

# USING THE PRINT FUNCTION

`print` *filehandle LIST* – Create output.

Perl uses the `print` function to output information from a script. If you are running a Perl script from the command prompt, the output is sent to the screen. If a visitor to a Web site processes a Perl script, the `print` function outputs information to the Web browser.

The `print` function can be used to output a single item of information such as a string, a number, the value of a variable or the result of a function. If you want to use the `print` function to output multiple items, you must separate each item with a comma. In some circumstances, you may need

to use parentheses with the `print` function to output multiple items.

If you do not specify the item you want to output, the `print` function will output the value of the default special variable `$_`.

The `print` function returns a value to indicate whether or not the output operation was successful. If the operation was successful, a value of `true` is returned. If the operation was unsuccessful, the function returns a value of `false`.

## Extra

You can have the `print` function output information to a file instead of a screen. To output information to a file, you must use the `open` function to open a filehandle and specify the file to which you want to output information. You then use the filehandle in the `print` function to output information to the file. After you have outputted information to a file, you can open the file to view the information. In the following example, the filehandle `TODAY` is used to output information to the file `today.log`.

TYPE THIS:

```
open(TODAY, ">> today.log");
print TODAY time, " Output this to a file.\n";
close TODAY;
```

By default, when you use the `print` function to output multiple items, the function outputs the items with no separating character or white space. To have the `print` function output a character or white space between items, you can set the value of the output field separator special variable `$,` in the script.

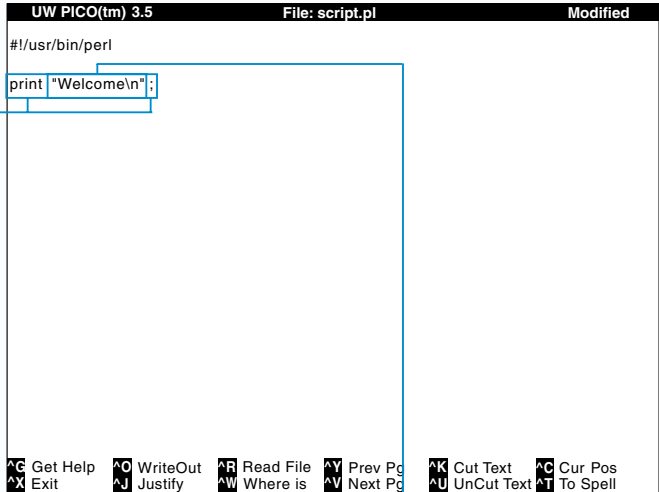
TYPE THIS:

```
@userRecord = ("UserID", 46865, "Darren Meiss");
print @userRecord;
print "\n";
$, = ":";
print @userRecord;
```

RESULT:

```
UserID46865Darren Meiss
UserID:46865:Darren Meiss
```

### USING THE PRINT FUNCTION



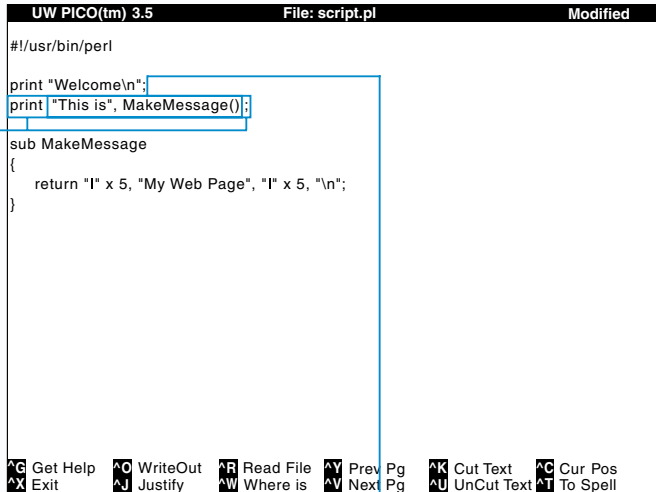
**1** Position the cursor where you want to output one item and type `print ;`.

**2** Position the cursor over the semi-colon and type the item you want to output.  
■ Strings must be enclosed in quotation marks.



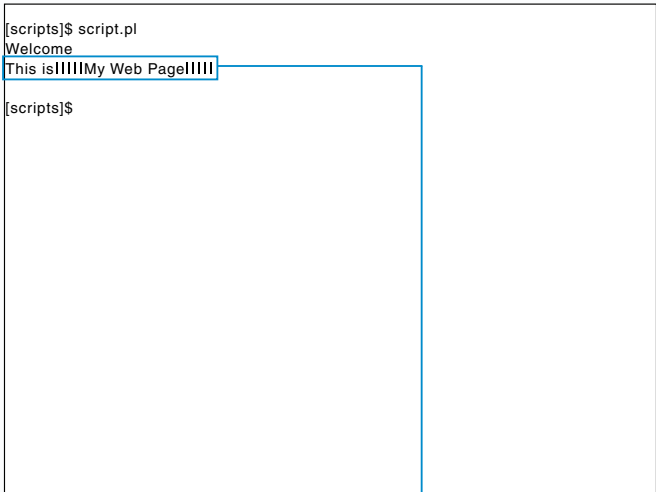
**3** Save and execute the script.

■ Perl generates the result of using the `print` function to output one item.



**1** Position the cursor where you want to output multiple items and type `print ;`.

**2** Position the cursor over the semi-colon and type each item you want to output, separated with a comma (,).



**3** Save and execute the script.

■ Perl generates the result of using the `print` function to output multiple items.

# FORMAT OUTPUT WITH THE PRINTF FUNCTION

`printf` *filehandle* *FORMATSTRING*, *LIST* – Format output.

The `printf` function allows you to format a string for output to a screen or file. For information about outputting information to a file, see page 138.

The `printf` function is similar to the `sprintf` function, except `sprintf` is used to format a string which can then be used in the script, while `printf` allows you to output a formatted string. For information about the `sprintf` function, see page 44.

The `printf` function requires two arguments—a format string and a list of values you want to

format. A format string contains the formatting rules that will be applied to the values. Each formatting rule in the format string must have a corresponding value in the list.

Formatting rules are specified using a percentage symbol (%) and a letter that indicates the type of formatting to be applied. Formatting rules may also include numbers that indicate a minimum width or a precision for values. For example, the formatting rule `%8s` formats a string value so it will have a minimum width of eight characters. The formatting rule `%.2f` formats a floating point number with a precision of two decimal places.

## Extra

You can use the following letters to indicate the type of formatting you want to apply using the `printf` function.

c Single character, by value	lu Long unsigned decimal integer
d Decimal integer	lx Long hexadecimal integer
e Scientific notation, lowercase	o Octal integer
E Scientific notation, uppercase	s String
f Floating-point number	u Unsigned decimal integer
g Floating-point number, scientific notation, lowercase	x Hexadecimal integer, lowercase
G Floating-point number, scientific notation, uppercase	X Hexadecimal integer, uppercase
ld Long decimal integer	p Hexadecimal address of value
lo Long octal integer	n Number of characters outputted

There are additional characters, called flags, which can be placed between the percentage symbol (%) and the letter in a formatting rule to further format a string.

-	Left justify
#	Add 0 or 0x to non-zero octal or hexadecimal number
+	Add plus sign to positive number
0	Right justify using zeros, instead of spaces
(space)	Put a space before positive number

## FORMAT OUTPUT WITH THE PRINTF FUNCTION

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

%hitInfo = ('Home Page', 2045.56, 'Contact Information', 500.34,
'Products and Services', 1267.23);

print "Page Statistics\n";
printf "%22s %9d hits\n";
```

1 To format a string for output, type `printf "" ;`.

2 Position the cursor over the closing quotation mark and type the formatting rules you want to use.

*Note: You can also include characters you do not want to format.*

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

%hitInfo = ('Home Page', 2045.56, 'Contact Information', 500.34,
'Products and Services', 1267.23);

print "Page Statistics\n";
printf "%22s %9d hits\n", $page;
```

3 Position the cursor over the semi-colon and type a comma (,) followed by the value you want to use the first formatting rule you specified.

Text values must be enclosed in quotation marks.

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

%hitInfo = ('Home Page', 2045.56, 'Contact Information', 500.34,
'Products and Services', 1267.23);

print "Page Statistics\n";

while (($page, $hits) = each(%hitInfo))
{
    print "Web page:";
    printf "%22s %9d hits\n", $page, $hits;
}
```

4 Repeat step 3 for each formatting rule you specified.

5 Position the cursor where you want to type the code that uses the `printf` function and type the code.

```
[scripts]$ script.pl
Page Statistics
Web page:      Contact Information      500 hits
Web page: Products and Services      1267 hits
Web page:           Home Page      2045 hits
[scripts]$
```

6 Save and execute the script.

Perl generates the result of using the `printf` function to format output.

DEFINE A FORMAT

Defining a format allows you to produce formatted output, such as columns of data. A format definition begins with the `format` keyword and ends with a period (`.`) that appears on its own line. Format definitions are usually placed at the end of a script.

When you enter text in a format definition, the text will be output using the spacing you type. To add a value you want to format, type a fieldholder where you want the value to appear. A fieldholder consists of the `@` symbol followed by characters that specify an alignment for the data. The `<` character left aligns data, `>` right aligns data and `|` centers data.

The width of a formatted value depends on the number of characters you type for its fieldholder. For example, typing `@<<<<<<<<` will left align a value in a block nine characters wide.

Each line that contains a fieldholder must be followed by a line that specifies the variable that stores the value you want to display.

To display the results of a format, you assign the format definition to the format name special variable `$~`. You then use the `write` function to output the value of the special variable.

Extra

When creating a fieldholder for a number value, you can use the hash symbol (`#`) to specify a precision for the value. To specify a precision, type a decimal point followed by one hash symbol for each decimal place you want the value to display.

TYPE THIS:

```
$rev1 = 935.876;  
$rev2 = 1315.5;  
format SALES =  
Jan.-June      July-Dec.  
-----  
@####.##      @####.##  
$rev1, $rev2  
.  
$~ = "SALES";  
write;
```

RESULT:

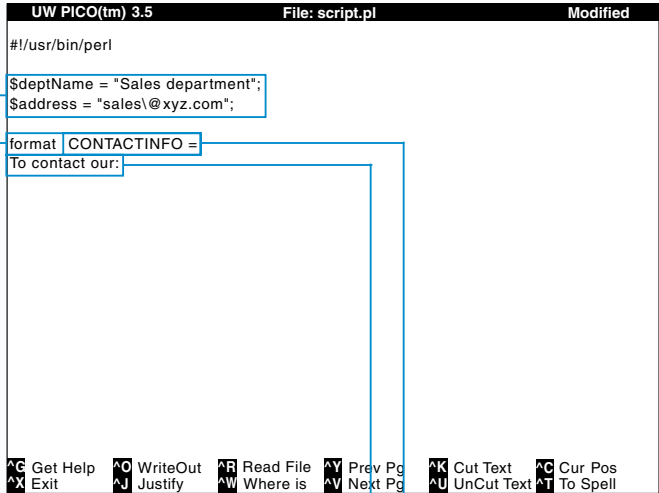
Jan.-June	July-Dec.
-----	-----
935.88	1315.50

If a format you define will produce output that spans several pages, you may want to create a document header format to specify information that will appear at the top of each output page. To name a document header format, add `_TOP` to the name of the format definition that will use the header.

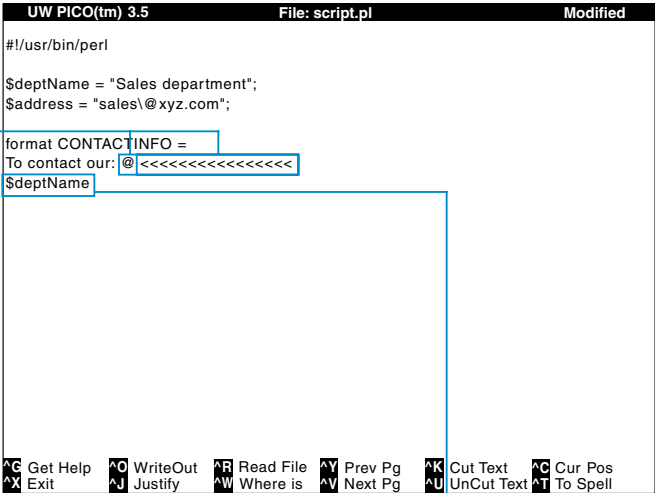
Example:

```
format EMPLOYEES_TOP =  
EMPLOYEE CONTACT LIST  
-----  
LAST NAME  GIVEN NAME  ADDRESS  
-----  
.
```

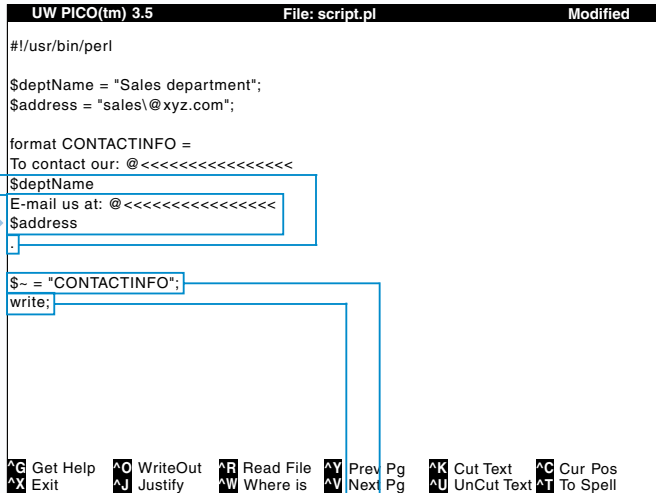
DEFINE A FORMAT



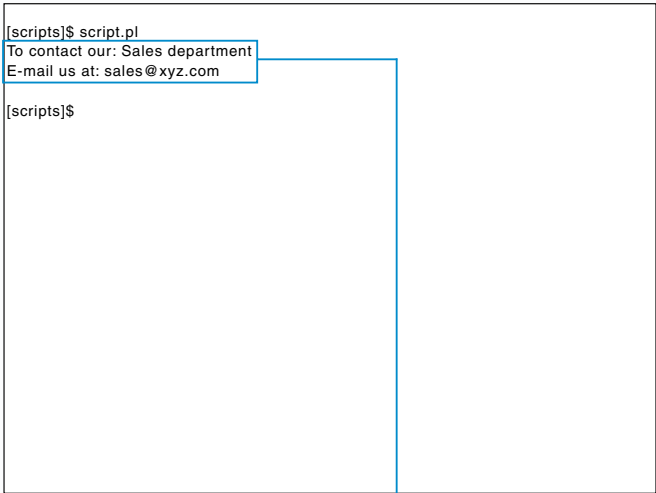
- 1 Type the code that creates the variables you want to format and assign their values.
- 2 Position the cursor where you want to define a format and then type `format`.
- 3 Type a name for the format definition followed by an equal sign (`=`). Then press Enter.
- 4 To output formatted text, type the text.



- 5 To enter a fieldholder for a value you want to format, type `@`.
- 6 Type a character to specify how you want to align the value (`<`, `>` or `|`). Repeat this step until the fieldholder displays the number of characters you want the value to use.
- 7 Position the cursor directly below the line containing the fieldholder and type the value you want to format.



- 8 Repeat steps 4 to 7 until you have specified all the text and values you want to format.
- 9 Position the cursor below the last line of the format definition and type a period (`.`).
- 10 To assign the format definition to the format name special variable, type `$~ =` followed by the name of the format definition. Then press Enter.
- 11 To output the formatted text and values, type `write;`.



- 12 Save and execute the script.
- Perl generates the result of defining a format and outputting the formatted text and values.

# GET KEYBOARD INPUT

You can use the `<STDIN>` line input operator to set up your script to read information entered using a keyboard. A line input operator is made up of angle brackets `<>` and a filehandle. `STDIN` is the filehandle Perl typically uses to access input from a keyboard and stands for *standard input*. There are also filehandles available that perform other tasks, such as a filehandle that reads information from a file you specify.

Filehandles usually appear in all uppercase letters to prevent people who read your code from

confusing the filehandle with the name of other variables or functions in the script. Using all uppercase letters for filehandles also helps make your code easier to read.

Perl allows you to assign the data accessed by the `<STDIN>` line input operator to a variable in your script. When the script is being executed and Perl reaches the line of code that assigns the data to the variable, Perl will wait for input from the keyboard before processing any further code. Data input using a keyboard is usually sent to the script when the user presses Enter.

## Extra

Using the `while` loop with the `<STDIN>` line input operator as its condition allows you to create a loop that will process each line of data as it is entered by a user. On each iteration of the loop, Perl will check to see if a new line of data has been entered. When data is input, Perl will process the block of code for the loop.

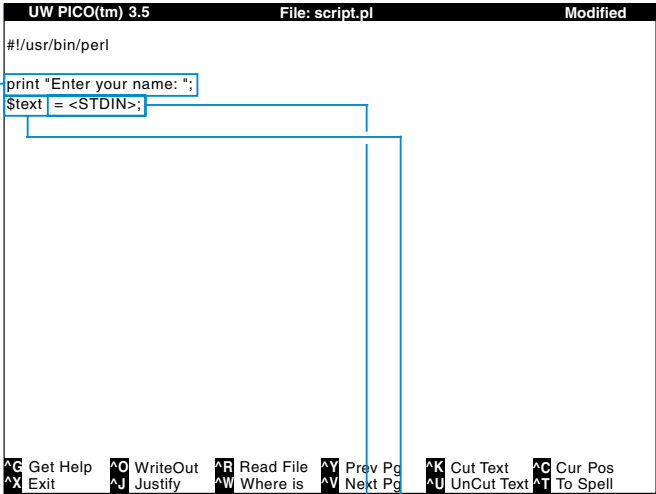
```
TYPE THIS:
while (<STDIN>)
{
    print "Reversed:", scalar reverse, "\n";
}
```

```
RESULT:
hello
Reversed:
olleh
goodbye
Reversed:
eybdoog
```

If you use a `while` loop to read keyboard input, users must use a specific keyboard combination to generate an EOF (end-of-file) character to specify when they want to stop entering data and end the loop. When using DOS or the Windows command prompt, a user can type **Ctrl+Z** to generate the EOF character. In UNIX-based systems, a user can usually type **Ctrl+D** to end keyboard input.

By default, Perl will use the `<STDIN>` line input operator if you omit the filehandle and type only the angle brackets `<>` in the script. For example, the code `while (<>)` will perform the same task as the code `while (<STDIN>)`.

### GET KEYBOARD INPUT

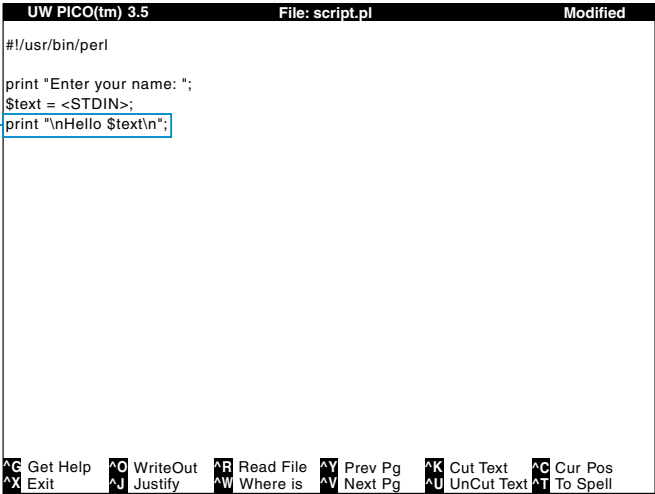


#### SET UP A SCRIPT TO GET KEYBOARD INPUT

1 Type the code that will prompt the user to enter data and then press Enter.

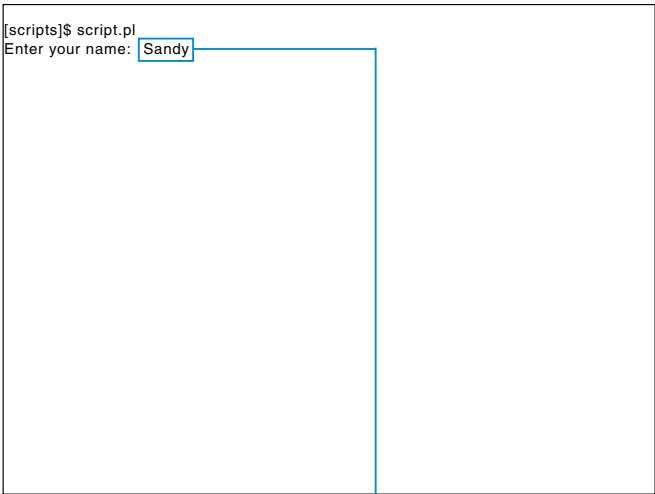
2 Type the name of the variable you want to use to store the keyboard input.

3 Type `= <STDIN>;` and then press Enter.



4 Position the cursor where you want to type the code that uses the `<STDIN>` line input operator and type the code.

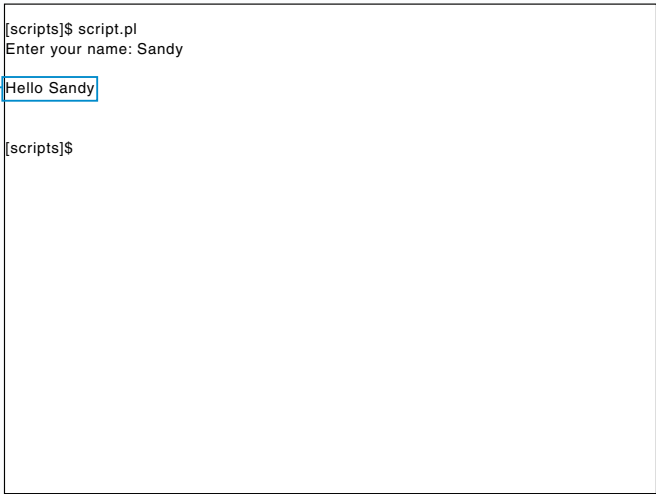
You can now enter data using the keyboard.



#### ENTER DATA USING THE KEYBOARD

1 Save and execute the script.

2 Type the data you want to send to the script and then press Enter.



Perl generates the result of using the `<STDIN>` line input operator to get keyboard input from a user.

# EXECUTE A SYSTEM COMMAND

**exec** *COMMAND* – Execute a system command, ending a script.  
**system** *COMMAND* – Execute a system command, pausing a script.

The `exec` and `system` functions allow you to execute a system command from a Perl script. This lets you perform tasks such as creating files to store directory lists and running programs such as backup programs. The system commands you will use depend on the operating system you use.

When you use the `exec` function, Perl will stop processing the script when the system command you specify is executed. Any code after the `exec` function in the script will not be processed.

When you use the `system` function in a script, Perl will pause the processing of the script until

the system command you specify is executed. Perl will then resume processing any code that appears after the `system` function in the script.

The `system` function returns an *exit value* generated by its system command. The exit value is useful for troubleshooting and for verifying that the command was completed without errors. The `exec` function does not return a value.

The `exec` and `system` functions may not run on computers using the Windows operating system.

## Apply It

The `exec` and `system` functions are not limited to literal strings for their arguments. You can include variables and the results of functions, operators and subroutines within an argument to increase the flexibility of the system commands. In this example, a variable is used to create a dynamic command that allows users to input a name for a file that will store a list of files.

TYPE THIS:

```
print "Enter a file name for your directory list: ";
$fileName = <STDIN>;
chomp $fileName;
system "ls > $fileName";
print "\n";
open(SAVEDFILE, $fileName);
print while <SAVEDFILE>;
```

RESULT:

```
Enter a file name for your directory list: files.txt

case1.pl
contacts.pl
employees.pl
orderlist.pl
products1.pl
products2.pl
salesform.pl
timesheet.pl
```

### EXECUTE A SYSTEM COMMAND

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

print "A list of files will now be written to the directory.txt file.\n";

exec "ls > directory.txt";

open(IN, "directory.txt");
while (<IN>)
{
    print;
}
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

[scripts]\$ script.pl

A list of files will now be written to the directory.txt file.

[scripts]\$

#### USING THE EXEC FUNCTION

- 1 To execute a system command and end the script, type `exec ""`;
- 2 Position the cursor over the closing quotation mark and type the system command you want to execute.
- 3 Save and execute the script.
- Perl generates the result of using the system command and stops processing the Perl script.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

print "A list of files will now be written to the directory.txt file.\n";

system "ls > directory.txt";

open(IN, "directory.txt");
while (<IN>)
{
    print;
}
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

#### USING THE SYSTEM FUNCTION

- 1 To execute a system command and pause the script, type `system ""`;
- 2 Position the cursor over the closing quotation mark and type the system command you want to execute.
- 3 Save and execute the script.
- Perl generates the result of using the system command and then continues to process the Perl script.