Computer Systems
Fall 2021
Course Information

Prof. Kaplan

1 Introduction

This course will employ a number of tools that are likely new to you. Indeed, a side-goal of this course is to throw you into the deep end of using a wider variety of tools to program, debug, communicate, and share code, all in an environment that is not your computer.

These tools are not chosen willy-nilly, or even just for the experience of using them. They are chosen because they are powerful tools that can benefit our work. However, they seem frustrating (if not outright maddening) if you do not learn how to use them. You should expect to spend time getting acclimated to these; the more you do so, the more quickly and effectively you will be able to delve into the projects that are so central to this course.

This document is not meant to teach you about all of these tools; instead, its goal is to help you set them up and be sure that they are working for you. It is the initial step into the course projects—a kind of *Project 0*. Go through its contents thoroughly, installing software as needed, asking for help if you have problems, and making sure that you have access to each tool as soon as possible.

2 Slack

Communication for this course will be conducted in a number of ways:

- **Via website:** The course website will be the primary location for announcements, the posting of assignments, and links to any other materials.

  `https://sfkaplan.people.amherst.edu/courses/2021/fall/COSC-171`

- **Via email:** For time-sensitive announcements, or for more involved and structured exchanges, good ol' email remains essential.
- **Via Slack:** For quick questions and informal one-on-one or group interactions, we are going to use *Slack*. It's a fancy texting platform that allows us easily to interact individually, as a class, or in small groups.

For starters, you should login to Slack using your Amherst College credentials:

`https://amherstcollege.slack.com`

You can use Slack within a browser from that site, or you can install the Desktop application:

`https://amherstcollege.slack.com/get`

I recommend installing the Desktop application (and, if you ultimately like this tool, you can also install it on your phone) so that you get notifications when new messages arrive.

When you install Slack, you should `send me a direct message` (`sfkaplan`), and then I will add you into the class channel (`cosc-171-f21`).

Protip: In the upper left corner, if you click on the drop-down menu next to *Amherst College*, you can click on *Preferences*. Within the *Preferences* window, select *Advanced* at the bottom of categories on the left. At the top, under *Input Options*, there will be two check-boxes:

- `When typing code with ```, Enter should not send the message`
- `Format messages with markup`

Select both of these. And then, if you're not familiar with markdown,[1], then you should start to get familiar with it. If you're going to have conversations involving code, it's terribly handy. To get started:

- **Tutorial:** If walking through an interactive tutorial is a good way for you to pick up some new skills, try this markdown tutorial.
- **Cheat sheet:** If you like to have a quick reference on markdown handy, this markdown cheat sheet is a simple reference that shows the basics.
- **More depth:** Search the web for *markdown*, and you will find ample documentation, help, and examples.

3 `ssh`, `git`, and *X11*

To connect to a Linux server (see below, Section 5) and manage the code of this course's projects, you will need a collection of tools that work together. Specifically, you will need an environment that provides:

- `ssh` (the *secure shell*): This software allows you to connect to other computers via the network through encrypted communication channels.
- `git`: This *revision control software* is used to keep a history of your source code over time. It will also allow you to document bugs, collaborate with others, and synchronize copies of your code on multiple systems. It uses `ssh` to communicate with our *repository server*.

[1] Yes, that's right, the markup that Slack uses is one called *markdown*; it's common for programmers.

- *X11 windows server*: This old but effective software will allow you to run a program on our class Linux server, but then have that system use your computer as the display, communicating the management of the program windows over the network. We will use `ssh` to carry that communication between this software and the Linux server.

3.1 Installing

How you set up these tools depends on the type of computer you have. Jump to the section below for your computer.

3.1.1 macOS

***XQuartz*:** Begin by installing the *X11 server*, known on the Mac as *XQuartz*. You can go to `https://www.xquartz.org` to download this software. When you do, open the installer (named something like `XQuartz-2.8.1.dmg`), and follow the standard installer prompts.

When you are done, run *XQuartz*—it should be in your `Applications` folder. You should then get a window that looks much like the *Terminal* app, showing you a *shell prompt* at which you can type commands.[2] This window is known as an *xterm*, and will be the starting point working on projects on your computer.

`ssh`: Luckily, `ssh` is already installed by default on Macs. There is nothing you need to do here.

`git`: To install `git`, download the *Git for Mac installer* at:

`https://sourceforge.net/projects/git-osx-installer/`

Just click the green *Download* button to start the download. Once downloaded, run the installer.[3] Once the installer has completed, jump down now to Section 3.2 to make sure that you can type commands and that they work.

3.1.2 Windows

Installing: The easiest way to install all of these tools is to install a *UNIX emulation layer* known as *Cygwin*. To install what we need, follow these steps:

1. Go to the *Cygwin* home page:

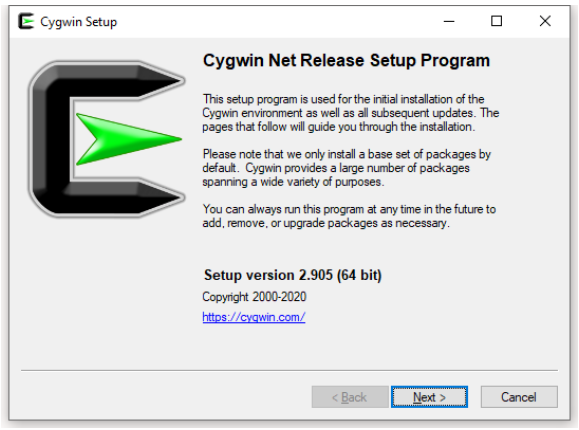
   `http://cygwin.com`

   On that page, under *Installing Cygwin*, click on the link for `setup-x86_64.exe` to download the installer. Once it has finished downloading, run it.

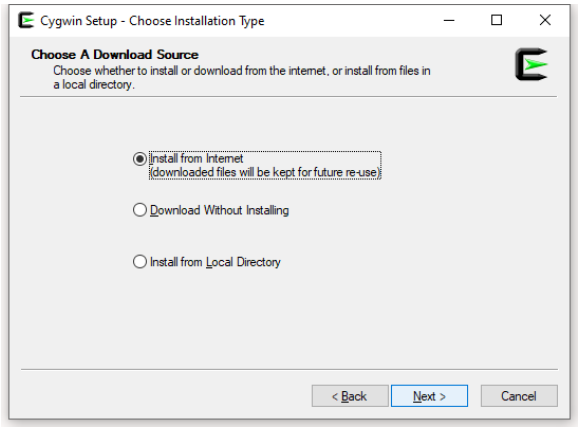
[2] Don't worry if you don't know any commands to type.

[3] Your security preferences may not be set to allow this installation. If that happens, go to *Settings -> Security and Privacy*, and under *Allow apps downloaded from*, select *git*.

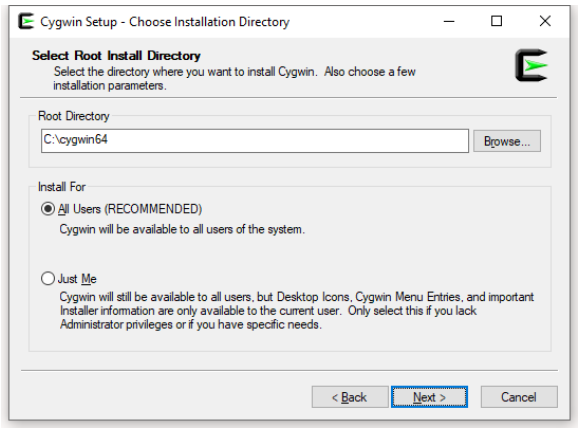
2. Just click `Next`.



3. Accept the default choice (`Install from internet`) by clicking `Next`.

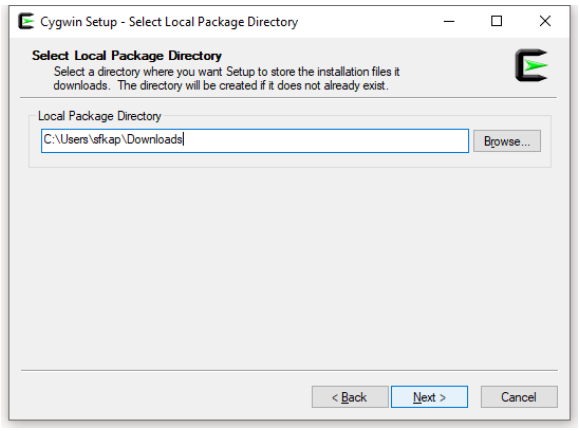


4. Accept the default directory (`C:\cygwin64`) into which *Cygwin* will be installed, and the default choice of installing for `All users`.[4] Just click `Next`.

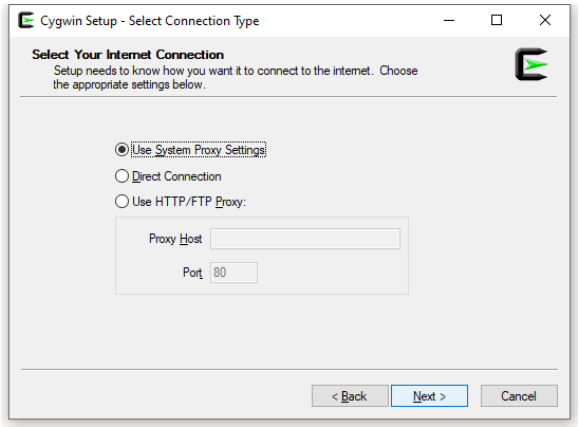


5. Once again, click `Next` to accept the default download directory.

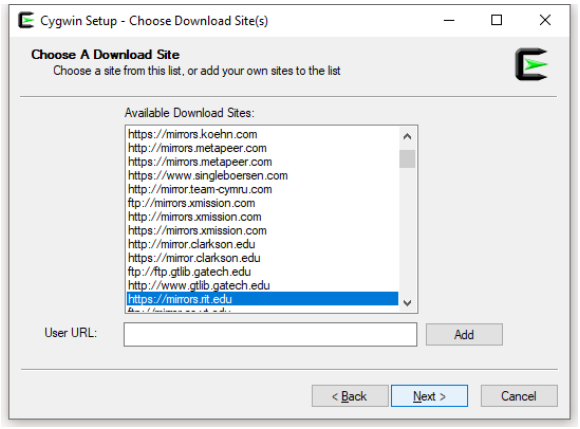
[4] You can choose a different directory if you need to do so, but be sure to have a reason.



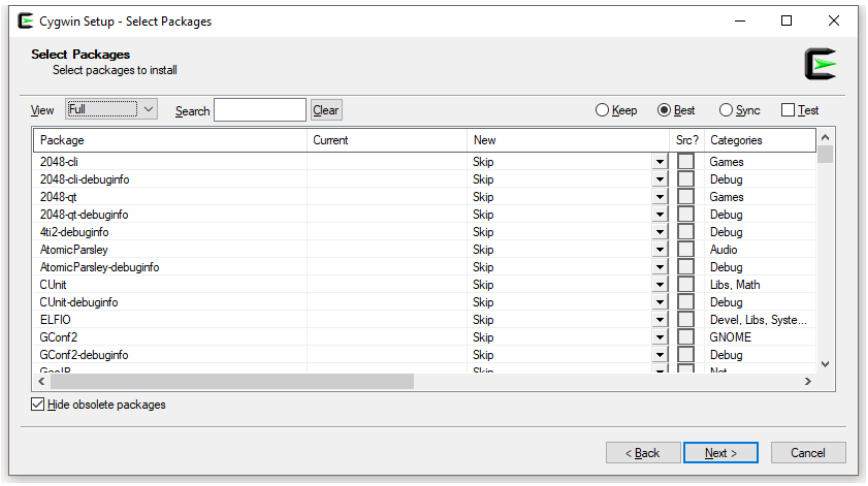
6. Yup, click `Next`, accepting the default to `Use system proxy settings`.



7. When prompted to `Choose a download site`, you can choose any one of the sites listed. Choose one that seems close to you for faster downloads of the *Cygwin* packages; they all have the same content, so there's no wrong choice, only slow choices (depending on your locale). Complete your choice by clicking `Next`.

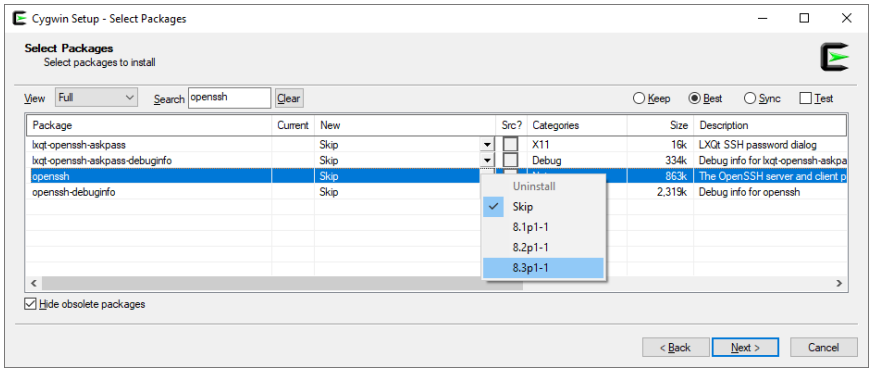


8. After downloading a list of packages, the installer will show you window in which you select packages to install. It initially will look something like this:



For the `View`, in the upper left, choose `Full` to see all of the packages. Then, next to that, you can enter the name of a package in the `Search` bar, and only packages that contain that name will be shown.

For example, enter `openssh`, and you will see a list of packages whose names include that string. The third column, labeled, `New`, is where you select which version of the package you want to install. The `openssh` package itself will be labeled `Skip` in this column. If you click on the drop-down menu for that entry, you can select the most recent version, at the bottom of that drop-down menu:



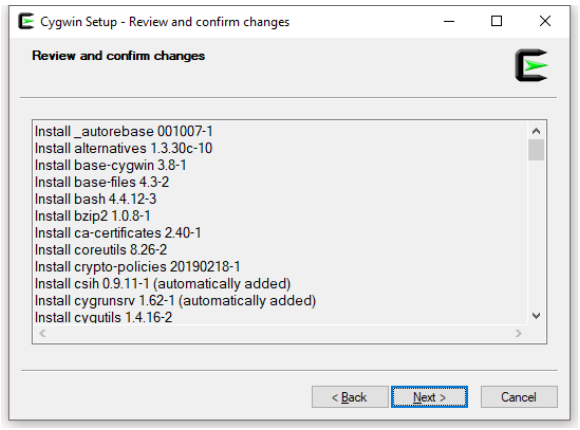
Having selected a version to install, you can now click the `Clear` button next to the `Search` bar, allowing you to enter a different package name.

Follow this process to select the following packages to install:

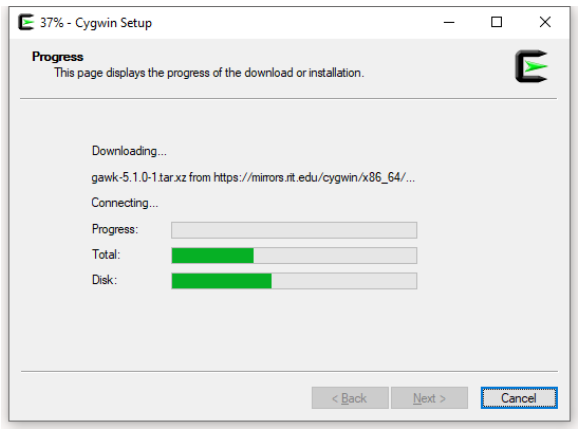
- `openssh` (already selected, if you followed the above)
- `git`
- `xorg-server`
- `xinit`
- `xlaunch`
- `xterm`
- `emacs-w32` (optional, but may be handy)

When you have selected these, click the `Next` button in the lower right, as usual.

9. You will see a window to `Review and confirm changes`. It will show a list of packages that form the base of *Cygwin*, plus the packages that you selected, plus any packages on which your selections depend (e.g., libraries). Click `Next`.



10. The `Progress` window will display the downloading and installing of each of the packages.

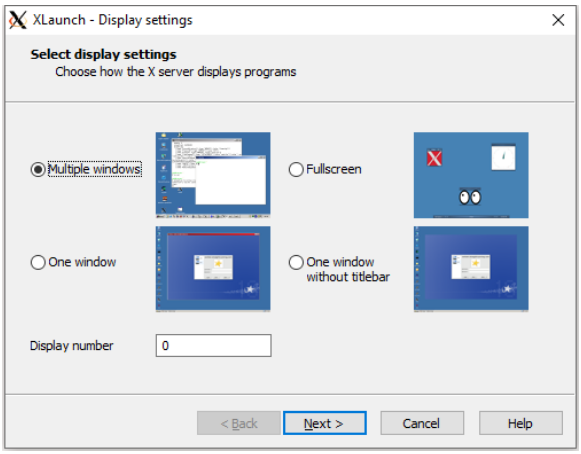


11. Finally, you will see a completion window. Decide whether you want *Cygwin* to appear on your desktop and/or start menu. Finally, click `Finish`.

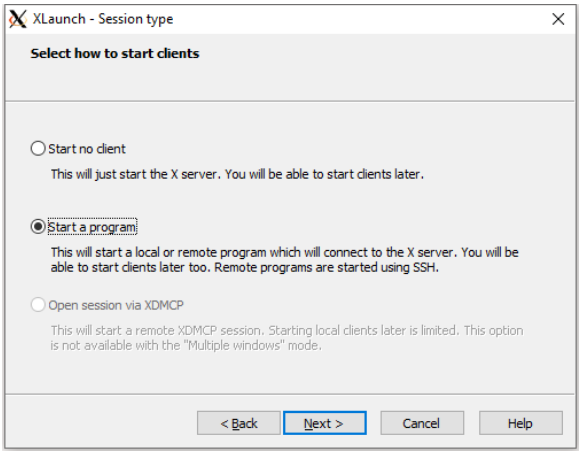
Running: Now that you've installed *Cygwin*, here is how you run it:

1. Click on your *start menu*, and type in `XLaunch`. The icon for that program should appear. (You can right-click on it to pin it to the start menu or the taskbar.) Double-click it to run `XLaunch`.

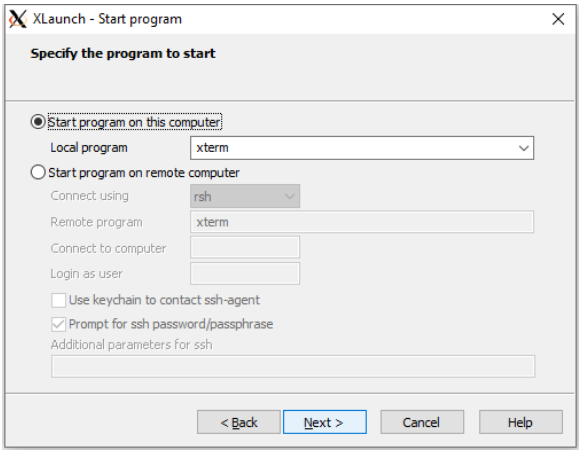
2. You will see a window to `Select display settings`. Accept the default, `Multiple windows`, by clicking `Next`.



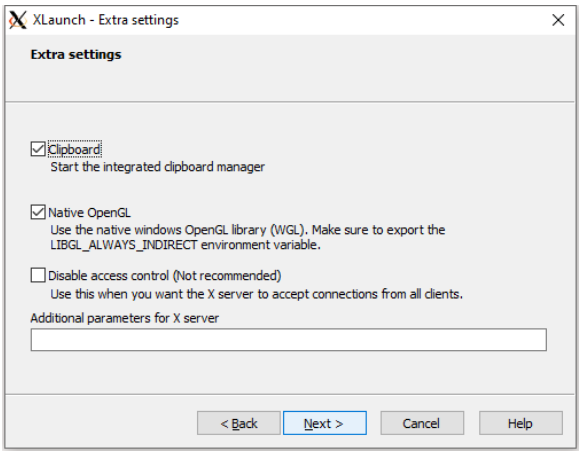
3. For the `Session type` window, select `Start a program`, and then click `Next`.



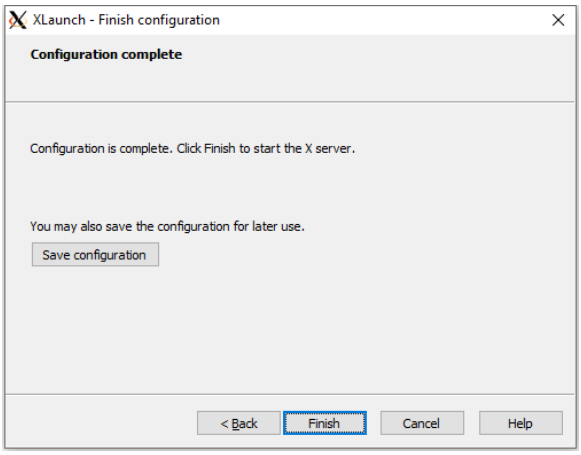
4. When asked to `Specify the program to start`, accept the default, `Start program on this comp` namely `xterm`. Click `Next`.



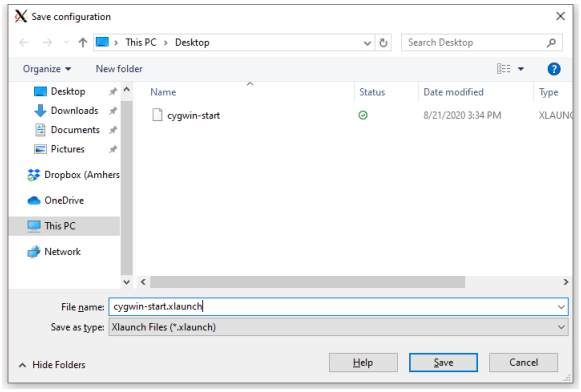
5. For `Extra settings`, the defaults are just fine. Click `Next`.



6. On the `Finish configuration` window, first take a detour by clicking `Save configuration`.



That will bring you to a file-saving dialog. Choose a folder (I think the Desktop is a good place for this file), and name the file `cygwin-start.xlaunch`.



Click `Save`, and then you will return to the `Finish configuration` window. Click `Finish`.

After a few moments, you should then get a *terminal window* that shows you a *shell prompt*, at which you can type commands.[5] This window is known as an *xterm*, and will be

[5]Don't worry if you don't know any commands to type.

the starting point working on projects on your computer.

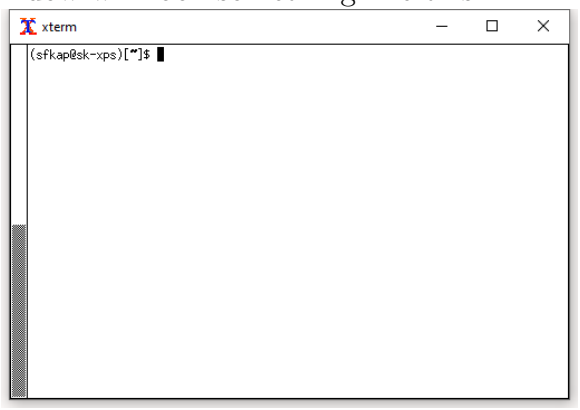
Jump down now to Section 3.2 to make sure that you can type commands and that they work.

3.1.3 Chromebooks, Linux, anything else

If you have a Chromebook, contact me, and we can try to figure out how to proceed. If you have a Linux laptop, then all of the tools are already installed and ready to go. If you have some other kind of computer, also contact me, and we can figure out what to do.

3.2 Testing your installation

Now that you have an *xterm* window open, with a shell prompt awaiting your input, let's try a few commands to be sure that our tools are properly installed and working. Your *xterm* window will look something like this:



Notice that you may have somewhat different text prior to the *prompt*, which is the `$` symbol. Traditionally, in documentation like this, the text prior to the prompt is not shown (since it varies from system to system, and from user to user), but the **`$` is** shown to signify that what follows is the command that should be typed. That is, **don't type the dollar sign**; type what follows it.

A simple command: The `ls` command (short for *list directory*) will show the list of files and directories (folders) in your current directory. Try it:

```
$ ls
Documents Downloads Templates Videos
```

The output you will see may not perfectly match the above—that's OK. What you are seeing is a listing of your *Cygwin home directory*, sometimes signified with a *tilde* (`~`) character. You can see the *absolute pathname*—the listing of directories starting from the top of your hard drive—by using the `pwd` (*print working directory*) command:

```
$ pwd
/home/sfkaplan
```

Testing `ssh`: First, let's verify that `ssh` is now an available command:

```
$ which ssh
/usr/bin/ssh
```

The `which` command is showing you that it found the `ssh` program in the `bin` directory, which itself lies within the `usr` directory at the base of the hard drive. If you see something like this…

```
$ which ssh
which: no ssh in (/usr/bin/:/bin:/usr/sbin:/sbin:...)
```

…then something has gone wrong with your installation of `ssh`, and you should seek help. (Note that the list of directories following the `no ssh in` message may be much longer that what is shown above.)

Assuming that `ssh` was found, let's try using it:

```
$ ssh -Y yourusername@remus.amherst.edu

The authenticity of host 'remus.amherst.edu (148.85.1.64)' can't be
established.  RSA key fingerprint is
SHA256:FspduCkkgtlHw6ZcBak54qgwHFOxMLCm+4K2ygb+THs.  Are you sure you
want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added 'remus.amherst.edu,148.85.1.64' (RSA) to
the list of known hosts.

sfkaplan@remus.amherst.edu's password:

Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Fri Aug 21 17:27:01 2020 from c-24-91-28-111.hsd1.ma.comcast.net
(remus)[1]~$
```

First, observe that you should not literally specify `yourusername` in the command; you should substitute your college username (e.g., `sfkaplan@remus.amherst.edu`). Also note that `remus` is the name of a Linux server to which you have login access. It is not the system we will be using for this course, but it serves as a good test.

When asked, `Are you sure that you want to continue connecting?`, you may answer `yes`. Luckily, this question, and the warning that precedes it, will only appear this once. If you login to `remus` again later, it will not ask this question.[6]

You will notice that you have a shell prompt again, except this time, the commands will be carried out by the server at the other end of the connection. You can try `ls` and `pwd`, for

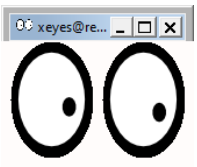
[6]What is this question doing, then? It is verifying the *public key fingerprint* that `ssh` will use to communicate with the server. When you connect to `remus` later, it will match the fingerprint that the server provides with the one you approved upon the first connection. If they don't match, it will warn you, since that may imply that the security of the connection is compromised.

example, but you'll see different things than when those commands were being carried out on your computer.

As a final test of the ability to give commands that open windows, try this:

```
$ xeyes &
```

After a few moments, you should see something like this:



As you move around your cursor, the simulated eyes will follow it. That's a sign that the login has worked, and that the server (`remus`) is using your screen as its display.

When you have enjoyed this silly program for a moment, you may close it (click the close button, as usual), and then exit from remus:

```
$ exit
```

Doing so will bring you back to the shell prompt running on your own computer.

Testing `git`: First, use the `which` command to make sure that the shell can find the `git` program:

```
$ which git
/usr/bin/git
```

You can see a menu of commands (which may not make much sense to you now) by entering `git help` at the prompt; go ahead and try it. Otherwise, we cannot more deeply test `git` until we do some other things first. Onto the next section.

4 *GitLab* and key pairs

In order to share code, track changes, keep a history of versions, and document issues, we will use `git` in concert with *GitLab*, which is a *repository platform*.[7] The college has its own *GitLab* server, `gitlab.amherst.edu`, that will allow us to work together in groups easily via our college logins.

Finally, *GitLab* allows us secure and easy logins via *ssh keys*. We are going to leverage that capability to enable access to this class's Linux server, which we will address in Section 5.

To get our *GitLab* server configured for our uses, follow these steps:[8]

1. Access the college's *GitLab* server through a browser. Specifically, use this link:

[7] You may also have heard of *GitHub*, which is quite similar.

[8] These instructions are derived, in part, from a help-page on our *GitLab* server: `https://gitlab.amherst.edu/help/ssh/README`. Feel free to refer to these instructions for additional help.

```
https://gitlab.amherst.edu
```

You will be presented with a login page, where you should use your Amherst College username and password.

2. Go back to your *xterm*, and at your shell prompt, enter the following command. Be careful to recreate the sequence of characters exactly:

```
$ ls ~/.ssh/id_rsa
```

What you do next depends on the output of this command:

Do you see output something like, `/home/username/.ssh/id_rsa`: You **already have an ssh key**. (Did you set it up for some previous class or project?) **Skip to the next step.**

If you see an error ending, *No such file or directory*, then we need to make an ssh key. Doing so requires a few steps:

(a) Begin by entering the following command, substituting your actual Amherst College username for `yourusername` in the example:

```
$ ssh-keygen -o -t rsa -b 4096 -C "yourusername@amherst.edu"
```

(b) When prompted to choose where to save the key pair, just press `Enter` to accept the default suggestion:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sfkap/.ssh/id_rsa):
```

(c) You will be prompted to enter a password (a.k.a. passphrase), needing to type it twice to verify that you typed it correctly. Notice that when you type, **nothing** will appear in the terminal—not even placeholder dots or asterisks.

Once you have entered your password twice, you'll see some additional output:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yourusername/.ssh/id_rsa
Your public key has been saved in /home/yourusername/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:aApoeU48NYIeNmSORL4fv0CVsVd1NPJnmLBVFx0/imw
yourusername@amherst.edu
The key's randomart image is:
+---[RSA 4096]----+
```

```
|.=   .   ..+.=.+=|
|B .   + .   B + +|
|.B . * .   . + +.|
|o.* + o.  . . + .|
|.=.B  o S  E .   |
|. *.+o    .      |
|   +..           |
|    . .          |
|     .           |
+----[SHA256]-----+
```

This last bit of output is a graphical-ish, textual representation of your public key's fingerprint. Don't worry about it.

3. Copy the contents of that public key into your clipboard, using the `pbcopy` command if you use **macOS**, and `clip` if you use **Windows**:

```
For macOS:
$ pbcopy < ~/.ssh/id_rsa.pub
For Windows:
$ clip < ~/.ssh/id_rsa.pub
```

4. Now return to your browser and *GitLab*. In the upper right corner, click on your *avatar* (it may be a little picture of you, or appear as a little circle with your initials) to get a drop-down menu. Select `Settings`.

   A new screen will appear, with a column of sections on the far left. Choose `SSH Keys`.

   Look at the bottom of the window, under *Your SSH keys*: **Is there a key already there? If so, then skip to the next step**. If not, continue here.

   Click in the text box under *Key*, and then *paste*. You should see pasted something that looks like this:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDcC6LMDcpFoVLvQx9/PmfaQHPl1Pg2
AB1gA5gAKWU3H9HUGtZFktDapOB4aqzUgkEea0k8NLB8hLu7M+C4cd+tZWYDuZM7oWFD
zX6kI5Mlgs6WwZOvp8ikle34589GYf43hzQQqd0SnJv2l3jnYteOivEFUbZOikwGpMuT
Y4bZ46rVxQkMY5hdX3QzIu8qOWMk0FOJjNPLU53uLA3ZFn+2ZA48UssieulByuNlxWd+
P8acazGym3sDXPUv/OkIlDd/0nDXBfZnOBcbVcFqfAj4pC1/nqYfVetpYLzGc87RR46a
NhY+GHV2UT7pLd7Nh6jbQgz6dnxKdIFg4o9j5I/ayCsbJVWx3CM2AbNxSMv3APbvNhzG
pn5LZThOemrpMbuRZoC6dls5uaH1VYdlusy6mgZlVJSfxF5UzFWEiP+HZRQsYYWZDft1
4nOGXdiWBe+MWt2TzxXwUHgUzT/5KRQR7CtSAq1eKRIYykqKOEez/hpcIkBfFOeNIwW4
Sp190KXxW4lnfGugeS0HTY8uC41v5QKFpZUj6Yx5ZK3NFCG1D9HU6DmMF1MjoBgo0mrd
AK8UCFcihfhGI31kG2LOxxkyKW/URMoJDwhq+nWifN1iU+nHXA/8vthJxJzeiu+B0vAC
I5rcWUUe8f6NRS1jCFdqX4Vi1r7PMGgcIycRMVa18Q== yourusername@amherst.edu
```

Then, under *Title*, just put your Amherst College email address, with the `@amherst.edu`. Finally, click the `Add key` button.

5. Finally, to test that everything is configured correctly, go back to your *xterm* and enter this command into your shell:

```
$ ssh -T git@gitlab.amherst.edu
```

   Just as you saw a warning when you first made an `ssh` connection to `remus.amherst.edu`, you will see a similar warning and a prompt, asking whether you want to accept the fingerprint for the server. Enter `yes`.

   You will be prompted for the password you chose when making the key, and then you will see a welcome message:

```
Welcome to GitLab, @yourusername!
```

6. One last step! Open *Slack*, and send me a direct message, telling me that you've completed the configuration of `ssh` keys for *GitLab*.

5 The `systems.aws` server

When I get the message on *Slack* that you've configured a key on *GitLab*, I will add you to a special group on that platform. Within a handful of minutes, some code will run that will copy your key onto another system: our class Linux server.

This server is where we will do all of our work. It contains all of the software that we need—assembler, compiler, debugger, editors, specialized libraries and kernel modules—for our projects. You will connect to it via `ssh`, and it will know your identity via your `ssh` key.

To connect to the server, go to your *xterm*, and enter the following command at your shell prompt:

```
$ ssh -Y yourusername@systems.aws.amherst.edu
```

You will first get the warning about the fingerprint and the request to continue; enter `Yes`. You will then be prompted for your ssh key passphrase; enter it, remembering that nothing will appear as you type.

And then you should see some login information, ending with a shell prompt, something like this:

```
Warning: No xauth data; using fake authentication data
for X11 forwarding.
Linux ip-172-21-144-196 4.19.0-10-cloud-amd64 #1 SMP Debian
4.19.132-1 (2020-07-24) x86_64

The programs included with the Debian GNU/Linux system are
```

```
free software; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Sun Aug 16 22:33:25 2020 from 68.235.43.14
yourusername@ip-172-21-144-196:~$
```

At this prompt, you can use the same commands that you have typed on your computer and on `remus`: `ls`, `xeyes`, and more. Try the `xeyes` command to make sure they appear. When you are done, `exit`.

6 Conclusion

If you have gotten this far, then you have the tools you need to get started with the projects for this course. Do not worry that you don't really know yet how to use these tools; that will come later, as we learn *x86 assembly* and *C*, and dig into the projects themselves.

If any component of these tools doesn't work, contact me on *Slack* or via email.