CHAPTER 6, LAB 1: INTRODUCTION TO THE vim EDITOR (20 MINUTES)

LEARNING OBJECTIVES AND OUTCOMES

In this lab you will learn to use some basic commands of the vim editor.

READING

Read "Tutorial: Using vim to Create and Edit a File" on Sobell, pages 161–168.

PROCEDURE

Each Linux distribution includes several text editors. The default graphical editor with the GNOME desktop is gedit and with KDE is kedit. Both are similar in features to Notepad or WordPad. If the graphical environment is not available, you can use nano, a simple editor that is similar to DOSEdit. Advanced editing, including cut and paste, search and replace, and applying filters in the text environment, can be performed using vim or emacs.

The vim editor is large and powerful; you can keep learning things about vim for years. "Chapter 3, Lab 1: Introducing a Few Utilities" on page 8 of this manual introduced vim. This lab builds on that introduction, showing you how to edit a file.

1. Start vim (step 1 on page 8 of this manual and Sobell, page 161) to create a new file named **pizza**.
2. Type **i** (lowercase **i** for *input;* Sobell, pages 164 and 167) to put vim in Input mode (Sobell, page 163) and enter the following text, pressing RETURN at the end of each line. Ignore typing mistakes you make for now. Press ESCAPE when you are done typing to put vim back in Command mode (Sobell, page 163).

```
Pizza is an oven-baked, flat, round bread
typically topped with a tomato sauce, cheese
and various toppings. Pizza was originally
invented in Naples, Italy, and the dish has
since become popular in many parts of the world.
(from Wikipedia)
```

3. Use the ARROW keys to move the cursor so that it is over the **o** in **originally**. Press the **x** (delete character; Sobell, page 166) key ten times to delete each of the letters in **originally**. The editor remains in Command mode throughout this step.
4. Search for the word **Italy** by first pressing the **/** (forward slash; Sobell, page 184) key. Pressing this key puts vim in Last Line mode (Sobell, page 164); the cursor moves to the bottom line of the screen. Now type **Italy** and press RETURN to search for the word **Italy**. Delete the word **Italy** by giving

This text is associated with Sobell/A Practical Guide to Linux® Commands, Editors, and Shell Programming, Third Edition ISBN-10: 0-13-308504-X (ISBN-13: 978-0-13-308504-4)]. Copyright 2013 by Mark G. Sobell. Do not redistribute.

the command **dw** (delete word; Sobell, page 166). The comma and the following SPACE remain. You could remove them using the **x** command, but this step introduces a different command.

Before you can try removing **Italy** another way, you must restore the word you just deleted. Type **u** (undo; Sobell, page 167) to undo the last command you gave; **Italy** reappears. Whereas **dw** deletes a word but not adjacent punctuation, **dW** deletes the word including adjacent punctuation. Give the command **dW**.

5. The **?** (question mark; Sobell, page 184) searches backward for a string of characters the same way the **/** searches forward. Search backward for the word **topped** by typing **?topped** followed by RETURN. The cursor is now on the **t** in **topped**.

6. The **cw** (change word; Sobell, page 181) command removes a word and puts vim in Input mode so you can type a word to replace the original one. You must press ESCAPE to return vim to Command mode when you are finished typing the new word. With the cursor on the first letter of **topped**, give a **cw** command and type the word **covered** followed by an ESCAPE.

7. The **o** (open; Sobell, page 167) command opens a blank line below the line the cursor is on, moves the cursor to the new line, and puts vim in Input mode. The **O** command works the same way except it opens a line above the one the cursor is on. With vim in Input mode and the cursor on a blank line, you can enter as many lines of text as you like. When you are done, press ESCAPE to return vim to Command mode.

   Give an **H** (home; Sobell, page 177) command to move the cursor to the first letter of the document you are editing. Give an **O** (uppercase "oh") command to open a line above the document and type the title **PIZZA** followed by a RETURN and an ESCAPE.

8. Save your work and exit from vim by giving the command **ZZ** (Sobell, page 168). If the **ZZ** appears in the document, vim is in Input mode. Press ESCAPE to put vim back in Command mode, use the **x** command and ARROW keys to remove the **ZZ**, and give another **ZZ** command.

Try other commands from Chapter 6 as you experiment with the **pizza** file and create other files to work with. The vim help system displays vim documentation while you are using vim; see Sobell, page 165.

DELIVERABLES

This lab gives you practice using the vim editor to create and modify files.

This text is associated with Sobell/A Practical Guide to Linux® Commands, Editors, and Shell Programming, Third Edition
ISBN-10: 0-13-308504-X (ISBN-13: 978-0-13-308504-4)]. Copyright 2013 by Mark G. Sobell. Do not redistribute.

CHAPTER 8, LAB 1: WRITING AND EXECUTING A SHELL SCRIPT (20 MINUTES)

LEARNING OBJECTIVES AND OUTCOMES

In this lab you will learn to write and execute a shell script that includes comments. You will use chmod to make the file that holds the script executable and include a line that starts with **#!** in the script to make sure bash executes it. This lab also provides an introduction to positional parameters.

READING

Read "Writing and Executing a Simple Shell Script" on pages 284–289 of Sobell.

PROCEDURE

1. Use vim or cat (see page 16 of this lab manual for instructions) to create a file named **short** with the following line in it:

```
echo 'hi there'
```

2. Use cat to verify the contents of **short** and then try to execute it. Use **ls –l** to display the permissions for **short**. Read the tip on Sobell, page 286.
3. Use chmod (Sobell, pages 99 and 285) to make the file executable, display the permissions for **short**, and try executing the file again.
4. Add a line that starts with **#!** (Sobell, page 287) to the beginning of **short** to make sure it is executed by bash.
5. Add a comment line (Sobell, page 288) to **short** that explains what the script does.
6. Within a shell script, the shell expands **$1** (a variable called a *positional parameter;* Sobell, page 462) to the first argument on the command line the script was called with. Write and execute a script named **first** that displays (sends to standard output) the first argument on the command line it was called with. Include the **#!** line and a comment. Remember to make the file executable.
7. Write a shell script that copies the file named by its first argument to a file with the same name with the filename extension of **.bak**. Thus, if you call the script with the argument **first** (and a file named **first** exists in the working directory), after the script runs you would have two files: **first** and **first.bak**. Demonstrate that the script works properly.
8. Read the caution titled "Always quote positional parameters" on page 462 of Sobell. Use touch to create a file whose name has a SPACE in it. What hap-

pens when you give that filename as an argument to **cptobak** from the previous step?

Modify the **cptobak** script from the previous step by quoting the positional parameters in the cp line. Now what happens when you use the script to make a copy of a file with a SPACE in its name?

DELIVERABLES

This lab gives you practice writing and executing shell scripts.

CHAPTER 8, LAB 2: SHELL PARAMETERS AND VARIABLES (15 MINUTES)

LEARNING OBJECTIVES AND OUTCOMES

In this lab you will learn about user-created variables and keyword variables.

READING

Read "Parameters and Variables" on page 300 of Sobell up to "Pathname expansion in assignments" on page 303.

PROCEDURE

Although variables are mostly used in scripts and read by programs, you can experiment with them on the command line.

1. Assign your name to the variable named **myname** and use echo to display the value of **myname** when it is unquoted, quoted using double quotation marks, and quoted using single quotation marks. (Refer to "Parameter substitution" on page 302 of Sobell and "Quoting the $" on page 302 of Sobell.)
2. Use the readonly (Sobell, page 305) builtin to make the **myname** variable you created in the previous step a readonly variable and then assign a new value to it. What happens?
3. What is the value of your **HOME** (Sobell, page 307) keyword variable?

   Demonstrate that the tilde (~; Sobell, page 307) holds the same value as **HOME**. List the contents of your home directory using a tilde.
4. The **PATH** (Sobell, page 308) keyword variable specifies the directories in the order bash should search them when it searches for a script or program you run from the command line. What is the value of your **PATH** variable?

   Append the absolute pathname of the **bin** directory that is a subdirectory of your home directory to the **PATH** variable. What does this change allow you to do more easily?
5. The **PS1** (Sobell, page 309) keyword variable holds the value of your primary shell prompt. Change the value of this variable so that your prompt is simply a **$** followed by a SPACE when you are working as yourself and a **#** followed by a SPACE when you are working with **root** privileges.
6. The date (Sobell, page 62) utility displays the date and time. Write and execute a shell script that displays the date and time, the name of your home directory, and the value of your **PATH** variable.

DELIVERABLES

This lab gives you practice working with user-created variables and the **HOME**, **PATH**, and **PS1** keyword variables as well as practice using the date utility.