

CREATE AND WRITE TO A FILE

open FILEHANDLE, FILENAME – Open a file.
close FILEHANDLE – Close a file.

It is often necessary to access files from within a Perl script. This allows you to perform tasks such as reading and manipulating data in a file.

You use the `open` function to create or open a file. You must assign a filehandle to a file you create or open. Once the file is open, you can use the filehandle to work with the file. A filehandle usually appears in uppercase letters to help distinguish the filehandle from the rest of your code.

You must specify the name of the file you want to create or open, which can include a path to the file.

If you do not specify a path, Perl will place the file in the current directory.

If you want to write data to a file, use the `>` prefix in the file name. Using the `>` prefix to open a file you have previously created allows you to overwrite the existing file with new data.

After you create or open a file, you can write data to the file. The `print` function is commonly used to write data to a file from a Perl script.

When you finish writing data to a file, use the `close` function to close the file.

Extra

When writing to a file, UNIX and DOS-compatible systems indicate newlines differently. This can cause incorrect formatting in files. UNIX systems always represent newlines with the `\n` character. In text mode, DOS-compatible systems indicate newlines by using the carriage return and newline characters, `\r\n`. To have DOS-compatible systems use binary mode so they do not indicate newlines with two characters, use the `binmode` function after the `open` function in your code.

Example:

```
open(IN, "document.pdf");
binmode IN;
while (read IN, $data, 1;)
{
    print OUT $data;
}
close IN;
```

The `>>`, `<` and `+<` prefixes are also used when working with files. If you want to append data to an existing file, use the `>>` prefix in the file name. Use the `<` prefix if you want to read data from a file. To be able to both read and write data in a file, use the `+<` prefix.

TYPE THIS:

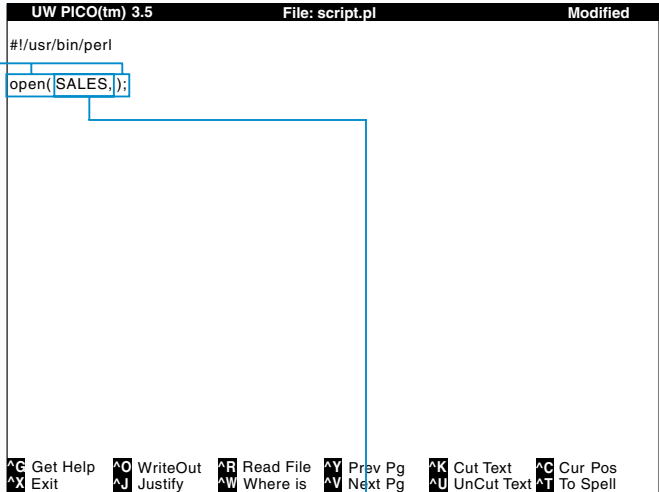
```
open(SALES, ">>salesdata.txt");
print SALES "Rodrigues, Sandy          \\\n";
close SALES;
```

RESULT:

Sales for July:

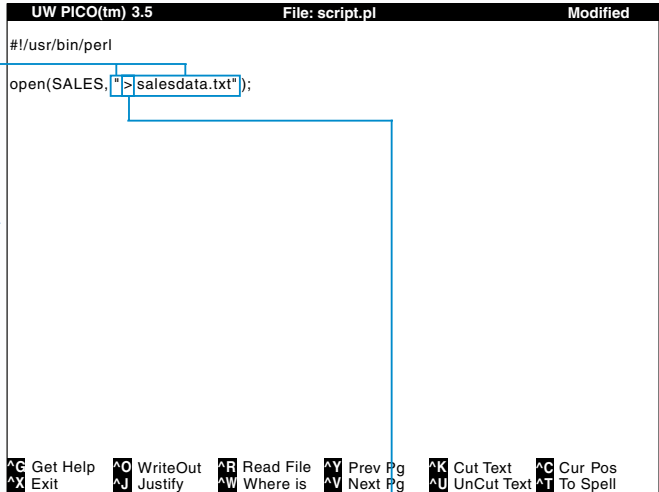
Edwards, Martine	\$955.39
Corder, Mary	\$1235.99
O'Brien, Pat	\$1015.26
Whitney, April	\$1000.50
Rodrigues, Sandy	\$1110.90

CREATE AND WRITE TO A FILE



1 To create or open a file, type `open();`.

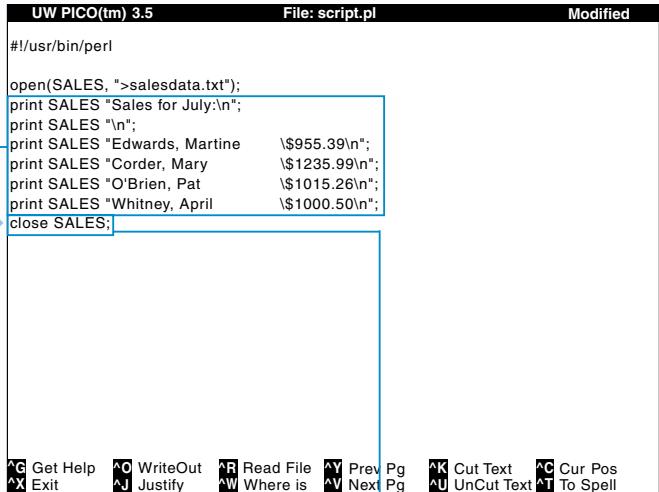
2 Position the cursor over the closing parenthesis and type a filehandle for the file followed by a comma (,).



3 Type the name of the file you want to create or open, enclosed in quotation marks.

Note: You can also include the path for the file in the file name.

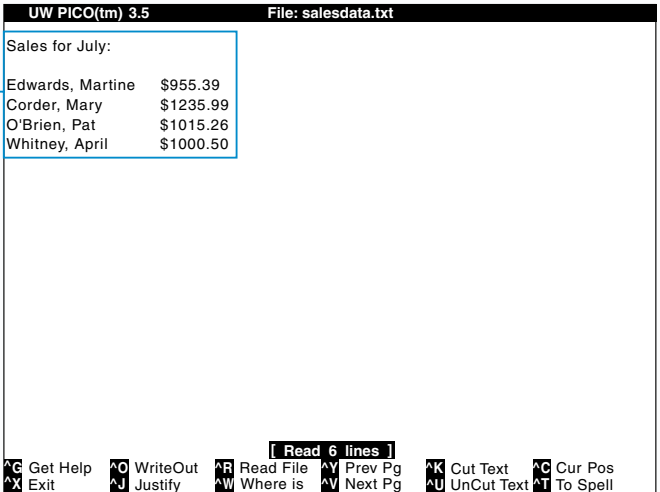
4 To be able to write to the file, position the cursor over the first letter of the file name and type `>`.



5 Position the cursor where you want to type the code that will write data to the file and type the code.

6 To close the file, type `close` followed by the filehandle for the file. Then type a semi-colon (;).

7 Save and execute the script.



You can use your text editor to open the file you created and view its contents.

READ A FILE

An open file can be read using the filehandle for the file combined with the input, or angle, operator <>. Reading data from a file into a Perl script allows you to manipulate the data in the script.

When reading a file, a while loop is often used to process each line of data in the file. Assigning a variable to the filehandle allows you to temporarily store each line of data in the variable as it is read. With each iteration of the while loop, the next line of data is assigned to the variable until each line of data has been

read. By default, Perl uses the newline character to determine where a line of data ends.

When reading a file, you cannot write data to the file. To write data to a file you have read, you must first close the file and then re-open it using either the > or >> prefix with the file name. For more information, see page 138.

When you have finished reading the necessary data from a file, you should close the file using the close function. See page 138 for information about the close function.

Extra

If the functions in the while loop you are using automatically use the default special variable, \$_, you do not need to create a variable to store each line of data in the file. The while loop will automatically assign each line of data to the default special variable. This can help you save time when typing your code.

```
TYPE THIS:
open(PICLIST, "picture_list.txt");
while (<PICLIST>)
{
    print;
}
close PICLIST;
```

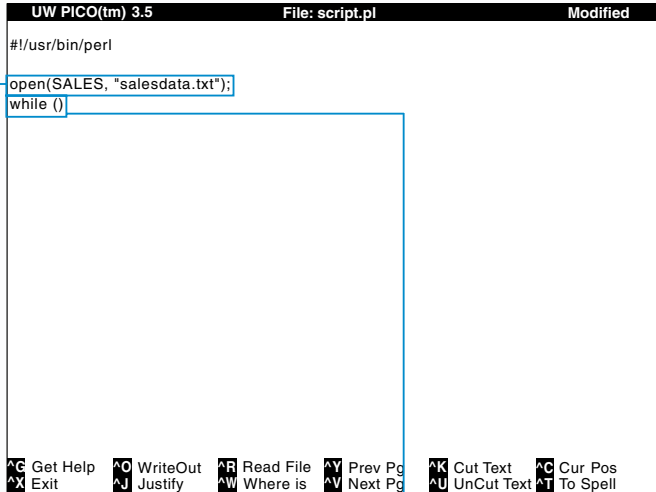
RESULT:
brochureimage.jpg
products.jpg
logo.gif

By default, if Perl cannot open a file you want to read, the open function will return an undefined value and the script will continue to run. If the file is essential to the successful operation of your script, you may want to use the die function to stop processing the script if the file cannot be opened.

```
TYPE THIS:
open(SAVEDDAT, "savedinfo.dat")
or die "Cannot open savedinfo.dat";
while ($line = <SAVEDDAT>)
{
    print $line;
}
close SAVEDDAT;
```

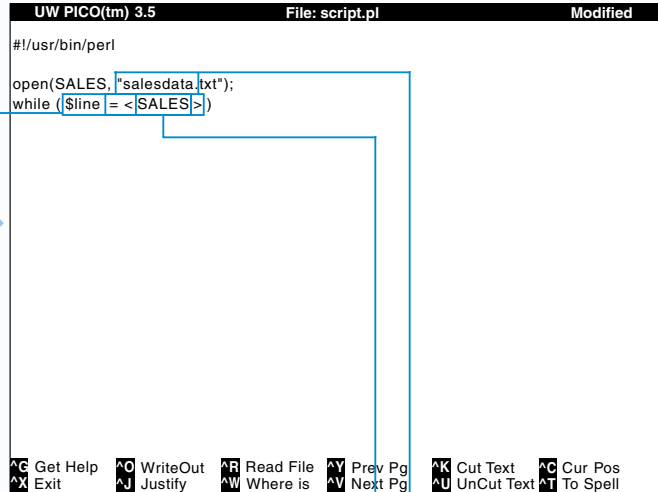
RESULT:
Cannot open savedinfo.dat
at ./script.pl line 3.

READ A FILE



1 Perform steps 1 to 3 on page 138 to type the code that opens a file and assigns it a filehandle.

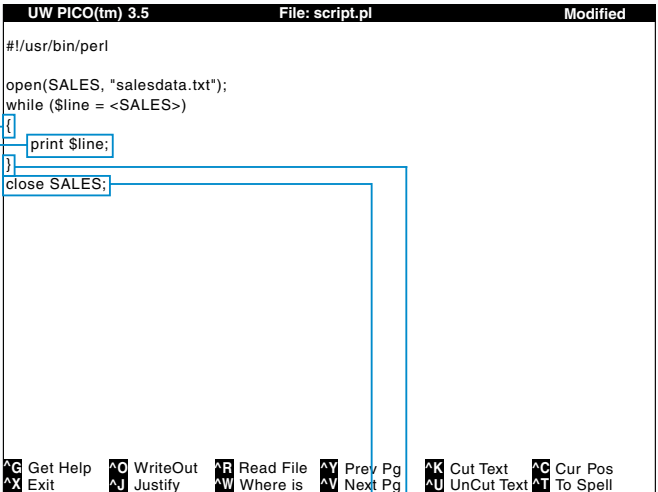
2 Type while ().



3 Position the cursor over the closing parenthesis and type a name for the variable that will store each line of data in the file.

4 Type = <>.

5 Position the cursor over the closing angle bracket (>) and type the filehandle for the file.

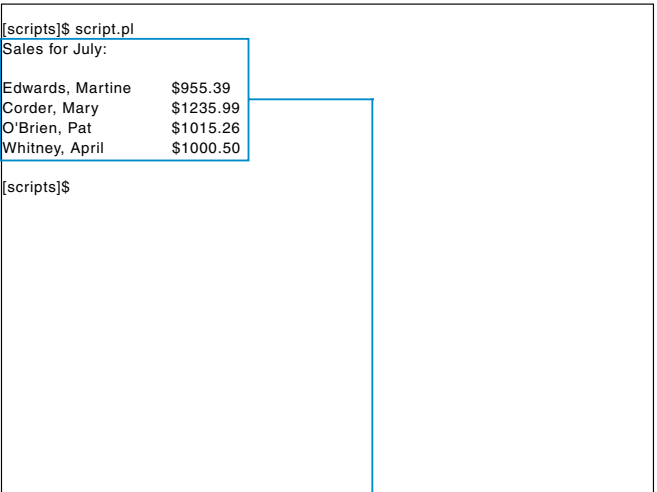


6 Position the cursor directly below the while statement and type {. Then press Enter.

7 Press Tab and type the code that will use each line of data in the file. Then press Enter.

8 Type } and then press Enter.

9 Type the code that closes the file.



10 Save and execute the script.

Perl generates the result of reading a file using the input operator.

READ CHARACTERS FROM A FILE

`read FILEHANDLE, VARIABLE, LENGTH, offset` – Read characters from a file.

The `read` function allows you to read a specific number of characters from a file, beginning at the current position in the file. A file must be assigned a filehandle before it can be read using the `read` function. The `read` function is useful for reading from files that store information in specific sizes, such as some database files.

To use the `read` function, you must specify the filehandle of the file you want to read, a variable that will store the data Perl reads from the file and the number of characters you want to read.

Using a `while` loop with the `read` function allows you to read an entire file, a specific number of characters at a time.

The `read` function returns the value of the number of characters read from a file. If you want to access the value returned by the `read` function, assign the function to a variable.

When you have finished reading information from a file, you should close the file using the `close` function. See page 138 for information about the `close` function.

Extra

By default, each time the `read` function stores information in its variable, any information previously stored in the variable is overwritten. To have Perl store multiple values read from a file in the variable, specify an offset value for the `read` function. The offset value specifies where you want to store a piece of data in the variable. For example, to have Perl store new data five characters in, specify an offset value of 5. In this example, the offset value is incremented by five in each iteration of the loop.

TYPE THIS:

```
open(CUSTOMERS, ">customers.txt");
print CUSTOMERS "Lindsay, Maureen, Mark, David";
close CUSTOMERS;

open(CUSTOMERS, "customers.txt");
while (read(CUSTOMERS, $text, 5, $offset += 5))
{
    print $text, "\n";
}
close CUSTOMERS;
```

RESULT:

Linds
Lindsay, M
Lindsay, Mauree
Lindsay, Maureen, Ma
Lindsay, Maureen, Mark, D
Lindsay, Maureen, Mark, David

READ CHARACTERS FROM A FILE

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

open(EMPLOYEES, "employeedata.txt");
while (read())
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

1 Perform steps 1 to 3 on page 138 to type the code that opens a file and assigns it a filehandle.

2 Type `while (read()`.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

open(EMPLOYEES, "employeedata.txt");
while (read(EMPLOYEES, $text, 10))
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

3 Position the cursor over the first closing parenthesis and type the filehandle followed by a comma (,).

4 Type a name for the variable that will store the data from the file. Then type a comma (,).

5 Type the number of characters you want to read from the file.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

open(EMPLOYEES, "employeedata.txt");
while (read(EMPLOYEES, $text, 10))
{
    print $text, "\n";
}
close EMPLOYEES;
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

6 Position the cursor directly below the `while` statement and type `{`. Then press Enter.

7 Press Tab and type the code that will use the data read from the file. Then press Enter.

8 Type `}` and then press Enter.

9 Type the code that closes the file.

[scripts]\$ script.pl

Martine Edwards
Rev Mengle
Jill Johnson
Kyle Loope
r David Gregory
Pat O'Brien
Tim Borek

[scripts]\$

10 Save and execute the script.

Perl generates the result of reading information from a file using the `read` function. In this example, each employee name in the list is stored in a space 10 characters wide.

CHANGE POSITION IN A FILE

seek FILEHANDLE, POSITION, STARTPOSITION – Change the position in a file.

The seek function allows you to move to a specific position in a file. This is useful if you want to begin reading information in the middle of a line. The seek function is often used to access specific information from files that are divided into records of the same size, such as databases.

To use the seek function, you must open the file you want to use and specify the filehandle of the file. You must then specify the number of bytes you want to move to set the new position in the file. One byte represents one

character, with the first byte in a file being 0. To move backwards through a file, specify a negative number.

You must also specify where in the file you want Perl to begin counting bytes to set the new position. You can have Perl begin counting at the beginning of the file, the current position in the file or the end of the file.

If the seek function successfully moves to the position you specify, Perl returns a value of 1. If the function fails, Perl returns a value of 0.

Extra You use specific values to specify where you want Perl to begin counting bytes to set the new position in a file. The chart below displays the values you can use, as well as a description and an example for each value.

VALUE:	DESCRIPTION:	EXAMPLE:
0	Determines the new position by counting bytes from the start of the file. You cannot specify a negative number for the position when this value is used.	seek TEXT, 5, 0 Moves the position to the sixth character from the start of the file.
1	Determines the new position by counting bytes from the current position in the file.	seek TEXT, -4, 1 Moves the position to the second character from the start of the file if the current position is 5.
2	Determines the new position by counting bytes from the end of the file.	seek TEXT, -5, 2 Moves the position to the fifth character from the end of the file.

CHANGE POSITION IN A FILE

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
open(SALES, "salesdata.txt");
seek SALES,;
```

- 1 Position the cursor where you want to change the position in the file and type **seek ;**.
 - 2 Position the cursor over the semi-colon and type the filehandle followed by a comma (,).
- To open a file and assign it a filehandle, see page 138.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
open(SALES, "salesdata.txt");
seek SALES, 57, 0;
```

- 3 Type the number of bytes you want to move to set the new position in the file. Then type a comma (,).
- 4 Type the value that specifies whether you want Perl to count the bytes from the beginning, end or current position in the file.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
open(SALES, "salesdata.txt");
seek SALES, 57, 0;
while ($line = <SALES>)
{
    print $line;
}
close SALES;
```

- 5 Type the code that reads data from the file. For information about reading data from a file, see page 140.
- 6 Type the code that closes the file. For information about closing a file, see page 138.

```
[scripts]$ script.pl
Mary           $1235.99
O'Brien, Pat  $1015.26
Whitney, April $1000.50
[scripts]$
```

- 7 Save and execute the script.
- Perl generates the result of using the seek function to change the position in a file.
- In this example, the script prints the data starting from the 58th character in the file.

DETERMINE THE POSITION IN A FILE

tell FILEHANDLE – Determine the current position in a file.
getc FILEHANDLE – Read a character from a file.

The **tell** function allows you to determine the current position in an open file. This is useful when you have been working with a file and need to know where the next **read** operation you perform will begin.

The **tell** function returns the current position as the number of characters from the beginning of the file. Nonprintable characters such as tab and newline are counted when determining the position.

After determining the current position in a file, you can use the **getc** function to read a single character from the current position.

Using the **getc** function in a loop allows you to read several characters from a file. This is useful when you want to extract a small amount of data from a file. For reading larger amounts of data, the **read** function is more efficient. For information about the **read** function, see page 142.

The **tell** and **getc** functions both use the filehandle of a file as their argument, though the filehandle is optional for the **tell** function. If you do not specify a filehandle for the **tell** function, the function will use the last file read in the script.

Apply It

The **tell** function is often used in conjunction with the **seek** function, which changes the current position in a file. If you plan to later return to a specific position in your file, use the **tell** function to store the position in a variable. To return to the position, use the variable as the position argument in the **seek** function.

Example:

```
open (ADDRESS, "addresses.txt");
read (ADDRESS, $nameinfo, 15);
$savedPosition = tell ADDRESS;
read (ADDRESS, $streetaddress, 20);
seek ADDRESS, $savedPosition, 0;
```

The **getc** function can also be used to read a character entered using a keyboard. This is useful for a script that prompts a user to select an option using a single character. Information entered using a keyboard can be accessed using the **STDIN** filehandle.

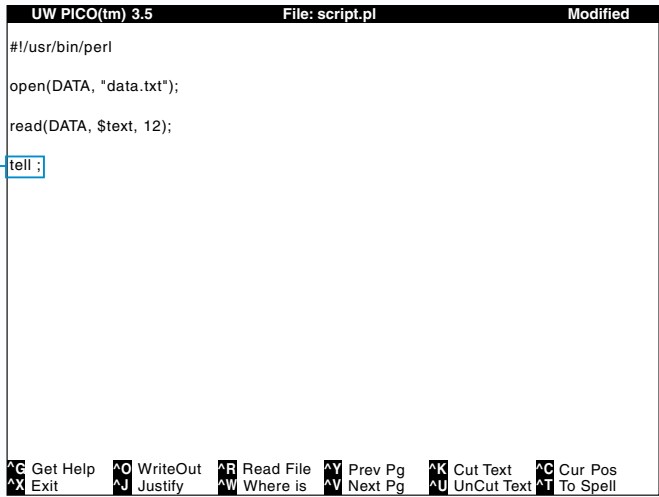
TYPE THIS:

```
until (lc($char) eq "n")
{
    print "\nAre you sure you want to continue? (y/n): ";
    $char = getc STDIN;
    getc STDIN;
}
print "Goodbye!";
```

RESULT:

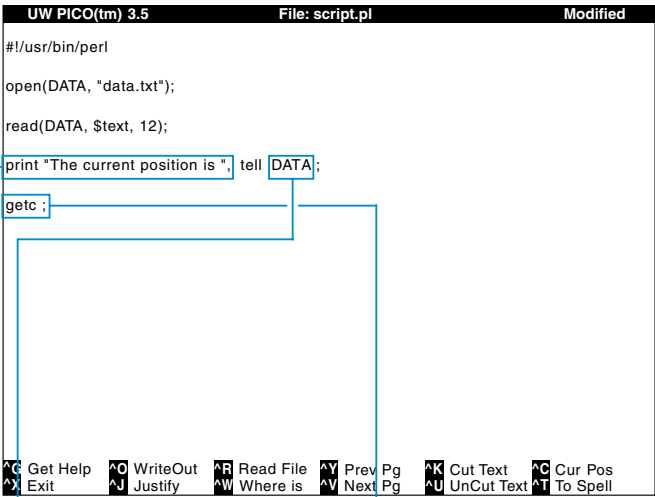
```
Are you sure you want to continue? (y/n): y
Are you sure you want to continue? (y/n): n
Goodbye!
```

DETERMINE THE POSITION IN A FILE



1 Position the cursor where you want to type the code that determines the current position in an open file and type **tell ;**.

To open a file and assign it a filehandle, see page 138.

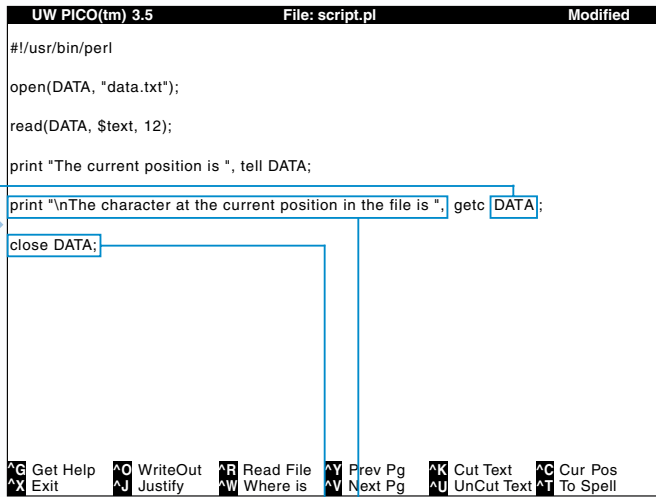


2 Position the cursor over the semi-colon and type the filehandle of the file.

3 Position the cursor where you want to type the code that uses the result of the **tell** function and type the code.

READ A CHARACTER

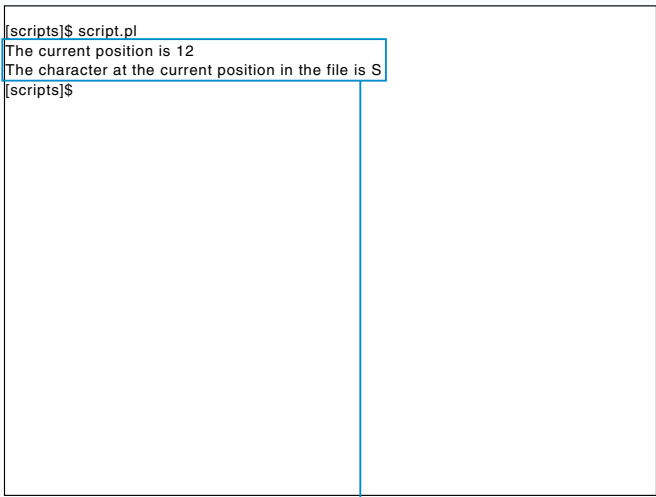
4 To read the character located at the current position, type **getc ;**.



5 Position the cursor over the semi-colon and type the filehandle of the file.

6 Position the cursor where you want to type the code that will use the character read from the file and type the code.

7 Type the code that closes the file. For information about closing a file, see page 138.



8 Save and execute the script.

Perl generates the result of determining the current position in a file and printing the character located at the current position.

DETERMINE THE STATUS OF A FILE

`stat FILENAME` – Determine the status of a file.

The `stat` function provides information about a file you want to work with, such as the size of the file and when the file was last accessed. The information returned by the `stat` function depends on the operating system that Perl is running on. You must have the appropriate file and security permissions to determine the status of a file. For more information about permissions, consult your operating system documentation.

You must specify the name of the file you want to determine the status of. The file name can contain the path of the file. If you do not specify the path,

Perl will assume the file is located in the current directory.

The `stat` function returns a list of 13 values that provide information about the file you specified. To use the returned values, you store the values in variables in a list or in an array. If your operating system does not support all the values, some of the variables in the list may contain a null value. Assigning values to an array is useful if you want to access only one or two values without having to create individual variables to store each value.

Extra

Perl always returns the values of the `stat` function in the same order. The following chart displays the variables, in order, that are commonly used to store the values returned by the `stat` function, and a description of the value each variable stores.

VARIABLE NAME:	DESCRIPTION:
<code>\$dev</code>	The device number, which identifies the file system containing the file.
<code>\$ino</code>	The inode number, which specifies the location of the file.
<code>\$mode</code>	The file type and permissions.
<code>\$nlink</code>	The number of names that refer to the file.
<code>\$uid</code>	The user ID of the file's owner.
<code>\$gid</code>	The group ID of the file's owner.
<code>\$rdev</code>	The device identifier for special files.
<code>\$size</code>	The total size of the file, in bytes.
<code>\$atime</code>	The last access time, in seconds, since the beginning of the epoch.
<code>\$mtime</code>	The last modification time, in seconds, since the beginning of the epoch.
<code>\$ctime</code>	The last modification time of the inode number, in seconds, since the beginning of the epoch.
<code>\$blksize</code>	The preferred block size used for input and output of the file, in bytes.
<code>\$blocks</code>	The number of blocks the file uses.

DETERMINE THE STATUS OF A FILE

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime,
$ctime, $blksize, $blocks) = stat("");

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

1 Type the list of variables that will store the values returned by the `stat` function.

2 Type `= stat("");`.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime,
$ctime, $blksize, $blocks) = stat("data.txt");

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

3 Position the cursor over the closing quotation mark and type the name of the file you want to determine the status of.

4 If the file is not stored in the current directory, type the path of the file.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime,
$ctime, $blksize, $blocks) = stat("data.txt");

print "\nDevice number: ", $dev, "\nInode number: ", $ino,
"\nFile mode: ", $mode, "\nLinks: ", $nlink, "\nUser ID: ", $uid,
"\nGroup ID: ", $gid, "\nDevice: ", $rdev, "\nSize in bytes: ", $size,
"\nLast access time: ", $atime, "\nLast modification time: ", $mtime,
"\nLast inode change: ", $ctime, "\nBlock size: ", $blksize, "\nBlocks: ", $blocks);

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

4 Position the cursor where you want to type the code that uses the values returned by the `stat` function and type the code.

```
[scripts]$ script.pl
Device number: 773
Inode number: 67597
File mode: 33261
Links: 1
User ID: 503
Group ID: 519
Device: 1815
Size in bytes: 571
Last access time: 963329229
Last modification time: 963329222
Last inode change: 963329222
Block size: 4096
Blocks: 8
[scripts]$
```

5 Save and execute the script.

6 Perl generates the result of using the `stat` function to determine the status of a file.

CHANGE THE DEFAULT FILEHANDLE

`select FILEHANDLE` – Specify the default filehandle.

When using a Perl function that outputs information, such as the `print` or `write` function, a filehandle is used to specify where the information should be outputted. You can use the `select` function to change the default filehandle.

On most systems, the default filehandle is set to `STDOUT`. `STDOUT` usually outputs information to the screen. Changing the default filehandle is particularly useful when you want to output multiple statements to a specific file. The new default filehandle will remain in effect for the script until the `select` function is used again.

Care should be taken when changing the default filehandle, as all functions and subroutines that use the default filehandle will output information to the new default filehandle.

The `open` function is often used to open the filehandle you want to set as the new default filehandle. The `close` function closes a filehandle. Once the default filehandle is closed, you will no longer be able to output information using the filehandle. For information about the `open` and `close` functions, see page 138.

Apply It

Even when the `select` function is used to change the default filehandle, the function stores the name of the original default filehandle. By assigning the `select` function to a variable, you can restore the original default filehandle. This is especially useful if other parts of the script require the original default filehandle.

TYPE THIS:

```
$, = " ";
@scores = (65, 75, 23, 98, 79, 86);
print "Ready to save test scores: \n\n";

open(SCOREOUT, ">testscores.txt");
$origHandle = select SCOREOUT;
print @scores;
close SCOREOUT;

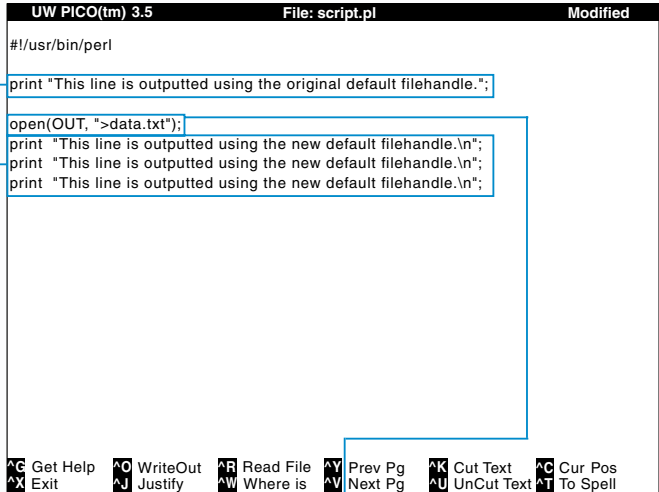
select $origHandle;
print "Scores saved: \n";
print @scores;
```

RESULT:

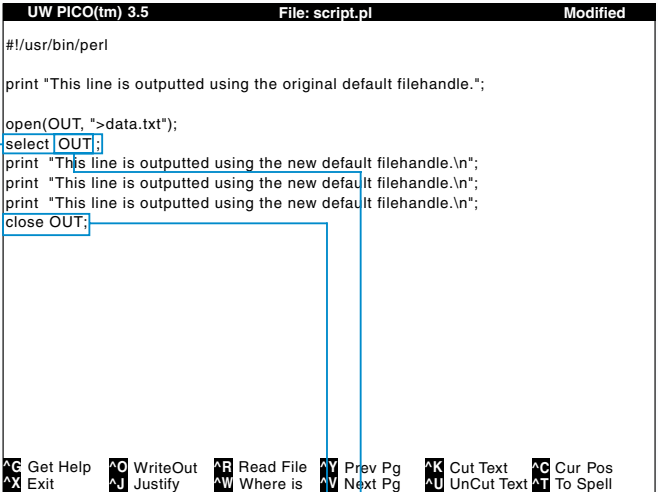
Ready to save test scores:

Scores saved:
65 75 23 98 79 86

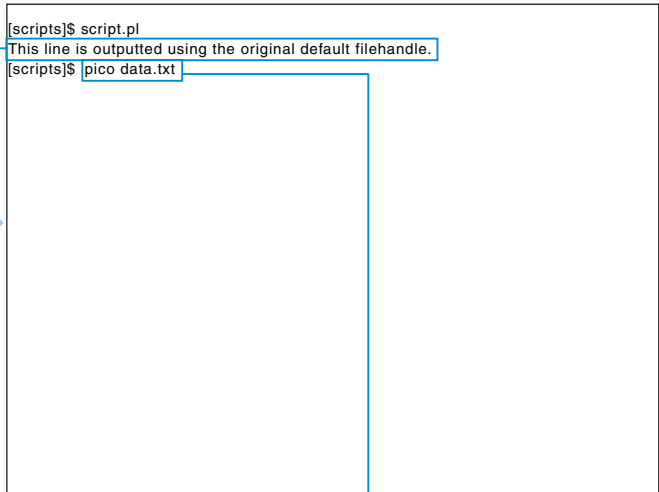
CHANGE THE DEFAULT FILEHANDLE



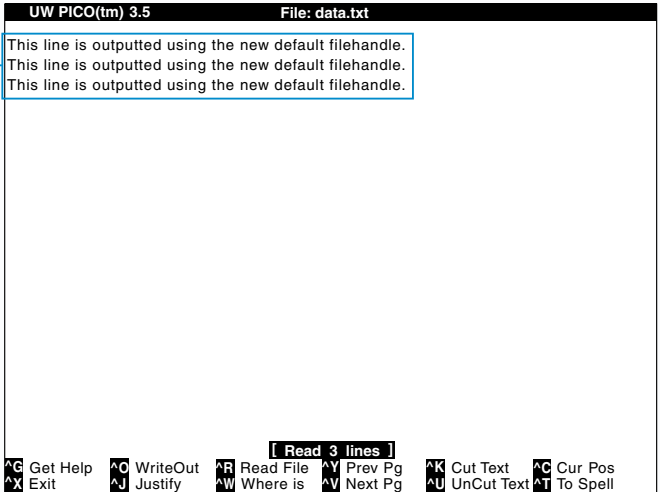
- 1 Type the code you want to output.
- 2 Type the code that opens the new filehandle you want to use to output data.



- 3 Position the cursor above the first line of code you want to output using the new filehandle and type `select ;`.
- 4 Position the cursor over the semi-colon and type the name of the new default filehandle.
- 5 Position the cursor below the last line of code you want to output using the new filehandle and type the code that closes the filehandle.



- 6 Save and execute the script.
- 7 To view the information outputted using the new default filehandle, type the code that displays the contents of the file opened by the new default filehandle.



- The information outputted using the new default filehandle appears.

RENAME OR DELETE A FILE

rename *OLDFILENAME, NEWFILENAME* – Rename a file.
unlink *LIST* – Delete a file.

Perl allows you to manipulate files by renaming or deleting the files. To rename or delete files, you must have the appropriate file and operating system permissions. For information about the permissions for your operating system, refer to the operating system's documentation.

The `rename` function lets you change the name of a file and returns a value of 1 if the operation is successful.

On many operating systems, including UNIX, if the new file name you specify is the same as an existing file name, the existing file will be overwritten. On

some Windows systems, however, the `rename` function will simply fail and the existing file will be left intact.

To delete a file you no longer need using the `unlink` function, specify the name of the file. To delete multiple files at once, separate each file name with a comma (,).

Due to the way UNIX works with files, the `unlink` function may not delete a file but may simply remove the link to the file. For more information about linking and unlinking UNIX files, refer to the documentation provided with your UNIX system.

Extra

There are many different errors that may prevent the `rename` or `unlink` function from working properly. For example, you may not have the correct permissions, the files you specified may no longer exist or you may have misspelled a file name. To find errors, use the `die` function with the `$!` special variable. This will instruct Perl to stop processing the script and display the last system error stored in `$!`.

TYPE THIS:

```
rename "actors.db", "actorslist.db" or die "Can't rename actors.db: $!";
print "File renaming successful.";
```

RESULT:

```
Can't rename actors.db: No such file or directory at script.pl line 3.
```

The `rename` function can be used to move a file to another directory. To move a file, specify the path to the directory where you want to place the file. If you are not currently working in the directory where the file is stored, you must also type the current path of the file you want to move.

Example:

```
rename "personal/weeklog.txt", "logdir/weeklog.txt";
```

RENAME A FILE

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

rename "data.txt", "data.bak";

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ ls

contacts.pl employees.pl products1.pl salesform.pl

data.bak orderlist.txt products2.pl timesheet.pl

[scripts]\$

- 1

To rename a file in the current directory, type **rename**.
- 2

Type the name of the file you want to rename enclosed in quotation marks.
- 3

Type a comma (,) followed by a new name for the file enclosed in quotation marks. Then type semi-colon (;).
- 4

Save and execute the script.
- 5

To view the contents of the current directory, type **ls** and press Enter.
- The contents of the directory appear, displaying the renamed file.

DELETE A FILE

UW PICO(tm) 3.5File: script.plModified

#!/usr/bin/perl

unlink "data.bak";

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ ls

contacts.pl orderlist.txt products2.pl timesheet.pl

employees.pl products1.pl salesform.pl

[scripts]\$

- 1

To delete a file from the current directory, type **unlink**.
- 2

Type the name of the file you want to delete enclosed in quotation marks. Then type a semi-colon (;).
- 3

Save and execute the script.
- 4

To view the contents of the current directory, type **ls** and press Enter.
- The contents of the directory appear. The file you deleted is no longer listed.

TEST A FILE

Before working with a file, it is often a good idea to first test the file for certain characteristics. Since files are often manipulated by other users and programs, you may want to verify characteristics before using the file in your script. For example, you may want to perform a test to determine the size of a file.

To test a file, you use a file test operator in your script. There are many file test operators you can use. For example, the `-T` operator

allows you to determine whether a file is a text file, while the `-z` operator lets you determine whether a file has a size of zero. File test operators are most often used with conditional statements, such as the `if` statement.

One of the most commonly used file test operators is the `-e` operator, which tests to see if a file exists. You may want to determine if the file exists before you perform any operations, such as overwriting the file.

TEST A FILE

UW PICO(tm) 3.5

File: script.pl

Modified

```
#!/usr/bin/perl

$file = "data.txt";

if (-e $file)
{
    print "File exists\n";
}
else
{
    die "File does not exist\n";
}
```

Get Help

WriteOut

Read File

Prev Pg

Cut Text

Cur Pos

Exit

Justify

Where is

Next Pg

UnCut Text

To Spell

```
[scripts]$ script.pl
File does not exist

[scripts]$
```

- 1 Type the code that creates a variable and assigns it the name of the file you want to test.
- 2 Position the cursor where you want to test the file and type the file test operator you want to use followed by the name of the variable that stores the file name.
- 3 Position the cursor where you want to type the code that uses the result of testing the file and type the code.
- 4 Save and execute the script.
- Perl generates the result of testing the file.

Extra

The following chart lists the file test operators you can use to test a file.

OPERATOR:	RETURNS:
-A	Days since the file was last accessed.
-b	1 if the file is a block special file, which is a type of file that accesses a device.
-B	1 if the file is a binary file.
-c	1 if the file is a character special file, which is a type of file that accesses a device.
-C	Days since the file's inode was last changed.
-d	1 if the directory exists.
-e	1 if the file exists.
-f	1 if the file is an ordinary file.
-g	1 if the file has its SETGID bit set. A program that has the SETGID bit set considers the group ID of the person who executes the program as if it is also the group ID of the program.
-k	1 if the sticky bit of a directory is set. The sticky bit prevents a user from renaming or removing files owned by someone else in a directory even if the user owns the directory.
-l	1 if the file is a symbolic link to a file.
-M	Days since the file was modified.
-o	1 if the file's owner is also the effective owner of the Perl script.
-O	1 if the file's owner is also the real owner of the Perl script.
-p	1 if the file is a named pipe. A pipe transfers data between programs and processes.
-r	1 if the file is readable by the effective owner of the Perl script.
-R	1 if the file is readable by the real owner of the Perl script.
-s	Size of the file in bytes.
-S	1 if the file is a socket. A socket transfers data between a network protocol and a program or process.
-t	1 if the filehandle refers to input from or output to the terminal rather than a file.
-T	1 if the file is a text file.
-u	1 if the file has its SETUID bit set. A program that has the SETUID bit set treats the person who executes the program as if they are the program's owner.
-w	1 if the file can be written to by the effective owner of the Perl script.
-W	1 if the file can be written to by the real owner of the Perl script.
-x	1 if the file can be executed by the effective owner of the Perl script.
-X	1 if the file can be executed by the real owner of the Perl script.
-z	1 if the file's size is zero.

WORK WITH DIRECTORIES

mkdir *DIRECTORY, PERMISSIONS* – Create a directory.
rmdir *DIRECTORY* – Remove a directory.
chdir *DIRECTORY* – Change to a different directory.

Perl provides functions to help you manage your directories. The `mkdir` function creates a new directory, while the `rmdir` function removes an existing directory.

When working with directories in the current directory, you specify the name of the directory you want to create or remove. If you want to work with directories in a different directory, you must specify the path of the directory you want to create or remove.

When creating a new directory, you must specify permissions for the directory to control access to the directory.

A directory you want to remove must not contain any files or subdirectories. You can remove files from

a directory using the `unlink` function. See page 152 for information about the `unlink` function.

When the `mkdir` and `rmdir` functions successfully create or remove a directory, they return a value of 1. If the functions are not successful, they return a value of 0 and store error information in the `$!` special variable.

As with all functions that access a computer's file system, you must have the appropriate file and operating system permissions to use the `mkdir` and `rmdir` functions. For information about the permissions for your operating system, refer to the operating system's documentation.

Extra

Perl allows you to specify permissions for a directory in octal notation. You can convert UNIX permissions to octal notation using the following chart. The UNIX permissions include read (r), write (w) and execute (x). The octal value will consist of three digits, representing the permissions for the owner, followed by the permissions for the owner's group and the permissions for other users. For example, a directory with permissions `-rwxrw-r--` will have the octal value 764. When entering the permissions in a Perl script, precede the octal value with a zero (0).

UNIX Permissions	Octal Value
---	0
--X	1
-W-	2
-WX	3
r--	4
r-X	5
rw-	6
rwX	7

The `chdir` function allows you to change which directory you are working in. This is useful when you want to work with files stored in another directory. To change the current directory, specify the name of the directory enclosed in quotation marks.

Example:

```
chdir "datadir";
```

CREATE A DIRECTORY

UW PICO(tm) 3.5

File: script.pl

Modified

```
#!/usr/bin/perl
mkdir "datafiles", 0774;
```

Get Help
Exit

WriteOut
Justify

Read File
Where is

Prev Pg
Next Pg

Cut Text
UnCut Text

Cur Pos
To Spell

[scripts]\$ ls

customers.pl employees.pl picturelist.pl schedule.pl
datafiles passfile.dat products.pl timesheet.txt

[scripts]\$

- 1

To create a directory within the current directory, type `mkdir "", ;` and then press Enter.
- 2

Position the cursor over the closing quotation mark and type a name for the new directory.
- 3

Position the cursor over the semi-colon and type the octal value for the permissions you want the directory to use.
- 4

Save and execute the script.
- 5

To view the contents of the current directory, type `ls` and press Enter.
- The contents of the directory appear, displaying the new directory.

REMOVE A DIRECTORY

UW PICO(tm) 3.5

File: script.pl

Modified

```
#!/usr/bin/perl
rmdir "datafiles";
```

Get Help
Exit

WriteOut
Justify

Read File
Where is

Prev Pg
Next Pg

Cut Text
UnCut Text

Cur Pos
To Spell

[scripts]\$ ls

customers.pl employees.pl picturelist.pl schedule.pl
passfile.dat products.pl timesheet.txt

[scripts]\$

- 1

To remove a directory from the current directory, type `rmdir ""`; and then press Enter.
- 2

Position the cursor over the closing quotation mark and type the name of the directory you want to remove.
- 3

Save and execute the script.
- 4

To view the contents of the current directory, type `ls` and press Enter.
- The contents of the directory appear. The directory you removed is no longer displayed.

ACCESS ENVIRONMENT VARIABLES

A Perl script can be used to access environment variables, which store information about the user and the computer running the script. The available environment variables depend on the type of operating system Perl is running on.

Environment variables are stored in a hash named %ENV. To access the value of an individual environment variable, specify the name of the hash and the name of the variable you want to access. You must use uppercase letters when specifying the name of an environment variable.

You can also change the value of an environment variable you access. For example, you could change

the value of the HOME variable to change the home directory. Altering the value of an environment variable affects only the script in which the change is made.

To access the name and current value of each environment variable, use the each function with code that creates a loop, such as a while statement.

Additional environment variables are created when a Perl script uses a CGI program. For information about CGI environment variables, see page 210.

Extra

The following environment variables are commonly found on UNIX systems.

VARIABLE:	VALUE:
HOSTTYPE	The type of computer running the script.
USERNAME	The login name of the user running the script.
LANG	The locale of the computer running the script.
MAIL	The path of the user's e-mail inbox.
LOGNAME	The login name of the user running the script.
OSTYPE	The type of operating system running the script.
INPUTRC	The location of the initialization file for the READLINE program, which allows the user to edit text at the command prompt.
BASH_ENV	The location of the shell script that runs when the shell starts.
LC_ALL	The locale setting. This variable overrides all other locale settings, including the LANG setting.
SHLVL	The number of shell levels the script is running under.
HOME	The path of the home directory of the user running the script.
PATH	The paths the computer will search for programs.
LINGUAS	The locale of the computer running the script.
HISTSIZE	The maximum number of commands that can be saved in the shell's command history.
HISTFILESIZE	The maximum number of lines that can be saved in the shell's command history.
PS1	The prompt used by the shell.
TERM	The type of terminal that the operating system output is configured for.
SHELL	The path of the shell that interprets the system commands.
_	The last command entered at the command prompt.
HOSTNAME	The name of the computer running the script.
USER	The login name of the user running the script.

ACCESS ENVIRONMENT VARIABLES

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
print $ENV{'HOME'};
print " is the home directory.\n\n";
```

ACCESS AN ENVIRONMENT VARIABLE

- 1 To access an environment variable, type \$ENV{''}.
- 2 Position the cursor over the closing quotation mark and type the name of the environment variable you want to access.
- 3 Position the cursor where you want to type the code that uses the environment variable and type the code.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
print $ENV{'HOME'};
print " is the home directory.\n\n";
print $ENV{'HOME'} = "HOME/HTTPD/";
print " is the new home directory.\n\n";
```

CHANGE THE VALUE OF AN ENVIRONMENT VARIABLE

- 4 To change the value of an environment variable, perform steps 1 and 2.
- 5 Position the cursor outside the closing brace (}) and type = followed by the new value enclosed in quotation marks.
- 6 Position the cursor where you want to type the code that uses the environment variable and type the code.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl
print $ENV{'HOME'};
print " is the home directory.\n\n";
print $ENV{'HOME'} = "HOME/HTTPD/";
print " is the new home directory.\n\n";
while (($name, $value) = each(%ENV))
{
    print "$name => $value\n";
}
```

ACCESS ALL THE ENVIRONMENT VARIABLES

- 7 Type the code that creates a loop and uses the each function to access the name and value of each environment variable in the %ENV hash.
- 8 Position the cursor where you want to type the code that uses each name and value and type the code.

```
[scripts]$ script.pl
/home/april is the home directory.
HOME/HTTPD/ is the new home directory.
HOSTTYPE => i386
USERNAME =>
LANG => en_US
MAIL => /var/spool/mail/april
LOGNAME => april
OSTYPE => Linux
INPUTRC => /etc/inputrc
BASH_ENV => /home/april/.bashrc
LC_ALL => en_US
SHLVL => 1
HOME => HOME/HTTPD/
PATH => /usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/april/bin
LINGUAS => en_US
HISTSIZE => 1000
HISTFILESIZE => 1000
PS1 => \n[W]$
TERM => vt100
SHELL => /bin/bash
_ => ./script.pl
HOSTNAME => server
USER => april
[scripts]$
```

Save and execute the script.

Perl displays the results of accessing and changing the value of environment variables.