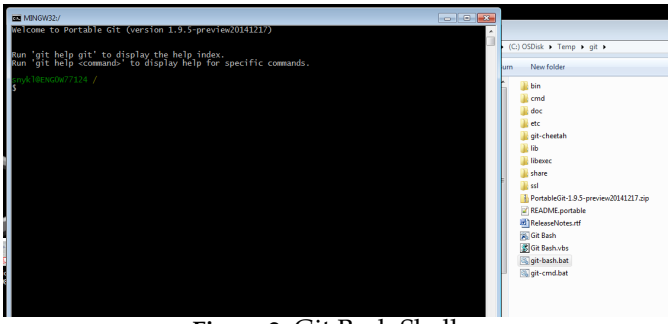


**Figure 2.** Git Bash Shell

4. Now that we have a shell and access to git commands, we must configure GIT to use AFIT's proxy, otherwise all git requests will be denied and a 407 Server Error will be received. To configure these, we will issue the following two git configuration commands:

```
$ git config --global http.proxy http://username@proxyedu1.afit.edu:8080
$ git config --global https.proxy https://username@proxyedu1.afit.edu:8080
```

Replace username with your AFIT proxy username. This username matched the username on my AFIT computer, and it also matched the OutLook "alias" to my email:

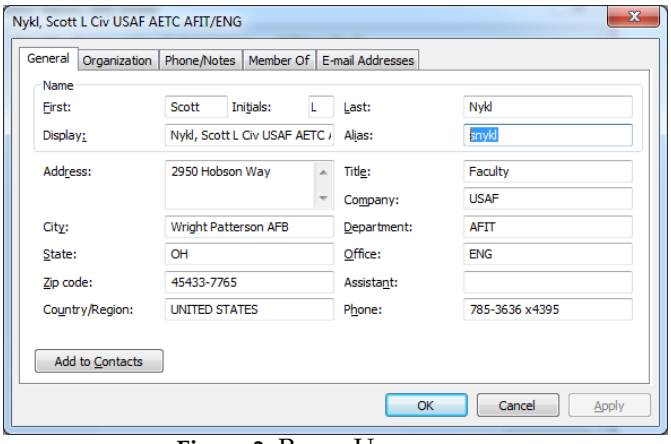


**Figure 3.** Proxy Username

Optionally, to be a good "git neighbor" one ought to configure their git client's user name and email so any commits clearly delineate the author.

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

5. After issuing the above two (or four) commands, they can be verified from the git shell by issuing the following 2 commands:

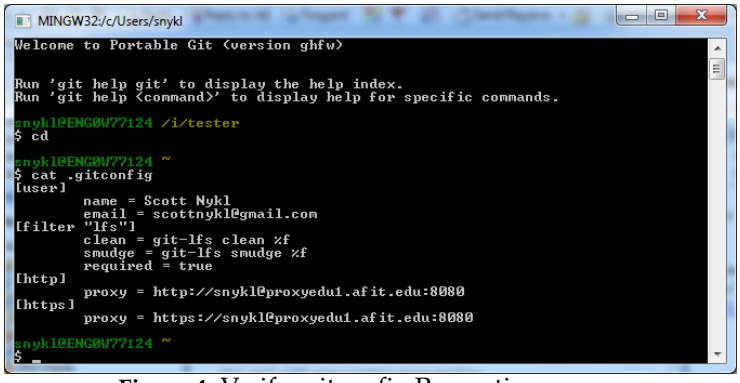


**Figure 4.** Verify .gitconfig Properties

Note that the email address / username is only used for the git commit logs and is not used for any other purposes.

6. Now we can clone a github / gitlab / http git repo, modify, pull, push, etc:

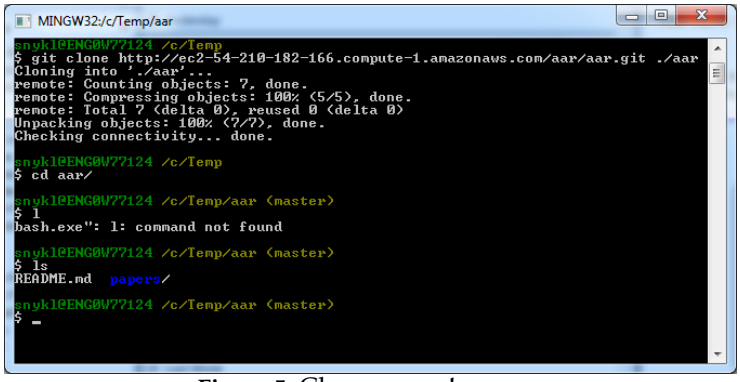


**Figure 5.** Clone a repo!

At this point, git can push, pull, add, commit, etc. You're all set to locally work on your research!

### 2.2.2 GIT with SSH

For security, many servers may utilize either SSH or HTTPS to encrypt data sent to and from the server. When using SSH for authentication, no configuration changes need to happen with respect to the GIT client. The only thing that must be done is to:

- Load your private key on your local machine. Use a utility such as Pageant.exe (windows), keyring (linux/os x), keychain (linux / os x), etc.
- Copy the corresponding public key to the server. With GitLab or GitHub, you may login to your account via the website and choose to upload your public key as shown in Fig. 6. Login and goto "Profile Settings" → "SSH Keys" → "ADD SSH KEY".

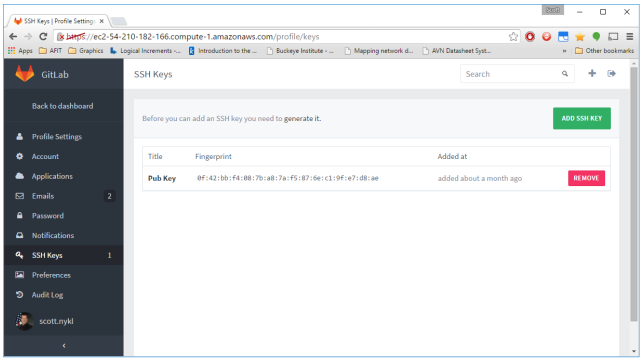


**Figure 6.** Upload your Public Key to the server. Load your Private key on your local machine using a utility such as Pageant.exe, keyring, keychain, etc.

- When checking out a repo, use the "SSH" url from GitLab or GitHub as shown below in Fig. 7.

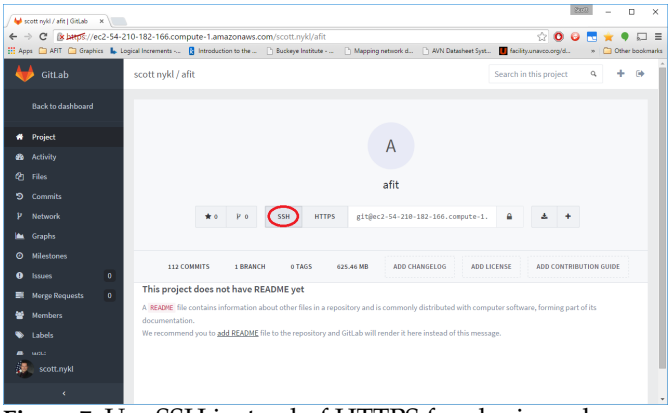


**Figure 7.** Use SSH instead of HTTPS for cloning when you have copied your public key to the server.

### 2.2.3 GIT with HTTPS

For security, many servers may utilize either SSH or HTTPS to encrypt data sent to and from the server. When using HTTPS for authentication, your GIT client needs to be configured to use the HTTPS server's certificate authority information (this certificate is essentially the server's public key along with information about who created the key (which is itself signed). In many cases for small servers, the key is "self-signed" which may cause your browser to pop up an error about "self-signed" certificates - this is not necessarily a security hazard if the users know this and verifies the certificate themselves).

If you are using https to check out the repo, you will need to tell your git client to either ignore the self-signed certificate or to use the self signed certificate. Commonly, the linux, OS X, and Windows certificate file ends with .crt. These are simply Base64 encoded X.509 certificates with the corresponding OS-specific line endings (LF vs CRLF).

To ignore the self-signed certificate on your AFIT machine, simply type:

```
$ git config --global http.sslVerify false
```

The above command will make your global GIT settings ignore self-signed certificates, which may be a security hazard if you use multiple GIT repos. To alleviate this, one may simply disable ssl verification on a per repo basis by simply entering the root directory of that repo once it is checked out and typing:

```
$ git config http.sslVerify false
```

To use the GitHub / GitLabs server's SSL certificate on your machine, you may tell your GIT client where the certificate file is located on your hard drive (after obtaining the certificate from the GIT server admin).

To make this a system wide setting for your machine:

```
$ git config --system http.sslCAInfo /c/Temp/httpsServerCertificate.crt
```

To set this for just one repo (after that repo is checked out):

```
$ git config http.sslCAInfo /c/Temp/httpsServerCertificate.cer
```

Finally, when cloning the repo, use the HTTPS path as shown below in Fig. 8

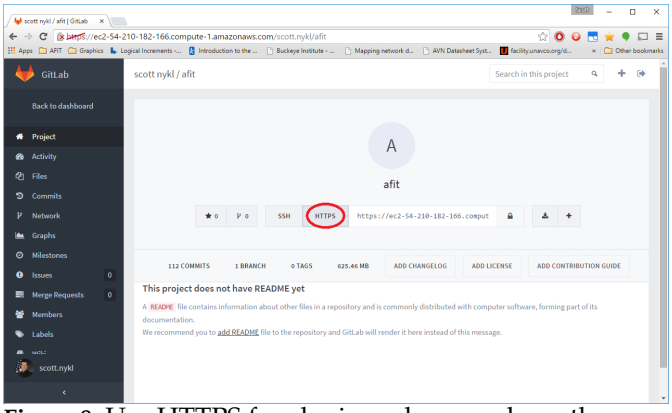


**Figure 8.** Use HTTPS for cloning when you have the server's certificate install and your GIT client's sslCAInfo configured.

## 2.3 cURL

cURL is a command line tool that can be used to send http and https requests. There are many uses for this tool, but I will specifically discuss one helpful use here at AFIT. Namely, AFIT's commercial WiFi (AFITCIS) and the automated login page. Let's suppose we have several Raspberry Pi's or other pseudo-embedded devices that have built in WiFi interfaces. Without a web browser, however, our devices are not able to "Accept" the Login Agreement as shown below.

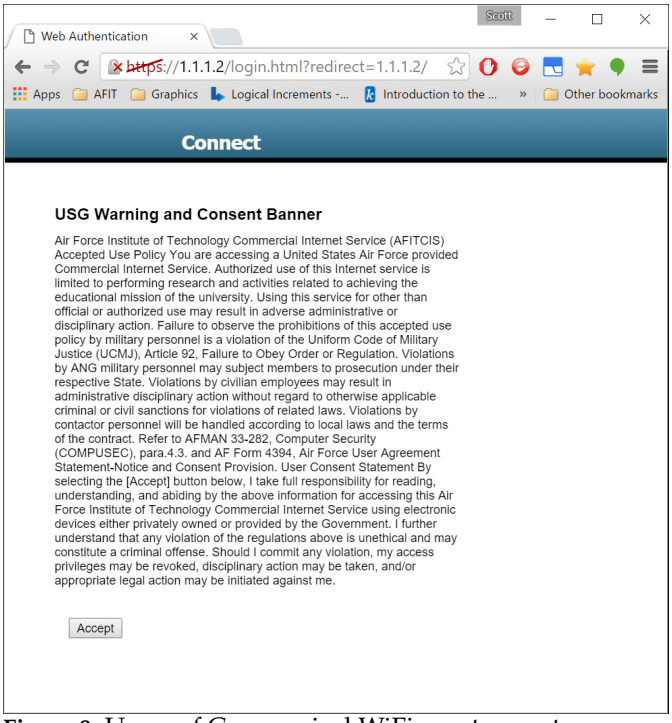


**Figure 9.** Users of Commerical WiFi must accept an agreement before connecting.

Similarly, if we enjoy using your own laptop at AFIT, it may be an annoyance to continually have to "Accept" the Commercial WiFi banner. Using cURL, we can automate this procedure and even maintain a continuous connection where a machine automatically reauthenticates every 4 hours – I've observed an auto-disconnect after 8 hours on AFITCIS. cURL let's us utilize a cross-platform, command line-based toolset to automate this procedure for us (cURL will even run on your phone). cURL may be downloaded from `https://curl.haxx.se/`, but it is also included as an executable in the GIT archive discussed in Sec. 2.2.

The simple script shown below logs a machine on to the AFIT's commercial WiFi. This assumes the machine is already "connected" to the AFITCIS WiFi (via the operating system), but has not yet clicked the "Accept" button. These two lines may be placed in a .bat file or a .sh file, depending on your preference / platform.

```
curl -k -v --data "userStatus=1&err_flag=0&err_msg=" https://1.1.1.2/logout.html
curl -k -v --data "buttonClicked=4&err_flag=0&err_msg=&info_flag=0&info_msg=&red
                                    irect_url=" https://1.1.1.2/login.html
```

In the script above, the first line automatically logs out of AFITCIS just to ensure the machine fully disconnects to restart the session time associated with each connection (AFITCIS will abruptly disconnect your machine after 8 hours – even if you are downloading a file). The second line sends an http command to the AFITCIS login screen that clicks "Accept" automatically.