Mastering EOS

Release 1.0

Sean Fisk and Ira Woodring

April 26, 2015

Mastering EOS is an initiative to produce better documentation for the Grand Valley State University School of Computer and Information Systems Exploratory Operating System Labs (GVSU CIS EOS Labs). It is intended to augment the original EOS documentation located on the School of CIS website.

Mastering EOS is written by:

- Sean Fisk, BS and MS in Computer Science from GVSU
- Ira Woodring, GVSU EOS Lab system administrator

Contents:

CHAPTER

ONE

INTRODUCTION

The Exploratory Operating Systems Labs (EOS Labs) are a collection of computer labs maintained by the GVSU School of Computing and Information Systems. Some are for general CIS student use, and others are specific to certain courses, labs, or applications. The labs within the EOS Lab umbrella include:

Name	Location	Purpose
Exploratory Operating Systems Lab (EOS Lab)	MAK A-1-171	studying operating systems; general CIS computing use
Architecture Lab (Arch Lab)	MAK A-1-101	studying computer architectures; general CIS computing use
Data Communications Lab (Datacomm Lab)	MAK 1-1-167	studying networking
Hardware Lab	MAK 1-1-105	studying hardware; multi-purpose

When addressing this collections of labs, we often refer to it simply as *EOS*. Though the EOS Lab is only one of the CIS department's computing labs, it was the original lab and other labs are largely based upon it. It is also the one most often used by the greatest variety of students.

1.1 Physical Access (Keycards)

The EOS Labs contain equipment specific to the CIS majors and are therefore closed to the general public. To access the labs, you are required to obtain a keycard. To be eligible for a keycard, you must be currently enrolled in a course which utiltizes the lab you'd like to access. After registration, visit the computing office in MAK C-2-100 to receive your card. A $25 deposit paid from your student account is charged upon receipt of the card. When the card is returned in acceptable condition, the deposit will be refunded. Please note that once the $25 deposit has been posted to your student account you must pay the balance, otherwise you will be charged a late fee. The late fee will not be refunded once you return the keycard.

To then gain entry to a lab to which you have been granted access, simply swipe the card at the reader next to the door to unlock it. In addition to room access, the card grants 24-hour access to Mackinac Hall. The courtyard door which is closest to the Architecture Lab possesses a reader which will open when the key is swiped. No other building doors are equipped with readers.

It is important that you do not allow anyone else to use your card. In addition, do not keep the card in your wallet, as the cards contain tiny hair-like wires that break easily when the card is flexed.

1.2 Computer Access (Credentials)

The EOS system uses separate user accounts from the general university computing infrastructure. EOS uses the same network ID given by GVSU's IT department, but authenticates against the CIS department's own LDAP server. The

accounts are not interchangeable and administrators for one account cannot reset passwords for the other.

If you are registered for one or more eligible CIS courses [#eligible-cis-courses], you qualify for an EOS account. On the Friday before the semester starts, an account will be automatically created for you and a temporary password sent to your GVSU email address. To activate your account, you need to log in to the system once, either at a physical machine or *remotely using SSH*, and follow these steps:

- Enter your username.
- Enter the temporary password you have been given.
- If the temporary was entered correctly, you will be asked for the Current Password again. Enter the temporary password again.
- If the two temporary passwords match, you will be asked to create a new password.

Note the following rules when creating a new password:

- Your password must be at least 7 characters.
- Your password must not be based on a dictionary word.
- Your password should not be all numbers. The system will accept such a password, but it is incredibly insecure. Additionally, the system will often prevent login with such a password.

Please take the time to memorize your password! Password resets are available by contacting Ira Woodring, but it often takes a day to get to it. Professors are also able to reset passwords via an SSH reset mechanism, though some are unaware of this mechanism.

CHAPTER

TWO

RULES AND PROCEDURES

It is important that some basic rules and procedures be established to help maintain the lab and aid in its shared use. To those ends, please be aware of the following guidelines:

2.1 Disk Space

You are given 8 gigabytes (GB) of disk space. Once you exceed this limit, you can no longer write new data to the filesystem. This often leads to being unable to login via a graphical session, as the desktop manager must be able to write to disk. You must then log in via a text-based console and delete files to make space.

The amount of disk space you have currently used may be checked with the `quota` command:

```
$ quota --human-readable
Disk quotas for user smithj (uid 1234):
     Filesystem   space   quota   limit   grace   files   quota   limit   grace
148.61.162.101:/home
                  9728M*  8192M  10240M            303k       0       0
```

The output of this command is two sets of four columns. The first four refer to physical disk space, while the second four refer to the number of files. The columns have the following meanings:

Column	Meaning
space	The amount of disk space you are currently using.
files	The number of files in your home directory.
quota	The soft limit for space or files. You may exceed this for the time listed in the grace period.
limit	The hard limit for space or files. You may never exceed this limit.
grace	The amount of time for which you can exceed the soft limit.

For the previous output, the user has a soft limit of 8192 MiB / 1024 MiB/GiB = 8 GiB and a hard limit of 10240 MiB / 1024 MiB/GiB = 10 GiB. The user is currently using 9728 MiB / 1024 MiB/GiB = 9.5 GiB of disk space, and is over quota (indicated by the $\star$). The output is expressed in mebibytes, which might be different than the megabytes to which you are accustomed. There is no grace period set, so the user is able to write files until reaching the hard limit. There is also no limit on the number of files the user can create, only a limit on the amount of space consumed.

2.2 Copyrighted Material

Any files may be stored in your home directory. This includes games, movies, and music. However, allowing others to transfer copyrighted material may constitute a copyright infringement.

These incidents are usually caused by unintentional permissions issues or deliberate misuse of the system. Whatever the cause, copyright infringement is a violation of school policy.

These cases are taken very seriously. Your user account will be terminated immediatly upon discovering the infringement. Additionally, this offense is against our school's honor code, so you may be expelled or even face criminal proceedings.

Suffice to say, please be very careful when dealing with copyrighted materials within the EOS system. Be familiar with how permissions work, and take the time to set them correctly.

2.3 Food and Drink

You are allowed to eat and drink in the labs — these labs were made for you and we want them to be as comfortable and useful as possible. Many restaraunts in the area deliver to the labs. However, it is your responsibility to clean up after yourself. If the lab becomes too messy, policies to limit food and drinks will be instituted.

2.4 Overnight Parking

Even if you have a parking pass, it is still neccessary to obtain a special permit to park overnight. These permits are granted by Campus Safety. Parking overnight without one of these permits can result in a ticket or towing.

2.5 Living in the Lab

It should go without saying (but hasn't in the past) that you cannot live in the lab. People have been found living in the lab for brief periods of time between leases, etc. We must note that this is not only a huge safety violation but is illegal. If caught living in the lab, you will be removed and Campus Safety notified.

2.6 Malicious Activity

The infrastructure provided by the EOS Labs includes very powerful tools that can be used to secure applications and network infrastructure. Unfortunately, these tools may also be used for malicious purposes. We provide these tools for you to learn to defend future systems it may be your job to secure. Under no circumstances should these tools be used to attack other students, machines, or entities. We do not provide these resources without reasonable oversight as to their use, and those using them illegally will be noticed and face strong consequences — possibly including removal from the university and criminal charges.

2.7 Games

We allow playing games in the lab. However, if you are playing games and all machines are currently in use, please be polite and yield your machine to students needing to complete coursework. Failure to do so may result in suspension of your account, or all games being removed from the system.

CHAPTER

THREE

USING THE COMMAND LINE

What is a shell? A shell is an interface to a computer's operating system. You are probably already familiar with at least one shell: your machine's *graphical shell* , also know as a *graphical user interface* (GUI). This shell allows you to interact with windows, menus, buttons, and many other widgets using the keyboard, mouse, and other peripherals.

Although there are shells of many types, the other most common type of shell is a text shell, also known as a *command-line interface* (CLI). Text shells are typically command interpreters: they wait for you to type commands, then execute them when you press `Enter`. Interaction typically happens with the keyboard only. Unlike most graphical shells, text shells also excel at batch execution.

Although text and graphical shells share some capabilities, each type of shell has strengths and weaknesses. For many tasks, text shells represent a faster and more concise way to interact with the operating system. A successful computer scientist will learn to be comfortable using both graphical and text shells.

Please see the Wikipedia article on shells for a more detailed explanation on the history and types of shells, as well as the articles on the Command-line interface and Graphical user interface.

Read on to continue your adventure with the shell!

Please make sure to read about interactive use of the shell before reading about shell scripting, as many interactive concepts are also used while scripting.

3.1 Command-Line Basics

The absolute basics of the command line are currently not covered in Mastering EOS because they are covered in the GVSU CIS curriculum. Please visit the EOS Lab Tutorials to learn about command-line basics. This is a great place to learn, but be aware that some topics and utilities may be outdated due to system updates. In the future, we hope to update the EOS Lab Tutorials and merge them with Mastering EOS!

Another great guide on command-line basics is The Command Line Crash Course by Zed Shaw.

3.2 Available Shells

3.2.1 Bash

GNU Bash (*Bourne-again shell*) is the default shell on many Unix-like systems, including most GNU/Linux distributions and OS X. It is a full-featured shell with many capabilities. Although other shells exist, we recommend first learning Bash because of its ubiquity and popularity. In addition, by learning Bash, you will have a better understanding of other shells if you choose to use them.

The GNU Bash Manual is the comprehensive guide to all of the workings of the shell. It is a reference manual, so it does not provide a lot of examples. However, it is clear and concise and recommended reading for Bash users. It is also available for the version specific to EOS by typing `info bash` on any EOS computer.

> **Warning:** OS X is notorious for including a very outdated version of Bash as the default shell. If you use Bash under OS X, you are recommended to install a newer version using a package manager such as Homebrew.

3.2.2 sh

The Bourne shell (also known by sh) is one of the original Unix shells. It is the precursor to many modern shells, including *Bash* and *Zsh*. Although it is still used for scripts, its use as an interactive shell is primarily historical.

The sh interpreter is specified by POSIX in the sh and Shell Command Language pages. On many systems, the command `sh` actually activates *Bash* in an sh-emulation mode (try `sh --version`).

3.2.3 Zsh

Z shell (Zsh) is a very capable shell based on the Bourne shell with too many features to count. Although it is not directly related to *Bash*, those who are proficient in Bash will undoubtedly feel comfortable using Zsh. In many ways, Zsh can be seen as having a superset of Bash's features.

3.2.4 fish

Another interesting shell is fish, the friendly interactive shell. fish aims to be a modern shell that works well out-of-the-box. It is especially focused on interactive features such as on-the-fly syntax highlighting, auto-suggestions, and auto-completion. However, it also focuses on having a clean, consistent command interface.

fish is not installed by default on EOS, so you will have to install using *Linuxbrew* or compile it manually.

3.2.5 Other Shells for Unix-like Systems

Two other traditional shells are Tcsh and KornShell. Tcsh (TENEX C SHell) is a modern shell based on the C shell. Kornshell is an older shell developed at AT&T Bell Laboratories based on the best features of the Bourne and C shells. For an overview of Unix shell history, and comparison of these shells as well as those listed above, see the UNIX shell differences article.

The concept of a shell is neither complex nor set in stone. Many projects have striven to create alternative shells. These are usually based on enhanced interaction with a specific programming language or environment, yet retain some of the usage and concepts of traditional shells:

Shell	Environment
Eshell	Emacs
Zoidberg, psh	Perl
IPython	Python
ShellJS	NodeJS
Pry	Ruby

For a more complete list of shells, see Wikipedia's shell comparison and article on Unix shells. For a side-by-side comparison of the syntax of different shells, see Hyperpolyglot's Unix shells section.

3.2.6 Which Shell?

When deciding which shell to use, it is important to consider both types of use: *interactive work* and *scripting*. When choosing to use a shell interactively, you are really making that decision only for yourself. However, when deciding which shell to use for a script, you are choosing that shell for the script's audience. That could be for yourself only, for the members of your project, or for the world if the script is part of a public project. It is also obviously an advantage to use the same shell for both interactive and script work.

For interactive work, we recommend starting with *Bash*. Bash is currently the default shell on OS X and most GNU/Linux distributions (including EOS systems). Because Bash is prevalent on many systems and is a full-featured shell, we recommend becoming comfortable with it first.

Bash is a great shell even for power users. However, there are other shells that offer more built-in features and opportunity for extensibility. Once you have become proficient in Bash, we recommend trying *Zsh*. Zsh offers a similar experience to Bash but has even more features, plugins, and frameworks.

For those who have tried *Bash* or *Zsh* and are looking for something more modern and different, *fish* is a good choice. However, be aware that because it is less popular, less programs, plugins, and frameworks are compatible with it.

When it comes to scripting, we once again recommend using *Bash*. More exotic shells such as *Zsh* and *fish* are often not found on systems, meaning that the shell has to be installed before the script can be run. However, Bash's prevalence means that it is installed by default on many systems, making it a good choice for scripting. If scripting features of more advanced shells are needed, we recommend instead moving straight to a true programming language such as Python or Ruby.

You may find some recommendations to use *sh* for scripting because it is specified by POSIX while Bash is not. Because Bash is widely available and offers significant benefits over sh, we recommend that you script on the EOS machines using Bash. You will certainly get differing recommendations on this, but we feel Bash's features and availability justify its use for scripting.

3.2.7 Windows Shells

While Unix-like operating systems are known for full-featured shells, Windows also has quite a few shells available.

Command Prompt

Command Prompt (or cmd.exe) is the classic Windows shell, inherited mostly from the DOS shell COMMAND.COM. It is a relatively limited shell, with only a basic set of commands. For those looking for a daily shell on Windows, you are recommended to look elsewhere.

Windows PowerShell

Windows Powershell is a relatively new advanced shell for Windows released by Microsoft in 2006. Different versions of PowerShell are distributed with Windows starting from Windows 7. PowerShell is based on the .NET Framework, but is inspired by Unix shells like *Bash*.

Since it has been released, PowerShell has become increasingly popular with developers looking for a decent shell on Windows, and a large number of scripts and plugins have been developed by the community. This makes it a great choice for your Windows shell.

If you are interested in learning PowerShell and already know Bash, see Hyperpolyglot's OS automation section for a side-by-side comparison of POSIX shell, Command Prompt, and PowerShell features.

Tip: Although Microsoft created a great shell with *Windows PowerShell*, they did nothing to replace the terrible

Win32 console provided with Windows, which runs both *Command Prompt* and *Windows PowerShell* by default. For a better console on Windows, we recommending checking out ConEmu or Console 2.

Cygwin

Cygwin is a project which provides a Unix-like environment for Windows, including shells such as *Bash*. If you are looking for many of the tools and libraries you use on Unix-like operating systems, but on Windows, Cygwin is a great choice.

3.3 Interactive Shell Use

Most shells may be used in one of two modes: interactive or batch. In interactive mode, the shell waits for you to type commands, then evaluates them when you press `Enter`. This is called a Read-eval-print loop. In batch mode, the shell runs commands listed in a script file, just like a regular interpreted programming language like Python or Ruby.

Most of the following features are usable in both interactive or batch mode. Some features are more useful in one than the other. For example, *Aliases* are seldom used in scripts, while the *Shebang* is never used interactively.

Warning: These examples are expected to be run with *Bash* 4 on EOS. Most of them will also work in a POSIX-compatible shell (*sh*) and *Zsh*. Bash 4 is the default shell, that is, the shell that you are presented with when starting a terminal through a GUI session or logging in through SSH.

3.3.1 Aliases

An alias can be a shortcut to typing a frequently-used command. For example, if you visit a certain directory often:

```
$ alias hw='cd ~/classes/cis452/hw/01'
$ pwd
/home/smithj
$ hw
$ pwd
/home/smithj/classes/cis452/hw/01
```

After running the `alias` command, `hw` will now switch to that directory when typed.

An alias can also be used to remember a lesser-used command. For example:

```
$ alias extract='tar -xvf'
$ extract bash-4.3.tar.gz
bash-4.3/
...
```

Notice that we can still pass arguments to the alias. In fact, it is almost exactly like typing those characters at the command-line.

3.3.2 Built-Ins

For a typical command, the shell finds an executable file on the system and executes it. An example is the `ls` command, which is an executable on the system, typically located at `/usr/bin/ls`:

```
$ /usr/bin/ls mydir
file1.txt  file2.txt
```

However, some commands are actually part of the shell, called *shell built-ins*. A shell built-in differs from a normal command in that it typically operates on things *within the shell itself*, rather than just for one command. A prime example is `cd`, which changes the current working directory within the shell, affecting all subsequent commands:

```
$ pwd
/home/smithj
$ cd mydir
$ pwd
/home/smithj/mydir
```

To find out if a command is an executable or a shell built-in, use the `type` command:

```
$ type ls
ls is /usr/bin/ls
$ type cd
cd is a shell builtin
$ type type
type is a shell builtin
```

The `type` command is also a shell built-in!

To see all shell built-ins, consult the Bash manual.

3.3.3 Quoting

In certain cases, Bash will interpret the command you give it in a different way than you might expect:

```
$ ls directory name with spaces
ls: directory: No such file or directory
ls: name: No such file or directory
ls: spaces: No such file or directory
ls: with: No such file or directory
```

This is because Bash splits the command line it is given based upon spaces, and passes each argument to the program in question. To get Bash to interpret the spaces as an actual character, use single quotes:

```
$ ls 'directory name with spaces'
file1.txt  file2.txt
```

Single quotes remove any special meaning from all the characters inside them. *Always use single quotes when the characters inside the quotes should not be interpreted by the shell.*

Double quotes may be used to expand only the meaning of certain shell metacharacters. They are most often used for variable substitution. For example, to print your current working directory:

```
$ echo "My current directory is: $PWD"
My current directory is: /home/smithj/directory name with spaces
```

Double quotes are very frequently used, but it is easy to use them incorrectly. Know their behavior and test your commands with different values to make sure they are behaving correctly.

3.3.4 Environment Variables

A process' environment is a mapping of key-value pairs possessed by every running process in the system. They are typically used to modify the behavior of programs. You are probably familiar with some common ones; for example, `PATH`, `EDITOR`, and `PWD`. Environment variable names on Linux are case-sensitive and can contain most characters, although by convention they are usually named in all caps with words separated by underscores.

Bash supports manipulation of the environment variables for programs it runs (*child processes*) in various ways. To see what environment variables Bash is giving to its child processes, use the `env` program:

```
$ env
...
```

You should see a list of `VAR=value` printed. These are the variables and values Bash is giving to its child processes.

In Bash, regular variables can be set in any shell session rather easily:

```
$ GVSU_CS='Computer Science'
$ GVSU_IS='Information Systems'
```

However, these variables are only seen by commands built into the shell. After settings these variables, verify this with `env`:

```
$ env | grep -E '^GVSU'
```

You should see no output. However, you can instruct Bash to send these variables to child processes by using the `export` built-in:

```
$ export GVSU_CS
$ env | grep -E '^GVSU'
GVSU_CS=Computer Science
```

Notice that `GVSU_CS` has been sent to the program, but `GVSU_IS` has not. When the value of the variable is changed, the value sent to the child processes is also changed. It does not need to be exported again:

```
$ GVSU_CS='Cool Stuff'
$ env | grep -E '^GVSU'
GVSU_CS=Cool Stuff
```

To see all variables in Bash marked for export, use the following command:

```
$ declare -x
...
declare -x GVSU_CS="Cool Stuff"
...
```

Here some other shortcuts to do with environment variables:

```
$ export GVSU_CS='Cool Stuff' # Set and export in one line
$ GVSU_CS='Not my major' env | grep -E '^GVSU' # Set for one command only
GVSU_CS=Not my major
```

Example

A real-life example of an environment variable in use is the `EDITOR` environment variable. Various programs use this variable to determine what editor they should use when a file needs to be edited. An example is the `crontab -e` command. Try the following commands:

```
$ EDITOR=nano crontab -e # type Ctrl-X to exit
$ EDITOR=vim  crontab -e # type :q<Enter> to exit
```

You should see each respective editor open up when the command is run!

Note: Cron is not set up for GVSU students; this command just edits your Cron configuration file. You do not need to modify it.

3.3.5 Pipelines

The concept of a pipeline is central to the philosophy of Unix-like operating systems. A pipeline is typically used to combine the capabilities of multiple programs to perform a task. This is accomplished by sending the output of each program to the next program in the series. Pipelines can be formed easily using the shell with the vertical bar (aka pipe) character:

```
$ ls | grep -F cis162 # Look for files/directories with 'cis162' in the name
```

Example: Paging

Often times, a command will produce output which fills the screen. The `dmesg` command reports kernel events and frequently has lots of output:

```
$ dmesg
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.10.0-123.20.1.el7.x86_64 (builder@kbuilder.dev.centos.org) (gcc versio
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-3.10.0-123.20.1.el7.x86_64 root=UUID=967c544c-5
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009d7ff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009d800-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000e0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000001fffffff] usable
...
```

If you want to see anything besides the last screenful of output, one option is to use the scrollback feature provided by most terminal emulators (just scroll up). However, scrollback is limited and does not work under multiplexers such as tmux (although tmux has its own scrollback buffer). Another option is to use a pager. A *pager* is a program that allows you to browse and scroll through the output of a command, much like opening the output of the command in an editor. Two default pagers available on most Unix-like systems are `less` and `more`. While `more` allows only paging forward, `less` allows scrolling forward and back, making `less` the preferred choice for most tasks. To page the output of `dmesg`, type:

```
$ dmesg | less
```

You will be taken into a text-based user interface in which you can use arrow keys, Vim keys, or Emacs keys to scroll around. Press `q` to quit.

Example: Filtering

Sometimes, the entire output of a program is not needed. One of the most common uses of a pipeline is to filter output of a command using grep. For example, to find all files with a `.png` extension in the current directory:

```
$ ls
a.jpg
b.jpg
c.jpg
d.png
e.png
f.png
g.gif
h.gif
i.gif
$ ls | grep '\.png$'
d.png
e.png
f.png
```

`grep` only prints the lines that match the given regular expression. For more information on `grep` and the regular expression syntaxes it supports, see the man page.

Example: Extraction

While `grep` excludes certain lines based on a pattern, sometimes we wish to filter or exclude based on different criteria. Let's attempt to print the last 10 people who logged on to this computer.

To begin, the `last` command will output a "table" of previous logins to the machine on which you are currently working:

```
$ last
smithj   pts/0        148.61.121.9     Wed Mar 11 14:30   still logged in
smithj   pts/1        148.61.121.9     Wed Mar 11 13:00 - 14:00  (01:00)
woodriir pts/0        :0.0             Wed Mar 11 12:00 - 12:30  (00:30)
millers  pts/1        148.61.121.9     Wed Mar 11 11:30 - 11:45  (00:15)
woodriir pts/0        :0.0             Wed Mar 11 11:00 - 11:15  (00:15)
woodriir pts/0        :0.0             Wed Mar 11 10:00 - 10:15  (00:15)
...

wtmp begins Tue Nov 25 09:26:28 2014
```

The first column is the username. We want to extract the usernames for further use in the pipeline. That's doable, but we have a two-line footer that we first need to remove. We can use the `head` utility to print out all of the lines *except for the last two*:

```
$ last | head --lines -2
smithj   pts/0        148.61.121.9     Wed Mar 11 14:30   still logged in
smithj   pts/1        148.61.121.9     Wed Mar 11 13:00 - 14:00  (01:00)
woodriir pts/0        :0.0             Wed Mar 11 12:00 - 12:30  (00:30)
millers  pts/1        148.61.121.9     Wed Mar 11 11:30 - 11:45  (00:15)
woodriir pts/0        :0.0             Wed Mar 11 11:00 - 11:15  (00:15)
woodriir pts/0        :0.0             Wed Mar 11 10:00 - 10:15  (00:15)
...
```

Perfect! Now, we need to remove all the information besides the username. We can use the `cut` utility to extract it:

```
$ last | head --lines -2 | cut --fields 1 --delimiter ' '
smithj
smithj
woodriir
millers
woodriir
woodriir
...
```

Looks good. Now, let's use the `uniq` utility to delete repeated lines, effectively collapsing repeated logins:

```
$ last | head --lines -2 | cut --fields 1 --delimiter ' ' | uniq
smithj
woodriir
millers
woodriir
...
```

Great! Lastly, the output is rather long. We only want to see the last 10, so we can use `head` again to truncate the results:

```
$ last | head --lines -2 | cut --fields 1 --delimiter ' ' | uniq | head
smithj
woodriir
millers
woodriir
reboot
millers
smithj
millers
smithj
reboot
```

That's it! See the manual pages for each of these utilities to learn more about them.

Example: Advanced

Although the last example was neat, pipelines need not stop there. You can chain many programs together to create a new tool which does something uniquely useful. For example, we can combine various tools to find all lunch items served at Fresh Food this week, highlighting all items involving chicken. First, write a file that will be used in the pipeline, and set a variable:

Important: This example works only with Bash 4 (the default shell on EOS). Ensure you are running Bash 4 with:

```
$ echo $BASH_VERSION
4.2.45(1)-release
```

The first number should be 4.

```
cat <<'EOF' >/tmp/fresh-menu.xsl
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <xsl:for-each select="//a[@class='recipeLink']">
            <xsl:value-of select="."/>
            <xsl:if test="position() != last()">
                <xsl:text>&#xA;</xsl:text>
```

```
            </xsl:if>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
EOF
readonly URL='http://www.campusdish.com/en-US/CSMW/GVSU/Menus/FreshFoodCompany.htm'
```

Next, run the pipeline, which pages the menu items, one per line, with chicken items highlighted!

```
wget --quiet --output-document - "$URL"                 | # download web page
  xsltproc --html /tmp/fresh-menu.xsl - 2>/dev/null     | # parse HTML
  tail --lines +2                                       | # trim initial XML declaration
  sort --unique                                         | # sort, collapse unique items
  sed 's/&amp;/\&/g'                                    | # replace '&amp;' with '&'
  while read -r line; do                                  # embedded shell loop
    line=${line,,}                                        # make lowercase
    line=($line)                                          # convert to array
    echo ${line[@]^}                                      # capitalize each word
  done                                                  | # loop: all-caps to title case
  tee lunch-menu.txt                                    | # write to file
  grep -E --ignore-case --color=always '.*chicken.*|$' | # highlight chicken dishes
  less --RAW-CONTROL-CHARS                                # page the output
```

Once you are done, press `q` to exit the pager. You should also see that the lunch menu items were written to the file, `lunch-menu.txt`, in the current directory:

```
$ less lunch-menu.txt
```

As you can see, pipelines can be used to accomplish any number of tasks. Although they are not always the solution, they are a great choice when batch processing is needed.

Pipelines can also be created in most programming languages. Languages such as Python and Ruby offer facilities to easily create pipelines, and most pipeline implementations (including most shells) are implemented using the POSIX C API calls fork, exec, and pipe.

Enjoy creating your own pipelines!

3.3.6 Redirection

Many commands provide the option to read input from and write output to files. For example, a C compiler:

```
$ cc -o main main.c
```

In this case, `main.c` is the C source code input file while `main` is the output executable.

However, many commands have no option to specify an input or output file. For example, back to the most basic: `ls` features no way to send its output to a file:

```
$ ls # outputs to terminal
a.jpg  b.jpg  c.jpg  d.png  e.png  f.png  g.gif  h.gif  i.gif
```

And the *write* command offers no option to read from a file:

```
$ write smithj # waiting for input
```

However, this does not mean that it is not possible to write to or read from files using these commands!

Enter redirection. Redirection is a shell feature which allows you to send output of a command to a file or send the contents of a file as input to a command.

Basic Redirection

Output redirection is accomplished with the greater-than sign:

```
$ ls >myfiles.txt # no output; output was written to the file
```

The file `myfiles.txt` will now contain the usual output of `ls`. Input redirection is accomplished using the less-than sign:

```
$ write smithj <hello-john.txt # does not wait for input
```

Both input and output redirection can be used simulatenously. To find all words which have the letters "eos" in them, and write them to a file:

```
$ grep --fixed-strings --ignore-case eos </usr/share/dict/words >eos-words.txt
```

Tip: Many programs including `grep` offer the option to read from file. So the previous example could also be written without input direction using:

```
$ grep --fixed-strings --ignore-case eos /usr/share/dict/words >eos-words.txt
```

Far fewer commands support writing to an output file, making output redirection the more frequently-used feature.

Error Redirection

So far, we have only talked about the basic input and output streams, stdin and stdout. However, there is one more standard stream, stderr, to which well-behaved programs will write error messages. When redirecting output, these error messages will not be redirected. For example, we can count the number of lines in the password file file and write that to a file using `wc`:

```
$ wc --lines /etc/passwd >num-lines.txt
```

No output is written to the terminal. However, if we try to access the shadow file, which has different permissions:

```
$ wc --lines /etc/shadow >num-lines.txt
wc: /etc/shadow: Permission denied
```

This will create an empty file `num-lines.txt` and write the error to the terminal.

In many cases, this is the desired behavior, since you will be notified of errors immediately when they happen. However, there are times when you would like to log both errors and output. This can be done by using the stderr file descriptor as such:

```
$ wc --lines /etc/shadow >out.txt 2>err.txt
```

It is also possible to combine both stdout and stderr by redirecting stderr to the stdout file descriptor:

```
$ wc --lines /etc/shadow 2>&1 >out-and-err.txt
```

Tip: In *Bash* 4 and *Zsh*, a shortcut to this syntax is:

```
$ wc --lines /etc/shadow &>out-and-err.txt
```

Discarding Output

Sometimes a command produces unnecessary output that is not useful. For example, to look for TXT files in the `/var` directory:

```
$ find /var -name '*.txt'
...
```

Because `find` tries to look in all subdirectories, you will likely see an avalanche of *Permission denied* errors. To eliminate this from the output, we can discard all `find` errors by sending them to the null device:

```
$ find /var -name '*.txt' 2>/dev/null
...
```

The output should now be much more reasonable, and not include any *Permission denied* errors!

> **Warning:** Be aware that redirecting stderr to `/dev/null` discards *all* error messages, not just *Permission denied* errors. For example, if the `/var` directory did not exist (unlikely, but possible), the error message reporting that would not be shown.

Appending Files

When using output redirection, the destination file is truncated (contents are erased) before writing the output of the command. To append to the file instead of truncating, use `>>`:

```
$ echo 'this command truncates the file' >out.txt
$ echo 'this command appends to the file' >>out.txt
```

3.3.7 Path Manipulation

Using the shell, it is possible to invoke programs by typing the full path to the executable file:

```
$ /usr/bin/ls mydir
file1.txt  file2.txt
```

However, this gets unwieldy quickly. Fortunately, the operating system (which the shell uses) provides a feature that can be used to address this issue, yielding the more normal-looking:

```
$ ls mydir
file1.txt  file2.txt
```

This feature is the the `PATH` environment variable, and it is a very important concept on Unix-like systems, both for interactive and scripted commands. Despite its importance, the concept is rather simple: the `PATH` environment variable contains a list of paths to search for executables.

The `PATH` variable contains a list of paths separated by colons (`:`). When instructed to do so, the shell searches through these paths looking for a executable with the given relative path. You can view your `PATH` with:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

Your output will probably contain more paths, and trying to decipher them from a long string can be headache-inducing. For a nicer output, instruct Bash to replace the colons with newlines:

```
$ echo "${PATH//:/$'\n'}"
/usr/local/bin
/usr/bin
/usr/local/sbin
/usr/sbin
$ alias path-print='echo "${PATH//:/'$'\n''}"' # create an alias (nasty quoting)
```

Tip: *Zsh* users can use one of:

```
$ (IFS=$'\n'; echo "${path[*]}") # zsh maintains the array $path as a mirror of string $PATH
/usr/local/bin
/usr/bin
/usr/local/sbin
/usr/sbin
$ alias path-print='(IFS='"$'\n'"'; echo "${path[*]}")' # similar to the Bash technique
$ echo "${PATH//:/\n}"
/usr/local/bin
/usr/bin
/usr/local/sbin
/usr/sbin
$ alias path-print='echo "${PATH//:/\n}"'
```

When you type `ls`, the shell searches for an executable named `ls` in each of the directories listed in the `PATH`, starting from the top. To confirm where Bash is finding `ls`, use the `which` command:

```
$ command which ls
/usr/bin/ls
```

Note: Some shells (e.g., *Zsh*) offer `which` as one of a number of *Built-Ins*. We use `command which` instead of just `which` to access the actual executable file for consistent output.

Now that you know how the `PATH` works, and how to view it, you can manipulate it for your own use. Let's add a script which implements our own version of `ls` that "emphasizes privacy", overriding the default version. We'll put this script in `~/bin`, which is the conventional location for a user's personal scripts:

```
$ mkdir -p ~/bin # create the scripts directory
$ cat <<EOF > ~/bin/ls # write the script
#!/usr/bin/env bash
echo 'Nothing to see here. Move along.'
EOF
$ chmod +x ~/bin/ls # make the script executable
```

You can now run this script with `~/bin/ls`. However, just typing `ls` results in the real `ls` being run:

```
$ ~/bin/ls
Nothing to see here. Move along.
$ ls
file1.txt  file2.txt
```

To allow our version of `ls` to be run, we need to add our `~/bin` directory to the front of the `PATH`. We can do that with:

```
$ export PATH=~/bin:"$PATH"
$ path-print
/home/smithj/bin
/usr/local/bin
/usr/bin
```

```
/usr/local/sbin
/usr/sbin
```

Note: Exporting the `PATH` variable is not necessary for this example to work. However, in most cases it is desirable to also give the value of `PATH` to any scripts run from the shell, in addition to using it within the shell itself.

Now, running plain `ls` runs our version:

```
$ ls
Nothing to see here. Move along.
```

Success! However, the `~/bin` directory will not be on the `PATH` for future shells. To make this change permanent, add it to your `.bash_profile`:

```
$ echo 'export PATH=~/bin:"$PATH"' >> ~/.bash_profile
```

The `export` line should now be the last line of your `.bash_profile`. Now follow the directions for *Applying the Changes*. `ls` should now permanently run our version!

Important: Help! Give me my `ls` back!

After following this example, to restore the `ls` command to its former glory, run:

```
$ rm ~/bin/ls
```

This will remove the modified `ls` example script. You can also always access the real `ls` by running:

```
$ /usr/bin/ls
file1.txt  file2.txt
```

3.3.8 Startup Files

The behavior of Bash can be customized by modifying its *rc files* (runtime configuration files). These are files containing Bash commands which are run within the current shell session at different points throughout the session.

Although there are more files which are run, two of the most important Bash rc files are `.bashrc` and `.bash_profile`. There is a lot of confusion about when these files are run. However, for an interactive Bash session [1], the answer is simple:

- `.bash_profile` is run for login shells.
- `.bashrc` is run for non-login shells.

So what is a login shell? A login shell is a shell that is run when you first log into the computer. For example, when you SSH into EOS, you have started a login shell. All subsequent shells started from that shell are non-login shells (unless otherwise specified by the `--login` option). Desktop environment terminal emulators (such as GNOME Terminal) typically start non-login shells by default (though this can be configured otherwise).

Here is a table describing where your shell customizations should go:

[1] Handling of startup files for non-interactive (also known as *script* or *batch*) sessions are more complex, and there is more variation between shells in how they are handled. For more information on which files are run in which scenarios, consult this blog post for an excellent table and flowchart.

Customization	File
Exported variables	`.bash_profile`
Non-exported variables	`.bashrc`
Aliases	`.bashrc`
Functions	`.bashrc`
Key bindings	`.bashrc`
Umask	`.bash_profile`
Shell Prompt (falls under non-exported variables)	`.bashrc`
Path Manipulation (falls under exported variables)	`.bash_profile`

Your startup files contain the heart of your shell customizations. Almost any command that you would run interactively or in a script can go in your startup files.

Example Startup Files

Here are some example startup files which illustrate many of the features mentioned in this chapter. Although these are a good example, you are recommended to use these only as a starting point for your own personal startup files. The path manipulation is useful if you are using software installed through *Manual Installation* or *Linuxbrew*. You can read more about it in *User-level Hierarchies*.

```
# .bash_profile
# Run for interactive Bash login shells

# Set umask for more privacy. Child processes inherit the umask from
# parent processes, so it is correct to put this in the profile, not
# the rc. See <http://en.wikipedia.org/wiki/Umask#Processes>.
umask u=rwx,g=,o=

# Prepend a path to a variable if the path exists.
# $1: the path variable name
# $2: the path to check and possibly prepend
path_prepend () {
  [[ -d "$2" ]] && eval "$1="'"$2:${!1}"'
}

# Add hierarchy directories to paths. Manually-installed programs
# (~/.local) should override Linuxbrew programs (~/.linuxbrew).
#
# Inspired by
# <https://technotales.wordpress.com/2010/09/19/managing-path-and-manpath/>
for prefix in ~/.linuxbrew ~/.local; do
  path_prepend PATH "$prefix/bin"
  path_prepend MANPATH "$prefix/share/man" # usual manpage install directory
  path_prepend MANPATH "$prefix/man" # older manpage install directory
  path_prepend INFOPATH "$prefix/share/info"
done

# Personal scripts directory
path_prepend PATH ~/bin

# Unset definition of path_prepend
unset -f path_prepend

# Export path variables
export PATH MANPATH INFOPATH

# Set the editor to use when a program needs to edit a file
```

```
export EDITOR='gedit --wait'

# Bash doesn't run the .bashrc for login shells -- only .bash_profile.
# However, we want to run everything in the .bashrc as well.
source ~/.bashrc

# .bashrc
# Run for interactive Bash non-login shell

# Use a custom multiline prompt
PS1='\u@\H:\w (\s-\V:\l) [\t]
$ '

# Aliases and functions
alias ll='ls -l' # long format
alias la='ls -la' # long format; show all including hidden files
alias u='cd ..' # go up a directory
alias path-print='echo "${PATH//:/'$'\n''}"'
cdl() { cd "$1" && ls; } # cd then list
mk() { mkdir -p "$1" && cd "$1"; } # make a directory then cd to it

# Key bindings
## Control-j pipes stdout and stderr of the typed command to less
bind '"\C-j": " |& less\C-m"'
## Alt-j jumps up a directory using the 'u' alias
bind '"\ej": "u\C-m"'
```

Applying the Changes

To allow changes to your shell startup files to take effect, you need to restart processes which have read it in addition to the children of such processes. See the table below for what this means:

Type of session	Modified `.bash_profile`	Modified `.bashrc`
Physical graphical session	Log out and log back in.	Close the terminal tab and re-open it. Restarting the terminal emulator is not necessary.
VNC session	Log out and log back in. Closing your VNC window or terminating your SSH tunnel is unnecessary (unless you are typing commands in the SSH tunnel session; see below).	Close the terminal tab and re-open it. Restarting the terminal emulator is not necessary.
SSH session	End the session and SSH back in.	Close the shell and start it again. If you only have one shell open (e.g., not using tmux), this will be the same as ending your SSH ssession and logging back in.
Text terminal	End the session and log back in.	Close the shell and start it again. If you only have one shell open (e.g., not using tmux), this is the same as ending your session and logging back in.

Warning: What doesn't [always] work?

1. Any of `source .bashrc`, `source .bash_profile`, `. .bashrc`, `. .bash_profile` (`source` and `.` perform the same function). This does not start a new shell, but simply re-runs the commands in the specified file.
2. `exec bash`. This starts a new shell, but does it within the current shell's environment.

Why not? Commands like `export PATH=~/bin:"$PATH"` unconditionally add a path to the existing value of `PATH`. Running these commands again in the same shell can result in duplicate paths being added. In addition, removing these commands from the startup file and re-running the startup file in the same shell doesn't remove these paths from the actual environment.

In general, commands in the startup files represent how to change the initial existing environment changed rather than representing the desired state of the environment. Because of this, they must always be executed from the same context.

3.3.9 Utilities

Various utilities can help streamline your use of the shell. Although they take some effort to install and use, the time saved by using them usually outweighs the setup cost.

Directory Navigation

One frequent task which can be expedited by utilities is changing directories. There are a few different tools which are popular for this task.

The first entry in this field is autojump. autojump is *a cd command that learns*. By simply changing directories with `cd` as usual, you can teach autojump to learn your most frequently-used directories. You can then jump to a frequently-used directory with the `j` command. autojump uses Python and is available under a variety of shells.

z is utility inspired by autojump which also takes the recency of a directory's visit into account. Unlike autojump, it is implemented in pure shell script and is available for *Bash* and *Zsh*.

fasd is a tool inspired by autojump and z which extends the concept of z beyond directory navigation. fasd allows opening frequent/recent files and directories with pre-configured programs. It is available under a wide variety of shells.

If you want to use any of these tools on EOS, the recommended method of installation is *Linuxbrew*.

Programming Language Version Management

Serious users of specific programming languages may want to have different versions of the language installed simultaneously. For this task, version managers are extremely helpful. Here is a list of version managers for various languages.

- **Python:** pyenv
- **Ruby:** rbenv, RVM
- **Java:** jenv
- **Perl:** plenv, perlbrew

Many of these tools can be installed using *Linuxbrew*.

3.3.10 Frameworks

There are a great many frameworks out there for shell enhancement and customization. One of the first to become popular was Oh My Zsh, a giant framework of functions, themes, and plugins for *Zsh*. Zsh also has a number of other frameworks, listed in unixorn's list of awesome-zsh-plugins, which includes a list of frameworks, many based on Oh My Zsh. Oh My Zsh has spawned a number of frameworks for other shells, including Bash it for *Bash* and Oh My Fish! for *fish*.

Because the so-called "dotfiles" are an important part of customizing your terminal experience, many people use version control to track and store their dotfiles. We recommend visiting dotfiles.github.io, an unofficial guide to storing your dotfiles on GitHub using Git. This site includes example dotfiles, lists of frameworks for shells and editors, and dotfile management utilities. For anything more than trivial customizations, tracking your dotfiles with a version control system is highly recommended.

3.4 Shell Scripting

3.4.1 Shebang

The shebang is a feature of Unix-like operating systems that allows text-based interpreted programs to be treated as executable files. The Shebang Wikipedia article does a great job of explaining how the shebang works. There are two common shebangs used for Bash scripts:

```
#!/bin/bash
echo 'This script runs with the system Bash interpreter.'

#!/usr/bin/env bash
echo 'This script runs with the Bash interpreter on the PATH.'
```

Each shebang is relatively self-explanatory. The second uses the `env` utility (see *Environment Variables*) to execute with the first Bash interpreter found on the `PATH`. Unless you have reason to specify the system Bash interpreter, the `#!/usr/bin/env bash` shebang is preferred because it gives the user greater flexibility over which Bash interpreter to use.

3.4.2 Boilerplate

Although shell scripts allow rapid development, they suffer from a number of weaknesses. However, some of these weaknesses can be mitigated by making shell scripts more responsive to possible errors.

By default, Bash continues to execute commands in a script even if a command fails. This is rather non-intuitive, and usually not desired because each successive command typically depends on the result of previous ones. To tell Bash to report errors by exiting when a command fails, use the `errexit` option at the top of your script:

```
set -o errexit
```

In addition, by default Bash substitutes an empty string for any undefined variable references. This is usually not desired because this typically represents a typo of some sort. To tell Bash to report any undefined variables, use the `nounset` option at the top of your script:

```
set -o nounset
```

Combining these options with the shebang above yields the following shell script boilerplate:

```
#!/usr/bin/env bash
```

```
set -o errexit
set -o nounset
```

By using these options, you will increase the chances of having robust, error-free scripts.

3.4.3 When to Shell Script

Shell scripts can become unwieldy very quickly. When writing shell scripts, you must remember their trade-offs: although they frequently allow you to accomplish tasks very quickly, they are often ugly and error-prone. In this case, it is usually better to rewrite said scripts in a more full-featured programming language such as Python or Ruby.

CHAPTER

FOUR

REMOTE ACCESS (SSH/VNC)

When not sitting at a physical machine in the EOS, Arch, or Datacomm labs, EOS can be accessed from home using a protocol called Secure Shell (SSH). SSH gives you a prompt open to *Bash*, the default shell on EOS. From this shell, you can run commands as you would inside a normal terminal emulator in an EOS desktop session. The commands will be run on the EOS machine to which you have connected.

SSH is a command-line-only technology. However, a graphical remote desktop session is available through a protocol called Virtual Network Computing (VNC). VNC allows access to a graphical desktop as if sitting at an EOS machine. Because the VNC protocol has no security of its own, our lab setup requires tunnelling VNC traffic through the SSH protocol. Each respective guide describes how to do this, but remember that you will first need to successfully set up SSH before attempting to use VNC.

An alternative to starting an entire desktop session is to run individual graphical applications from an SSH session using the X Window System. This takes advantage of X's network transparency feature and SSH's X forwarding capabilities to render on your local machine applications running on an EOS machine.

In addition to command-line and graphical access, you will likely need to transfer files between your local machine and EOS. This can be accomplished using the protocols Secure Copy (SCP) and Secure File Transfer Protocol (SFTP). Using these protocols, files can be transferred to and from EOS as well as synced between EOS and your local machine. In addition, the use of software FUSE and SSHFS allows you to treat files on EOS as if they were located on your local machine.

The hostnames for the EOS machines are organized as follows: `eos`*`XX`*`.cis.gvsu.edu` where *`XX`* is 01 through 32, `arch`*`XX`*`.cis.gvsu.edu` where *`XX`* is 01 through 10, and `dc`*`XX`*`.cis.gvsu.edu` where *`XX`* is 01 through 24. Use these names to connect to a specific EOS machine.

Your SSH client, VNC client, and X server of choice depend on your machine's operating system.

4.1 Inter-EOS SSH

EOS Lab machines are installed with OpenSSH, the most popular implementation of the SSH protocol. Because EOS machines run GNU/Linux, please read the *GNU/Linux Remote Access* section for details.

Additional information specific to EOS follows.

4.1.1 Trust All EOS Machines

For certain tasks (e.g., MPI for HPC, *Contributing to Mastering EOS*, or just plain utility) it can be useful to SSH into any EOS machine from any other EOS machine without a password. To accomplish this, follow the commands in this script. You do not need to run `ssh-keygen` if you have done so before:

```
# Generate SSH key
ssh-keygen # Press Enter at each prompt
# Add new SSH key to authorized_keys
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
_trust_host() {
  ssh -o StrictHostKeyChecking=no -o ConnectTimeout=2 "$1" exit
}
# Trust all EOS machines
for i in {1..32}; do _trust_host "eos$(printf '%02d' $i)"; done
# Trust all Arch machines
for i in {1..10}; do _trust_host "arch$(printf '%02d' $i)"; done
# Trust all Datacomm machines
for i in {1..24}; do _trust_host "dc$(printf '%02d' $i)"; done
ssh eos15 # Automatic login
```

Note: These commands temporarily disable `StrictHostKeyChecking`, which refers to the showing of *this confirmation prompt*. Since we are operating within the EOS network, this is probably OK.

4.2 Microsoft Windows

4.2.1 Shell Access (SSH)

The most popular SSH client for Windows is called PuTTY. It can be installed by visiting the PuTTY download page. We recommend installing via the Windows installer, labeled *A Windows installer for everything except PuTTYtel*.

Logging In

The first step we will take is to create a saved session for our EOS connection configuration. This will save time for future logins.

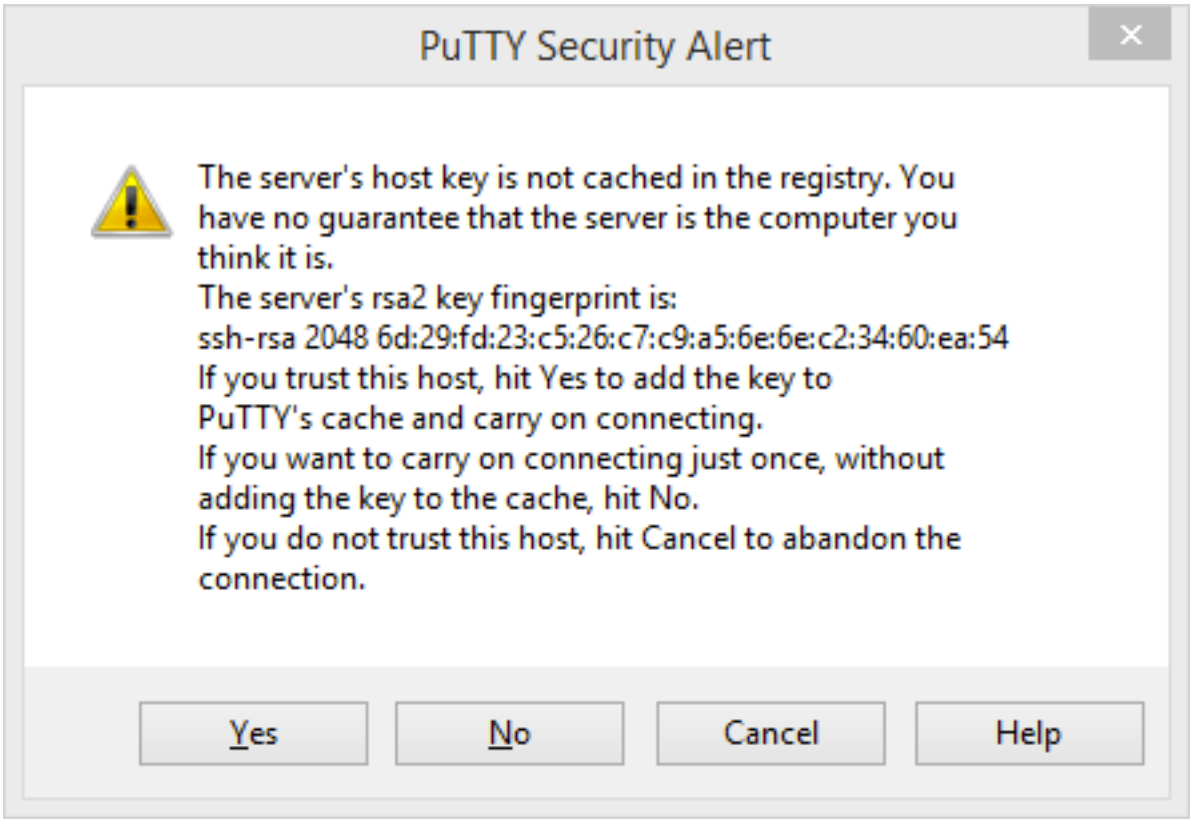
Open PuTTY and enter your username and the hostname of the EOS machine into the *Host Name* field. This has the form *user@host*, for example, `smithj@eosXX.cis.gvsu.edu`, where `XX` is the number of the chosen machine.

In the field under *Saved Sessions*, type `EOS` (this name is not strictly required, but is assumed in the next section). Click *Save*.

To log in to EOS, click *Open*. For future logins, simply select the session you created and click *Open* to connect.

Checking Host Fingerprints

When logging in to an EOS machine for the first time, you will see a dialog like this:



This is your SSH client requesting you to validate the identity of the machine to which you are connecting.

Each EOS machine has a so-called fingerprint, a series of characters which is used to verify its identity. To ensure that an attacker between your client and the actual EOS machine is not pretending to be an EOS machine, you must check that the machine's fingerprint matches the table below. Please report any mismatches to Ira Woodring immediately.

Host	Fingerprint
arch01	41:a3:8e:22:65:e7:cb:35:0a:51:9c:0f:cc:83:9c:20
arch02	41:e0:df:d1:e4:bd:b3:7f:c0:d4:15:dc:d1:dc:39:bf
arch03	64:fb:e5:5f:94:e4:d4:56:9d:8d:27:a1:a8:d8:e4:02
arch04	88:45:6e:3e:e5:20:7b:90:7d:ac:fe:f4:8f:46:3b:4e
arch05	f3:a7:a0:14:92:8d:29:0c:ab:d9:4d:d2:73:01:1e:f0
arch06	fe:75:66:3f:65:fc:63:3a:ce:c5:cd:82:ec:f9:21:fd
arch07	c6:a3:27:17:16:8a:7f:0c:8e:9f:07:77:86:b1:7b:20
arch08	b5:f9:51:c8:81:45:92:70:5a:33:06:ee:f8:7b:52:52
arch09	9b:8e:0c:48:ea:73:f6:16:b6:eb:7a:66:d4:4c:43:bc
arch10	9e:0c:a3:90:c1:12:79:72:92:c6:16:cd:06:77:78:2b
dc01	ff:7b:d3:f0:35:3c:a9:58:e7:82:07:10:ba:0d:e1:24
dc02	b5:ce:1c:6e:58:ee:8b:cc:03:b1:1f:a0:8f:7d:c7:fa
dc03	68:85:4d:d5:2f:c1:3b:88:ca:8c:f8:36:68:61:c4:54
dc04	down for maintenance
dc05	35:6a:ad:56:3a:94:94:3b:97:8b:2e:11:5d:6b:1f:bf
dc06	c8:f6:6a:b5:49:21:3e:ae:11:f7:66:e3:8e:19:47:41
dc07	0e:70:b2:f5:71:aa:f5:a8:45:47:68:6d:93:1e:b9:68
dc08	b2:37:6c:6c:56:04:a0:44:b7:60:e4:41:7e:c2:6e:58
dc09	14:5c:95:4d:f2:7c:10:8b:10:6c:f7:5d:f7:b0:57:6e
dc10	65:1c:eb:ca:91:0d:fe:b8:d6:e5:9d:26:83:c4:17:d8
dc11	ab:ee:8d:64:cf:27:ec:eb:23:c9:a2:27:e9:ef:2e:e4

Continued on next page

Table 4.1 – continued from previous page

Host	Fingerprint
dc12	0f:9a:31:da:ad:2a:6c:be:a6:d1:f9:2c:89:29:fc:05
dc13	b9:88:ec:49:ff:6a:18:bc:f7:d6:ca:d4:3b:3b:c9:17
dc14	10:58:24:90:13:a2:ad:2f:f6:2a:9e:b1:31:bb:f3:50
dc15	4d:bb:a8:2b:16:60:90:a9:4f:34:06:bb:d5:99:c2:44
dc16	93:33:fe:09:ac:97:a4:e4:ec:19:84:ce:ea:2a:be:ae
dc17	d4:3c:57:4a:ee:aa:13:da:b2:99:00:28:cf:86:7b:19
dc18	41:aa:1e:5b:d0:39:81:2a:36:19:fe:77:36:89:66:48
dc19	6a:fb:89:63:30:cb:93:67:f3:3a:31:7e:99:71:56:20
dc20	f9:39:c3:02:18:4f:70:11:37:bc:f1:0e:f7:0a:7c:64
dc21	65:77:1d:25:92:5e:0d:56:54:06:94:d0:bc:35:eb:68
dc22	1d:17:42:59:bc:92:d3:97:8c:d3:1e:82:a5:0f:77:ee
dc23	b6:c2:ee:92:51:b4:60:06:36:d6:2d:f5:c1:cf:96:82
dc24	6d:42:2a:11:3c:27:bf:50:25:fa:6b:cc:c2:0c:99:57
eos01	6d:31:03:c3:d4:76:ff:cc:89:dc:39:90:07:09:18:0d
eos02	6b:aa:3f:31:0f:4a:a8:c4:f8:48:db:30:8f:7e:87:db
eos03	58:21:fa:89:fc:5d:ca:76:99:b3:28:53:fa:5c:73:cf
eos04	a2:cf:9f:f3:6a:ed:df:5b:79:22:d8:00:db:cc:7b:bc
eos05	b5:c2:d4:40:23:d0:0c:af:32:28:36:b0:92:cc:86:da
eos06	8d:92:40:46:3b:47:53:c6:65:96:2f:9e:99:00:63:d7
eos07	86:c9:3f:a4:0c:33:42:db:66:a9:4d:88:43:6a:bc:98
eos08	48:1a:76:fc:ea:89:42:fb:01:9d:9c:94:87:2e:18:9d
eos09	47:0e:ab:d3:9d:56:09:60:93:3f:7f:e1:75:da:4a:a9
eos10	5d:89:88:4b:41:b8:3c:b3:e6:a7:97:33:18:31:a7:8c
eos11	2b:2c:b6:e1:0e:23:62:d0:32:9a:60:64:98:c8:6e:1b
eos12	c2:ed:49:40:72:49:6c:57:cf:f3:17:9b:04:18:f1:e1
eos13	62:b0:46:b3:9d:77:f4:6d:17:ad:53:b6:36:6b:72:24
eos14	cb:39:d3:5a:b2:84:a8:63:0d:cf:7c:40:ff:b6:1a:4d
eos15	10:fd:78:c3:37:79:a5:eb:ed:71:bd:d2:ee:3e:b7:ab
eos16	bb:8a:b7:f7:d5:64:d7:6a:21:d9:e6:0a:f6:9e:3c:09
eos17	e2:be:c4:1f:fc:b5:8a:ab:3d:b8:31:6a:f8:4a:4c:ae
eos18	df:d4:13:e5:74:71:77:0b:f6:5c:58:a5:b4:00:c4:c2
eos19	e2:2c:1e:20:a0:00:c9:38:c7:85:58:f2:8c:d5:71:bc
eos20	b0:63:3d:46:cc:a6:75:47:ea:0b:92:cf:26:9f:c6:54
eos21	df:ac:2b:cf:b1:1f:65:1c:c4:23:ff:b1:89:e0:08:a5
eos22	6e:1c:31:0b:37:12:56:32:e4:0d:7c:52:9f:3f:3d:ef
eos23	03:d0:ec:be:74:75:c7:b9:e6:b3:bc:b1:b2:db:10:cf
eos24	4c:b4:c5:36:ee:f2:5c:87:55:4f:a6:28:7b:80:c2:af
eos25	86:7a:af:f0:a6:ea:70:e4:69:6c:13:62:ac:59:2a:28
eos26	47:07:b9:d6:c5:a6:48:f7:7c:b5:3c:9a:48:d8:a0:c6
eos27	89:7a:06:61:87:b5:8e:df:9e:93:d2:26:a0:a4:b5:19
eos28	74:e2:00:99:ce:b1:ca:df:70:b5:6d:64:99:e4:1f:eb
eos29	07:07:e7:ff:c9:1c:31:11:2a:ad:80:69:d6:90:ee:cf
eos30	2d:d1:63:05:69:39:32:77:49:bf:d7:f4:60:93:62:6a
eos31	53:9c:a6:98:b7:1e:55:55:29:92:06:75:4b:e3:23:46
eos32	c0:2a:f1:6c:41:52:f8:49:5f:5c:c7:bb:a6:f2:85:29

Password-less Logins (SSH keys)

It is often handy to be able to SSH into a host without having to type a password, for instance as part of a script. First, we need to generate your public/private key pair. Open PuTTYgen from the PuTTY distribution to begin the generation process. Click *Generate* and do the mouse nonsense to generate your keys.

Copy the text from the field labelled *Public key for pasting into OpenSSH authorized_keys file*. Open Notepad and paste this text. Save it to the desktop as `id_rsa.pub`. Now open Windows PowerShell from the Start Menu and run the following command. If your EOS saved session is named something other than "EOS", you will need to change it in the command below.

```
$puttySessionName = 'EOS'; Get-Content "$env:USERPROFILE\Desktop\id_rsa.pub" | & "$env:SYSTEMDRIVE\P
```

Your public key has now been uploaded to EOS. The file `id_rsa.pub` may be deleted now.

However, we still need to be able to tell PuTTY to use your private key to log in to EOS. Back in PuTTYgen, click *Save private key* and save the resulting PPK file to a location of your choosing. We recommend your home directory. Answer *Yes* when you are warned about saving the private key without a passphrase.

Note: If you would like to use a passphrase for your key, see the PuTTY Guide to Pageant after completing this guide. Setting up an SSH agent is out of the scope of this guide.

Now start up PuTTY, select your saved session, then click *Load*. This loads our previously configured session for editing. In the configuration tree to the left, expand *Connection* → *SSH* and click on *Auth*. Click *Browse...* to the right of the field labelled *Private key file for authentication*. Select the PPK file you saved earlier.

Go back to *Session* and click *Save*. PuTTY is now configured to use this private key to connect to EOS. Click *Open* to log in, which you should be able to do without a password.

Important: When you make changes to your configuration, make sure to go back to *Session* and click *Save*. If you click *Open* after making changes, those changes will be applied to the current session but will not be saved for the next time you open PuTTY.

Hint: You can also save a modified configuration under another name by editing the session name in the text box under *Saved Sessions* and clicking *Save*.

As is obvious from these instructions, SSH key management is not a simple process. We recommend reading the PuTTY Guide to SSH Keys, which is the source for much of this information. If you would like to use a passphrase with your key, please see the PuTTY Guide to Pageant, PuTTY's SSH agent.

Tunnelling / Port Forwarding

The SSH protocol possesses a special feature which allows it to tunnel other protocols within itself. This is called tunnelling or port forwarding. SSH can forward local ports (allowing the local machine access to resources on the remote machine) and remote ports (allowing the remote machine access to resources on the local machine).

Local port forwarding is the more used feature, and is explained in the following sections. Remote port forwarding is similar but is outside the scope of this guide.

Fortunately, port forwarding with PuTTY is quite easy. Fire up PuTTY and select your saved session, then click the *Load*. In the configuration tree to the left, expand *Connection* → *SSH* and click on *Tunnels*.

In the *Source port* field, we will enter the port to which traffic should arrive on our local machine. In the destination field, we will enter the host and port from which the traffic should be forwarded in the form `host:port`. The host will usually match the EOS machine to which you are connecting using SSH, although this is not required. The radio buttons should be left at *Local* for the forwarding type and *Auto* for the Internet protocol.

For example, where `eosXX.cis.gvsu.edu` is the remote EOS machine, to access a web server running on port 8000 on the EOS machine from your machine on port 5555, enter the following:

Source port	5555
Destination	`eosXX.cis.gvsu.edu:8000`

Click *Add* to add this as a forwarded port, then click *Open* (we will not save this configuration).

You can test the forwarding by running this in the SSH prompt:

```
python -m SimpleHTTPServer
```

and opening http://localhost:5555/ in your local web browser. You should see a web listing of your home directory! Press `Control-C` to kill the web server.

The remote host which is hosting the resource need not be the EOS machine to which you are connecting with SSH. Let's access the CIS web server through the SSH tunnel.

Restart PuTTY, load your session, and navigate back to the *Tunnels* screen. Enter the following information:

Source port	5678
Destination	`cis.gvsu.edu:80`

Click *Add* and *Open*, then visit http://localhost:5678/ in your local web browser. The CIS home page should appear!

For a guide to using PuTTY and SSH as a proxy for *all* your web traffic, check out this TechRepublic article.

4.2.2 Graphical Access (VNC/X11)

VNC

First, we need to create a tunnel in order to forward VNC through our SSH connection. The remote port to which we must connect depends on the desired resolution of the remote desktop. Select a desired resolution from the following table, and note the port to which it corresponds.

Display	Port	Geometry
0	5900	1280x1024
1	5901	1024x768
2	5902	800x600
3	5903	640x480
4	5904	1440x900
5	5905	1280x800
6	5906	1152x864
7	5907	1680x1050
8	5908	1920x1200
9	5909	1400x1050
10	5910	1440x1000
11	5911	1024x600
12	5912	1600x900
13	5913	1920x1080
14	5914	1360x768

In the following instructions, replace `REMOTE_PORT` with the port that you have selected.

Restart PuTTY, load your session, and navigate back to the *Tunnels* screen. Enter the following information:

Source port	5900
Destination	`eos`*`XX`*`.cis.gvsu.edu:`*`REMOTE_PORT`*

Go back to *Session* and click *Save*. You are now ready to tunnel your VNC session. Click *Open* to start the tunnel.

Hint: If you clone a session for an EOS machine (using *Load* and *Save*), don't forget to change the tunnel to forward ports to that machine.

The recommended VNC client for Windows is TightVNC. Download it, install, then open. In the field labelled *Remote Host*, type `localhost`. Click *Connect* to start the connection.

For future connections, simply start TightVNC and click *Connect*. Alternatively, during the session, you can save the configuration to a file by clicking the *Save* button, shown as a diskette. After saving the configuration to a `*.vnc` file, double click the file to start the connection.

X Forwarding

There are a few X servers available for Windows, but the most popular is Xming, a native Windows X server based on X.Org. Download and install the public domain release from the Xming release page. During installation, feel free to choose not to install an SSH client if you already have PuTTY installed (which you should).

First, start up the Xming server by simply opening it. Next, open PuTTY and load your EOS session.

In the configuration tree to the left, expand *Connection* → *SSH* and click on *X11*. Tick the checkbox labeled *Enable X11 forwarding*. You may open the session immediately or go back to the *Session* screen to save a session with X forwarding automatically enabled.

From your remote shell, try running a graphical program:

```
gedit
```

You may want to send the program to the background to regain use of the shell:

```
gedit &
```

X11 is a heavyweight protocol, and X11 sessions function best on high bandwidth, low latency connections. Remote applications running through X forwarding will typically be much less responsive than if they were running on an EOS machine. If you experience performance problems (and you probably will, depending on the applications that you use), consider using VNC. X forwarding is good for one-off applications, like viewing images or PDFs, but typically not good for editing text, web browsing, or running full desktop sessions. Always keep this in mind when using this technology.

Xming will stay open even after your PuTTY session has ended. Feel free to quit Xming from the system tray when you are finished using it.

Another alternative X server for Windows is Cygwin/X, Cygwin's X server.

4.2.3 File Transfer

Graphical

The recommend graphical client for file transfer on Windows is WinSCP, which can be found on the WinSCP downloads page. We recommend downloading the latest stable installer, labeled *Installation package*. It should be near the top.

After downloading, run the installer. The *Typical installation* is usually fine, but feel free to customize the installation options. You can choose either the *Commander* or *Explorer* interface, but keep in mind that most people use the *Commander* interface. Don't forget to disable the sneaky Google Chrome installer included with this installer.

After the installer copies its files, it may detect your sessions from PuTTY. If so, click *OK* to import them. Select the sessions you'd like to import and click *OK* again. This is the single easiest way to start quickly with WinSCP.

At the end of the installer, leave the box labeled *Launch WinSCP* checked. You can choose to open the *Getting started* page as well, although there is really no need to do so.

If you didn't choose to import your sites from PuTTY in the installer, you can also import them from the WinSCP Login screen by clicking *Tools* → *Import Sites...*, selecting the sites, and clicking *OK*.

There is really no reason not to import your sites from PuTTY if you already have them configured (you should). However, if you'd like to create a custom site, click *New Site*. Choose SFTP as the protocol, and enter in the EOS machine for *Host name* as well as your username. For authentication, you can use a password or SSH keys. To select a key, click *Advanced...*, then *SSH* → *Authentication* → *Authentication parameters* → *Private key file* to select the private key file. Click *Save* to save your site.

Tip: You can create a desktop shortcut for your site by right-clicking your site in the WinSCP Login screen, then clicking *Desktop Icon*. This allows you to open your site directly without visiting the WinSCP Login screen. Creating a 'Send To' shortcut for Windows Explorer is similarly useful.

Automatic Synchronization

Automatic synchronization of local to remote directories is a very useful advanced feature of WinSCP. It is especially useful when developing a website on EOS. This partially makes up for the lack of a maintained free Windows SSHFS or rsync implementation.

To start using it, click *Commands* → *Keep Remote Directory up to Date...*. You can get more information about this task and its use in the WinSCP Keep Remote Directory up to Date documentation.

For more information on using WinSCP, consult the excellent WinSCP Documentation.

Command Line (SCP)

Files can be transferred on the command line using a utility called SCP, implemented in PuTTY through a command called `pscp`. Because `pscp` uses PuTTY for authentication, if you have set up *Password-less Logins (SSH keys)*, you will not have to type any passwords. SCP stands for *Secure Copy* and works very similar to the GNU/Linux `cp` command, except that it can also transfer files across the network. Make sure you are familiar with the operation of `cp` before using SCP.

PuTTY's commands are not added to the Windows `Path` by default. To add them to the `Path`, open Windows PowerShell from the Start Menu and run the following command. If you installed PuTTY to a non-default location, you will need to change it in the command below.

```
$puttyInstallPath = 'C:\Program Files (x86)\PuTTY'; [Environment]::SetEnvironmentVariable('Path', [En
```

Restart PowerShell or your terminal emulator after running this command to allow your updates to the `Path` to take effect. The `pscp` utility may now be used from PowerShell by simply typing `pscp`.

The following examples showcase the typical use of `pscp`. Each file can be prefixed with a PuTTY session name or user/host, which tells `pscp` where it is or should be located. The session name "EOS" is used in these examples; change it to match your PuTTY session name if it is different. Files with no prefix are assumed to be on the local machine. Paths on the remote machine start at your home directory, so there is typically no need to include `/home/smithj` in the path. Here are some examples of use of `pscp`:

```
# Typical upload
pscp classes\cis162\hw1.txt EOS:classes/cis162
# Typical download
pscp EOS:classes/cis162/hw2.txt classes\cis162
# Upload a directory
pscp -r projects EOS:classes/cis163
# User/host instead of EOS session name (will likely require password)
pscp smithj@eos01.cis.gvsu.edu:classes/cis162/hw3.txt classes/cis162
```

Note: Windows uses \ as a path separator by default, while GNU/Linux uses /. While Windows is generally forgiving and will accept / as well, GNU/Linux is not. *Always use / as a path separator when specifying GNU/Linux paths.*

SSHFS

A further alternative to transferring files is to use the Secure Shell Filesystem, SSHFS. SSHFS is based on Filesystem in Userspace, FUSE. By using SSHFS, you can mount your EOS home directory on your local machine as a separate drive. This allows you to in effect run programs within your local machine on files within your EOS account. It is very similar to editing files on a flash drive. SSHFS can be used for any purpose, but is especially useful for web development on EOS.

Unfortunately, there are no stand-out options for SSHFS on Windows. The following programs are possible solutions of which we are aware:

- ExpanDrive is a commercial product with a free trial available. While the product works well, the prices are in the expensive range.
- win-sshfs is an open-source SSHFS implementation for Windows. Unfortunately, it is not being maintained and therefore we cannot recommend it.
- dimov-cz's win-sshfs fork is a maintained fork of win-sshfs. However, no binaries are provided, so it must be compiled from source. If you are familiar with compiling and installing .NET programs, this may be a viable alternative for you.

Although none of these programs are supported, you are welcome to try them if they seem useful to you.

For another guide to SSHFS, check out DigitalOcean's guide to SSHFS.

4.2.4 Alternative Clients

Though PuTTY is the recommended SSH client for Windows, OpenSSH is also available. The recommended way of running OpenSSH on Windows is through Cygwin. Cygwin is not simple to use and configure, but depending on your needs, it may provide a better SSH experience. OpenSSH is well-known as the best SSH client out there, and EOS uses OpenSSH as an SSH server as well.

There are a plethora of alternate VNC viewers available for Windows, many based on the same original RealVNC code.

UltraVNC and TigerVNC offer relatively simple user interfaces with an appropriate amount of configuration options. If you don't like or are having trouble with TightVNC, give UltraVNC a try.

RealVNC Viewer and Viewer Plus are freeware viewers, but require registration. RealVNC also offers RealVNC Viewer for Google Chrome, a free Google Chrome extension which does not require registration.

Cyberduck is also available for Windows. Cyberduck has a more attractive and intuitive interface than WinSCP. However, unlike WinSCP, Cyberduck does not support automatic synchronization. This is important because high-quality free versions of SSHFS and rsync are not available for Windows.

MobaXterm is an all-in-one solution for SSH, SCP, VNC, RDP, and more. Since it is a unified product, it provides a smoother experience than a collection of standalone applications. However, because it includes much more capability, it can be difficult to configure. It is worth a try if your time spent on EOS warrants it.

Danger: Another popular client for SFTP is FileZilla. However, because FileZilla stores its passwords insecurely, we cannot recommend it. Please avoid its use to keep your password secure.

4.3 Mac OS X

4.3.1 Shell Access (SSH)

Mac OS X comes preinstalled with OpenSSH, the most popular implementation of the SSH protocol. The client can be run from the command-line and is simply called `ssh`.

Logging in

First, open your terminal emulator to start a shell. To connect to a specific machine, pass your username and hostname to the command-line client in the form *user@host*. For example, where *XX* is the number of the chosen machine, run:

```
ssh smithj@eosXX.cis.gvsu.edu
```

Checking Host Fingerprints

When logging in to an EOS machine for the first time, you will see a message like this:

```
The authenticity of host 'eos01.cis.gvsu.edu (148.61.162.101)' can't be established.
RSA key fingerprint is 6d:29:fd:23:c5:26:c7:c9:a5:6e:6e:c2:34:60:ea:54.
Are you sure you want to continue connecting (yes/no)?
```

This is your SSH client requesting you to validate the identity of the machine to which you are connecting.

Each EOS machine has a so-called fingerprint, a series of characters which is used to verify its identity. To ensure that an attacker between your client and the actual EOS machine is not pretending to be an EOS machine, you must check that the machine's fingerprint matches the table below. Please report any mismatches to Ira Woodring immediately.

Host	Fingerprint
arch01	`41:a3:8e:22:65:e7:cb:35:0a:51:9c:0f:cc:83:9c:20`
arch02	`41:e0:df:d1:e4:bd:b3:7f:c0:d4:15:dc:d1:dc:39:bf`
arch03	`64:fb:e5:5f:94:e4:d4:56:9d:8d:27:a1:a8:d8:e4:02`
arch04	`88:45:6e:3e:e5:20:7b:90:7d:ac:fe:f4:8f:46:3b:4e`
arch05	`f3:a7:a0:14:92:8d:29:0c:ab:d9:4d:d2:73:01:1e:f0`
arch06	`fe:75:66:3f:65:fc:63:3a:ce:c5:cd:82:ec:f9:21:fd`
arch07	`c6:a3:27:17:16:8a:7f:0c:8e:9f:07:77:86:b1:7b:20`
arch08	`b5:f9:51:c8:81:45:92:70:5a:33:06:ee:f8:7b:52:52`
arch09	`9b:8e:0c:48:ea:73:f6:16:b6:eb:7a:66:d4:4c:43:bc`
arch10	`9e:0c:a3:90:c1:12:79:72:92:c6:16:cd:06:77:78:2b`
dc01	`ff:7b:d3:f0:35:3c:a9:58:e7:82:07:10:ba:0d:e1:24`
dc02	`b5:ce:1c:6e:58:ee:8b:cc:03:b1:1f:a0:8f:7d:c7:fa`
dc03	`68:85:4d:d5:2f:c1:3b:88:ca:8c:f8:36:68:61:c4:54`
dc04	down for maintenance
dc05	`35:6a:ad:56:3a:94:94:3b:97:8b:2e:11:5d:6b:1f:bf`
dc06	`c8:f6:6a:b5:49:21:3e:ae:11:f7:66:e3:8e:19:47:41`
dc07	`0e:70:b2:f5:71:aa:f5:a8:45:47:68:6d:93:1e:b9:68`
dc08	`b2:37:6c:6c:56:04:a0:44:b7:60:e4:41:7e:c2:6e:58`
dc09	`14:5c:95:4d:f2:7c:10:8b:10:6c:f7:5d:f7:b0:57:6e`
dc10	`65:1c:eb:ca:91:0d:fe:b8:d6:e5:9d:26:83:c4:17:d8`
dc11	`ab:ee:8d:64:cf:27:ec:eb:23:c9:a2:27:e9:ef:2e:e4`
dc12	`0f:9a:31:da:ad:2a:6c:be:a6:d1:f9:2c:89:29:fc:05`
dc13	`b9:88:ec:49:ff:6a:18:bc:f7:d6:ca:d4:3b:3b:c9:17`

Continued on next page

Table 4.2 – continued from previous page

Host	Fingerprint
dc14	10:58:24:90:13:a2:ad:2f:f6:2a:9e:b1:31:bb:f3:50
dc15	4d:bb:a8:2b:16:60:90:a9:4f:34:06:bb:d5:99:c2:44
dc16	93:33:fe:09:ac:97:a4:e4:ec:19:84:ce:ea:2a:be:ae
dc17	d4:3c:57:4a:ee:aa:13:da:b2:99:00:28:cf:86:7b:19
dc18	41:aa:1e:5b:d0:39:81:2a:36:19:fe:77:36:89:66:48
dc19	6a:fb:89:63:30:cb:93:67:f3:3a:31:7e:99:71:56:20
dc20	f9:39:c3:02:18:4f:70:11:37:bc:f1:0e:f7:0a:7c:64
dc21	65:77:1d:25:92:5e:0d:56:54:06:94:d0:bc:35:eb:68
dc22	1d:17:42:59:bc:92:d3:97:8c:d3:1e:82:a5:0f:77:ee
dc23	b6:c2:ee:92:51:b4:60:06:36:d6:2d:f5:c1:cf:96:82
dc24	6d:42:2a:11:3c:27:bf:50:25:fa:6b:cc:c2:0c:99:57
eos01	6d:31:03:c3:d4:76:ff:cc:89:dc:39:90:07:09:18:0d
eos02	6b:aa:3f:31:0f:4a:a8:c4:f8:48:db:30:8f:7e:87:db
eos03	58:21:fa:89:fc:5d:ca:76:99:b3:28:53:fa:5c:73:cf
eos04	a2:cf:9f:f3:6a:ed:df:5b:79:22:d8:00:db:cc:7b:bc
eos05	b5:c2:d4:40:23:d0:0c:af:32:28:36:b0:92:cc:86:da
eos06	8d:92:40:46:3b:47:53:c6:65:96:2f:9e:99:00:63:d7
eos07	86:c9:3f:a4:0c:33:42:db:66:a9:4d:88:43:6a:bc:98
eos08	48:1a:76:fc:ea:89:42:fb:01:9d:9c:94:87:2e:18:9d
eos09	47:0e:ab:d3:9d:56:09:60:93:3f:7f:e1:75:da:4a:a9
eos10	5d:89:88:4b:41:b8:3c:b3:e6:a7:97:33:18:31:a7:8c
eos11	2b:2c:b6:e1:0e:23:62:d0:32:9a:60:64:98:c8:6e:1b
eos12	c2:ed:49:40:72:49:6c:57:cf:f3:17:9b:04:18:f1:e1
eos13	62:b0:46:b3:9d:77:f4:6d:17:ad:53:b6:36:6b:72:24
eos14	cb:39:d3:5a:b2:84:a8:63:0d:cf:7c:40:ff:b6:1a:4d
eos15	10:fd:78:c3:37:79:a5:eb:ed:71:bd:d2:ee:3e:b7:ab
eos16	bb:8a:b7:f7:d5:64:d7:6a:21:d9:e6:0a:f6:9e:3c:09
eos17	e2:be:c4:1f:fc:b5:8a:ab:3d:b8:31:6a:f8:4a:4c:ae
eos18	df:d4:13:e5:74:71:77:0b:f6:5c:58:a5:b4:00:c4:c2
eos19	e2:2c:1e:20:a0:00:c9:38:c7:85:58:f2:8c:d5:71:bc
eos20	b0:63:3d:46:cc:a6:75:47:ea:0b:92:cf:26:9f:c6:54
eos21	df:ac:2b:cf:b1:1f:65:1c:c4:23:ff:b1:89:e0:08:a5
eos22	6e:1c:31:0b:37:12:56:32:e4:0d:7c:52:9f:3f:3d:ef
eos23	03:d0:ec:be:74:75:c7:b9:e6:b3:bc:b1:b2:db:10:cf
eos24	4c:b4:c5:36:ee:f2:5c:87:55:4f:a6:28:7b:80:c2:af
eos25	86:7a:af:f0:a6:ea:70:e4:69:6c:13:62:ac:59:2a:28
eos26	47:07:b9:d6:c5:a6:48:f7:7c:b5:3c:9a:48:d8:a0:c6
eos27	89:7a:06:61:87:b5:8e:df:9e:93:d2:26:a0:a4:b5:19
eos28	74:e2:00:99:ce:b1:ca:df:70:b5:6d:64:99:e4:1f:eb
eos29	07:07:e7:ff:c9:1c:31:11:2a:ad:80:69:d6:90:ee:cf
eos30	2d:d1:63:05:69:39:32:77:49:bf:d7:f4:60:93:62:6a
eos31	53:9c:a6:98:b7:1e:55:55:29:92:06:75:4b:e3:23:46
eos32	c0:2a:f1:6c:41:52:f8:49:5f:5c:c7:bb:a6:f2:85:29

Password-less Logins (SSH keys)

It is often handy to be able to SSH into a host without having to type a password, for instance as part of a script. First, generate your public/private key pair with:

```
ssh-keygen
```

Accept the default values by pressing `Enter` at each prompt unless you know what you are doing. Once the keys

have been generated, you can copy the public key over to the remote system by entering:

```
ssh smithj@eos01.cis.gvsu.edu 'umask u=rwx,go= && mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys' <
```

When you SSH into EOS now, you should be able to do so without having to provide a password:

```
ssh smithj@eos01.cis.gvsu.edu
```

Note: In this setup, we created our public/private key pair without a passphrase, which is less secure. If you would like to use a passphrase, please see Mark Hershberger's excellent guide to ssh-agent and Github's guide to SSH passphrases.

Tunnelling / Port Forwarding

The SSH protocol possesses a special feature which allows it to tunnel other protocols within itself. This is called tunnelling or port forwarding. SSH can forward local ports (allowing the local machine access to resources on the remote machine) and remote ports (allowing the remote machine access to resources on the local machine).

Local port forwarding is the more used feature, and is explained in the following sections. Remote port forwarding is similar but is outside the scope of this guide.

Port forwarding can be accomplished with OpenSSH by passing arguments to the command-line client or by editing the client configuration file.

Forwarding on the Command Line

Forwarding local ports on the command-line can be accomplished using the following syntax:

```
ssh -L local_port:remote_host:remote_port user@host
```

For example, to access a web server running on port 8000 on `eos01.cis.gvsu.edu` from your machine on port 5555, use the following command line:

```
ssh -L 5555:eos01.cis.gvsu.edu:8000 smithj@eos01.cis.gvsu.edu
```

You can test the forwarding by running this in the SSH prompt:

```
python -m SimpleHTTPServer
```

and opening http://localhost:5555/ in your local web browser. You should see a web listing of your home directory! Press `Control-C` to kill the web server.

The remote host which is hosting the resource need not be the EOS machine to which you are connecting with SSH. For example, to access the CIS web server through your SSH tunnel, you can run:

```
ssh -L 5678:cis.gvsu.edu:80 smithj@eos01.cis.gvsu.edu
```

and visit http://localhost:5678/ in your local web browser. The CIS home page should appear!

Forwarding in the Config File

The command-line works well for one-off tunnels, but for frequently established tunnels, it pays to alter the OpenSSH client configuration file. The OpenSSH client configuration resides on your local machine in the file `~/.ssh/config`. This is a file inside a hidden directory inside your home directory. To setup this directory and file, please run the following commands:

```
umask u=rwx,go=
mkdir -p ~/.ssh
touch ~/.ssh/config
```

The umask command ensures that the directory and file are created with the correct permissions. This is important when dealing with SSH-related files.

Now open the file `~/.ssh/config` in a text editor.

To establish the CIS web server forwarding shown in the last section, one could use the following configuration:

```
Host eoscisweb
HostName eos01.cis.gvsu.edu
User smithj
LocalForward 5678 cis.gvsu.edu:80
```

To use this host from the command line, simply type:

```
ssh eoscisweb
```

4.3.2 Graphical Access (VNC/X11)

VNC

First, we need to create a tunnel in order to forward VNC through our SSH connection. The remote port to which we must connect depends on the desired resolution of the remote desktop. Select a desired resolution from the following table, and note the port to which it corresponds.

Display	Port	Geometry
0	5900	1280x1024
1	5901	1024x768
2	5902	800x600
3	5903	640x480
4	5904	1440x900
5	5905	1280x800
6	5906	1152x864
7	5907	1680x1050
8	5908	1920x1200
9	5909	1400x1050
10	5910	1440x1000
11	5911	1024x600
12	5912	1600x900
13	5913	1920x1080
14	5914	1360x768

In the following instructions, replace `REMOTE_PORT` with the port that you have selected.

To create the tunnel, use the following command line:

```
ssh -L 5900:eosXX.cis.gvsu.edu:REMOTE_PORT smithj@eosXX.cis.gvsu.edu
```

Or the following configuration file:

```
Host eosvnc
HostName eosXX.cis.gvsu.edu
User smithj
LocalForward 5900 eosXX.cis.gvsu.edu:REMOTE_PORT
```

If you used the configuration file, run the following to create the tunnel:

```
ssh eosvnc
```

You are now ready to tunnel your VNC session.

The recommended VNC client for OS X is Chicken, which is free and open-source software. Visit the website to download, install as you would any other Mac OS X application, and open.

You will be prompted to create a new server. If not prompted, click *Connection → Open Connection...* from the menu bar. Double click "New Server" in the list on the left and rename it to "EOS", or create a new session with the + button if none exist. Leave the *Host* field as `localhost` or fill it in if missing. Leave the *Display or port* field at 0 or fill it in if missing.

Chicken has had some problems with the ZRLE encoding with our server. As this can cause a premature end to your session, our recommendation is to manually disable this encoding. To do this, first click the drop-down menu next to *Profile*, and click *Edit Connection Profiles....* The *Profile Manager* configuration window will open. In the bottom left, enter "EOS" into the field and click the + button. Now click the checkbox next to the ZRLE encoding to disable it for EOS sessions. Close the *Profile Manager* window.

Click *Connect* to begin your VNC session with EOS. To connect in the future, select *Connection → Open Connection...* from the menu, select your EOS configuration, and click *Connect*.

Note: Although Chicken offers an option to tunnel directly through SSH, we have not had luck using this option with our setup. We recommend sticking with the traditional SSH tunnel, as it is tested and works well.

X Forwarding

Mac OS X has a high-quality X server called XQuartz. It is based on X.Org, and is partially maintained by Apple. Visit the XQuartz website to download and install the latest release.

The `ssh` program needs to be told to initiate X forwarding when the connection is started. This can be done with the `-X` command-line flag:

```
ssh -X smithj@eosXX.cis.gvsu.edu
```

This can also be accomplished in the SSH configuration file:

```
Host eosx
HostName eos01.cis.gvsu.edu
User smithj
ForwardX11 yes
```

Upon establishing the connection, you should see the XQuartz app launch (it has the X.Org logo). All remote applications will be rendered within the XQuartz app.

From your remote shell, try running a graphical program:

```
gedit
```

You may want to send the program to the background to regain use of the shell:

```
gedit &
```

X11 is a heavyweight protocol, and X11 sessions function best on high bandwidth, low latency connections. Remote applications running through X forwarding will typically be much less responsive than if they were running on an EOS machine. If you experience performance problems (and you probably will, depending on the applications that you use), consider using VNC. X forwarding is good for one-off applications, like viewing images or PDFs, but

typically not good for editing text, web browsing, or running full desktop sessions. Always keep this in mind when using this technology.

XQuartz will stay open even after your SSH session has ended. Feel free to quit XQuartz when you are finished using it.

4.3.3 File Transfer

Graphical

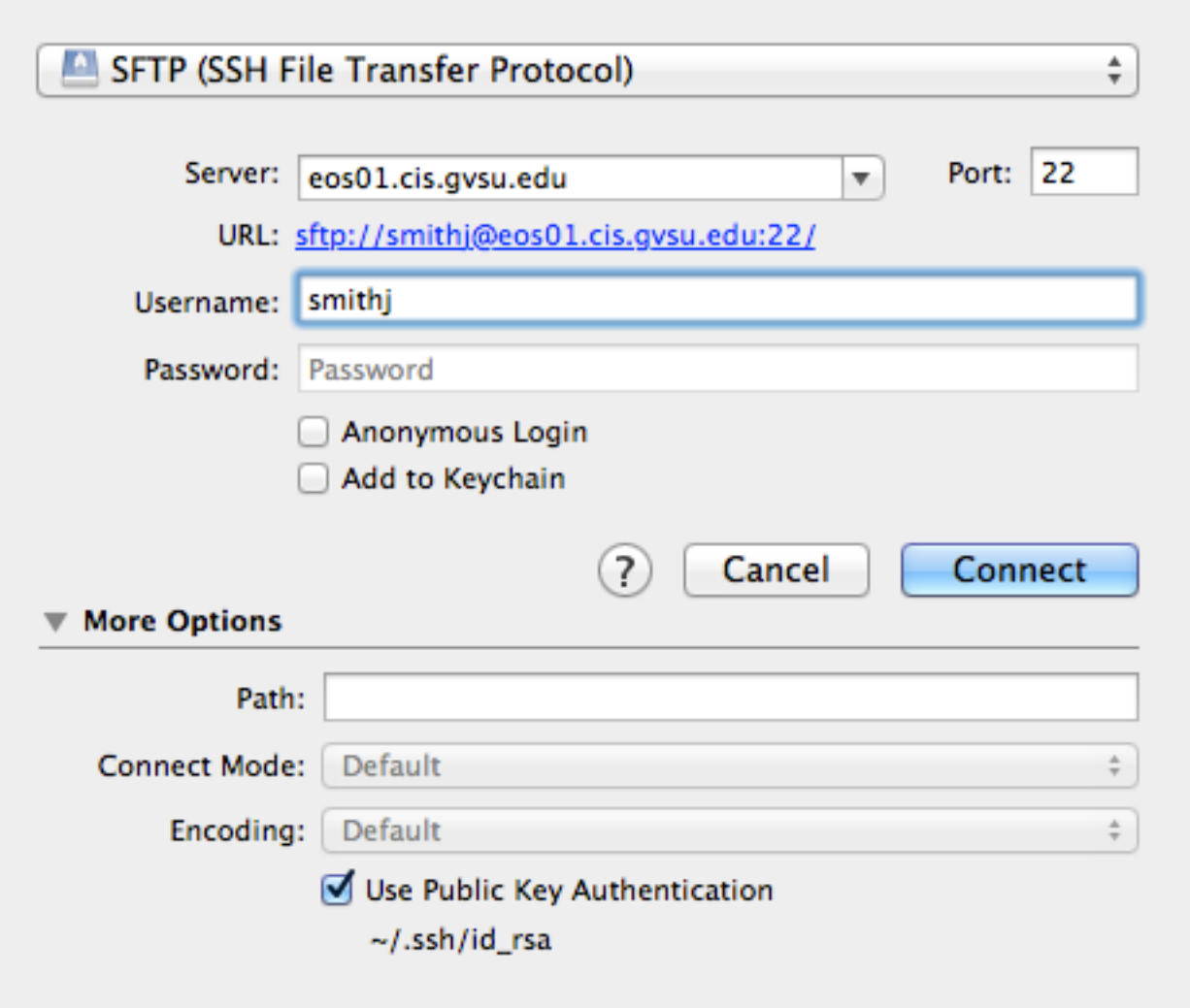
The recommend graphical client for file transfer on Mac OS X is Cyberduck. It is quite easy to use. Download the application, install as normal, and start Cyberduck.

Click *Open Connection*. From the drop-down box, select *SFTP (SSH File Transfer Protocol)*. Type in the EOS machine to which you'd like to connect in the *Server* field and fill in your username.

If you have set up *Password-less Logins (SSH keys)*, click the *Use Public Key Authentication* checkbox and select the file `id_rsa`. This is the identity file that you use to log in to EOS. If you have not set up your keys, you can still use password authentication, although this is not recommended.

If you have created a config file with hostname aliases, you may also notice Cyberduck auto-detect your configuration and fill in some information.

Your connection information should look something like this:



After connecting, we recommend creating a bookmark so that you can easily return. Click *Bookmark* → *New Bookmark* to create one. You can change the nickname if you like. When you start Cyberduck again, simply click your bookmark to connect.

Most of Cyberduck's action are available through the *File* menu or the right-click context menu. In addition, Cyberduck has great support for dragging between it and Finder.

Command Line (SCP)

Files can be transferred on the command line using a utility called SCP. Because SCP uses SSH for authentication, if you have set up *Password-less Logins (SSH keys)*, you will not have to type any passwords. SCP stands for *Secure Copy* and works very similar to the `cp` command, except that it can also transfer files across the network. Make sure you are familiar with the operation of `cp` before using SCP.

Each file can be prefixed with a machine name, which tells SCP where it is or should be located. Files with no prefix are assumed to be on the local machine. Paths on the remote machine start at your home directory, so there is typically no need to include `/home/smithj` in the path. Here are some examples of use of SCP:

```
# Typical upload
scp classes/cis162/hw1.txt eos01.cis.gvsu.edu:classes/cis162
# Typical download
```

```
scp eos01.cis.gvsu.edu:classes/cis162/hw2.txt classes/cis162
# Upload a directory
scp -r projects eos01.cis.gvsu.edu:classes/cis163
# Include username as well
scp smithj@eos01.cis.gvsu.edu:classes/cis162/hw3.txt classes/cis162
# Hostname aliases make this easier
scp eos01:classes/cis162/hw4.txt classes/cis162
```

rsync

rsync is the go-to tool for remote synchronization. rsync is the best choice for file transfer when the desire is to mirror a large file or directory of data, transferring it between your local machine and an EOS system. The rsync program is almost always installed by default.

Operation of rsync is very similar to SCP. The main difference between rsync and SCP is that rsync features a remote-update protocol which transfers only the differences between the local and remote files, decreasing sync time and bandwidth.

A typical use of rsync is as follows:

```
rsync \
    --verbose \
    --archive \
    --compress \
    classes/cis163/projects \
    eos01.cis.gvsu.edu:classes/cis163/projects
```

This syncs a local projects directory with a remote one. The `--verbose` option instructs rsync to tell you what it is doing, the `--archive` option tells rsync to duplicate the files almost exactly, and the `--compress` option tells rsync to use compression within the protocol.

Here is another example for deploying a website. Specifically, this deploys a Mastering EOS build to your user's personal website:

```
rsync \
    --verbose \
    --archive \
    --compress \
    --delete \
    --chmod=go=rX \
    build/website \
    eos01.cis.gvsu.edu:public_html/mastering-eos
```

The `--delete` option tells rsync to delete files on the remote machine not present on the local machine. This allows true synchronization of the two directories. However, be careful with this option, as it will delete files without confirmation!

The `--chmod=go=rX` option tells rsync to set the permissions of the files to represent the correct permissions for use on a website.

Because rsync command lines can get quite long, it is often useful to record them in a script. See *rsync(1)* by running `man rsync` for more information on the use of rsync.

There is also an rsync GUI available called Grsync, though we cannot attest to its effectiveness.

SSHFS

A further alternative to transferring files is to use the Secure Shell Filesystem, SSHFS. SSHFS is based on Filesystem in Userspace, FUSE. By using SSHFS, you can mount your EOS home directory on your local machine as a separate drive. This allows you to in effect run programs within your local machine on files within your EOS account. It is very similar to editing files on a flash drive. SSHFS can be used for any purpose, but is especially useful for web development on EOS.

Installation

The OSXFUSE project maintains high-quality packages of FUSE and SSHFS. Visit the homepage and download and install the stable versions of *both* OSXFUSE and SSHFS to get started.

Use

To mount your EOS home directory, first create a mount point for it:

```
mkdir ~/eos
```

Next, mount your EOS home directory using SSHFS:

```
sshfs -o volname=EOS smithj@eos01.cis.gvsu.edu: ~/eos
```

Tip: If you set up *Hostname Aliases*, you can use these with SSHFS:

```
sshfs -o volname=EOS eos01: ~/eos
```

Test the mount point by listing your EOS files:

```
ls ~/eos
```

You should now be able to use files on your EOS account as if they were on your own machine. For example, you can open and browse your files using Finder:

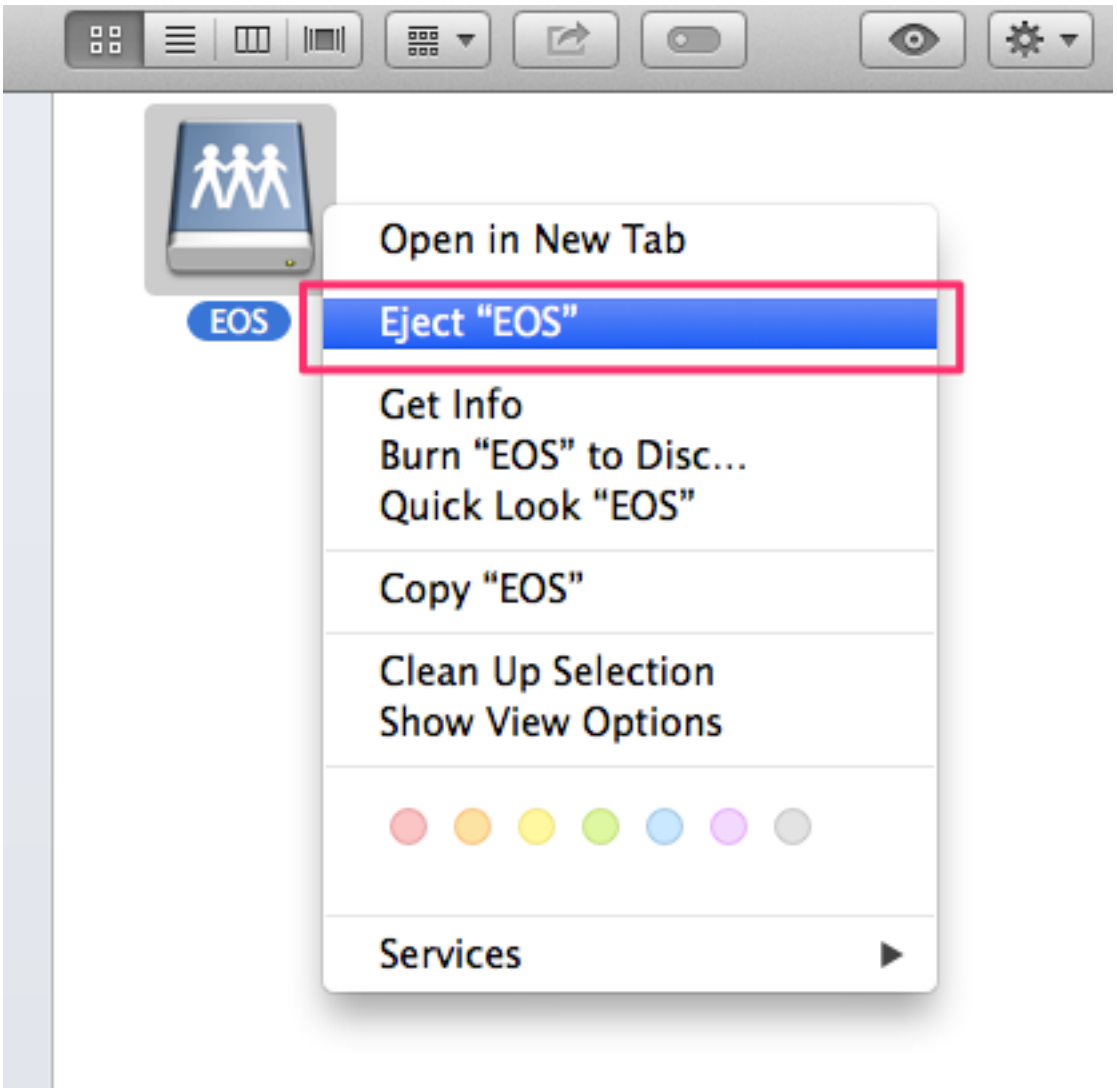
```
open ~/eos
```

Be aware that there may be some lag in the filesystem, especially when using Finder or other programs which access the filesystem frequently.

When finished with the mount point, make sure to unmount it:

```
diskutil unmount ~/eos
```

The mount point can also be unmounted using Finder:



SSHFS accepts many options which can be viewed with `man ssh` or `sshfs --help`. For example, to enable caching and automatic reconnection (recommended):

```
sshfs -o auto_cache,reconnect,volname=EOS smithj@eos01.cis.gvsu.edu: ~/eos
```

If you use this command often, you may want to create script or alias for it.

For another guide to SSHFS, check out DigitalOcean's guide to SSHFS.

Another option for SSH file transfer on Mac OS X is ExpanDrive, a commercial product.

4.3.4 Advanced OpenSSH

OpenSSH can do much more than simply allow the user to establish connections with remote servers. If you use OpenSSH, there are a great many neat tricks available to you.

This section is based in large part on the Smylers SSH Productivity Tips blog post. Please visit this post for *even more SSH awesomeness!*

Hostname Aliases

It's useful not to have to type out the entire full-qualified domain names to EOS machines. What you might normally type would be something like this:

```
ssh smithj@eos02.cis.gvsu.edu
# or
ssh smithj@arch04.cis.gvsu.edu
```

By adding a section to the config file, this becomes easier. Add this to your `~/.ssh/config` as mentioned earlier:

```
# EOS
# Match all eos01, eos11, arch08, etc.
Host eos?? arch??
HostName %h.cis.gvsu.edu
User smithj
```

With this, now you need only type:

```
ssh eos02
# or
ssh arch04
```

Shared Connections

There is a about a minute timeout between allowed successive connnections to each individual EOS machine. This can prove very annoying when establishing multiple SSH connections to use **scp** to copy files or opening multiple terminals (but see terminal multiplexing). One way to mitigate this annoyance is by using GVSU's VPN. Another way is to used SSH shared connections. These solutions are also not mutually exclusive.

Shared connections are established by creating a socket which multiplexes multiple connections. This socket is controlled by the `ControlMaster` and `ControlPath` keywords. The first connection is called the "master" and creates the socket. Subsequent connections use the already-created socket. This behavior can be automated by setting `ControlMaster` to the value `auto`. `ControlPath` specifies the path to the socket, with variables substituted as necessary. The following config amend the previous config to add connection sharing to EOS machines.

```
# EOS
# Match all eos01, eos11, arch08, etc.
Host eos?? arch??
HostName %h.cis.gvsu.edu
User smithj
ControlMaster auto
#                         Host
#                           |Port
#                           |   | Username
#                           V   V   V
ControlPath /tmp/ssh_mux_%h_%p_%r
```

Connection sharing may be useful to enable for most hosts. However, it needs to be done with care because it typically conflicts with X forwarding and port forwarding.

Persistent Connections

It is often useful to keep connections open in the background even after the terminal has actually been closed. This is useful as it allows OpenSSH to reconnect to the server without re-establishing a connection. Turning this behavior on is trivially simple. Add the following line under the host for which you would like connections to persist:

```
# Persist connections for 2 hours.
ControlPersist 2h
```

For GitHub users, this is especially useful when using Git over SSH. Within this period, OpenSSH does not need to re-establish a connection to the Git server, which makes pushes and pulls much faster.

Multi-Hop Connections

Oftentimes a machine is only available when SSH'ing into another machine. For example, this is the case with the DEN's Okami server, used in CIS 677 High-Performance Computing. In addition, Okami's SSH server is only available on a non-standard port. This typically results in the user going through this process:

```
smithj@local$ ssh smithj@eos01.cis.gvsu.edu
smithj@eos01$ ssh -p 43022 okami
smithj@okami$ # Finally here!
```

This is annoying and unnecessary. By using the `ProxyCommand` keyword in our config file, we can automate this process:

```
# DEN Okami
Host okami
User smithj
Port 43022
ProxyCommand ssh eos01 -W %h:%p
```

The `-W` flag allows us to hop through the first host to the host and port specified by the variables (`okami:43022`). Note that the use of `eos01` here requires presence of the aliases set up in Hostname Aliases.

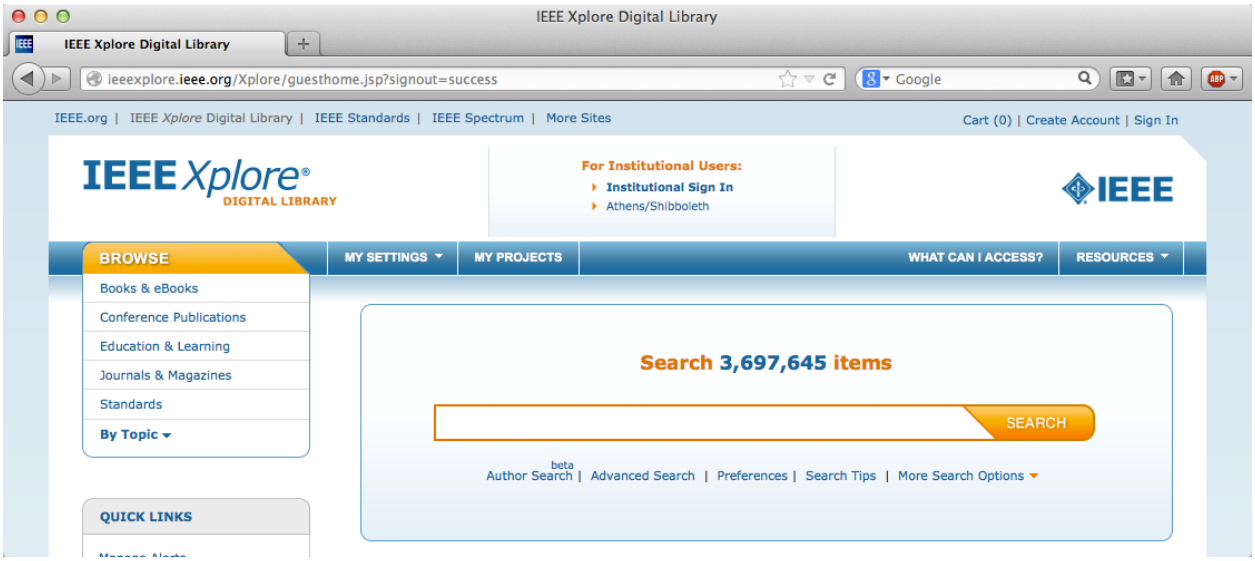
The process has now been simplified to:

```
smithj@local$ ssh okami
smithj@okami$ # Yay! Easy!
```

Using SSH as a Proxy

It is also possible to use SSH as a proxy for all network traffic. This can be useful if there are resources available from the SSH server that are not available from the local machine.

An example of such a resource is the IEEE Xplore Digital Library, which contains technical articles targeted at computer scientists and engineers. GVSU subscribes to this library, but access to the subscription is only available while *on campus*. If you try to access it off campus, you will see the following:



By using a proxy through the EOS machines, we can transparently access the IEEE library as if we were on campus.

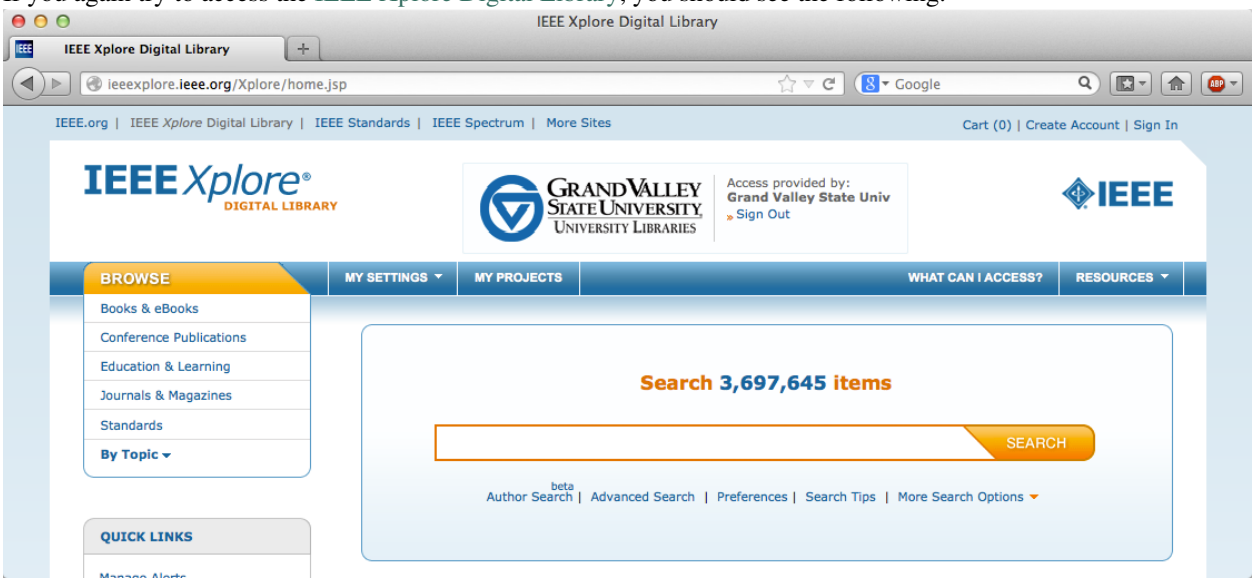
OpenSSH support the SOCKS protocol for proxying. Activating the SOCKS feature is accomplished with the `-D` flag like so:

```
ssh -D 5555 eos01
```

This establishes a SOCKS proxy with EOS01 served up on the local machine on port 5555. Now we must configure our operating system or browser to use this proxy.

On Mac OS X, configuring your system to use our SOCKS proxy is quite simple. First, open *System Preferences*. From here, choose *Network* → *Advanced...* → *Proxies* → *SOCKS Proxy*. Under the label *SOCKS Proxy Server*, enter `localhost` and `5555` to match the port passed to the `-D` flag. This can be any port as long as these numbers match. Check the box next to *SOCKS Proxy*, then click *OK* and *Apply* to turn on the proxy.

If you again try to access the IEEE Xplore Digital Library, you should see the following:



You have now successfully used OpenSSH to establish a SOCKS proxy!

Warning: By using a SOCKS proxy, *all* your network traffic is sent through the proxy. This has two implications:

- Network access will likely be slower.
- GVSU will be able to monitor your traffic as they do when you are on campus.

Please keep this in mind when using the proxy feature.

Example

For an example OpenSSH configuration file, see Sean's SSH config.

4.3.5 Alternative Clients

Chicken is not the only VNC viewer available for Mac OS X. Some alternatives are:

- TigerVNC is a capable free and open source VNC viewer. Its interface is not as Mac-friendly as Chicken, but it works well. If you are having problems with Chicken, try TigerVNC.
- RealVNC Viewer is a freeware viewer, but requires registration. RealVNC also offers RealVNC Viewer for Google Chrome, a free Google Chrome extension which does not require registration.
- JollysFastVNC is a full-featured VNC client with trial and paid versions available.
- Chicken of the VNC is an older version of Chicken and is not recommended.

Danger: Another popular client for SFTP is FileZilla. However, because FileZilla stores its passwords insecurely, we cannot recommend it. Please avoid its use to keep your password secure.

4.4 GNU/Linux

4.4.1 Shell Access (SSH)

The most popular implementation of the SSH protocol on GNU/Linux is OpenSSH. The SSH client can be run from the command-line and is simply called **ssh**.

Many GNU/Linux distributions come with OpenSSH pre-installed. If your GNU/Linux distribution does not have it installed by default, please install it with your package manager. You should not install this software from scratch or on your own.

On Debian-based systems (Ubuntu, Linux Mint, and friends), run the following command:

```
sudo apt-get install openssh-client
```

On Red Hat-based systems (Fedora, CentOS, RHEL, and friends), run the following command:

```
sudo yum install openssh-clients
```

For other distributions (Arch, etc.), the package name and command should be similar. Consult your package management tool for details.

Logging in

First, open your terminal emulator to start a shell. To connect to a specific machine, pass your username and hostname to the command-line client in the form *user@host*. For example, where `XX` is the number of the chosen machine, run:

```
ssh smithj@eosXX.cis.gvsu.edu
```

Checking Host Fingerprints

When logging in to an EOS machine for the first time, you will see a message like this:

```
The authenticity of host 'eos01.cis.gvsu.edu (148.61.162.101)' can't be established.
RSA key fingerprint is 6d:29:fd:23:c5:26:c7:c9:a5:6e:6e:c2:34:60:ea:54.
Are you sure you want to continue connecting (yes/no)?
```

This is your SSH client requesting you to validate the identity of the machine to which you are connecting.

Each EOS machine has a so-called fingerprint, a series of characters which is used to verify its identity. To ensure that an attacker between your client and the actual EOS machine is not pretending to be an EOS machine, you must check that the machine's fingerprint matches the table below. Please report any mismatches to Ira Woodring immediately.

Host	Fingerprint
arch01	41:a3:8e:22:65:e7:cb:35:0a:51:9c:0f:cc:83:9c:20
arch02	41:e0:df:d1:e4:bd:b3:7f:c0:d4:15:dc:d1:dc:39:bf
arch03	64:fb:e5:5f:94:e4:d4:56:9d:8d:27:a1:a8:d8:e4:02
arch04	88:45:6e:3e:e5:20:7b:90:7d:ac:fe:f4:8f:46:3b:4e
arch05	f3:a7:a0:14:92:8d:29:0c:ab:d9:4d:d2:73:01:1e:f0
arch06	fe:75:66:3f:65:fc:63:3a:ce:c5:cd:82:ec:f9:21:fd
arch07	c6:a3:27:17:16:8a:7f:0c:8e:9f:07:77:86:b1:7b:20
arch08	b5:f9:51:c8:81:45:92:70:5a:33:06:ee:f8:7b:52:52
arch09	9b:8e:0c:48:ea:73:f6:16:b6:eb:7a:66:d4:4c:43:bc
arch10	9e:0c:a3:90:c1:12:79:72:92:c6:16:cd:06:77:78:2b
dc01	ff:7b:d3:f0:35:3c:a9:58:e7:82:07:10:ba:0d:e1:24
dc02	b5:ce:1c:6e:58:ee:8b:cc:03:b1:1f:a0:8f:7d:c7:fa
dc03	68:85:4d:d5:2f:c1:3b:88:ca:8c:f8:36:68:61:c4:54
dc04	down for maintenance
dc05	35:6a:ad:56:3a:94:94:3b:97:8b:2e:11:5d:6b:1f:bf
dc06	c8:f6:6a:b5:49:21:3e:ae:11:f7:66:e3:8e:19:47:41
dc07	0e:70:b2:f5:71:aa:f5:a8:45:47:68:6d:93:1e:b9:68
dc08	b2:37:6c:6c:56:04:a0:44:b7:60:e4:41:7e:c2:6e:58
dc09	14:5c:95:4d:f2:7c:10:8b:10:6c:f7:5d:f7:b0:57:6e
dc10	65:1c:eb:ca:91:0d:fe:b8:d6:e5:9d:26:83:c4:17:d8
dc11	ab:ee:8d:64:cf:27:ec:eb:23:c9:a2:27:e9:ef:2e:e4
dc12	0f:9a:31:da:ad:2a:6c:be:a6:d1:f9:2c:89:29:fc:05
dc13	b9:88:ec:49:ff:6a:18:bc:f7:d6:ca:d4:3b:3b:c9:17
dc14	10:58:24:90:13:a2:ad:2f:f6:2a:9e:b1:31:bb:f3:50
dc15	4d:bb:a8:2b:16:60:90:a9:4f:34:06:bb:d5:99:c2:44
dc16	93:33:fe:09:ac:97:a4:e4:ec:19:84:ce:ea:2a:be:ae
dc17	d4:3c:57:4a:ee:aa:13:da:b2:99:00:28:cf:86:7b:19
dc18	41:aa:1e:5b:d0:39:81:2a:36:19:fe:77:36:89:66:48
dc19	6a:fb:89:63:30:cb:93:67:f3:3a:31:7e:99:71:56:20
dc20	f9:39:c3:02:18:4f:70:11:37:bc:f1:0e:f7:0a:7c:64
dc21	65:77:1d:25:92:5e:0d:56:54:06:94:d0:bc:35:eb:68
dc22	1d:17:42:59:bc:92:d3:97:8c:d3:1e:82:a5:0f:77:ee
dc23	b6:c2:ee:92:51:b4:60:06:36:d6:2d:f5:c1:cf:96:82
dc24	6d:42:2a:11:3c:27:bf:50:25:fa:6b:cc:c2:0c:99:57
eos01	6d:31:03:c3:d4:76:ff:cc:89:dc:39:90:07:09:18:0d
eos02	6b:aa:3f:31:0f:4a:a8:c4:f8:48:db:30:8f:7e:87:db

Continued on next page

Table 4.3 – continued from previous page

Host	Fingerprint
eos03	58:21:fa:89:fc:5d:ca:76:99:b3:28:53:fa:5c:73:cf
eos04	a2:cf:9f:f3:6a:ed:df:5b:79:22:d8:00:db:cc:7b:bc
eos05	b5:c2:d4:40:23:d0:0c:af:32:28:36:b0:92:cc:86:da
eos06	8d:92:40:46:3b:47:53:c6:65:96:2f:9e:99:00:63:d7
eos07	86:c9:3f:a4:0c:33:42:db:66:a9:4d:88:43:6a:bc:98
eos08	48:1a:76:fc:ea:89:42:fb:01:9d:9c:94:87:2e:18:9d
eos09	47:0e:ab:d3:9d:56:09:60:93:3f:7f:e1:75:da:4a:a9
eos10	5d:89:88:4b:41:b8:3c:b3:e6:a7:97:33:18:31:a7:8c
eos11	2b:2c:b6:e1:0e:23:62:d0:32:9a:60:64:98:c8:6e:1b
eos12	c2:ed:49:40:72:49:6c:57:cf:f3:17:9b:04:18:f1:e1
eos13	62:b0:46:b3:9d:77:f4:6d:17:ad:53:b6:36:6b:72:24
eos14	cb:39:d3:5a:b2:84:a8:63:0d:cf:7c:40:ff:b6:1a:4d
eos15	10:fd:78:c3:37:79:a5:eb:ed:71:bd:d2:ee:3e:b7:ab
eos16	bb:8a:b7:f7:d5:64:d7:6a:21:d9:e6:0a:f6:9e:3c:09
eos17	e2:be:c4:1f:fc:b5:8a:ab:3d:b8:31:6a:f8:4a:4c:ae
eos18	df:d4:13:e5:74:71:77:0b:f6:5c:58:a5:b4:00:c4:c2
eos19	e2:2c:1e:20:a0:00:c9:38:c7:85:58:f2:8c:d5:71:bc
eos20	b0:63:3d:46:cc:a6:75:47:ea:0b:92:cf:26:9f:c6:54
eos21	df:ac:2b:cf:b1:1f:65:1c:c4:23:ff:b1:89:e0:08:a5
eos22	6e:1c:31:0b:37:12:56:32:e4:0d:7c:52:9f:3f:3d:ef
eos23	03:d0:ec:be:74:75:c7:b9:e6:b3:bc:b1:b2:db:10:cf
eos24	4c:b4:c5:36:ee:f2:5c:87:55:4f:a6:28:7b:80:c2:af
eos25	86:7a:af:f0:a6:ea:70:e4:69:6c:13:62:ac:59:2a:28
eos26	47:07:b9:d6:c5:a6:48:f7:7c:b5:3c:9a:48:d8:a0:c6
eos27	89:7a:06:61:87:b5:8e:df:9e:93:d2:26:a0:a4:b5:19
eos28	74:e2:00:99:ce:b1:ca:df:70:b5:6d:64:99:e4:1f:eb
eos29	07:07:e7:ff:c9:1c:31:11:2a:ad:80:69:d6:90:ee:cf
eos30	2d:d1:63:05:69:39:32:77:49:bf:d7:f4:60:93:62:6a
eos31	53:9c:a6:98:b7:1e:55:55:29:92:06:75:4b:e3:23:46
eos32	c0:2a:f1:6c:41:52:f8:49:5f:5c:c7:bb:a6:f2:85:29

Password-less Logins (SSH keys)

It is often handy to be able to SSH into a host without having to type a password, for instance as part of a script. First, generate your public/private key pair with:

```
ssh-keygen
```

Accept the default values by pressing `Enter` at each prompt unless you know what you are doing. Once the keys have been generated, you can copy the public key over to the remote system by entering:

```
ssh-copy-id smithj@eos01.cis.gvsu.edu
```

When you SSH into EOS now, you should be able to do so without having to provide a password:

```
ssh smithj@eos01.cis.gvsu.edu
```

Note: In this setup, we created our public/private key pair without a passphrase, which is less secure. If you would like to use a passphrase, please see Mark Hershberger's excellent guide to ssh-agent and Github's guide to SSH passphrases.

Tunnelling / Port Forwarding

The SSH protocol possesses a special feature which allows it to tunnel other protocols within itself. This is called tunnelling or port forwarding. SSH can forward local ports (allowing the local machine access to resources on the remote machine) and remote ports (allowing the remote machine access to resources on the local machine).

Local port forwarding is the more used feature, and is explained in the following sections. Remote port forwarding is similar but is outside the scope of this guide.

Port forwarding can be accomplished with OpenSSH by passing arguments to the command-line client or by editing the client configuration file.

Forwarding on the Command Line

Forwarding local ports on the command-line can be accomplished using the following syntax:

```
ssh -L local_port:remote_host:remote_port user@host
```

For example, to access a web server running on port 8000 on `eos01.cis.gvsu.edu` from your machine on port 5555, use the following command line:

```
ssh -L 5555:eos01.cis.gvsu.edu:8000 smithj@eos01.cis.gvsu.edu
```

You can test the forwarding by running this in the SSH prompt:

```
python -m SimpleHTTPServer
```

and opening http://localhost:5555/ in your local web browser. You should see a web listing of your home directory! Press `Control-C` to kill the web server.

The remote host which is hosting the resource need not be the EOS machine to which you are connecting with SSH. For example, to access the CIS web server through your SSH tunnel, you can run:

```
ssh -L 5678:cis.gvsu.edu:80 smithj@eos01.cis.gvsu.edu
```

and visit http://localhost:5678/ in your local web browser. The CIS home page should appear!

Forwarding in the Config File

The command-line works well for one-off tunnels, but for frequently established tunnels, it pays to alter the OpenSSH client configuration file. The OpenSSH client configuration resides on your local machine in the file `~/.ssh/config`. This is a file inside a hidden directory inside your home directory. To setup this directory and file, please run the following commands:

```
umask u=rwx,go=
mkdir -p ~/.ssh
touch ~/.ssh/config
```

The umask command ensures that the directory and file are created with the correct permissions. This is important when dealing with SSH-related files.

Now open the file `~/.ssh/config` in a text editor.

To establish the CIS web server forwarding shown in the last section, one could use the following configuration:

```
Host eoscisweb
HostName eos01.cis.gvsu.edu
```

```
User smithj
LocalForward 5678 cis.gvsu.edu:80
```

To use this host from the command line, simply type:

```
ssh eoscisweb
```

4.4.2 Graphical Access (VNC/X11)

VNC

First, we need to create a tunnel in order to forward VNC through our SSH connection. The remote port to which we must connect depends on the desired resolution of the remote desktop. Select a desired resolution from the following table, and note the port to which it corresponds.

Display	Port	Geometry
0	5900	1280x1024
1	5901	1024x768
2	5902	800x600
3	5903	640x480
4	5904	1440x900
5	5905	1280x800
6	5906	1152x864
7	5907	1680x1050
8	5908	1920x1200
9	5909	1400x1050
10	5910	1440x1000
11	5911	1024x600
12	5912	1600x900
13	5913	1920x1080
14	5914	1360x768

In the following instructions, replace `REMOTE_PORT` with the port that you have selected.

To create the tunnel, use the following command line:

```
ssh -L 5900:eosXX.cis.gvsu.edu:REMOTE_PORT smithj@eosXX.cis.gvsu.edu
```

Or the following configuration file:

```
Host eosvnc
HostName eosXX.cis.gvsu.edu
User smithj
LocalForward 5900 eosXX.cis.gvsu.edu:REMOTE_PORT
```

If you used the configuration file, run the following to create the tunnel:

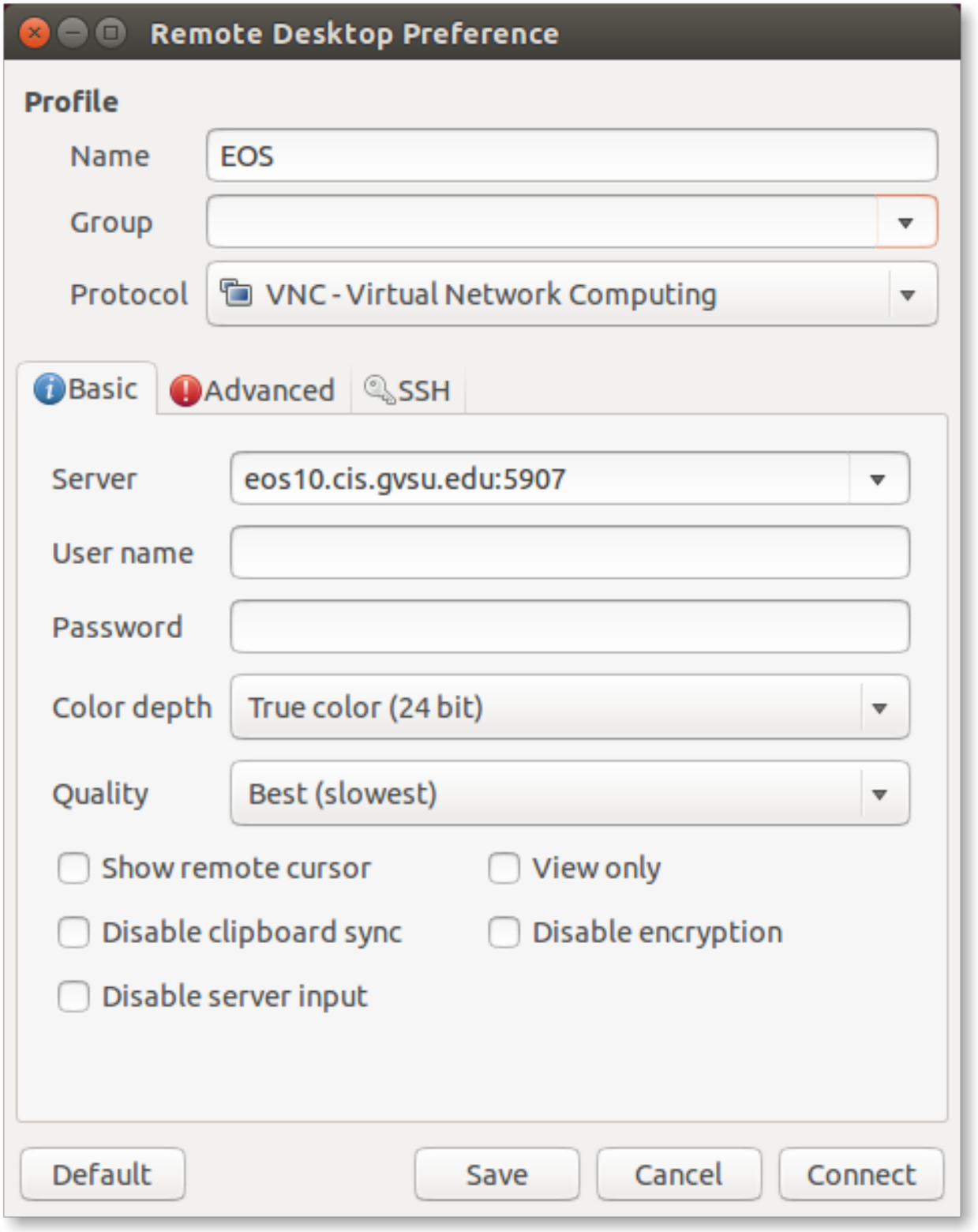
```
ssh eosvnc
```

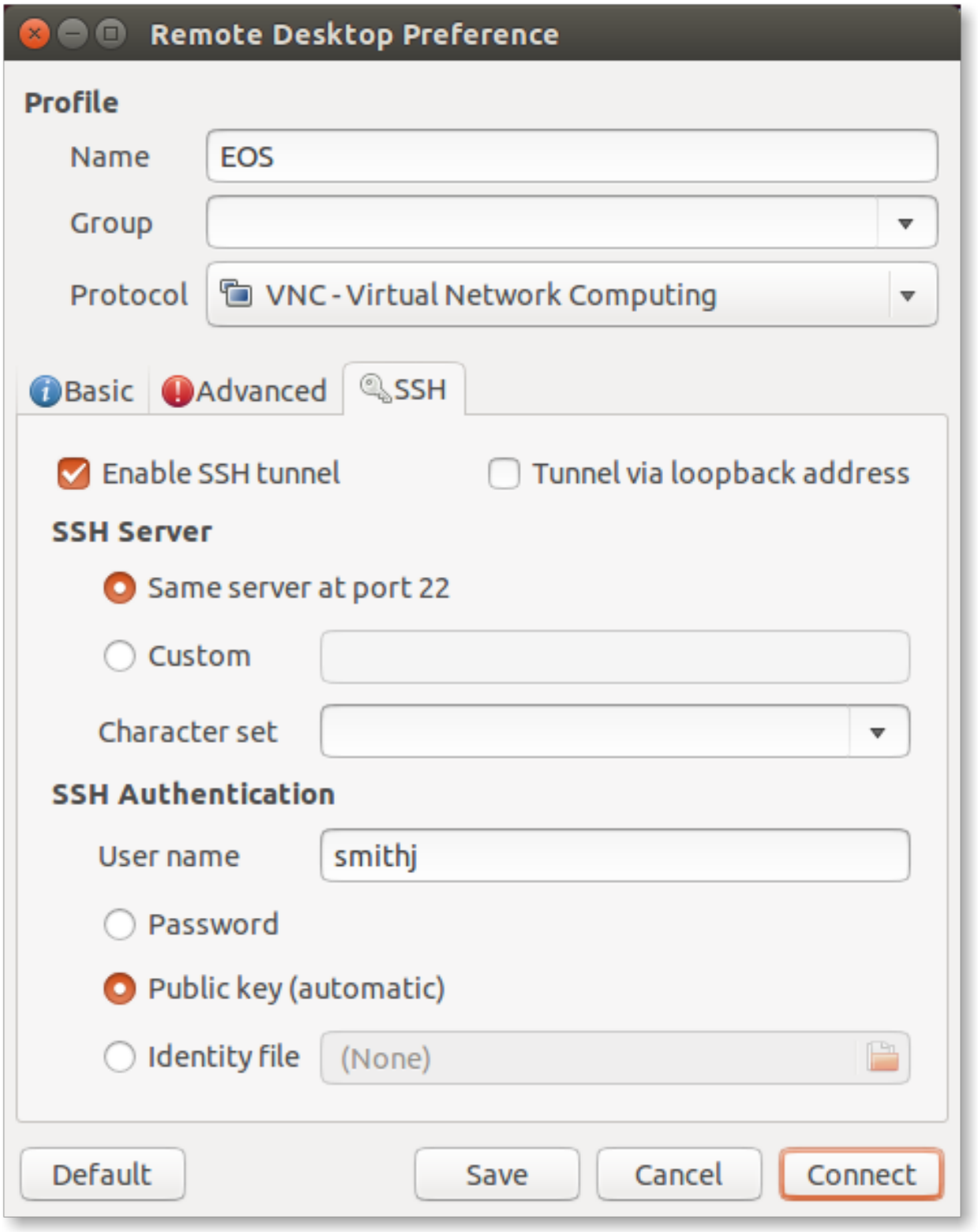
You are now ready to tunnel your VNC session.

There are a number of VNC clients for GNU/Linux, but the most capable and intuitive is Remmina (formerly tsclient). Remmina also supports RDP, so you can use it with *Winserv*. It installed by default in Ubuntu 14.04. If it is not installed in your distribution, you should install from your package manager.

In addition, Remmina supports automatic SSH tunneling. You do not need to establish a tunnel beforehand as shown in the previous section. However, if you need a shell or otherwise want to do it that way, there is nothing stopping you as it works just as well.

To configure Remmina for VNC with automatic SSH tunneling, open Remmina and select *Connection* → *New* to create a new connection. Under the *Basic* and *SSH* tabs, respectively, enter the following information. This configuration uses EOS10 and port 5907, but use the host of your choice and the port which matches your resolution from the previous section. Because *Public Key* is selected, if you have set up password-less login earlier, the login should be automatic.




Remote Desktop Preference
Profile
Name
EOS
Group
Protocol
VNC - Virtual Network Computing
Basic
Advanced
SSH
Enable SSH tunnel
Tunnel via loopback address
SSH Server
Same server at port 22
Custom
Character set
SSH Authentication
User name
smithj
Password
Public key (automatic)
Identity file
(None)
Default
Save
Cancel
Connect

X Forwarding

Almost all GNU/Linux distributions come pre-installed with a fully functional X server (typically X.Org), so no installation is needed.

The `ssh` program needs to be told to initiate X forwarding when the connection is started. This can be done with the `-X` command-line flag:

```
ssh -X smithj@eosXX.cis.gvsu.edu
```

This can also be accomplished in the SSH configuration file:

```
Host eosx
HostName eos01.cis.gvsu.edu
User smithj
ForwardX11 yes
```

From your remote shell, try running a graphical program:

```
gedit
```

You may want to send the program to the background to regain use of the shell:

```
gedit &
```

X11 is a heavyweight protocol, and X11 sessions function best on high bandwidth, low latency connections. Remote applications running through X forwarding will typically be much less responsive than if they were running on an EOS machine. If you experience performance problems (and you probably will, depending on the applications that you use), consider using VNC. X forwarding is good for one-off applications, like viewing images or PDFs, but typically not good for editing text, web browsing, or running full desktop sessions. Always keep this in mind when using this technology.

4.4.3 File Transfer

Graphical

GNOME

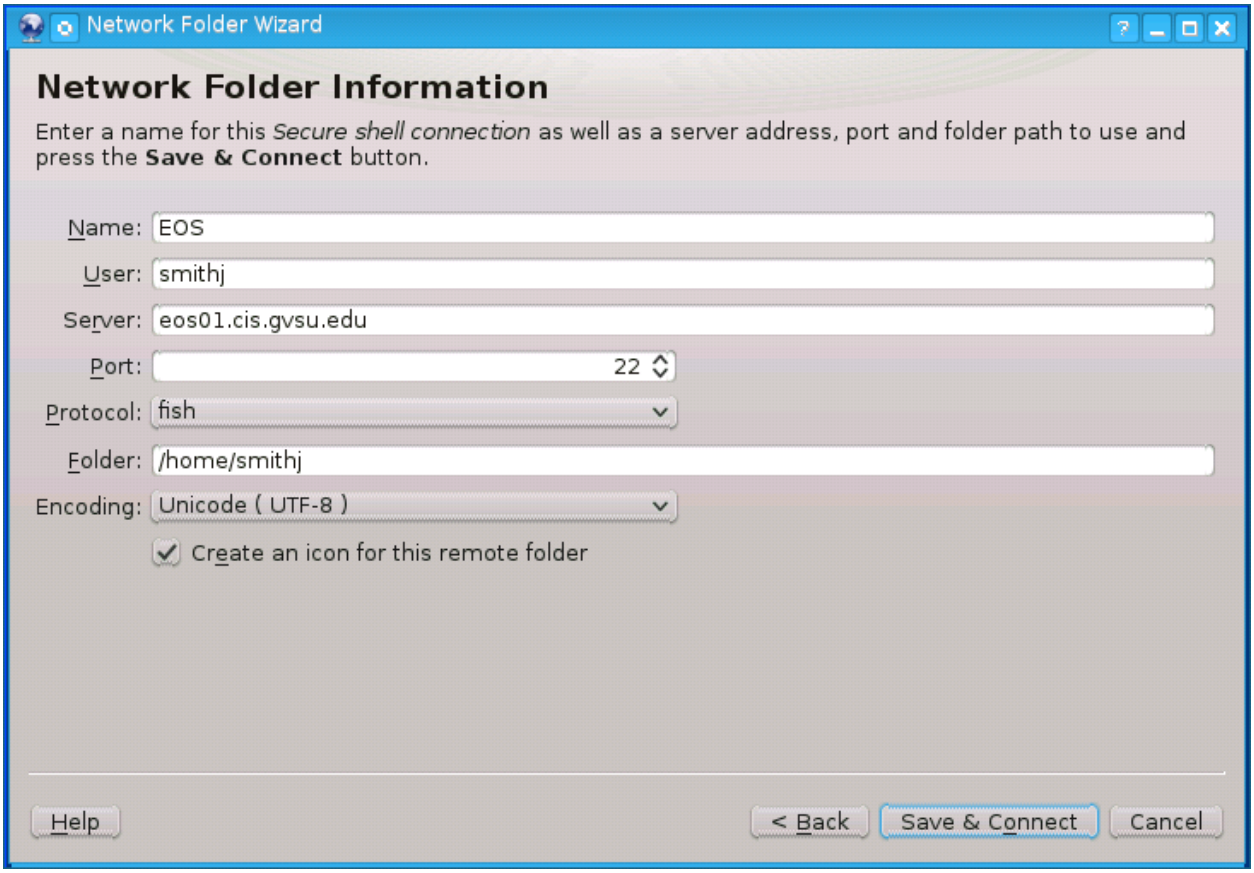
The GNOME desktop features the GNOME Virtual Filesystem (GVFS), which has native support for SFTP. First, start up the file manager (GNOME Files, formerly known as Nautilus). On the left pane, click *Network* -→ *Connect to Server*. In the *Server Address* field, enter `sftp://smithj@eos01.cis.gvsu.edu`. Your files on EOS should be displayed.

If you need to access these files from the command line, you can find your EOS account mounted as a subdirectory of this directory:

```
ls $XDG_RUNTIME_DIR/gvfs
```

KDE

KDE's file manager, Dolphin, also support SFTP URLs. First, click *Places* -→ *Network* in the pane on the left. Then, click *Add Network Folder*. Select *Secure shell (ssh)* as the type and click *Next*. Fill out the form as follows, then click *Save & Connect*:



Both the SFTP and FISH protocols *should* work, although we have had the most luck with the FISH protocol.

Other Desktop Managers

This guide only covers GNOME and KDE. However, many GNU/Linux file managers are very similar, and most offer support for SSH file transfer. Consult your desktop's documentation for more information, or just try to figure it out on your own.

Command Line (SCP)

Files can be transferred on the command line using a utility called SCP. Because SCP uses SSH for authentication, if you have set up *Password-less Logins (SSH keys)*, you will not have to type any passwords. SCP stands for *Secure Copy* and works very similar to the `cp` command, except that it can also transfer files across the network. Make sure you are familiar with the operation of `cp` before using SCP.

Each file can be prefixed with a machine name, which tells SCP where it is or should be located. Files with no prefix are assumed to be on the local machine. Paths on the remote machine start at your home directory, so there is typically no need to include `/home/smithj` in the path. Here are some examples of use of SCP:

```
# Typical upload
scp classes/cis162/hw1.txt eos01.cis.gvsu.edu:classes/cis162
# Typical download
scp eos01.cis.gvsu.edu:classes/cis162/hw2.txt classes/cis162
# Upload a directory
scp -r projects eos01.cis.gvsu.edu:classes/cis163
# Include username as well
```

```
scp smithj@eos01.cis.gvsu.edu:classes/cis162/hw3.txt classes/cis162
# Hostname aliases make this easier
scp eos01:classes/cis162/hw4.txt classes/cis162
```

rsync

rsync is the go-to tool for remote synchronization. rsync is the best choice for file transfer when the desire is to mirror a large file or directory of data, transferring it between your local machine and an EOS system. The rsync program is almost always installed by default.

Operation of rsync is very similar to SCP. The main difference between rsync and SCP is that rsync features a remote-update protocol which transfers only the differences between the local and remote files, decreasing sync time and bandwidth.

A typical use of rsync is as follows:

```
rsync \
    --verbose \
    --archive \
    --compress \
    classes/cis163/projects \
    eos01.cis.gvsu.edu:classes/cis163/projects
```

This syncs a local projects directory with a remote one. The `--verbose` option instructs rsync to tell you what it is doing, the `--archive` option tells rsync to duplicate the files almost exactly, and the `--compress` option tells rsync to use compression within the protocol.

Here is another example for deploying a website. Specifically, this deploys a Mastering EOS build to your user's personal website:

```
rsync \
    --verbose \
    --archive \
    --compress \
    --delete \
    --chmod=go=rX \
    build/website \
    eos01.cis.gvsu.edu:public_html/mastering-eos
```

The `--delete` option tells rsync to delete files on the remote machine not present on the local machine. This allows true synchronization of the two directories. However, be careful with this option, as it will delete files without confirmation!

The `--chmod=go=rX` option tells rsync to set the permissions of the files to represent the correct permissions for use on a website.

Because rsync command lines can get quite long, it is often useful to record them in a script. See *rsync(1)* by running `man rsync` for more information on the use of rsync.

There is also an rsync GUI available called Grsync, though we cannot attest to its effectiveness.

SSHFS

A further alternative to transferring files is to use the Secure Shell Filesystem, SSHFS. SSHFS is based on Filesystem in Userspace, FUSE. By using SSHFS, you can mount your EOS home directory on your local machine as a separate drive. This allows you to in effect run programs within your local machine on files within your EOS account. It is

very similar to editing files on a flash drive. SSHFS can be used for any purpose, but is especially useful for web development on EOS.

Installation

SSHFS is popular package and is usually available through your operating system's package manager.

On Debian-based systems (Ubuntu, Linux Mint, and friends), run the following command:

```
sudo apt-get install sshfs
```

On Red Hat-based systems (Fedora, CentOS, RHEL, and friends), run the following command:

```
sudo yum install fuse-sshfs
```

For other distributions (Arch, etc.), the package name and command should be similar. Consult your package management tool for details.

To mount your EOS home directory, first create a mount point for it:

```
mkdir ~/eos
```

Next, mount your EOS home directory using SSHFS:

```
sshfs -o volname=EOS smithj@eos01.cis.gvsu.edu: ~/eos
```

Tip: If you set up *Hostname Aliases*, you can use these with SSHFS:

```
sshfs -o volname=EOS eos01: ~/eos
```

Test the mount point by listing your EOS files:

```
ls ~/eos
```

You should now be able to use files on your EOS account as if they were on your own machine. For example, you can open and browse your files using your file browser:

```
xdg-open ~/eos
```

Be aware that there may be some lag in the filesystem, especially when using programs which access the filesystem frequently.

When finished with the mount point, make sure to unmount it:

```
fusermount -u ~/eos
```

SSHFS accepts many options which can be viewed with `man ssh` or `sshfs --help`. For example, to enable caching and automatic reconnection (recommended):

```
sshfs -o auto_cache,reconnect smithj@eos01.cis.gvsu.edu: ~/eos
```

If you use this command often, you may want to create script or alias for it.

For another guide to SSHFS, check out DigitalOcean's guide to SSHFS.

4.4.4 Advanced OpenSSH

OpenSSH can do much more than simply allow the user to establish connections with remote servers. If you use OpenSSH, there are a great many neat tricks available to you.

This section is based in large part on the Smylers SSH Productivity Tips blog post. Please visit this post for *even more SSH awesomeness!*

Hostname Aliases

It's useful not to have to type out the entire full-qualified domain names to EOS machines. What you might normally type would be something like this:

```
ssh smithj@eos02.cis.gvsu.edu
# or
ssh smithj@arch04.cis.gvsu.edu
```

By adding a section to the config file, this becomes easier. Add this to your `~/.ssh/config` as mentioned earlier:

```
# EOS
# Match all eos01, eos11, arch08, etc.
Host eos?? arch??
HostName %h.cis.gvsu.edu
User smithj
```

With this, now you need only type:

```
ssh eos02
# or
ssh arch04
```

Shared Connections

There is a about a minute timeout between allowed successive connnections to each individual EOS machine. This can prove very annoying when establishing multiple SSH connections to use **scp** to copy files or opening multiple terminals (but see terminal multiplexing). One way to mitigate this annoyance is by using GVSU's VPN. Another way is to used SSH shared connections. These solutions are also not mutually exclusive.

Shared connections are established by creating a socket which multiplexes multiple connections. This socket is controlled by the `ControlMaster` and `ControlPath` keywords. The first connection is called the "master" and creates the socket. Subsequent connections use the already-created socket. This behavior can be automated by setting `ControlMaster` to the value `auto`. `ControlPath` specifies the path to the socket, with variables substituted as necessary. The following config amend the previous config to add connection sharing to EOS machines.

```
# EOS
# Match all eos01, eos11, arch08, etc.
Host eos?? arch??
HostName %h.cis.gvsu.edu
User smithj
ControlMaster auto
#                         Host
#                           |Port
#                           |   | Username
#                           V   V   V
ControlPath /tmp/ssh_mux_%h_%p_%r
```

Connection sharing may be useful to enable for most hosts. However, it needs to be done with care because it typically conflicts with X forwarding and port forwarding.

Persistent Connections

It is often useful to keep connections open in the background even after the terminal has actually been closed. This is useful as it allows OpenSSH to reconnect to the server without re-establishing a connection. Turning this behavior on is trivially simple. Add the following line under the host for which you would like connections to persist:

```
# Persist connections for 2 hours.
ControlPersist 2h
```

For GitHub users, this is especially useful when using Git over SSH. Within this period, OpenSSH does not need to re-establish a connection to the Git server, which makes pushes and pulls much faster.

Multi-Hop Connections

Oftentimes a machine is only available when SSH'ing into another machine. For example, this is the case with the DEN's Okami server, used in CIS 677 High-Performance Computing. In addition, Okami's SSH server is only available on a non-standard port. This typically results in the user going through this process:

```
smithj@local$ ssh smithj@eos01.cis.gvsu.edu
smithj@eos01$ ssh -p 43022 okami
smithj@okami$ # Finally here!
```

This is annoying and unnecessary. By using the `ProxyCommand` keyword in our config file, we can automate this process:

```
# DEN Okami
Host okami
User smithj
Port 43022
ProxyCommand ssh eos01 -W %h:%p
```

The `-W` flag allows us to hop through the first host to the host and port specified by the variables (`okami:43022`). Note that the use of `eos01` here requires presence of the aliases set up in Hostname Aliases.

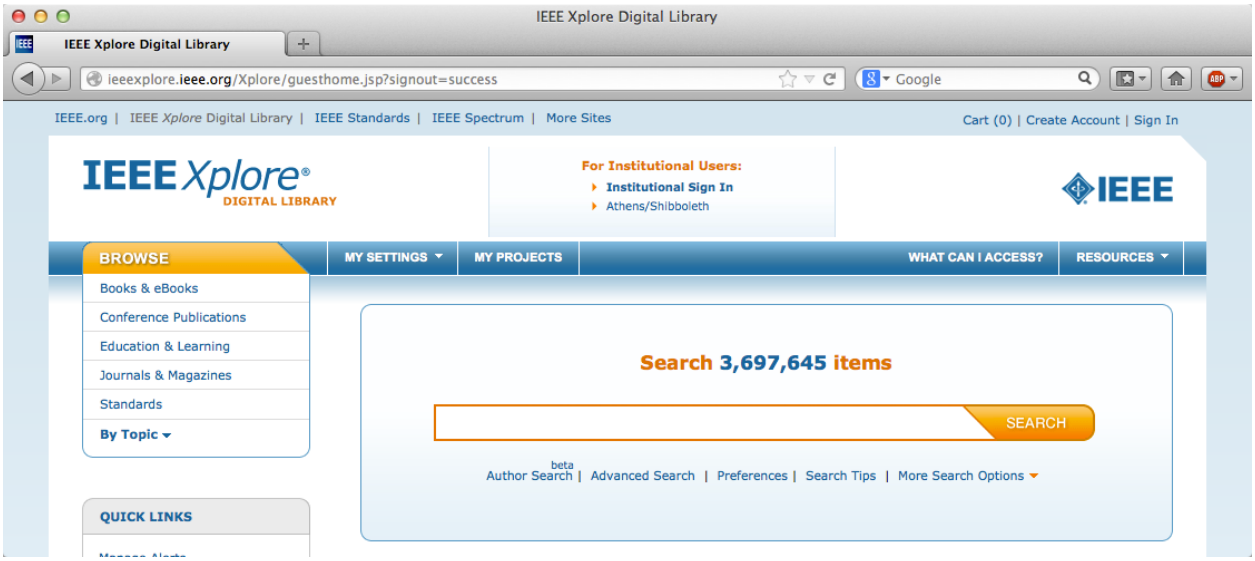
The process has now been simplified to:

```
smithj@local$ ssh okami
smithj@okami$ # Yay! Easy!
```

Using SSH as a Proxy

It is also possible to use SSH as a proxy for all network traffic. This can be useful if there are resources available from the SSH server that are not available from the local machine.

An example of such a resource is the IEEE Xplore Digital Library, which contains technical articles targeted at computer scientists and engineers. GVSU subscribes to this library, but access to the subscription is only available while *on campus*. If you try to access it off campus, you will see the following:



By using a proxy through the EOS machines, we can transparently access the IEEE library as if we were on campus.

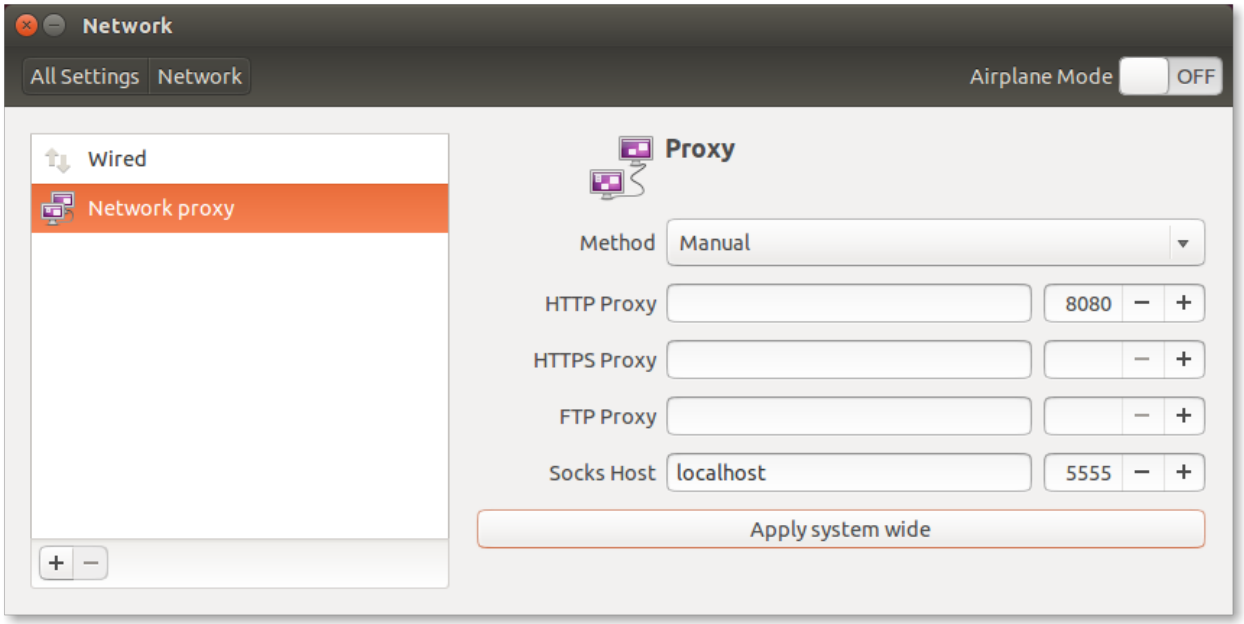
OpenSSH support the SOCKS protocol for proxying. Activating the SOCKS feature is accomplished with the `-D` flag like so:

```
ssh -D 5555 eos01
```

This establishes a SOCKS proxy with EOS01 served up on the local machine on port 5555. Now we must configure our operating system or browser to use this proxy.

Proxy configuration varies from distro to distro. These steps show how to configure a system-wide SOCKS proxy on Ubuntu 14.04, but other distros may be similar.

First, from the menu, select *System Settings* → *Network* → *Network Proxy*. For *Method*, select *Manual*, and under *Socks Host* enter `localhost` and `5555`. Then click *Apply system wide* and enter your password to turn on the proxy. To turn off the proxy, switch *Method* back to *None* and click *Apply system wide*.



If you again try to access the IEEE Xplore Digital Library, you should see the following:

You have now successfully used OpenSSH to establish a SOCKS proxy!

Warning: By using a SOCKS proxy, *all* your network traffic is sent through the proxy. This has two implications:

- Network access will likely be slower.
- GVSU will be able to monitor your traffic as they do when you are on campus.

Please keep this in mind when using the proxy feature.

Example

For an example OpenSSH configuration file, see Sean's SSH config.

4.4.5 Alternative Clients

We have tried various VNC clients, but found Remmina to be the easiest to use. However, other VNC clients for GNU/Linux exist and include:

- KRDC — free and open-source, part of KDE
- Vinagre — free and open-source, part of GNOME
- TigerVNC — command-line based, free and open-source
- RealVNC Viewer — free and paid versions available
- RealVNC Viewer for Google Chrome — free Google Chrome extension

Operation of each of these applications is similar. For the host, enter in the hostname of the EOS machine to which you have SSH'ed. If a display is requested, enter `0`; if a port is requested, enter `5900` (these mean the same thing). If the viewer offers support for multiple protocols, make sure you select "VNC".

Danger: Another popular client for SFTP is FileZilla. However, because FileZilla stores its passwords insecurely, we cannot recommend it. Please avoid its use to keep your password secure.

CHAPTER

FIVE

USER-LEVEL SOFTWARE INSTALLATION

The EOS Labs already have many packages installed to augment the base system. However, it is entirely possible that your work requires software which is not already installed. In this case, one option is to request the installation of this software from Ira Woodring. This is a good option if you feel the software would also be useful to others. If the software is primarily for personal use, however, you are advised to first attempt a user-level installation of the software.

Begin by reading about the *The Standard Hierarchy*, and then try your hand at *Manual Installation*. Once you have succeeded at installing the examples manually, give *Linuxbrew* a try.

5.1 The Standard Hierarchy

Before installing software on your own, it is important to understand the concept of the standard hierarchy. The standard hierarchy is a way of organizing files on the filesystem such that it may be used cohesively by multiple programs. If you are a Windows or Mac OS X user, you may be used to programs having their own subdirectories in `C:\Program Files` or `/Applications`, respectively. However, Unix-like machines do not typically work in this way, electing instead to separate installed files by type or purpose.

The root hierarchy, `/`, is the directory which contains all other files. Files which are absolutely essential to the system's operation may be installed here. The primary hierarchy for programs used by a standard user is the `/usr` hierarchy. Both these hierarchies can contain each of the following directories:

Name	Purpose
`bin`	Program executable files; *bin* stands for *binary* which is another name for *executable*
`sbin`	Programs used for system administration
`include`	Include files (headers) for the C programming language
`lib`	Shared libraries, which frequently correspond to headers in the `include` directory
`etc`	Configuration files [1]
`src`	Source code for programs installed to said hierarchy
`man`, `share/man`	Manual pages for programs installed to said hierarchy
`share/info`	Info documentation for programs installed to said hierarchy

Although there are more directories which can be present in each hierarchy, these directories are the most important ones with which to be familiar.

For more information on the standard hierarchy, please see the very well-written Filesystem Hierarchy Standard, which is the source of most of this information.

[1] Unlike most hierarchy directories which contain files related to other files in their hierarchy, configuration files in `/etc` are usually used to configure programs in many different hierarchies. Software configuration is a complex beast — consult each specific piece of software's documentation for the exact files used for configuration.

Files in the root and `/usr` hierarchies are usually readable but not writable by standard users. Standard users usually only have one directory to which persistent data can be written: their home directory. This is typically `/home/username`. As such, this is the place where a user installs their own programs. Although the system hierarchies cannot be written by a standard user, the structure of these system hierarchies is often mirrored by hierarchies created in a user's home directory.

Tip: A hierarchy is not a special directory, simply a sane way of organizing files. It can be created by a build system like *CMake* or *Autotools*, a package manager like *Linuxbrew*, or manually on your own!

Programs typically look in the system hierarchies for programs, headers, libraries, configuration files, and other data files which they may use or need. Special considerations usually need to be applied in order to make programs compile and run correctly from a user's home directory.

After installing programs to a user-level hierarchy, follow the directions in *User-level Hierarchies* in order to correctly use them.

5.2 Manual Installation

System package managers like Apt and Yum usually install pre-compiled software, binaries which are typically compiled on a build server infrastructure like Launchpad. Mac OS X and Windows users are also used to pre-compiled software, as most downloadable programs are distributed as application bundles or ready-to-run EXEs, respectively.

However, due to the myriad of different GNU/Linux distributions, software for GNU/Linux is often distributed in source code form only. This requires a potential user of the software to build the software from its source code, colloquially known as *compiling from source*. In addition, due to the lack of an accepted structure for installing user-level software, almost all software that is installed in a per-user fashion will need to be compiled from source.

5.2.1 Ready the Prefix

The hierarchy to which a program is installed is typically called its *prefix*. Most build systems will install by default to `/usr/local`, the system hierarchy typically used for custom-installed programs or programs compiled from source. However, because a standard user cannot write to this prefix, we are required to change our installation prefix to a directory within our home directory.

The prefix that we recommend for manually-installed user-level programs is `~/.local`, which is the hidden `.local` directory immediately within your home directory. This directory has some precedent, being used by Python and in the Freedesktop specifications. Setting this prefix is the primary step to successfully installing user-level software.

To ready your prefix for downloading and compiling source code, run the following command:

```
mkdir -p ~/.local/src
```

This command creates a `src` directory inside of your `~/.local` prefix. We will use this directory later for housing downloaded program source code.

5.2.2 Download and Extract

To compile and install software, you must first obtain the source code. The first step is to visit the project's web site and ascertain the location of its source code. The source code is typically distributed in a tar or zip archive, so look for files ending in `.tar.gz`, `.tar.bz2`, or `.zip`. After downloading, the files should be extracted to the source code directory, `~/.local/src`.

Example: GNU Bash and tar archives

Let's download the source code for GNU Bash, the default shell on the EOS system. First, begin by switching to the directory containing our source code:

```
cd ~/.local/src
```

The latest version of Bash at this time of writing is 4.3, so that is what we will download. Start by visiting the Bash home page. Under *Downloading Bash*, click the HTTP link. You will be taken to a directory index which contains a list of downloadable files. Scroll down to find a file named `bash-4.3.tar.gz`.

Although you can download this file directly using your browser, it is often easier to copy the URL and download on the command line. This is especially true if accessing EOS using SSH. Copy the URL by clicking *Copy Link Location* or similar in your browser, then download the source code using `wget`:

```
wget http://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz
```

This should create a `bash-4.3.tar.gz` file in the current working directory. This file is a tar archive which has been compressed with the GNU zip compression algorithm. A file of this type typically has a `.tar.gz` or `.tgz` file extension and is colloquially known as a *tarball* [*tarball* refers to tar archives of any or no compression scheme].

The contents of this archive can be extracted using the `tar` program as follows:

```
tar -xf bash-4.3.tar.gz
```

Hint: `tar` can automatically detect the compression format by the extension, so passing the compression algorithm as you may see elsewhere is usually unnecessary.

> **Warning:** Most source code tarballs are "well-behaved", meaning that they create one directory which matches the name of the tarball. This is a convention, but is by no means required. Make sure you trust the source of the tarball before extracting the files. You can view the contents of a tarball with the `less` built-in tar viewer:
>
> ```
> less bash-4.3.tar.gz
> ```

This command should have created a `bash-4.3` directory in the current working directory. Make that directory your working directory:

```
cd bash-4.3
```

You are now in the root of the GNU Bash source distribution.

Hint: If you do not care about saving the original source tarball, you can download and extract simultaneously with:

```
wget http://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz -O - | tar -xz
```

Note that you must pass the compression algorithm to `tar` because it is not able to detect the type by file extension when input is given through a pipe.

Example: EditorConfig and zip archives

For our zip example, we will download the source code for the EditorConfig C Core. EditorConfig is a project which helps developers establish formatting standards for a project (and is used by Mastering EOS!). First switch to the directory containing our source code:

```
cd ~/.local/src
```

The latest version of the EditorConfig C Core at this time of writing is 0.12.0, so that is what we will download. Visit the download page for EditorConfig C Core 0.12.0 and select the link for the source code zip archive. The project also offers a tarball download, but we will use the zip for the purposes of this example.

Download the file with `wget` as shown in the earlier example:

```
wget http://sourceforge.net/projects/editorconfig/files/EditorConfig-C-Core/0.12.0/source/editorconfi
```

This should create a `editorconfig-core-c-0.12.0.zip` file in the current working directory. This file is a zip archive just like those you may have seen on your desktop operating system. This file can be extracted using the InfoZip `unzip` utility:

```
unzip editorconfig-core-c-0.12.0.zip
```

> **Warning:** Unlike source tarballs, zip files sometimes have all files in one directory or sometimes have all files immediately in the root directory. Again, however, this is convention — make sure you trust the source of the archive before extracting the files. You can view the contents of a zip archive with `less` built-in zip viewer:
>
> ```
> less editorconfig-core-c-0.12.0.zip
> ```

This command should have created a `editorconfig-core-c-0.12.0` directory in the current working directory. Make that directory your working directory:

```
cd editorconfig-core-c-0.12.0
```

You are now in the root of the Editorconfig C Core source distribution.

5.2.3 Build the Software

Almost all professional-grade software projects use a build system for compilation and installation. A build system automates the tedious task of constructing compiler commands and installing files to the proper places. Using a build system should not be viewed as running a program which automagically produces another program, but rather as a practical solution to a real problem.

There are several build systems used by typical software on GNU/Linux. Read the following sections to learn about the different build system and how to identify and use them.

Autotools

Autotools, also known as the GNU Build System, is the build system currently used by most programs on GNU/Linux. You can usually identify a program using Autotools by the presence of a `configure` script in the root of the source distribution.

The software which makes up Autotools itself is usually not necessary to build a program using Autotools as a build system. Instead, the functionality is embedded into the `configure` script itself. Autotools build systems typically only require the presence of Make.

Example: GNU Bash

An example of a piece of software that uses Autotools is GNU Bash, the subject of our earlier example. We will compile the version of GNU Bash that we extracted earlier. Start by switching to the source code root directory if not already there:

```
cd ~/.local/src/bash-4.3
```

The next step is to create the build directory, which we'll create inside the source directory for convenience:

```
mkdir build
cd build
```

Now, we must configure the software by running the `configure` script. It is to the `configure` script that we must also pass the all-important `--prefix` option. Run the following:

```
../configure --prefix ~/.local
```

You will see many lines printed to the terminal, which is the script doing various checks on the system and compiler and adjusting the build to our specific system.

`configure` scripts typically also accept a myriad of other options, which can be viewed with:

```
../configure --help | less
```

Passing other options is typically unnecessary unless you would like to customize the build. Piping to `less` is recommended due to the usual length of the output.

> **Warning:** Note that:
>
> ```
> ../configure --prefix=~/.local
> ```
>
> will *not* work, as Bash will not expand the tilde properly unless the path is its own argument.

Important: Many build systems (including Autotools) support both *in-source* and *out-of-source* builds. In-source builds take place when the `configure` script is run in the same directory as the source code, that is:

```
./configure
```

Running the `configure` script in any other directory is referred to as an out-of-source build. Out-of-source builds are generally preferred because they allow separation of build artifacts from the source code. However, not all build systems or projects support out-of-source builds. The build illustrated in this example is an out-of-source build.

After configuring the software, it is time to build. This can be accomplished with:

```
make
```

Running this command typically produces an avalanche of output. The lines that you see printed are primarily compiler commands, which are printed as they are being run.

After running this command, you should have a workable version of the Bash shell. Test this out by running:

```
$ ./bash --version
GNU bash, version 4.3.0(1)-release (x86_64-unknown-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

The final step is to install the files generated by the build. Do this with:

```
make install
```

GNU Bash has now been installed to your home directory! You can run it with:

```
$ ~/.local/bin/bash --version
GNU bash, version 4.3.0(1)-release (x86_64-unknown-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Continue reading *User-level Hierarchies* to find out how to use it without typing the full path and to make your locally-installed version override the system version.

CMake

CMake is a popular alternative to Autotools as a build system on GNU/Linux. You can usually identify a CMake build system by the presence of a `CMakeLists.txt` file in the root of the source distribution.

The `cmake` program needs to be installed in order to build projects using CMake as a build system. It is installed by default on EOS.

Example: EditorConfig

An example of a project that uses CMake as a build system is the EditorConfig C Core, the subject of our earlier example. We will compile the version of the EditorConfig C Core that we extracted earlier. Start by switching to the source code root directory if not already there:

```
cd ~/.local/src/editorconfig-core-c-0.12.0
```

The next step is to create the build directory, which we'll create inside the source directory for convenience:

```
mkdir build
cd build
```

Now, we must configure the software by running CMake. Similar to the `configure` script, we tell CMake the install prefix at this stage. Run the following:

```
cmake -DCMAKE_INSTALL_PREFIX="$HOME/.local" ..
```

You will see various checks on the system and compiler printed to the terminal as with Autotools.

After configuring the software, it is time to build. This can be accomplished with:

```
cmake --build .
```

During the build, CMake will display which file is currently being built along with a percentage of files built on the left.

After running this command, you should have a workable version of EditorConfig. Test this out by running:

```
$ bin/editorconfig --version
EditorConfig C Core Version 0.12.0
```

The final step is to install the files generated by the build. Do this with:

```
cmake --build . --target install
```

EditorConfig has now been installed to your home directory! You can run it with:

```
$ ~/.local/bin/editorconfig --version
EditorConfig C Core Version 0.12.0
```

Continue reading *User-level Hierarchies* to find out how to use it without typing the full path.

Other Build Systems

The majority of C and C++ software that you may want to install to your EOS account likely uses Autotools or CMake as its build system. For those that don't, we recommend consulting the project's `README` or `INSTALL` file or the project's documentation or website for compilation instructions.

5.2.4 Library Dependencies

Bash and the EditorConfig C Core both compile without issue on EOS. However, programs frequently have compile-time dependencies: libraries which need to be installed before compiling the program.

As with the project itself, one option is to ask the Ira Woodring to install the library for you. If you would like to compile and install the dependency on your own, it is possible, but is currently out of the scope of this guide. Here are some hints:

- When compiling the program, you may need to set the `CPPFLAGS` and `LDFLAGS` environment variables to allow the compiler to locate headers and libraries, respectively. See the Autoconf manual on Preset Output Variables for descriptions of each of these variables. Some build systems are able to locate headers and libraries automatically in the specified install prefix.
- If you installed the libraries to your home directory, the operating system will not know to search for them there when running a program (even if that program is in your home directory). To allow the program to find its shared library dependencies at runtime, you must either set its rpath (recommended) or use the `LD_LIBRARY_PATH` environment variable (not recommended). See the following links for hints on this topic:
  - Russ Allbery's notes on Shared Library Search Paths
  - The Autoconf manual on Preset Output Variables
  - The Wikipedia entry on rpath

You can see the default paths in which the system looks for libraries by running:

```
ldconfig -v | less
```

5.2.5 Conclusion

As you can see, manual installation of programs is a complex but predictable process. This is where package managers like *Linuxbrew* become useful.

5.3 Linuxbrew

Linuxbrew is a package manager for GNU/Linux systems. The main advantage of Linuxbrew over system package managers like Apt and Yum is that it allows installation of software on a per-user basis. Linuxbrew is a Linux port of Homebrew, the popular package manager for Mac OS X. As such, some of its packages still contain Mac-specific code or do not yet build on GNU/Linux. Your mileage may vary, but in general Linuxbrew works quite well.

Important: Before using Linuxbrew, please make sure that you are comfortable with compiling and installing

software manually. Although Linuxbrew generally makes installing user-level software much easier, there is no magic — it performs the same steps as you would during a manual install. When Linuxbrew does not behave as intended, you will need knowledge of manual installation to fix the problem. What we are saying is this: do not come to Ira Woodring with Linuxbrew package installation issues unless you have first tried to compile the software on your own.

The Linuxbrew dependencies *should* already be satisfied, so you will be able to install without issue. If they are not, please talk to Ira Woodring. To install, then, please follow the installation instructions on the homepage.

After installation, run the following to uncover possible issues that you may have when installing packages:

```
brew doctor
```

Before moving forward, do your best to correct any issues reported by this command.

Installing packages with Linuxbrew is quite easy. For example,

```
brew install tmux
```

installs the latest version of tmux, the terminal multiplexer.

In this one command, Linuxbrew does a lot for you. It first installs tmux's dependency, libevent. Then it configures, builds, and installs tmux, setting the prefix to the correct location automatically. Furthermore, it sets the `tmux` executable's rpath (see *Library Dependencies*), meaning that the executable will automagically find the necessary libraries within your Linuxbrew prefix:

```
$ patchelf --print-rpath ~/.linuxbrew/bin/tmux
/home/smithj/.linuxbrew/lib
```

This command uses patchelf, which can also be installed using Linuxbrew ;)

You can use the version of tmux you just installed by typing:

```
$ ~/.linuxbrew/bin/tmux
```

Continue reading *User-level Hierarchies* to find out how to use it without typing the full path and to make your locally-installed version override the system version.

Enjoy installing packages using Linuxbrew!

5.4 User-level Hierarchies

To use programs in a user-level hierarchy effectively, certain environment variables must be manipulated. This section shows how to accomplish this for a well-functioning environment.

> **Warning:** The examples in this section assume you have followed through extraction, compilation, and installation of the GNU Bash example in *Manual Installation*.

5.4.1 Executable Path

You can always use executables installed to your home directory by typing the full path to the executable, for example:

```
$ ~/.local/bin/bash --version
GNU bash, version 4.3.0(1)-release (x86_64-unknown-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

For obvious reasons, typing the full path can get tedious if you are using the executable frequently. In addition, other utilities may assume that the executable in question is available on the `PATH` and not in a custom prefix. If you are installing an executable that is already present on the system, there is yet another consideration — you may want to override the system version with the version that you installed to your home directory. This is typically useful if you would like to use a newer version of a program than one installed to a system hierarchy.

Begin by reading *Path Manipulation* to learn how to effectively manipulate the executable path. Then, add the following line to your `.bash_profile`:

```
export PATH=~/.local/bin:$PATH
```

This line prepends the path of your locally-installed executables to the executable search path. Your executable will now not only be accessible without typing the full path, but it will also override any executables of the same name in system hierarchies.

Follow the directions for your session for *Applying the Changes*. After this, the following should yield:

```
$ command which bash
~/.local/bin/bash
```

Now you should be able to simply type:

```
$ bash
```

to start the GNU Bash installed to your home directory!

5.4.2 Man and Info Paths

Although you are now able to run your new Bash without typing the full path, the commands:

```
$ man bash
$ info bash
```

still show the Bash documentation for the system Bash. Although this may not seem like a big deal, small changes between versions of the same program can be the difference between a working and non-working script. Locally-installed, version-correct documentation helps avoid this problem. To allow `man` and `info` to find the locally-installed documentation, add the following lines to your `~/.bash_profile`:

```
export MANPATH=~/.local/share/man:~/.local/man:$MANPATH
export INFOPATH=~/.local/share/info:$INFOPATH
```

There is unfortunately some inconsistency with the location of installed man pages, which why we added both directories to the `MANPATH`. `INFOPATH` does not have these problems.

Follow the directions for your session for *Applying the Changes*. After this, the commands at the beginning of this section should bring up the correct documentation.

5.4.3 Managing Paths

Although the previous `export` commands work fine, there is a lot of repetition, especially if you would like to use *Linuxbrew* as well. In this case, the following snippet from the *Example Startup Files* is recommended as a replacement:

```
# Prepend a path to a variable if the path exists.
# $1: the path variable name
# $2: the path to check and possibly prepend
path_prepend () {
  [[ -d "$2" ]] && eval "$1="'"$2:${!1}"'
}

# Add hierarchy directories to paths. Manually-installed programs
# (~/.local) should override Linuxbrew programs (~/.linuxbrew).
#
# Inspired by
# <https://technotales.wordpress.com/2010/09/19/managing-path-and-manpath/>
for prefix in ~/.linuxbrew ~/.local; do
  path_prepend PATH "$prefix/bin"
  path_prepend MANPATH "$prefix/share/man" # usual manpage install directory
  path_prepend MANPATH "$prefix/man" # older manpage install directory
  path_prepend INFOPATH "$prefix/share/info"
done

# Personal scripts directory
path_prepend PATH ~/bin

# Unset definition of path_prepend
unset -f path_prepend

# Export path variables
export PATH MANPATH INFOPATH
```

CHAPTER

SIX

SYSTEM AND SOFTWARE INFORMATION

For various reasons, it is sometimes necessary to know the version of a piece of hardware or software that you are using on EOS. Rather than list the versions of everything here, this section instructs you on *how to obtain version information* from the piece of software or operating system.

6.1 System

6.1.1 All-In-One Tools

These tools show large amounts of information about the system, including but not limited to information on the operating system, hardware, and network.

inxi

inxi is a full-featured Bash script that can be used to obtain information about many parts of the system including the operating system and hardware. inxi is unfortunately not installed on EOS computers by default, but is not difficult to install. It is unnecessary but beneficial to be familiar with the steps involved in *Manual Installation* as well.

First, follow the instructions in *Ready the Prefix*. Then run the following commands to install the inxi script and its man page:

```
pushd ~/.local
(mkdir -p bin && cd bin && wget http://smxi.org/inxi && chmod +x inxi)
(mkdir -p share/man/man1 && cd share/man/man1 && wget http://inxi.googlecode.com/svn/trunk/inxi.1.gz)
popd
```

After installing, follow the steps for using software installed to *User-level Hierarchies*. You should now be able to run `inxi` from the command line. inxi is a command-line program, so it can be run through SSH or a graphical terminal emulator.

To show a basic, medium-length output:

```
inxi -b
```

To show most everything:

```
inxi -v 7 -Z
```

For more information on the options, run:

```
man inxi
```

HardInfo

HardInfo is far and away the best tool to obtain organized information related to the system. HardInfo displays on the operating system, kernel, hardware, peripherals, network and more. Furthermore, it can also run benchmarks on the CPU, FPU, and GPU.

Unfortunately, it is not installed by default on the EOS computers, so it must be compiled from source. Don't be afraid, though — by following these steps, you should be able to install HardInfo quickly and simply. It is unnecessary but beneficial to be familiar with the steps involved in *Manual Installation* as well.

First, follow the steps to *Ready the Prefix*. Then download the source code:

```
cd ~/.local/src
wget https://github.com/lpereira/hardinfo/archive/master.tar.gz -O - | tar -xz
cd hardinfo-master
```

HardInfo uses *CMake* as a build system, so the steps will be very similar to those shown in that section. First, create the build directory and configure the build system:

```
mkdir build
cd build
cmake -Wno-dev -DCMAKE_INSTALL_PREFIX="$HOME/.local" ..
```

> **Warning:** If you use *Linuxbrew* as a package manager, CMake may find your locally-installed version of pkg-config and fail. To remedy this, run:
>
> ```
> cmake -Wno-dev -DPKG_CONFIG_EXECUTABLE=/usr/bin/pkg-config -DCMAKE_INSTALL_PREFIX="$HOME/.local" ..
> ```

Next, build and install the program:

```
cmake --build . --target install
```

After installing, follow the steps for using software installed to *User-level Hierarchies*. Once this is done, run:

```
hardinfo
```

HardInfo is a graphical program, so make sure to run it at a physical machine or through VNC. A GUI should pop up containing copious amounts of information on different parts of the current system. Enjoy using HardInfo!

6.1.2 Specific Tools

Although the all-in-one tools provide convenient ways to access lots of information about the system, sometimes all that is needed is one specific piece of information. This can be useful for scripts or when the other information simply isn't needed.

GNU/Linux Distribution

The `lsb_release` Standard Base) command will show you information regarding your distribution:

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: CentOS
Description:    CentOS release 6.5 (Final)
Release:        6.5
Codename:       Final
```

Linux Kernel

The `uname` command will tell you about the operating system, including the Linux kernel version:

```
$ uname -a
Linux eos04.cis.gvsu.edu 3.10.0-123.8.1.el7.x86_64 #1 SMP Mon Sep 22 19:06:58 UTC 2014 x86_64 x86_64
```

The third value is the Linux kernel version.

For more information, please see the following nixCraft articles:

- Linux Command: Show Linux Version
- HowTo: Find Out My Linux Distribution Name and Version

6.1.3 References

The following links contain many commands that can be used to obtain information from the operating system.

- http://www.binarytides.com/linux-commands-hardware-info/
- http://www.cyberciti.biz/tips/linux-command-to-gathers-up-information-about-a-linux-system.html
- http://www.cyberciti.biz/faq/linux-list-hardware-information/
- http://www.cyberciti.biz/faq/linux-display-information-about-installed-hardware/

6.2 Software

Most programs respond to the `--version` option by printing their version, for example:

```
$ bash --version
GNU bash, version 4.2.45(1)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

However, this is a de-facto standard, and may not work for all programs. A notable example is Java:

```
$ java --version
Unrecognized option: --version
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

The correct way is to use the `-version` flag:

```
$ java -version
java version "1.7.0_65"
OpenJDK Runtime Environment (rhel-2.5.1.2.el7_0-x86_64 u65-b17)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Be aware that there is also no standard for displaying the version of the program, so other information may be provided.

If the program was installed with a package manager, the package manager is able to output information about the program in a standard format. Yum is the package manager on CentOS and will print the following information on a package:

```
$ yum info bash
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
Installed Packages
Name        : bash
Arch        : x86_64
Version     : 4.2.45
Release     : 5.el7_0.4
Size        : 3.5 M
Repo        : installed
From repo   : updates
Summary     : The GNU Bourne Again shell
URL         : http://www.gnu.org/software/bash
License     : GPLv3+
Description : The GNU Bourne Again shell (Bash) is a shell or command language
            : interpreter that is compatible with the Bourne shell (sh). Bash
            : incorporates useful features from the Korn shell (ksh) and the C shell
            : (csh). Most sh scripts can be run by bash without modification.
```

If you use *Linuxbrew*, it will also print information about its packages:

```
1  $ brew info bash
2  bash: stable 4.3.30, HEAD
3  http://www.gnu.org/software/bash/
4  /home/smithj/.linuxbrew/Cellar/bash/4.3.30 (59 files, 7.9M) *
5    Built from source
6  From: https://github.com//homebrew/blob/master/Library/Formula/bash.rb
7  ==> Dependencies
8  Required: readline
9  ==> Caveats
10 In order to use this build of bash as your login shell,
11 it must be added to /etc/shells.
```

Note that Linuxbrew shows the current version of the package (line 2) *and* the version that is installed (highlighted line 4) [if one is installed].

6.3 Web Server

For PHP development or general web development using EOS, it is sometimes necessary to obtain information about PHP and Apache, the web server. To do this, follow these steps to enable a so-called PHPInfo page:

```
echo '<?php phpinfo(); ?>' > ~/public_html/info.php
chmod o+x ~ ~/public_html
chmod o+r ~/public_html/info.php
```

Now visit `http://cis.gvsu.edu/~smithj/info.php` in your browser, replacing with your username where appropriate. Upon visiting the page, PHP will dump a large amount of information on itself and the web server.

While this page in itself does not present a security risk, it can be a valuable tool for potential attackers. You are therefore requested to remove the page after you have obtained the necessary information:

```
rm ~/public_html/info.php
```

Please contact Ira Woodring with further questions about PHP and the web server.

CHAPTER

SEVEN

WINSERV

Part of the department infrastructure is a Windows Server installation, called Winserv. Accounts for this machine are given as necessary — for example, when taking a course that involves projects which require the Windows platform. This server is named `winserv.cis.gvsu` and can be accessed via the Remote Desktop Protocol (RDP). The following are methods for accessing Winserv from various RDP clients.

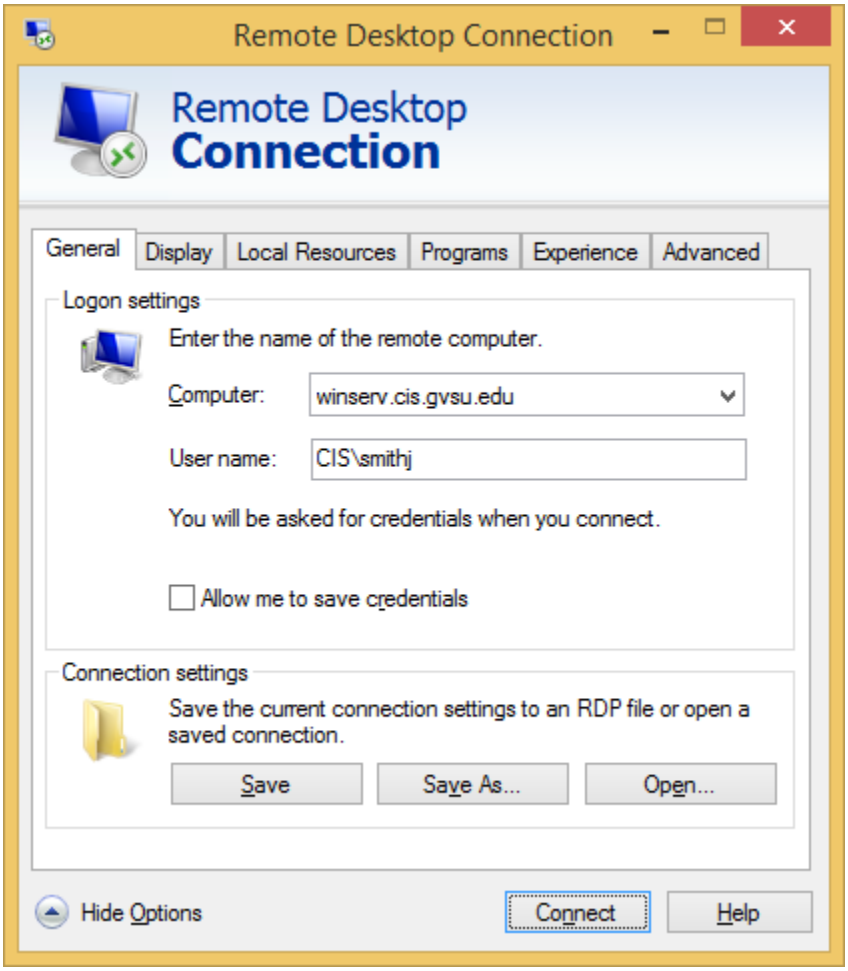
7.1 Common Settings

Regardless of the RDP client or platform you use, please remember a few things:

- The machine's IP address may change; use the DNS name instead.
- If outside of the EOS network, you will need to use the fully qualified domain name, `winserv.cis.gvsu.edu`. While on the EOS network, you can simply use `winserv`.
- You must login to the CIS domain. For instance, if your username is `smithj`, your login would be `CIS\smithj`. These may be specified together with the backslash or separately depending on your client.
- Our certificate is self-signed. You may want to instruct your client to save this information, or you will have to accept a security warning each time you login.

7.2 Microsoft Windows

All versions of Windows have the built-in Microsoft Terminal Services Client. You can find this program by searching for **mstsc** or *Remote Desktop Connection* in the Start Menu. Here is a sample configuration for Winserv:

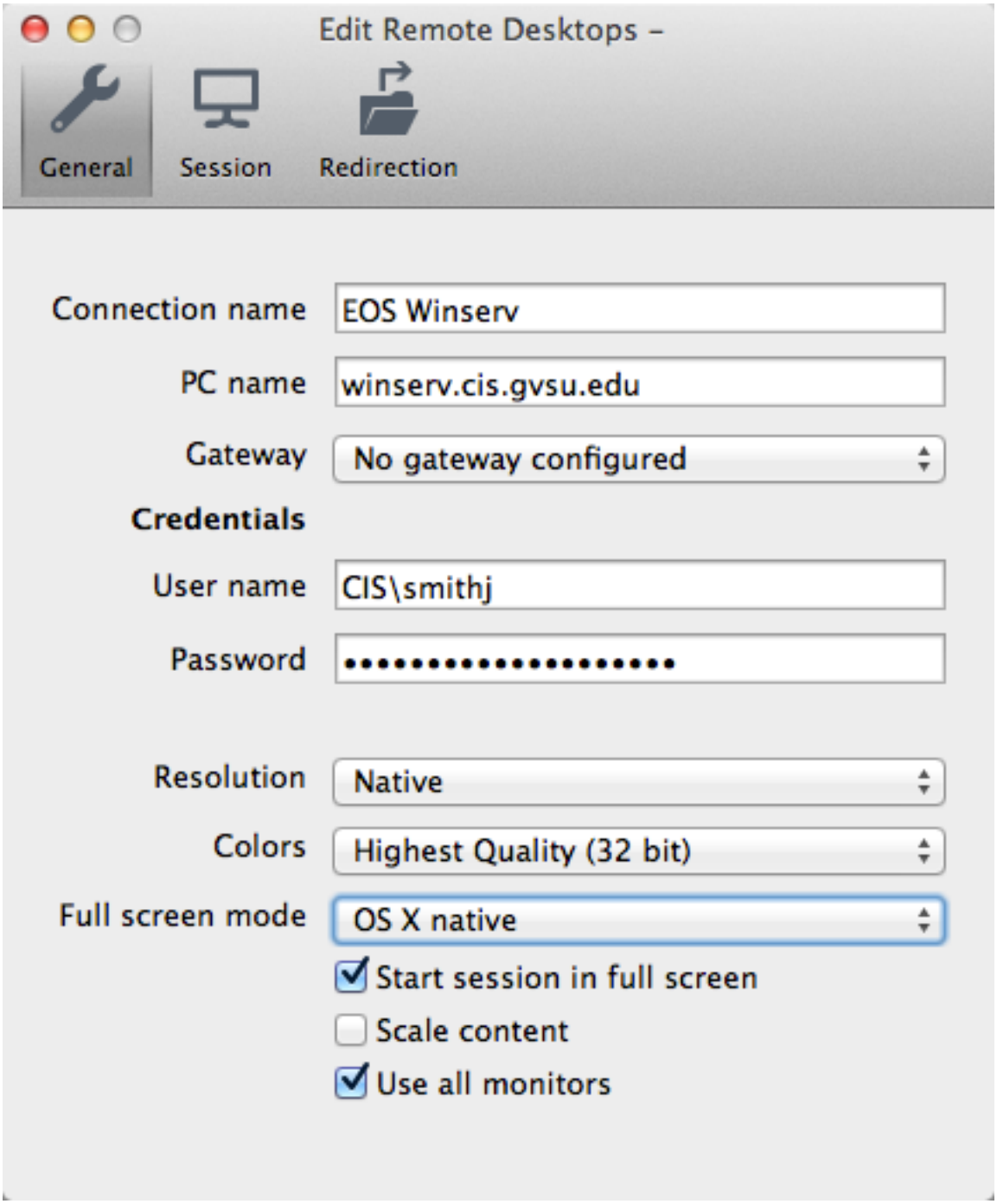


7.3 Mac OS X

Multiple options exist for Mac OS X, including the official Microsoft client and an open-source client called CoRD. Both work well and provide a similar set of features, so it is up to you which one you'd like to use. See here for a comparison of three different options including the two just mentioned.

7.3.1 Microsoft Remote Desktop

This application is available as Microsoft Remote Desktop in the Mac App Store. *Do not* attempt to download Microsoft Remote Desktop Connection Client for Mac 2.1.1, as it does not work for more recent versions of Mac OS X. You do not need to configure a gateway in order to use this client with Winserv. Here is a sample configuration:



7.3.2 CoRD

CoRD is an open-source rdesktop-based RDP implementation for Mac OS X. Although their website states that CoRD "doesn't seem to work on 10.9 Mavericks", we have had no issues with the latest version. Here is a sample configuration for Winserv:

7.4 GNU/Linux

GNU/Linux systems usually have the xfreerdp command-line tool in their repositories. If not, it can most likely be built from source. xfreerdp provides a geometry flag that accepts both screen percentages or resolutions from the command-line to help adjust the client to an appropriate size. For instance, to allow the client to take up 90% of your screen:

```
xfreerdp winserv.cis.gvsu.edu -g 90%
```

Alternatively, to force a resolution of 1024x768 pixels:

```
xfreerdp winserv.cis.gvsu.edu -g 1024x768
```

For a graphical RDP client that can also be used for *VNC*, check out Remmina.

CHAPTER

EIGHT

DATABASES

Learning to setup and maintain a database is essential to any Computer Science curriculum. The EOS infrastructure supports multiple database variants depending on the your needs. To save space and setup time database accounts are only given to those who need them for a course or a specific project. Contact your professor and Ira Woodring if you need a database account for any reason.

8.1 MySQL

MySQL is a powerful open-source database. To access it via the command-line, login to an EOS machine and enter:

```
mysql -u smithj -p -h cis.gvsu.edu
```

This attempts to log you in with the provided username, using a password, to the host cis.gvsu.edu (our MySQL server).

8.2 Oracle

Oracle is a very powerful and complex enterprise quality system. Once you have been granted access, you can access it with the command:

```
sqlplus smithj@orcl
```

Please note that when you change your password for Oracle that you must not use the `@` character. Oracle will accept this but you will be unable to login.

8.3 Oracle APEX

Oracle also provides the APEX system for web based database development. An APEX account is separate from the normal Oracle account; a password for one will not work for the other. You may login to APEX once you have been granted access by opening your web browser to the URL

http://dbserv.cis.gvsu.edu:5560/apex

You will need to provide a workspace name, username, and password. If your username is `smithj`, a sample login would be

Workspace smithj_ws

Username smithj

Password **********

Do not attempt to use the *Reset Password* feature on the APEX homepage; it has never worked properly. If you attempt to use it you will be unable to login until a system administrator can delete and recreate your account.

8.4 MSSQL

Microsoft also provides an enterprise quality database server that we provide. Microsoft's database is called MSSQL. We host MSSQL on the Winserv machine, and accounts are granted when needed.

8.5 SQLite

The SQLite system is a relational database that can exist within your home directory. SQLite is different from the above mentioned databases in that it does not operate as a client/server set of processes, but instead can be linked to the application being programmed. As many databases as need be created (within storage limits) can be created by you, as each database is merely a separate file on the filesystem.

Outside of a programming context, SQLite can be accessed from the commandline with:

```
sqlite3
```

This will provide you with a command-line interface from which you can work using SQL statements.

8.6 Remote Database Connections

It is often advantageous for programs to connect their programs to databases to do work. There are a variety of ways to accomplish this task, and many are language specific. It is of note though that our databases are not accessible from outside of our network due to firewall restrictions. However, programs running from within the EOS infrastructure can make connections to databases.

CHAPTER

NINE

EDITORS

Choosing an editor is an important step for any programmer. As the task of programming is composed primarily of writing code, you will be spending a lot of time in your editor. We encourage you to try out many different editors before choosing the one that is right for you. Once you have chosen an editor, we encourage you to spend time becoming proficient with it — it will reward you throughout your career.

This section lists various editors and explains the different classes under which they fall. These classes are not official; they are used only for the purposes of grouping like options together. All editors listed directly under their class are available on EOS; editors listed under the *Other* sections are not available on EOS but can be downloaded for other platforms (or in some cases, self-installed on EOS).

9.1 Basic Text Editors

Basic text editors are a class of editors that usually include only simple text editing features. Some include syntax highlighting for languages, but do not usually include code completion. Their extensibility is usually minimal.

9.1.1 GNU nano

GNU nano is an improved clone of the Pico text editor. Editing takes place entirely within the terminal in its text user interface. Its features are very basic, with many of the controls outlined at the bottom of its screen.

GNU nano is best suited for small edits, for example, editing your `.bashrc` or other configuration files. It is also useful over an SSH session because it operates full within the terminal.

9.1.2 gedit, KWrite, and Leafpad

gedit (GNOME), KWrite (KDE), and Leafpad (Xfce and LXDE) are the default text editors for their respective desktop environments. All have a similar feature set, including syntax highlighting and other expected features of a text editor. They are useful for simple editing tasks, but we don't recommend using any of them as a full-time code editor.

9.1.3 Others

Other basic text editors not available on EOS, listed here for completeness:

- Microsoft Windows' Notepad and WordPad
- Mac OS X's TextEdit

9.2 Advanced Text Editors

Advanced text editors are a class of editor which excel in editing quickly and efficiently. They usually allow extensive user customization and configuration, and have packages to augment their base features.

Vim and Emacs are two classic editors known to programmers everywhere. Both have a reputation for steep learning curves and massive extensibility. However, advanced users of each of these editors also have a reputation for being extremely productive.

Advanced text editors are recommended for those who have the time to learn how to be efficient with an editor. Effective use requires dedication of time *specifically to learning the editor*.

9.2.1 Vim

Vim (Vi IMmproved) is an editor inspired by the classic vi editor. One of the most popular editors of all time, Vim is included with many operating systems and available for many others. Vim is unique as compared to other popular editors in that it features a modal interface in which different keys execute different actions. Vim is customizable through Vim script as well as other languages. The first version of Vim was released in 1991, while the first version of vi was released in 1976.

As mentioned, Vim is available for many platforms including Windows, Mac OS X, and GNU/Linux.

9.2.2 GNU Emacs

GNU Emacs (Editor MACroS) is an extensible and programmable editor. It is one of the oldest and most popular editors of all time. Emacs features almost infinite customizability of almost all of its features. GNU Emacs was initially released in 1985, with an initial version of Emacs released in 1976.

Before using Emacs, check out GNU's guided tour of Emacs features. You can also check out EmacsMovies (basic) and Emacs Rocks! (advanced) for screencasts showcasing what the editor can do. After starting Emacs, press `C-h t` (that's Emacs parlance for `Control-h`, then `t`) for the in-application Emacs tutorial. Another great guide is Jessica Hamrick's Absolute Beginner's Guide to Emacs.

Like most GNU software, Emacs has a very detailed and complete manual that is available online, as well within Emacs by entering `C-h r` (that's `Control-h`, then `r`). No mention of Emacs would be complete without mentioning the gigantic resource that is EmacsWiki.

Emacs is not difficult to use out-of-the-box, but requires customization and commitment to use effectively. We recommend using vanilla Emacs for a bit, then moving on to customization when you are comfortable. A great way to

get started is to use a so-called *starter kit*, a collection of packages that include useful Emacs packages and turn on useful Emacs functionality disabled by default. Xah Lee maintains a list of Emacs starter kits. Although all of these are good, we recommend Emacs Prelude because it is full-featured and always up-to-date.

Emacs 24 includes a package manager called ELPA (Emacs Lisp Package Archive). Its use is highly recommended. Two additional package archives that any Emacs user needs to know about are Marmalade and MELPA.

Emacs is available for many platforms including Windows, Mac OS X, and GNU/Linux.

9.2.3 Kate

Kate (KDE Advanced Text Editor) is KDE's entry into the field of advanced text editors. Kate's list of features is comparable to other editors of its class.

Although Kate is primarily used on GNU/Linux through KDE, it is also available for Windows and Mac OS X.

9.2.4 Others

The following are other advanced text editors not available on EOS. You can use these on your personal machine, and you may be able to install them on EOS manually.

Atom

Atom is an open-source, customizable text editor produced by GitHub. As an editor, it seems to be in the spirit of Vim or Emacs, but using modern design principles and technologies. Atom comes with many useful features out-of-the box, including a fuzzy-finder, multiple cursors, and a package manager. Atom is based on the Atom Shell, which is in turn based on node.js and Chromium.

Atom is available on Windows, Mac OS X, GNU/Linux, and FreeBSD.

Sublime Text

Sublime Text is a free, proprietary, shareware advanced text editor. It features easy navigation within files, multiple selections, split panes, and a package manager, among other features. Sublime Text is very customizable and features a Python API.

Sublime Text is available for Windows, Mac OS X, and GNU/Linux.

Notepad++

Notepad++ is a simple open-source editor for Windows. It supports split editing, syntax highlighting, and many other features come to be expected by an editor of this class.

TextMate

TextMate is a popular advanced text editor for Mac OS X. Originally a proprietary product, TextMate is now free and open-source software. TextMate has been called the "culmination of Emacs and OS X" and features a slick but minimal interface, file browser, auto-pairing, simple completion, and many more features. TextMate has been particularly popular in the Ruby on Rails community.

9.3 Integrated Development Environments

Integrated development environments (IDEs) are the most complex class of text editors. Although editing is usually still at the forefront, the IDE's text editor is only a *component* of the larger program. Like advanced text editors, IDEs typically feature syntax highlighting, multiple panes, and many other editing features. Unlike advanced text editors, IDEs often feature deep integration with certain environments, such as semantic code completion, support for refactoring, graphical user interface (GUI) composers, and ability invoke test harnesses or debuggers directly from the editing interface. IDEs also have a focus on projects as opposed to individual files.

IDEs often focus on a specific language or environment. As such, for people who work in multiple environments, the choice is often between use of a single advanced text editor for all environments or use of an individual IDE for each environment.

9.3.1 BlueJ

BlueJ is a beginner's IDE for Java. Its interface offers the ability to selectively instantiate classes and run methods with different parameters, making it excellent for learning. BlueJ also features an object inspector which allows examination of objects as they progress through a software instance's life cycle.

BlueJ is available for Windows, Mac OS X, and GNU/Linux.

9.3.2 Eclipse

Eclipse is an extensible, feature-complete, free and open-source IDE. Although Eclipse is written in and was initially written for Java, it also supports many other languages and environments. By default, Eclipse on EOS supports Java, PHP, Python, C, C++, and Android.

Popular language/environment plugins for Eclipse include:

- Eclipse JDT (Java Development Tools)
- Eclipse CDT (C/C++ Development Tooling)
- Eclipse PDT (PHP Development Tools)
- PyDev (Python)
- NodeClipse (JavaScript and Node.JS)
- ADT (Android Development Tools)
- Aptana Studio (HTML5, CSS3, JavaScript, Ruby, Rails, PHP, Python)

Eclipse is available for Windows, Mac OS X, and GNU/Linux.

9.3.3 IntelliJ IDEA

IntelliJ IDEA (colloquially known as "IntelliJ") is a professional IDE for Java and other languages developed by developer tool company JetBrains. IntelliJ's primary advantage over other IDEs and editors is deep integration with many specific languages and technologies. IntelliJ is available in two versions: Community and Ultimate. The Community edition is free and open-source software, while the Ultimate edition includes more features for a price. For details on specific features, consult IntelliJ's advertised feature list or Wikipedia's list of IntelliJ features.

IntelliJ is available for Windows, Mac OS X, and GNU/Linux. On EOS, IntelliJ may be started from the command line with:

```
idea.sh
```

9.3.4 Visual Studio

Visual Studio is an IDE for Windows developed and maintained by Microsoft. It is primarily aimed at development of applications and libraries for Microsoft platforms, including Windows, Windows Phone, Microsoft Silverlight, and IIS. Visual Studio's supported languages include C, C++, C#, F#, and Visual Basic. Visual Studio also supports Python through PyTools and web development with HTML, CSS, JavaScript, and ASP.NET. The IDE has almost all modern features including IntelliSense code completion, refactoring support, and integrated debugger, graphical interface and web design tools, and a database editor. ReSharper by JetBrains is a Visual Studio add-on considered necessary by many .NET developers.

Visual Studio is the recommended IDE when developing exclusively for Microsoft platforms.

9.3.5 Xcode

Xcode is an IDE for Mac OS X developed and maintained by Apple. It is primarily aimed at development of Mac OS X and iOS Cocoa applications written in Objective-C and/or Swift. Xcode includes Interface Builder for composition of user interfaces, the Apple LLVM Compiler based on Clang and LLVM, a graphical debugger based on LLDB, and Instruments, a tracing and profiling tool based on DTrace. It also features excellent code completion, also implemented using Clang. For more information, see the full list of Xcode features.

In addition to Objective-C and Swift, Xcode supports development in C, C++, Python, Ruby, and AppleScript. [1]

If you are developing native Mac OS X or iOS applications, Xcode is the recommended IDE.

9.3.6 Qt Creator

Qt Creator is an IDE for developing applications using the *Qt* cross-platform framework. Qt Creator supports C++ and QtQuick/QML (JavaScript), two languages used for developing Qt applications. It also includes project navigation tools, code completion, an integrated debugger based on GDB, and a drag-and-drop interface designer (formerly known as Qt Designer). Additionally, Qt Designer supports integration with various build systems. A full feature list is available on the homepage.

Qt Creator is the recommended IDE if you are developing cross-platform applications in *Qt*. Qt Creator is available

[1] According to the Wikipedia Xcode article.

on all platforms Qt is available, which includes Windows, Mac OS X, and GNU/Linux.

9.3.7 Geany

Geany is a lightweight IDE based on Scintilla. It includes syntax highlighting of numerous languages, project support, simple code completion, and code navigation. In addition, Geany includes support for invoking build systems through external tools. Geany is a good choice if you want to use a consistent interface for many different languages and basic IDE features for development.

Geany is available for Windows, Mac OS X, and GNU/Linux.

9.3.8 Bluefish

Bluefish is a lightweight IDE primarily aimed at web development. On the web development side, Bluefish supports PHP, ASP.NET, ColdFusion, Java Server Pages (JSP), and Wordpress as well as the standard HTML, CSS, and JavaScript. In addition to that, Bluefish includes support for C, C++, Python, Ruby, and SVG, among others. One of Bluefish's interesting features is integration of external scripts by sending the document text through a pipe to the script. For more information, see the full list of Bluefish features.

Bluefish is available for Windows, Mac OS X, and GNU/Linux.

For a full list of text editors, please see Wikipedia's list of text editors and comparison of text editors.

9.3.9 Others

The following are other IDEs not available on EOS. You can use these on your personal machine, and you may be able to install them on EOS manually.

NetBeans

NetBeans is a free and open-source IDE for Java and others sponsored by Oracle. Although Java is the main focus of NetBeans, it also supports PHP, C and C++, and web development. One of the more popular features of NetBeans is its integrated GUI builder for Java/Swing.

NetBeans is available for Windows, Mac OS X, and GNU/Linux.

9.4 EditorConfig

If your project is developed by a team or you work with multiple editors, consider using EditorConfig. EditorConfig plugins for various editors allow developers to maintain a consistent formatting style throughout the projects, supporting such things as tabs vs. spaces, tab width, end of line characters, and character encoding, among others (full list here). Many editors listed in this section are supported.

All logos are copyrights of their respective projects.

CHAPTER

TEN

GUI TOOLKITS

For many programs, you may find that a console or command-line interface is not enough. In this case, consider using a library to build a graphical user interface (GUI). Below are the most common GUI toolkits and bindings to various languages. Though you may find bindings of various quality on the Net, care has been taken to include only official or high-quality maintained bindings in this list. Consider the listed bindings the ones recommended by the authors.

Using a cross-platform toolkit is best when you need to target multiple platforms for your application. All of these toolkits support use of a common code base for multiple operating systems.

Please see Wikipedia's list of GUI toolkits for a fuller list.

Note: Although you can use all of these toolkits on EOS systems, this information is not specific to EOS.

10.1 Choosing a Toolkit

Choosing a cross-platform toolkit can be a challenge. It is an important decision because it typically represents a commitment to a specific toolkit. There are a few different issues to consider before making this decision.

The first consideration is whether the toolkit uses *native widgets*. Toolkits that use native widgets (native toolkits) call platform-specific APIs to render widgets on the screen. Since native APIs are used, these toolkits often produce applications which look closer to other applications on the platform. They also sometimes offer better system integration.

Non-native toolkits draw their own widgets on the screen using graphics APIs. Because native APIs are not used, widgets typically have more consistent behavior between platforms.

A second consideration is the language of development. You will typically want to choose a toolkit that has a binding for your language of choice. However, it is not uncommon for the GUI toolkit to *determine* the language of development.

Third, some of the toolkits mentioned offer more that just GUI abstractions. Of these, Qt offers the most features beyond a GUI library. wxWidgets offers more features as well. GTK+, Tk, Swing, and SWT are purely GUI libraries (although GTK+ is typically used with GLib, while Swing and SWT are used with the Java standard library).

The final (and possibly most important) consideration is the level of maintenance of the library binding and the library itself. Because GUI libraries and application are typically high in complexity, it is highly suggested that you choose an official or well-maintained binding. Choosing a subpar library typically results in many issues and general frustration.

10.2 Qt

Qt is a high-quality cross-platform application framework which includes a full-featured GUI library. Qt applications run on Windows, Mac OS X, GNU/Linux, Android, and iOS. Qt is free to use for both open-source and commercial

applications (though some of its bindings are not). The KDE GNU/Linux desktop environment is developed using Qt. Qt does not use native widgets, but uses native style APIs to render its widgets for a look and feel that is usually consistent with the platform.

10.2.1 Bindings

- **C++:** Native (built-in)
- **JavaScript:** QtQuick/QML (built-in)
- **Python:** PyQt (free for open-source, but not commercial)

Also see Wikipedia's list of bindings for Qt 4 and Qt 5.

10.2.2 Tools

Qt has an official IDE, *Qt Creator*, which includes a code editor and GUI designer among other features.

10.3 wxWidgets

wxWidgets is a popular cross-platform GUI toolkit. wxWidgets applications run on Windows, Mac OS X, and GNU/Linux. wxWidgets and most of its bindings are free to use for both open-source and commercial applications. wxWidgets is a fully native toolkit, meaning that it uses native APIs for all of its widgets.

10.3.1 Bindings

- **C++:** Native
- **Python:** wxPython
- **PHP:** wxPHP
- **Haskell:** wxHaskell
- **Perl:** wxPerl

Also see the Wikipedia list of wxWidgets bindings.

10.3.2 Tools

wxGlade is the GUI designer tool for wxWidgets.

10.4 GTK+

GTK+ is a popular and complete cross-platform library for GUI programming. GTK+ applications run on Windows, Mac OS X, and GNU/Linux. GTK+ and most of its bindings are free to use for both open-source and commercial applications. Many popular GNU/Linux desktop environments are written using GTK+, including GNOME, Xfce and Ubuntu's Unity. GTK+ is a non-native toolkit, and renders its widgets using Cairo.

Note: Although GTK+ is purely a GUI library, it is typically used with GLib, which offers many other application features.

10.4.1 Bindings

The GTK+ Project maintains a list of language bindings and their status. The more popular ones include:

- **C:** Native (official)
- **C++:** gtkmm (official)
- **Python:** PyGObject (official) [note: PyGTK not recommended for new programs]
- **JavaScript:** Gjs and Seed
- **C#:** Gtk# (official, but incomplete)

Also see the Wikipedia list of GTK+ bindings.

10.4.2 Tools

Glade is the official GUI designer tool for GTK+.

10.5 Tk

Tk is a relatively basic cross-platform GUI toolkit, and its applications run on Windows, Mac OS X, and GNU/Linux. Tk and most of its bindings are free to use for both open-source and commercial applications. Tk is a non-native toolkit, but does include the option to emulate the styles of certain platforms.

10.5.1 Bindings

- **Tcl:** Native
- **C:** Native
- **Python:** Tkinter
- **Perl:** Perl/Tk
- **Ruby:** Ruby/Tk

10.6 Swing

Swing is the most popular GUI toolkit for Java and is part of the Java standard library. As part of Java, Swing applications run for the most part wherever Java runs. Swing is a non-native toolkit, and draws all of its widgets using Java graphics APIs.

10.7 SWT

SWT (Standard Widget Toolkit) is cross-platform GUI toolkit for Java programs and an alternative to Swing. SWT applications run on Windows, Mac OS X, and GNU/Linux. The main difference between Swing and SWT is that SWT is a native toolkit, meaning that its widgets are wrappers around native APIs (using JNI) whenever possible.

SWT's notable user and maintainer is the Eclipse project, where it is used to create the GUI for the Eclipse IDE.

10.8 Native Toolkits

Native toolkits are libraries which are usually designed for one platform only. Use these when cross-platform portability is not a concern. We recommend considering Windows Presentation Foundation on Windows and Cocoa on Mac OS X. On GNU/Linux, depending on the desktop environment used, Qt (for KDE) and GTK+ (most others) *are the native toolkits*.

CHAPTER

ELEVEN

MISCELLANEOUS

This section contains tips that are useful but are not at home in the other sections.

11.1 Kill the X Server

If another user has a session on a computer you would like to use, it is possible to log them out and return to the login screen. The key combination `Control-Alt-Backspace` kills the X server, which ends the current user's desktop session. This is also useful should your session crash or freeze.

> **Warning:** Be advised that any of the current user's unsaved data will be lost.

11.2 Directory Navigation

These tools help you navigate directories quickly.

11.2.1 pushd

The command `pushd` *`dir`* will save your current directory on the stack and move you to `dir`.

11.2.2 popd

After you have used `pushd` and stored one or more directories on the stack, you can use `popd` to move back to your previous directory.

11.2.3 dirs

`dirs` allows you to see what directories you have stored on the stack. The far left directory is the most recently saved. Another way to view the directory from most recent to least recent is by using the commmand `dirs -v`.

11.3 xdg-open

xdg-open is a command that can be used to open a file or URL in the user's preferred application. Example:`xdg-open myfile.pdf`

11.4 uptime

The command `uptime` will display the following information in one line: the current time, the amount of time the system has been running, the number of users currently logged in, and system load averages for the past 1, 5, and 15 minutes.

11.5 User Interactions

11.5.1 w

The `w` command provides a summary of the current users on the machine. The header displays the same information as the *uptime* command.

11.5.2 last

`last` displays all the users that have logged into the machine since the creation of the wtmp log file. The date and time that file was created is shown at the bottom of the output. To limit the output to the last 10 logins, use `last -10`.

11.5.3 users

`users` displays a list of all the users currently logged in to a host.

11.5.4 write

The `write` command allows communication with other logged-in users through the terminal. It copies typed lines from your terminal to theirs, and if they wish to reply they must also use the `write` command. The usual syntax is `write` *username*. Any subsequent lines entered will be copied to the user's terminal. To terminate `write` use `Control-D`.

11.5.5 mesg

To control whether someone can *write* to your terminal use the `mesg` command. By default `write` is allowed, to turn it off use `mesg n`.

CHAPTER

TWELVE

CONTRIBUTING TO MASTERING EOS

This manual is a living document. It is maintained by its authors, but both students and faculty (you!) are encouraged to contribute to the guide. The manual is written by GVSU CIS students and faculty for GVSU CIS students and faculty.

The easiest way to to contribute is by reporting an issue or requesting a section with our issue tracker. One of the authors should respond to your issue and give feedback.

The Mastering EOS manual is written using Sphinx, an excellent multi-target documentation generator. The documentation is written in the markup language reStructuredText. The poster is written in LaTeX using Beamer and beamerposter, although contribution to the poster is not necessarily as useful for obvious reasons. Our build system is Waf, and all of our tooling is written in Python. Although these tools may seem overwhelming, intimate familiarity with them is fortunately not necessary for contribution.

The source code can be found at the Mastering EOS GitHub repository. GitHub is a popular site for hosting code repositories using Git, a popular version control system. If you are familiar with the GitHub contribution process, contributing to Mastering EOS is exactly the same.

Mastering EOS is intended to be developed on the EOS Lab computers! This means that you don't have to install Git or do any complicated setup of Python or LaTeX environments — just use your EOS account! The contribution guide is frequently used and tested on EOS, but please let us know of any problems with our issue tracker.

For those unfamiliar with Git and GitHub, contributing to Mastering EOS may seem rather involved. However, by following the steps outlined, it should be rather straightforward. We request that you first give it a try on your own, but failing that, please don't hesitate to contact the authors. We want your contributions!

Continue reading to find out how to make your own contributions to Mastering EOS.

12.1 Repository Setup

To obtain the source code for Mastering EOS (which you will be editing), you must go through the Git version control system. Fortunately, the GitHub contribution process is excellently documented. Follow these steps:

1. Read GitHub's guide to Contributing to a Project.
2. Set up Git if you have not already done so. You can skip the step of downloading and installing Git — it is already installed on the EOS computers.
3. Fork and create a local clone of the Mastering EOS repository.

You have now downloaded the source code to Mastering EOS! Continue on to find out how to build the documents and make changes.

12.2 Pre-requisites

Before running any commands, make sure you change to the repository root directory. For example:

```
cd ~/mastering-eos
```

All following commands will be run from this directory unless otherwise noted.

12.2.1 Install Requirements

Now that you have the source code, the first step in building the manual is to install the Python requirements. For this, we need to install Pip, the Python package manager, to our user site:

```
wget https://bootstrap.pypa.io/get-pip.py -O - | python - --user
```

For the technical details on what this is doing, please see the Pip installation docs.

Pip should now be installed to `~/.local/bin/pip`. However, we don't want to have to type the full path every time we want to run `pip` or any other executable installed there.. To remedy this, add the following to your `~/.bash_profile`:

```
export PATH=~/.local/bin:$PATH
```

Restart your shell to effect the changes to your `~/.bash_profile` *by logging out and logging back in.* Run the following to ensure that this worked:

```
$ pip --version
pip 1.5.6 from /home/smithj/.local/lib/python2.7/site-packages (python 2.7)
```

After installing Pip, use Pip to install this project's requirements to your user site:

```
pip install --user -r requirements.txt
```

The Python requirements for the project have now been met.

12.2.2 Configure Your Editor

EditorConfig

We use EditorConfig to maintain consistent formatting between developers. Our EditorConfig preferences are recorded in the `.editorconfig` file in the root of the repository.

If you do not have an editor preference, we suggest that you use gedit for the following reasons:

- It is simple and lightweight.
- It has EditorConfig support.
- It is already installed on EOS.

An editor is a very personal choice, so if you do have an editor preference, please use the editor with which you are comfortable. There are EditorConfig plugins for many different editors. If yours is listed, we suggest you take the time to get your editor's plugin working. If you cannot get it working or there is no EditorConfig support for your editor, please read the `.editorconfig` file for the formatting standards — they should be relatively easy to follow manually.

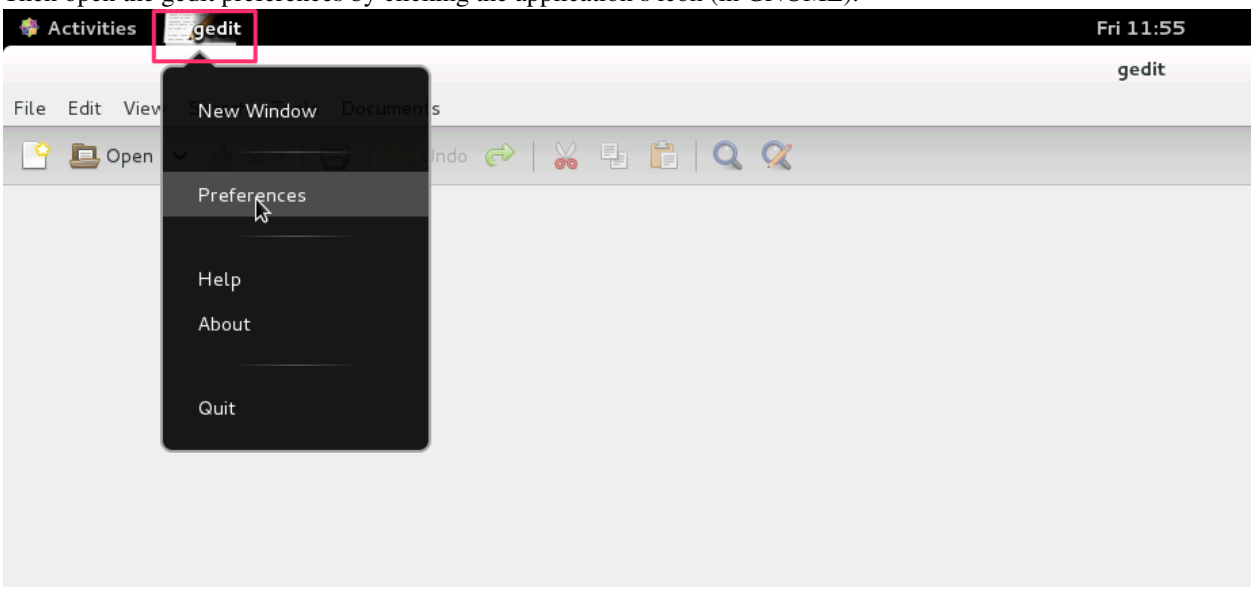
If you do choose gedit, we have a script in our repository to assist in installing EditorConfig support on EOS. To install the plugin, run:

```
scripts/install-gedit-editorconfig-eos
```

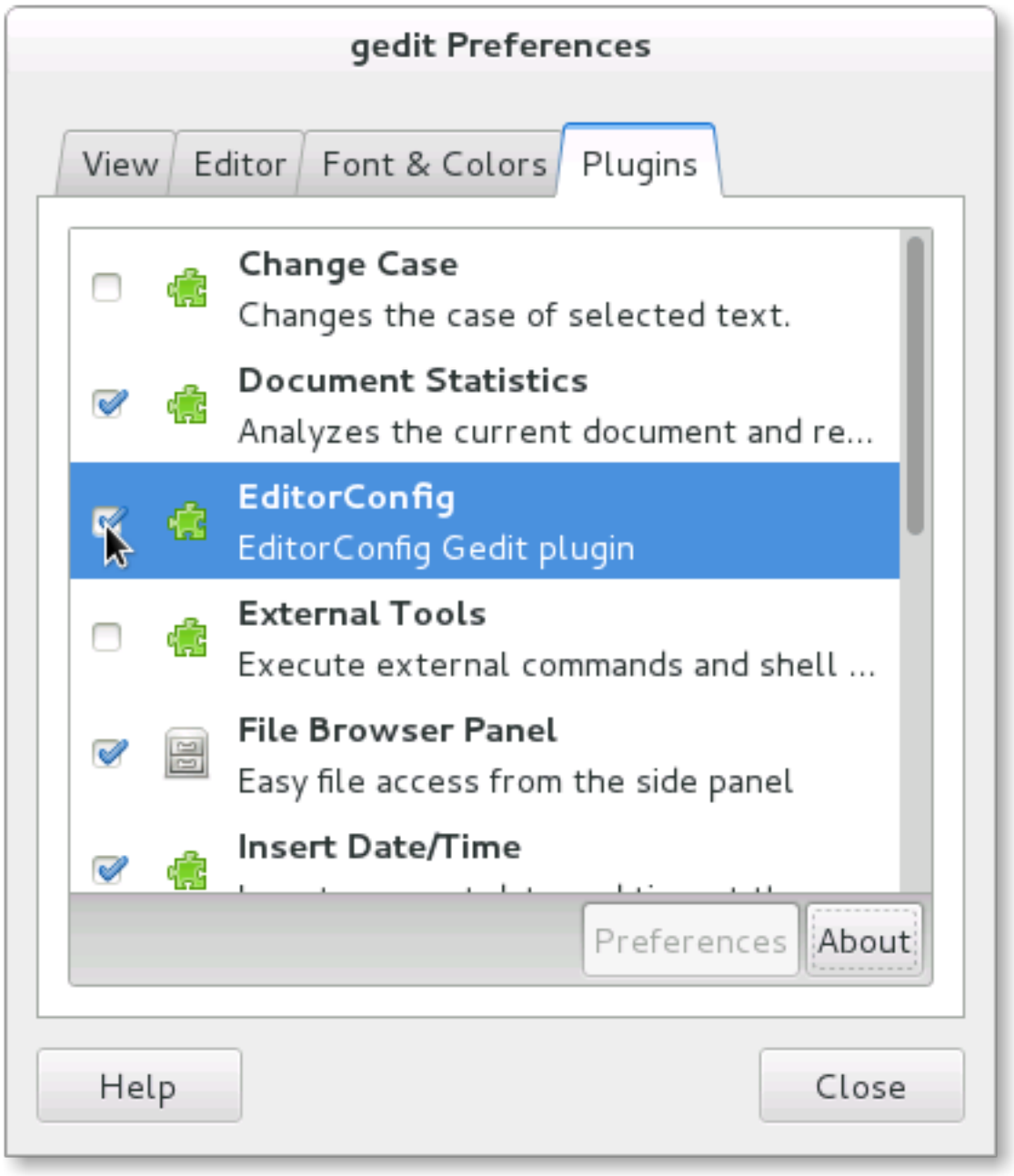
This installs the gedit EditorConfig plugin, but we still need to enable it. Open gedit with:

```
gedit
```

Then open the gedit preferences by clicking the application's icon (in GNOME):



Select the *Plugins* tab, then scroll down and check the *EditorConfig* plugin to enable:



The checkbox should become checked. If it turns to a red warning sign, please report an issue. EditorConfig is now enabled for gedit!

Git Configuration

We want to set up gedit as the editor for Git commit messages. Do this with:

```
git config --global core.editor 'gedit --wait'
```

We use the `--wait` flag here because Git expects the editor to block until the commit message has been finished.

gedit also creates backup files of each file that you save. These files end with a tilde (~) and get annoying when they clutter the output of `git status`. Fortunately, we can tell Git to ignore them. Run the following:

```
$ cat > ~/.gitignore-global <<EOF
# gedit backup files
*~
EOF
$ git config --global core.excludesfile '~/.gitignore-global'
```

Your editor has now been set up for developing Mastering EOS!

12.2.3 SSH Setup

Part of building the documentation is building the table of SSH fingerprints containing a fingerprint for each EOS machine. SSH is used to generate this table. Follow the directions in the following sections to correctly set up SSH to allow this.

Shared and Persistent SSH Connections (optional)

In theory, you should never have to fully rebuild the manual. However, in practice, sometimes a full rebuild is necessary. With a full rebuild, you will have to wait while the SSH fingerprints table is rebuilt. Since this can take a long time, we recommend that you set up *Shared Connections* and *Persistent Connections* as shown in *Advanced OpenSSH*. These are known to dramatically decrease the build time if you have done a full rebuild within the time given to `ControlPersist`.

Inter-EOS SSH Trust

To be able to rebuild the fingerprints table without user intervention, please follow the steps in *Trust All EOS Machines* before continuing.

12.3 Using the Build System

12.3.1 Configure the Build

For building the documents, we use Waf as a build system. Waf is similar to build systems like Autotools and CMake in that it has separate configuration and build stages. To configure the project, run:

```
./waf configure --dev-mode
```

Important: Waf may warn about a missing `makeglossaries` tool. This is safe to ignore.

Waf will check that all the requirements are met before preparing the build. If any besides `makeglossaries` are not met, please re-check your steps in *Pre-requisites* or report an issue. The `--dev-mode` disables auto-update of operations that require SSH. Since the deployment of Mastering EOS is automated, there is no reason for a contributor to leave auto-update on. A shortcut for this command is:

```
./waf configure -d
```

12.3.2 Build the Documentation

The entire project can be built with:

```
./waf build
```

You will see various build steps being executed as they are printed to the screen. This command builds both the manual and the poster and places the build artifacts in the `build/manual` and `build/poster` directories, respectively.

If you are only interested in one output, you can restrict the build by running *one* of the following:

```
./waf html
./waf pdf
./waf man
./waf info
./waf poster
```

12.3.3 View the Results

The build will create outputs in various formats. Run *one* of the following commands to build the specific format, then open it in the operating system's preferred viewer:

```
./waf ohtml
./waf opdf
./waf oman
./waf oinfo
./waf oposter
```

Hint: It is not necessary to run `./waf build` before running any of these commands — for example, `./waf ohtml` will build the HTML docs, then open them in a browser. However, it will not update the poster, PDF, etc.

Although we'd like all the outputs to look good, we primarily focus on the HTML output.

12.3.4 Other commands

In certain circumstances, it is useful to do a full rebuild. The command:

```
./waf clean
```

undoes the actions of `./waf build`, meaning that the build artifacts are removed (but the build directory still exists). The command:

```
./waf distclean
```

undoes the actions of both `./waf configure` and `./waf build`, meaning that the build directory is removed and the project must be re-configured to be rebuilt again.

If you would like to see (almost) exactly what a user visiting the Mastering EOS website sees, run:

```
./waf archive
```

This command creates the directory `build/website`, which contains all the files that will be uploaded to the official website. You can then view the website in this directory with:

```
xdg-open build/website/index.html
```

It is also possible to run multiple commands at once, for example:

```
./waf distclean configure -d build
```

This runs a full rebuild all in one command.

Another helpful task is:

```
./waf linkcheck
```

This checks the Sphinx manual for broken links, redirects, and missing anchors. It's not enabled by default because there will be many false positives, including localhost links (obviously valid only in a certain context) and some URLs which are actually fine. Nevertheless, it's a useful tool for detecting broken and outdated links. Use your best judgment. You can find the full output in `build/manual/linkcheck/output.txt`.

12.4 Editing the Manual

12.4.1 Sync Your Fork

Unless you have just forked the repository, always begin every editing session by syncing your fork.

12.4.2 Make Your Changes

Let's start by making our actual changes to the document. As mentioned before, the manual is written using Sphinx in the markup language reStructuredText. Unless your changes are very minor, at least a surface understanding of reStructuredText is necessary. We recommend fully reading Sphinx's reStructuredText Primer to start. Because Mastering EOS makes heavy use of Sphinx and reStructuredText features, you can also look around at existing source documents and their output to get an idea of how documents are written in Sphinx.

Consider the following sentence:

> I like spinach.

Let's change this very sentence. We'll use this as an example throughout this section. Although we're using this example, feel free to adapt the example to your actual changes.

Everyone loves bacon, so we'll change it to that instead.

Fire up your editor to open the file:

```
gedit manual/contributing/edit.rst
```

Find the sentence and make the change.

As shown in the previous section, you will want to *Build the Documentation* and *View the Results* after you make your change. Once you are happy with how it looks, it is time to create a branch on which to commit your change.

12.4.3 Create a Feature Branch

The next step is to create a *feature branch* for your change. A feature branch represents the addition of one cohesive set of changes. For example, a feature branch could contain related changes to sections on SSH and SCP, but should not contain unrelated changes to the personal website and VNC sections.

Try to pick a name for your branch that represents your change. Create a new feature branch for our example with:

```
git checkout -b bacon-rules-spinach-drools
```

Hint: In this example, we only make one commit on the feature branch. Although one commit is fine, you are free to make more commits on a feature branch as well.

12.4.4 Commit Your Change

Once you have made your change, it's time to commit. First, let's ask Git for a status report:

```
$ git status
On branch bacon-rules-spinach-drools
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   manual/contributing/edit.rst
```

As you can see, Git is aware of our change, but we have not yet told Git that we intended to make them.

Assure Git of our intentions with:

```
git add manual/contributing/edit.rst
```

Looking at the status now yields:

```
$ git status
On branch bacon-rules-spinach-drools
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   manual/contributing/edit.rst
```

We are now ready to commit our change. As part of the commit, you will be asked for a commit message. The commit message should be a short (less than 50 characters), high-level summary of what has been done in this commit. Before writing a message for this commit, look back at commit messages for prior commits with:

```
git log
```

These messages should give you an idea of what a typical commit message for this project looks like. Press `q` to quit the log viewer. To commit your change, run the following:

```
git commit
```

This should open gedit, or another editor if you have configured one. Enter your commit message:

> I like bacon, not spinach. Geez; get it right.

12.4.5 Push The Branch

Your changes have now been committed. The last step in this section is to push them to your fork. Do so with the following:

```
git push -u origin bacon-rules-spinach-drools
```

Your branch has now been pushed to your forked repository! Continue on to the next section to find out how to propose them as changes to the Mastering EOS official repository.

12.4.6 Git Resources

This guide illustrates the bare minimum amount of Git commands that you will need to complete this task. For more guidance on using Git, please check out GitHub's list of Git resources. In particular, GitHub's Try Git is great for beginners.

12.5 Making a Pull Request

Warning: If you were following along with the bacon example, don't actually create a pull request.

Once your branch has been pushed to your fork, follow the GitHub instructions on making a pull request.

After you've made the request, you're almost done! An author should respond to your pull request soon. They may respond with approval, at which time they may merge your pull request. They may also respond with some discussion, at which time you may want to go back and revise your changes.

With any luck, once resolved, your changes should go live! We look forward to seeing your contributions, and please let us know of any problems through our issue tracker!

12.6 Writing Guidelines

12.6.1 Writing Style

When writing technical documentation, it is important to follow a consistent writing style. Within this manual, we attempt to follow the OpenStack writing conventions. Their documentation presents a great summary of how to write coherent technical documentation.

Sphinx uses SmartyPants to transform quotes, dashes, and ellipses into typographically correct entities for HTML output. You can use straight quotes, straight apostrophes, and three dots — they will be transformed into the correct characters. For dashes, first read up on the three types of dash. The transformations are as follows: Hyphens stay as is, two hyphens will be transformed into an en dash, and three hyphens will be transformed into an em dash.

Please also see the following links for writing technical documentation:

- ACS Distance Education Guidelines for Technical Writing
- Novell Open Source Documentation Style Quick Start
- BlueBream Documentation Guidelines
- Description of Imperative Mood on Wikipedia

12.6.2 reStructuredText Conventions

Because there are many ways to write reStructuredText, we have set some conventions to be followed when editing the documentation.

Section Headers

Although docutils avoids "imposing a fixed number and order of section title adornment styles", adhering to conventions produces results of higher quality and greater consistency. Please use *the exact order* used in the description of reStructuredText Sections when writing documentation. See the source code of this page for an example.

When using the first two header styles, pad the title with a space on both sides as shown in reStructuredText Sections (typing a space is not necessary on the right; just make it look correct).

Indentation

Indentation in reStructuredText is both simple and complex. Instead of dictating that the indent must be a certain number of spaces, or use tabs, indentation is always relative to the prior input. Unfortunately, this tends to make things confusing, so we've set some conventions.

Default indentation is 3 spaces, and should be used for almost everything including literal blocks, directives, and comments:

```
This is some Python::

   print('This is Python')
   if True:
       print('so we still indent 4 spaces *within* the code')

.. toctree::
   :maxdepth: 2

   one
   two
   three

..

   Here is a comment.
   And a second line.
```

The 3-space indent is used because it looks better than 2 or 4 when using the directive syntax.

Lists differ from the 3-space indent. Each item should include one space after its delimiting character, and the contents of the item should be lined up with that, as shown in the primer example. Here is another example:

```
- First item
- Some Bash:

  .. code-block::

     ls -l

- Here we have
  two lines.

#. This list
#. has a different
   indent
```

Correct Usage of Roles

- When referring to a command or program, use our project-specific `:cmd:` role:

  ```
  The :cmd:`ls` command lists files in a directory.
  ```

  However, when referring inline to a command line to be run, use the inline literal syntax:

  ```
  The ``ls -l`` command lists files in a directory in a long format.
  ```

  Never use the command role or program role.

- Use the envvar role when appropriate, and create a corresponding entry in *Environment Variables*.

- Do not use the option role, as it emits a warning if a corresponding option directive is not found. This would be fine, but we want to remain warning-free. Instead, just use normal literal text.

Backticks and Hyperlinks

When creating hyperlinks in rST, do not use backticks when they are not necessary:

```
.. Good
abcd_
ab-cd_
ab.cd_
`ab cd`_
`ab+cd`_

.. Bad
`abcd`_
`ab-cd`_
`ab.cd`_
```

Linking to Sections

Although regular reST supports implicit hyperlink targets to section titles, etc., we recommend using the Sphinx-specific ref role for cross-referencing. See the last sentence of the ref documentation if you are curious as to why.

12.7 Reviewing Pull Requests

Note: This section assumes general familiarity with pull requests. If you are not familiar with pull requests, please read up on them first.

Reviewing pull requests is a common part of developing and contributing to Mastering EOS. The diff view is handy when reviewing changes made to content, but is not particularly helpful when reviewing entirely new sections. In this case, it is often more convenient to look at the content in built form.

Luckily, this is not difficult. Since we are not planning to edit the code, only build and review it, we can check out the remote branch directly. First, find out the name of the branch by looking at the pull request. We'll go with the bacon example here:

```
git fetch
git checkout origin/bacon-rules-spinach-drools
```

Git will now alert you that you are in detached head state. Don't worry, that's OK. Now, build and open the docs as usual, for example:

```
./waf ohtml
```

You can now view the documentation with the changes mentioned in the pull request. If the pull request is updated, you can see the new changes with:

```
git fetch
git checkout origin/bacon-rules-spinach-drools
```

Notice that these are the same commands as before.

To return to a named branch, just check it out:

```
git checkout master
```

12.8 Maintenance Tasks

The Mastering EOS project is written and organized to minimize the amount maintenance required to keep it up-to-date. However, like any other project, some maintenance is required. This page attempts to document many of ways in which Mastering EOS needs to be maintained.

12.8.1 Automated

The following tasks are automated, and should not have to be closely maintained:

- The number and list of EOS hosts. All `eos`, `arch`, and `dc` hosts are read from the `/etc/hosts` file on `eos01`.
- SSH fingerprints. Fingerprints are read from each EOS host detected in the last step.
- The VNC port/geometry mappings. These are read from the `/etc/xinetd.d/vncts` file on `eos01`.

The automation implies external maintenance of said files. Coordinate with Ira Woodring to manage changes. In addition, the automated tasks themselves may need to be updated if for some reason they break.

12.8.2 Documentation

Although everything in the manual having to do with EOS needs to be kept up-to-date, we've identified some pieces that are likely to become out-of-date. These should be checked periodically.

- The quota in the *Disk Space* section. We have considered automating this.
- Update versions of the example programs (Bash and EditorConfig) in *Manual Installation*.
- Ensure the software in *Manual Installation* still compiles without issue.
- Ensure that *Linuxbrew* is still working as advertised.
- Ensure that *HardInfo* still compiles without issue.
- If the system administrator changes, update the name and email address in the Sphinx configuration file.
- Ensure that we still run each of the *Databases* and that the login mechanisms still work.
- Ensure that the URL for *Oracle APEX* is still accurate.
- Ensure that all advertised *Editors* are still available on EOS. Those that aren't should be placed in their respective section's *Other* subsection.
- Ensure that *Bash* 4 is still the default shell on EOS. At this time of writing, a user's default shell can't be changed. If it can in the future, make a note about this in *Interactive Shell Use*.
- Ensure that *sh* still invokes Bash. That would be embarassing if it isn't and we use it as an example.
- Ensure that all the steps outlined in *Contributing to Mastering EOS* still work.
- Ensure that the *list of maintainers* stays up-to-date.

Project Infrastructure

Producing great documentation is the main aim of Mastering EOS. However, certain project infrastructure is needed to maintain documentation of this complexity. Maintenance of this infrastructure is key to the project's continued success. Here are some areas of the project that need to be kept up-to-date.

Build System

Our build system is Waf. The Waf build system itself is bundled with the repository in the `waf` file in the project root. We try to keep our version of Waf at the newest version, and it should be checked for updates periodically. Sometimes the updates break part of the build system, which then requires fixing. See the Waf changelog for a list of changes.

Python Requirements

Our Python package requirements file, `requirements.txt`, contains a list of Python packages upon which our build depends. All of these requirements need to be periodically checked for updates. Application of the updates may break our build, which then of course needs to be fixed.

Python 3

We would like to transition to Python 3 in the near future. The only thing currently preventing us from doing so is Fabric. Fabric is an integral part of our build and to remove or replace it would be a significant disadvantage. Fabric 2.x is already in the works, and it appears that Python 3 compatibility will be added with this release.

When the project runs successfully under Python 3, use of the six module may be removed.

CHAPTER

THIRTEEN

ENVIRONMENT VARIABLES

This page contains a list of all environment variables referred to in the manual. Another list is the Guide to Unix's Environment Variables section.

CPPFLAGS
: Flags given to the C pre-processor during compilation. See the Autoconf manual on Preset Output Variables.

EDITOR
: The command for the editor to run when a file needs editing. Programs which use this variable include crontab, *Linuxbrew*, and various version control systems. There is no standard for how it is interpreted. Depending on the program reading it, it could be interpreted as a command-line interpreted by the shell (likely passed to popen) or a single name or path to an editor (likely passed to exec family). For maxiumum portability, set the value to a single name or path without spaces so that it can be utilized in both ways. See *this example* for how it may be used with crontab.

INFOPATH
: The search path for manual pages readable by `info`. See *Man and Info Paths* for instructions on how to set it.

LD_LIBRARY_PATH
: Additional paths in which the dynamic linker should search for shared libraries. See Russ Allbery's notes on Shared Library Search Paths, the Autoconf manual on Preset Output Variables, and the Wikipedia entry on rpath.

LDFLAGS
: Flags given to the linker during compilation. See the Autoconf manual on Preset Output Variables.

MANPATH
: The search path for manual pages readable by `man`. See *Man and Info Paths* for instructions on how to set it.

PATH
: The GNU/Linux search path for executable files. See *Path Manipulation* for how to manipulate it and *Executable Path* for how to set it correctly for user-level hierarchies.

PWD
: The current working directory. This variable is usually set and exported by the shell.

Path
: The Windows search path for executable files.

CHAPTER

FOURTEEN

ABOUT THIS DOCUMENTATION

14.1 People

The current maintainers of Mastering EOS are:

- **Sean Fisk**; GVSU '14 Master of Computer Information Systems, GVSU '12 Bachelor of Computer Science
- **Ira Woodring**; GVSU CIS EOS Labs System Administrator
- **April Lautenbach**; GVSU Computer Science undergraduate
- **Mick Lautenbach**; GVSU Computer Science undergraduate

Mastering EOS was originally created by Sean Fisk and Ira Woodring.

14.2 Licensing

The license for the content of Mastering EOS is as follows:

- All text is licensed under the Creative Commons BY-NC-SA 4.0 License. We chose this license because we want others to be able to share this information with whomever they want. However, we would not like to see our work adapted into a book and sold without our approval. In general, you can use our text if you would like to share it with others within a non-commercial venture.
- All code is licensed under the Apache 2.0 License unless otherwise stated. You are welcome to use and adapt our code for almost any purpose, commercial or otherwise. We chose this license because we'd like to see this code and these examples reach as many people as possible. Many of the code snippets were based on examples from man pages or articles on the 'Net, and we'd like them to be available to just as wide an audience.

Although we have expressed our intentions on use of our material here, it is the text of each respective license which is legally binding. Please keep this in mind when using or adapting our material.

In general, please be respectful and intelligent. If you would like to use our material for a purpose not covered by these licenses, feel free to contact us.

14.3 Special Thanks

Written by Sean Fisk

Thanks to GVSU students:

- James Donley for getting me into programming, user-level installation, and far more than I could possibly remember or list here.

- Dave Medawar for teaching me about SSH, Bash, VNC, remote printing, the VPN, web servers, and much more.
- John Roys and James Vanderzouwem for advice and encouragement.
- Paul Wand for learning to program with me.
- Joseph Roth for teaching me how to use Git.
- Sam DeHaan for working for many hours on various projects with me in the Arch lab.
- Kurt O'Hearn for showing me LaTeX and helping me with system administration.
- Taylor Countryman, Chelsea Champion, and Robb Nemecek for listening to endless amounts of technobabble from me.
- Eric Vanklompenberg for helping me to learn more about PowerShell and the Windows console.

Thanks to GVSU faculty:

- Carl Strebel for teaching me about system administration, Bash, VNC, and xinetd.
- Dr. Zach Kurmas for teaching me about SSH keys and version control, and for other advice.
- Dr. Hans Dulimarta for teaching me about C++ and Qt, and for other advice.
- Dr. Christian Trefftz for advising and encouragement.
- Dr. Greg Wolffe for advising me throughout my last 3 years of college.
- Ira Woodring for supporting me in countless ventures including Mastering EOS.
- Dr. Jonathan Engelsma for advice and GitHub help.

Also thanks to my NCAR advisor Davide Del Vento for inviting me to SEA 2014 to speak about many of the topics in Mastering EOS.