Unix Survival Guide

Lincoln D. Stein[1,2]

[1]Ontario Institute for Cancer Research, Toronto, Ontario, Canada
[2]Department of Molecular Genetics, University of Toronto, Ontario, Canada

Most bioinformatics software has been designed to run on Linux and other Unix-like systems. Unix is different from most desktop operating systems because it makes extensive use of a text-only command-line interface. It can be a challenge to become familiar with the command line, but once a person becomes used to it, there are significant rewards, such as the ability to string a commonly used series of commands together with a script. This appendix will get you started with the command line and other Unix essentials. © 2015 by John Wiley & Sons, Inc.

Keywords: Linux • Unix • shell script • command line

How to cite this article:
Stein, L.D. 2015. Unix survival guide. *Curr. Protoc. Bioinform.* 51:A1.C.1-A1.C.27.
doi: 10.1002/0471250953.bia01cs51

INTRODUCTION

For a mixture of historical and practical reasons, much of the bioinformatics software discussed in this series runs on Linux, Mac OS X, or one of several other Unix variants. This appendix provides the minimum information needed to survive in a Unix environment.

Currently, the two dominant Unix-based operating systems that you are likely to encounter are Mac OS X, which typically runs on desktop and laptop machines, and Linux, which typically powers high-end servers and compute clusters. Many of the tools described in this series will run well on both OS X and Linux, but you will inevitably encounter packages that are Linux-only. Very few tools will run acceptably or at all on Windows systems. If you are intending to work with command-line oriented bioinformatics tools, your life will be much simpler if you become comfortable working in the Linux environment. You will be able to perform small-scale testing and integration of tools on your personal system by running Linux on a virtual machine, and then make the transition to a high-powered server when you are ready to scale up to large data sets.

The first part of this guide walks you through the basics of interacting with Unix, with particular emphasis on working in the command-line environment of a Linux system. It assumes that you already have access to a Linux system. Since this may not be the case, the second part of the guide shows how to install a fully-functioning Linux system on your desktop or laptop machine, or alternatively in the Amazon compute cloud.

LOGGING INTO A UNIX SYSTEM

Unix dates from the time when computers were very expensive, necessitating that multiple users share the same computer hardware. For this reason, a session on a Unix system begins with a login prompt. You provide the system with a username and password in order to gain access to the system's resources. If your system is managed by a system administrator from your institution's Information Technology (IT) department, the username and password will have been assigned to you. If you have installed Unix yourself, for example in a VirtualBox Linux image, you will have been prompted

Current Protocols in Bioinformatics A1.C.1-A1.C.27, September 2015
Published online September 2015 in Wiley Online Library (wileyonlinelibrary.com).
doi: 10.1002/0471250953.bia01cs51

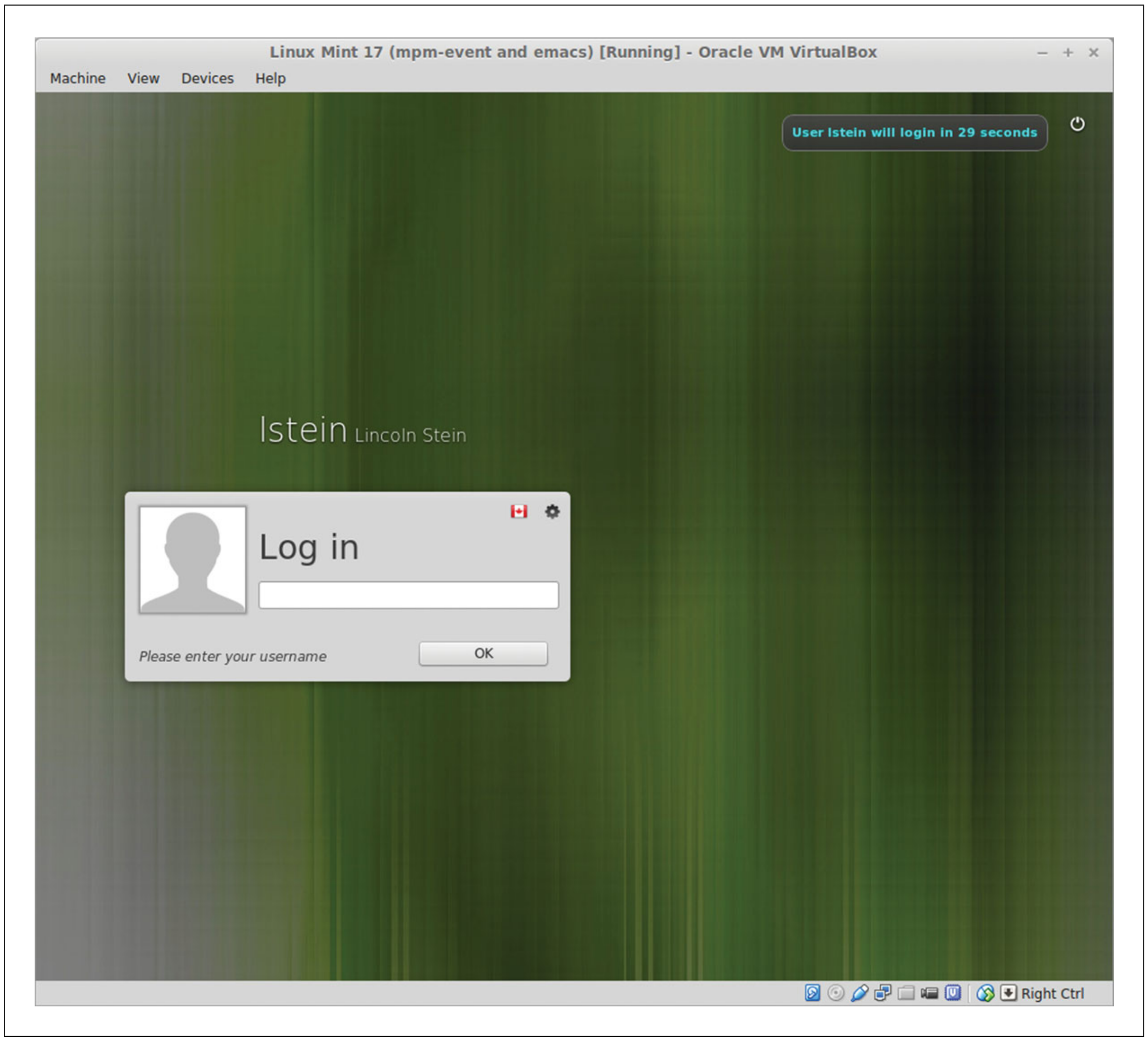

Linux Mint 17 (mpm-event and emacs) [Running] - Oracle VM VirtualBox
Machine
View
Devices
Help
User lstein will login in 29 seconds
lstein Lincoln Stein
Log in
Please enter your username
OK
Right Ctrl

Figure A1.C.1 The default login screen on Linux Mint prompts you for a username and password.

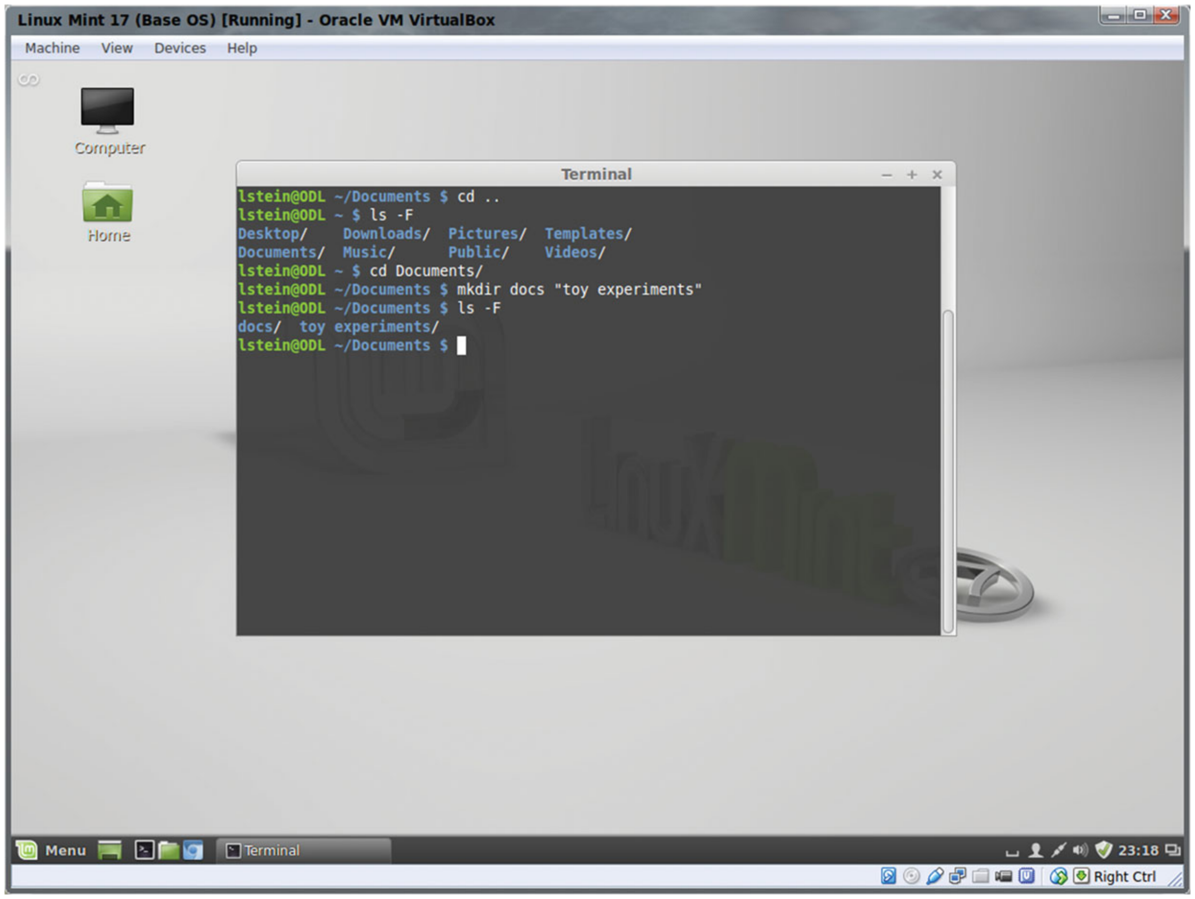
for a username-password pair at the time of installation. If you are using an Amazon cloud instance, the username is hard-coded into the template from which the instance was launched: for example, when using the stock Ubuntu instances, the username is `Ubuntu`.

There are two common login scenarios. In the first, you are sitting in front of the Unix computer itself and are using its monitor and keyboard directly or via virtual machine emulation (a situation sentimentally called "logged in at the console"). In the second, you use a Windows or Macintosh desktop machine to connect via the network to a Unix server located at some remote location.

Logging in at the Console

In the first scenario, you will be presented with a login window. A typical login window seen in Linux Mint running under VirtualBox is shown in Figure A1.C.1. Because of the great variability in Unix distributions, yours may look different; however, all login windows prompt for a username and password. Type yours in and press the appropriate button ("OK" in the example shown in Fig. A1.C.1).

If the username and password are recognized, the system will log you in and display a graphical desktop (Fig. A1.C.2). The desktop shown in that figure is the one used by


Linux Mint 17 (Base OS) [Running] - Oracle VM VirtualBox
Machine View Devices Help
Computer
Home
Terminal
Menu
Terminal
23:18
Right Ctrl

```
lstein@ODL ~/Documents $ cd ..
lstein@ODL ~ $ ls -F
Desktop/    Downloads/  Pictures/  Templates/
Documents/  Music/      Public/    Videos/
lstein@ODL ~ $ cd Documents/
lstein@ODL ~/Documents $ mkdir docs "toy experiments"
lstein@ODL ~/Documents $ ls -F
docs/  toy experiments/
lstein@ODL ~/Documents $ 
```

Figure A1.C.2 The Linux Mint desktop looks superficially like a Microsoft Windows system. The command-line terminal emulator window in the center is a bit of a giveaway.

Linux Mint, and is similar in many respects to the Windows and Mac OS X desktops: use the Menu icon at the bottom left to access applications and utilities. The icons of frequently used applications are shown in the taskbar at the bottom of the screen, and important locations on the system are accessible by icons on the desktop. The left mouse button selects icons and menus, while the right mouse button pops up context-specific menus. (On single-button Macintosh systems, command-click emulates the right mouse button.). In the center of this screenshot is a terminal window running the command-line shell that will be the subject of much of this survival guide.

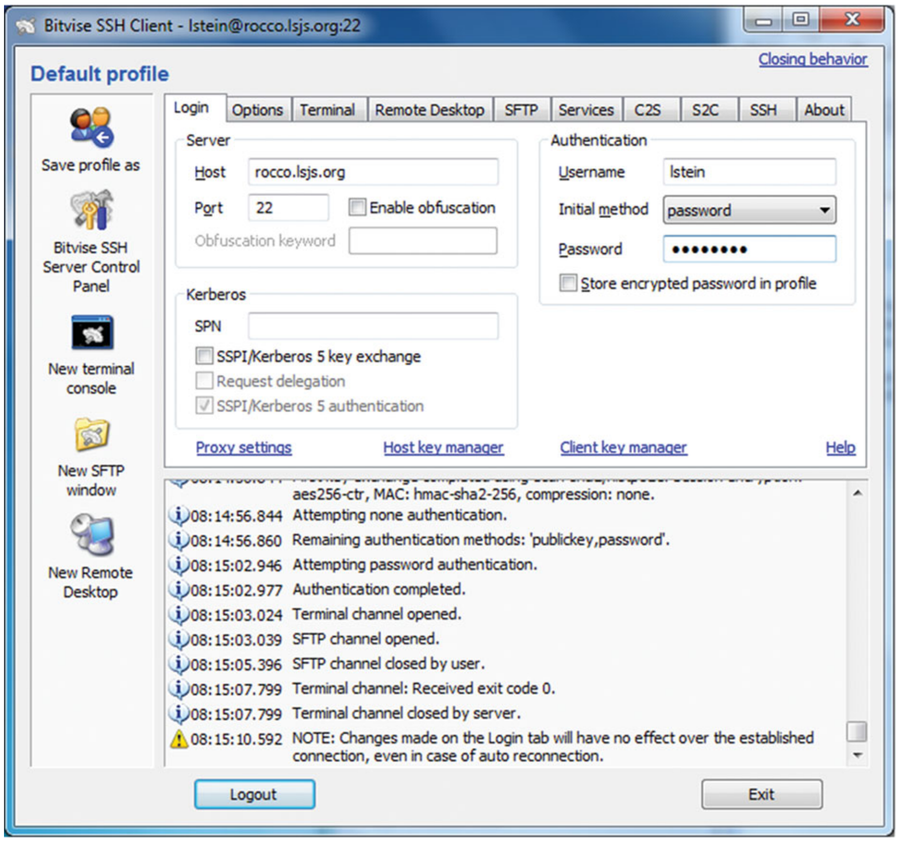
To log out of Linux Mint, click on Menu and select the Logout icon.

Other Linux distributions will use a range of different desktop interfaces, and each can take some getting used to.

Logging in Remotely

If the Unix system you wish to access is located remotely, you will use one of several remote access programs to log into it from your desktop machine. These programs range from bare-bones terminal emulators that provide you with a text-only window to remote desktop viewers that will display the remote Unix graphical desktop on your local machine.

Which terminal emulation program you use depends on what software is installed on the Unix machine and how the network is configured. Common remote login packages are listed below (see Internet Resources).



Server
Host: rocco.lsjs.org
Port: 22 ☐ Enable obfuscation
Obfuscation keyword:

Kerberos
SPN:
☐ SSPI/Kerberos 5 key exchange
☐ Request delegation
☑ SSPI/Kerberos 5 authentication

Authentication
Username: lstein
Initial method: password
Password: ••••••••
☐ Store encrypted password in profile

```
             aes256-ctr, MAC: hmac-sha2-256, compression: none.
08:14:56.844 Attempting none authentication.
08:14:56.860 Remaining authentication methods: 'publickey,password'.
08:15:02.946 Attempting password authentication.
08:15:02.977 Authentication completed.
08:15:03.024 Terminal channel opened.
08:15:03.039 SFTP channel opened.
08:15:05.396 SFTP channel closed by user.
08:15:07.799 Terminal channel: Received exit code 0.
08:15:07.799 Terminal channel closed by server.
08:15:10.592 NOTE: Changes made on the Login tab will have no effect over the established
             connection, even in case of auto reconnection.
```

Figure A1.C.3 The BitVise program for Microsoft Windows enables you to log in to a remote Unix machine via the Secure Shell Protocol.

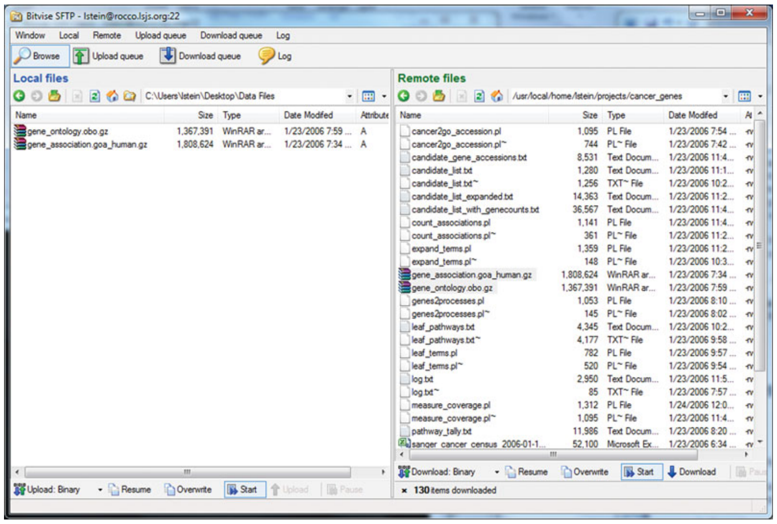
More information on using the graphical remote access protocols based on RDP, VNC, and the X Windows systems are given in *APPENDIX 1D* (Stein, 2007). Here we will assume that you will be logging in using a text-only terminal emulator.

Logging into a remote system with SSH

The Secure Shell (SSH) is the standard for remote logins to Linux and other Unix-based operating systems. SSH encrypts all its communications, preventing hackers from intercepting passwords and other confidential information. SSH also can be set up to use a secure cryptographic key pair system that avoids the use of passwords entirely.

If your local machine runs some version of Microsoft Windows, then you can use either the PuTTY or the BitVise SSH communications program. The former is open source and completely free for use, while the latter is commercial but free for personal use. I personally prefer BitVise because it offers a built-in graphical SFTP file browser and has more intuitive key pair management facilities.

To connect to a remote system using BitVise, you will need the host name or IP address of the remote system, as well as a valid username and password. From BitVise's main screen (Fig. A1.C.3) enter the host's name or IP address as well as your username and password in the appropriately labeled fields. If you do not see a place to enter the password, you may need to select "password" from the pop-up menu "Initial method" in the Authentication section. Press Login and you will be connected to the remote machine's command-line interface. To log in using a key pair rather than a password, follow the procedure to load the key into BitVise described later in this guide (at the end of Running a Linux Server in the Cloud, below).



Local files: C:\Users\lstein\Desktop\Data Files

Name	Size	Type	Date Modifed	Attribute
gene_ontology.obo.gz	1,367,391	WinRAR ar...	1/23/2006 7:59 ...	A
gene_association.goa_human.gz	1,808,624	WinRAR ar...	1/23/2006 7:34 ...	A

Remote files: /usr/local/home/lstein/projects/cancer_genes

Name	Size	Type	Date Modifed	At
cancer2go_accession.pl	1,095	PL File	1/23/2006 7:54 ...	-rv
cancer2go_accession.pl~	744	PL~ File	1/23/2006 7:42 ...	-rv
candidate_gene_accessions.txt	8,531	Text Docum...	1/23/2006 11:4...	-rv
candidate_list.txt	1,280	Text Docum...	1/23/2006 11:1...	-rv
candidate_list.txt~	1,256	TXT~ File	1/23/2006 10:2...	-rv
candidate_list_expanded.txt	14,363	Text Docum...	1/23/2006 11:2...	-rv
candidate_list_with_genecounts.txt	36,567	Text Docum...	1/23/2006 11:4...	-rv
count_associations.pl	1,141	PL File	1/23/2006 11:4...	-rv
count_associations.pl~	361	PL~ File	1/23/2006 11:2...	-rv
expand_terms.pl	1,359	PL File	1/23/2006 11:2...	-rv
expand_terms.pl~	148	PL~ File	1/23/2006 10:3...	-rv
gene_association.goa_human.gz	1,808,624	WinRAR ar...	1/23/2006 7:34 ...	-rv
gene_ontology.obo.gz	1,367,391	WinRAR ar...	1/23/2006 7:59 ...	-rv
genes2processes.pl	1,053	PL File	1/23/2006 8:10 ...	-rv
genes2processes.pl~	145	PL~ File	1/23/2006 8:02 ...	-rv
leaf_pathways.txt	4,345	Text Docum...	1/23/2006 10:2...	-rv
leaf_pathways.txt~	4,177	TXT~ File	1/23/2006 9:58 ...	-rv
leaf_terms.pl	782	PL File	1/23/2006 9:57 ...	-rv
leaf_terms.pl~	520	PL~ File	1/23/2006 9:54 ...	-rv
log.txt	2,950	Text Docum...	1/23/2006 11:5...	-rv
log.txt~	85	TXT~ File	1/23/2006 7:57 ...	-rv
measure_coverage.pl	1,312	PL File	1/24/2006 12:0...	-rv
measure_coverage.pl~	1,095	PL~ File	1/23/2006 11:4...	-rv
pathway_tally.txt	11,986	Text Docum...	1/23/2006 8:20 ...	-rv
sanger_cancer_census_2006-01-1...	52,100	Microsoft Ex...	1/23/2006 6:34 ...	-rv

× 130 items downloaded

Figure A1.C.4 BitVise comes with a graphical file browser that makes it simple to copy files back and forth between your desktop machine and a remote system running Secure Shell.

If your local machine runs Mac OS X, Linux, or another Unix variant, you will use the command-line utility named "ssh." Bring up a terminal (see Using the Command Shell, below), and type the following command:

```
ssh username@remote.host
```

Replace `username` and `remote.host` with the correct user and hostnames for the remote system. The ssh utility will prompt you for your password, connect you to the remote system, and then launch a command-line interpreter on the remote system.

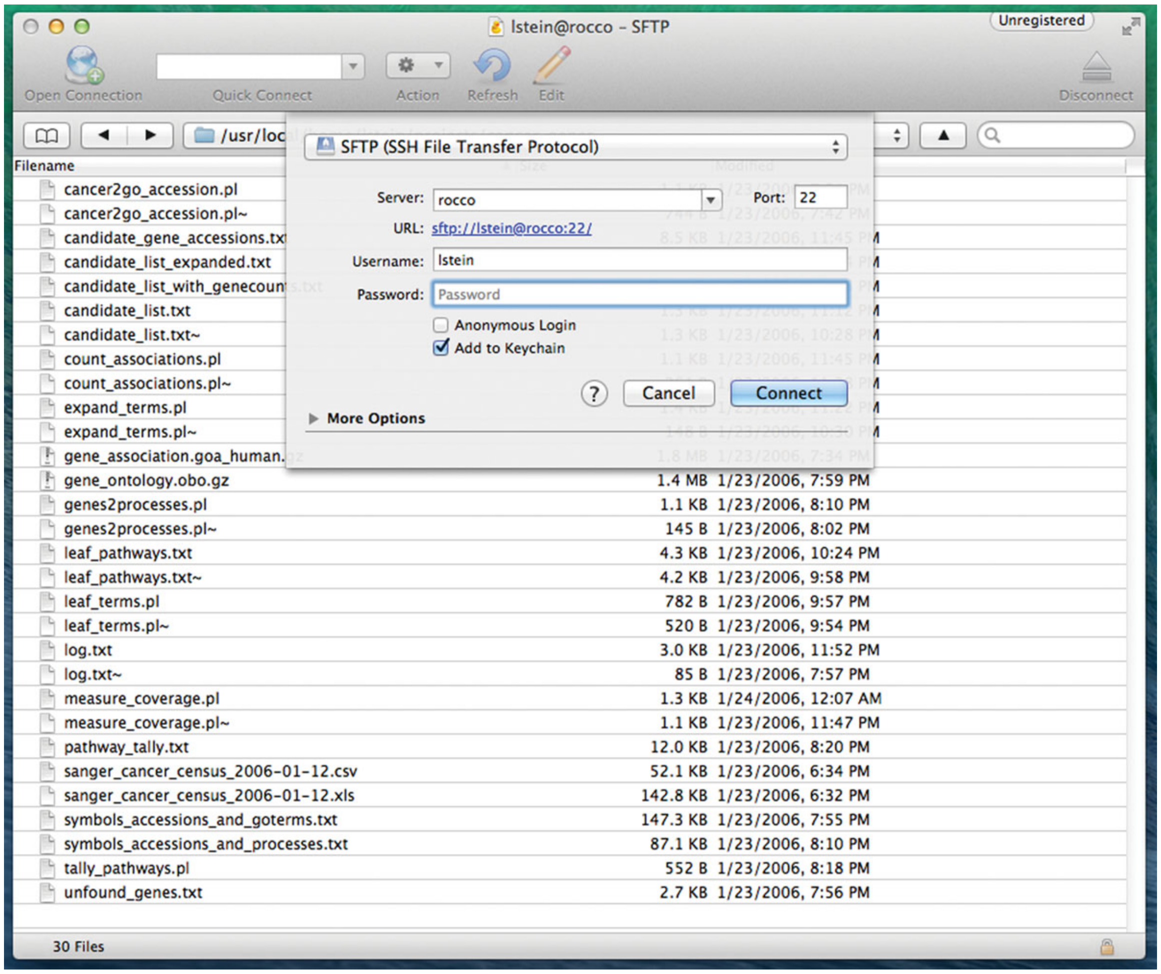
To authenticate to the remote system that uses a cryptographic key pair instead of a username/password, follow the command template described towards the end of Running a Linux Server in the Cloud, below.

Transferring files using SFTP

You can easily transfer individual data files and whole directories between local and remote systems using the Secure Shell's file transfer protocol, SFTP. On Windows hosts, the BitVise system opens a graphical file transfer window automatically whenever you log into a remote machine (Fig. A1.C.4). You can navigate up and down the directory structure of the remote and local machines and transfer files back and forth using the familiar drag-and-drop actions. In the event that you inadvertently close the file transfer window or it does not appear in the first place, click the "New SFTP window" icon in the BitVise main window to reopen it.

On Mac OS X, the free Cyberduck application provides a similar SFTP client (Fig. A1.C.5). Simply select Open Connection and choose SFTP from the pull-down menu. Type in the name or address of the remote system and provide your log-in username and password. You can then transfer files and directories back and forth, as well as perform most directory maintenance operations such as deleting and renaming files.

A graphical SFTP interface is also built into most Linux-based desktops. Simply open the file manager and look for a menu item called "Connect to Server . . . " or similar. The remote host will open up on the desktop, and you can browse directories and transfer files back and forth in the same way you'd manage local files.



lstein@rocco – SFTP

SFTP (SSH File Transfer Protocol)

Server: rocco Port: 22

URL: sftp://lstein@rocco:22/

Username: lstein

Password: Password

Anonymous Login

[✓ Add to Keychain]

```
Filename
cancer2go_accession.pl
cancer2go_accession.pl~
candidate_gene_accessions.txt
candidate_list_expanded.txt
candidate_list_with_genecounts.txt
candidate_list.txt
candidate_list.txt~
count_associations.pl
count_associations.pl~
expand_terms.pl
expand_terms.pl~
gene_association.goa_human.gz
gene_ontology.obo.gz                              1.4 MB  1/23/2006, 7:59 PM
genes2processes.pl                                1.1 KB  1/23/2006, 8:10 PM
genes2processes.pl~                                145 B  1/23/2006, 8:02 PM
leaf_pathways.txt                                 4.3 KB  1/23/2006, 10:24 PM
leaf_pathways.txt~                                4.2 KB  1/23/2006, 9:58 PM
leaf_terms.pl                                      782 B  1/23/2006, 9:57 PM
leaf_terms.pl~                                     520 B  1/23/2006, 9:54 PM
log.txt                                           3.0 KB  1/23/2006, 11:52 PM
log.txt~                                            85 B  1/23/2006, 7:57 PM
measure_coverage.pl                               1.3 KB  1/24/2006, 12:07 AM
measure_coverage.pl~                              1.1 KB  1/23/2006, 11:47 PM
pathway_tally.txt                                12.0 KB  1/23/2006, 8:20 PM
sanger_cancer_census_2006-01-12.csv              52.1 KB  1/23/2006, 6:34 PM
sanger_cancer_census_2006-01-12.xls             142.8 KB  1/23/2006, 6:32 PM
symbols_accessions_and_goterms.txt              147.3 KB  1/23/2006, 7:55 PM
symbols_accessions_and_processes.txt             87.1 KB  1/23/2006, 8:10 PM
tally_pathways.pl                                  552 B  1/23/2006, 8:18 PM
unfound_genes.txt                                 2.7 KB  1/23/2006, 7:56 PM

30 Files
```

Figure A1.C.5 The Cyberduck program, available for both Macintosh OS X and Microsoft Windows, provides a graphical user interface for file transfers using the Secure Shell.

For those who are comfortable with the command line, the “rsync” utility is highly recommended for efficiently keeping directories of files synchronized among multiple machines. Please see the discussion of Rsync in the following section under Managing Files and Directories, below.

USING THE COMMAND SHELL

Many or most bioinformatics packages eschew graphical user interfaces in favor of command-line interfaces. You issue instructions to the system by typing cryptic commands in a terminal window, and the output of the program is displayed as text inside the same window. The Unix program that accepts and processes commands is called the “shell.” It is a simple program that prints out a command-line prompt, waits for you to type a command and press the Enter key, and then runs the command. After the command is complete, the shell again prints the command line prompt, awaiting further instructions. You can issue several commands at the same time by separating commands with a semicolon (`;`) characters.

While typing in commands at a command line prompt can be error prone and frustrating, the big advantage of Unix/Linux and shell commands is that repetitive series of commands can be saved to a file, and then run from that file as a “shell script.” Analyses that are run from a set of commands saved to a file are reproducible, and once you have some experience running commands from a shell script, it becomes possible to add flexibility to the script so the same analysis pipeline can be easily applied to multiple datasets.

If you have logged into a remote system using SSH, you are already running a shell. Otherwise, if you are logged into the console and using a graphical desktop, you will need to launch a terminal emulator within the desktop environment in order to interact with the shell. On Linux Mint, the name of the terminal emulator is Terminal, and it can be found in the Accessories section of the main menu or by searching for “terminal” in the main menu search box. Once you find the application icon, you can create a shortcut for it in the task bar or on the desktop by right-clicking and selecting the desired action from a pull-down menu.

Mac OS X’s command-line shell is similarly named Terminal and is found in the Utilities subfolder of Applications. You may also find it using a Spotlight search. I suggest that you add Terminal to the application dock for convenient access.

On other Unix systems, you will find terminal emulators lurking under a variety of names. Look for applications called Shell, Terminal, Console, Xterm, lxterm, Gnome terminal, or some variant of the above. Icons that launch terminal emulators take the form of stylized shells, little desktop PCs, or black rectangles containing a command-line prompt. Your system may offer several different terminal emulators, and you can experiment with them to find the one you prefer. Helpful features to look for include the ability to resize the terminal window, the ability to customize the background and font, and the ability to open up multiple independent tabbed or windowed shell sessions.

Regardless of whether you have logged in graphically or remotely, the terminal emulator will be displaying a command-line prompt. The exact appearance of this prompt depends on the variant of Unix you are using, which shell program (there are several), and how the system has been configured. The default command line prompt in Linux Mint (and other Ubuntu-derived distributions) is this:

```
username@hostname:~$
```

This prompt displays the name of the user that is currently logged in, the short name of the computer system itself, and the current directory (as explained below, the home directory is usually displayed as “~”). The `$` symbol marks the end of the prompt, where you would start typing. When running commands with elevated privileges, the `$` symbol will change to `#`. On my office system, my login name is `lstein` and my workstation is named `ODL`, so the command-line prompt displays as `lstein@ODL:~$`

Working at the command line will be a foreign experience to many readers. Although it will never be completely painless, a few features do make working at the command line easier. First, most command-line shells offer in-place editing. You can use the left and right cursor keys to move the text insertion point back and forth on the command line in order to insert and delete characters. The backspace key will delete characters to the left of the insertion point, and the delete key will delete characters to the right of the insertion point.

If you find yourself repeating many commands with minor variations, the up (↑) and down (↓) cursor keys will activate the shell’s “command history” feature. Pressing the up-cursor key will insert the last-issued command at the prompt. Pressing again will fetch the command previous to that, and so forth. You can press Enter to re-issue the command, or use the cursor keys to edit the command prior to issuing it again.

Most shells also offer a “command completion” feature. With this feature, you can type the first few letters of a command or file name and then press the Tab key. The shell will complete the command for you, or, if what you typed was ambiguous, display a number of alternatives from which to make a selection.

Command Syntax

Unix commands are case-sensitive, meaning that the commands `mkdir` and `Mkdir` are not the same. The first command will create a new directory. The second is not recognized on typical Unix systems and will result in a `Command not found` warning. Unix commands typically take arguments, which are separated from the command name by one or more spaces called “whitespace.” As a concrete example, the `mkdir` command takes a series of arguments giving the names of some directories to create. This command will create three directories named `docs`, `toy`, and `experiments`:

```
lstein@ODL:~$ mkdir docs toy experiments
```

Although it is not a good idea to create filenames that contain whitespace, you may do so by surrounding the name with double or single quotes. In contrast to the previous example, this one will create two directories, one named `docs` and one named `toy experiments`:

```
lstein@ODL:~$ mkdir docs "toy experiments"
```

In this and other examples, I distinguish what the user types by using boldface.

Options

Many Unix commands accept “options” which modify their behavior. Depending on the command, its options may be single-letter codes preceded by a hyphen, as in `-v`, or fully spelled-out words preceded by two hyphens, as in `--verbose`. Options come after the command name and before any arguments. For example, to have the `mkdir` command print out what it is doing, use the `--verbose` option:

```
lstein@ODL:~$ mkdir --verbose docs toy experiments
mkdir: created  directory 'docs'
mkdir: created  directory 'toy'
mkdir: created  directory 'experiments'
```

Getting Information on Commands

When given the `-h` or `--help` options, most commands will print out a brief usage summary. Try `-h` first, and if that doesn't work try `--help` as shown here:

```
lstein@ODL:~$ mkdir -help
Usage: mkdir [OPTION] ... DIRECTORY ...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE set file mode (as in chmod), not a=rwx - umask
  -p, --parents no error if existing, make parent directories as needed
  -v, --verbose print a message for each created directory
  -Z, --context=CTX set the SELinux security context of each created
       directory to CTX
        --help display this help and exit
        --version output version information and exit
```

`man` *command*

For more detailed help, the `man` (manual) command is extremely useful. Invoke it with the name of the command you wish help on (e.g., `man mkdir`). This will display a page of detailed information on how to use the command. If you don't know the name of the command for which you are looking, try the `apropos` command (e.g., `apropos directory`) to generate a list of commands that might have something to do with the function for which you're looking.

The `man` command users a "pager" to display a manual page that is longer than will fit comfortably into a terminal window. The pager is very simple. It displays a single page for your perusal. When you are ready for the next page, hit the space bar or Page Down. To go back a page, press "b" or Page Up. When you are done reading, press "q" to quit. The pager has numerous other key commands. Press "h" for help.

Suspending and Killing Commands

At some point while working with the command shell you will issue a command that either produces large amounts of output, takes a long time to run, or does something unexpected (and undesirable). In this case, you can interrupt a command in either of two ways.

To interrupt a command before it has finished running press Control-C. This means pressing the Control key (marked Ctrl on most PC keyboards) and simultaneously pressing the (lowercase) "c" key. In most cases, this will interrupt the command and return you to the command prompt.

To temporarily suspend a command without killing it entirely, press Control-Z. This will put the command into suspended animation and return you to the command prompt. To resume the command, type `fg` (foreground). You can suspend and resume a command as many times as you like.

All the Unix commands we have seen so far are short lived. For example, the `mkdir` command does its work and returns almost instantly. However, other commands are long lived. This is particularly true of commands that launch graphical programs such as Web browsers or text editors. In such cases, you will not be able to use the command line until the program has finished executing and the command-line prompt has reappeared.

To avoid losing the use of the command line, you can place an ampersand (`&`) after the name of a program that will take a long time to execute. This will place the program in the "background" and return you to the command-line prompt immediately. For example, the command `libreoffice&` will launch the Libre Office word processor in the background. The Libre Office window will appear, and you will be returned to the command-line prompt in the terminal window.

If you forget to add the ampersand and lose your command line, you can temporarily suspend the running program by typing Control-Z in the terminal window. The command-line prompt will reappear. Type `bg` (background), and the suspended program will be restarted in background mode. The `bg` command is handy whenever you wish to place a long-running command into the background and continue using the shell in the foreground.

MANAGING FILES AND DIRECTORIES

Like other operating systems, a fundamental part of Unix is its support for files and directories. A file can contain text, computer code, word processing data, images, sounds, or any other data. A directory, equivalent to the Macintosh or Windows "folder," contains files and/or other directories.

If you are logging in via a terminal emulator, you will have to learn to work with files via the command line. If you have a graphical login, chances are that the desktop environment provides a file browser. With the browser, you can view the contents of directories, peek into files, create new directories, move existing files and directories around, and so forth. Even so, you will need to learn the basic shell commands for manipulating files and directories.

Unix has the concept of the "current working directory," the default directory that the various file manipulation commands operate on if not otherwise specified. When you first log in, the current working directory is set to your "home directory," a directory to which you have full access and where you will normally store your personal files and other data.

List Command

To see the contents of your home directory, issue the `ls` (list) command:

```
lstein@ODL:~$ ls
chloroplast.png   Documents    Downloads   INBOX   Music
Pictures          plastid.png  Projects
```

Fancy option

The `ls` command shows a formatted list of files and directories, but does not provide any indication about which is which. For a more informative display, use the `-F` (fancy) option. The `ls -F` command shows a marked-up version of the directory listing. Directories end in the slash character (`/`), executable files (those that contain computer code) end in an asterisk (`*`), and symbolic links (a type of alias or shortcut) end in the "at" character (@), while regular files have no special character at the end.

```
lstein@ODL:~$ ls -F
chloroplast.png   Documents/   Downloads/  INBOX   Music/
Pictures/         plastid.png  Projects/
```

Some of the files shown in the example above are text files. An example is `INBOX`, which contains a list of recent E-mail messages to the author. Others contain image data such as `chloroplast.png` and `plastid.png`, which are both images of genomic annotations of the rice chloroplast. Unix distinguishes file types by using distinctive file name extensions. For example, `.png` is used for a file that contains portable network graphics image data. Unlike some systems, where file extensions are limited to three characters, Unix extensions can be of any length.

Long version option

Another useful variant of `ls` is the long version, invoked with `ls -lF`. This adds detailed information to the listing. This form will tell you how large the file or directory is, which user owns it, and what its access permissions are:

The first column of the long listing indicates the file permissions and its interpretation is beyond the scope of this appendix; however, it is handy to know that the `d` that sometimes appears at the beginning of the column indicates that the corresponding item is a directory.

```
lstein@ODL:~$ ls -lF
-rw-rw-r--   1   lstein lstein 28089    May 11 15:04 chloroplast.png
drwxrwxr-x   2   lstein lstein 4096     May 11 15:05 Documents
drwxrwxr-x   2   lstein lstein 4096     May 11 15:05 Downloads
-rw-rw-r--   1   lstein lstein 11242    May 11 15:04 INBOX
drwxrwxr-x   2   lstein lstein 4096     May 11 15:05 Music
drwxrwxr-x   2   lstein lstein 4096     May 11 15:05 Pictures
-rw-rw-r--   1   lstein lstein 153719   May 11 15:04 plastid.png
drwxrwxr-x   2   lstein lstein 4096     May 11 15:05 Projects
```

`-lF` is actually a combination of two command-line options: `-l`, which creates a detailed (long) listing and the `-F` (fancy) option that we discussed earlier. You can cluster single-letter options as an alternative to the longer form `ls -l -F`.

"All" option

By convention, Unix uses files whose names begin with a period (`.`) to hold software configuration information. Since there are many of these in your home directory, the `ls` command skips over these hidden files by default. To force `ls` to show all files, including those that are ordinarily hidden, use the `-a` option:

```
lstein@ODL:~$  ls -a
.ptksh_history        .qtella
.DCOPserver_pesto@    .qtella.hosts
.FVWM2-errors         .registry*
.FVWM95-errors        .rhmapper
.ICEauthority         .rhosts
.MCOP-random-seed     .rnd
...
```

Directory Paths

To view the contents of a directory, you have several options. One is to use the directory name (e.g., `Documents`) as the argument of the `ls` command, as in `ls Documents`). We can peek down even further by providing `ls` with a directory "path." A path is simply a list of directories separated by slashes, as in `ls Documents/Papers/Epigenetics`. A shorthand for the home directory (the one you enter when you first start a shell session) is "~", while the current directory is denoted by "`.`" and the parent directory by "`..`". From within the home directory, all these `ls` commands give equivalent results:

```
lstein@ODL:~$     ls Documents
lstein@ODL:~$     ls ~/Documents
lstein@ODL:~$     ls ./Documents
```

If Unix paths remind you of Web URLs in any way, that is not a total coincidence. The Web was originally built on top of Unix.

Change directory command

Another way of exploring a directory is to make it the current working directory so that `ls` operates on it by default. You do this with the `cd` (change directory) command:

```
lstein@ODL:~$ cd Documents
lstein@ODL: ~/Documents$
```

The `cd` command takes a single argument, the directory path, to make the current working directory. The indicated directory then becomes the default directory for `ls` and other file utilities. The prompt usually shows you what directory you are working in as illustrated in the above example.

To change back to your home directory, simply type the `cd` command without any arguments.

Print working directory command

If your prompt doesn't have a working directory indicator, you can find out the current directory with the `pwd` (print working directory) command:

```
lstein@ODL:~$ pwd
/home/lstein/Documents
```

Unlike the shell prompt, `pwd` doesn't indicate the home directory with a tilde (~), but prints out the complete path, which in this case is `/home/lstein/Documents`.

Common Commands and Shortcuts

This section describes common command-line utilities and shortcuts.

Move/rename command

To move a file or directory from one location to another, use the `mv` (move) command. It takes two arguments: the file or directory to move and the location to move it to. For example, the following command will move the directory `networking_tutorial.txt` and all its contents into the directory `talks`:

```
lstein@ODL:~/Documents$ mv networking_tutorial.txt talks/
```

The `mv` command can also be used to rename an existing file or directory. This example will rename the file `genbank.tar.gz` to `genbank.tgz`:

```
lstein@ODL:~/Documents$ mv genbank.tar.gz genbank.tgz
```

The difference between the two commands is that in the first case the second argument was an existing directory, and so was interpreted by the `mv` command as an instruction to move the first argument into that directory. In the second case, the second argument was not a directory, and so was interpreted by `mv` as a command to rename the indicated file.

Copy command

The `cp` command will make a copy of a file (but not a directory). It is simple to use:

```
lstein@ODL:~$ cp INBOX INBOX.bak
```

This creates an identical copy of the file `INBOX` named `INBOX.bak`.

To copy a directory and all its contents, pass the `-r` (recursive) option. This is often combined with the `-p` (preserve) option to keep the owner and creation/modification timestamps the same on the new copies. The command shown here will copy the directory `talks` and all its contents into a new directory named `talks.bak`:

```
lstein@ODL:~$ cp -pr talks talks.bak
```

Remote "copy" and Rsync commands

Moving files among two networked Unix/Linux systems is almost the same as moving them within the same system, as long as the remote machine runs Secure Shell and you have a log-in there. Instead of `cp` use the `scp` (secure copy) command. The syntax looks like this:

```
lstein@ODL:~$ scp -pr talks lstein@host.domain.edu:talks.bak
```

This command will copy the directory **talks** and all its contents into the home directory of user `lstein` on the remote machine *host.domain.edu*, storing the copy under the name `talks.bak`. Notice the "`:`" symbol that separates the remote host name from the remote directory.

If you find yourself using this command frequently to keep a backup of a directory in a remote location; then you may wish to use the Rsync command. Rsync behaves like `scp`, except that it intelligently only copies files that are newer on the local side than on the remote side. The syntax is very similar to the `scp` example shown earlier, except that you are recommended to use the `-a` (archive) option when copying whole directories:

```
lstein@ODL:~$ rsync -a talks lstein@host.domain.edu:talks.bak
```

Make and remove directory commands

To create a new directory, use the `mkdir` (make directory) command. This takes a list of one or more directory names and creates them in the current working directory. To remove an empty directory, use `rmdir` (remove directory). This command will fail if the directory is not empty.

Remove command

To delete a file, use the `rm` (remove) command. It takes a list of files and deletes them. The deletion is irrevocable—i.e., unlike Windows and Macintosh systems there is no recycle bin or trashcan from which to retrieve deleted files. A useful variant of `rm` is `rm -r`, which will delete a directory and all its contents; however, be careful with this, as it is easy to delete more than you intend.

Wild cards

The shell provides several convenient shortcuts. One is the use of wild cards, which allow you to refer to several files or directories at once. An asterisk (`*`) appearing in a command line argument is treated as a wild card that can match any series of zero or more characters, while a question mark (`?`) can match any single character.

Using wild cards, you can get a detailed listing of all PNG files or all files with `plastid` in their name, as shown in these two examples:

```
lstein@ODL:~$ ls -lF *.png
lstein@ODL:~$ ls -lF *plastid*
```

Using directory abbreviations

As described earlier, if you are in a nested subdirectory and you want to refer to the directory above the current one, you can refer to this directory with the special name "`..`" (two dots). For example, the following command, when executed from your home directory, will list the contents of the directory that contains it:

```
lstein@ODL:~$ ls -lF ..
```

The symbol "`.`" stands for the current working directory.

The shell also lets you type the tilde symbol (~, found in the upper left-hand corner of most keyboards) to refer to your home directory. You can obtain a listing of your home directory like this:

```
lstein@ODL:~$ ls -F ~
```

and return to your home directory from wherever you are by typing:

```
lstein@ODL:~$ cd ~
```

For your convenience, typing `cd` without arguments will also return you to your home directory, making it the current working directory.

WORKING WITH TEXT FILES

Creating and manipulating files of text is central to most bioinformatics activities. Unix gives you a large number of ways to manipulate text files.

The fastest and easiest way to view the contents of a text file is with the `less` command. It takes a list of one or more text file names, and displays them on the screen one page at a time. This works even with very large files. When `less` is displaying the contents of a text file, you will see a subtle "`:`" prompt at the bottom of the screen. This prompt indicates that there is more of the file to display. As described earlier, you can page through the file by pressing the Page Up and Page Down keys. It also allows you to navigate a line at a time using the up and down cursor keys, and to search through the file for words and phrases by typing `/search_term`. The `less` command has many options and features: type `h` to bring up a help screen.

Redirecting Output to a File

Much of the software used in bioinformatics produces large amounts of text data. This information is often written directly to the terminal, and it can be frustrating to see something interesting scroll by into the irretrievable oblivion beyond the top of the terminal window.

One way to deal with this situation is to use the output redirection feature of the Unix shell. The output of any command can be redirected into a file by following the command with a > sign followed by the name of the file you wish to create. For example, the `blastn` command (see *UNIT 3.3*; Ladunga, 2009) will write the results of its search to the terminal window. To redirect this to a file named `blastn.out`, issue the command as shown here:

```
lstein@ODL:~$ blastn -query mydata.fa -db nr -remote >blastn.out&
```

Notice that we appended an ampersand (`&`) to the end of the command in order to make it run in the background. Without this, we would not be able to use the command line until the blastn program completed (although we could always open up another terminal session). After the command completes, you will find its output in `blastn.out`, which you can then inspect with `less`, or a text editor.

If the file indicated with > already exists, it will be overwritten, erasing whatever was there before. If you prefer to append the command output to the file, leaving its previous contents intact, use >> instead of >.

Redirecting Output to Less

Another handy alternative, useful for those cases when there is more output from a command than will fit onto a terminal screen, but you don't need to save the information to a file, is to redirect output directly to the `less` program. You can do this using the "pipe" or vertical bar symbol "`|`":

```
lstein@ODL:~$ blastn -query mydata.fa -db nr -remote | less
```

Blastn's output will now be captured by `less` and displayed a screen at a time for easy viewing.

Unix Text Editors

To work effectively with Unix-based bioinformatics software, you will need to be able to create and modify text files from scratch. This means becoming proficient with one or more of the Unix text editors.

Unlike the more familiar word processors, Unix text editors produce files that are devoid of any fancy fonts or formatting. They also have a reputation for being unfriendly to novice users. This is only partly true. The graphical desktop environments are each equipped with user-friendly text editors similar in style to the Windows Notepad desk accessory. In Linux distributions, you will usually find a suitable text editor lurking in the Accessories section of the applications menu. If the application name is not obvious (like "Text Editor"), look for Gedit, Kedit, NEdit, and/or Kate. Be aware that when you are editing plain text files, such as those used for software tool configuration files, you should always use a simple text editor rather than a word processor such as Microsoft Word. Word processors introduce formatting characters into the text file that will confuse most software. Indeed, even the tiny text editors that are provided as desktop accessories in Microsoft Windows and Mac OS X can lead to problems unless you are careful to save the files as "text only"; you are usually best off creating and editing text files within the Unix environment.

If you have access to one of the graphical editors, you should have no problem creating and editing text files, since everything can be done using mouse clicks and menu commands. If you must interact with Unix via a text-only terminal, life will be slightly more interesting.

nano

If you have never used a Unix text editor before, it is suggested that you begin with the nano text editor. This editor is installed on most (but not all) Unix systems, and has a relatively straightforward user interface. To launch it, type `nano` at the command line. This will replace the contents of the terminal window with the editor screen shown in Figure A1.C.6. The middle of the screen is the current contents of the text file. Use the cursor keys to move around in the file, the Backspace key to delete text to the left of the insertion point, and the Delete key to delete text to the right of the insertion point.

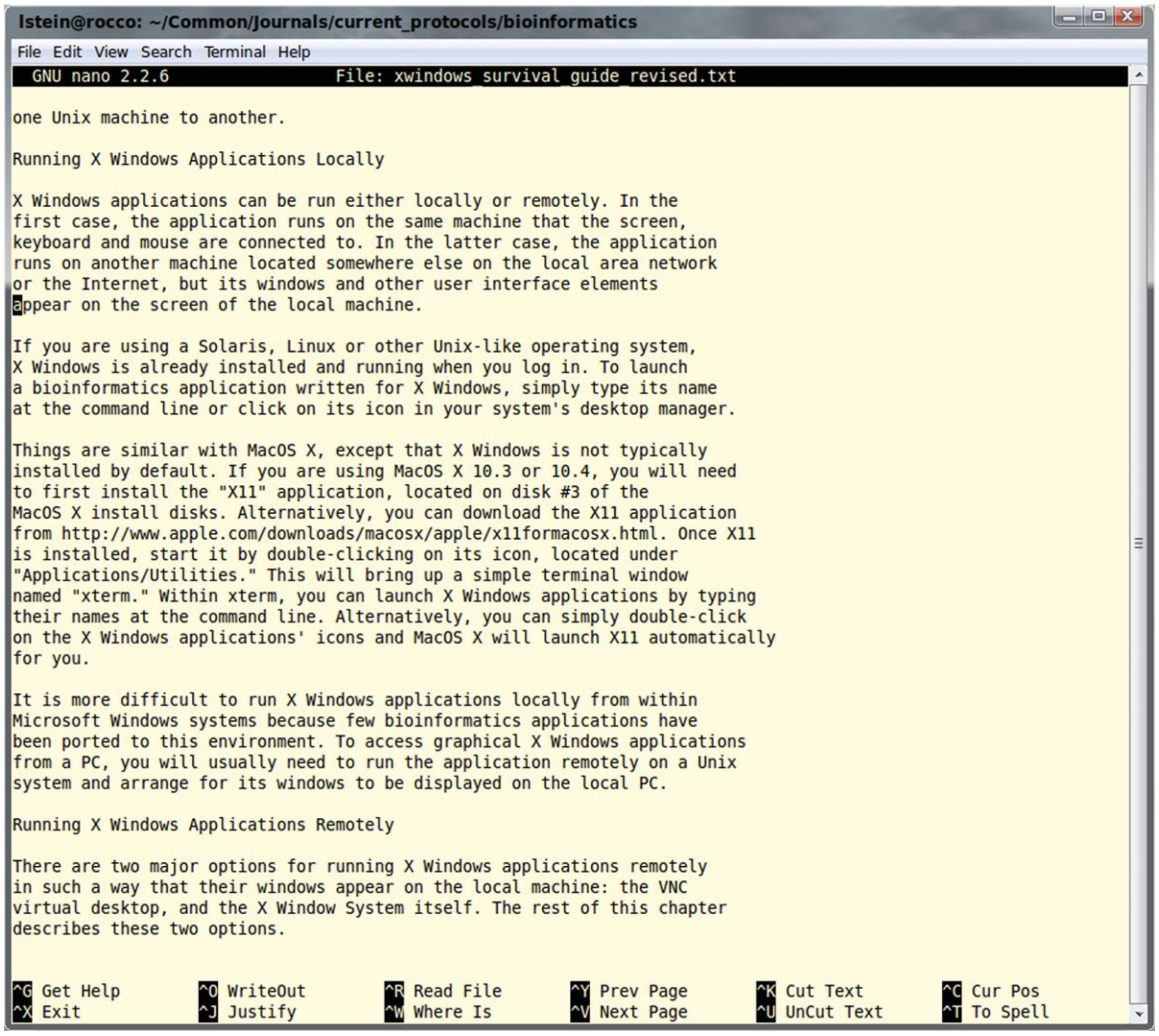
Various Control key combinations allow you to read files, save files, and exit the program. The currently available commands are listed at the bottom of the nano window using notation in which a caret (`^`) means the Control key. So `^X Exit` means to press Control-X in order to exit the program.

To create a text file from scratch, launch nano, type the text, and then press Control-O to write out (i.e., output) the file. You will be prompted to type in the name of the file. To read in an existing file, press Control-R. You will be prompted for a file name, which will then be appended to the bottom of whatever is currently on display. Another way to edit an existing file is to give its name to nano on the command line. For example, for the file `test.txt`, the following will cause nano to open and edit the file:

```
lstein@ODL:~$ nano test.txt
```

Other text editors

Nano has relatively limited abilities. Much more powerful Unix text editors include the vi ("visual") editor, and Emacs. Using vi requires learning a set of cryptic keyboard-based commands. Emacs is slightly easier to use in the graphical desktop environment because


lstein@rocco: ~/Common/Journals/current_protocols/bioinformatics
File Edit View Search Terminal Help

```
  GNU nano 2.2.6              File: xwindows_survival_guide_revised.txt

one Unix machine to another.

Running X Windows Applications Locally

X Windows applications can be run either locally or remotely. In the
first case, the application runs on the same machine that the screen,
keyboard and mouse are connected to. In the latter case, the application
runs on another machine located somewhere else on the local area network
or the Internet, but its windows and other user interface elements
appear on the screen of the local machine.

If you are using a Solaris, Linux or other Unix-like operating system,
X Windows is already installed and running when you log in. To launch
a bioinformatics application written for X Windows, simply type its name
at the command line or click on its icon in your system's desktop manager.

Things are similar with MacOS X, except that X Windows is not typically
installed by default. If you are using MacOS X 10.3 or 10.4, you will need
to first install the "X11" application, located on disk #3 of the
MacOS X install disks. Alternatively, you can download the X11 application
from http://www.apple.com/downloads/macosx/apple/x11formacosx.html. Once X11
is installed, start it by double-clicking on its icon, located under
"Applications/Utilities." This will bring up a simple terminal window
named "xterm." Within xterm, you can launch X Windows applications by typing
their names at the command line. Alternatively, you can simply double-click
on the X Windows applications' icons and MacOS X will launch X11 automatically
for you.

It is more difficult to run X Windows applications locally from within
Microsoft Windows systems because few bioinformatics applications have
been ported to this environment. To access graphical X Windows applications
from a PC, you will usually need to run the application remotely on a Unix
system and arrange for its windows to be displayed on the local PC.

Running X Windows Applications Remotely

There are two major options for running X Windows applications remotely
in such a way that their windows appear on the local machine: the VNC
virtual desktop, and the X Window System itself. The rest of this chapter
describes these two options.

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure A1.C.6 Nano is a good text editor with which to get started. The ˆ characters in the menu at the bottom stand for the Ctrl key.

it provides menus for most common commands, but it is not nearly as straightforward as more familiar word processors. However, if you plan to work heavily in the Unix environment, it is worth investing some time learning one of these editors. Good introductions to vi and Emacs can be found in most general-audience Unix books.

CHANGING THE ENVIRONMENT

Various Unix commands, and several bioinformatics programs, are dependent on “environment variables,” a set of configuration variables that are set up for you each time you log into the system. In this section, we walk through a practical example of changing an environment variable.

The `EDITOR` *environment variable*

Various Unix commands will automatically invoke a text editor for you when needed (e.g., when examining a configuration file). The default text editor is vi, a powerful but extremely cryptic text-based editor. To make nano your default editor, you must alter the value of an environment variable named `EDITOR`.

To change the `EDITOR` environment variable, you must edit one of the hidden “dot” files located in your home directory. Which file you edit depends on which shell interpreter you are using, but Ubuntu, Linux Mint, and most other modern distributions use the "bash" shell. You will want to change the `.bashrc` hidden file located in your home directory. You will see the file listed if you issue the `ls` command with the `-a` option, as described earlier.

First, create a backup copy of `.bashrc` by running the `cp` command:

```
lstein@ODL:~$ cp .bashrc .bashrc.bak
```

Now using nano (or another text editor), open `.bashrc` by issuing the command:

```
lstein@ODL:~$ nano .bashrc
```

Scroll down to the bottom of the file and add the following line:

```
export VISUAL=nano
```

Log out and in again, and run `echo $NANO` (be sure to include the `$`) to confirm that the environment variable has indeed been set.

Other variables

You can follow this procedure to add or modify any number of environment variables. Just be sure to put each `export` command on a separate line. If you make a mistake, your shell may start misbehaving. Do not panic. Just copy the original back into place:

```
lstein@ODL:~$ cp .bashrc.bak .bashrc
```

INSTALLING UNIX SOFTWARE

Introduction

The next topic that we'll cover in this survival guide is installing and upgrading software.

Unix software can be as a binary package distributed using the operating system's package manager, as a source code archive, or using a programming language-specific installer. In this section we will show how to install binary and source code packages. See the language-specific guides on the Web for instructions on how to install python, ruby, perl and R modules.

Installing a Binary Package

If the software you wish to install is widely used, chances are that it is available as a binary package. In this case you are in luck—installation is extremely easy!

In Linux Mint and other Ubuntu-based Linux systems, you will use the `apt-cache` and `apt-get` commands to search for and install binary packages. First make sure that your system's list of installable packages is up to date. The command `apt-get update` synchronizes the local package list with a series of remote archives:

```
lstein@ODL:~$ sudo apt-get update
[sudo] password for lstein: ******
Ign http://security.ubuntu.com trusty-security InRelease
Ign http://dl.google.com stable InRelease
Ign http://packages.linuxmint.com qiana InRelease
Get:1 http://security.ubuntu.com trusty-security Release.gpg [933 B]
Ign http://extra.linuxmint.com qiana InRelease
Hit http://dl.google.com stable Release.gpg
Get:2 http://security.ubuntu.com trusty-security Release [63.5 kB]
```

```
Hit http://dl.google.com stable Release
...
```

Note the use of the sudo command in front of apt-get. For security reasons, installing and removing software is a privileged task on Linux systems. In order to temporarily obtain escalated privileges, you prefix the command line with sudo and provide your login password.

After the update completes, search for the package of interest using apt-get search *searchterm*. In this example, we are searching for RNA-seq analysis programs:

```
lstein@ODL:~$ apt-cache search rna-seq
cufflinks - Transcript assembly, differential expression and regulation for RNA-Seq
kissplice - Detection of various kinds of polymorphisms in RNA-seq data
python-htseq - high-throughput genome sequencing read analysis utilities
r-bioc-cummerbund - tool for analysis of Cufflinks RNA-Seq output
r-bioc-edger - Empirical analysis of digital gene expression data in R
tophat - fast splice junction mapper for RNA-Seq reads
```

We found just six hits in this case. Notice that we did not need to use sudo here because searching for packages does not require special privileges. Let us install the “tophat” spliced junction mapper using apt-get install:

```
lstein@ODL:~$ sudo apt-get install tophat
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bowtie2
Suggested packages:
  bowtie2-examples cufflinks bowtie
The following NEW packages will be installed:
  bowtie2 tophat
0 upgraded, 2 newly installed, 0 to remove and 310 not upgraded.
Need to get 1,642 kB of archives.
After this operation, 6,016 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu/trusty/universebowtie2amd64
2.1.0-2 [1,029 kB]
Get:2 http://archive.ubuntu.com/ubuntu/trusty/universetophatamd64
2.0.9-1ubuntu1 [614 kB]
Fetched 1,642 kB in 0s (1,671 kB/s)
Selecting previously unselected package bowtie2.
(Reading database ... 224542 files and directories currently installed.)
Preparing to unpack .../bowtie2_2.1.0-2_amd64.deb ...
Unpacking bowtie2 (2.1.0-2) ...
Selecting previously unselected package tophat.
Preparing to unpack .../tophat_2.0.9-1ubuntu1_amd64.deb ...
Unpacking tophat (2.0.9-1ubuntu1) ...
Processing triggers for doc-base (0.10.5) ...
Processing 1 added doc-base file...
Registering documents with scrollkeeper...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up bowtie2 (2.1.0-2) ...
Setting up tophat (2.0.9-1ubuntu1) ...
```

Notice that we needed to again prefix the apt-get command with `sudo` in order to increase privileges. Also notice that the installer installed `bowtie2` as well as `tophat`. This is because `tophat` is dependent upon `bowtie2`, and the installer is intelligent enough to install all the prerequisites needed to run a package.

That is all that is needed. You can now run `tophat` or read through its manual page.

On RedHat and CentOS based Linux flavors, you will use the `yum` command instead of `apt`. The `yum` command works in almost the same way as `apt-cache` and `apt-get`, but the syntax is slightly different.

Installing a Source Code Package

Installing from source code is a time-honored tradition in bioinformatics. Unfortunately it is a difficult and error-prone process. I will illustrate the process using the latest iteration of the BWA next-generation DNA sequence aligner, which at the time of this writing had not made it into the binary packaging system.

First, make sure that your Linux system has all the software installed that is needed to compile source code. You do this by installing the “build-essential” binary package:

```
lstein@ODL: ~$ sudo apt-get install build-essential
```

Next, download the source code package. You could use a Web browser for this, or if you happen to know the exact URL, you can use the command-line tool `wget`:

```
lstein@ODL:~$ wget -O bwa-latest.tar.bz2 'http://sourceforge.net/projects/bio-
bwa/files/latest/download?source=files'
Saving to: 'bwa-latest.tar.bz2'
100%[=======================================>] 184,599 382 KB/s in 0.5s
2015-05-11 22:05:56 (382 KB/s) - 'bwa-latest.tar.bz2' saved [184599/184599]
```

In this example, the wget `-O` option specifies the name of the destination file.

The source-code file ends with `.tar.bz2`. This means that the directory was packaged up into a single archive file using the “tar” command, and then the archive was compressed using the “bzip2” command. You can uncompress and untar the file in a single step like this:

```
lstein@ODL:~$ tar jxvf bwa-latest.tar.bz2
bwa-0.7.12/
bwa-0.7.12/bamlite.c
bwa-0.7.12/bamlite.h
bwa-0.7.12/bntseq.c
bwa-0.7.12/bntseq.h
bwa-0.7.12/bwa.1
bwa-0.7.12/bwa.c
bwa-0.7.12/bwa.h
bwa-0.7.12/bwakit/
bwa-0.7.12/bwamem.c
...
```

The mysterious `jxvf` string following `tar` is actually a set of unpacking options. Most source code packages are distributed using bzip2 compression, but some of the older ones are distributed using gzip and have the extension `.tar.gz` or `.tgz`. In this case change the option string to `zxvf`.

Enter the newly-created directory using `cd` and generate a directory listing using `ls`:

```
lstein@ODL:~$ cd bwa-0.7.12
lstein@ODL:~/bwa-0.7.12$ ls --F
bamlite.c        bwase.c      bwt_lite.h       khash.h         malloc_wrap.h
bamlite.h        bwase.h      bwtsw2_aux.c     kopen.c         NEWS.md
bntseq.c         bwaseqio.c   bwtsw2_chain.c   kseq.h          pemerge.c
bntseq.h         bwashm.c     bwtsw2_core.c    ksort.h         QSufSort.c
bwa.1            bwtaln.c     bwtsw2.h         kstring.c       QSufSort.h
bwa.c            bwtaln.h     bwtsw2_main.c    kstring.h       qualfa2fq.pl*
bwa.h            bwt.c        bwtsw2pair.c     ksw.c           README-alt.md
bwakit/          bwtgap.c     ChangeLog        ksw.h           README.md
bwamem.c         bwtgap.h     COPYING          kthread.c       utils.c
bwamem_extra.c   bwt_gen.c    example.c        kvec.h          utils.h
bwamem.h         bwt.h        fastmap.c        main.c          xa2multi.pl*
bwamem_pair.c    bwtindex.c   is.c             Makefile
bwape.c          bwt_lite.c   kbtree.h         malloc_wrap.c
```

There's a `README.md` file in there. Maybe we should read it?

```
lstein@ODL:~/bwa-0.7.12$ less README.md
##Getting started
  cd bwa-0.7.12; make
               ./bwa index ref.fa
               ./bwa mem ref.fa read-se.fq.gz | gzip -3 > aln-se.sam.gz
               ./bwa mem ref.fa read1.fq read2.fq | gzip -3 > aln-pe.sam.gz
##Introduction
BWA is a software package for mapping DNA sequences against a large
reference genome, such as the human genome. It consists of three algorithms:
...
```

The instructions tell us to enter the `bwa-0.7.12` directory, which we have already done, and then to run the "make" program. In the example they give us, the `cd` and `make` commands are issued on the same line using the semicolon to separate them.

Let's run "make" and see what happens:

```
lstein@ODL:~/bwa-0.7.12$ make
gcc -c -g -Wall -Wno-unused-function -O2 -DHAVE_PTHREAD -
DUSE_MALLOC_WRAPPERS utils.c -o utils.o
gcc -c -g -Wall -Wno-unused-function -O2 -DHAVE_PTHREAD -
DUSE_MALLOC_WRAPPERS kthread.c -o kthread.o
gcc -c -g -Wall -Wno-unused-function -O2 -DHAVE_PTHREAD -
DUSE_MALLOC_WRAPPERS kstring.c -o kstring.o
...
gcc -g -Wall -Wno-unused-function -O2 -DHAVE_PTHREAD -
DUSE_MALLOC_WRAPPERS QSufSort.o bwt_gen.o bwashm.o bwase.o bwaseqio.o
bwtgap.o bwtaln.o bamlite.o is.o bwtindex.o bwape.o kopen.o pemerge.o
bwtsw2_core.o bwtsw2_main.o bwtsw2_aux.o bwt_lite.o bwtsw2_chain.o
fastmap.o bwtsw2_pair.o main.o -o bwa -L. -lbwa -lm -lz -lpthread -lrt
lstein@ODL:~/bwa-0.7.12$
```

When `make` runs, it marshals a bunch of other commands that compile the source code into an executable binary. If all went well, there should be a "bwa" executable in the current directory after "make" finishes up:

```
lstein@ODL:~/bwa-0.7.12$ ls -F
bamlite.c          bwape.o         bwt_lite.c        is.c          Makefile
bamlite.h          bwase.c         bwt_lite.h        is.o          malloc_wrap.c
bamlite.o          bwase.h         bwt_lite.o        kbtree.h      malloc_wrap.h
bntseq.c           bwase.o         bwt.o             khash.h       malloc_wrap.o
bntseq.h           bwaseqio.c      bwtsw2_aux.c      kopen.c       NEWS.md
bntseq.o           bwaseqio.o      bwtsw2_aux.o      kopen.o       pemerge.c
bwa*               bwashm.c        bwtsw2_chain.c    kseq.h        pemerge.o
bwa.1              bwashm.o        bwtsw2_chain.o    ksort.h       QSufSort.c
bwa.c              bwtaln.c        bwtsw2_core.c     kstring.c     QSufSort.h
bwa.h              bwtaln.h        bwtsw2_core.o     kstring.h     QSufSort.o
bwakit/            bwtaln.o        bwtsw2.h          kstring.o     qualfa2fq.pl*
bwamem.c           bwt.c           bwtsw2_main.c     ksw.c         README-alt.md
bwamem_extra.c     bwtgap.c        bwtsw2_main.o     ksw.h         README.md
bwamem_extra.o     bwtgap.h        bwtsw2_pair.c     ksw.o         utils.c
bwamem.h           bwtgap.o        bwtsw2_pair.o     kthread.c     utils.h
bwamem.o           bwt_gen.c       ChangeLog         kthread.o     utils.o
bwamem_pair.c      bwt_gen.o       COPYING           kvec.h        xa2multi.pl*
bwamem_pair.o      bwt.h           example.c         libbwa.a
bwa.o              bwtindex.c      fastmap.c         main.c
bwape.c            bwtindex.o      fastmap.o         main.o
```

Indeed, there is a new executable called “bwa" (the * in the fancy listing indicates that the file is an executable command). We can now test that it works:

```
lstein@ODL:~/bwa-0.7.12$./bwa
Program:     bwa (alignment via Burrows-Wheeler transformation)
Version:     0.7.12-
             r1039
Contact:     Heng Li <lh3@sanger.ac.uk>
Usage:       bwa <command> [options]
Command:     index       index sequences in the FASTA format
             mem         BWA-MEM algorithm
             fastmap     identify super-maximal exact matches
             pemerge     merge overlapping paired ends (EXPERIMENTAL)
             aln         gapped/ungapped alignment
             samse       generate alignment (single ended)
             sampe       generate alignment (paired ended)
             bwasw       BWA-SW for long queries
...
```

Indeed, the package appears to be working, at least up to the point of printing out its usage instructions. Note that we had to invoke the command by prefixing the file name with `./`. This tells the shell to look for the `bwa` file in the current directory (recall that “`.`” is a shortcut for the current working directory). If you change your working directory, you will have to tell the system where to look for the executable. For example, from some other directory, you would invoke `~/bwa-0.7.12/bwa`.

The last step is to copy the `bwa` executable into a location where it can be found by the shell without having to indicate its location with a directory path. For many software packages, you can ask `make` to do this for you by issuing the command `make install`. However, with `bwa` you have to do it manually by copying the executable into the special directory `/usr/local/bin.` You can do this with `cp`:

```
lstein@ODL:~/bwa-0.7.12$ sudo cp bwa /usr/local/bin
[sudo] password for lstein: ******
```

Because `/usr/local/bin` is a privileged directory, you must again use `sudo` to escalate your privileges prior to executing the copy command. You have now added `bwa` to the set of commands recognized by the shell, and can invoke it by typing its name without the directory path.

There are an infinite number of variations for compiling and installing source code packages. Always read the accompanying documentation before starting down this path.

INSTALLING LINUX

In the last part of this guide, I will show you how to install a version of Linux on your own laptop or desktop machine, or, alternatively, how to run a remote Linux server on the Amazon compute cloud and connect to it remotely across the Internet.

Creating a Linux Virtual Machine on your Desktop

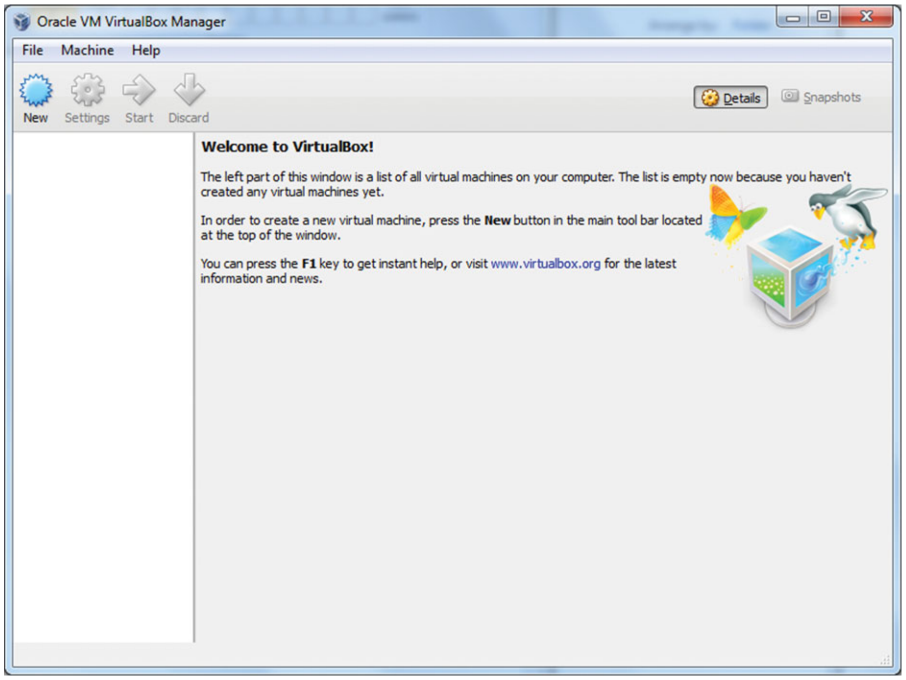
The recommended way to run Linux on your personal system (desktop or laptop) is to run it within a virtual machine (VM). The VM emulates the CPU, disk drive, keyboard, mouse and other hardware of a real machine, and provides an environment in which you can install a fresh version of Linux (or another operating system). When running, the VM will give you access to all Linux's facilities, including both command-line tools and the graphical desktop, and you can easily create, copy, and adapt multiple virtual machines that are customized to the tasks at hand. There is a slight performance penalty when running virtual machines, but their combined convenience and flexibility easily outweighs this problem.

The two VM management systems that I recommend are VirtualBox and VMWare Workstation. The former is an Open Source project that can be downloaded and used free of charge, and provides all the core features you need. VMWare Workstation requires a paid license. It is a more polished and feature-rich product, and is particularly recommended for users who wish to run 3-D graphics in the Linux environment or need access to USB 3.0 devices. In the walk-through described here, we will use VirtualBox as the example, but the process of installing a Linux VM in VMWare Workstation is broadly similar.

You will need two software packages to run Linux inside VirtualBox: an installable DVD image for the Linux distribution of your choice and the VirtualBox host software itself. There are many varieties of Linux to choose from. The one I recommend is Linux Mint (*http://www.linuxmint.com*); it is based on the popular Ubuntu distribution but has a desktop graphical environment that is less jarring to users familiar with the Windows and OS X desktops. Linux Mint itself comes in a variety of flavors: I suggest you download the DVD image for "Cinnamon 64-bit edition" (this will be a file ending in the extension `.iso`) and put it somewhere where you can find it later. You will not need to actually burn this image onto a DVD.

Next, download and install VirtualBox from *https://www.virtualbox.org/wiki/Downloads*; you will find installable packages for Windows and OS X systems. You may also wish to download VirtualBox Extension Pack, an add-on that provides support for USB 2.0 devices and remote desktops.

After installing VirtualBox, launch the VirtualBox Manager application and start the process of configuring a new virtual machine (Fig. A1.C.7) by clicking on the "New" icon. This will start up a wizard that will guide you through the configuration process. Give the new VM a descriptive name such as "Linux Mint," choose Type as Linux, and select Version a "Ubuntu (64 bit)". If at this point you only see 32-bit options, it means that you are running on older hardware and will have to go back and download the 32-bit version of Linux Mint. You can accept the defaults on the remainder of the screens,



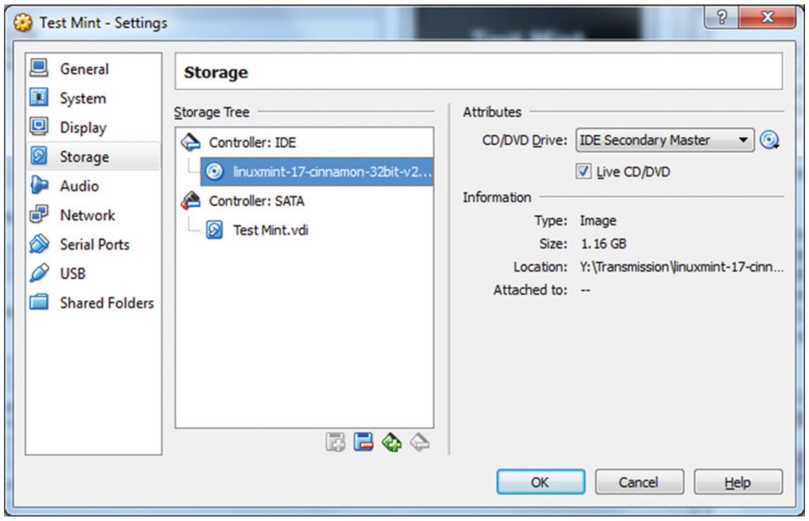
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Figure A1.C.7 The VirtualBox home screen is used to organize and control the virtual machines that you create. In this screenshot, we are about to create our first one.



CD/DVD Drive: IDE Secondary Master

Live CD/DVD

Information

Type: Image

Size: 1.16 GB

Location: Y:\Transmission\linuxmint-17-cinn...

Attached to: --

Figure A1.C.8 To boot a Linux installer image, navigate to the Storage part of the virtual machine settings and click the DVD icon to the right of the pull-down menu labeled IDE Secondary Master.

except that I recommend that when you get to the disk configuration screen, increase the amount of disk storage from the default 8 Gb to at least 10 Gb.

Once the new VM wizard is finished, there is one additional step needed before you can boot and install Linux. Select “Settings” from the VirtualBox Manager and then “Storage.” You will see a CD icon on the upper right side of this dialog (Fig. A1.C.8). Click this to pull down a CD configuration menu, select “Choose a virtual CD/DVD disk file . . . ” and then select the `.iso` image you downloaded from the Linux Mint Web site. Close the Settings dialog and then press the Start button in the main VirtualBox manager.

This will now boot the DVD installer image, which then guides you through the process of installing Linux on the virtual machine.

When the installation is done, the VM will reboot and take you into a graphical desktop environment where you can interact with the system, install further software, and execute bioinformatics tools (see Fig. A1.C.2). The Linux desktop can be set to run in a resizable window on your host system, or you can go into full-screen mode.

If something goes wrong during installation, you can easily delete the VM and start over again. The great thing about virtualization is that no permanent changes are made to your host machine.

For more information on running Linux within VirtualBox, please see the VirtualBox end-user documentation, located at *https://www.virtualbox.org/wiki/End-user_documentation*.

Running a Linux Server in the Cloud

An alternative to installing Linux on your desktop or laptop is to create a Linux VM on the commercial Amazon Elastic Compute Cloud (EC2). The advantage to doing this is that you can create quite powerful server machines on a pay-as-you go basis. If you need a powerful machine to complete a complex analysis, you can create a suitable VM, run the analysis job overnight, and then destroy the VM, paying only for the time that you used it for. You can also take advantage of the fact that several popular bioinformatics tools, such as Galaxy, exist in the form of EC2 Amazon machine images (AMIs); you can select and run these images to instantaneously become the owner of a server that has the tools already installed and configured.

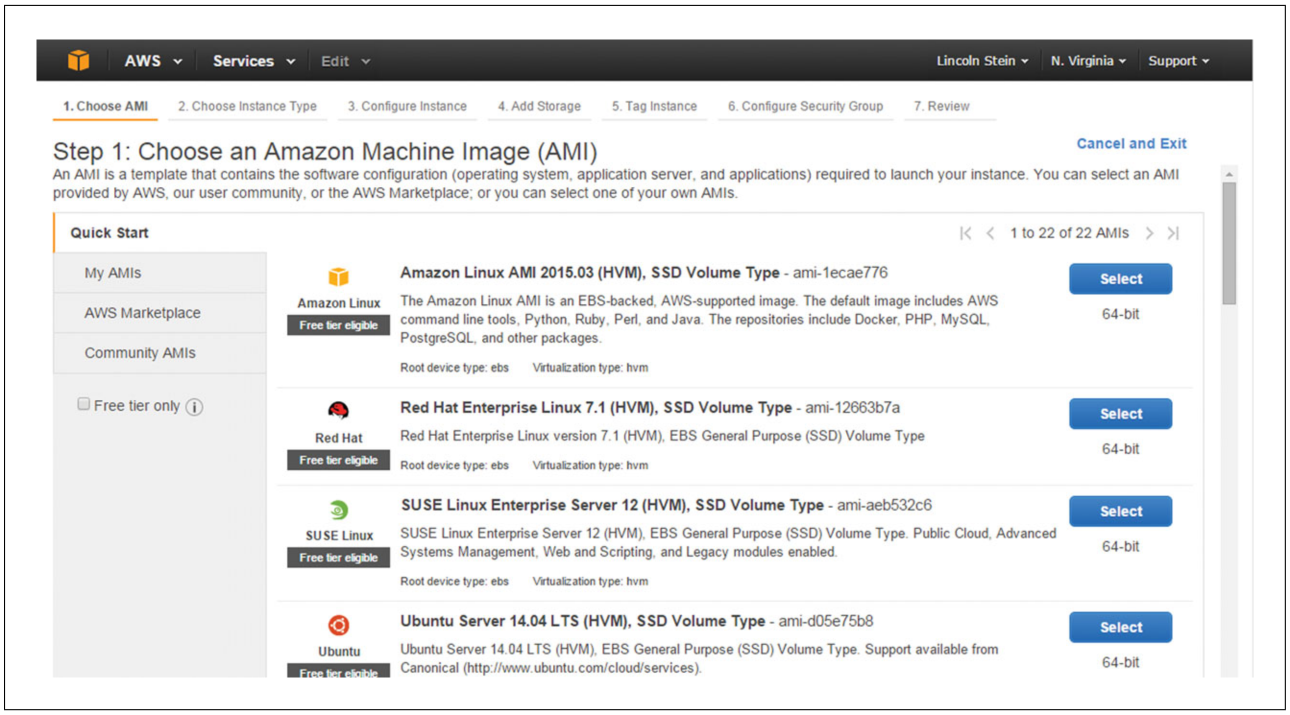
The disadvantage of running a server in EC2 is that you will only have access to it through the command line. In addition, because EC2 VMs run remotely, you will need to transfer data back and forth across the Internet. The Secure Shell (ssh) and Secure FTP (sftp) tools described earlier in this guide allow you to do this securely and conveniently.

To get started with EC2, you will need to set up an account and payment method (e.g., credit card) with Amazon. New users of EC2 have access to a free usage tier which provides roughly a year's worth of free time on a low-power VM. Point your browser to *http://aws.amazon.com/ec2/* and select the "Sign In to the Console" button in order to sign in using an existing Amazon account, or to create a new one. Upon logging in, you will be greeted by a screen listing a large number of Amazon services. Select "EC2."

This will take you to a dashboard that describes the VMs you have running; the EC2 nomenclature for a VM is an "instance." To provision and boot a new instance, press the prominent button labeled "Launch Instance". This takes you to the first screen of a Web-based wizard (Fig. A1.C.9). The first step is to select the operating system of the machine you wish to provision. There are a large number of choices, but for consistency with the examples in this guide, please choose "Ubuntu Server XX.XX LTS," where XX.XX is the version number (14.04 at the time of the writing, but likely to change in the future).

The second step of the wizard prompts you for the instance hardware configuration. There are a number of options ranging from small, low-performance machines to large multi-CPU systems. For this example, choose type `t2.micro`, a low-performance machine that is eligible for the free usage tier. Then, click Next.

The next three screens allow you to adjust various aspects of the instance's storage, network connection, and name. The defaults are generally fine, so click through them.



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Amazon Linux — **Amazon Linux AMI 2015.03 (HVM), SSD Volume Type** - ami-1ecae776 — 64-bit

The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.

Root device type: ebs Virtualization type: hvm

Red Hat — **Red Hat Enterprise Linux 7.1 (HVM), SSD Volume Type** - ami-12663b7a — 64-bit

Red Hat Enterprise Linux version 7.1 (HVM), EBS General Purpose (SSD) Volume Type

Root device type: ebs Virtualization type: hvm

SUSE Linux — **SUSE Linux Enterprise Server 12 (HVM), SSD Volume Type** - ami-aeb532c6 — 64-bit

SUSE Linux Enterprise Server 12 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.

Root device type: ebs Virtualization type: hvm

Ubuntu — **Ubuntu Server 14.04 LTS (HVM), SSD Volume Type** - ami-d05e75b8 — 64-bit

Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

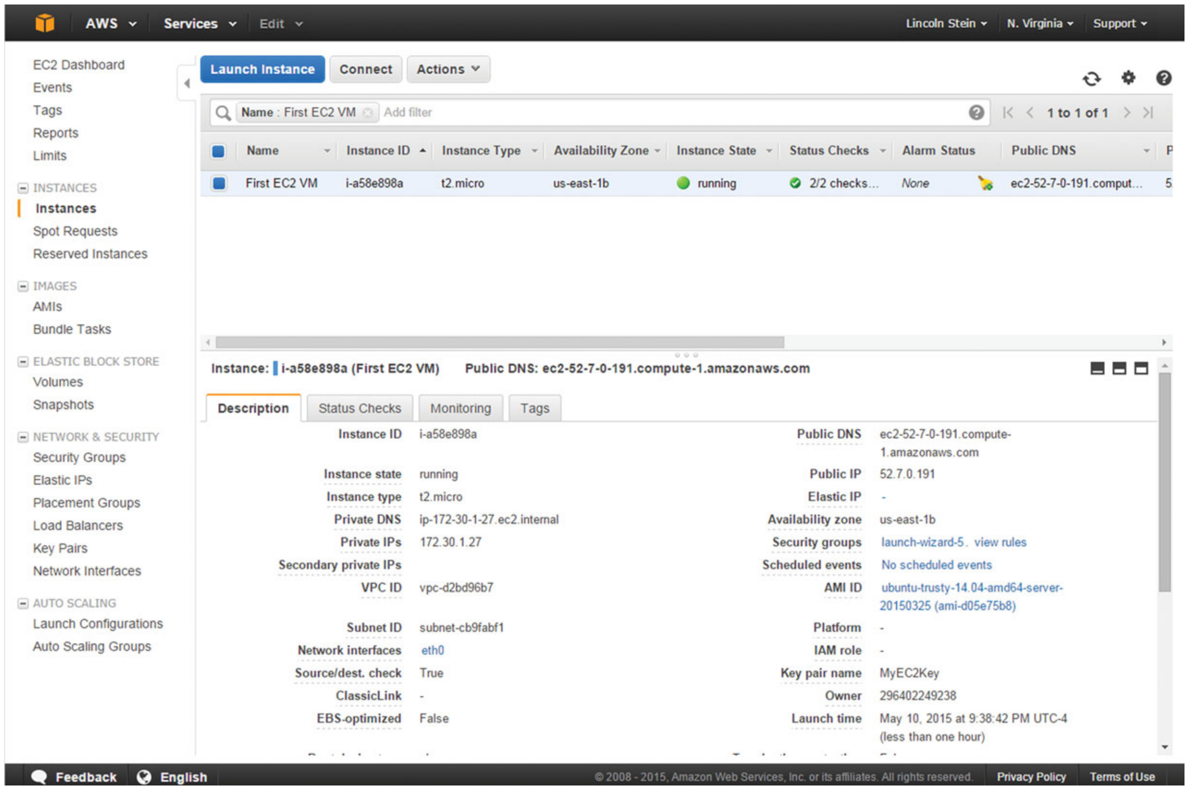
Figure A1.C.9 The first page of this Web-based wizard allows you to select the desired operating system for a new virtual machine.

The sixth screen is an important one. It controls the firewall rules that regulate which network connections the instance will accept. The wizard's default is to create a new set of rules that allow incoming connections from anywhere on the Internet to the Secure Shell (ssh) login service running on the instance. This is fine for an example, but when you are doing real work it is a good idea to restrict incoming connections to known, trusted IP addresses. You can do this now by selecting the pop-up menu under Source, and changing Anywhere to My IP. This will limit incoming connections to the computer at which you are currently working. Click "Review and Launch."

The final launch screen summarizes the configuration of the instance and allows you to make any desired changes. If everything looks good, click Launch. This will pop up a dialog box that prompts you to create or select a "key pair" for connections to the instance. The key pair is a cryptographic identity that identifies you to the instance when you log into it across the network. From the pop-up menu, select "Create a new key pair" and type in a desired name for the key. In this example we will use `MyEC2Key`. Click on the button labeled Download Key Pair to initiate download of a file named `MyEC2Key.pem`. Save this file somewhere where you can get access to it later. Now click the button labeled Launch Instance.

You will be taken to a page that reports that the instance is launching and asks you to click on a link to monitor progress. Go ahead and do so. You will be taken to the Instances Summary (Fig. A1.C.10), where you can monitor the status of the newly launched instance. When the instance's Instance State column changes from "pending" to "running," your instance is ready for use.

To log into the instance, you will use Secure Shell in conjunction with the key pair saved earlier under the name `MyEC2Key.pem`. From a Linux or Mac OS X system, you will use the built-in command line utility "ssh." From a Windows system, you will use the BitVise SSH client or PuTTY. These tools are described in more detail in the section Logging in Remotely, above.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
First EC2 VM	i-a58e898a	t2.micro	us-east-1b	running	2/2 checks...	None	ec2-52-7-0-191.comput...

Instance: i-a58e898a (First EC2 VM) Public DNS: ec2-52-7-0-191.compute-1.amazonaws.com

Instance ID	i-a58e898a	Public DNS	ec2-52-7-0-191.compute-1.amazonaws.com
Instance state	running	Public IP	52.7.0.191
Instance type	t2.micro	Elastic IP	-
Private DNS	ip-172-30-1-27.ec2.internal	Availability zone	us-east-1b
Private IPs	172.30.1.27	Security groups	launch-wizard-5. view rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-d2bd96b7	AMI ID	ubuntu-trusty-14.04-amd64-server-20150325 (ami-d05e75b8)
Subnet ID	subnet-cb9fabf1	Platform	-
Network interfaces	eth0	IAM role	-
Source/dest. check	True	Key pair name	MyEC2Key
ClassicLink	-	Owner	296402249238
EBS-optimized	False	Launch time	May 10, 2015 at 9:38:42 PM UTC-4 (less than one hour)

Figure A1.C.10 The Amazon console's EC2 "Instances" page summarizes the state and characteristics of each of your active or inactive VMs.

First, look up the IP address of the instance. You'll find it listed on the Instances screen under the "Public IP" column, as well as in the Description tab at the bottom of the screen. On a Macintosh or Linux system, bring up the Terminal application and type the following command:

```
ssh -i MyEC2Key.pem ubuntu@xx.xx.xx.xx
```

Replace `MyEC2Key.pem` with the correct name and directory path of your key pair file. Replace `xx.xx.xx.xx` with the public IP address of your instance. If all goes well, you will be logged into the remote system as user `ubuntu` and see a new command-line prompt. See Using the Command Shell for information on what to do next.

If SSH complains that the file access permissions of `MyEC2Key.pem` are too relaxed, you can fix the issue on OS X and Linux systems by issuing the following command and then attempting the SSH connection again:

```
chmod 0400 MyEC2Key.pem
```

On Windows systems, it is easiest to use the BitVise application. Start by importing the SSH key pair into BitVise by selecting the "Client key manager" option, and then clicking the Import button. Use the file selector to choose the EC2 key file. The newly-imported key will be named "Global 1" if this is the first time you have imported a key, or "Profile X" for subsequently imported keys. Then, in the main application window, enter the instance's public IP address into the field labeled Host, enter `ubuntu` into the field labeled Username, select "publickey" in the pop-up menu labeled Initial Method, and select the appropriate key from the menu labeled "Client key." Now, press the Login button to start a remote command-line session.

You may wish to save the BitVise profile to make it easier to connect in the future.

A running instance will cost you some money for every hour or fraction of an hour it is running, with the exception of `t2.micro` instances running in the free tier. To stop an instance, right click on it in the Instances screen of the Amazon Console and select Instance State > Stop. This places the VM into a power-off mode that preserves any data you may have placed on the instance or any customizations that you have implemented. Once stopped, you can start an instance up again by selecting Instance State > Start, and the system will boot up. To permanently delete an instance, destroying any data you have stored on it, you can select Instance State > Terminate. Be aware that every time you restart an instance, its public IP address changes and you will need to alter the SSH destination host appropriately. Fortunately, Amazon provides a way of associating a fixed IP address to an instance.

For more information on the many things you can do with EC2 instances, please see "Getting Started with AWS," located at *https://aws.amazon.com/documentation/gettingstarted/*.

CONCLUSION

Unix will feel alien and intimidating at first. Do not be inhibited, but feel free to explore and experiment with the Unix environment. With experience, you may eventually come to tolerate, if not appreciate, Unix's alternative take on the world.

LITERATURE CITED

Ladunga, I. 2009. Finding similar nucleotide sequences using network BLAST searches. *Curr. Protoc.Bioinform.* 26:3.3.1-3.3.26.

Stein, L.D. 2007. X Window System Survival Guide. *Curr. Protoc. Bioinform.* 17:A1D.1-A.1D.11.

KEY REFERENCES

Bradnam, K. and Korf, I. 2012. UNIX and Perl to the Rescue!: A Field Guide for the Life Sciences (and Other Data-rich Pursuits). Cambridge University Press, Cambridge, U.K.

Great introduction to the Unix command line and key tools, oriented toward the life sciences.

Frisch, A. 2002. Essential System Administration, 3rd Edition. O'Reilly and Associates, Sebastopol, Calif.

A more advanced guide to working on Unix systems. Although aimed at system administrators, it is highly recommended for newcomers to the Unix environment.

Helmke M. 2014. Ubuntu Unleashed, 2014 Edition. O'Reilly and Associates, Sebastopol, Calif.

A comprehensive introduction to the most popular Linux distribution.

Newham, C. 2005. Learning the bash Shell: Unix Shell Programming. O'Reilly and Associates, Sebastopol, Calif.

Learn the ins and outs of the shell itself.

Van Vliet, J. 2011. Programming Amazon EC2. O'Reilly and Associates, Sebastopol, Calif.

A good introduction to managing virtual machines on EC2.

INTERNET RESOURCES

Remote login packages for Windows:

http://www.bitvise.com

BitVise Web site. A polished terminal emulator for Windows systems using Secure Shell protocol (free for personal use).

https://cyberduck.io/

Cyberduck file transfer utility for Mac OS X and Windows. Supports file transfers over SSH (SFTP protocol).

http://www.chiark.greenend.org.uk/~sgtatham/putty

PuTTY Web site. A simple terminal emulator for Windows systems using Secure Shell protocol (free).

Linux distributions:

http://www.linuxmint.com

Linux Mint, an easy-to-use and attractive version of Linux suitable for first-timers (based on Ubuntu).

http://www.ubuntu.com

Ubuntu, the most popular free Linux distribution, but burdened by an unfamiliar graphical user interface.

http://www.redhat.com

RedHat, the most popular commercial Linux distribution.

http://www.centos.com

A free version of RedHat.