ECE/CS 250 – Summer 2017 – Prof. Bletsch
Recitation #1 – Unix

Objective: This recitation builds on the skills gained in the Unix course you completed for Homework 0. Here, you will practice logging into a Unix machine and doing some basic file manipulation and text editing. You will need these skills so that you can develop C programs. These are also useful skills if you plan to have a career in computing.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

Note: **The auto-magic power of Eclipse will not be here to help you.** You need to be able to navigate Unix-style systems using the basics: shell interaction, file upload/download, and a plain text editor. In industry, if you can only code if you have an IDE, your career is going to be painfully limited and simple tasks will seem needlessly complex. **Let the Unix flow through you.**
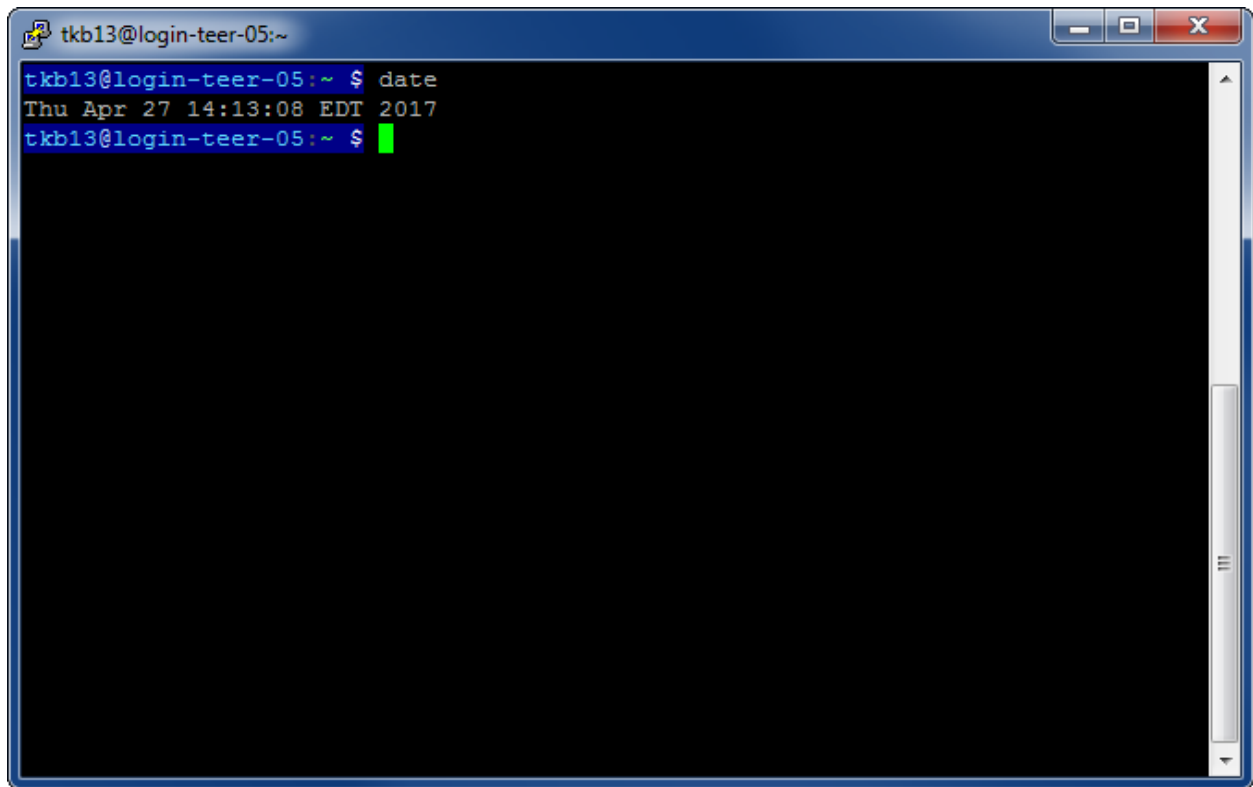
1. Logging in to a Unix machine

Duke maintains a cluster of x86/Linux machines in a top secret location. Fortunately, through the magic of networking, we can use them from wherever we are. To access them, we need to use a **secure shell (SSH) client**, and it can also be useful to use **X-windows** so that we can run programs with graphical user interfaces (GUIs) over the network as well. Below are instructions for installing/running X-windows and SSH.

Mac	Windows
X-Windows	
1) Download and install Xquartz from http://xquartz.macosforge.org. 2) Logout of your MAC 3) Login to your MAC	1) Download and install x-win32 from the OIT software website at https://oit.duke.edu/comp-print/software/license/detail.php?id=119 Download the latest version and follow the instructions in the PDF for licensing. If you want to use X-Win from off campus you should use the Activation License, not the License Server.

| Secure Shell (SSH)	

Secure shell is easy on a Mac since it is built into the Terminal Application
 	
1) Open the Terminal App. You can find it in the Applications/Utilities folder or by searching in Spotlight for Terminal
 	
2) At the command prompt, type `ssh -XY` netID@login.oit.duke.edu, where netID is your Duke NetID. This command initiates a secure shell connection to a Linux machine in the cluster. The '-XY' means "forward GUI applications".
 	
3) Enter your password.
	1) Download and install PuTTY from the OIT software website at https://oit.duke.edu/comp-print/software/license/detail.php?id=7
	
	2) Start x-win32 (i.e., run the program)
	
	3) Open a PuTTY terminal window
	  a. The first time, you should get a configuration screen. For Host Name put in login.oit.duke.edu.
	  b. Connection type should be SSH and Port should be 22.
	  c. Go to Connection category, open the SSH option, and click X11. Ensure that the check box next to X11 forwarding is checked.
	
	4) You can save a session for subsequent use by giving it a name and saving the session. Then you can later reload the session by selecting it and clicking "load".
	
	5) Click Open to start the PuTTY session. This will open a Terminal Window and prompt you for your NetID (i.e., login as:).
	
	6) Type your Duke NetID.
	
	7) Enter your password.

Congratulations – you are now successfully connected to a remote Linux machine!	

You now have a terminal session that is connected to **login.oit.duke.edu**. You should see a command prompt that is something like [netID@login-<something>].   Your command prompt may be a bit different, but that doesn't matter. At this prompt, type `date` (then hit enter/return, as you have to do after all commands on the command line). This Unix command displays today's date, as shown below.

You can have as many concurrent SSH windows as you like (e.g. one to edit code and one to compile/run).

Now type `xterm` at the prompt. This should open a window on your machine's screen that gives you another terminal on the remote machine:
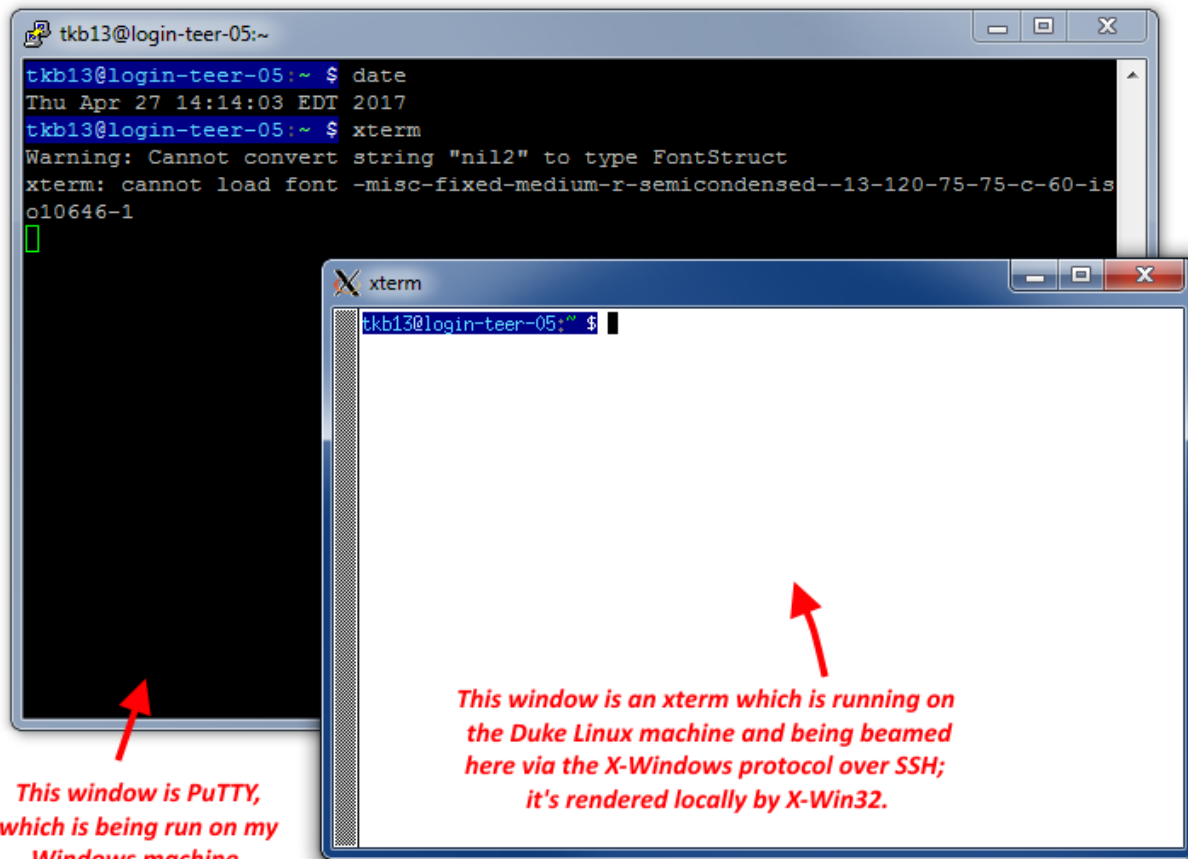
If xterm fails to load, you've done something wrong in setting up X-windows; please review the steps above according to your operating system.

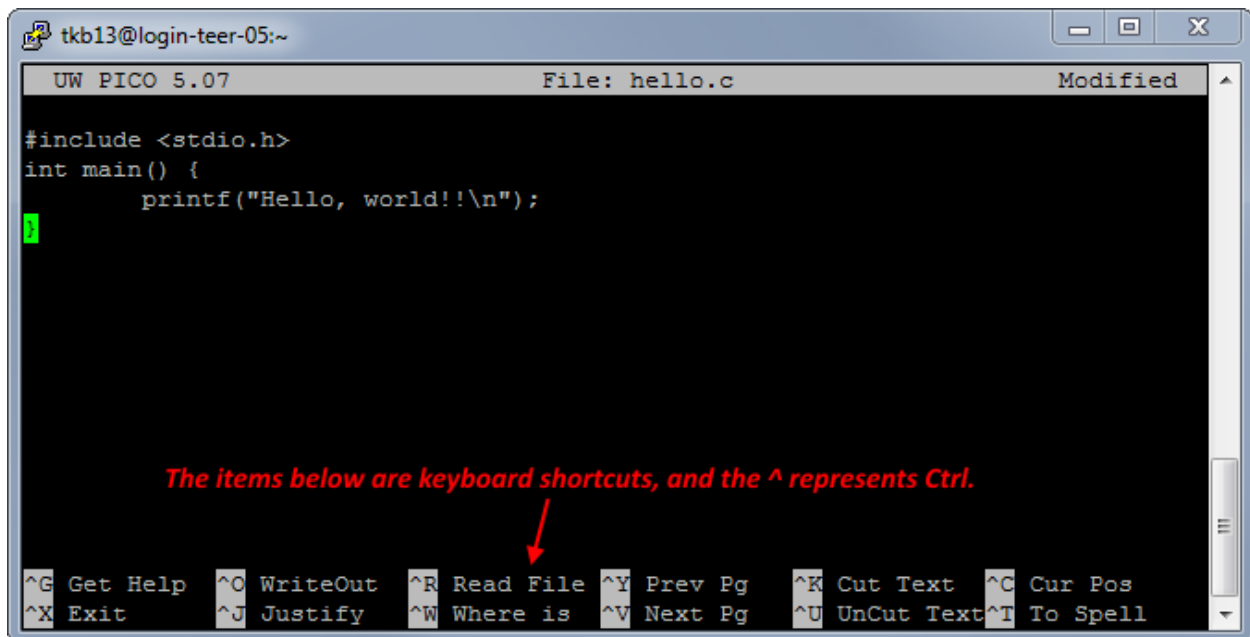You can close the xterm window by typing `exit` at the command prompt.

2. Using a Text Editor

You're going to be writing programs using a text editor. There are many options, including: `pico` (also known as `nano` on some systems), `vim`, `emacs`, `nedit`, `gedit`, etc. The `pico`, `vim`, and `emacs` editors run inside the command line window, whereas the `nedit` and `gedit` editors appear in separate GUI windows. Students in ECE/CS 250 tend to prefer `nedit`, but the choice of text editor is a personal preference (with strongly held religious attitudes).

Big picture: you need to get comfortable using one text editor. To do so, create a file with one of them. If you already know some C or C++, write a program in that language. If you want to write a (simple) Java program, do that. Just make sure that you can create, edit, and save a file. To start editing a file with `pico`, you can type the following at the command line:

```
pico hello.c
```

This line will start pico for use in editing a file called `hello.c`. If that file already exists, it will be opened for editing. If it doesn't already exist, a new file with that name will be created and opened for editing.

Once the file is open, please type some stuff—C code, if you know some, but even gibberish is fine for now—and then save the file and exit the editor (Ctrl+X in pico).

Now from the command line, type the following command to list the files in the current working directory: `ls`. You should now see hello.c (and other files).

Using the GUI-type editors (`nedit`, `gedit`) is similar, but you will likely want to suffix the command with an ampersand (&) to run the editor in the background so the terminal can still be used at the same time:

```
gedit hello.c &
```
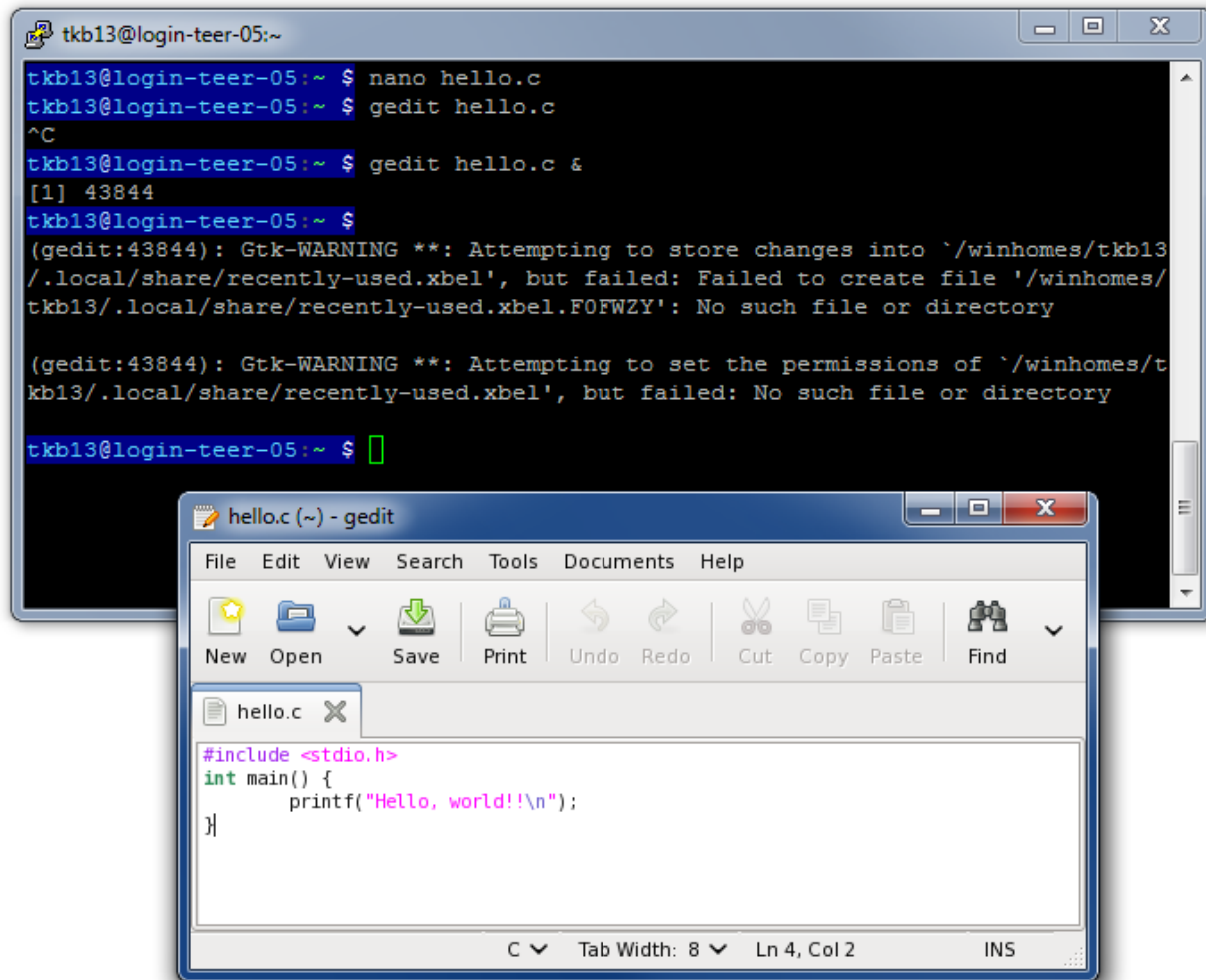

Alternative methods

This document describes the default, supported method for using the Duke Linux machines. Other methods are possible. Note that while alternative methods of accessing Duke resources can work quite well, the course staff will only be able to provide "best effort" support for these techniques.

Alternative 1: Local editor with your CIFS home directory

You can use ssh/PuTTY as described above to access a terminal, but using a local text editor to write code. This can be achieved by attaching your Duke home directory to your local computer over the network via the "CIFS" protocol (also known as "Windows sharing"). Note that this technique requires you to either be on campus or to use Duke VPN (which makes it like you're on campus). Duke OIT provides documentation on this here:

- [General OIT guidance](#).
- [Tutorial for Windows](#).
- [Tutorial for Mac](#).

Alternative 2: Local editor with SFTP synchronization

You can also use the SFTP protocol to synchronize local and remote files. SFTP can be used on any host that provides SSH access (such as login.oit.duke.edu). This can be achieved by using an editor with a built-in SFTP client (such as Notepad++ on Windows), or using a standalone SFTP client (such as WinSCP) to synchronize local files to the Duke Linux environment. You can also use command-line tools for the purpose, such as rsync and scp. This protocol works on and off campus; no VPN required.

3. Hello World

As a warm-up, your first task is to write a program that prints out "Hello World" or some other string of your choice. ("ECE/CS 250 is awesome" is another fine example.) First, login to a Duke Linux machine. Use the text editor of your choice to create a file called `hello.c` and then compile it with `g++` and run it:

```
g++ -o hello hello.c
./hello
```

The first line compiles `hello.c` into an executable program called `hello`, and the second line runs the program `hello`. The "-o hello" part of the first line tells g++ to create an executable called `hello`. By default, g++ would've otherwise created an executable called `a.out`. In the second line, you may wonder what the deal is with the "./" – that tells the terminal to look in the current directory for the file to run, which is necessary for running a program from the current directory[1], but not necessary for reading it or moving it or renaming

---

[1] The reason for this requirement is security. Imagine a malicious person put a program called "`ls`" in the current directory. When you type `ls`, you might run that program instead of the usual `ls` command. To disambiguate the situation, Linux requires you to be explicit when running a program from the current directory by prefixing it with "`./`".

it, etc. (Your current directory can be referred to with "." and its parent directory can be referred to with "..". So if you type "cd .." that'll take you to the parent directory.)

```
g++ -o hello hello.c
./hello
```

4. OPTIONAL: Additional Unix practice

You're going to be using Unix machines for your C programming, and thus you're going to need to be somewhat adept at working with these machines. If you'd like additional practice with Unix, you can work through the tutorials at the following website.

http://www.cs.duke.edu/~alvy/courses/unixtut/