Perl Programming

Wikibooks.org

December 1, 2012

Contents

0.1 Section 1: Beginning Perl

0.2 Getting Started

This book assumes that you know absolutely nothing about programming at all and that Perl is your first language. However, basic operations such as making text files are outside of the realm of this tutorial.

0.2.1 Obtaining Perl

To find out if you already have Perl installed on your computer, go into the command line and type:

```
perl -v
```

This will display which version of Perl you have installed on your computer, if it is installed.

There are at least two easy ways to install Perl on Windows: the ActiveState[1] distribution, and the Strawberry Perl[2] distribution. Both are downloadable as native Windows installers. ActivePerl has a prebuilt package repository and is supported by a corporation, while Strawberry Perl includes a compiler (gcc) so that perl modules can be installed "on the fly" and is community-supported.

Most Unix-like operating systems will include Perl by default, and Linux Standard Base mandates that all compliant Linuxes ship with Perl installed. However, if for some reason you don't have perl, you can explore the options available to you at the main Perl download page[3], which will provide links to source and binaries.

0.2.2 Writing programs

A Sample Program

Perl is an *interpreted* language, which means you will always need the Perl interpreter which will *compile* and *execute* your program each time you run it. Instead of compiling your program into bytecode like in Pascal[4] or C++[5] and then executing it, you can simply copy your program's source code to a different computer (that has the perl interpreter) and run it.

For our first example, run your favorite text editor, and type something like this:

```
#!/usr/bin/perl
use strict;
use warnings;

print "Hello World";
```

If you don't understand this yet, don't worry; This will be explained in more depth later.

Save the file as **myprog.pl** and you have a perl program ready to run.

0.2.3 Running programs

Windows

To run a perl program with a modern version of ActivePerl[6] installed, you simply click on it. If the screen flashes and you can't see the output you might have to execute the file from within the windows shell (ie. cmd.exe or PowerShell). With Strawberry Perl[7], you'll have to execute a perl program from the command line as shown below.

[1] http://activestate.com/Products/ActivePerl/?_x=1
[2] http://strawberryperl.com
[3] http://www.perl.com/download.csp
[4] http://en.wikibooks.org/wiki/Pascal
[5] http://en.wikibooks.org/wiki/C%2B%2B
[6] http://www.activestate.com/Products/Download/Download.plex?id=ActivePerl
[7] http://strawberryperl.com/

From a windows command-line interface, you can run the program thusly:

```
C:\> perl path\to\foo\myprog.pl
```

or, if perl.exe is not in your path:

```
C:\> c:\perl\bin\perl.exe myprog.pl
```

Note: You may have to specify the full path to your program unless you are running the command prompt in that directory.

UNIX-like Systems

You can run a perl program by running perl itself, and telling the shell the name of the file:

```
perl myprog.pl
```

Usually, perl programs are made executable on their own. This involves two changes to the sample program. First, edit it and put the following shebang line[8] at the top of the file:

```
#!/usr/bin/perl
```

Then, at a command prompt, make your program executable by using chmod.

```
chmod +x myprog.pl
```

Your program is now executable and ready to run, just like any other file. To execute, type:

```
./myprog.pl
```

By convention, .pl identifies a perl script, and .pm a perl library. The .pl file extension isn't needed for either of these examples; it's just a useful way of identifying files. The only time the convention *should* be violated is if the program is to be installed outside of the current working directory, and there runs a chance you might want to some day rewrite them in a different language.

0.3 A First Taste of Perl

Here's a simple program written in Perl to get us started:

```
#!/usr/bin/perl

# Outputs Hello world to the screen.

print "Hello world!\n";
```

[8] http://en.wikipedia.org/wiki/Shebang%20%28Unix%29

Let's take a look at this program line by line:

- `#!/usr/bin/perl`

  On Unix systems this tells the Operating System to execute this file with the program located at `/usr/bin/perl`. This is the default Unix location for the perl interpreter, on Windows `#!C:\Perl\bin\perl.exe` or `#!C:\strawberry\perl\bin\perl.exe` (depending on whether ActivePerl or Strawberry Perl was installed) should be used instead.

  Shebang: A line at the start of a file, beginning with #!, that gives instructions to the operating system.

- `# Outputs ...`

  This line is a comment - it is ignored by the perl interpreter, but is very useful. It helps you to debug and maintain your code, and explain it to other programmers.

  Comment: A line of plain text ignored by the interpreter in a file of code.

- `print "Hello world!\n";`

  The **print** instruction writes whatever follows it to the screen. The `\n` at the end of the *string* puts a new line to the screen. The semicolon at the end of the line tells the perl interpreter that the instruction is finished; you must put a semicolon at the end of **every** instruction in Perl code.

  String: A sequence of characters used as data by a program.

0.3.1 Exercises

- Change the program so it says hello to *you*.
- Change the program so that after greeting you, it asks how you are doing, on the next line. The output should look like this:

Hello *your_name*! How are you?

- Experiment with the `\n` character, what happens when you take it away? What happens if you put two in a row?

Remember: if you add another `print` instruction you will need to put a semicolon after it.

0.4 Strings

Any sequence of characters put together as one unit, is a string. So, the word `the` is a string. This sentence is a string. Even this entire paragraph is a string. In fact, you could consider the text of this entire book as one string.

Strings can be of any length and can contain any characters, numbers, punctuation, special characters (like *!* *#*, and *%*), and even characters in natural languages besides English. In addition, a string can contain special whitespace formatting characters like newline, tab, and the *bell* character. We will discuss special characters more later on. For now, we will begin our consideration of strings by considering how to insert literal strings into a Perl program.

To begin our discussion of strings in Perl, we will consider how to work with *string literals* in Perl. The word **literal** here refers to the fact that these are used when you want to type a string directly to Perl. This can be contrasted with storing a string in a **variable**.

Any string literal can be used as an expression. We will find this useful when we want to store string literals in variables. However, for now, we will simply consider the different types of string literals that one can make in Perl. Later, we will learn how to assign these string literals to variables in the Scalar Variables section[9].

0.5 Single Quoted Strings

String literals can be represented in primarily three ways in Perl. We have already used one type in the simple programming examples, using double quote marks. Using double or single quote marks in Perl each has a special meaning.

Single quotes can be thought of as literal strings. In the previous examples, you may have noticed that variable names were included inside the strings with double quotes. When the results were printed, the value of the variable was placed in the printed line, not the name of the variable. If single quote marks were used, the actual variable name would have been printed because nearly all special characters that might be interpreted differently are taken at *face value* when using single quotes.

To see what is meant by this, try this simple program:

```
my $name = "Fred";
print "Hello $name\n";
print ,Hello $name\n,;
```

You should see "Hello Fred" on the first line and "Hello $name\n" on the second (without a newline after it). Putting the *value* of `$name` into the string in the first print statement is called "interpolation." If you don't need interpolation, you should use single quotes, because it makes your intent clearer.

0.5.1 Special Characters in Single-quoted Strings

There are two characters in single quoted strings that do not always represent themselves. This is due to necessity, since single-quoted strings start and end with the `'` character. We need a way to express inside a single-quoted string that we want the string to contain a `'` character.

The solution to this problem is to preceded any `'` characters we actually want to appear in the string itself with the backslash (\ character). Thus we have strings like this:

```
 ,xxx\,xxx,;  # xxx, a single-quote character, and then xxx
```

We have in this example a string with 7 characters exactly. Namely, this is the string: `xxx'xxx`. It can be difficult at first to become accustomed to the idea that two characters in the input to Perl

[9] Chapter 2.12.2 on page 26

actually produce only one character in the string itself. (C programmers are already probably used to this idea.) However, just keep in mind the rules and you will probably get used to them quickly.

Since we have used the \ character to do something special with the `'` character, we must now worry about the special cases for the backslash character itself. When we see a \ character in a single-quoted string, we must carefully consider what will happen.

Under most circumstances, when a \ is in a single-quoted string, it is simply a backslash, representing itself, as most other characters do. However, the following exceptions apply:

- The sequence `\'` yields the character `'` in the actual string. (This is the exception we already discussed above).
- The sequence `\\` yields the character \ in the actual string. In other words, two backslashes right next to each other actually yield only one backslash.
- A backslash, by itself, cannot be placed at the end of a the single-quoted string. This cannot happen because Perl will think that you are using the \ to escape the closing `'`.

The following examples exemplify the various exceptions, and use them properly:

```
  ,I don\,t think so.,;          # Note the , inside is escaped with
 \
  ,Need a \\ (backslash) or \?,; # The \\ gives us \, as does \
  ,You can do this: \\,;         # A single backslash at the end
  ,Three \\\,s: "\\\\\\",;       # There are three \ chars between ""
```

In the last example, note that the resulting string is `Three \'s:  "\\\"`. If you can follow that example, you have definitely mastered how single-quoted strings work!

Instead of unreadable backslash escapes, Perl offers other ways[10] of quoting strings. The first example above could be written as:

```
  q{I don,t think so};            # No \ needed to escape the ,
```

0.5.2 Newlines in Single-quoted Strings

Note that there is no rule against having a single-quoted string span several lines. When you do this, the string has *newline* characters embedded in it.

A newline character is a special ASCII character that indicates that a new line should be started. In a text editor, or when printing output to the screen, this usually indicates that the cursor should move from the end of the current line to the first position on the line following it.

Since Perl permits the placement of these newline characters directly into single quoted strings, we are permitted to do the following:

```
  ,Time to
  start anew.,;   # Represents the single string composed of:
                  # ,Time to, followed by a newline, followed by
                  # ,start anew.,
```

[10] http://perldoc.perl.org/perlop.html#Quote-and-Quote-like-Operators

This string has a total of twenty characters. The first seven are `Time to`. The next character following that is a newline. Then, the eleven characters, `start anew.` follow. Note again that this is **one string**, with a newline as its eighth character.

Further, note that we are not permitted to put a comment in the middle of the string, even though we are usually allowed to place a # anywhere on the line and have the rest of the line be a comment. We cannot do this here, since we have yet to terminate our single-quoted string with a `'`, and thus, any # character and comment following it would actually become part of the single-quoted string! Remember that single-quotes strings are delimited by `'` at the beginning, and `'` at the end, and everything in between is considered part of the string, included newlines, # characters and anything else.

0.5.3 Examples of Invalid Single-quoted Strings

In finishing our discussion of singled-quoted strings, consider these examples of strings that are **not** legal because they violate the exceptions we talked about above:

```
  ,You cannot do this: \,; # INVALID: the ending \ cannot be alone
  ,It is 5 o,clock!,       # INVALID: the , in o,clock should be
 escaped
  ,Three \\\,s: \\\\\,;      # INVALID: the final \ escapes the ,,
 thus
                           #          the literal is  not terminated
  ,This is my string;      # INVALID: missing close quote
```

Sometimes, when you have invalid string literals such as in the example above, the error message that Perl gives is not particularly intuitive. However, when you see error messages such as:

```
(Might be a runaway multi-line string starting on line X)
Bareword found where operator expected
Bareword "foo" not allowed while "strict subs" in use
```

It is often an indication that you have runaway or invalid strings. Keep an eye out for these problems. Chances are, you will forget and violate one of the rules for single-quoted strings eventually, and then need to determine why you are unable to run your Perl program.

0.6 Brief Digression from Strings Alone: The print Function

Before we move on to our consideration of double-quoted strings, it is necessary to first consider a small digression. We know how to represent strings in Perl, but, as you may have noticed, the examples we have given thus far do not do anything interesting. If you try placing the statements that we listed as examples in Single Quoted Strings[11], into a full Perl program, like this:

```
#!/usr/bin/perl

use strict;
```

[11] Chapter 0.5 on page 5

```
 use warnings;

 'Three \\\'s: "\\\\\"'; # There are three \ chars between ""
 'xxx\'xxx';             # xxx, a single-quote character, and then
xxx
 'Time to
 start anew.';
```

you probably noticed that nothing of interest happens. Perl gladly runs this program, but it produces no output.

Thus, to begin to work with strings in Perl beyond simple hypothetical considerations, we need a way to have Perl display our strings for us. The canonical way of accomplishing this in Perl is to use the `print` function.

The `print` function in Perl can be used in a variety of ways. The simplest form is to use the statement `print STRING;`, where `STRING` is any valid Perl string.

So, to reconsider our examples, instead of simply listing the strings, we could instead print each one out:

```
#!/usr/bin/perl

use strict;
use warnings;

print 'Three \\\'s: "\\\\\"'; # Print first string
print 'xxx\'xxx';             # Print the second
print 'Time to
start anew.
';    # Print last string, with a newline at the end
```

This program will produce output. When run, the output goes to what is called the *standard output*. This is usually the terminal, console or window in which you run the Perl program. In the case of the program above, the output to the standard output is as follows:

```
Three \'s: "\\\"xxx'xxxTime to
start anew.
```

Note that a newline is required to break up the lines. Thus, you need to put a newline at the end of every valid string if you want your string to be the last thing on that line in the output.

Note that it is particularly important to put a newline on the end of the last string of your output. If you do not, often times, the command prompt for the command interpreter that you are using may run together with your last line of output, and this can be very disorienting. So, **always** remember to place a newline at the end of each line, particularly on your last line of output.

Finally, you may have noticed that formatting your code with newlines in the middle of single-quoted strings hurts readability. Since you are inside a single-quoted string, you cannot change the format of the continued lines within the print statement, nor put comments at the ends of those lines because that would insert data into your single-quoted strings. To handle newlines more elegantly, you should use double-quoted strings, which are the topic of the next section.

0.7 Double Quoted Strings

Double-quoted strings are another way of representing scalar string literals in Perl. Like single-quoted strings, you place a group of ASCII characters between two delimiters (in this case, our delimiter is `"`). However, something called *interpolation* happens when you use a double-quoted string.

0.7.1 Interpolation in Double-quoted Strings

Interpolation is a special process whereby certain special strings written in ASCII are replaced by something different. In Single-quoted strings section[12], we noted that certain sequences in single-quoted strings (namely, `\\` and `\'`) were treated differently - these are called *backslash escape sequences*. This is very similar to what happens with interpolation.

For example, in interpolated double-quoted strings, various sequences preceded by a \ character act differently according to the chart below:

String	Interpolated As
`\\`	an actual, single backslash character
`\$`	a single $ character
`\@`	a single @ character
`\"`	a single double-quote character
`\t`	tab
`\n`	newline
`\r`	hard return
`\f`	form feed
`\b`	backspace
`\a`	alarm (bell)
`\e`	escape
`\056`	character represented by octal value, 056 (same as `.`)
`\x2E`	character represented by hexadecimal value, 2E (same as `.`)

As you may have noticed in the previous chapter, you can put the name of a variable within a string with its leading dollar sign. This form of interpolation replaces the name of the variable in the string with the content of the variable.

0.7.2 Examples of Interpolation

Let us consider an example that uses a few of these characters:

```
#!/usr/bin/perl

use strict;
use warnings;
```

[12] Chapter 0.5 on page 5

```
    print "A backslash: \\\n";
    print "Tab follows:\tover here\n";
    print "Ring! \a\n";
    print "Please pay someone\@example.org \$20.\n";
```

This program, when run, produces the following output on the screen:

```
    A backslash: \
    Tab follows:    over here
    Ring!
    Please pay someone@example.org $20.
```

In addition, when running, you should hear the computer beep. That is the output of the `\a` character, which you cannot see on the screen. However, you should be able to hear it.

Notice that the `\n` character ends a line. `\n` should always be used to end a line. Those students familiar with the C language will be used to using this sequence to mean *newline*. When writing Perl, the word *newline* and the `\n` character are roughly synonymous.

0.7.3 String Operators

Operators manipulate two or more strings in some way.

The Concatenation Operator

Perl uses the `.` operator to concatenate or connect two strings together, like this:

```
   "Hello" . "World"  # This is the same as "HelloWorld"
```

If you want to make the string have a space between Hello and World you could write it like this:

```
   "Hello" . " " . "World"  # This is the same as "Hello World"
```

Or like this:

```
   "Hello" . " World" # This is the same as "Hello World"
```

The `x` Operator

This is called the *string repetition* operator and is used to repeat a string. All you have to do is put a string on the left side of the `x` and a number on the right side. Like this:

```
"Hello" x 5 # This is the same as "HelloHelloHelloHelloHello"
```

If you wish to insert a line break after each output of the string, use:

```
"Hello\n" x 5
```

0.7.4 Exercises

- Write a program that uses the `.` operator to print "Hello Sir!".
- Write another program which uses the `x` operator to print "HelloHelloHelloHello". Put comments in this program that explain how it works
- Remember to take some time to play with single and double quoted strings, the more practice you get, the better you will be.

0.8 Numbers

Numbers in Perl do not have to be enclosed in any kind of punctuation; they can be written as straight numbers.

0.8.1 Floating Point Numbers

Here are some acceptable floating point numbers:

0.1, -3.14, 2.71828...

0.8.2 Integers

Integers are all whole numbers and their negatives (and 0): {... -3, -2, -1, 0, 1, 2, 3 ...}.
Here are a few examples of integers:

```
12, -50, 20, 185, -6654, 6654
```

The following examples are **not** integers:

```
15.5, -3.458, 3/2, 0.5
```

0.8.3 Non-decimal Numbers

I'll dwell on this topic for a little longer than the other types of numbers. In Perl you can specify not only decimal numbers, but also numbers in hex, octal, and binary. If you are not familiar with how these systems work, you can try these Wikipedia articles:

- Hexadecimal[13]
- Octal[14]
- Binary[15]

In Perl you have to specify when you are going to write a non-decimal number. Binary numbers start with an 0b, so here are some possible binary numbers:

```
0b101011101

0b10
```

Octal numbers start with 0 ("zero"), so here are some possible octal numbers:

```
015462

062657

012
```

Hexadecimal numbers start with 0x, so here are some possible hexadecimal numbers:

```
0xF17A

0xFFFF
```

0.8.4 Number Operators

Just like strings, numbers have operators. These operators are quite obvious so I'll just give a quick example of each one.

The +, - , /, and * Operators

These operators are pretty obvious, but here are some examples:

```
100 + 1 # That's 101
100 - 1 # That's 99
100 / 2 # That's 50
100 * 2 # That's 200
```

[13] http://en.wikipedia.org/wiki/Hexadecimal
[14] http://en.wikipedia.org/wiki/Octal
[15] http://en.wikipedia.org/wiki/Binary_numeral_system

Perl also has the familiar increment, decrement, plus-equals, and minus-equals operators from C:

```
$a++    # evaluate, then increment
++$a    # increment, then evaluate
$a--    # evaluate, then decrement
--$a    # decrement, then evaluate
$a += 5 # plus-equals operator, adds 5 to $a. Equivalent to $a = $a
+ 5
$a -= 2 # minus-equals operator, subtracts 2 from $a. Equivalent to
$a = $a-2
```

Now let's look at one more operator that's a little less obvious.

The ** Operator

The `**` operator is simply the exponentation operator. Here's another example:

```
2**4  # That's 16, same as $2^4$
4**3**2 # that's 4**(3**2), or $4^9$, or 262144
```

Extra!
The modulus operator (`%`) can be used to find the remainder when dividing two numbers.
If that doesn't make sense now, that's fine, it's not that important.(Note, this returns 0 when used on floating point numbers)

0.8.5 Exercises

- Remember the `x` operator? Use a mathematical expression as the number of times to repeat the string, see what happens.
- Write a program like our original hello world program except make it print a mathematical expression.

In Perl, there are five types of variables: $calars, @rrays, %hashes, &subroutines, and *typeglobs.

0.9 Simple variables

Variables, called scalars, are identified with the $ character, and can contain nearly any type of data. For example:

```
$my_variable = 3;                              # integers
$my_variable = 3.1415926;                      # floating point
$my_variable = 3.40282366920938463463e+38;     # exponents
$my_variable = $another_variable + 1;          # mathematical
 operation
$my_variable = ,Can contain text,;             # strings
$my_variable = \$another_variable;             # scalar reference
$my_variable = \@array_variable;               # array reference

print $my_variable;
```

0.9.1 Case sensitivity

Note that the perl interpreter is case sensitive[16]. This means that identifier names containing lowercase letters will be treated as being different and separate from those containing uppercase letters.

0.10 Arrays

Arrays in Perl use the @ character to identify themselves.

```
@my_array = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);      # numeric list
@my_array = (1 .. 10);                            # same as above
@my_array = (,John,, ,Paul,, ,Kanai,, ,Mahenge,); # strings
@my_array = qw/John Paul Kanai Mahenge/;           # the same -
 one-word strings, with less typing
@my_array = qw/red blue 1 green 5/;               # mixed types
@my_array = (\@Array1, \@Array2, \@Array3);       # array of arrays

foreach my $Item (@my_array) {
    print "Next item is $Item \n";
}
```

However, when you deal with just one element of the array (using square brackets so it's not confused), then that element of the array is considered a scalar which takes the $ sigil:

```
$my_array[0] = 1;
```

As in the C programming language, the number of the first element is 0 (although as with all things in Perl, it's possible to change this if you want). Array subscripts can also use variables:

```
$my_array[$MyNumber] = 1;
```

0.11 Associative arrays

Associative arrays, or "hashes," use the % character to identify themselves.

```
%my_hash = (,key1, => ,value1,, ,key2, => ,value2,);
```

When using the => the left side is assumed to be quoted. For long lists, lining up keys and values aids readability.

```
%my_hash = (
    key1    => ,value1,,
    key2    => ,value2,,
    key3    => ,value3,,
);
```

However, when you deal with just one element of the array (using braces), then that element of the array is considered a scalar and takes the $ identifier:

[16] http://en.wikibooks.org/wiki/%2Fcase%20sensitivity%2F

```
$my_hash{,key1,} = ,value1,;
```

Associative arrays are useful when you want to refer to the items by their names.

0.12 Subroutines

Subroutines are defined by the `sub` function, and used to be called using `&` (using `&` is now deprecated). Here's an example program that calculates the Fibonnaci sequence:

```
sub fib {
    my $n = shift;
    return $n if $n < 2;
    return fib( $n - 1 ) + fib( $n - 2 );
}

print fib(14);
```

The `if` statement is the primary conditional structure in Perl. The syntax is as follows:

```
if (,,boolean expression,,) {
    ,,statement,,;
}
```

If the *boolean expression* evaluates to true, the statements between the two braces will be executed. The braces around statements are mandatory, even if there is only one statement (unlike C or Java).

An alternative syntax to the `if` statement may be used on a single statement. This involves putting the conditional at the end of the statement rather than before, and does not include braces:

```
,,statement,, if (,,boolean expression,,) ;
```

The following statements are synonymous:

```
if ( $x == 20 ) { print "hello"; }
print "hello" if ( $x == 20 );
```

You should choose whichever one is clearer in a given situation. For example, the following is legal, but unclear:

```
foreach my $word (@words) {
    if ($word eq ,end,) { last; }
    print "$word\n";
}
```

This *hides* the `last` (which is like `break`, and ends the loop) over at the right. Instead use a postfix `if`:

```
foreach my $word (@words) {
    last if $word eq ,end,;
    print "$word\n";
}
```

The *boolean expression* conditional can contain any one of the comparison operators covered in the next section.

Multiple conditions can be checked together using the boolean expression operators:

- `&&` - logical and, C style; used for most conditionals
- `and` - logical and, but with a lower precedence; used for flow control
- `||` - logical or, C style; used for most conditionals
- `or` - logical or, but with a lower precedence; used for flow control
- `!` - logical not, C style
- `not` - logical not, but with a lower precedence

```
if ( ($x == 20) || ( ($x > 0)&&($x < 10)&& !($x == 5) ) ){
    print "x is equal to 20 or either between 0 and 10 but not 5.\n";
}
```

Conditional statements can also be extended with the `elsif` and `else` structures:

```
 if (,,boolean expression 1,,) {
     ,,statement 1;,,
 }
 elsif (,,boolean expression 2,,) {
     ,,statement 2;,,
 }
 else {
     ,,statement 3;,,
 }
```

Note that an `if` statement is followed by any number (including zero) of `elsif` statements, and finally an optional `else` statement. The statements of an `elsif` will be executed if its boolean expression is true, and no preceding `(els)if` statement's boolean expression is true. The trailing `else` (if present) is executed if none of the preceding statements' boolean expressions are true.

1 Introduction

Perl's set of operators borrows extensively from the C programming language[1]. Perl expands on this by infusing new operators for string functions (`.=`, `x`, `eq`, `ne`, etc.). C by contrast delegates its subset of Perl functionality to a library *strings.h*, and *ctype.h*, and includes no such functionality by default compilation. Perl also includes a highly flexible Regex[2] engine inspired by Sed with improvements to standard POSIX regexes, most notably the support of Unicode.

[1] `http://en.wikibooks.org/wiki/C%20programming%20language`

[2] Chapter 4.24 on page 75

2 The operators

2.1 Arithmetic

Most arithmetic operators are *binary* operators; this means they take two arguments. *Unary* operators only take one argument. Arithmetic operators are very simple and often transparent.

2.1.1 Binary

All the basic arithmetic operators are present: addition (+), subtraction (−), multiplication (*), and division (/).

The modulus operator is %. Modulus returns the remainder of a division (/) operation.

```
# 3 goes into 4, 1 time  with 1 left over.
print 4 % 3;   # prints 1

# 2 goes into 4, 2 times with 0 left over.
print 4 % 2;   # prints 0

# 3 goes into -4, -2 times with 2 left over.
print -4 % 3;   # prints 2
```

The exponentiation operator is **. It allows you to raise one value to the power of another. If you raise to a fraction you will get the root of the number. In this example the second result when raised to the power of 2 should return 2 (`( 2 ** (1/2) ) ** 2 = 2`).

```
# Four squared:
print 4 ** 2; # prints 16

# Square root of 2
print 2 ** (1/2); # prints 1.4142135623731
```

The function `sqrt` is provided for finding a Square Root. Other fractional powers (i.e., (1/5), (2/13), (7/5), and similar) are suitably found using the ** operator.

2.1.2 Unary

The auto-decrement (−−), and auto-increment (++) operators are unary operators. They alter the scalar variable they operate on by one logical unit. On numbers, they add or subtract one. On letters and strings, only the auto-increment shift one up in the alphabet, with the added ability to roll-over. Operators that come in post- and pre- varieties can be used two ways. The first way returns the value of the variable *before* it was altered, and the second way returns the value of the variable *after* it was altered.

```
my $foo = 1;

# post decrement (printed and then decremented to 0)
print $foo--; # prints 1
print $foo;   # prints 0
```

```
my $foo = 1;

# pre-decrement (decremented to 0 then printed)
print --$foo;  # prints 0
print $foo;    # prints 0
```

```
my $foo = ,d,;

# pre-increment (incremented to e then printed)
print ++$foo;  # prints e
print $foo;    # prints e
```

```
my $foo = ,Z,;

# post-increment (printed the incremented to AA)
print $foo++;  # prints Z
print $foo;    # prints AA
```

2.2 Assignment

The basic assignment operator is "=" which sets the value on the left side to be equal to the value on the right side. It also returns the value. Thus you can do things like $a = 5 + ($b = 6), which will set $b to a value of 6 and $a to a value of 11 (5 + 6). Why you would want to do this is another question.

The assignment update operators from C, "+=", "-=", etc. work in perl. Perl expands on this basic idea to encompass most of the binary operators in perl.

operator	**name**
+=	add assign, plus-equals
subtract assign, minus-equals	
*=	multiply assign
/=	divide assign
%=	modulo assign
**=	exponent assign
.=	concatenate assign
repeat assign	
&&=	logical AND assign
||=	logical OR assign
&=	bitwise AND assign
|=	bitwise OR assign
ˆ=	bitwise XOR assign
˜=	bitwise NOT assign
<<=	left shift assign
>>=	right shift assign

```
my $foo = ,Hello,;
$foo .= ,, world,;
print $foo; # prints ,Hello, world,;

my $bar = ,+,;
$bar x= 6;
print $bar; # prints ,++++++,;
```

2.3 Comparison

Perl uses different operators to compare numbers and strings. This is done because in most cases, Perl will happily *stringify* numbers and *numify* strings. In most cases this helps, and is consistent with Perl's *DWIM Do-What-I-Mean* theme. Unfortunately, one place this often does not help, is comparison.

name	**numeric**	**string**
equal	==	eq
not equal	!=	ne
less than	<	lt
greater than	>	gt
less or equal	<=	le
greater or equal	>=	ge
compare	<=>	cmp

2.4 Logical

Perl has two sets of logical operators, just like the comparison operators, however not for the same reason.

The first set (sometimes referred to as the C-style logical operators because they are borrowed from C) is &&, `||`, and `!`. They mean logical AND, OR, and NOT respectively. The second set is `and`, `or`, and `not`.

The only difference between these two sets is the precedence they take (See Precedence[1]). The symbolic operators take a much higher precedence than the textual.

2.4.1 Conditionals

Most of the time, you will be using logical operators in conditionals.

```
# Only prints "I like cookies\n" if both $a is 5 and $b is 2
if($a == 5 && $b == 2){
    print "I like cookies\n";
}
```

[1] Chapter 2.10 on page 24

In this case, you could safely substitute `and` for `&&` and the conditional would still work as expected, however, this is not always the case.

```
#True if $a is 5, and either $b, $c, or both are 2
if($a == 5 and $b == 2 || $c == 2){
    print "I like cookies\n";
}
#Using brackets, the order is made more clear.
#This conditional acts in the same way as the last.
if($a == 5 and ($b == 2 || $c == 3)){
    print "I like cookies\n";
}
```

This, however, is completely different.

```
if($a == 5 && $b == 2 or $c == 3){
    print "I like cookies\n";
}
#Equivalent and easier to understand with brackets
if(($a == 5 && $b == 2) or $c == 3){
    print "I like cookies\n";
}
```

Most people prefer to use C-style logical operators and use brackets to enforce clarity rather than using a combination of textual and C-style operators (when possible), which can be very confusing at times.

2.4.2 Partial evaluation

Partial evaluation (or "short circuiting") is the property of logical operators that the second expression is only evaluated if it needs to be.

```
($a, $b) = (5, 2);
#$b < 3 is not evaluated at all because when the interpreter
#finds that $a == 4 is false, there is no need to evaluate $b < 3
#because the conditional is automatically false
if($a == 4 && $b < 3){
    print "I like cookies\n";
}
```

This also works with logical OR statements. If the first expression evaluates as true, then the second is never evaluated because the conditional is automatically true.

This becomes useful in a case like this:

```
sub foo {
#returns a true or false value
}
foo() or print "foo() failed\n";
```

Here, if the foo() subroutine returns false, then "foo() failed\n" is printed. However, if it returns true, then "foo() failed\n" is not printed, because the second expression (`print "foo() failed\n"`) does not need to be evaluated.

2.5 Bitwise

These operators perform the same operation as the logical operators, but instead of being performed on the true/false value of the entire expressions, it is done on the individual respective bits of their values.

- "&" (bitwise AND)
- "|" (bitwise OR)
- "^" (bitwise XOR)
- "~" (bitwise NOT)

The left and right shift operators move the bits of the left operand (e.g. $a in the case of $a << $b) left or right a number of times equal to the right operand ($b). Each move to the right or left effectively halves or doubles the number, except where bits are shifted off the left or right sides. For example, $number << 3 returns $number multiplied by 8 (2**3).

- "<<" (left shift)
- ">>" (right shift)

2.6 String

The string concatenation operator is `.`, not + which some other languages use.

```
print ,Hello, . , world,; # prints "Hello world" without a newline at
 the end
```

There is a repeat operator for strings (x) which repeats a string a given number of times.

```
my $str = "hi";
my $repeated_str = $str x 5;
print "$repeated_str\n"; # prints "hihihihihi" with a newline at the
 end
```

2.7 Comparing strings

To compare strings, use `eq` and `ne` instead of == or `!=` respectively. You can also look for a substring with `substr()`, or pattern-match with regular expressions.

2.8 File Test

See Perl Programming/Function Reference#-X[2]

[2] Chapter 4.53.1 on page 109

2.9 Other

The range operator (..) returns a list of items in the range between two items; the items can be characters or numbers. The type of character is determined by the *first* operand; the code:

```
print ('A'..'Z');
print ('a'..'z');
print ('A'..'z');
print (1..'a');
print (1..20);
print ('&'..'!');
print (10..-10);
print "$_\n" foreach (1..10);
```

Outputs (Newlines added for readability):

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567891011121314151617181920
&

1
2
3
4
5
6
7
8
9
10
```

Note that the case is defined by the first operand, and that the `1..'a'` and `(10..-10)` operations don't return anything.

2.10 Precedence

Precedence is a concept that will be familiar to anyone who has studied algebra or coded in C/C++. Each operator has its place in a hierarchy of operators, and are executed in order. The precedence of perl operators is strict and should be overridden with parentheses, both when you are knowingly going against precedence and when you aren't sure of the order of precedence. For a complete listing of the order, check perlop[3].

[3] http://www.perl.com/doc/manual/html/pod/perlop.html#SYNOPSIS

2.11 The smart match operator (˜˜)

The **smart match operator** is new in perl 5.10. To use it, you'll need to explicitly say that you're writing code for perl 5.10 or newer:

```
#!/usr/bin/perl
use strict;
use warnings;
use 5.10.0; # We will be using the smart match operator

my $scalar = ,hi,;
my @array = qw(one two three);
my %hash = (
    hi  => 1,
    ho  => 2,
    he  => 3,
);

if ($scalar ~~ @array) { print "1\n"; } # Doesn,t print; ,hi, isn,t
 an element in @array
if ($scalar ~~ %hash)  { print "2\n"; } # Does print; ,hi, is a key
 in %hash
if (@array ~~ %hash)   { print "3\n"; } # Doesn,t print; none of the
 elements of @array match a key in %hash
```

The smart match operator is versatile and *fast* (often faster than the equivalent comparison without ˜˜). See smart matching in detail[4] for the comparisons it can do. ˜˜ is also used in the `given/when` switch statement new in 5.10, which will be covered elsewhere.

2.12 Dereferencing

2.12.1 The doubledollar

A variable, previously referenced with the reference operator can be dereferenced by using a doubledollar symbol prefix:

```
$number = 12;
$refnum = \$number; # backslash is the reference operator
$$refnum = 13; # $$ is used as a dereference to the original variable
$($refnum) = 11; # This is an alternative syntax using brackets
print $number; # the original variable has changed
```

2.12.2 The arrow operator

If the left hand operand of the arrow operator is an array or hash reference, or a subroutine that produces one, the arrow operator produces a look up of the element or hash:

```
$result = $hashreference -> {$key}; # look up a hash key from a
 reference variable
@arrayslice = $arrayreference -> [3 .. 5]; # obtain a slice from an
 array reference
```

[4] `http://perldoc.perl.org/perlsyn.html#Smart-matching-in-detail`

Perl has four fundamental data type[5]s: scalars, lists, hashes, and typeglobs.

scalar

is a funny way of saying a single value; it may be a number, a string, or a reference[6].

list

is an ordered collection of scalars. A variable that holds a list is called an *array*. Items in a list or array can be accessed by their position in the list; programs can retrieve the first, second, third, etc. item in a list.

hash

is like an array, in that a hash holds many values, but the values are identified by a unique "key", rather than an ordinal index position.

typeglob

is a variable representing an entry within the internal symbol table. It is used to manipulate file handles, and to create references or aliases.

All variables are marked by a leading sigil[7], which identifies the data type. The same name may be used for variables of different types, without conflict.

```
$foo   # a scalar
@foo   # a list
%foo   # a hash
*foo   # a typeglob
```

2.13 Scalar Variables

2.13.1 Introduction to Scalar Variables

Now that you understand how to use strings and numbers in Perl, you need to start learning how to use variables. The best way to learn about scalar variables - Perl talk for a single variable, as against a group or list of values - is to look at an example.

```
#!/usr/bin/perl

use warnings;

$my_scalar_variable = "Hello, Sir!\n";
print $my_scalar_variable;
```

Now let's break this program down:

- The first two lines you already know, `#!/usr/bin/perl` and `use warnings;`
- The third line is more interesting, it contains a scalar variable. There are a few important things to point out:

[5] http://en.wikipedia.org/wiki/data%20type

[6] http://en.wikipedia.org/wiki/Reference%20%28computer%20science%29

[7] http://en.wikipedia.org/wiki/Sigil%20%28computer%20programming%29

1. In case you haven't figured this out, the scalar variable in this line is `$my_scalar_variable`
2. Notice the `$` before the name *my_scalar_variable*, in order to define a scalar variable, this sign must appear before the name.

- Now let's look at the last line. This is just the familiar print function being told to print the value of `$my_scalar_variable`.

Try it!
Type in the program mentioned above and run it.

2.13.2 Assigning and Using Scalar Variables

In the course of writing a program, you will most likely use a variable. What is a variable? A variable is something that stores data. A *scalar* variable holds a single value.

Naming Conventions

- All scalar variables names must start with a `$` symbol. You can remember this by thinking `$`calar.
- Variable names can be comprised of alphanumeric characters and underscores.
- Numeric characters are allowed in names of variables, but not as the first character after the `$`.

Using Scalar Variables

Scalar Variables and Strings

You may recall that earlier in the book, I said that whether you use `"` or `'` in strings makes a big difference in the interaction of strings and variables. Well now I am going to explain what I meant.

Now that you know what a variable is, what if you wanted to put a variable in a string? Here's the difference:

- With a double quoted string, this program:

```
#/usr/bin/perl

use warnings;

$variable = 4;
print "I saw $variable lions!";
```

Would return "I saw 4 lions!"

- With a single quoted string, this program:

```
#/usr/bin/perl

use warnings;

$variable = 4;
```

```
print ,I saw $variable lions!,;
```

Would return "I saw $variable lions!"

Try it!
Type in the programs mentioned above and run them.

This effect is because of what I said before, single quoted strings[8] are interpreted literally.

2.13.3 Comparison Operators

Main article: Perl Programming/Operators[9]

There are operators that are used for comparing numbers and strings. This can be very useful when you get to more advanced programming. Both numbers and strings have their own set of operators which test for a condition such as equal or not equal and return either true or false.

Numeric Comparison Operators

Here is the list of numeric comparison operators:

- == - Equal to
- != - Not equal to
- < - Less than
- > - Greater than
- <= - Less than or equal to
- >= - Greater than or equal to
- <=> - Numeric Comparison

String Comparison Operators

Here is the list of string comparison operators:

- **`eq`** - Equal to
- **`ne`** - Not equal to
- **`lt`** - Less than
- **`gt`** - Greater than
- **`le`** - Less than or equal to
- **`ge`** - Greater than or equal to
- **`cmp`** - String Comparison

[8] Chapter 0.5 on page 5
[9] Chapter 0.12 on page 16

Note
The two 'Comparison' operators <=> and **cmp** *are slightly different from the rest. Rather than returning only true or false, these operators return 1 if the left argument is greater than the right argument, 0 if they are equal, and -1 if the right argument is greater than the left argument.*

2.13.4 Exercises

- Try writing a program like the Hello World program except elaborate it by storing `"Hello, world!\n"` in a variable and then printing the variable.
- Play around with all the things we have learned so far. Try to create a program that has an example of everything we have learned so far.

Perl syntax includes both lists and arrays.

2.14 Lists

A *list* in perl is an ordered set of scalar values. It is represented in your code as a comma-separated sequence of values, which may or may not be contained in scalar variables. Lists can be used to make multiple assignments at once, and can be passed as arguments to several built-in and user-defined functions:

```
#!/usr/bin/perl
use strict;
use warnings;

my ($length, $width, $depth) = (10, 20, 15);

print "The values are: ", $length, $width, $depth;
```

Note
Parentheses are not*required in the construction of a list. They are used only for precedence.*

2.14.1 Alternate List Construction

When creating a list of several strings that do not include spaces, Perl provides a shortcut to get around typing multiple quotes and commas. Instead of

```
($name1, $name2, $name3, $name4) = (,Paul,, ,Michael,, ,Jessica,,
 ,Megan,);
```

you can use the `qw//` operator. This operator uses any non-alpha-numeric character as a delimiter (typically the / character), and encloses a space-separated sequence of barewords. A delimeter separates the command with the arguments. The above line is identical to the following:

```
($name1, $name2, $name3, $name4) = qw/Paul Michael Jessica Megan/;
```

and both are equal to this:

```
($name1, $name2, $name3, $name4) = qw(Paul Michael Jessica Megan);
```

The last example uses the open and close parenthesis as a different delimeter. If there is an open and close version of the delimiter you choose, you need to use them both. Otherwise just repeat the same symbol twice. For example, you cannot type `qw<Paul Michael<` you have to type `qw<Paul Michael>`.

You can also abuse the glob syntax, when the strings do not include shell metacharacters:

```
($name1, $name2, $name3, $name4) = <Paul Michael Jessica Megan>;
```

Note
The resulting strings from the `qw//` *operator are* single-quoted, *meaning no interpolation happens in the set. If you need to include a variable in your list, you cannot use this method.*

2.14.2 List Assignments

As shown above, lists can be used to make several assignments at once. If the number of variables on the left is the same as the number of values on the right, all variables are assigned to their corresponding values, as expected.

If there are fewer variables on the left than values on the right, the 'extra' values are simply ignored:

```
#!/usr/bin/perl

($length, $width) = (10, $w, 15);  #$length gets 10, $width gets the
 value of $w.  15 is ignored
```

If there are more variables on the left than values on the right, the 'extra' variables are assigned the default `undef` value:

```
#!/usr/bin/perl

($length, $width, $depth) = (10, $w); #$length gets 10, $width gets
 the value of $w.  $depth is undef
```

The existence of list assignment creates the ability to 'swap' two variables' values without the need of an intermediary temporary variable:

```
#!/usr/bin/perl

$foo = 10;

$bar = 5;

($foo, $bar) = ($bar, $foo);  #$foo now equals 5, while $bar equals
 10;
```

2.15 Arrays

An array in Perl is a variable which contains a list. An array can be modified, have elements added and removed, emptied, or reassigned to an entirely different list. Just as all scalar variables start with the `$` character, all array variables start with the `@` character.

Note
It is a common and frequent mistake in Perl to use the terms 'list' and 'array' interchangeably. They do not have the same meaning.
A decent analogy is that a list (such as `qw/foo bar baz/`) is to an array (such as `@values`) as a string (such as `'Paul'`) is to a scalar variable (such as `$name`).

2.15.1 Array Assignment

Arrays are assigned lists of values. The list of values can be arbitrarily large or small (it can even contain 0 elements).

```
#!/usr/bin/perl

@nums = (1,2,3,4,5);

@more = 6..1000; #using the ,range, operator

@none = ();  # empty array.

@names = qw/Paul Michael Jessica Megan/;

@all = (@nums, @more);  #@all contains all integers from 1 to 1000
```

That last example exemplifes a feature of Perl known as 'array flattening'. When an array is used in a list, it is the array's **elements** that populate the list, not the array itself. As stated above, a list is a set of **scalar** values only. Therefore, the `@all` array contains 1000 elements, not 2.

Note
Although this implies you cannot create an 'array of arrays', or 'two-dimensional arrays', such things do exist in Perl. They are simulated by using references[10].

2.15.2 Arrays in Scalar context

When an array is used in scalar context - either by assigning a scalar variable to the array's value, or using it in an operation or function that expects a scalar - the array returns its size. That is, it returns the number of elements it currently contains

```
#!/usr/bin/perl

@names = (,Paul,,,Michael,,,Jessica,,,Megan,);

$how_many = @names;
```

[10] Chapter 4.21 on page 75

```
print "I have a total of $how_many names\n";
```

Note
A common misconception is that a ***list*** *in scalar context will also return its size. This is untrue. In fact, there is no such thing as a list in scalar context: using the comma operator in a scalar context does not create a list, instead it evaluates each of its arguments, left to right, and returns the last one:*

```
$name = ('Paul','Michael','Jessica','Megan');
 print "The last name in my list is $name\n";
```

2.15.3 Printing an Array

There are two general ways of printing the values of an array. You can either print the list of items in the array directly, or you can interpolate the array in a double-quoted string.

```
@names = qw/Paul Michael Jessica Megan/;
print "My names are: ", @names, ".\n";
print "My names are: @names.\n";
```

In the first example, the `print` function is being given a list of 6 arguments: the string 'My names are: ', each of the four values in `@names`, and the string ".\n". Each argument is printed separated by the value of the `$,` variable (which defaults to the empty string), resulting in the values from the array being 'squished' together:

```
My names are: PaulMichaelJessicaMegan.
```

In the second example, the `print` function is being given exactly one argument: a string that contains an interpolated array. When Perl interpolates an array, the result is a string consisting of all values in the array separated by the value of the `$"` variable (which defaults to a single space):

```
My names are: Paul Michael Jessica Megan.
```

Note
Both the `$,` and `$"` variables can be changed to any string you like. For example, to separate the array's items with a comma and a space instead of just a space:

```
$" = ', ';
 print "My names are: @names.\n";
```

My names are: Paul, Michael, Jessica, Megan.*You generally do not want to do that as this may cause problems in other parts of your program depending on the default values of those variables though! A safer way to print your arrays with custom separator will be explained later.*

2.15.4 Accessing Elements of an Array

The elements of an array are accessed using a numerical reference within square brackets. Because each item within an array is a scalar value, you need to use $ when referencing a value. The first element of an array is number 0 and all the others count up from there.

A negative number will count down from the right side of the array. This means that -1 references the last element of the array and -3 references the third to last element. Let's see some examples:

```
@array = (1, 2, 3, 4, 5);
print $array[0];   # Prints 1
print $array[3];   # Prints 4
print $array[-1];  # Prints 5
```

What if you need to know the last index? `$#array` will return it for you:

```
@array = (1, 2, 3, 4, 5);
print $array[4];          # Prints 5
print $array[-1];         # Same as above
print $array[ $#array ]; # Also prints 5
```

A common mistake is to do this:

```
print @array[0];  # Also prints 1, but for the wrong reasons
```

In fact `@array[0]` is a *slice* (that is, a sub-array of an array) that contains one element, whereas `$array[0]` is a scalar which contains the value 1.

2.15.5 Common Array Functions

2.16 Command line arguments

As you may wonder, Perl scripts support command line arguments. The entire list of parameters is stored in the array @ARGV, with the first entry containing the first command line argument. If no command line parameters were passed, @ARGV is an empty array.

The array functions and operators listed above can easily be used to detect the passed command line arguments and to detect the number of arguments provided.

2.17 Related Articles

- Data Structures/Arrays[11]
- List Functions[12]
- Array Functions[13]

[11] `http://en.wikibooks.org/wiki/Data%20Structures%2FArrays`

[12] Chapter 4.49 on page 104

[13] Chapter 4.48 on page 103

- Perl Arrays[14]

A Perl hash is similar to an ordinary array, but instead of using integer indexes, a hash uses "keys" that can take on any scalar value. These are usually strings or numbers.

Syntax: instead of the `@` operator, associative arrays use the `%` symbol, and rather than square brackets `[]`, as in `$myarray[0]`, hash elements are referenced using curly brackets `{}`, as in `$myhash{"george"}`.

Hashes are one of the most powerful and commonly used features in Perl. A typical use would be to build a hash that contains a "dictionary", with each key being a word in the dictionary, and the corresponding values being the definitions of those words.

A hash containing the sounds various household pets make is below

```
my %petsounds = ("cat" => "meow",
                 "dog" => "woof",
                 "snake" => "hiss");
```

'=>' and ',' are actually interchangeable, so the right side could look exactly like an array. This means that you can assign an array to a hash. In such an assignment, each element with an even index (starting from 0) in the array becomes a key in the hash. The following statements create the same hash as the previous one does

```
my @array = ("cat", "meow", "dog", "woof", "snake", "hiss");
my %petsounds = @array;
```

But the first style is more preferred because it makes the statement more readable.

To access a hash element, use the curly brackets:

```
 print STDOUT "The cat goes " . $petsounds{"cat"} . ".\n";
```

will print the following to STDOUT

```
The cat goes meow.
```

To add a new sound item to a hash

```
$petsounds{"mouse"} = "squeak!";
```

To overwrite an existing element, just reassign it

```
$petsounds{"dog"} = "arf!";  # The dog now goes "arf!"
```

To remove an item from a hash, use `delete`. Setting the value to `undef` does not delete the item; using `exists` on a key that has been set to `undef` will still return true.

```
delete($petsounds{"cat"});  # will remove "cat" from our hash
```

[14] http://www.programmingbulls.com/perl-array

2.18 "Associative Arrays"

Originally, a "hash" was called an "associative array", but this term is a bit outdated (people just got sick and tired of using seven syllables). Although it isn't intuitive for newcomers to programming, "hash" is now the preferred term. The name is derived from the computer science term, hashtable[15].

2.19 Working with hashes

2.19.1 Printing hash contents

If you know PHP, you may have thought by now of some convenient way to print the contents of your array the way `print_r` does...

```
use Data::Dumper;

print Dumper(\%hash);
```

2.19.2 Counting the number of entries in a hash

To get the size of the hash, simply find the size of the result of the `keys` function, by evaluating it in scalar context:

```
my %hash = (
   ,key1, => 1,
   ,key2, => 2
);

print "Hash has " . keys(%hash) . " elements\n";
my $num_elements = scalar(keys(%hash));
```

2.20 Hash of Hashes of Hashes

You can define multidimensional hash array variables. An example may look like this:

```
#!/usr/bin/perl

use Data::Dumper;

my %a=();

$a{1}{"a"}{"A"}="FIRST";
$a{1}{"c"}{"B"}="THIRD";
$a{1}{"b"}{"C"}="SECOND";

foreach my $k1 ( sort keys %a ) {
  foreach my $k2 ( sort keys %{$a{$k1}} ) {
    foreach my $k3 ( sort keys %{$a{$k1}{$k2}} ) {
      print "$k1\t$k2\t$k3\t$a{$k1}{$k2}{$k3}\n";
```

[15] http://en.wikipedia.org/wiki/Hashtable

```
    }
  }
}

print Dumper(\%a);
```

This code will produce:

```
1       a       A       FIRST
1       b       C       SECOND
1       c       B       THIRD
$VAR1 = {
          '1' => {
                   'c' => {
                            'B' => 'THIRD'
                          },
                   'a' => {
                            'A' => 'FIRST'
                          },
                   'b' => {
                            'C' => 'SECOND'
                          }
                 }
        };
```

Input/Output or **IO**, is an all-encompassing term that describes the way your program interacts with the user. IO comes in two forms, or *stream* types: the program's stimuli are collectively referred to as *input,* while the medium that the program uses to communicate back, write logs, play sounds, etc. is known as *output.* Both types of streams can be redirected either at a lower level than Perl, as is the case when done through the operating system by the shell; or, in Perl itself, as is the case when you reopen the file handles associated with the stream.

2.21 Output

You have already learned how to output with the `print` statement. A simple reference is provided:

```
print "Hello World";
```

What this `print` statement is actually doing is printing to *STDOUT*, which stands for *standard output*. Standard output is the default destination for all output. If you wish to print anywhere else you must be explicit. We will revisit this later.

2.22 Input

As you may have imagined, it's very hard to write a good program without any type of input; here is an example program to teach you these concepts:

```
#!/usr/bin/perl
use strict;
use warnings;

print "What is your name?\n";
```

```
## Get the users $name from Standard In
my $name = <STDIN>;

print "Your name is $name\n";
```

Standard input is usually the keyboard though this can be changed at a lower level than your program. For now we will assume it isn't changed. However, this might not be an assumption you wish to make in production code.

3 Unit Exercise

- Write a program which prompts the user for a number and then returns the number multiplied by four (or any other number).

3.1 Advanced Output Overview

In many situations, especially for web programming, you will find that you want to put certain things, such as backslashes or quotes, in your text that aren't allowed in a traditional print statements. A statement such as

```
print "I said "I like mangos and bananas". ";
```

will not work because the interpreter would think that the quotes mark the end of the string. As with all things in Perl, there are many solutions to this problem.

3.2 Use other quotes

The quickest solution to this problem would be to use single quotes to surround the string, allowing the use of double quotes in the middle.

```
# I said "I like mangos and bananas".
print 'I said "I like mangos and bananas".';
```

This is obviously not the best solution, as it is conceivable that you are trying to print a string containing both kinds of quote:

```
# I said "They're the most delicious fruits".
print 'I said "They're the most delicious fruits".';
```

3.3 Escape characters

For situations like the above where only a short amount of text is being quoted, a common solution is to *escape* any quotes in the string. By preceding any quotes with a backslash they are treated as literal characters.

```
print ,I said "They\,re the most delicious fruits".,;
print "I said \"They\,re the most delicious fruits\".";
```

Using single quotes, the characters that require escaping are \'.

Using double quotes, the characters that need escaping are the variable sigils, (i.e. $@%*) in addition to \"

Using \ to escape reserved characters of course implies that you also need to escape any backslashes you want to use in your string. To print the second line literally using perl, you would need to write:

```
print " print \"I said \\\"They\\\,re the most delicious
fruits\\\".\";"
```

Luckily perl provides us with another way of quoting strings that avoids this problem.

3.4 Custom Quotes

Perl provides the operators q and qq that allows you to decide which characters are used to quote strings. Most punctuation characters can be used. Here are a few examples:

```
print qq{ I said "They,re the most delicious fruits!". };
print q! I said "They,re the most delicious fruits\!". !;
```

The only symbols I have found that cannot be used for these quotes are $ ` /

3.5 Block Output

As can be seen, while the custom quotes option works for short strings, it can run into problems if a lot of text containing a lot of punctuation is output. For this situation, a technique called Block quoting can be used.

```
print <<OUTPUT
 I said "They,re the most delicious fruits!".
OUTPUT
```

Any string of characters can be used instead of OUTPUT in the example above. Using this technique anything can be output no matter what characters it contains. The one caveat of this method is that the closing OUTPUT must be the first character on the line, there cannot be any space before it.

```
print <<EverythingBetween
 ...
 ...
EverythingBetween
```

3.6 Variable Output

It is possible to output variables within strings when you use some of these methods:

```
my $one = ,mangoes,;

print "I like $one.";     # I like mangoes.
print ,I like $one.,;     # I like $one.
print qq@ I love $one.@; #  I love mangoes.
print q#I love $one.#;   # I love $one.

print <<OUT
 I love $one
OUT
;                        #  I love mangoes

print <<,OUT,
 I love $one
OUT
;                        #  I love $one
```

Perl will figure out where your variable ends if the character after it is neither a letter, number nor an underscore. If that is not your case, put your variable inside curly braces:

```
my $one = ,lemon,;

print "A $one is too sour; ";        # A lemon is too sour;
print "${one}ade is better.\n";        # lemonade is better.

print <<OUT
 I love ${one}s in $one souffle.
OUT
;                        #  I love lemons in lemon souffle.
```

3.7 Caveats

The single quote `'` `q{` and double quote `"` `qq` `<<A` operators, behave differently. Whereas when using double quotes, you can include variables and escape any characters, when you use single quotes you can only escape single quotes and you cannot include variables.

3.8 Control structures

The basic control structures do not differ greatly from those used in the C[1] programming language or Java[2] programming language:

3.8.1 Loops

```
while ($boolean) {
    # do something
}

until ($boolean) {
```

[1] http://en.wikibooks.org/wiki/Programming%3AC

[2] http://en.wikibooks.org/wiki/Programming%3AJava

```
    # do something
}
```

Note that the statements in a `while` (or `until`) loop are not executed if the Boolean expression evaluates to false (or true, respectively) on the first pass.

```
do {
    # something
} while ($boolean);

do {
    # something
} until ($boolean);
```

The `do {} while` and the `do {} until` loops are technically statement modifiers[3] and not actual control structures. The statements will be executed at least once.

```
for (my $i=0 ; $i<10 ; $i++) { # for (initialization; termination
 condition; incrementing expr) { ... }
    print "$i\n";
}

foreach my $variable (@list) {
    print "$variable\n";
}
```

`$variable` is an alias to each element of the `@list`, starting at the first element on the first pass through the loop. The loop is exited when all the elements in the list have been exhausted. Since `$variable` is an alias, changing the value will change the value of the element in the *list*. This should generally be avoided to enhance maintainability of the code.

If `$variable` is omitted, the default variable `$_` will be used.

Note that `for` and `foreach` are actually synonyms and can be used interchangeably.

3.8.2 If-then statements

```
if ($boolean_expression) {
    # do something
}

unless ($boolean_expression) {
    # do something
}
```

Statements with `else` blocks (these also work with `unless` instead of `if`)

```
if ($boolean) {
    # do something
}
else {
    # do something else
}

if ($boolean) {
    # do something
```

[3] Chapter 4.2 on page 46

```
}
elsif ($boolean) {
    # do something else
}
```

Control statements can also be written with the conditional following the statements (called "postfix"). This syntax functions (nearly) identically to the ones given above.

```
statement if Boolean expression;
statement unless Boolean expression;
statement while Boolean expression;
statement until Boolean expression;
statement foreach list;
```

3.9 See Also

w:Perl control structures[4]

[4] http://en.wikipedia.org/wiki/Perl%20control%20structures

4 Read files

4.1 Procedural Interface

4.1.1 By slurping file

This method will read the whole file into an array. It will split on the special variable `$/`

```
# Create a read-only file handle for foo.txt
open ( my $fh, ,<,, ,foo.txt, );

# Read the lines into the array @lines
my @lines=<$fh>;

# Print out the whole array of lines
print @lines;
```

4.1.2 By line processing

This method will read the file one line at a time. This will keep memory usage down, but the program will have to poll the input stream on each iteration.

```
# Create a read-only file handle for foo.txt
open ( my $fh, ,<,, ,foo.txt, );

# Iterate over each line, saving the line to the scalar variable
 $line
while ( my $line = <$fh> ) {

  # Print out the current line from foo.txt
  print $line;

}
```

4.2 Object Oriented Interface

Using IO::File you can get a more modern object-oriented interface to a perl file handle.

```
# Include IO::File which will give you the interface
use IO::File;

# Create a read-only file handle for foo.txt
my $fh = IO::File->new( ,foo.txt,, ,r, );

# Iterate over each line, saving the line to the scalar variable
 $line
while ( my $line = $fh->getline ) {
  # Print out the current line from foo.txt
  print $line;

}


# Include IO::File which will give you the interface
use IO::File;

# Create a read-only file handle for foo.txt
my $fh = IO::File->new( ,foo.txt,, ,r, );

my @lines = $fh->getlines;

# Print out the current line from foo.txt
print @lines;
```

In addition to the basic control structures, Perl allows the use of statement modifiers. The statement modifier is placed at the end of the statement that it modifies. Note that the do {} until and do {} while loop constructs are actually statement modifiers. The complete list of modifiers is:

- *statement* **if** *expression*
- *statement* **unless** *expression*
- *statement* **while** *expression*
- *statement* **until** *expression*
- *statement* **foreach** *list*

Unlike BASIC-PLUS, statement modifiers in Perl cannot be stacked.

4.3 String functions

4.3.1 chomp

Action

Removes the last characters from a string only if they're recognized as a record separator (e.g. a newline character)

Returns

?

Syntax

chomp($String = $_);

Example

```
chomp; # removes the last character from $_ if it is a record
separator
chomp(); # (same)
chomp($String); # removes the last character from $String if it is a
record separator
```

See Also

- chop[1] - To remove the last character from a string

4.3.2 chop

Action

Removes the last character from a string regardless

Returns

?

Syntax

chop($String = $_);

Example

```
chop; # removes the last character from $_
chop(); # (same)
chop($String); # removes the last character from $String
```

See Also

- chomp[2] - To remove the last character from a string if it is a record seperator

Removes the last character from a string (e.g. removes the newline characters when reading from a file)

4.3.3 chr

```
print chr(65);  # Prints a capital A
```

[1] Chapter 4.46.2 on page 95
[2] Chapter 4.46.1 on page 95

Gets an ASCII character, given it's code

4.3.4 crypt

```
# One-way hash function
my $HashedWord = crypt($Word, $Salt);
```

(See also MD5[3])

The salt string needs only be 2 characters long, and provides a way of randomising the hash, such that the same word can produce several different hashes, if used with different values of $Salt;!

4.3.5 hex

```
print hex(11);  # Prints B
```

Converts a number to hexadecimal

Other way around - converts hex to number: print hex(11); # prints 17

you can use

print sprintf("%X",11); # Prints B

4.3.6 index

Search for one string within another. (see *rindex* to search from end-to-start)

```
$Result = index($Haystack, $Needle);
$Result = index($Haystack, $Needle, $StartPosition);

index("Some text", "bleh"); # Returns -1 (not found)
index("Some text", "Some"); # Returns 0 (first character)
index("Some text", "text"); # Returns 5 (sixth character)
```

The special variable $[always gets added to the return value, but $[is normally 0, and the manual recommends leaving it at 0.

4.3.7 lc

```
$Lowercase = lc($String);
```

Converts a string to lower-case

[3] http://search.cpan.org/~gaas/MD5-2.03/MD5.pm

4.3.8 lcfirst

Converts the first character of a string to lowercase

4.3.9 length

```
print "String is " . length($String) . " characters long\n";
```

Returns the length of a string

4.3.10 oct

```
print oct(8);  # Prints 10
```

Converts a number to octal

4.3.11 ord

Converts a character to its number.

```
print ord("A"); # prints 65
```

4.3.12 pack

Takes a list and converts it into a string using a supplied set of rules.

```
my $String = pack($Template, @ListOfNumbers);
my $String = pack("CCCC",65,66,67,68); # Result: "ABCD"
```

$Template can be made up of:

```
   a    A string with arbitrary binary data, will be null padded.
   A    An ascii string, will be space padded.
   Z    A null terminated (asciz) string, will be null padded.
```

```
   b    A bit string (ascending bit order inside each byte, like
vec()).
   B    A bit string (descending bit order inside each byte).
   h    A hex string (low nybble first).
   H    A hex string (high nybble first).
```

```
   c    A signed char value.
   C    An unsigned char value.  Only does bytes.  See U for
Unicode.
```

```
   s    A signed short value.
   S    An unsigned short value. (Exactly 16 bits unless you use the
! suffix)
```

```
   i    A signed integer value.
   I    An unsigned integer value. (At least 32 bits wide,
machine-dependant)
```

```
   l    A signed long value.
   L    An unsigned long value. (Exactly 32 bits unless you use the
! suffix)
```

```
   n    An unsigned short in "network" (big-endian) order.
   N    An unsigned long in "network" (big-endian) order.
   v    An unsigned short in "VAX" (little-endian) order.
   V    An unsigned long in "VAX" (little-endian) order. (Exactly 16
bits and 32 bits respectively)
```

```
   q    A signed quad (64-bit) value.
   Q    An unsigned quad value. (Only available if your system
supports 64-bit integers and Perl has been compiled to support them)
```

```
   f    A single-precision float in the native format.
   d    A double-precision float in the native format.
```

```
   p    A pointer to a null-terminated string.
   P    A pointer to a structure (fixed-length string).
```

```
   u    A uuencoded string.
   U    A Unicode character number.  Encodes to UTF-8 internally.
```

```
   w    A BER compressed integer.  Its bytes represent an unsigned
integer in base 128, most significant digit first, with as few digits
as possible.  Bit eight (the high bit) is set on each byte except the
last.
```

```
   x    A null byte.
   X    Back up a byte.
   @    Null fill to absolute position.
```

Each letter may optionally be followed by a number giving a repeat count.

The integer types s, S, l, and L may be immediately followed by a ! suffix to signify native shorts or longs

4.3.13 reverse

Reverses a string (in scalar context) or a list (in list context):

```
my @ReversedList = reverse(@List);
```

```
# As commonly seen in Perl programs:
foreach( reverse( sort( @List )))
{
...
}
```

```
my $ReversedString = reverse($String);
```

```
my @List = ("One ", "two ", "three...");
my $ReversedListAsString = reverse(@List); # Prints "...eerht owt
enO"
```

4.3.14 rindex

Search for one string within another, starting at the end of the string.

```
$Result = rindex($Haystack, $Needle);
$Result = rindex($Haystack, $Needle, $StartPosition);

rindex("Some text", "bleh"); # Returns -1 (not found)
rindex("Some text", "Some"); # Returns 0 (first character)
rindex("abbbbb", "b");       # Returns 5 (first "b" found, when
starting at the end)
```

4.3.15 sprintf

Prints a formatted string:

```
my $Text = sprintf("%d / %d is %08.5f", 1, 3, 1/3); # Result: "10 /
3 is 003.33333"
```

```
sprintf("Character: %c", 65);
sprintf("String %s", "Hello");
sprintf("Signed integer: %d", 15);
sprintf("Unsigned integer: %u", 15);
sprintf("Unsigned int (in octal): %o", 15);
sprintf("Unisgned int (in hex): %x", 15);      # Use %X to get
upper-case output
sprintf("Binary number: %b", 15);
sprintf("Scientific notation: %e", 5000);      # Use %E to get
upper-case output
sprintf("Floating point number: %f", 1/3);     # 0.3333333
sprintf("Floating point number: %g", 1/3);     # Decides between
scientific and float.  %G is uppercase
sprintf("Pointer: %p", $Variable);
```

Use %% to get a percent-sign.

Use %n to request the number of characters written so far, and put it into the next variable in the list. You may want to check that user-supplied formatting rules don't contain this code.

```
sprintf("%02d", $Minutes);  # Forces leading zeros to make the
string 2 characters long
sprintf("%1.5f", $Number);  # Limits the number of decimal places
```

4.3.16 substr

Return part of a string (a *substring*)

Format: substr *string start-position length*

start-position is zero-based.

A negative number starts from the end of the string.

```
$FirstLetter   = substr($Text, 0, 1);   # First letter
$First3Letters = substr($Text, 0, 3);   # First three letters
$Last3Letters  = substr($Text, -3);     # Last three letters
```

You can use **substr** on the left side of an assignment statement to change part of a string. This can actually shorten or lengthen the string.

```
 $text = 'cat dog';
 substr ($mystring, 3, 1) = ' and ';  # $text now contains 'cat and
dog'
```

4.3.17 uc

```
$Uppercase = uc($String);
```

Converts a string to upper-case

4.3.18 ucfirst

Converts the first character of a string to uppercase

4.4 Numeric functions

4.4.1 abs

Returns the absolute(positive) value of a number

```
$Number = abs(-100); # Returns 100;
```

4.4.2 atan2

```
# Converts cartesian(x,y) coordinates into an angle
$Number = atan2($Y, $X);
```

4.4.3 cos

```
# Returns the cosine of an angle (radians)
$Number = cos($Angle);  # Cosine = Adjacent/Hypotenuse
```

4.4.4 exp

```
# Raises e to a specified power
$Number = exp(2); # Returns e^2
```

```
e ≈ 2.71828183 more about e^4
```

4.4.5 hex

```
# Interprets a string as hexidecimal, and returns its value
$Number = hex("10"); # Returns 16
$Number = hex("0xFF"); # Returns 255
```

4.4.6 int

Rounds a number towards zero, returning an integer

```
$Number = int(-1.6);  # Returns -1
$Number = int(0.9);   # Returns 0
$Number = int(28.54); # Returns 28
```

4.4.7 log

```
# Returns the natural logarithm of a number
$Number = log(2.71828183);   # Returns 1
$Number = exp(log($X));      # Returns $X
$Number = log($X) / log(10); # Returns log10($X). Alternately, you
can use the log10() function in the POSIX module
$Number = log($X) / log(15); # Returns log to the base 15 of $X
```

4.4.8 oct

```
# Interprets a string as octal, and returns its value
$Number = oct("10"); # Returns 8
$Number = oct("21"); # Returns 17
```

4.4.9 rand

```
# Gets a random number (may automatically call srand() if that's not
been done)
$Number = rand();  # Returns a random number from 0 to 1
$Number = int(rand(800));  # Returns a random integer from 0 to 799
$Number = 1 + int(rand(999));  # Returns a random integer from 1 to
999
```

4.4.10 sin

```
# Returns the sine of an angle (radians)
$Number = sin($Angle);  # Sine = Opposite/Hypotenuse
```

4.4.11 sqrt

```
# Returns the square-root of a number
$Number = sqrt(4);                  # Returns 2
$Number = sqrt($X ** 2 + $Y ** 2);  # Returns the diagonal distance
across a $X x $Y rectangle
```

```
See the Math::Complex module if you need to take roots of negative
numbers;
```

4.4.12 srand

```
# Seeds (sets-up) the random-number generator
srand();
```

Version-dependant, and older versions of Perl are not guaranteed to have a good seed value. See the Math::TrulyRandom module for more possibilities. The current version of Perl uses the urandom device if it's available.

4.5 Array functions

4.5.1 pop

```
$LastElement = pop(@MyArray);
```

Take the last element from an array

4.5.2 push

```
push(@MyArray, "Last element");
push(@MyArray, "several", "more", "elements");
```

Push a list of elements onto the end of an array

4.5.3 shift

```
shift(@MyArray); #Delete the first element
$FirstElement = shift(@MyArray); #Delete the first element, load it
into $FirstElement instead
```

Take the first element out of an array

4.5.4 splice

```
# Removes elements from an array, optionally replacing them with a
new array
splice(@Array); # Removes all elements from array
splice(@Array, 10); # Removes from element 10 to the end of the
array
splice(@Array, -10); # Removes the last 10 elements of the array
splice(@Array, 0, 10); # Removes the first 10 elements of the array
@NewArray = splice(@Array, 0, 10); # Removes the first 10 elements
of the array and returns those 10 items
splice(@Array, 0, 10, @Array2); # Replaces the first 10 elements of
the array with Array2
```

4.5.5 unshift

```
unshift(@MyArray, "New element");
unshift(@MyArray, "several", "more", "elements");
```

Add a list of elements onto the beginning of an array

4.6 List functions

4.6.1 grep

```
# Returns a list of elements for which an expression is true
@TextFiles = grep(/\.txt$/, @AllFiles);
$NumberOfTextFiles = grep(/\.txt$/, @AllFiles);
```

```
# Can use a block of code instead of an expression
@TextFiles = grep({return(substr($_, -3) eq "txt");}, @AllFiles);
```

4.6.2 join

```
# Joins the items of a list into a single string
$OneItemPerLine = join( "\n", @List);
$EverythingBunchedTogether = join( "", @List);
$Filename = join( "/", ($Directory, $Subdirectory, $Filename));
```

4.6.3 map

```
# Evaluates a block of code for each item in a list, and returns
# a list of the results
@UppercaseList = map(uc, @List);
@Numbers = map {"Number $_"} 1..100;
```

4.6.4 reverse

```
# Reverses the order of a list
@ReversedList = reverse(@List);
# In scalar context, concatenates the list and then reverses the
string
$ReversedString = reverse('foo','bar','baz'); # gives 'zabraboof'
```

4.6.5 sort

```
# Sorts the elements in a list
@AsciiSort = sort(@RandomList);
@AsciiSort = sort @RandomList;
foreach $Item (sort @RandomList)
  {...}
```

```
# Can specify a function to decide the sort order
@CaseInsensitiveSort = sort {uc($a) cmp uc($b)} @RandomList;
@NumericSort = sort {$a <=> $b} @RandomList;
@CustomSort = sort custom_function_name @RandomList;
```

4.6.6 unpack

Unpacks a string into a list - see the templates available for the pack() function for details

4.7 Associative array functions

4.7.1 delete

```
#Remove an element from a hash
%h = ('a'=>1, 'cow'=>'moo', 'b'=>2);
delete $h{cow};
# %h now contains ('a'=>1, 'b'=>2)
```

4.7.2 each

```
#Return the 'next' key/value pair (in a random order)
while (($key, $value) = each (%hash)){
   print "$key => $value\n";
}
```

4.7.3 exists

```
 #Tests whether or not a key exists in a hash (even if the value for
that key is undef)
 if (exists $hash{$key}){
    print "\%hash contains a value for key '$key'\n";
 }
```

4.7.4 keys

```
 #Returns a list of all keys from the hash, in same 'random' order
as each
 foreach $key (keys %hash){
    print "$key => $hash{$key}\n";
 }
```

4.7.5 values

```
 #Returns a list of all values from the hash, in same 'random' order
as keys
 foreach $value (values %hash){
    print "\%hash contains a value '$value'\n";
 }
```

4.8 Input and output functions

4.8.1 binmode

4.8.2 close

```
#closes a filehandle when it is no longer needed
close(STDERR); #hide debugging info from the user
```

4.8.3 closedir

```
# Close a directory open by opendir
closedir(DIRHANDLE);
```

4.8.4 dbmclose

4.8.5 dbmopen

4.8.6 die

Exits the program, printing to "STDERR" the first parameter and the current file and line. Used to trap errors.

```
die "Error: $!\n" unless chdir '/';
```

4.8.7 eof

```
eof FILEHANDLE
eof()
eof
```

This function returns true if the next read on FILEHANDLE would return end-of-file, or if FILEHANDLE is not open. FILEHANDLE may be an expression whose value gives the real filehandle, or a reference to a filehandle object of some sort. An eof without an argument returns the end-of-file status for the last file read. An eof() with empty parentheses () tests the ARGV filehandle (most commonly seen as the null filehandle in <>). Therefore, inside a while (<>) loop, an eof() with parentheses will detect the end of only the last of a group of files. Use eof (without the parentheses) to test each file in a while (<>) loop. For example, the following code inserts dashes just before the last line of the last file:

```
while (<>) {
    if (eof()) {
        print "-" x 30, "\n";
    }
    print;
}
```

On the other hand, this script resets line numbering on each input file:

```
# reset line numbering on each input file
while (<>) {
    next if /^\s*#/;        # skip comments
    print "$.\t$_";
} continue {
    close ARGV if eof;      # Not eof()!
}
```

Like "$" in a sed program, eof tends to show up in line number ranges. Here's a script that prints lines from /pattern/ to end of each input file:

```
while (<>) {
    print if /pattern/ .. eof;
}
```

Here, the flip-flop operator (..) evaluates the pattern match for each line. Until the pattern matches, the operator returns false. When it finally matches, the operator starts returning true, causing the lines to be printed. When the eof operator finally returns true (at the end of the file being examined), the flip-flop operator resets, and starts returning false again for the next file in @ARGV

4.8.8 fileno

4.8.9 flock

4.8.10 format

4.8.11 getc

4.8.12 print

Prints the parameters given.

Discussed in the following sections:

Digression on print in *Strings* section[5]

[5] Chapter 0.7.3 on page 10

4.8.13 printf

4.8.14 read

4.8.15 readdir

4.8.16 rewinddir

4.8.17 seek

4.8.18 seekdir

4.8.19 select

4.8.20 syscall

4.8.21 sysread

4.8.22 sysseek

4.8.23 syswrite

4.8.24 tell

4.8.25 telldir

4.8.26 truncate

4.8.27 warn

4.8.28 write

4.9 Functions for working with fixed length records

4.9.1 pack

See the entry for **pack** further up the page

4.9.2 read

```
# Reads data from a file-handle
read(FILEHANDLE, $StoreDataHere, $NumberBytes);
```

```
# Returns the number of bytes read
$NumberBytesRead = read(FILEHANDLE, $StoreDataHere, $NumberBytes);
```

```
# Optional offset is applied when the data is stored (not when reading)
read(FILEHANDLE, $StoreDataHere, $NumberBytes, Offset);
```

4.9.3 syscall

```
# Runs a system command
syscall( $Command, $Argument1, $Argument2, $Argument3);
```

```
# (maximum 14 arguments)
$ReturnValue = syscall($Command);
```

4.9.4 sysread

4.9.5 syswrite

4.9.6 unpack

```
# See the pack function for details (unpack does the opposite!)
unpack($Template, $BinaryData);
```

4.9.7 vec

4.10 Filesystem functions

4.10.1 -X

```
if( -r $FullFilename) // File is readable by effective uid/gid.
if( -w    $FullFilename) // File is writable by effective uid/gid.
if( -x    $FullFilename) // File is executable by effective uid/gid.
if( -o    $FullFilename) // File is owned by effective uid.
```

```
if( -R    $FullFilename) // File is readable by real uid/gid.
if( -W    $FullFilename) // File is writable by real uid/gid.
if( -X    $FullFilename) // File is executable by real uid/gid.
if( -O    $FullFilename) // File is owned by real uid.
```

```
if( -e    $FullFilename) // File exists.
if( -z    $FullFilename) // File has zero size.
if( -s    $FullFilename) // File has nonzero size (returns size).
```

```
if( -f    $FullFilename) // File is a plain file.
if( -d    $FullFilename) // File is a directory.
if( -l    $FullFilename) // File is a symbolic link.
if( -p    $FullFilename) // File is a named pipe (FIFO), or
Filehandle is a pipe.
if( -S    $FullFilename) // File is a socket.
if( -b    $FullFilename) // File is a block special file.
if( -c    $FullFilename) // File is a character special file.
if( -t    $FullFilename) // Filehandle is opened to a tty.
if( -u    $FullFilename) // File has setuid bit set.
if( -g    $FullFilename) // File has setgid bit set.
if( -k    $FullFilename) // File has sticky bit set.
```

```
if( -T    $FullFilename) // File is an ASCII text file.
if( -B    $FullFilename) // File is a "binary" file (opposite of
-T).
```

```
$Age = -M $FullFilename; // Age of file in days when script started.
$Age = -A $FullFilename; // Same for access time.
$Age = -C $FullFilename; // Same for inode change time.
```

4.10.2 chdir

```
chdir $Directory;
chdir $Directory || die("Couldn't change directory");
```

4.10.3 chmod

```
chmod 0744 $File1;
chmod 0666 $File1, $File2, $File3;
# 0 for octal, at the beginning of a number
```

```
        | Owner | Group | Others |
Execute |   4   |   4   |   4    |
Write   |   2   |   2   |   2    |
Read    |   1   |   1   |   1    |
======--+======-+======-+======--+
Total   |       |       |        |
```

4.10.4 chown

```
# Change the owner of a file
chown($NewUserID, $NewGroupID, $Filename);
chown($NewUserID, $NewGroupID, $File1, $File2, $File3);
```

```
chown($NewUserID, -1, $Filename); # Leave group unchanged
chown(-1, $NewGroupID, $Filename); # Leave user unchanged
```

4.10.5 chroot

```
chroot $NewRootDirectory;
```

Sets the root directory for the program, such that the "/" location refers to the specified directory.

Program must be running as root for this to succeed.

4.10.6 fcntl

4.10.7 glob

```
#Expands filenames, in a shell-like way
my @TextFiles = glob("*.txt");
```

See also File::Glob

4.10.8 ioctl

4.10.9 link

```
# Creates a link to a file
link($ExistingFile, $LinkLocation);
link($ExistingFile, $LinkLocation) || die("Couldn't create link");
```

4.10.10 lstat

Identical to stat(), except that if given file is symbolic link, stat link not the target.

4.10.11 mkdir

```
mkdir $Filename || die("Couldn't create directory");
mkdir $Filename, 0777; # Make directory with particular
file-permissions
```

4.10.12 open

```
open(my $FileHandle, $Filename) || die("Couldn't open file");
open(my $fp, "<", $Filename);   # Read from file
open(my $fp, ">", $Filename);   # Write to file
open(my $fp, ">>", $Filename);  # Append to file
```

```
open(my $fp, "<$Filename");     # Read from file
open(my $fp, ">$Filename");     # Write to file
open(my $fp, ">>$Filename");    # Append to file
```

```
open(my $fp, "<", "./    filename with whitespace   \0");
open(my $fp, "<", "./->filename with reserved characters\0");
```

```
open(my $fp, "$Program |");     # Read from the output of another
program
open(my $fp, "| $Program");     # Write to the input of another
program
```

```
open(my $fp, "<", "-");         # Read from standard input
open(my $fp, ">", "-");         # Write to standard output
```

4.10.13 opendir

```
opendir(my $DirHandle, $Directory) || die("Couldn't open
directory");
while (my $Filename = readdir $DirHandle){
  # Do something with $Filename in $Directory
}
closedir($DirHandle);
```

```
opendir(DIR, $Directory) || die("Couldn't open directory");
foreach(readdir(DIR)){
  # Do something with $_ in $Directory
}
closedir(DIR);
```

4.10.14 readlink

```
# Finds the value of a symbolic link
$LinkTarget = readlink($LinkPosition);
```

4.10.15 rename

```
rename $OldFile, $NewFile or die("Couldn't move file");
```

May work differently on non-*nix operating systems, and possibly not at all when moving between different filesystems. See

Figure 1

for more complicated file operations.

4.10.16 rmdir

```
rmdir $Filename || die("Couldn't remove directory");
```

4.10.17 stat

```
@FileStatistics = stat($Filename);
```

```
$DeviceNum    = $FileStatistics[0]; # device number of filesystem
$Inode        = $FileStatistics[1]; # inode number
$FileMode     = $FileStatistics[2]; # (type and permissions)
$NumHardLinks = $FileStatistics[3]; # number of (hard) links to the
file
$UserID       = $FileStatistics[4]; # numeric user ID
$GroupID      = $FileStatistics[5]; # numeric group ID
$DeviceIdent  = $FileStatistics[6]; # Device identifier (special
files only)
$SizeBytes    = $FileStatistics[7];
$AccessTime   = $FileStatistics[8]; # seconds since the epoch
$ModifyTime   = $FileStatistics[9];
```

```
$ChangeTime   = $FileStatistics[10];
$BlockSize    = $FileStatistics[11];
$NumBlocks    = $FileStatistics[12];
```

4.10.18 symlink

```
# Creates a new filename symbolically linked to the old filename
symlink($OldFilename, $NewFilename);
symlink($OldFilename, $NewFilename) || die("Couldn't create
symlink");
eval(symlink($OldFilename, $NewFilename));
```

4.10.19 umask

```
# Sets or returns the umask for the process.
my $UMask = umask();
umask(0000); # This process can create any type of files
umask(0001); # This process can't create world-readable files
umask(0444); # This process can't create executable files
```

4.10.20 unlink

```
# Deletes a file
unlink $Filename;
unlink $Filename || die("Couldn't delete file");
unlink $File1, $File2, $File3;
(unlink($File1, $File2, $File3) == 3) || die("Couldn't delete
files");
```

4.10.21 utime

```
# Updates the modification times of a list of files
my $AccessTime = time();
my $ModificationTime = time();
```

```
utime($AccessTime, $ModificationTime, $Filename);
my $NumFilesChanged = utime($AccessTime, $ModificationTime, $File1,
$File2, $File3);
```

4.11 Program functions

4.11.1 caller

Returns information about the current function call stack. In scalar context, returns only the name of the package from where the current subroutine was called. In list context, returns the package, filename, and line number. In list context with a numeric argument passed, returns several pieces of information (see below). The argument represents how many levels in the call stack to go back.

```
#!/usr/bin/perl
```

```
foo();
sub foo {
   $package = caller; #returns 'main'
   ($package, $filename, $line) = caller; #returns 'main', the file
name, and 3
   # Line below returns all 10 pieces of info. (Descriptions
self-explanatory from variable names)
   ($package, $filename, $line, $subroutine, $hasargs, $wantarray,
$evaltext, $is_require, $hints, $bitmask) =
      caller(0);
 }
```

4.11.2 import

There is no actual 'import' function. Rather, it is a convention when writing a module to create a subroutine named 'import' which populates the current namespace with that module's needed variables and/or methods.

The standard 'Exporter' module provides an import method if your class has it as a base class.

4.11.3 package

Declares all lines that follow (until EOF or the next package statement) to belong to the given package's namespace.

```
#!/usr/bin/perl

$x = 5;  #sets $main::x

package Foo;
$x = 5;  #sets $Foo::x
sub bar { #defines &Foo::bar
   print "hello world";
}

package Temp;
$x = 5; #sets $Temp::x
```

4.11.4 require

includes the specified module's code into the current program. The module can be specified either with an absolute or relative path, or with a bareword. If a bareword is given, a '.pm' extention is added, and :: is replaced with the current operating system's path seperator:

```
require Foo::Bar;
#identical to:
require 'Foo/Bar.pm';
```

4.11.5 use

Requires and imports the given module or pragma, at compile time. The line

```
use Foo qw/bar baz/;
```

is identical to:

```
BEGIN {
   require Foo;
   import Foo qw/bar baz/;
}
```

4.12 Misc functions

4.12.1 defined

```
#returns true if argument is not undef
$x = 0;
print "X defined\n" if defined $x; #prints
print "Y defined\n" if defined $y; #does not print
```

4.12.2 dump

4.12.3 eval

```
eval('$a=30;$b=40;');
print $a,$b;
```

4.12.4 formline

4.12.5 local

```
#assigns temporary value to global variable for duration of lexical
scope
$x = 5;
print "x = $x\n"; # 5
{
  local $x = 10;
  print "x = $x\n"; # 10
}
print "x = $x\n"; # 5
```

4.12.6 my

```
#creates new lexical (ie, not global) variable
$x = 5;  #refers to $main::x
{
  my $x = 10;
  print "x = $x\n"; # the lexical - 10
  print "main's x = $main::x\n" # the global - 5
}
print "x = $x\n";  #the global, because no lexical in scope - 5
```

4.12.7 reset

```
#resets hash's internal pointer, to affect lists returned by each
while ($k, $v = each %h){
  print "$k = $v\n";
  last if ($i++ == 2);
}
#if another each done here, $k,$v will pick up where they left off.
reset %h
#now each will restart from the beginning.
```

4.12.8 scalar

```
#forces scalar context on an array
@sizes = (scalar @foo, scalar @bar);
#creates a list of the sizes of @foo and @bar, rather than the
elements in @foo and @bar
```

4.12.9 undef

```
#undefines an existing variable
$x = 5;
undef $x;
print "x = $x\n" if defined $x; #does not print
```

4.12.10 wantarray

```
#returns 'true', 'false', or undef if function that called it was
called in list, scalar, or void context, respectively.
sub fctn {
   my @vals = (5..10);
   if (wantarray) {
      return @vals;
   } elsif (defined wantarray) {
      return $vals[0];
   } else {
      warn "Warning!  fctn() called in void context!\n";
   }
}
```

4.13 Processes

4.13.1 alarm

4.13.2 exec

4.13.3 fork

```
#clones the current process, returning 0 if clone, and the process
id of the clone if the parent
my $pid = fork();
if ($pid == 0) {
   print "I am a copy of the original\n";
} elsif ($pid == -1)  {
   print "I can't create a clone for some reason!\n";
} else {
   print "I am the original, my clone has a process id of $pid\n";
}
```

4.13.4 getpgrp

4.13.5 getppid

4.13.6 getpriority

4.13.7 kill

4.13.8 pipe

4.13.9 qx/STRING/

4.13.10 setpgrp

4.13.11 setpriority

4.13.12 sleep

4.13.13 system

4.13.14 times

4.13.15 wait

4.13.16 waitpid

4.14 Modules

4.14.1 do

4.14.2 import

4.14.3 no

4.14.4 package

4.14.5 require

4.14.6 use

4.15 Classes and objects

See also Perl Objects[6]

[6] Chapter 4.25.3 on page 82

4.15.1 bless

4.15.2 dbmclose

4.15.3 dbmopen

4.15.4 package

4.15.5 ref

4.15.6 tie

4.15.7 tied

4.15.8 untie

4.15.9 use

4.16 Sockets

4.16.1 accept

4.16.2 bind

4.16.3 connect

4.16.4 getpeername

4.16.5 getsockname

4.16.6 getsockopt

4.16.7 listen

4.16.8 recv

4.16.9 send

4.16.10 setsockopt

4.16.11 shutdown

4.16.12 socket

4.16.13 socketpair

4.17 Login information

4.17.1 endgrent

4.17.2 endhostent

4.17.3 endnetent

```
@TimeParts = gmtime();
@TimeParts = gmtime($Time);
```

```
$Seconds    = $TimeParts[0]; # 0-59
$Minutes    = $TimeParts[1]; # 0-59
$Hours      = $TimeParts[2]; # 0-23
$DayOfMonth = $TimeParts[3]; # 1-31
$Month      = $TimeParts[4]; # 0-11
$Year       = $TimeParts[5]; # Years since 1900
$DayOfWeek  = $TimeParts[6]; # 0:Sun 1:Mon 2:Tue 3:Wed 4:Thu 5:Fri
6:Sat
$DayOfYear  = $TimeParts[7]; # 1-366
```

4.19.2 localtime

Converts a timestamp to local time

```
@TimeParts = localtime();
@TimeParts = localtime($Time);
```

```
$Seconds    = $TimeParts[0]; # 0-59
$Minutes    = $TimeParts[1]; # 0-59
$Hours      = $TimeParts[2]; # 0-23
$DayOfMonth = $TimeParts[3]; # 1-31
$Month      = $TimeParts[4]; # 0-11
$Year       = $TimeParts[5]; # Years since 1900
$DayOfWeek  = $TimeParts[6]; # 0:Sun 1:Mon 2:Tue 3:Wed 4:Thu 5:Fri
6:Sat
$DayOfYear  = $TimeParts[7]; # 1-366
```

4.19.3 time

```
$Time = time();
```

Returns number of seconds since an epoch (which is system-dependant, but may be Jan 1 1970)

See also ../Time::Hires/[7]

4.19.4 times

```
@CPUTimes = times();
$UserTimeForProcess    = $CPUTimes[0];
$SystemTimeForProcess  = $CPUTimes[1];
$UserTimeForChildren   = $CPUTimes[2];
$SystemTimeForChildren = $CPUTimes[3];
```

[7] http://en.wikibooks.org/wiki/..%2FTime%3A%3AHires%2F

4.20 Functions that reverse each other

Some functions in perl reverse or otherwise cancel the effect of each other, so running a string through both of them will produce the same output as the input, for example

```
print ord(chr(1));
```

will echo `1` to standard output,

`ord()`[8] will convert a character to its number in the character set, while `chr()`[9] will convert a number to its corresponding character, therefore

in the same way that $\sqrt{x^2} = x$ and $\sqrt{x}^2 = x$ in Mathematics[10] (assuming x is non-negative), `ord(chr(1)) = 1` and `chr(ord(1)) = 1` in Perl.

List of functions that reverse each other:

- `lc()`[11] and `uc()`[12]
- `lcfirst()`[13] and `ucfirst()`[14]
- `ord()`[15] and `chr()`[16]
- `join()`[17] and `split()`[18]
- `push()`[19] and `pop()`[20]
- `unshift()`[21] and `shift()`[22]

These are a set of eight exercises that can be used to test your ability to write Perl programs. In some cases, these exercises might include material not covered from the textbook; in those cases, you may have to consult your platform documentation to identify a necessary function or otherwise implement one yourself.

- ../Exercise 1/[23]
- ../Exercise 2/[24]
- ../Exercise 3/[25]
- ../Exercise 4/[26]

[8] Chapter 4.46.11 on page 97
[9] Chapter 4.46.3 on page 96
[10] http://en.wikipedia.org/wiki/Mathematics
[11] Chapter 4.46.7 on page 97
[12] Chapter 4.46.17 on page 101
[13] Chapter 4.46.8 on page 97
[14] Chapter 4.46.18 on page 101
[15] Chapter 4.46.11 on page 97
[16] Chapter 4.46.3 on page 96
[17] Chapter 4.49.2 on page 104
[18] Chapter 4.51.28 on page 108
[19] Chapter 4.48.2 on page 103
[20] Chapter 4.48.1 on page 103
[21] Chapter 4.48.5 on page 104
[22] Chapter 4.48.3 on page 103
[23] http://en.wikibooks.org/wiki/..%2FExercise%201%2F
[24] http://en.wikibooks.org/wiki/..%2FExercise%202%2F
[25] http://en.wikibooks.org/wiki/..%2FExercise%203%2F
[26] http://en.wikibooks.org/wiki/..%2FExercise%204%2F

- ../Exercise 5/[27]
- ../Exercise 6/[28]
- ../Exercise 7/[29]
- ../Exercise 8/[30]

4.21 Section 2: In-depth Perl ideas

- http://perldoc.perl.org/perlstyle.html

4.22 Introduction

So you've been plodding along with your perl scripts, fiddling with arrays and hashes and suddenly you realize that you would like to pass a function to another function depending on the data you encounter, or perhaps you would like to get back a hash when you look up an array index. References are the thing for you, allowing you to build and pass around ever more complex data structures.

4.23 Referencing and Dereferencing Syntax

```
my $nightmare = "clowns";
my $ref = \$nightmare;
print "I laugh in the face of " . ${$ref} . "\n";
```

Output should be *I laugh in the face of clowns*.

The curly brackets are optional, but generally recommended.

4.24 External links

- http://perldoc.perl.org/perlref.html
- http://perldoc.perl.org/perlreftut.html

Regular expressions are tools for complex searching of text, considered one of the most powerful aspects of the Perl language. A regular expression can be as simple as just the text you want to find, or it can include wildcards, logic, and even sub-programs.

To use regular expressions in perl, use the =˜ operator to bind a variable containing your text to the regular expression:

[27] http://en.wikibooks.org/wiki/..%2FExercise%205%2F
[28] http://en.wikibooks.org/wiki/..%2FExercise%206%2F
[29] http://en.wikibooks.org/wiki/..%2FExercise%207%2F
[30] http://en.wikibooks.org/wiki/..%2FExercise%208%2F

```
$Haystack =˜ /needle/;
```

This returns 1 if "needle" is contained within $HayStack, or 0 otherwise.

```
$Haystack =˜ /needle/i;    # The i means "case-insensitive"
```

```
$Haystack =˜ /(needle|pin)/;  # Either/Or statements
```

```
$Haystack =˜ /needle \d/;     # "needle 0" to "needle 9"
```

Regular expression can also be used to modify strings. You can search and replace complex patterns by using the regex format s///

```
 $msg = "perl is ok";
 $msg =˜ s/ok/awesome/;           # search for the word "ok" and
replace it with "awesome"
 ($msg is now "perl is awesome")
```

4.24.1 Match a string

```
# Shorthand form uses // to quote the regular expression
$Text =˜ /search words/;
```

```
# The m function allows you to use your choice of quote marks
$Text =˜ m|search words|;
$Text =˜ m{search words};
$Text =˜ m<search words>;
$Text =˜ m#search words#;
```

4.24.2 Split a string into parts

```
# The split function allows you to split a string wherever a regular
expression is matched
@ArrayOfParts = split( /,/, $Text);     # Splits wherever a comma is
found
@ArrayOfParts = split( /\s+/, $Text);   # Splits where whitespace is
found
@ArrayOfParts = split( /,\s*/, $Text);  # Comma followed by optional
whitespace
@ArrayOfParts = split( /\n/, $Text);    # Newline marks where to
split
```

4.24.3 Search and replace a string

```
# The s function allows you to search and replace within a string.
s(ubstitute)
$Text =˜ s/search for/replace with/;
$Text =˜ s|search for|replace with|;
$Text =˜ s{search for}{replace with};
```

```
# Putting a g (global) at the end, means it replaces all occurances
and not just the first
$Text =˜ s/search for/replace with/g;
```

```
# As with everything, putting an i (insensitive) at the end ignores
the differences between
# uppercase and lowercase.
Use Locale;
$Text =˜ s/search for/replace with/i;
```

4.24.4 Extracting values from a string

```
# This function sets the variables $1, $2, $3 ...
#   to the information that it has extracted from a string.

$Text =˜ m/before(.*)after/;
# So, if $Text was "beforeHelloafter", $1 is now "Hello"

$Text =˜ m/bef(.*)bet(.*)aft/;
# This time, if $Text was "befOnebetTwoaft", $1 is now "One" and $2
is "Two"
```

```
# It can also be used to extract certain kind of information.
$Text =˜ m|([ˆ=]*)=(\d*)|;

#If $Text was "id=889", $1 now equals "id" and $2 equals 889.
```

4.24.5 Regular Expressions with Perl Examples

Metacharacter	Description	Example Note that all the if statements return a TRUE value

.	Matches an arbitrary character, but not a newline.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/...../) {`
		`print "$string1 has length >= 5\n";`
		`}`

( )	Groups a series of pattern elements to a single element. When you match a pattern within parentheses, you can use any of $1, $2, ... $9 later to refer to the previously matched pattern.	**Program:**
		`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/(H..).(o..)/) {`
		`print "We matched '$1' and '$2'\n";`
		`}`
		Output:
		`We matched 'Hel' and 'o W';`

+	Matches the preceding pattern element one or more times.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/l+/) {`
		`print "There are one or more consecutive l's in $str`
		`}`

Metacharacter	**Description**	**Example** **Note that all the if statements return a TRUE value**

?	Matches zero or one times.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/H.?e/) {`
		`print "There is an 'H' and a 'e' seperated by ";`
		`print "0-1 characters (Ex: He Hoe)\n";`
		`}`

?	Matches the *, +, or {M,N}'d regexp that comes before as few times as possible.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/(l+?o)/) {`
		`print "The non-greedy match with one or more 'l'`
		`print "followed by an 'o' is 'lo', not 'llo'.\n";`
		`}`

*	Matches zero or more times.	`$string1 = "Hello World\n";`
		`if ($string =˜ m/el*o/) {`
		`print "There is a 'e' followed by zero to many";`
		`print "'l' followed by 'o' (eo, elo, ello, elllo)\n`
		`}`

{M,N}	Denotes the minimum M and the maximum N match count.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/l{1,2}/) {`
		`print "There exists a substring with at least 1";`
		`print "and at most 2 l's in $string1\n";`
		`}`

[...]	Denotes a set of possible matches.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/[aeiou]+/) {`
		`print "$string1 contains a one or more";`
		`print "vowels\n";`
		`}`

[ˆ...]	Matches any character not in the square brackets.	`$string = "Sky.";`
		`if (String =˜ /[ˆaeiou]/) {`
		`print "$string doesn't contain any vowels";`
		`}`

|	Matches one of the left or right operand.	`$string1 = "Hello World\n";`
		`if ($string1 =˜ m/(Hello|Hi)/) {`
		`print "Hello or Hi is ";`
		`print "contained in $string1";`
		`}`

Metacharacter	**Description**	**Example** **Note that all the if statements return a TRUE value**

`\b`	Matches a word boundary.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/ello?\b/) {`
		`print "There is a word that ends with";`
		`print " 'ello'\n";`
		`} else {`
		`print "There are no words that end with";`
		`print "'ello'\n";`
		`}`

`\w`	Matches alphanumeric, including "_".	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\w/) {`
		`print "There is at least one alpha-";`
		`print "numeric char in $string1 (A-Z, a-z, 0-9, _-`
		`)\n";`
		`}`

`\W`	Matches a non-alphanumeric character.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\W/) {`
		`print "The space between Hello and ";`
		`print "World is not alphanumeric\n";`
		`}`

`\s`	Matches a whitespace character (space, tab, newline, formfeed)	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\s.*\s/) {`
		`print "There are TWO whitespace ";`
		`print "characters seperated by other characters in $`
		`}`

`\S`	Matches anything BUT a whitespace.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\S.*\S/) {`
		`print "There are TWO non-whitespace ";`
		`print "characters seperated by other characters in $`
		`}`

`\d`	Matches a digit, same as [0-9].	`$string1 = "99 bottles of beer on the wall.";`
		`if ($string1 =~ m/(\d+)/) {`
		`print "$1 is the first number in '$string1'\n";`
		`}`
		`'''Output:'''`
		`99 is the first number in '99 bottles of beer on the`

`\D`	Matches a non-digit.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/\D/) {`
		`print "There is at least one character in $string1";`
		`print "that is not a digit.\n";`
		`}`

Metacharacter	**Description**	**Example** **Note that all the if statements return a TRUE value**

^	Matches the beginning of a line or string.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/^He/) {`
		`print "$string1 starts with the characters 'He'\n";`
		`}`

$	Matches the end of a line or string.	`$string1 = "Hello World\n";`
		`if ($string1 =~ m/rld$/) {`
		`print "$string1 is a line or string";`
		`print "that ends with 'rld'\n";`
		`}`

4.25 Overview

Perl modules (Files that end with the pm extension) are files of perl code that can be reused from program to program. There is an online repository of perl modules called CPAN (Comprehensive Perl Archive Network) at `http://cpan.org.` Many of these modules come standard with Perl, but others must be installed as needed.

There are thousands of perl modules that do everything from creating a temporary file to calling Amazon web services. These modules can make it easy to quickly write your application if you know how to find, install, and use the appropriate Perl modules. If you are thinking of writing your own Perl module, the best thing to do is to first search at `http://Search.cpan.org` to make sure you are not about to reinvent the wheel.

There are two major styles of Perl modules:

1. Object-Oriented
2. Functional

Some perl modules use both approaches.

To use an object-oriented Perl module you would do something like this:

```
use Foo;
my $foo = Foo->new();
print $foo->bar;  #call Foo's bar method and print the output.
```

A functional perl module might get used like this:

```
use Foo qw/bar/; # Import the name of the subroutine you want to use.
print bar();
```

4.25.1 How to install a Perl module

Find the perl module you want at `http://cpan.org,` and download the gzipped file. Untar and unzip the file:

```
tar -zxvf MyModule.tgz
```

Then cd into the directory, and follow the instructions in the README or INSTALL file.

You can also use a command-line program called cpan, if you have it installed:

```
sudo cpan -imt Module::I::Want
```

4.25.2 To Write your own Perl Module

Perl modules differ from perl scripts in two key and simple ways. Instead of starting the module with "#!/path/to/perl", you start the file with the following:

```
package My::Module::Name;
```

You need to end the module with a true value, so the common practice is to do this at the end of the file:

```
1;
```

The following is a valid perl module:

```
package My::Module::Name;

1;
```

Example

We create a new file called ExampleModule.pm, and in it have the following code:

```
package ExampleModule;
use strict;
use base "Exporter";
our @EXPORT = qw/hello_world/;

sub hello_world
{
   print "hello, world!\n";
}

1;
```

We can test to see if the syntax is valid by running:

```
perl -c ExampleModule.pm
```

It will print out "ExampleModule.pm syntax OK" if all is well. Otherwise, you can debug using the messages that are printed out.

Now we can use it in a script to see if it works:

```
#!/usr/bin/perl

use ExampleModule;

hello_world();

exit;
```

Voilá! You have made a perl module.

4.25.3 Create a CPAN-style Perl module

CPAN-style modules have test suites and a way to build the module into the perl library.

Download and install: Module::Starter from CPAN. Once this is installed, there will be a program called module-starter in your path. To create a new module, do the following from the command line:

```
module-starter --module=My::Module::Name, My::Other::Module::Name,
 --author="My Name" --email="myemail@gmail.com"
```

It will then create a set of directories for you, including some shell module files with starter POD documentation. The perl modules will be inside the lib directory inside the directory that is created. These are the files to edit. You can put your tests for the modules into the "t" directory. To install and build the module, you do the following:

```
>perl Makefile.PL
>make
>make test
>sudo make install
```

When Perl was initially developed there was no support at all for Object Orientated (OO) programming. Since Perl 5 OO has been added using the concept of Perl packages (namespaces), an operator called bless, some magic variables (@ISA, AUTOLOAD, UNIVERSAL), the -> and some strong conventions for supporting inheritance and encapsulation.

An object is created using the `package` keyword. All subroutines declared in that package become object or class methods.

A class instance is created by calling a constructor method which must be provided by the class, by convention this method is called new()

Let's see this constructor.

```
package Object;

sub new {
  return bless {}, shift;
}
```

```
sub setA {
  my $self = shift;
  my $a = shift;
  $self->{a}=$a;
}

sub getA {
  my $self = shift;
  return $self->{a};
}
```

Client code can use this class something like this.

```
my $o = Object->new;
$o->setA(10);
print $o->getA;
```

This code prints 10.

Let's look at the `new` contructor in a little more detail:

The first thing is that when a subroutine is called using the -> notation a new argument is pre-pended to the argument list. It is a string with either the name of the Package or a reference to the object (Object->new() or $o->setA. Until that makes sense you will find OO in Perl very confusing.

To use private variables in objects and have variables names check, you can use a little different approach to create objects.

```
package my_class;
use strict;
use warnings;
{
  # All code is enclosed in block context

  my %bar;  # All vars are declared as hashes
  sub new {
    my $class = shift;
    my $this = \do{ my $scalar }; # object is a reference to scalar
(inside out object)
    bless $this, $class;
    return $this;
  }

  sub set_bar {
    my $this = shift;
    $bar{$this} = shift;
  }

  sub get_bar {
    my $this = shift;
    return $bar{$this};
  }
}
```

Now you have good encapsulation - you cannot access object variables directly via $o->{bar} but only using set/get methods. It's also impossible to make mistakes in object variable names, because they are not a hash-keys but normal perl variables, needed to be declared.

We use them the same way like hash-blessed objects:

```
my $o = my_class->new();
$o->set_bar(10);
print $o->get_bar();
```

prints 10

4.26 Section 3: Interfacing Perl

There are several GUI widget sets available as additions to perl, though the most common is probably Perl/Tk.

- Perl Tk[31] (sometimes pTk or ptk) is a collection of modules and code which attempts to wed the simple Tk widget set to perl 5.
- Tcl::Tk[32] same as perlTk, but uses existing Tcl/Tk via Tcl, so allowing Tcl widgets
- Tkx[33] a different, lightweight, access to Tk via Tcl.
- Gtk[34] uses Gtk+, the Gimp Toolkit.
- Gtk2[35] uses Gtk+ 2.x.
- Gtk3[36] uses Gtk+ 3.x.
- Qt[37] uses the Qt toolkit.
- Wx[38] uses the platform independent wxWidgets toolkit.
- Prima[39] uses its own toolkit.

4.27 Links

The comp.lang.perl.tk FAQ[40]

A huge collection of freely usable perl modules, ranging from advanced mathematics to database connectivity, networking and more, can be downloaded from a network of sites called CPAN. Most or all of the software on CPAN is also available under either the Artistic License, the GPL, or both. CPAN.pm is also the name of the perl module that downloads and installs other perl modules from one of the CPAN mirror sites; such installations can be done with interactive prompts, or can be fully automated.

Look for modules on CPAN[41]

[31] http://search.cpan.org/dist/Tk/

[32] http://search.cpan.org/dist/Tcl-Tk/

[33] http://search.cpan.org/dist/Tkx/

[34] http://search.cpan.org/dist/Gtk/

[35] http://search.cpan.org/dist/Gtk2/

[36] http://search.cpan.org/dist/Gtk3/

[37] http://search.cpan.org/dist/qt/

[38] http://search.cpan.org/dist/Wx/

[39] http://search.cpan.org/~karasik/Prima/Prima.pm

[40] http://w4.lns.cornell.edu/~pvhp/ptk/ptkFAQ.html

[41] http://search.cpan.org/

4.28 Installing modules

4.28.1 With ActivePerl (Windows systems)

From a command-line, type the command

```
ppm
```

This will give you a "Perl Package Manager" prompt, which allows you to download and install modules from the internet. For example, to install the Time::HiRes module, type:

```
search time::hires
```

That will give you a list of modules which match your search query. Once you know the module is available and what its exact name is, you can install the module with:

```
install Time::HiRes
```

4.28.2 With Perl

If you're using a normal version of Perl, the way to activate the package manager is this:

```
perl -MCPAN -e shell;
```

This will load the CPAN module, and let you search for, download, install, and manage the modules on your computer the same as PPM.

4.28.3 With Perl (cpanm)

The Perl module cpanm (CPAN Minus) is another alternative for installing modules from the CPAN library `http://search.cpan.org/~miyagawa/App-cpanminus-1.1001/lib/App/cpanminus.pm`.

cpanm can be installed and used like this on a UNIX-like system:

```
curl -L "http://cpanmin.us" >cpanm
chmod +x cpnam
./cpanm LWP::Bundle
```

One must have root privileges in order to install module in the system-wide directories, however alternatives exist such as local::lib, which allows regular users to install and use Perl modules in their home folder `http://search.cpan.org/~getty/local-lib-1.006007/lib/local/lib.pm`.

4.28.4 With Strawberry Perl (Windows systems)

Strawberry Perl also includes the CPAN module, so you can use the command above to activate the package manager.

The start menu, however, also includes a shortcut (with the name of `"CPAN Client"`) so that you don't have to go to a command line to do so.

A number of modules are already included in Strawberry Perl, beyond what comes with a normal version of Perl, or what comes with ActivePerl, so you may wish to check if the module you want is already installed before you start the CPAN client.

4.29 Using a module in your program

To incorporate a module into your program, use the Use keyword:

```
use Time::HiRes;
```

You can supply an optional list of the functions you want to use from this module, if you're worried that some of the function names in the module are too similar to functions you're already using:

```
use Time::Hires qw(time gmtime);
```

With that done, you can simply use the supplied functions as normal. Most modules have example programs within their documentation, and the best way to start using a module is to copy and adapt one of the example programs.

4.30 Finding documentation

The documentation for each module is installed in your documentation directory when you get a new module, or you can browse documentation on search.cpan.org[42] and perldoc.perl.org[43].

4.30.1 Unix systems

On Unix systems, the documentation is usually installed as *man* pages in section 3p, so that the command below will work:

```
man 3p Module::Name
```

[42] http://search.cpan.org/

[43] http://perldoc.perl.org/

`perldoc Module::Name` will also work.

If you want documentation that is browseable in a web browser, you can install Perldoc::Server as noted below.

4.30.2 Windows systems running ActivePerl

Module documentation is installed as HTML files in ActivePerl. To find those files, try looking in some of the following directories:

- C:\Perl\html\lib
- C:\Perl\html\site\lib

If you're having real trouble finding the HTML documentation for a module, you may be able to read the *.pm perl file yourself for POD comments, or use the pod2html tool yourself to generate the HTML file.

4.30.3 Windows systems running Strawberry Perl

Strawberry Perl does not install module documentation as either manpages or html files. Instead, you can run the perldoc command to display module documentation.

```
perldoc Module::Name
```

You can also use `Perldoc::Server` to display module documentation, as illustrated below.

4.30.4 Perldoc::Server

The Perldoc::Server module (which can be installed via CPAN) will provide a local server that will display html files "on the fly" from Perl's documentation and the documentation for installed modules. Install it, and the command

```
perldoc-server
```

will be in your path. Run it, and then browse to `http://localhost:7375/` in your web browser to see the documentation.

Note that the perldoc-server command must be running to provide the documentation using this method.

4.31 Contributing your own modules to CPAN

In the event that a module you need isn't available on CPAN, the usual answer is to write the module yourself and add it to CPAN. That way, nobody else needs to waste time creating the same functionality that you're already written.

See How to contribute modules to CPAN[44]

4.32 Section 4: CGI and Apache

Assuming you already have an Apache server (or compatible server that reads a shebang! line - more on this in a moment) and a perl installation running, it is fairly simple to start running a perl program on the internet.

First, you must have some way to access the program. Here we will deal with form data and submission, so we will assume that your form code in HTML has a property saying ACTION="*programname.cgi*".

4.33 The Initial Setup

CGI scripts begin like any other Perl program, with a "shebang", something like:

```
#!/usr/bin/perl
```

(see Perl Programming/First Programs[45] for details)

Next load the CGI module:

```
use CGI;
```

The CGI module makes our work easy because it has pre-programmed functions in it for internet use. Then we must create a handle to CGI - something that allows us to access the functions. We do this with:

```
my $query = CGI->new();
```

This means that the variable $query is loading the CGI standard functions.

Now that our program is setup using the CGI module, it should look something like this:

```
#!/usr/bin/perl

use CGI;
my $query = CGI->new();
```

So we have a program, it just doesn't do anything yet, and will actually cause a server error because the server has no output or any idea of what kind of output to display even if it had some.

[44] http://cpan.org/misc/cpan-faq.html#How_contribute_modules

[45] Chapter 0.2.3 on page 3

4.34 Retrieving Information

Before we tell the server what to do with our output, we need to retrieve our input. To do this, we use the $query variable we declared earlier. Say we have a text box in the form that is named "Name" and we want to find out what was typed there. To do this, we put the following line of code in our program:

```
my $Name = $query->param('Name');
```

Now, this line of code introduces us to the param() function (for "parameter"). The param() function can do quite a few handy tricks for us, all of them nice ways to retrieve our variables. It processes all of the http coding so all we get is a nice clean variable. Another note, you aren't required to use $Name as your variable. It's simply more convenient to only remember one name for the same variable. Still, use what's best for you.

4.35 Output

Now we must create our header information. CGI even makes THIS easy for us. Instead of memorizing a bunch of mime-type declarations (which you may do as well), all we must do is type:

```
print $query->header();
```

and it prints out our header information. A note about headers. Inside the parenthesis, we may specify parameters like cookies to send to the user's browser. This becomes very useful later. For now we will just stick to the headers.

The last thing you need to put (though the program will run, displaying a blank page without it) is some output. Let's simply have it display the user's name back to him/her. This would look like.

```
print " You said your name was: $Name";
```

4.36 The Finished Code

So we now have a complete program that processes a form, using only 6 lines of code. Isn't Perl great? The final code looks like this:

```
#!/usr/bin/perl

use CGI;
my $query = new CGI;

my $Name = $query->param('Name');

print $query->header();
print "You said your name was: ", $query->escapeHTML($Name);
```

When put into perspective, we can see that the $query variable is a very important connection to the CGI module as it tells perl that the function you are referencing belongs to CGI; Again, you may declare any variable name in the place of $query so long as you are consistent, though you will find many developers use $query or $q. Also note the use of the escapeHTML method to avoid any HTML injection[46] problems.

Final note. Make sure you change /usr/bin/perl to the path of your perl installation (assuming that is not it) so perl will execute properly

4.37 Frameworks

There are a number of CGI frameworks to help with common CGI programming tasks:

- CGI::Application[47]
- Catalyst[48]

mod_perl is an optional module for Apache. It embeds a Perl[49] interpreter into the Apache server, so that dynamic content produced by Perl scripts can be served in response to incoming requests, without the significant overhead of re-launching the Perl interpreter for each request. As Lincoln D. Stein defined mod_perl in his words:

> mod_perl is more than CGI scripting on steroids. It is a whole new way to create dynamic content by utilizing the full power of the Apache web server to create stateful sessions, customized user authentication systems, smart proxies and much more. Yet, magically, your old CGI scripts will continue to work and work very fast indeed. With mod_perl you give up nothing and gain so much!

mod_perl can emulate a Common Gateway Interface[50] (CGI) environment, so that existing Perl CGI scripts can benefit from the performance boost without having to be re-written.

Unlike CGI (and most other web application environments), mod_perl provides complete access to the Apache API, allowing programmers to write handlers for all phases in the Apache request cycle, manipulate Apache's internal tables and state mechanisms, share data between Apache processes or threads, alter or extend the Apache configuration file parser, and add Perl code to the configuration file itself, among other things.

[46] http://en.wikipedia.org/wiki/HTML%20injection
[47] http://cgi-app.org/index.cgi
[48] http://www.catalystframework.org/
[49] http://en.wikibooks.org/wiki/Perl
[50] http://en.wikibooks.org/wiki/Common%20Gateway%20Interface

4.38 External links

- Main website[51]
- Why mod_perl?[52]
- The magic of mod_perl[53]
- Writing Apache Modules with Perl and C[54]
- The mod_perl Developer's Cookbook[55]
- Practical mod_perl[56]
- mod_perl2 User's Guide [57]

4.39 The Initial Setup

4.40 Retrieving Information

4.41 Output

4.42 The Finished Code

4.43 Section 5: Perl and beyond

Perl 6[58] will separate parsing and compilation and runtime, making the virtual machine more attractive to developers looking to port other languages to the architecture.

Parrot[59] is the Perl6 runtime, and can be programmed at a low level in Parrot assembly language. Parrot exists in a limited form as of June, 2003, and a small number of languages (Jako, Cola, Basic, Forth and a subset of Perl 6) exist simply to be 'compiled' down to Parrot assembly language opcodes.

While Perl6 is being developed, the best way to stay informed about what's happening is to keep an eye on the front-page of `http://www.perl.com/` and look out for articles. As each new language-feature is being developed, it gets discussed on Perl.com and the associated mailing lists, so subscribe to some of those to see glimpses of what Perl6 will be like.

[51] http://perl.apache.org/
[52] http://www.perl.com/pub/a/2002/02/26/whatismodperl.html
[53] http://www.revsys.com/writings/modperl.html
[54] http://www.modperl.com/
[55] http://www.modperlcookbook.org/
[56] http://modperlbook.org/
[57] http://modperl2book.org/
[58] http://en.wikipedia.org/wiki/Perl%206
[59] http://en.wikipedia.org/wiki/Parrot%20virtual%20machine

4.43.1 Obfuscated code

Some people claim Perl stands for 'Pathologically Eclectic Rubbish Lister' due to the high use of meaningful punctuation characters in the language syntax.

In common with C programming language[60], obfuscated code competitions are an interesting feature of the Perl culture. Similar to obfuscated code but with a different purpose, Perl Poetry is the practice of writing poems that can actually be compiled by perl. This practice is fairly unique to Perl, due to the large number of regular English words used in the language. New poems can regularly be seen in the Perl Poetry section of perlmonks.org[61].

4.43.2 Just another Perl Hacker

Your mission, should you choose to accept it, is to write a one-liner perl script which displays the phrase "Just another Perl hacker," (including the comma, and capitalization as shown). If successful, you win the right to use it as an email signature identifying yourself as a Perl hacker. Entries will be judged on how smart-ass the code is. Around 100 of the first JAPHs and some funky obfu Perl can be seen on CPAN[62].

4.43.3 Acme

There's always a place in Perl for odd modules, and one such place is the Acme:: namespace. If you have a module which knows how long a piece of string is, or one which converts your perl script into an image of Che Guevara, post it here.

4.43.4 Golf

Perl is a very compact language. So compact, that some have even create a game around perl's terseness called perlgolf. In perlgolf, you are given a problem to solve. You must solve it in the fewest number of characters possible. A scorecard is kept, and after 18 "holes", a winner is announced.

4.44 Section 6: Sample code

This script counts the number of occurences of each letter in a file:

```
#!/usr/bin/perl
# always enable compiler warnings, as they may highlight potential
 trouble
use warnings;
# let,s ask the compiler to be more strict, make sure we declare our
 variables etc.
use strict;
```

[60] http://en.wikibooks.org/wiki/Programming%3AC

[61] http://www.perlmonks.org/index.pl?node=Perl%20Poetry

[62] http://www.cpan.org/misc/japh

```
# This statement prompts the user for a filename to read from.
print "What file would you like to read from?\n";

# This statement assigns whatever is given on the standard
 input(usually your keyboard) to a scalar
# variable named $filename and removes the newline character that is
 included by default.
chomp (my $filename = <STDIN>);

# This line opens the file referred to in $filename for input via a
 lexical filehandle
# stored in a scalar variable named "$file".
open my $file, "<", $filename or die "Can,t open ,$filename, for
 reading: $^E\n";

# This loop goes through each line of the file, splits the line into
 separate characters and
# increments the number of occurrences of each letter using a hash
 called "%chars".

my %chars;
while(<$file>) {
    $_ = lc($_); # convert everything to lowercase
    my @characters = split (//, $_);  # Store list of characters in
 an array
    foreach (@characters) {
        if(/\w/) {                    # Ignore all characters except
 letters and numbers
            $chars{$_}++;
        }
    }
}
close $file;

# This loop goes through each letter in the %chars hash and prints a
 report informing the user of
# how many times each letter occurred.

foreach my $key (sort keys %chars) {
    if($chars{$key} == 1) {
        print "$key appeared once.\n";
    } else {
        print "$key appeared $chars{$key} times.\n";
    }
}
```

If you executed this program on a file containing the sentence "The quick, brown fox jumps over the lazy dog.", you would see this as output:

```
a appeared once.
b appeared once.
c appeared once.
d appeared once.
e appeared 3 times.
f appeared once.
g appeared once.
h appeared 2 times.
i appeared once.
j appeared once.
k appeared once.
l appeared once.
m appeared once.
n appeared once.
o appeared 4 times.
p appeared once.
q appeared once.
```

```
r appeared 2 times.
s appeared once.
t appeared 2 times.
u appeared 2 times.
v appeared once.
w appeared once.
x appeared once.
y appeared once.
z appeared once.
```

63

Hi-Lo: A simple game written in perl that asks you for a guess between 1 and 100 and tells you if you are too high or low.

```
use warnings;
use strict;

$| = 1;

print "Enter number of games to play: ";
chomp(my $Num_Games = <STDIN>);

my $Num_Guesses = 0;
for my $gameno (1 .. $Num_Games) {
        my $number = 1 + int rand 100;

        my $guess;
        do {
                print "Enter guess from 1 to 100: ";
                chomp($guess = <STDIN>);
                ++$Num_Guesses;

                if ($guess < $number) {
                        print "Higher!\n";
                } elsif ($guess > $number) {
                        print "Lower!\n";
                }
        } until $guess == $number;

        print "Correct!\nAverage guesses per game: ",
        $Num_Guesses / $gameno, "\n\n";
}

print "Games played: $Num_Games\n";
```

64

[63] http://en.wikibooks.org/wiki/Category%3APerl%20Programming

[64] http://en.wikibooks.org/wiki/Category%3APerl%20Programming

4.45 Section 7: Reference

4.46 String functions

4.46.1 chomp

Action

Removes the last characters from a string only if they're recognized as a record separator (e.g. a newline character)

Returns

?

Syntax

chomp($String = $_);

Example

```
chomp; # removes the last character from $_ if it is a record
separator
chomp(); # (same)
chomp($String); # removes the last character from $String if it is a
record separator
```

See Also

- chop[65] - To remove the last character from a string

4.46.2 chop

Action

Removes the last character from a string regardless

Returns

?

[65] Chapter 4.46.2 on page 95

Syntax

chop($String = $_);

Example

```
chop; # removes the last character from $_
chop(); # (same)
chop($String); # removes the last character from $String
```

See Also

- chomp[66] - To remove the last character from a string if it is a record seperator

Removes the last character from a string (e.g. removes the newline characters when reading from a file)

4.46.3 chr

```
print chr(65);  # Prints a capital A
```

Gets an ASCII character, given it's code

4.46.4 crypt

```
# One-way hash function
my $HashedWord = crypt($Word, $Salt);
```

(See also MD5[67])

The salt string needs only be 2 characters long, and provides a way of randomising the hash, such that the same word can produce several different hashes, if used with different values of $Salt;!

4.46.5 hex

```
print hex(11);  # Prints B
```

Converts a number to hexadecimal

Other way around - converts hex to number: print hex(11); # prints 17

you can use

print sprintf("%X",11); # Prints B

[66] Chapter 4.46.1 on page 95

[67] `http://search.cpan.org/~gaas/MD5-2.03/MD5.pm`

4.46.6 index

Search for one string within another. (see *rindex* to search from end-to-start)

```
$Result = index($Haystack, $Needle);
$Result = index($Haystack, $Needle, $StartPosition);

index("Some text", "bleh"); # Returns -1 (not found)
index("Some text", "Some"); # Returns 0 (first character)
index("Some text", "text"); # Returns 5 (sixth character)
```

The special variable $[always gets added to the return value, but $[is normally 0, and the manual recommends leaving it at 0.

4.46.7 lc

```
$Lowercase = lc($String);
```

Converts a string to lower-case

4.46.8 lcfirst

Converts the first character of a string to lowercase

4.46.9 length

```
print "String is " . length($String) . " characters long\n";
```

Returns the length of a string

4.46.10 oct

```
print oct(8);  # Prints 10
```

Converts a number to octal

4.46.11 ord

Converts a character to its number.

```
print ord("A"); # prints 65
```

4.46.12 pack

Takes a list and converts it into a string using a supplied set of rules.

```
my $String = pack($Template, @ListOfNumbers);
my $String = pack("CCCC",65,66,67,68); # Result: "ABCD"
```

$Template can be made up of:

```
   a    A string with arbitrary binary data, will be null padded.
   A    An ascii string, will be space padded.
   Z    A null terminated (asciz) string, will be null padded.
```

```
   b    A bit string (ascending bit order inside each byte, like
vec()).
   B    A bit string (descending bit order inside each byte).
   h    A hex string (low nybble first).
   H    A hex string (high nybble first).
```

```
   c    A signed char value.
   C    An unsigned char value.  Only does bytes.  See U for
Unicode.
```

```
   s    A signed short value.
   S    An unsigned short value. (Exactly 16 bits unless you use the
! suffix)
```

```
   i    A signed integer value.
   I    An unsigned integer value. (At least 32 bits wide,
machine-dependant)
```

```
   l    A signed long value.
   L    An unsigned long value. (Exactly 32 bits unless you use the
! suffix)
```

```
   n    An unsigned short in "network" (big-endian) order.
   N    An unsigned long in "network" (big-endian) order.
   v    An unsigned short in "VAX" (little-endian) order.
   V    An unsigned long in "VAX" (little-endian) order. (Exactly 16
bits and 32 bits respectively)
```

```
   q    A signed quad (64-bit) value.
   Q    An unsigned quad value. (Only available if your system
supports 64-bit integers and Perl has been compiled to support them)
```

```
   f    A single-precision float in the native format.
   d    A double-precision float in the native format.
```

```
   p    A pointer to a null-terminated string.
   P    A pointer to a structure (fixed-length string).
   u    A uuencoded string.
   U    A Unicode character number.  Encodes to UTF-8 internally.
```

```
   w    A BER compressed integer.  Its bytes represent an unsigned
integer in base 128, most significant digit first, with as few digits
as possible.  Bit eight (the high bit) is set on each byte except the
last.
```

```
   x    A null byte.
   X    Back up a byte.
   @    Null fill to absolute position.
```

Each letter may optionally be followed by a number giving a repeat count.

The integer types s, S, l, and L may be immediately followed by a ! suffix to signify native shorts or longs

4.46.13 reverse

Reverses a string (in scalar context) or a list (in list context):

```
my @ReversedList = reverse(@List);
```

```
# As commonly seen in Perl programs:
foreach( reverse( sort( @List )))
{
...
}
```

```
my $ReversedString = reverse($String);
```

```
my @List = ("One ", "two ", "three...");
my $ReversedListAsString = reverse(@List); # Prints "...eerht owt
enO"
```

4.46.14 rindex

Search for one string within another, starting at the end of the string.

```
$Result = rindex($Haystack, $Needle);
$Result = rindex($Haystack, $Needle, $StartPosition);

rindex("Some text", "bleh"); # Returns -1 (not found)
rindex("Some text", "Some"); # Returns 0 (first character)
rindex("abbbbb", "b");       # Returns 5 (first "b" found, when
starting at the end)
```

4.46.15 sprintf

Prints a formatted string:

```
my $Text = sprintf("%d / %d is %08.5f", 1, 3, 1/3); # Result: "10 /
3 is 003.33333"
```

```
sprintf("Character: %c", 65);
sprintf("String %s", "Hello");
sprintf("Signed integer: %d", 15);
sprintf("Unsigned integer: %u", 15);
sprintf("Unsigned int (in octal): %o", 15);
sprintf("Unisgned int (in hex): %x", 15);      # Use %X to get
upper-case output
sprintf("Binary number: %b", 15);
sprintf("Scientific notation: %e", 5000);      # Use %E to get
upper-case output
sprintf("Floating point number: %f", 1/3);     # 0.3333333
sprintf("Floating point number: %g", 1/3);     # Decides between
scientific and float.  %G is uppercase
sprintf("Pointer: %p", $Variable);
```

Use %% to get a percent-sign.

Use %n to request the number of characters written so far, and put it into the next variable in the list. You may want to check that user-supplied formatting rules don't contain this code.

```
sprintf("%02d", $Minutes);  # Forces leading zeros to make the
string 2 characters long
sprintf("%1.5f", $Number);  # Limits the number of decimal places
```

4.46.16 substr

Return part of a string (a *substring*)

Format: substr *string start-position length*

start-position is zero-based.

A negative number starts from the end of the string.

```
$FirstLetter   = substr($Text, 0, 1);   # First letter
$First3Letters = substr($Text, 0, 3);   # First three letters
$Last3Letters  = substr($Text, -3);     # Last three letters
```

You can use **substr** on the left side of an assignment statement to change part of a string. This can actually shorten or lengthen the string.

```
 $text = 'cat dog';
 substr ($mystring, 3, 1) = ' and ';  # $text now contains 'cat and
dog'
```

4.46.17 uc

```
$Uppercase = uc($String);
```

Converts a string to upper-case

4.46.18 ucfirst

Converts the first character of a string to uppercase

4.47 Numeric functions

4.47.1 abs

Returns the absolute(positive) value of a number

```
$Number = abs(-100); # Returns 100;
```

4.47.2 atan2

```
# Converts cartesian(x,y) coordinates into an angle
$Number = atan2($Y, $X);
```

4.47.3 cos

```
# Returns the cosine of an angle (radians)
$Number = cos($Angle);  # Cosine = Adjacent/Hypotenuse
```

4.47.4 exp

```
# Raises e to a specified power
$Number = exp(2); # Returns e^2
```

```
e ≈ 2.71828183 more about e^68
```

4.47.5 hex

```
# Interprets a string as hexidecimal, and returns its value
$Number = hex("10"); # Returns 16
$Number = hex("0xFF"); # Returns 255
```

4.47.6 int

Rounds a number towards zero, returning an integer

```
$Number = int(-1.6);  # Returns -1
$Number = int(0.9);   # Returns 0
$Number = int(28.54); # Returns 28
```

4.47.7 log

```
# Returns the natural logarithm of a number
$Number = log(2.71828183);   # Returns 1
$Number = exp(log($X));      # Returns $X
$Number = log($X) / log(10); # Returns log10($X). Alternately, you
can use the log10() function in the POSIX module
$Number = log($X) / log(15); # Returns log to the base 15 of $X
```

4.47.8 oct

```
# Interprets a string as octal, and returns its value
$Number = oct("10"); # Returns 8
$Number = oct("21"); # Returns 17
```

4.47.9 rand

```
# Gets a random number (may automatically call srand() if that's not
been done)
$Number = rand();  # Returns a random number from 0 to 1
$Number = int(rand(800));  # Returns a random integer from 0 to 799
$Number = 1 + int(rand(999));  # Returns a random integer from 1 to
999
```

4.47.10 sin

```
# Returns the sine of an angle (radians)
$Number = sin($Angle);  # Sine = Opposite/Hypotenuse
```

4.47.11 sqrt

```
# Returns the square-root of a number
$Number = sqrt(4);                  # Returns 2
$Number = sqrt($X ** 2 + $Y ** 2);  # Returns the diagonal distance
across a $X x $Y rectangle
```

```
See the Math::Complex module if you need to take roots of negative
numbers;
```

4.47.12 srand

```
# Seeds (sets-up) the random-number generator
srand();
```

Version-dependant, and older versions of Perl are not guaranteed to have a good seed value. See the Math::TrulyRandom module for more possibilities. The current version of Perl uses the urandom device if it's available.

4.48 Array functions

4.48.1 pop

```
$LastElement = pop(@MyArray);
```

Take the last element from an array

4.48.2 push

```
push(@MyArray, "Last element");
push(@MyArray, "several", "more", "elements");
```

Push a list of elements onto the end of an array

4.48.3 shift

```
shift(@MyArray); #Delete the first element
$FirstElement = shift(@MyArray); #Delete the first element, load it
into $FirstElement instead
```

Take the first element out of an array

4.48.4 splice

```
# Removes elements from an array, optionally replacing them with a
new array
splice(@Array); # Removes all elements from array
splice(@Array, 10); # Removes from element 10 to the end of the
array
splice(@Array, -10); # Removes the last 10 elements of the array
splice(@Array, 0, 10); # Removes the first 10 elements of the array
@NewArray = splice(@Array, 0, 10); # Removes the first 10 elements
of the array and returns those 10 items
splice(@Array, 0, 10, @Array2); # Replaces the first 10 elements of
the array with Array2
```

4.48.5 unshift

```
unshift(@MyArray, "New element");
unshift(@MyArray, "several", "more", "elements");
```

Add a list of elements onto the beginning of an array

4.49 List functions

4.49.1 grep

```
# Returns a list of elements for which an expression is true
@TextFiles = grep(/\.txt$/, @AllFiles);
$NumberOfTextFiles = grep(/\.txt$/, @AllFiles);
```

```
# Can use a block of code instead of an expression
@TextFiles = grep({return(substr($_, -3) eq "txt");}, @AllFiles);
```

4.49.2 join

```
# Joins the items of a list into a single string
$OneItemPerLine = join( "\n", @List);
$EverythingBunchedTogether = join( "", @List);
$Filename = join( "/", ($Directory, $Subdirectory, $Filename));
```

4.49.3 map

```
# Evaluates a block of code for each item in a list, and returns
# a list of the results
@UppercaseList = map(uc, @List);
@Numbers = map {"Number $_"} 1..100;
```

4.49.4 reverse

```
# Reverses the order of a list
@ReversedList = reverse(@List);
# In scalar context, concatenates the list and then reverses the
string
$ReversedString = reverse('foo','bar','baz'); # gives 'zabraboof'
```

4.49.5 sort

```
# Sorts the elements in a list
@AsciiSort = sort(@RandomList);
@AsciiSort = sort @RandomList;
foreach $Item (sort @RandomList)
  {...}
```

```
# Can specify a function to decide the sort order
@CaseInsensitiveSort = sort {uc($a) cmp uc($b)} @RandomList;
```

```
@NumericSort = sort {$a <=> $b} @RandomList;
@CustomSort = sort custom_function_name @RandomList;
```

4.49.6 unpack

Unpacks a string into a list - see the templates available for the pack() function for details

4.50 Associative array functions

4.50.1 delete

```
#Remove an element from a hash
%h = ('a'=>1, 'cow'=>'moo', 'b'=>2);
delete $h{cow};
# %h now contains ('a'=>1, 'b'=>2)
```

4.50.2 each

```
#Return the 'next' key/value pair (in a random order)
while (($key, $value) = each (%hash)){
   print "$key => $value\n";
}
```

4.50.3 exists

```
 #Tests whether or not a key exists in a hash (even if the value for
that key is undef)
 if (exists $hash{$key}){
    print "\%hash contains a value for key '$key'\n";
 }
```

4.50.4 keys

```
 #Returns a list of all keys from the hash, in same 'random' order
as each
 foreach $key (keys %hash){
    print "$key => $hash{$key}\n";
 }
```

4.50.5 values

```
 #Returns a list of all values from the hash, in same 'random' order
as keys
 foreach $value (values %hash){
    print "\%hash contains a value '$value'\n";
 }
```

4.51 Input and output functions

4.51.1 binmode

4.51.2 close

```
#closes a filehandle when it is no longer needed
close(STDERR); #hide debugging info from the user
```

4.51.3 closedir

```
# Close a directory open by opendir
closedir(DIRHANDLE);
```

4.51.4 dbmclose

4.51.5 dbmopen

4.51.6 die

Exits the program, printing to "STDERR" the first parameter and the current file and line. Used to trap errors.

```
die "Error: $!\n" unless chdir '/';
```

4.51.7 eof

```
eof FILEHANDLE
eof()
eof
```

This function returns true if the next read on FILEHANDLE would return end-of-file, or if FILEHANDLE is not open. FILEHANDLE may be an expression whose value gives the real filehandle, or a reference to a filehandle object of some sort. An eof without an argument returns the end-of-file status for the last file read. An eof() with empty parentheses () tests the ARGV filehandle (most commonly seen as the null filehandle in <>). Therefore, inside a while (<>) loop, an eof() with parentheses will detect the end of only the last of a group of files. Use eof (without the parentheses) to test each file in a while (<>) loop. For example, the following code inserts dashes just before the last line of the last file:

```
while (<>) {
    if (eof()) {
        print "-" x 30, "\n";
    }
    print;
}
```

On the other hand, this script resets line numbering on each input file:

```
# reset line numbering on each input file
while (<>) {
    next if /^\s*#/;        # skip comments
    print "$.\t$_";
} continue {
    close ARGV if eof;      # Not eof()!
}
```

Like "$" in a sed program, eof tends to show up in line number ranges. Here's a script that prints lines from /pattern/ to end of each input file:

```
while (<>) {
    print if /pattern/ .. eof;
}
```

Here, the flip-flop operator (..) evaluates the pattern match for each line. Until the pattern matches, the operator returns false. When it finally matches, the operator starts returning true, causing the lines to be printed. When the eof operator finally returns true (at the end of the file being examined), the flip-flop operator resets, and starts returning false again for the next file in @ARGV

4.51.8 fileno

4.51.9 flock

4.51.10 format

4.51.11 getc

4.51.12 print

Prints the parameters given.

Discussed in the following sections:

Digression on print in *Strings* section[69]

[69] Chapter 0.7.3 on page 10

4.51.13 printf

4.51.14 read

4.51.15 readdir

4.51.16 rewinddir

4.51.17 seek

4.51.18 seekdir

4.51.19 select

4.51.20 syscall

4.51.21 sysread

4.51.22 sysseek

4.51.23 syswrite

4.51.24 tell

4.51.25 telldir

4.51.26 truncate

4.51.27 warn

4.51.28 write

4.52 Functions for working with fixed length records

4.52.1 pack

See the entry for **pack** further up the page

4.52.2 read

```
# Reads data from a file-handle
read(FILEHANDLE, $StoreDataHere, $NumberBytes);
```

```
# Returns the number of bytes read
$NumberBytesRead = read(FILEHANDLE, $StoreDataHere, $NumberBytes);
```

```
# Optional offset is applied when the data is stored (not when reading)
read(FILEHANDLE, $StoreDataHere, $NumberBytes, Offset);
```

4.52.3 syscall

```
# Runs a system command
syscall( $Command, $Argument1, $Argument2, $Argument3);
```

```
# (maximum 14 arguments)
$ReturnValue = syscall($Command);
```

4.52.4 sysread

4.52.5 syswrite

4.52.6 unpack

```
# See the pack function for details (unpack does the opposite!)
unpack($Template, $BinaryData);
```

4.52.7 vec

4.53 Filesystem functions

4.53.1 -X

```
if( -r $FullFilename) // File is readable by effective uid/gid.
if( -w    $FullFilename) // File is writable by effective uid/gid.
if( -x    $FullFilename) // File is executable by effective uid/gid.
if( -o    $FullFilename) // File is owned by effective uid.
```

```
if( -R    $FullFilename) // File is readable by real uid/gid.
if( -W    $FullFilename) // File is writable by real uid/gid.
if( -X    $FullFilename) // File is executable by real uid/gid.
if( -O    $FullFilename) // File is owned by real uid.
```

```
if( -e    $FullFilename) // File exists.
if( -z    $FullFilename) // File has zero size.
if( -s    $FullFilename) // File has nonzero size (returns size).
```

```
if( -f    $FullFilename) // File is a plain file.
if( -d    $FullFilename) // File is a directory.
if( -l    $FullFilename) // File is a symbolic link.
if( -p    $FullFilename) // File is a named pipe (FIFO), or
Filehandle is a pipe.
if( -S    $FullFilename) // File is a socket.
if( -b    $FullFilename) // File is a block special file.
if( -c    $FullFilename) // File is a character special file.
if( -t    $FullFilename) // Filehandle is opened to a tty.
if( -u    $FullFilename) // File has setuid bit set.
if( -g    $FullFilename) // File has setgid bit set.
if( -k    $FullFilename) // File has sticky bit set.
```

```
if( -T    $FullFilename) // File is an ASCII text file.
if( -B    $FullFilename) // File is a "binary" file (opposite of
-T).
```

```
$Age = -M $FullFilename; // Age of file in days when script started.
$Age = -A $FullFilename; // Same for access time.
$Age = -C $FullFilename; // Same for inode change time.
```

4.53.2 chdir

```
chdir $Directory;
chdir $Directory || die("Couldn't change directory");
```

4.53.3 chmod

```
chmod 0744 $File1;
chmod 0666 $File1, $File2, $File3;
# 0 for octal, at the beginning of a number
```

```
        | Owner | Group | Others |
Execute |   4   |   4   |   4    |
Write   |   2   |   2   |   2    |
Read    |   1   |   1   |   1    |
======--+======-+======-+======--+
Total   |       |       |        |
```

4.53.4 chown

```
# Change the owner of a file
chown($NewUserID, $NewGroupID, $Filename);
chown($NewUserID, $NewGroupID, $File1, $File2, $File3);
```

```
chown($NewUserID, -1, $Filename); # Leave group unchanged
chown(-1, $NewGroupID, $Filename); # Leave user unchanged
```

4.53.5 chroot

```
chroot $NewRootDirectory;
```

Sets the root directory for the program, such that the "/" location refers to the specified directory.

Program must be running as root for this to succeed.

4.53.6 fcntl

4.53.7 glob

```
#Expands filenames, in a shell-like way
my @TextFiles = glob("*.txt");
```

See also File::Glob

4.53.8 ioctl

4.53.9 link

```
# Creates a link to a file
link($ExistingFile, $LinkLocation);
link($ExistingFile, $LinkLocation) || die("Couldn't create link");
```

4.53.10 lstat

Identical to stat(), except that if given file is symbolic link, stat link not the target.

4.53.11 mkdir

```
mkdir $Filename || die("Couldn't create directory");
mkdir $Filename, 0777; # Make directory with particular
file-permissions
```

4.53.12 open

```
open(my $FileHandle, $Filename) || die("Couldn't open file");
open(my $fp, "<", $Filename);   # Read from file
open(my $fp, ">", $Filename);   # Write to file
open(my $fp, ">>", $Filename);  # Append to file
```

```
open(my $fp, "<$Filename");     # Read from file
open(my $fp, ">$Filename");     # Write to file
open(my $fp, ">>$Filename");    # Append to file
```

```
open(my $fp, "<", "./   filename with whitespace   \0");
open(my $fp, "<", "./->filename with reserved characters\0");
```

```
open(my $fp, "$Program |");     # Read from the output of another
program
open(my $fp, "| $Program");     # Write to the input of another
program
```

```
open(my $fp, "<", "-");         # Read from standard input
open(my $fp, ">", "-");         # Write to standard output
```

4.53.13 opendir

```
opendir(my $DirHandle, $Directory) || die("Couldn't open
directory");
while (my $Filename = readdir $DirHandle){
  # Do something with $Filename in $Directory
}
closedir($DirHandle);
```

```
opendir(DIR, $Directory) || die("Couldn't open directory");
foreach(readdir(DIR)){
  # Do something with $_ in $Directory
}
closedir(DIR);
```

4.53.14 readlink

```
# Finds the value of a symbolic link
$LinkTarget = readlink($LinkPosition);
```

4.53.15 rename

```
rename $OldFile, $NewFile or die("Couldn't move file");
```

May work differently on non-*nix operating systems, and possibly not at all when moving between different filesystems. See

Figure 2

for more complicated file operations.

4.53.16 rmdir

```
rmdir $Filename || die("Couldn't remove directory");
```

4.53.17 stat

```
@FileStatistics = stat($Filename);
```

```
$DeviceNum    = $FileStatistics[0]; # device number of filesystem
$Inode        = $FileStatistics[1]; # inode number
$FileMode     = $FileStatistics[2]; # (type and permissions)
$NumHardLinks = $FileStatistics[3]; # number of (hard) links to the
file
$UserID       = $FileStatistics[4]; # numeric user ID
$GroupID      = $FileStatistics[5]; # numeric group ID
$DeviceIdent  = $FileStatistics[6]; # Device identifier (special
files only)
$SizeBytes    = $FileStatistics[7];
$AccessTime   = $FileStatistics[8]; # seconds since the epoch
$ModifyTime   = $FileStatistics[9];
```

```
$ChangeTime   = $FileStatistics[10];
$BlockSize    = $FileStatistics[11];
$NumBlocks    = $FileStatistics[12];
```

4.53.18 symlink

```
# Creates a new filename symbolically linked to the old filename
symlink($OldFilename, $NewFilename);
symlink($OldFilename, $NewFilename) || die("Couldn't create
symlink");
eval(symlink($OldFilename, $NewFilename));
```

4.53.19 umask

```
# Sets or returns the umask for the process.
my $UMask = umask();
umask(0000); # This process can create any type of files
umask(0001); # This process can't create world-readable files
umask(0444); # This process can't create executable files
```

4.53.20 unlink

```
# Deletes a file
unlink $Filename;
unlink $Filename || die("Couldn't delete file");
unlink $File1, $File2, $File3;
(unlink($File1, $File2, $File3) == 3) || die("Couldn't delete
files");
```

4.53.21 utime

```
# Updates the modification times of a list of files
my $AccessTime = time();
my $ModificationTime = time();
```

```
utime($AccessTime, $ModificationTime, $Filename);
my $NumFilesChanged = utime($AccessTime, $ModificationTime, $File1,
$File2, $File3);
```

4.54 Program functions

4.54.1 caller

Returns information about the current function call stack. In scalar context, returns only the name of the package from where the current subroutine was called. In list context, returns the package, filename, and line number. In list context with a numeric argument passed, returns several pieces of information (see below). The argument represents how many levels in the call stack to go back.

```
#!/usr/bin/perl
```

```
foo();
sub foo {
   $package = caller; #returns 'main'
   ($package, $filename, $line) = caller; #returns 'main', the file
name, and 3
   # Line below returns all 10 pieces of info. (Descriptions
self-explanatory from variable names)
   ($package, $filename, $line, $subroutine, $hasargs, $wantarray,
$evaltext, $is_require, $hints, $bitmask) =
      caller(0);
 }
```

4.54.2 import

There is no actual 'import' function. Rather, it is a convention when writing a module to create a subroutine named 'import' which populates the current namespace with that module's needed variables and/or methods.

The standard 'Exporter' module provides an import method if your class has it as a base class.

4.54.3 package

Declares all lines that follow (until EOF or the next package statement) to belong to the given package's namespace.

```
#!/usr/bin/perl

$x = 5;  #sets $main::x

package Foo;
$x = 5;  #sets $Foo::x
sub bar { #defines &Foo::bar
   print "hello world";
}

package Temp;
$x = 5; #sets $Temp::x
```

4.54.4 require

includes the specified module's code into the current program. The module can be specified either with an absolute or relative path, or with a bareword. If a bareword is given, a '.pm' extention is added, and :: is replaced with the current operating system's path seperator:

```
require Foo::Bar;
#identical to:
require 'Foo/Bar.pm';
```

4.54.5 use

Requires and imports the given module or pragma, at compile time. The line

```
use Foo qw/bar baz/;
```

is identical to:

```
BEGIN {
   require Foo;
   import Foo qw/bar baz/;
}
```

4.55 Misc functions

4.55.1 defined

```
#returns true if argument is not undef
$x = 0;
print "X defined\n" if defined $x; #prints
print "Y defined\n" if defined $y; #does not print
```

4.55.2 dump

4.55.3 eval

```
eval('$a=30;$b=40;');
print $a,$b;
```

4.55.4 formline

4.55.5 local

```
#assigns temporary value to global variable for duration of lexical
scope
$x = 5;
print "x = $x\n"; # 5
{
  local $x = 10;
  print "x = $x\n"; # 10
}
print "x = $x\n"; # 5
```

4.55.6 my

```
#creates new lexical (ie, not global) variable
$x = 5;  #refers to $main::x
{
  my $x = 10;
  print "x = $x\n"; # the lexical - 10
  print "main's x = $main::x\n" # the global - 5
}
print "x = $x\n";  #the global, because no lexical in scope - 5
```

4.55.7 reset

```
#resets hash's internal pointer, to affect lists returned by each
while ($k, $v = each %h){
  print "$k = $v\n";
  last if ($i++ == 2);
}
#if another each done here, $k,$v will pick up where they left off.
reset %h
#now each will restart from the beginning.
```

4.55.8 scalar

```
#forces scalar context on an array
@sizes = (scalar @foo, scalar @bar);
#creates a list of the sizes of @foo and @bar, rather than the
elements in @foo and @bar
```

4.55.9 undef

```
#undefines an existing variable
$x = 5;
undef $x;
print "x = $x\n" if defined $x; #does not print
```

4.55.10 wantarray

```
#returns 'true', 'false', or undef if function that called it was
called in list, scalar, or void context, respectively.
sub fctn {
   my @vals = (5..10);
   if (wantarray) {
      return @vals;
   } elsif (defined wantarray) {
      return $vals[0];
   } else {
      warn "Warning!  fctn() called in void context!\n";
   }
}
```

4.56 Processes

4.56.1 alarm

4.56.2 exec

4.56.3 fork

```
#clones the current process, returning 0 if clone, and the process
id of the clone if the parent
my $pid = fork();
if ($pid == 0) {
   print "I am a copy of the original\n";
} elsif ($pid == -1)  {
   print "I can't create a clone for some reason!\n";
} else {
   print "I am the original, my clone has a process id of $pid\n";
}
```

4.56.4 getpgrp

4.56.5 getppid

4.56.6 getpriority

4.56.7 kill

4.56.8 pipe

4.56.9 qx/STRING/

4.56.10 setpgrp

4.56.11 setpriority

4.56.12 sleep

4.56.13 system

4.56.14 times

4.56.15 wait

4.56.16 waitpid

4.57 Modules

4.57.1 do

4.57.2 import

4.57.3 no

4.57.4 package

4.57.5 require

4.57.6 use

4.58 Classes and objects

See also Perl Objects[70]

[70] Chapter 4.25.3 on page 82

4.58.1 bless

4.58.2 dbmclose

4.58.3 dbmopen

4.58.4 package

4.58.5 ref

4.58.6 tie

4.58.7 tied

4.58.8 untie

4.58.9 use

4.59 Sockets

4.59.1 accept

4.59.2 bind

4.59.3 connect

4.59.4 getpeername

4.59.5 getsockname

4.59.6 getsockopt

4.59.7 listen

4.59.8 recv

4.59.9 send

4.59.10 setsockopt

4.59.11 shutdown

4.59.12 socket

4.59.13 socketpair

4.60 Login information

4.60.1 endgrent

4.60.2 endhostent

4.60.3 endnetent

```
@TimeParts = gmtime();
@TimeParts = gmtime($Time);
```

```
$Seconds    = $TimeParts[0]; # 0-59
$Minutes    = $TimeParts[1]; # 0-59
$Hours      = $TimeParts[2]; # 0-23
$DayOfMonth = $TimeParts[3]; # 1-31
$Month      = $TimeParts[4]; # 0-11
$Year       = $TimeParts[5]; # Years since 1900
$DayOfWeek  = $TimeParts[6]; # 0:Sun 1:Mon 2:Tue 3:Wed 4:Thu 5:Fri
6:Sat
$DayOfYear  = $TimeParts[7]; # 1-366
```

4.62.2 localtime

Converts a timestamp to local time

```
@TimeParts = localtime();
@TimeParts = localtime($Time);
```

```
$Seconds    = $TimeParts[0]; # 0-59
$Minutes    = $TimeParts[1]; # 0-59
$Hours      = $TimeParts[2]; # 0-23
$DayOfMonth = $TimeParts[3]; # 1-31
$Month      = $TimeParts[4]; # 0-11
$Year       = $TimeParts[5]; # Years since 1900
$DayOfWeek  = $TimeParts[6]; # 0:Sun 1:Mon 2:Tue 3:Wed 4:Thu 5:Fri
6:Sat
$DayOfYear  = $TimeParts[7]; # 1-366
```

4.62.3 time

```
$Time = time();
```

Returns number of seconds since an epoch (which is system-dependant, but may be Jan 1 1970)

See also ../Time::Hires/[71]

4.62.4 times

```
@CPUTimes = times();
$UserTimeForProcess    = $CPUTimes[0];
$SystemTimeForProcess  = $CPUTimes[1];
$UserTimeForChildren   = $CPUTimes[2];
$SystemTimeForChildren = $CPUTimes[3];
```

[71] http://en.wikibooks.org/wiki/..%2FTime%3A%3AHires%2F

4.63 Functions that reverse each other

Some functions in perl reverse or otherwise cancel the effect of each other, so running a string through both of them will produce the same output as the input, for example

```
print ord(chr(1));
```

will echo `1` to standard output,

`ord()`[72] will convert a character to its number in the character set, while `chr()`[73] will convert a number to its corresponding character, therefore

in the same way that $\sqrt{x^2} = x$ and $\sqrt{x}^2 = x$ in Mathematics[74] (assuming x is non-negative), `ord(chr(1)) = 1` and `chr(ord(1)) = 1` in Perl.

List of functions that reverse each other:

- `lc()`[75] and `uc()`[76]
- `lcfirst()`[77] and `ucfirst()`[78]
- `ord()`[79] and `chr()`[80]
- `join()`[81] and `split()`[82]
- `push()`[83] and `pop()`[84]
- `unshift()`[85] and `shift()`[86]

See `http://search.cpan.org/`

Also, try subscribing to the use.perl.org[87] mailing list, which sends out daily summaries of new modules as they're added to CPAN.

- /File Tests/[88]
- Perl functions[89]
- /Regular Expressions/[90]
- /Database/[91]

[72] Chapter 4.46.11 on page 97
[73] Chapter 4.46.3 on page 96
[74] `http://en.wikipedia.org/wiki/Mathematics`
[75] Chapter 4.46.7 on page 97
[76] Chapter 4.46.17 on page 101
[77] Chapter 4.46.8 on page 97
[78] Chapter 4.46.18 on page 101
[79] Chapter 4.46.11 on page 97
[80] Chapter 4.46.3 on page 96
[81] Chapter 4.49.2 on page 104
[82] Chapter 4.51.28 on page 108
[83] Chapter 4.48.2 on page 103
[84] Chapter 4.48.1 on page 103
[85] Chapter 4.48.5 on page 104
[86] Chapter 4.48.3 on page 103
[87] `http://use.perl.org`
[88] `http://en.wikibooks.org/wiki/%2FFile%20Tests%2F`
[89] `http://en.wikibooks.org/wiki/%2FFunctions`
[90] `http://en.wikibooks.org/wiki/%2FRegular%20Expressions%2F`
[91] `http://en.wikibooks.org/wiki/%2FDatabase%2F`

- /Time and Date/[92]

4.64 Key Sites

- perl.org[93] - the home of the Perl programming language
- perldoc.perl.org[94] - Perl documentation
- cpan.org [95] - The Comprehensive Perl Archive Network, a huge repository for Perl modules and scripts
- dev.perl.org/perl6[96] - Perl 6 development site
  The Parrot virtual machine[97]

4.65 Community

- Perl Mongers[98], Perl User Group Index
- The Perl Monastery[99], themed Perl based help site

4.66 Other

- perl.com[100] - a Perl blog
- The DMOZ directory[101], DMOZ Perl index
- Perl Outsourcing Stats[102]
- Perl Tutorial[103]
- Wikipedia:Perl[104]

[92] http://en.wikibooks.org/wiki/%2FTime%20and%20Date%2F
[93] http://www.perl.org
[94] http://perldoc.perl.org/
[95] http://www.cpan.org/
[96] http://dev.perl.org/perl6/
[97] http://www.parrotcode.org
[98] http://www.pm.org
[99] http://www.perlmonks.org
[100] http://www.perl.com
[101] http://dmoz.org/Computers/Programming/Languages/Perl/
[102] http://www.odesk.com/trends/perl
[103] http://www.programmingbulls.com/perl-tutorial
[104] http://en.wikipedia.org/wiki/Perl

5 Contributors

Edits	User
1	354d[1]
26	Adrignola[2]
2	Ahy1[3]
2	AlanUS[4]
4	Albmont[5]
1	Alecclews[6]
2	Alksentrs[7]
3	Alsocal[8]
6	Amire80[9]
5	Archier[10]
1	Arkuat[11]
5	Bevatron[12]
9	Bkuhn[13]
1	Briefaddn[14]
2	Bseidel[15]
6	CSJewell[16]
1	Captain panda[17]
1	CarsracBot[18]
1	Castaway[19]
7	CharlesClarkson[20]
2	ChippendaleMupp[21]

[1] http://en.wikibooks.org/w/index.php?title=User:354d
[2] http://en.wikibooks.org/w/index.php?title=User:Adrignola
[3] http://en.wikibooks.org/w/index.php?title=User:Ahy1
[4] http://en.wikibooks.org/w/index.php?title=User:AlanUS
[5] http://en.wikibooks.org/w/index.php?title=User:Albmont
[6] http://en.wikibooks.org/w/index.php?title=User:Alecclews
[7] http://en.wikibooks.org/w/index.php?title=User:Alksentrs
[8] http://en.wikibooks.org/w/index.php?title=User:Alsocal
[9] http://en.wikibooks.org/w/index.php?title=User:Amire80
[10] http://en.wikibooks.org/w/index.php?title=User:Archier
[11] http://en.wikibooks.org/w/index.php?title=User:Arkuat
[12] http://en.wikibooks.org/w/index.php?title=User:Bevatron
[13] http://en.wikibooks.org/w/index.php?title=User:Bkuhn
[14] http://en.wikibooks.org/w/index.php?title=User:Briefaddn
[15] http://en.wikibooks.org/w/index.php?title=User:Bseidel
[16] http://en.wikibooks.org/w/index.php?title=User:CSJewell
[17] http://en.wikibooks.org/w/index.php?title=User:Captain_panda
[18] http://en.wikibooks.org/w/index.php?title=User:CarsracBot
[19] http://en.wikibooks.org/w/index.php?title=User:Castaway
[20] http://en.wikibooks.org/w/index.php?title=User:CharlesClarkson
[21] http://en.wikibooks.org/w/index.php?title=User:ChippendaleMupp

18	Conrad.Irwin[22]
1	Cspurrier[23]
1	Dallas1278[24]
7	Dan Polansky[25]
78	Darklama[26]
1	DavidCary[27]
1	Derbeth[28]
3	Dinomite[29]
39	Dirk Hünniger[30]
1	Dissident[31]
3	Dysprosia[32]
60	EvanCarroll[33]
2	Fishpi[34]
1	Flavio Poletti[35]
2	Geocachernemesis[36]
1	Guanabot[37]
1	Isaacto[38]
2	Jdstreng[39]
14	JendaCPAN[40]
1	Jfmantis[41]
16	Jguk[42]
1	JoaquinFerrero[43]
52	Joelnackman[44]
1	Jomegat[45]
4	Jonathan Webley[46]

[22] http://en.wikibooks.org/w/index.php?title=User:Conrad.Irwin
[23] http://en.wikibooks.org/w/index.php?title=User:Cspurrier
[24] http://en.wikibooks.org/w/index.php?title=User:Dallas1278
[25] http://en.wikibooks.org/w/index.php?title=User:Dan_Polansky
[26] http://en.wikibooks.org/w/index.php?title=User:Darklama
[27] http://en.wikibooks.org/w/index.php?title=User:DavidCary
[28] http://en.wikibooks.org/w/index.php?title=User:Derbeth
[29] http://en.wikibooks.org/w/index.php?title=User:Dinomite
[30] http://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger
[31] http://en.wikibooks.org/w/index.php?title=User:Dissident
[32] http://en.wikibooks.org/w/index.php?title=User:Dysprosia
[33] http://en.wikibooks.org/w/index.php?title=User:EvanCarroll
[34] http://en.wikibooks.org/w/index.php?title=User:Fishpi
[35] http://en.wikibooks.org/w/index.php?title=User:Flavio_Poletti
[36] http://en.wikibooks.org/w/index.php?title=User:Geocachernemesis
[37] http://en.wikibooks.org/w/index.php?title=User:Guanabot
[38] http://en.wikibooks.org/w/index.php?title=User:Isaacto
[39] http://en.wikibooks.org/w/index.php?title=User:Jdstreng
[40] http://en.wikibooks.org/w/index.php?title=User:JendaCPAN
[41] http://en.wikibooks.org/w/index.php?title=User:Jfmantis
[42] http://en.wikibooks.org/w/index.php?title=User:Jguk
[43] http://en.wikibooks.org/w/index.php?title=User:JoaquinFerrero
[44] http://en.wikibooks.org/w/index.php?title=User:Joelnackman
[45] http://en.wikibooks.org/w/index.php?title=User:Jomegat
[46] http://en.wikibooks.org/w/index.php?title=User:Jonathan_Webley

42	Justforasecond[47]
3	Jędrzej Pełka[48]
1	Kamanashisroy[49]
2	Karchnu[50]
6	Karl Dickman[51]
1	Kayau[52]
2	Krischik[53]
2	Larsnyg[54]
4	Lichray[55]
1	Luke X[56]
3	Mairi[57]
2	Malli kv2[58]
4	Merime[59]
1	Mhoram[60]
4	Mickraus[61]
1	Mike Segal[62]
3	Mike.lifeguard[63]
1	Mikecx08[64]
1	Mjbmrbot[65]
28	MrItty[66]
1	Mrajcok[67]
1	Msiren[68]
1	Nex3[69]
2	Nikai[70]
1	Nkrypt[71]

[47] http://en.wikibooks.org/w/index.php?title=User:Justforasecond
[48] http://en.wikibooks.org/w/index.php?title=User:J%C4%99drzej_Pe%C5%82ka
[49] http://en.wikibooks.org/w/index.php?title=User:Kamanashisroy
[50] http://en.wikibooks.org/w/index.php?title=User:Karchnu
[51] http://en.wikibooks.org/w/index.php?title=User:Karl_Dickman
[52] http://en.wikibooks.org/w/index.php?title=User:Kayau
[53] http://en.wikibooks.org/w/index.php?title=User:Krischik
[54] http://en.wikibooks.org/w/index.php?title=User:Larsnyg
[55] http://en.wikibooks.org/w/index.php?title=User:Lichray
[56] http://en.wikibooks.org/w/index.php?title=User:Luke_X
[57] http://en.wikibooks.org/w/index.php?title=User:Mairi
[58] http://en.wikibooks.org/w/index.php?title=User:Malli_kv2
[59] http://en.wikibooks.org/w/index.php?title=User:Merime
[60] http://en.wikibooks.org/w/index.php?title=User:Mhoram
[61] http://en.wikibooks.org/w/index.php?title=User:Mickraus
[62] http://en.wikibooks.org/w/index.php?title=User:Mike_Segal
[63] http://en.wikibooks.org/w/index.php?title=User:Mike.lifeguard
[64] http://en.wikibooks.org/w/index.php?title=User:Mikecx08
[65] http://en.wikibooks.org/w/index.php?title=User:Mjbmrbot
[66] http://en.wikibooks.org/w/index.php?title=User:MrItty
[67] http://en.wikibooks.org/w/index.php?title=User:Mrajcok
[68] http://en.wikibooks.org/w/index.php?title=User:Msiren
[69] http://en.wikibooks.org/w/index.php?title=User:Nex3
[70] http://en.wikibooks.org/w/index.php?title=User:Nikai
[71] http://en.wikibooks.org/w/index.php?title=User:Nkrypt

61	Ojw[72]
1	Omegatron[73]
6	Pengrate[74]
4	Plaicy[75]
2	Pnorcks[76]
4	Poccil[77]
6	Prolix[78]
3	QuiteUnusual[79]
28	RandalSchwartz[80]
2	Recent Runes[81]
1	Rich Farmbrough[82]
3	Rovenhot[83]
1	Schwern[84]
2	Sgibson[85]
1	Shadow42[86]
2	Shiman bb[87]
1	Shoecream[88]
21	Sigma 7[89]
2	Snarius[90]
1	Spoon![91]
2	Stennie[92]
1	The bellman[93]
2	ThorstenStaerk[94]
1	Thumperward[95]
2	Toreau[96]

[72] http://en.wikibooks.org/w/index.php?title=User:Ojw
[73] http://en.wikibooks.org/w/index.php?title=User:Omegatron
[74] http://en.wikibooks.org/w/index.php?title=User:Pengrate
[75] http://en.wikibooks.org/w/index.php?title=User:Plaicy
[76] http://en.wikibooks.org/w/index.php?title=User:Pnorcks
[77] http://en.wikibooks.org/w/index.php?title=User:Poccil
[78] http://en.wikibooks.org/w/index.php?title=User:Prolix
[79] http://en.wikibooks.org/w/index.php?title=User:QuiteUnusual
[80] http://en.wikibooks.org/w/index.php?title=User:RandalSchwartz
[81] http://en.wikibooks.org/w/index.php?title=User:Recent_Runes
[82] http://en.wikibooks.org/w/index.php?title=User:Rich_Farmbrough
[83] http://en.wikibooks.org/w/index.php?title=User:Rovenhot
[84] http://en.wikibooks.org/w/index.php?title=User:Schwern
[85] http://en.wikibooks.org/w/index.php?title=User:Sgibson
[86] http://en.wikibooks.org/w/index.php?title=User:Shadow42
[87] http://en.wikibooks.org/w/index.php?title=User:Shiman_bb
[88] http://en.wikibooks.org/w/index.php?title=User:Shoecream
[89] http://en.wikibooks.org/w/index.php?title=User:Sigma_7
[90] http://en.wikibooks.org/w/index.php?title=User:Snarius
[91] http://en.wikibooks.org/w/index.php?title=User:Spoon%21
[92] http://en.wikibooks.org/w/index.php?title=User:Stennie
[93] http://en.wikibooks.org/w/index.php?title=User:The_bellman
[94] http://en.wikibooks.org/w/index.php?title=User:ThorstenStaerk
[95] http://en.wikibooks.org/w/index.php?title=User:Thumperward
[96] http://en.wikibooks.org/w/index.php?title=User:Toreau

1	Tpraveen[97]
25	UltraAyla[98]
1	Verithrax[99]
3	Voxhumana[100]
3	WausauBill[101]
2	Webaware[102]
2	Wereon[103]
1	Wikimi-dhiann[104]
31	Yath[105]
1	YordanGeorgiev[106]
2	Yurik[107]
6	Zoohouse[108]
1	Zuco[109]
6	Ævar Arnfjörð Bjarmason[110]

[97] `http://en.wikibooks.org/w/index.php?title=User:Tpraveen`

[98] `http://en.wikibooks.org/w/index.php?title=User:UltraAyla`

[99] `http://en.wikibooks.org/w/index.php?title=User:Verithrax`

[100] `http://en.wikibooks.org/w/index.php?title=User:Voxhumana`

[101] `http://en.wikibooks.org/w/index.php?title=User:WausauBill`

[102] `http://en.wikibooks.org/w/index.php?title=User:Webaware`

[103] `http://en.wikibooks.org/w/index.php?title=User:Wereon`

[104] `http://en.wikibooks.org/w/index.php?title=User:Wikimi-dhiann`

[105] `http://en.wikibooks.org/w/index.php?title=User:Yath`

[106] `http://en.wikibooks.org/w/index.php?title=User:YordanGeorgiev`

[107] `http://en.wikibooks.org/w/index.php?title=User:Yurik`

[108] `http://en.wikibooks.org/w/index.php?title=User:Zoohouse`

[109] `http://en.wikibooks.org/w/index.php?title=User:Zuco`

[110] `http://en.wikibooks.org/w/index.php?title=User:%C3%86var_Arnfj%C3%B6r%C3%B0_Bjarmason`

List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[111]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

[111] Chapter 6 on page 135

6 Licenses

6.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

6.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

6.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.