

CREATE A REGULAR EXPRESSION

A regular expression allows you to find patterns of text within a string. You can also use regular expressions to perform tasks such as replacing a pattern in a string or determining where Perl finds a pattern match. Regular expressions are ideal for working with Web pages that contain only text.

The string you want to search using a regular expression must be a scalar value, which is typically stored in a variable. You can also search multiple values stored in an array. Using the =~ binding operator is a common

way to indicate which variable stores the string you want to search and the pattern you want to search for.

The pattern you want to search for is enclosed within forward slashes //, called pattern delimiters.

If a pattern you specify is found in the string, the result of the regular expression will be true. For this reason, regular expressions are often used with conditional statements, such as if and while, which process a result if a condition is true.

Extra

If a string you want to search is stored in the default special variable \$_, you can omit the variable and the binding operator from the regular expression to help you save time. Perl will use the regular expression to search the string stored in the default special variable.

TYPE THIS:

```
$_ = "Lindsay Sandman 123 Maple Lane";
if (/Lindsay/)
{
    print "Lindsay is a customer\n";
}
```

RESULT:

Lindsay is a customer

TYPE THIS:

```
print "Enter command: ";
while (<STDIN>)
{
    last if (/start/);
    print "Try again: ";
}
print "The program has started.\n";
```

RESULT:

Enter command: initiate
Try again: begin
Try again: start
The program has started.

CREATE A REGULAR EXPRESSION

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

$webPage = "www.xyz.com/index.html";

$webPage
```

1 Type the code that creates a string variable and assigns it a value.

2 To search the string using a regular expression, type the name of the variable that stores the string you want to search.

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

$webPage = "www.xyz.com/index.html";

$webPage =~ /index/
```

3 Type =~ followed by the pattern delimiters // and then press Enter.

4 Position the cursor over the closing pattern delimiter (/) and type the pattern you want to search for.

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

$webPage = "www.xyz.com/index.html";

if ($webPage =~ /index/)
{
    print "$webPage is the main page\n";
}
```

5 Position the cursor where you want to type the code that uses the regular expression and type the code.

```
[scripts]$ script.pl
www.xyz.com/index.html is the main page
[scripts]$
```

6 Save and execute the script.

Perl generates the results of using a regular expression to search for a pattern in a string.

MATCH CHARACTERS

Perl includes special characters, called metacharacters and escape sequences, which you can use to match specific patterns in a string.

Metacharacters allow you to match certain conditions in a string. For example, you can match a pattern only at the beginning of a string using the `^` metacharacter or you can match a pattern only at the end of a string using the `$` metacharacter.

If you want to search for characters that have special meaning in Perl, such as `^` or `$`, you must enter a backslash (`\`) in front of the

character in your code. For example, to find the `$` character, you must specify `\$` in the regular expression. Other characters that must be preceded with a backslash include `.`, `|`, `()`, `[`, `{`, `*`, `+` and `?`.

You can use escape sequences in regular expressions to match special characters, such as a newline or a blank space. Escape sequences begin with a backslash (`\`) followed by one or more characters. For example, you can specify the escape sequence `\D` in a regular expression to match any character that is not a number.

Extra

These are some of the most common escape sequences used to match special characters in a string.

ESCAPE SEQUENCE:	FINDS:	ESCAPE SEQUENCE:	FINDS:
<code>\a</code>	An alarm (bell).	<code>\Q</code>	The next metacharacters until <code>\E</code> is found.
<code>\c?</code>	The control character you specify for <code>?</code> .	<code>\r</code>	A carriage return.
<code>\d</code>	A digit between 0 and 9.	<code>\s</code>	A whitespace character.
<code>\D</code>	A non-digit character.	<code>\S</code>	A non-whitespace character.
<code>\e</code>	The character generated by pressing Escape.	<code>\t</code>	A tab.
<code>\E</code>	The end of a <code>\L</code> or <code>\U</code> sequence.	<code>\u</code>	The next uppercase character.
<code>\f</code>	A form feed.	<code>\U</code>	The next uppercase characters until <code>\E</code> is found.
<code>\l</code>	The next lowercase character.	<code>\w</code>	A word character. Word characters are alphanumeric characters or underscores.
<code>\L</code>	The next lowercase characters until <code>\E</code> is found.	<code>\W</code>	A non-word character.
<code>\n</code>	A new line of data.	<code>\0?</code>	The octal character you specify for <code>?</code> .
		<code>\x?</code>	The hexadecimal character you specify for <code>?</code> .

MATCH CHARACTERS

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

@webPages = ('index.html', 'index.asp', 'products.html');
foreach $page (@webPages)
{
    if ($page =~ //)
    {
        print "$page is a main page.\n";
    }
}
print "\n\n";

Get Help  WriteOut  Read File  Prev Pg   Cut Text   Cur Pos
Exit      Justify   Where is  Next Pg   UnCut Text To Spell
```

- 1 Type the code that creates an array or variable and assigns its value.
- 2 If you created an array in step 1, type the code that will process each item in the array.
- 3 To search the strings using a regular expression, type the name of the variable you want to search followed by `=~` and the pattern delimiters `//`.

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

@webPages = ('index.html', 'index.asp', 'products.html');
foreach $page (@webPages)
{
    if ($page =~ /*index/)
    {
        print "$page is a main page.\n";
    }
}
print "\n\n";

Get Help  WriteOut  Read File  Prev Pg   Cut Text   Cur Pos
Exit      Justify   Where is  Next Pg   UnCut Text To Spell
```

- 4 To search for a pattern using a metacharacter, position the cursor over the closing pattern delimiter `/` and type the metacharacter you want to use.

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

@webPages = ('index.html', 'index.asp', 'products.html');
foreach $page (@webPages)
{
    if ($page =~ /*index/)
    {
        print "$page is a main page.\n";
    }
}

print "\n\n";
@numbers = ("Lindsay", 26, "Ted", 43, "Mary", 16);
foreach $number (@numbers)
{
    if ($number =~ /\D/)
    {
        print "$number is not a number.\n";
    }
}

Get Help  WriteOut  Read File  Prev Pg   Cut Text   Cur Pos
Exit      Justify   Where is  Next Pg   UnCut Text To Spell
```

- 6 To search for a pattern using an escape sequence, perform steps 1 to 4, except type the escape sequence you want to use in step 4.

```
[scripts]$ script.pl
index.html is a main page.
index.asp is a main page.

Lindsay is not a number.
Ted is not a number.
Mary is not a number.

[scripts]$
```

- 7 Save and execute the script.
- Perl generates the results of matching characters in strings.

USING CHARACTER CLASSES

A character class can be used to specify the pattern you want to match in a string. A character class is made up of a group of characters. Only one of the characters in the class has to be present in a string for Perl to find a match.

The characters you want to use in a character class must be enclosed in brackets []. For example, to search a string for the lowercase letters a, b or c, use the character class [abc]. Do not separate the characters with commas or spaces, unless you want Perl to search for the commas or spaces.

You can use a dash (-) to specify a range of characters in a character class. For example, to search a string for an uppercase letter between A and Z, you could use the character class [A-Z].

To have Perl search for some special characters, such as \ | () [{ ^ \$ * + . or ?, you must enter a backslash (\) before the character. For example, to find the \$ character, you must specify \\$ in the character class.

Apply It

You can have Perl search for any character that is not in the character class you specify. To do so, place the caret character (^) immediately after the opening bracket, such as [^abc]. If you place the caret character in another position in the class, Perl will search for the caret character in the string. The following example checks each password entered by a user to ensure that at least one character is not a letter.

TYPE THIS:

```
print "Your password must not be all letters\n";
print "Enter password: ";

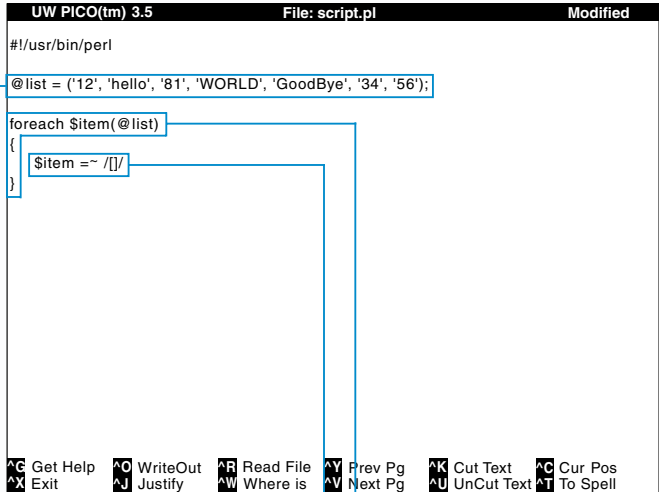
while (<STDIN>)
{
    chomp;
    last if (/[^A-Za-z]/);
    print "Try again: ";
}

print "Password approved";
```

RESULT:

```
Your password must not be all letters
Enter password: aaabbbccc
Try again: aaabbbccc1
Password approved
```

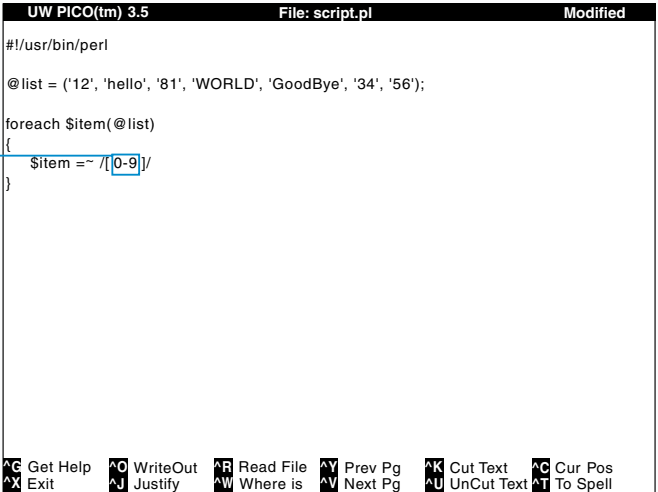
USING CHARACTER CLASSES



1 Type the code that creates an array or variable and assigns its value.

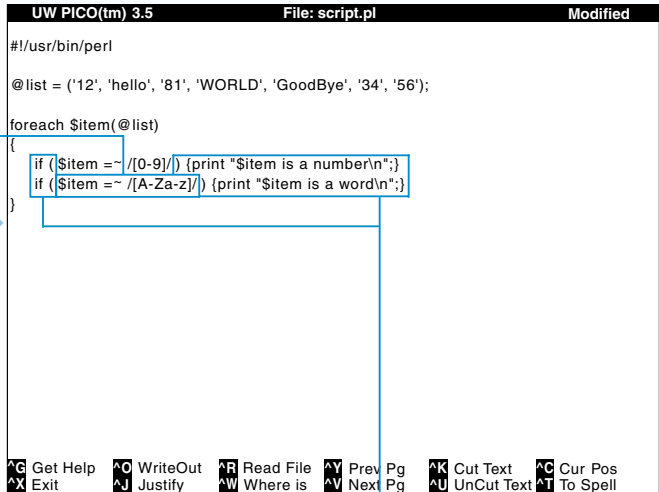
2 If you created an array in step 1, type the code that will process each item in the array.

3 Position the cursor where you want to use a character class in a regular expression and type the name of the variable you want to search followed by =~ /[]/.



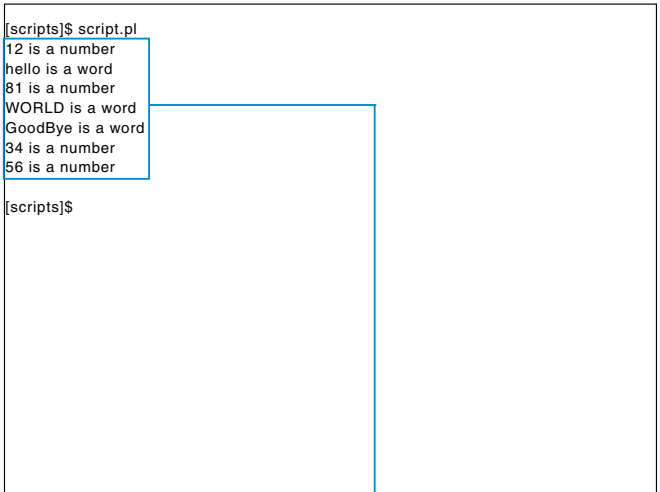
4 Position the cursor over the closing bracket (]) and type the character class you want to use.

Do not separate the characters in the class with commas or spaces.



5 Repeat steps 3 and 4 for each character class you want to use.

6 Type the code that uses the results of the regular expressions.



7 Save and execute the script.

Perl generates the results of using character classes in regular expressions.

USING QUANTIFIERS

Using a quantifier in a regular expression allows you to specify a number of times that a pattern must match in a string to be successful. A quantifier must appear immediately after the character or class you want to find. For information on classes, see page 126.

There are several quantifiers you can use. The quantifier you use will determine the number of times that a character or class you specify must repeat in a string. For example, the + quantifier will find an instance of a character

or class that is repeated one or more times. The ? quantifier will find an instance where the character or class is not repeated or is repeated only once.

To specify the exact number of times you want a character or class to match, use the {n} quantifier, where n is the number of times you want the character to repeat. For example, using the {3} quantifier for the character "x" will match either "xxx" or "xxxxxx" since both instances contain "x" repeated three times.

Extra

This chart displays quantifiers you can use to specify the number of times that a pattern must match to be successful.

QUANTIFIER:	DESCRIPTION:	EXAMPLE:
*	Matches zero or more instances.	/ba*d/ matches "abdomen" but not "abacus"
+	Matches one or more instances.	/ro+/ matches "troop" but not "transfer"
?	Matches zero or one instances.	/html?/ matches "html" but not "htd"
{n}	Matches "n" instances.	/B{2}/ matches "ABBA" but not "bush"
{n, }	Matches at least "n" instances.	/A{3, }/ matches "AAA" and "AAAAA"
{n, m}	Matches at least "n" and at most "m" instances.	/A{3, 5}/ matches "AAA" and "AAAA"

USING QUANTIFIERS

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

@list = ('12A', 'A41', '812', 'CCB', '1F2', '1A2', '2B3');

foreach $item (@list)
{
  if
  if
}
```

1 Type the code that creates an array or variable and assigns its value.

2 If you created an array in step 1, type the code that will process each item in the array.

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

@list = ('12A', 'A41', '812', 'CCB', '1F2', '1A2', '2B3');

foreach $item (@list)
{
  if ($item =~ /[0-9]{2}/)
  if
}
```

3 Type the name of the variable you want to search followed by =~ and the pattern delimiters //.

4 Position the cursor over the closing pattern delimiter (/) and type the character or class you want to search for.

5 Type the quantifier you want to use.

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

@list = ('12A', 'A41', '812', 'CCB', '1F2', '1A2', '2B3');

foreach $item (@list)
{
  if ($item =~ /[0-9]{2}/) {print "$item \t 2 or more numbers together\n";}
  if ($item =~ /C+/) {print "$item \t C followed by one or more Cs\n";}
}
```

6 Repeat steps 3 to 5 for each character or class you want to search for.

7 Type the code that uses the result of the regular expression(s).

```
[scripts]$ script.pl
12A  2 or more numbers together
A41  2 or more numbers together
812  2 or more numbers together
CCB  C followed by one or more Cs

[scripts]$
```

8 Save and execute the script.

Perl generates the result of using quantifiers in regular expressions.

USING THE MATCHING OPERATOR

`m/PATTERN/options` – Find a pattern in a string.

The matching operator (`m//`) is used to find a pattern in a string. For example, you can use the matching operator to find customers in a list whose last names begin with the letter "S".

Perl provides several special characters you can use in a regular expression to find a pattern in a string. For example, a period (`.`) will match any single character, except a newline, in a string. You can also use the asterisk character (`*`) to match any number of characters in a string. For example, `Mar*` will match both Mary and Martine.

Using the `=~` binding operator allows you to specify which variable stores the string you want to search. If you do not use the `=~` binding operator, the matching operator will use the value stored in the default special variable, `$_`.

If the matching operator finds a matching pattern, Perl will return a value of 1. This is useful if you want to use the matching operator with a statement modifier, such as an `if` statement. If the operator does not find a matching pattern, Perl returns a value of 0.

Extra

There are several options you can use with the matching operator to specify the way a string will be matched. The `g` option matches all occurrences of a pattern in a string, while the `i` option ignores the case of characters. You can use the `m` option to treat newline characters as line separators or use the `s` option to treat newline characters as characters in a string. Use the `o` option to avoid checking the values of variables each time a pattern is used or use the `x` option to ignore whitespace.

When you use the asterisk character (`*`) to find a matching pattern, Perl finds the longest match in a string. To make Perl find the smallest match in a string, add a question mark (`?`) immediately after the asterisk in your code. In this example, the asterisk finds all the text between the first `` tag and the last `` tag, unless a question mark is added to find the smallest possible match.

TYPE THIS:

```
$ = "HTML HTM XML";
if (m/html/i)
{
    print "Found HTML.\n";
}
```

RESULT:

Found HTML.

TYPE THIS:

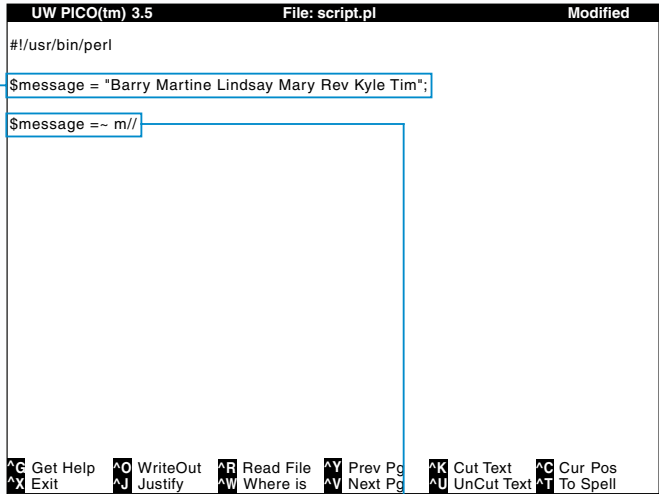
```
$_ = "Go to this <B>weather</B> site for the
latest <B>forecast</B>.";
m/(<B>.*</B>)/;
print "$1\n";

m/(<B>.*?</B>)/;
print $1;
```

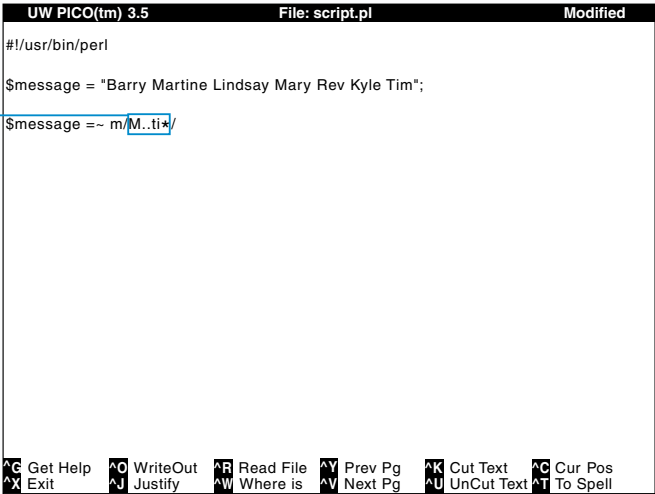
RESULT:

weather site for the latest forecast
weather

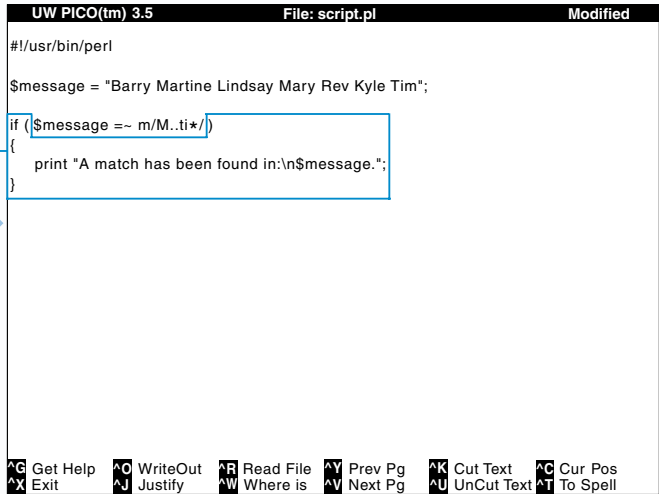
USING THE MATCHING OPERATOR



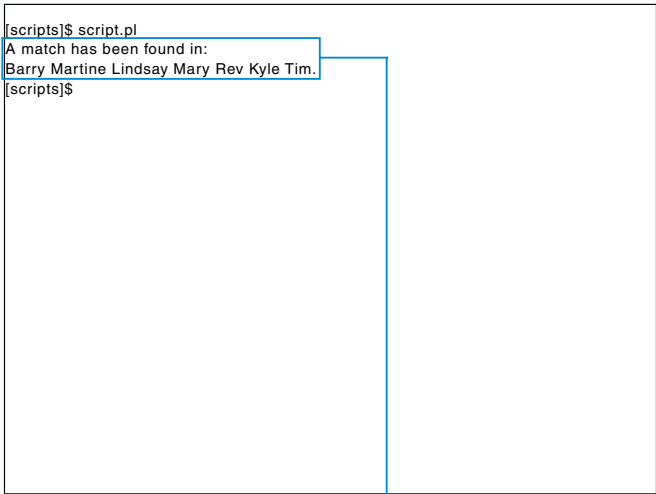
1 Type the code that creates a string variable and assigns it a value.



3 Position the cursor over the closing pattern delimiter (/) and type the pattern you want to search for.



4 Position the cursor where you want to type the code that generates the result of using the variable and type the code.



5 Save and execute the script.

Perl generates the result of using the matching operator to find a pattern in a string.

USING THE SUBSTITUTION OPERATOR

`s/OLD STRING/NEW STRING/options` – Replace a string.

The substitution operator `s///` allows you to replace a pattern in a string. The substitution operator requires two arguments—an old string and a new string. The old string is the pattern you want to replace with the new string. By default, the substitution operator replaces only the first occurrence of an old string.

Perl provides several options you can use to specify the way a string will be replaced. For example, the `g` option instructs Perl to replace all occurrences of an old string with a new string. You can also use multiple options at once. For example, specify `ig` to replace all occurrences of a pattern while ignoring the case of characters.

Using the `=~` binding operator allows you to specify which variable stores the string you want to replace. If you do not use the `=~` binding operator, the substitution operator will use the value stored in the default special variable, `$_`.

If an old string or new string contains forward slashes, you may want to change the pattern delimiters in the substitution operator to another character, such as `|`. This can make the code for the substitution operator easier to read. For example, you can write `s/c:/data/c:/files/` as `s|c:/data|c:/files|`.

Extra

There are several different options you can use with the substitution operator to specify the way a string will be replaced.

OPTION:	DESCRIPTION:
e	Evaluates a new string as an expression and uses the result as the new string.
g	Replaces all occurrences of the old string.
i	Ignores the case of characters.
m	Treats newline characters as line separators.
o	Does not check the values of variables each time a pattern is used, which speeds up the execution of the script.
s	Treats newline characters as characters in a string.
x	Ignores whitespace, which allows you to use spaces to make a pattern more readable.

USING THE SUBSTITUTION OPERATOR

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

$message = "This Web site contains html documents, such as index.html\n";
print $message;

$message =~ s///;
```

- 1 Type the code that creates an array or variable and assigns its value.
- 2 If you created an array in step 1, type the code that will process each item in the array.
- 3 To replace a string in the variable, type the name of the variable followed by `=~ s///;`.

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

$message = "This Web site contains html documents, such as index.html\n";
print $message;

$message =~ s/html/xml/;
```

- 4 Position the cursor over the second pattern delimiter (/) and type the old string you want to replace.
- 5 Position the cursor over the third pattern delimiter (/) and type the new string you want to add.

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

$message = "This Web site contains html documents, such as index.html\n";
print $message;

$message =~ s/html/xml/g;
print $message;
```

- 6 Position the cursor over the semi-colon and type the character for each option you want to use.
- 7 Type the code that generates the result of using the variable.

```
[scripts]$ script.pl
This Web site contains html documents, such as index.html
This Web site contains xml documents, such as index.xml

[scripts]$
```

- 8 Save and execute the script.
- Perl displays the results of using the substitution operator to replace a string.

USING THE TRANSLATION OPERATOR

`tr/MATCH LIST/REPLACEMENT LIST/option` – Modify characters in a string.

The translation operator `tr///` allows you to modify matched characters in a string. The translation operator requires two arguments—a match list and a replacement list. The match list contains one or more characters that you want to replace with the character(s) in the replacement list. You cannot use variables in the match list or replacement list.

When replacing multiple characters using the translation operator, Perl will replace each character you specify in the match list with the corresponding character in the replacement list. By default, if you specify more characters in the match list than in the replacement list, the last character in the

replacement list will replace all the extra characters in the match list.

Perl provides several options you can use to specify the way you want to modify characters using the translation operator. For example, the `d` option deletes characters you specify in the match list that do not have corresponding characters in the replacement list.

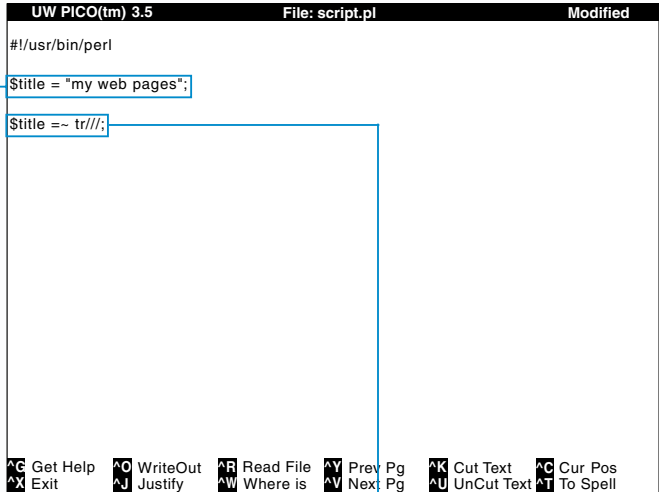
Using the `=~` binding operator allows you to specify which variable stores the string you want to modify. If you do not use the `=~` binding operator, the translation operator will use the value stored in the default special variable, `$_`.

Extra

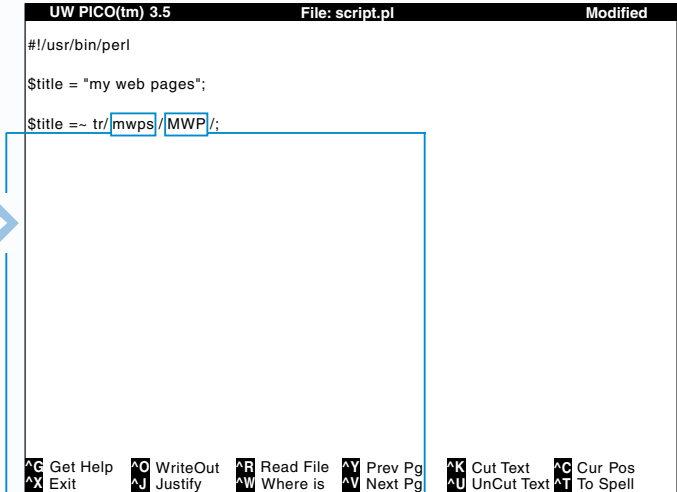
There are several options you can use with the translation operator to specify the way you want to modify characters in a string.

OPTION:	DESCRIPTION:	EXAMPLE:
c	Matches characters that do not appear in the match list.	<code>tr/aeiouAEIOU/./c</code> changes "Maureen" to ".au.ee."
d	Deletes characters in the match list that do not have corresponding characters in the replacement list.	<code>tr/pl/P/d</code> changes "perl" to "Per"
s	Deletes duplicate replaced characters.	<code>tr/d/b/s</code> changes "daddy" to "baby"

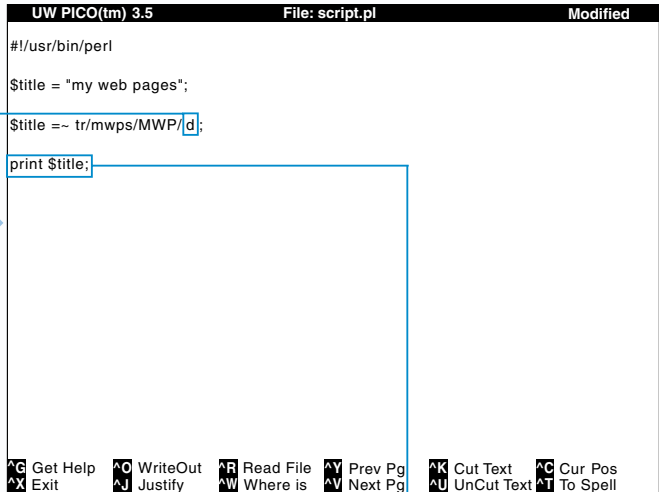
USING THE TRANSLATION OPERATOR



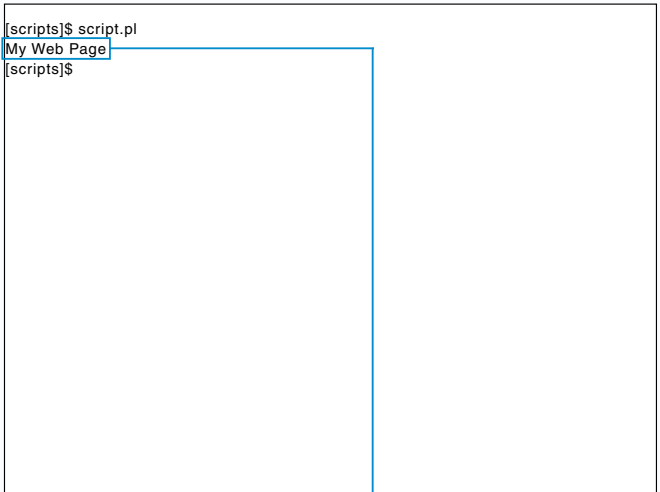
- 1
- Type the code that creates an array or variable and assigns its value.
- 2
- If you created an array in step 1, type the code that will process each item in the array.



- 3
- To modify characters stored in the variable, type the name of the variable followed by `=~ tr///;`.
- 4
- Position the cursor over the second pattern delimiter (/) and type each character you want to modify.
- 5
- Position the cursor over the third pattern delimiter (/) and type the characters you want to replace the characters you typed in step 4.



- 6
- To specify the way you want to modify the characters, position the cursor over the semi-colon and type the character for the option you want to use.
- 7
- Type the code that generates the result of using the variable.



- 8
- Save and execute the script.
- 9
- Perl generates the result of using the translation operator to modify a string.

USING BACKREFERENCES

When using a regular expression to find a pattern in a string, you can use a backreference to store the matched pattern so you can refer to the pattern later in your script.

You use the grouping operator () in a regular expression to enclose the pattern you want to store. You can use more than one grouping operator in a regular expression to store different parts of the same pattern.

When a match is found in a string, Perl assigns the matching pattern to a variable that consists of the \$ symbol followed by the number for the pattern.

Perl numbers the patterns by counting each left grouping operator from the beginning of the pattern, starting with 1. You use the variable to refer to the matched pattern in your script.

If you use multiple grouping operators, Perl will assign each matching pattern to a different variable. For example, the first matched pattern Perl finds is assigned to \$1, the second to \$2 and so on.

Keep in mind that if you want to find parentheses using a regular expression, you must prefix the parentheses with a backslash (\) to prevent Perl from interpreting the parentheses as the grouping operator.

Extra

Perl lets you refer to a matched pattern within the regular expression. To do so, you use a backslash (\) instead of the \$ symbol when referring to the matched pattern, such as \1. In the following example a backreference is used in a regular expression to check if each word in a list begins and ends with the same letter.

TYPE THIS:

```
@wordList = qw(sales candy modem recorder question);
foreach $word (@wordList)
{
    if ($word =~ /^([w]).*\1$/)
    {
        print "$word: Yes, $1\n";
    }
    else
    {
        print "$word: No\n";
    }
}
```

RESULT:

```
sales: Yes, s
candy: No
modem: Yes, m
recorder: Yes, r
question: No
```

A backreference can be used with the substitution operator (s///) to alter part of a string. This is useful when you want to alter text, such as uppercasing a letter, rather than replace the existing text with new text.

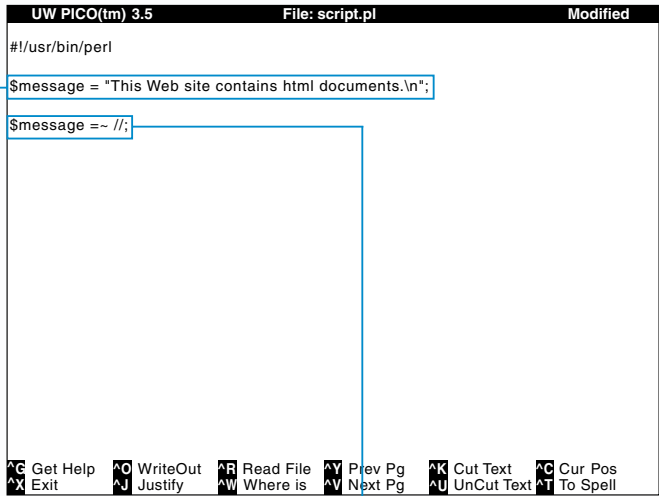
TYPE THIS:

```
$pet = "Fluffy";
$pet =~ s/^(.)($)/$1u$2/;
print $pet;
```

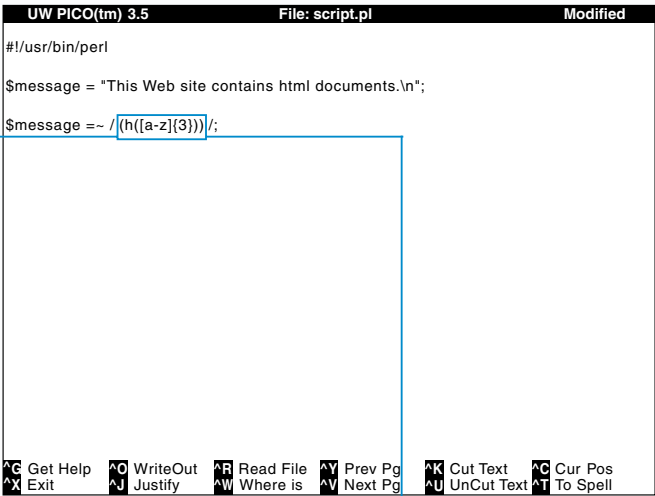
RESULT:

```
Fluffy
```

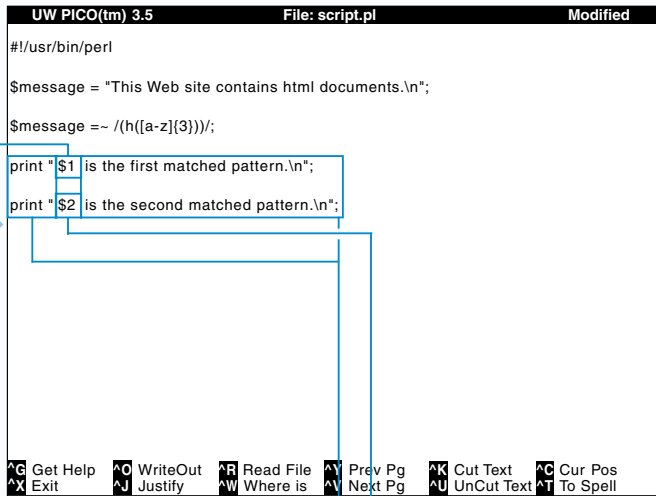
USING BACKREFERENCES



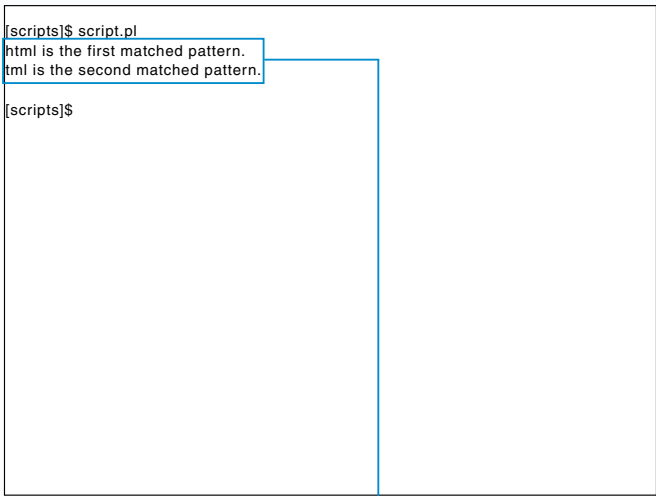
- 1 Type the code that creates a variable and assigns it a value.
- 2 Position the cursor where you want to create a regular expression and type the name of the variable you want to search followed by =~ and the pattern delimiters //.



- 3 Position the cursor over the closing pattern delimiter (/) and type the pattern you want to search for.
- 4 Enclose each part of the pattern you want to store in parentheses ().



- 5 Position the cursor where you want to refer to a matched pattern and type \$ followed by the number of the backreference you want to refer to.
- 6 Repeat step 5 for each backreference you want to refer to.
- 7 Position the cursor where you want to type the code that uses the backreferences and type the code.



- 8 Save and execute the script.
- Perl generates the results of using backreferences.