# *COMP 161 - Lecture Notes - 02 - Shells, part 1*

## *Spring 2014*

Shotts' tutorial introduces the shell[1] but discusses how to access it as if you're accessing the shell of the computer in front of you. In this class, we'll be accessing the shell of a the departmental server on the second floor of the CSB. So, we'll need to figure out how to access the shell of that computer remotely using a Secure Shell[2] client. We'll also briefly discuss some common patterns in shell commands, the command to that you'll use to submit most of your assignments, and how to move files to and from the remote system.

[1] `http://linuxcommand.org/lc3_lts0010.php`

[2] SSH

### *The Shell*

As Shotts explains quite well, the *shell* is your command line interface with the Operating system[3]. We're using *bash*[4], but there are many other shells you could check out[5]. These shells are all available for any Linux or Mac OS-X based system. If you want to explore a similar shell system for Windows, then you can check out Microsoft's PowerShell[6].

[3] which is the interface to the hardware

[4] run the command *echo $SHELL* to see which shell you're running

[5] ksh,tcsh, and zsh

[6] `http://msdn.microsoft.com/en-us/library/dd835506(v=vs.85).aspx`

The real power of shells comes from the fact that they are *programming languages*[7]. More specifically, they are languages designed to let you write programs involving the activities carried out with the shell[8]. This means that using you shell language, you can likely automate most of the repetitive tasks you carry out on a regular basis. For example, I used a bash script I wrote in order to create all of your accounts, generate random initial passwords for them, and be sure other account related settings were where they needed to be. Without this script I would have needed to type something like five commands per account. For a class of $n$ students that $5n$ commands. So, even for modest sized classes, that script saves me from a lot of repetitive typing. Perhaps more importantly, it prevents me from making typos and inadvertently messing something up[9].

[7] or scripting languages

[8] typically called *scripts*

[9] The value of this is huge. We all make mistakes.

We're not going to get into shell scripts, but Shotts has a wonderful tutorial on them[10]. I highly recommend you check it out. If that leaves you wanting more, there are an insane number of *bash scripting* tutorials out there[11].

[10] `http://linuxcommand.org/lc3_writing_shell_scripts.php`

[11] Look around `http://tldp.org` for starters

### *Secure, Remote Access*

It is very common to need or want remote access to the shell of a computer you're not currently sitting in front of. Smart users want to do so in a way that ensures nobody can easily snoop on what they're doing. Thus, a *secure shell* is needed. The defacto way to accomplish secure remote access to the shell is through an SSH client-server

setup[12]. The computer to which you want access[13] must be running an SSH server program, and the computer with which you're connecting[14] needs an SSH client program.

If your client is running Linux or OS X, then you're in luck. You already have an ssh client available through your terminal. If you're running Windows[15], then you need to get a client. Thankfully, there is a wonderful, free SSH client for Windows called PuTTy[16]. The PuTTy ssh client is available on all campus computers[17].

I'll break down the process of logging in to a computer's shell with SSH using a CLI client and PuTTy. First, I want to point a few thing you'll experience the first time you log in so that you'll be ready for them. Once you know what's different the first time, we'll talk about what's going to happen the rest of the time.

[12] `http://www.openssh.com/`
[13] the server
[14] the client
[15] like all our computer labs do
[16] `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`
[17] W:\Apps\mathcs\putty\PUTTY.EXE. I'd make a shortcut to it on your desktop if I were you.

### *The first time you login*

A few things out of the ordinary happen the first time you login.

- The first time a client and server communicate, they must establish trust.
- On our servers, the first time you've ever used the account, you'll need to reset your password to a password that only you know.

A big part of security is trust. So the first time an ssh client logins to an ssh server it must establish trust. This means your client will ask you if you know the server that you're logging in to and if it should really connect to that server. A server is identified not by it's name or address but by a unique *key*[18]. So, your job is to tell your client that connections to the server with that key are trusted. Here's what that looks like in linux:

[18] very big number

```
The authenticity of host 'cs.monm.edu (74.39.212.103)' can't be established.
RSA key fingerprint is 49:04:96:71:d9:a3:44:fd:2b:b7:83:56:bf:91:b3:48.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'cs.monm.edu' (RSA) to the list of known hosts.
```

and in Windows with PuTTy.

**ADD IMAGE**

There's really **no reason anyone other than you should know your password**[19]. The account begins with a randomly generated password[20]. The first time you login to your account, you'll be forced to reset your password. There are two big things to note about this process:

[19] including system administrators like me
[20] given to you in class

1. When you're typing a password, *it looks like nothing is happening on the screen*[21]
2. When you're done, you'll be booted from the server and will have to log back in with the new password.

The process is straight forward if you *read and follow the directions*[22].

[21] this is always true and always weirds at least a few students out

[22] at lease a few people don't read the prompts and mess up their password. It's fixable, but easily avoidable by **reading the directions**

### *Linux and OS X*

Linux and OS X both come with a command-line SSH client installed. In order to access it you need to first launch your local *terminal*[23]. Once your local terminal is up and running the command is fairly straight forward. The most basic command dictates the port[24], username, and server[25] to which you're connecting. The complete command looks like: *ssh -p 22 user@server*. Here you see an example of me logging in from my Linux machine.

[23] this is the shell access Shotts talks about

[24] -p 22

[25] cs.monm.edu

```
jlmayfield4@dunkel ~ » ssh -p 22 mayfieldjl@cs.monm.edu
mayfieldjl@cs.monm.edu's password:
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.2.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
No mail.
Last login: Sun Jan 12 12:08:28 2014 from 74.39.212.94
mayfieldjl@church:~ $
```

As you can see, once you enter the command, you'll connect to the server and be asked for the password for the given username. Once you're done with your shell session, use the command *exit* to logout.

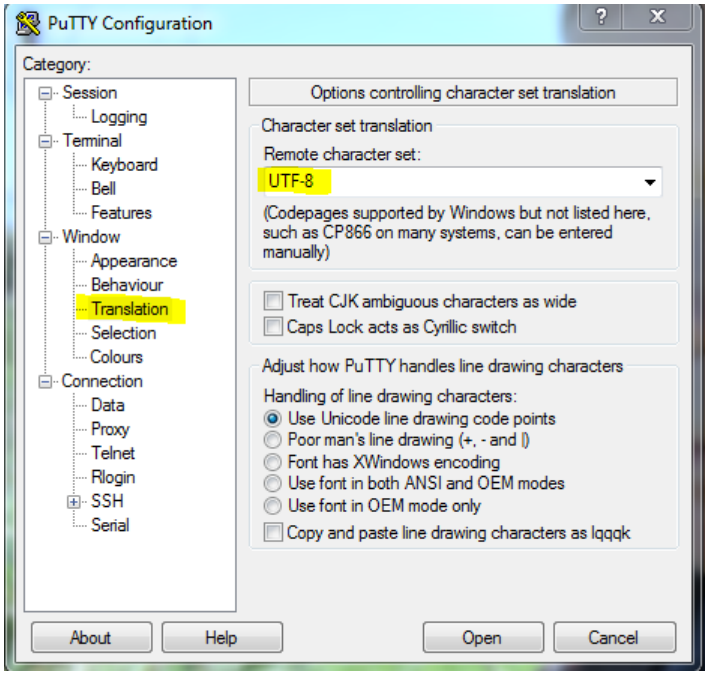
### *PuTTy (Windows)*

PuTTy is GUI based SSH client. On the surface it is very easy to get logged in, but it does have one little quirk you'll want to deal with. By default, PuTTy reads the letters coming from the server using a European standard. The result is that some of the letters get garbled[26]. So, to fix this we need to tell PuTTy to use UTF-8, the standard used by the server. You can either set this every time you launch PuTTy[27] or set it once and save your PuTTy settings. I'll show you how to do the later, and get logged in.

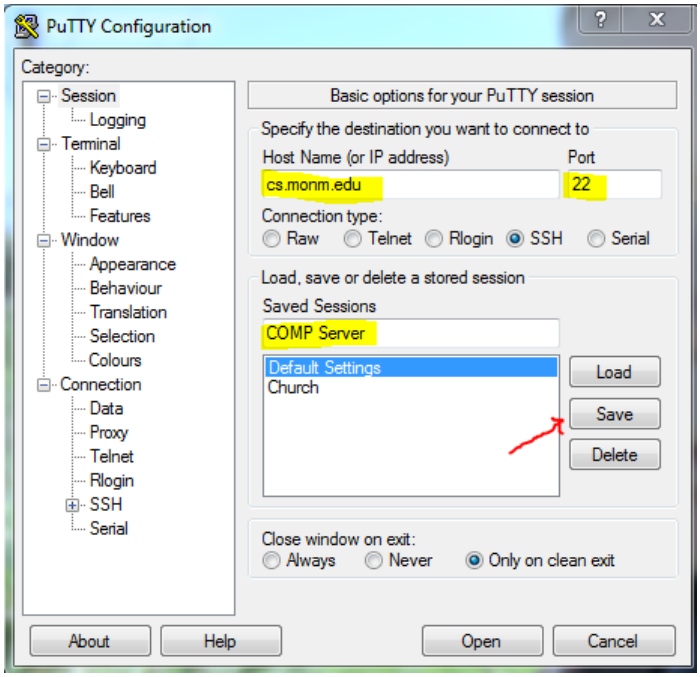
To set the encoding to UTF-8 you need to navigate the settings to *Window>Translation* and select *UTF-8* as the Remove character set.

[26] you'll probably see it first when you run the compiler for C++

[27] and waste a minute



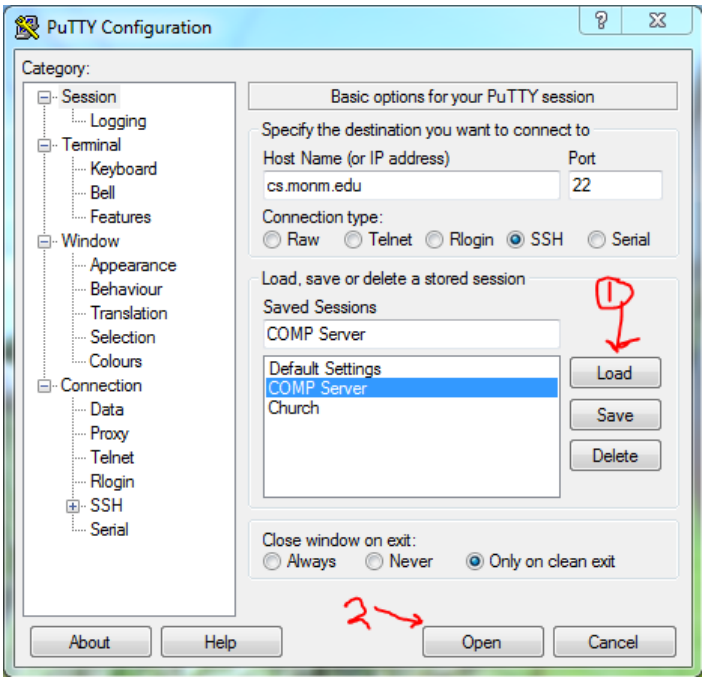

Once you've done that, navigate back to the session options, fill in the *Host Name* and *Port*, and then fill in a Saved Sessions name. With all of that filled in, you can hit save to save the encoding and session info for future use.



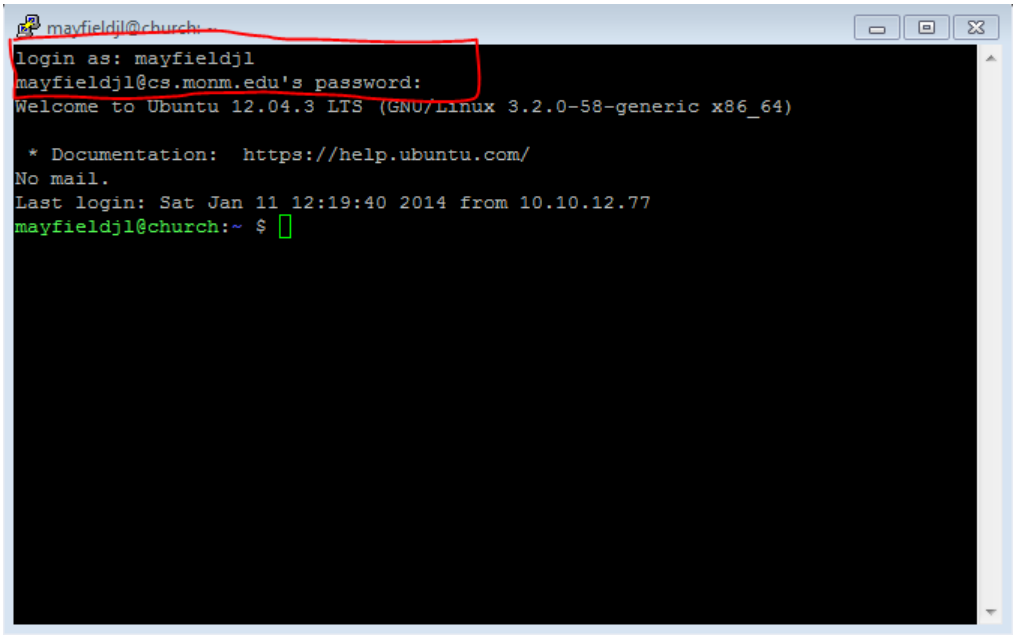

With the right encoding and server information saved, you can

now load your settings and open the connection[28]. Highlight the saved session name, hit *Load*, then hit *Open*.

[28] this is what you should be doing all but the first login. unless you like to waste a minute per login with a tasks that are easily automated...




You'll now be prompted for your username and password.




When you're all done with the shell, the command *exit* will log

you out of the server.

### *Submitting assignments with handin*

A script written for us by a former CS professor[29] here at Monmouth allows you to submit most of your assignments via the server. The name of the script[30] is *handin*. You'll work out its usage in the first lab. If you need a refresher, the following command will give you the help text for the command.

[29] Thanks, Don Blaheta!

[30] and the command

```
handin -h
```

### *Moving files*

Because of *handin*, you'll never really need to move files to and from the server. However, students regularly want to do this so I'll discuss a few ways to do this here. The traditional method is to piggy back on the SSH connection using *secure file transfer protocol*[31]. However, I personally use the cloud storage service Dropbox[32] which has a command-line client.

[31] sftp

[32] `http://dropbox.com`

#### *sftp and psftp*

If a computer is running an SSH server, then chances are it can handle sftp connections as well. An sftp client runs its own little CLI, so once you're logged in, there's a series of commands you'll use to move files. Many of them are the same as the shell commands you're learning, but this time there's a version for your local machine and another for the server[33].

[33] run the command *help* from within an sftp session to see the full list of commands

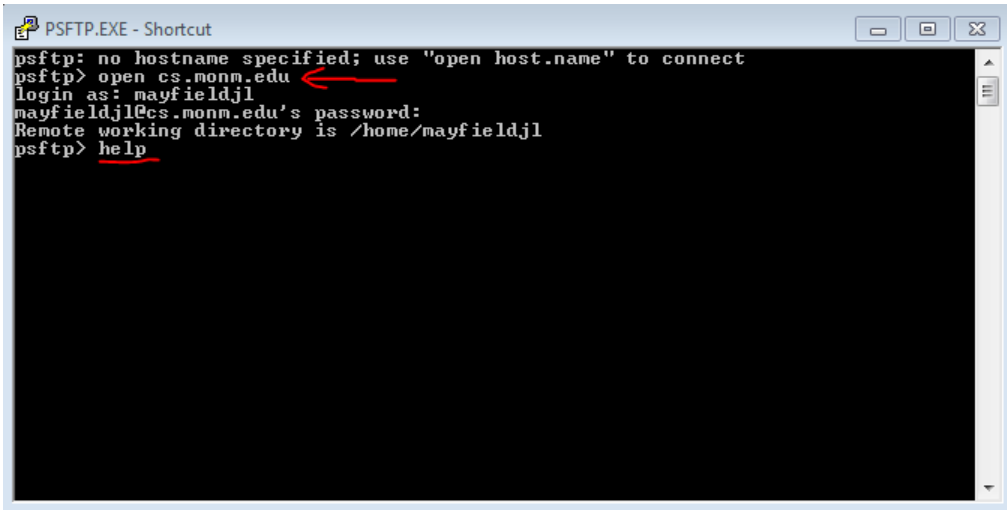
In Linux and OS X, there is a command-line client that you launch in pretty much the same way as ssh: *sftp user@server*.

```
jlmayfield4@dunkel ~ » sftp mayfieldjl@cs.monm.edu
mayfieldjl@cs.monm.edu's password:
Connected to cs.monm.edu.
sftp> help
```

In Windows, you can use the PuTTy SFTP[34] client, which can be downloaded[35] from the sample place you got PuTTy. Just launch *psftp.exe* and then follow the prompts to log on to the server.

[34] psftp

[35] and found on the campus network




Once you're logged in, it runs the same as the Linux client. Once again, use the *help* command to see the list of sftp commands.

### *Dropbox*

I'm a big Dropbox fan[36]. With a minimal amount of effort, it lets me share files between my personal computer, my work computer, the servers I use at work, and my smartphone. Those same file are also accessible via the web if I can't get to one of those devices.

If you want to go the dropbox route you'll need a dropbox account. If you need or want a dropbox account and want to do your professor a solid, signup via this link[37]:

`https://db.tt/XOjdfkS`

If you don't want me to get free storage with them, then feel free to sign-up from their website[38].

Once you have an account you need to install the Linux Command-Line client on the server.

1. Login to the server
2. Follow these *Install Dropbox via the command line* instructions: `https://www.dropbox.com/install?os=lnx`
   - Alternatively: `http://www.dropboxwiki.com/tips-and-tricks/install-dropbox-in-an-entirely-text-based-linux-environment`
3. Then look at these instructions on using the client: `http://www.dropboxwiki.com/tips-and-tricks/using-the-official-dropbox-command-line-interface-cli`

Once you're up and running, it's pretty much like having a folder that when you put files and folders in it, will sync them up with your dropbox. There are a few things to keep in mind with Dropbox:

[36] not a paid spokesperson

[37] I get free extra storage if you do

[38] `http://dropbox.com`

- You need to restart it every time the server reboots[39]
- You should not work out of your dropbox folder[40]. It'll probably get your in trouble.

[39] bound to happen a few times a semester
[40] It's a dropbox, not a workspace