A Guide to Unix Using Linux Fourth Edition

Chapter 3 -4

Mastering Editors

Objectives

- Understand the types of editors
- Create and edit files using the vi editor

Objectives

- Explain UNIX and Linux file processing
- Use basic file manipulation commands to create, delete, copy, and move files and directories
- Employ commands to combine, cut, paste, rearrange, and sort information in files
- Create a script file

ASCII Text Files

- **Byte** (binary term): string of eight bits
- A byte can be configured into fixed patterns of bits
  - **ASCII:** American Standard Code for Information Interchange
    - 256 different characters
  - **Unicode**
    - Supports up to 65,536 characters
- **Text files:** contain nothing but printable characters
- **Binary files:** contain nonprintable characters
  - Example: machine instructions

Printing Characters (Punctuation Characters)

Dec	*Octal*	*Hex*	*ASCII*
32	040	20	(Space)
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/

(Decimal Numbers—Print)

Dec	*Octal*	*Hex*	*ASCII*
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9

(Special Characters—Print)

Dec	*Octal*	*Hex*	*ASCII*
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	080	40	@

Printing Characters (Alphabet—Uppercase)

Dec	*Octal*	*Hex*	*ASCII*
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z

Printing Characters (Alphabet—Lowercase)

Dec	*Octal*	*Hex*	*ASCII*
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z

Nonprinting Characters (Abridged)

Control Characters

Dec	*Octal*	*Hex*	*ASCII*
0	000	00	^@ (Null)
7	007	07	Bell
8	010	08	Backspace
9	011	09	Tab
10	012	0A	Line Feed, Newline
11	013	0B	Vertical tab
12	014	0C	Form feed
13	015	0D	Carriage return

Figure 3-1 ASCII characters

Binary Files

- Some things cannot be represented with ASCII codes
- Binary files are used instead
  - Example: graphic files include bit patterns
    - **Bitmap:** made of rows and columns of dots

Executable Program Files

- Text files containing program code are **compiled** into machine-readable language
- Scripts are files containing commands
  - Typically interpreted, not compiled
- **Executables:** compiled and interpreted files that can be run

Using Editors

- **Editor:** program for creating and modifying files containing source code, text, data, memos, etc.
- **Text editor:** a simplified word-processing program
  - Used to create and edit documents
- Two text editors normally included in UNIX/Linux are **screen editors**
  - vi
  - Emacs
- **Line editor:** works with one line (or group of lines) at a time

Using the vi Editor

- vi is a visual editor
- vi is also a modal editor
  - Supports three modes
    - **Insert mode**
      - Accessed by typing “i”
    - **Command mode**
      - Accessed by typing Esc
    - Extended (**ex**) command set mode
      - Accessed by typing “:” in command mode

Creating a New File in the vi Editor

Figure 3-2 Creating a new file in the vi editor

Inserting Text

- When you start vi, you are in command mode
- To insert text in your file, switch to insert mode
  - Use *i* (insert) command
- To return to command mode, press Esc

Repeating a Change

- Use a period (.) to repeat the most recent change you made
  - Repeat command
  - Works in command mode

Moving the Cursor

- To move cursor use arrow keys (command/insert mode) or (in command mode) use:

Table 3-1 vi editor's cursor movement keys

Key	Movement
h or left arrow	Left one character position
l or right arrow	Right one character position
k or up arrow	Up one line
j or down arrow	Down one line
H	Upper-left corner of the screen
L	Last line on the screen
G	Beginning of the last line
nG	The line specified by a number, *n*
W	Forward one word
b	Back one word
0 (zero)	Beginning of the current line
$	End of the current line
Ctrl+u	Up one-half screen
Ctrl+d	Down one-half screen
Ctrl+f or *Page Down*	Forward one screen
Ctrl+b or *Page Up*	Back one screen

Deleting Text

- Deletion commands available (command mode)

Table 3-2 vi editor's delete commands

Command	Purpose
x	Delete the character at the cursor.
dd	Delete the current line (putting it in a buffer so it can also be pasted back into the file).
dw	Delete the word starting at the cursor. If the cursor is in the middle of the word, delete from the cursor to the end of the word.
d$	Delete from the cursor to the end of the line.
d0	Delete from the cursor to the start of the line.

- *dd* is used for "cutting" text
  - Use "yank" (*yy*) command for "copying" text

Undoing a Command

- Type *u* to use the undo command
- Example:
  - If you delete a few lines from a file by mistake, type *u* to restore the text

Searching for a Pattern

- To search forward for a pattern of characters:
  - Type a forward slash (*/*)
  - Type the pattern you are seeking
  - Press Enter

Table 3-3 Special characters used to match a pattern

Special Character*	Purpose
\>	Searches for the next word that ends with a specific string.
\<	Searches for the next word that begins with a specific string.
.	Acts as a wildcard for one character.
[]	Finds the characters between the brackets.
$	Searches for the line that ends with a specific character.
*All of these special characters must be preceded with a slash (/) from the command mode.	

- Examples: */\<top*, */s..n*, */pas[st]*, */!$*

Saving a File and Exiting vi

- To save file without exiting, use *:w*
- To save and exit, use *:wq*, *:x*, *ZZ* (command mode)

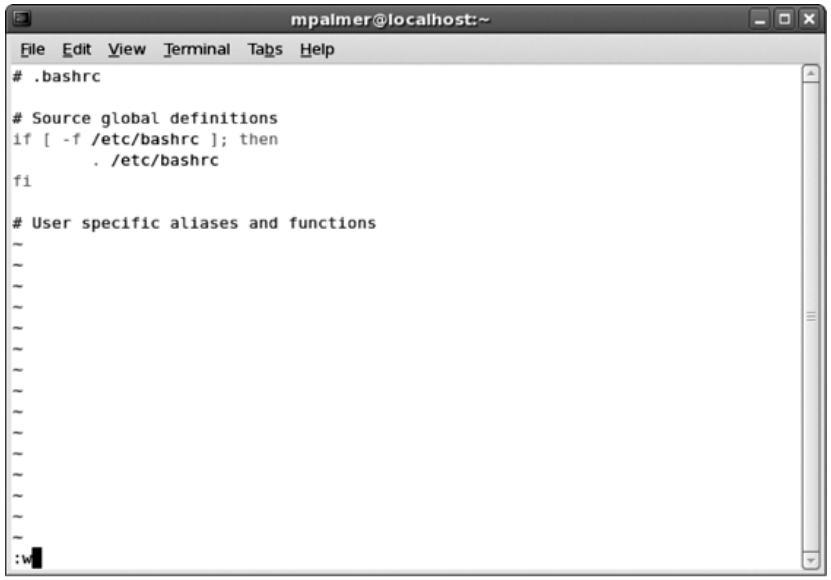

Figure 3-3 Saving without exiting

Adding Text from Another File

- To copy entire contents of one file into another file:
  - Use vi to edit the file you would like to copy into
  - Use the command *:r filename*
    - *filename* is the name of the file that contains the information you want to copy

Leaving vi Temporarily

- To launch a shell or execute other commands from within vi, use *:!*
  - Example:
    - *:!cal*
- To run several command-line commands in a different shell without closing vi session
  - Use *Ctrl+z* to display the command line
  - Type *fg* to go back to vi

Leaving vi Temporarily (continued)

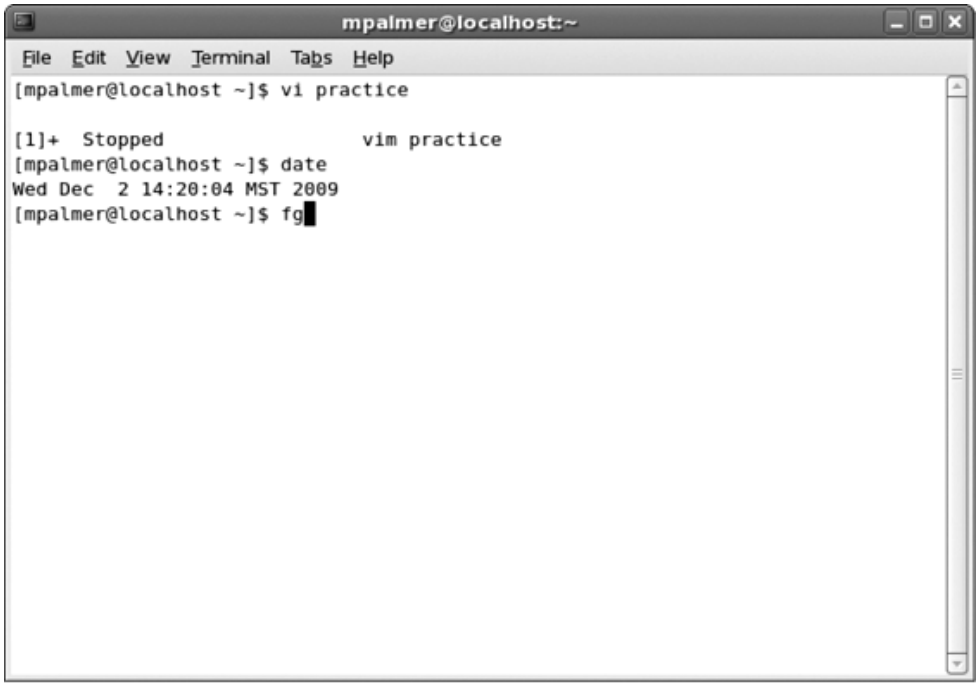

Figure 3-4 Accessing a shell command line from the vi editor

Copying or Cutting and Pasting

- The command *yy* copies (yanks) a specified number of lines
  - To cut the lines, use *dd*
  - Lines are placed in clipboard
- Use *p* to paste the clipboard contents

Canceling an Editing Session

- Canceling an editing session will discard all the changes you have made
- Or, save changes you made since last using *:w*
  - Saves file without exiting vi

Getting Help in vi

- Use the help command
  - *:help*
- Other alternatives:
  - *man vi*
    - From the command line
  - *:!man vi*
    - From vi (command mode)

Using Input and Error Redirection

- Use > and >> to redirect output
  - Example: *ls* > *homedir.list*

Manipulating Files

- Some ways to manipulate files:
  - Create files
  - Delete files
  - Remove directories
  - Copy files
  - Move files
  - Find files
  - Combine files
  - Combine files through pasting
  - Extract fields in files through cutting
  - Sort files

Deleting Files

- Delete a file using the *rm* (remove) command
  - Example: *rm test**

Syntax **rm** [-options] *filename* or *directoryname*

Dissection

- Used to delete files or directories
- Useful options include:

  -*i* displays a warning prompt before deleting the file (or directory)

  -*r* when deleting a directory, recursively deletes its files and subdirectories (to delete a directory that is empty or that contains entries, use the -*r* option with *rm*)

Removing Directories

- Use *rm* or *rmdir* to remove an empty directory

Syntax **rmdir** [-options] *directoryname*

Dissection

- Used to delete directories
- A directory must be empty to delete it with the *rmdir* command.

- Use *rm -r* to remove a non-empty directory

Copying Files

- Use *cp* for copying files

Syntax **cp** [-options] *source destination*

Dissection

- Used to copy files or directories
- Useful options include:

  -*i* provides a warning before *cp* writes over an existing file with the same name

  -*s* creates a symbolic link or name at the destination rather than a physical file (a symbolic name is a pointer to the original file, which you learn about in Chapter 6)

  -*u* prevents *cp* from copying over an existing file if the existing file is newer than the source file

- Examples:

```
cp class_of_88 duplicates/classmates
cp project1 project2 project3 duplicates
cp designs/* duplicates
```

Moving Files

- To move a file, use *mv* (move) along with the source file name and destination name
  - As insurance, a file is copied before it is moved
  - Moving and renaming a file are the same operation

Syntax **mv** [-options] *source destination*

Dissection

- Used to move and to rename files
- Useful options include:

  -*i* displays a warning prompt before overwriting a file with the same name

  -*u* overwrites a destination file with the same name, if the source file is newer than the one in the destination

Combining Files

- You can use *cat* to combine files
- For example:

```
cat janes_research marks_research > total_research
```

Combining Files with the paste Command

Syntax **paste** [-options] *source files* [> *destination file*]

Dissection

- Combines the contents of one or more files to output to the screen or to another file
- By default, the pasted results appear in columns separated by tabs
- Useful options include:

  -*d* enables you to specify a different separator (other than a tab) between columns

  -*s* causes files to be pasted one after the other instead of in parallel

- For example, two files (vegetables and bread):

```
Carrots
Spinach
Lettuce
Beans
```

```
Whole wheat
White bread
Sourdough
Pumpernickel
```

  – Can be pasted using *paste vegetables bread > food*

Combining Files with the paste Command (continued)

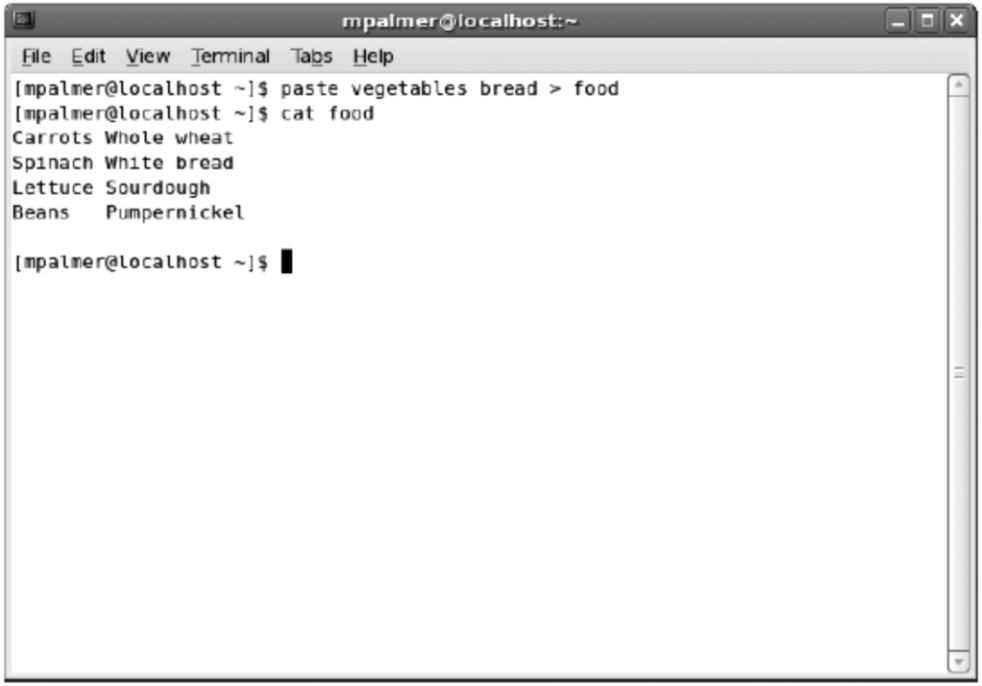

Figure 4-3 Using the *paste* command to merge files

- Another example:

```
paste -d',' vegetables bread > food
```

Extracting Fields Using the cut Command

Syntax **cut [-f** *list* **] [-d char**] [*file1 file2* . . .] or **cut** [**-c** *list*] [*file1 file2* . . .]

Dissection

- Removes specific columns or fields from a file
- Useful options include:

  -f specifies that you are referring to fields

  list is a comma-separated list or a hyphen-separated range of integers that specifies the field. For example, *-f 1* indicates field 1, *-f 1,14* indicates fields 1 and 14, and *-f 1-14* indicates fields 1 through 14.

  -d indicates that a specific character separates the fields

  char is the character used as the field separator (delimiter), for example, a comma. The default field delimiter is the tab character.

  file1, file2 are the files from which you want to cut columns or fields

  -c references character positions. For example, *-c 1* specifies the first character and *-c 1,14* specifies characters 1 and 14.

Extracting Fields Using the cut Command (continued)

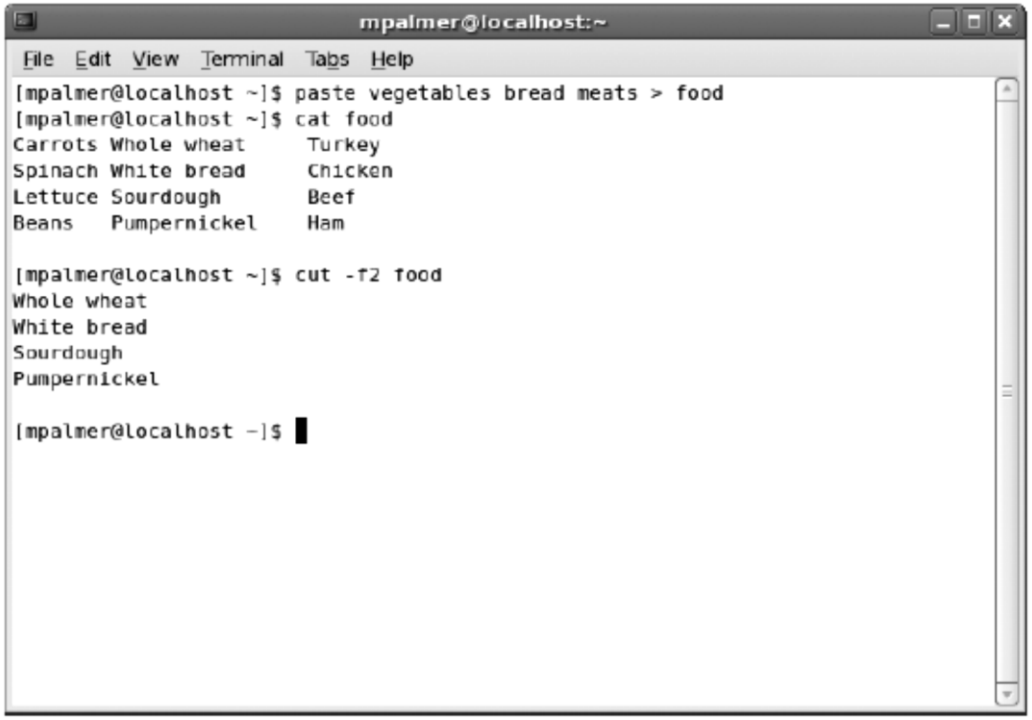

Figure 4-4 Using the *cut* command

Sorting Files

Syntax **sort** [-options] [*filename*]

Dissection

- Sorts the contents of files by individual lines
- Useful options include:

  -*k n* sorts on the key field specified by *n*

  -*t* indicates that a specified character separates the fields

  -*m* merges input files that have been previously sorted (does not perform a sort)

  -*o* redirects output to the specified file

  -*d* sorts in alphanumeric or dictionary order

  -*g* sorts by numeric (general) order

  -*r* sorts in reverse order

- Examples:

```
sort file1 > file2
sort -k 3 food > sortedfood
```

Sorting Files (continued)

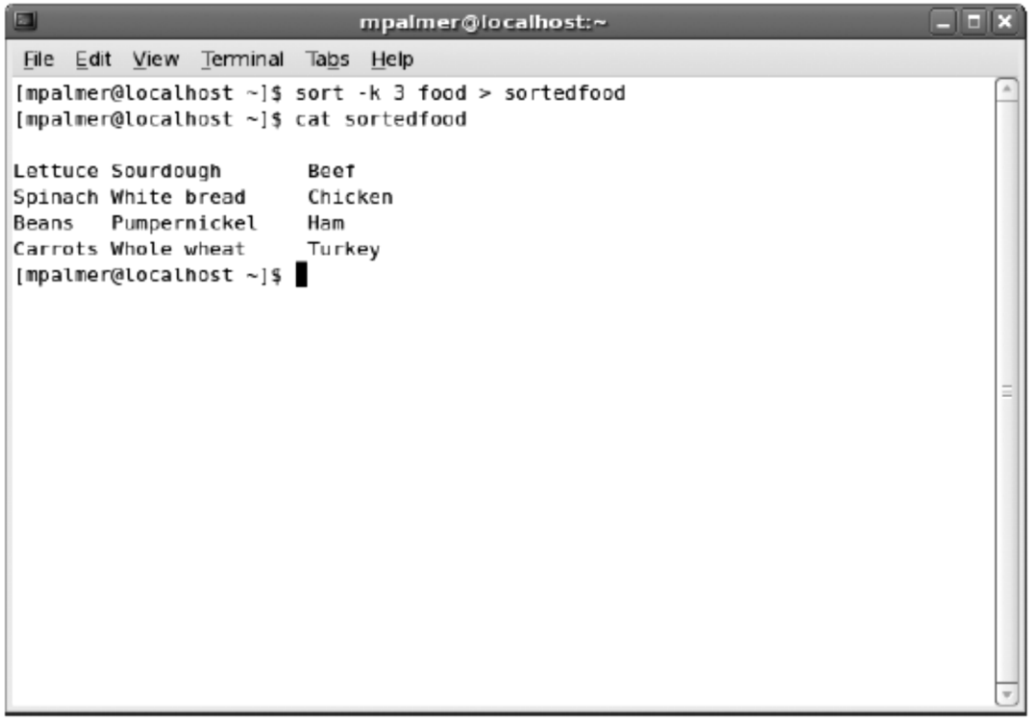

Figure 4-5 Results of sorting on the third field in the food file

Creating Script Files

- To automate tasks, MS-DOS and Windows users create batch files
  - Commands are executed when file is run
- UNIX/Linux users do the same:
  - **Shell script** contains command-line entries
- Steps:
  - Create script using a text editor (e.g., vi, Emacs)
  - Make file executable (use *chmod*)
  - Execute (e.g., *./myscript*)

Creating Script Files (continued)

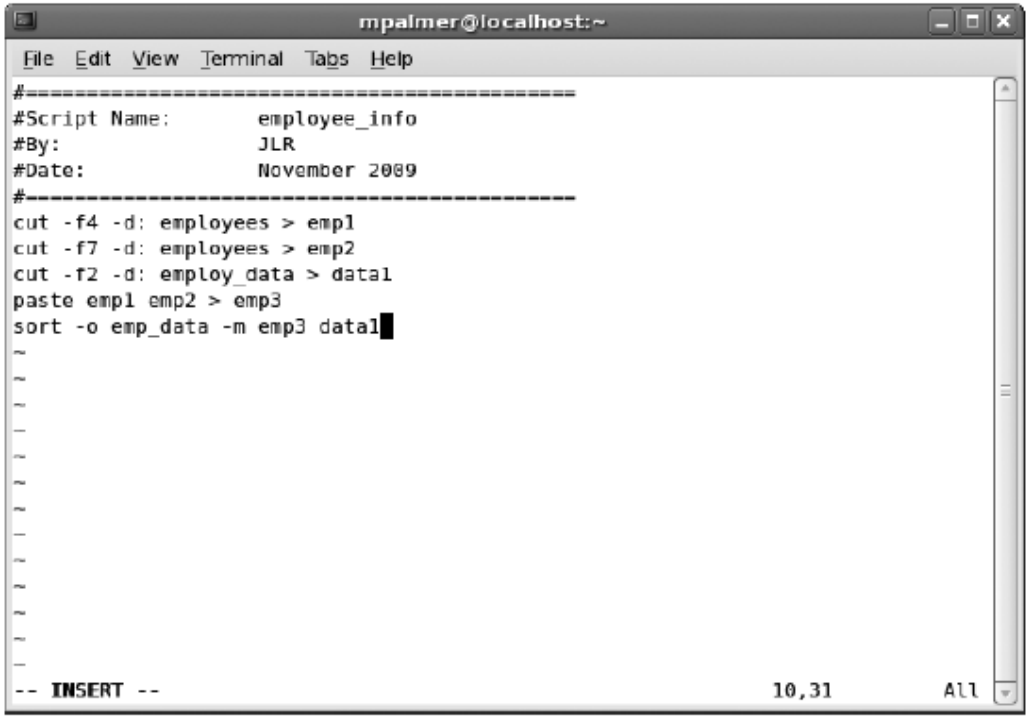

Figure 4-6 Sample script file

Using the join Command on Two Files

- Use *join* to associate lines in two files on the basis of a common field in them
  - Example:

```
Brown:82:53,000
Anders:110:32,000
Caplan:174:41,000
Crow:95:36,000
```

```
Brown:LaVerne:F:Accounting Department:444-7508: . . .
Anders:Carol:M:Sales Department:444-2130: . . .
Caplan:Jason:R:Payroll Department:444-5609: . . .
Crow:Lorretta:L:Shipping Department:444-8901: . . .
```

Files above can be joined to obtain:

```
Brown:LaVerne:Accounting Department:53,000
Anders:Carol:Sales Department:32,000
Caplan:Jason:Payroll Department:41,000
Crow:Lorretta:Shipping Department:36,000
```

Using the join Command on Two Files (continued)

Syntax **join** [-options] *file1 file2*

Dissection

- Used to associate information in two different files on the basis of a common field or key in those files
- *file1*, *file2* are two input files that must be sorted on the join field—the field you want to use to join the files. The join field is also called a key. You must sort the files before you can join them. When you issue the *join* command, UNIX/Linux compare the two fields. Each output line contains the common field followed by a line from file1 and then a line from file2. You can modify output using the options described next. If records with duplicate keys are in the same file, UNIX/Linux join on all of them. You can create output records for unpairable lines, for example, to append data from one file to another without losing records.
- Useful options include:

  -1 fieldnum specifies the common field in file1 on which to join

  -2 fieldnum specifies the common field in file 2 on which to join

  -o specifies a list of fields to output. The list contains blank-separated field specifiers in the form m.n, where *m* is the file number and *n* is the position of the field in the file. Thus, *-o 1.2* means "output the second field in the first file."

  -t specifies the field separator character. By default this is a blank, tab, or new line character. Multiple blanks and tabs count as one field separator.

  -a filenum produces a line for each unpairable line in the file *filenum*. (In this case, *filenum* is a 1 for *file1* or a 2 for *file2*.)

  -e str replaces the empty fields for the unpairable line in the string specified by *str*. The string is usually a code or message to indicate the condition, for example, *-e "No Vendor Record."*

A Brief Introduction to the Awk Program

- Awk: pattern-scanning and processing language
  - Helps to produce reports that look professional
  - Inventors: A. Aho, P. Weinberger, and B. Kernighan

Syntax **awk** [**- Fsep**] [‘*pattern* {action} ..’] *filenames*

Dissection

- *awk* checks to see if the input records in the specified files satisfy the *pattern* and, if they do, *awk* executes the *action* associated with it. If no pattern is specified, the action affects every input record.
- *-F:* means the field separator is a colon

  - Example:

```
awk ’BEGIN { print "This is an awk print line." }’
```

A Brief Introduction to the Awk Program (continued)

- Some of the tasks you can do with *awk* include:
  - Manipulate fields and records in a data file
  - Use variables
  - Use arithmetic, string, and logical operators
  - Execute commands from a shell script
  - Use classic programming logic, such as loops
  - Process/organize data into well-formatted reports
- Another example:

```
awk -F: '{printf "%s\t %s\n", $1, $2}' datafile
```

Command Summary

Command	Purpose
vi commands:	
. (repeat)	Repeat your most recent change.
/	Search forward for a pattern of characters.
:!	Leave vi temporarily.
:q	Cancel an editing session.
:r	Read text from one file and add it to another.
:set	Turn on certain options, such as line numbering.
:w	Save a file and continue working.
:wq	Write changes to disk and exit vi.
:x	Save changes and exit vi.
:!lpr filename	Print a file.
i	Switch to insert mode.
p	Paste text from the buffer.
u	Undo your most recent change.
vi	Start the vi editor.
yy	Copy (yank) text to the clipboard.
ZZ	In command mode, save changes and exit vi.
Ctrl+z	Use this shell-based command (not truly a vi command) to leave vi to temporarily access the command line—use the **fg** command to return to vi.

Summary

- UNIX/Linux support regular files, directories, character special files, and block special files
  - Three kinds of regular files: unstructured ASCII characters, records, and trees
  - Often, flat ASCII data files contain records and fields
  - Standard devices: stdin, stdout, and stderr
- *touch* updates a file's time/date stamp
  - Also used to create empty files
- *rmdir* removes an empty directory
  - Use *rm –r to* remove non-empty directories

Summary (continued)

- *cut* extracts specific columns or fields from a file
- *paste*: combines two or more files
- *sort*: sorts a file's contents
- Create shell scripts to automate tasks
- *join*: extracts information from two files sharing a common field
- Awk is a pattern-scanning and processing language
  - Creates a formatted report with a professional look

Command Summary

Command	Purpose	Options Covered in This Chapter
awk	Starts the *awk* program to format output	**-F** identifies the field separator. **-f** indicates code is coming from a disk file, not the keyboard.
cat	Views the contents of a file, creates a file, merges the contents of files	
cp	Copies one or more files	**-i** provides a warning before *cp* writes over an existing file with the same name. **-s** creates a symbolic link or name at the destination rather than a physical file. **-u** prevents *cp* from copying over an existing file, if the existing file is newer than the source file.
cut	Extracts specified columns or fields from a file	**-c** refers to character positions. **-d** indicates that a specified character separates the fields. **-f** refers to fields.
find	Finds files	**-iname** specifies the name of the files you want to locate, but the search is not case sensitive. **-name** specifies the name of the files you want to locate, but the search is case sensitive. **-mmin *n*** displays files that have been changed within the last *n* minutes. **-mtime *n*** displays files that have been changed within the last *n* days. ***-size n*** displays files of size *n*.

Command	Purpose	Options Covered in This Chapter
join	Combines files having a common field	**-a** ***n*** produces a line for each unpairable line in file ***n***. **-e** ***str*** replaces the empty fields for an unpairable file with the specified string. **-1** and **-2** with the field number are used to specify common fields when joining. **-o** outputs a specified list of fields. **-t** indicates that a specified character separates the fields.
mv	Moves one or more files	**-i** displays a warning prompt before overwriting a file with the same name. **-u** overwrites a destination file with the same name, if the source file is newer than the one in the destination.
paste	Combines fields from two or more files	**-d** enables you to specify a different separator (other than a tab) between columns. **-s** causes files to be pasted one after the other instead of in parallel.
rm	Removes one or more files	**-i** specifies that UNIX/Linux should request confirmation of file deletion before removing the files. **-r** specifies that directories should be recursively removed.
rmdir	Removes an empty directory	
sort	Sorts the file's contents	**-k** ***n*** sorts on the key field specified by ***n***. **-t** indicates that a specified character separates the fields. **-m** means to merge files before sorting. **-o** redirects output to the specified file. **-d** sorts in alphanumeric or dictionary order. **-g** sorts by numeric (general) order. **-r** sorts in reverse order.
touch	Updates an existing file's time stamp and date stamp or creates empty new files	**-a** specifies that only the access date and time are to be updated. **-m** specifies that only the modification date and time are to be updated. **-c** specifies that no files are to be created.

Summary

- Bytes: computer characters (a series of bits) stored using numeric codes
- The vi editor is popular among UNIX/Linux users
  - Three modes: insert (*i*), command (Esc), and ex (Esc :)
  - With vi, you edit a copy of the file placed in memory
    - File is not altered until you save it on disk