The Perl Programming Language

“”

Contents

.

The book is set out as a series of “recipes” in the style of a “cookbook” The perl language and its many modules is a large topic and this document has only been just begun.

Section 1

Perl

[+] Perl is a language which was originally inspired by the Bash shell syntax, as well as by the idea of writing terse but powerful programs. The name perl is not an acronym, since the creator, Larry Wall said he was looking for any name with “positive connotations”. Perl initially rose to fame through it suitability for writing web-server cgi scripts, since perl, like the unix shells, uses plain text as a kind of “data interchange format”. [+] The weaknesses of perl: it moves away from the Unix idea of using small programs to do one thing and linking them together with FIFO pipes or streams; it has no built in windowing commands; Its ease of use may encourage bad programming or attract bad programmers.

This section will concentrate on one line perl programs and integrating those programs with the bash shell.

Section 2

Learning Perl

```
http://www.perlmonks.org/
http://learn.perl.org/
```

Display the introduction to the perl documentation

```
man perl
```

```
perldoc perl     ~(the same)
```

Set variables f=“bi” and g=“sm a” using eval and a perl one liner.

```
eval $(perl -e 'print "f=bi;";print "g=\"sm a\"\n"')
```

(this demonstrates “exporting” variables from perl to the parent shell)

Check syntax of a perl one line program but dont run it

```
perl -wc -e 'print "\n\n"'
```

Gotchas

In a one-line script, print must use "\n" otherwise no output may appear

```
perl -e 'if ( -d ".") { print "folder"; }' ~(doesnt seem to print '
  ⇒ folder')
```

```
perl -e 'if ( -d ".") { print "folder\n"; }' ~(correct: prints 'folder
  ⇒ ')
```

Most one line perl scripts should be enclosed in single quotes

```
perl -e 'if (!$s) { print "s has no value\n" }'
```

(if double quotes were used, the shell would interpret "!$" first)

Lines read from standard input have a newline, so you must use '-l' or 'chomp'

```
ls | perl -ne 'if (-T "$_") {print "$_ text"}' ~(doesnt work since $_
  ⇒ has \n)
```

```
ls | perl -lne 'if (-T "$_") {print "$_ text"}' | less          ~(this
  ⇒ works)
```

```
ls | perl -ne 'chomp; if (-T "$_") {print "$_ text\n"}' | less ~(this
  ⇒ works)
```


Perl Documentation

Documentation for the perl language is installed along with the language

View the start page for the perl documentation

```
man perl
```

```
perldoc perl    ~(the same, more useful on ms windows)
```

some perl documentation pages

`man perl`	The start page, contains references to lots and lots of docs
`man perlintro`	An introduction
`man perltoc`	A table of contents of perl documentation

Query the perl "frequently asked questions" documents for a word or phrase

```
perldoc -q eval
```

Save the entire "perlfunc" man page in the file "file.txt"

```
perldoc -T perlfunc > file.txt
```

Query the perl faqs for the word 'file'

```
perldoc -q ' file'    ~(' file' seems to work better than 'file')
```

Show the documentation for the CGI module

```
perldoc CGI    ~(these names are case-sensitive, "perldoc cgi" doesnt
  ⇒ work)
```

```
man CGI
```

4.1 Pod The Perl Documentation Format

debian: perl-doc - to use the perldoc tool

The perl documentation format is known as the "pod" format and is accompanied by a variety of tools to transform it to other documentation formats. The "perldoc" tool can be used to query the perl documentation or the documentation for a module

Perl Modules

Modules are libraries of code which carry out specific task and save the programmer large amounts of time. One of the strenghts of perl is the very large number of open-source modules available.

perl modules documentation

`man perlmod`	How modules work
`man perlmodlib`	How to write and use a perl module
`man perlmodinstall`	How to install from CPAN

Show all programs which have perl in the short description

```
apt-cache search 'perl' | grep perl | sort | less
```

Use a module

```
perl -Mmodule -e "print 'hello'"
```

Use the CGI module with the "qw(:standard)" option

```
perl -MCGI=:standard -e "print header, h1('hello')"
```

The cpan program can be used to find and install perl modules

```
cpan
```

Some perl modules may be installed via debian packages or via cpan

```
sudo apt-get install libgd-barcode-perl
```

Check if the LWP module is installed

```
perl -MLWP -e1
```

Print the version number of the LWP module

```
perl -MLWP -e 'print $LWP::VERSION'
```

5.1 Using The Lwp Module

`http://www.darkspell.com/references/lwpcook.html`
a set of "recipes" for using the lwp module

Download a webpage for processing

```
perl -e 'use LWP::Simple; $doc = get "http://bumble.sf.net";'
```

Download and display a url using the lwp perl module

```
PERL -MLWP::Simple -e 'getprint "http://url"'
```

Check if a document exists

```
use LWP::Simple; if (head($url)) {# ok document exists}
```

5.2 Using The Cgi Module

Send error messages generated by perl to the browser

```
use CGI;
```

```
use CGI::Carp qw(fatalsToBrowser);
```

Print a very simple document with the cgi module

```
use CGI;
my $cgi = new CGI;
print $cgi->header(); print $cgi->start_html();
print "hello cgi"; print $cgi->end_html;
```

Indicate the title of the document using the cgi module

```
print $cgi->start_html( -title=> "testing cgi")
```

Show the file parameter which was sent from an html form

```
use CGI;
my $cgi = new CGI;
print $cgi->header(); print $cgi->start_html();
print "the file parameter is:", $cgi->param('file');
print $cgi->end_html;
```

Access the cgi "environment" variables from perl

```
$sDocumentRoot = $ENV{'DOCUMENT_ROOT'};
```

Section 6

Some Useful Modules

some useful modules

`HTML::LinkExtor`	Extract links from html
`File::Find`	Find files
`Getopt::Long`	Get long and short options for a script
`Cwd`	Print the current working folder
`URI::URL`	Extract portions of a url
`File::Basename`	Get the folder and filename
`File::Path`	Make folders and delete them (mkpath, rmtree)
`Benchmark`	Time how long perl code takes to run
`DataDumper`	Creates a string representation of arrays and hashes

Section 7

Windowing Programs

```
use tk; # #
```

Section 8

Cpan The Online Perl Code Repository

Cpan stands for "comprehensive perl archive network" and is a repository of open-source code modules and libraries which can be used to ease the task of the programmer. "cpan" is also an interactive program which allows one to find and download these modules from a command line. The name "cpan" was modelled on "ctan" which is the the "comprehensive tex archive network"

`http://www.cpan.org/`
: the cpan home site

`http://sial.org/howto/perl/life-with-cpan/`
: some information about cpan

Run the "cpan" interactive program

```
cpan
```

```
sudo cpan    ~(as a quick fix for permissions problems)
```

Show the documentation for the "cpan" module

```
perldoc CPAN
```

```
http://search.cpan.org/perldoc/CPAN   ~(the same, in a web-browser)
```

"cpanplus" is a more modern alternative to cpan
Install the latest version of cpan, with passive ftp for firewalls

```
perl -MCPAN -e '$ENV{FTP_PASSIVE} = 1; install CPAN'
```

```
install CPAN ~(the same, but from within the "cpan" program)
```

Search the cpan site for the documentation for the "LWP::UserAgent" package

```
http://search.cpan.org/perldoc/LWP::UserAgent
```

8.1 Using The Cpan Program

problems with the "cpan" program:

- ⋆ it doesnt tell you how big a module which you are going to install is.
- ⋆

Run the start up configuration for the cpan program

```
cpan
o conf init
```

Install "history" support for cpan (the up arrow obtain the previous command)

```
cpan
install Term::ReadKey Term::ReadLine  ##(didnt work)
```

Show the short help for the cpan program

```
h | less
```

Show details about the module whose name is CGI

```
m CGI      ~(the exact module name must be written, case sensitive)
```

Show all modules which have the text "CGI" in their names

```
m /CGI/ | less    ~(this is a case insensitive search)
```

(problem, hitting 'q' in less exits cpan)

Show information about the CGI module (using CPAN non-interactively)

```
perl -MCPAN -e' CPAN::Shell->m("CGI")' | less
```

Show a short description for all modules which have "CGI" in the name

```
perl -MCPAN -e' CPAN::Shell->m("/cgi/")' | less   ~(this is quite slow)
```

(these searches are Not case sensitive)

Show all available modules on cpan (approximately 70000)

```
perl -MCPAN -e' CPAN::Shell->m()' | less   ~(this will be VERY slow)
```

(this command took 3 minutes on my ASUS netbook computer)

Section 9

Perl One Line Scripts

perl command line switches

-p	Loops over each input line and prints it
-n	Loops over each input line but doesnt print it
-l	Remove newline characters when read and restore when writing
-e	Specify a perl expression to use, should be the last switch used

Print files in the current folder which are text files All the following versions do the same thing

```
ls | perl -lne '-T and print'   ~(a possible problem with spacey
  ⇒ filenames)
```

```
ls | perl -lne '-T && print'
```

```
ls | perl -lne 'print if -T'
```

```
ls | perl -lne '-T "$_" and print'
```

```
ls | perl -lne 'if (-T "$_") {print "$_"}'
```

```
ls | perl -lne '-T "$_" and print "$_"'
```

```
ls | perl -lne '-T $_ and print $_'
```

```
ls | perl -lne '(-T "$_") && (print "$_")'
```

```
ls | perl -ne 'chomp; if (-T "$_") {print "$_\n"}'
```

The large list of commands above, all of which do the same thing shows the flexibility of the perl syntax. Perl allows certain things to be implied (just like in real language). The most common thing which is implied is "$_" which can be translated as "that" and in a loop is generally the current line or variable.

Include 2 perl expressions with the -e expression

```
perl -e 'print "Hello";' -e 'print " World\n"'
```

Print the 2nd field of the input (fields delimited by spaces)

```
echo a b c | perl -lane 'print $F[1]' ~(the -n switch loops without
  ⇒ printing)
```

Print the 1st and 2nd fields of the input lines

```
echo a b c | perl -lane 'print "@F[0..1]"'
```

Print the first field of a password file (splitting on the ':' f character

```
perl -F: -lane 'print $F[0] if !/^#/' /etc/passwd
```

Print lines which dont contain the letter 'b'

```
(echo a; echo b) | perl -nle 'print if !/b/'
```

Print the 3rd line of a file

```
perl -nle 'print if $. == 1' file.txt
```

Print everyline except the first

```
perl -nle 'print unless $. == 1' file.txt
```

Section 10

Printing To Standard Output

Print the results of 2 functions to standard output

```
print header(), footer();
```

```
print header, footer;      ~(the same)
```

Print a string and a function result to standard output

```
print "Your name is", name();
```

Print text in single quotes

```
perl -e 'print q{#!/usr/bin/perl}' ~(the quotes dont appear?)
```

Section 11

General Perl Syntax

Section 12

Loops

12.1 The Foreach Loop

Loop through the elements of an array

```
@names = ('Larry', 'John', 'Jack');
foreach (@names) { print $_."\n"; }
```

Loop through the elements of a literal list

```
foreach (qw/one 2 three 4/) { print $_."\n"; }
```

12.2 If Statement

The if statement has a c-like syntax

```
if (test) {... }
```

12.3 Global Variables

Import the variables $TRUE, $FALSE etc

```
do 'global.pl'; use vars qw($TRUE $FALSE $LANGUAGE);
```

12.4 Different Types Of Quotes

quote characters

qq	Be used anywhere " can be used
qw	Quote a list of words, eg; qw/one 2 three/

12.5 Local Variables

Create a local variable

```
my $string;
```

12.6 Array Variables

Assign a list of text files in the current folder to the array @list

```
@list = grep { -f && -T } glob(   *   )
```

Show all text files in the current folder

```
perl -e 'print join "\n", grep {-T}<*>'
```

12.7 String Variables

Exit if the "s" variable has no value

```
perl -e 'if (!$s) { die "s has no value"; }'
```

Join to strings together

```
$s = "green"."tree";    ~($s is now 'greentree')
```

Append a 'here document' to a string

```
    $s .= <<ENDS;
    A multiline
    string variable
ENDS
```

Show the number of occurences of the 's' character in the variable "$text"

```
$i = ($text =~ tr/s//)
```

12.8 Here Documents

[+] A 'here document' is a way to print a large amount of text, or two assign that text to a variable without having to use lots of quote characters, or escape special characters. The syntax of the 'here document' was based on (but is slightly different to) the syntax of the Bash shell equivalent.

Assign a here document to a string

```
    $s = <<ENDS;
    A multiline
    string variable
ENDS
```

Section 13

Matching Text With Regular Expressions

perl documentation for regular expressions

man perlrequick	A quick introduction
man perlretut	More indepth look
man perlreref	A quick reference for perl regular expressions
man perlre	A complete reference

Print file names in the current folder which have 'tree' in the name

```
ls | perl -lne 'print if /tree/'
```

```
ls | perl -lne 'print if $_ =~ /tree/'          ~(the same but
  ⇒ unnecessary)
```

```
ls | perl -lne 'print($_) if ($_ =~ /tree/)'    ~(the same again)
```

Check if something doesnt match

```
"Hello World" !~ /World/
```

Section 14

String Substitutions

This section deals with replacing a string or a pattern with another string. This is one of perls particular strengths, giving rise to its reputation for being a "text oriented" programming language

Delete all occurences of the new-line character in a string

```
$text =~ tr/\n//;
```

Section 15

Perl Special Variables

`$_` this contains the current line when looping through the standard input, or else the current element from any list. use "chomp" to remove the newline from this when necessary @@

Section 16

Using Files

Test if a file exists and exit if it does not

```
if (!-e 'index.txt') { die "the file doesnt exist"; }
```

Test if a file is actually a folder

```
if (-d "work") { print "'work' is a folder\n" }
```

Test if a file is a plain text file

```
if (-T "index.txt") { print "index.txt is a text file\n" }
```

Test if a file can be executed

```
if (-x "index.sh") { print "index.txt is executable\n" }
```

```
-x "index.sh" and print "index.txt is executable\n" ~(the same)
```

16.1 Opening Files For Reading Or Writing

Attempt to open a file for reading, and, if not, show the error message

```
open(FILE, 'index.txt') or die "Can t open file: $!";   ~("$!" has the
  ⇒   error)
```

16.2 Write To A File

Write a string to a file (any previous file contents are destroyed)

```
$s="green tree";
open(F, ">index.txt") || die "Could not open the file for writing!";
print F "$s";
close F;
```

16.3 Copying Files

Copy a file to another name or exit if it is not possible

```
use File::Copy; my $f = "list";
copy($f, "$f.1") || die "could not copy file $f, because: $!";
```

16.4 Creating A Temporary File

Create a temporary file, without ever knowing its name

```
use IO::File;
$fh = IO::File->new_tmpfile() or die "Couldnt make the temp file: $
  ⇒ !";
```

Another way

```
use File::Temp
```

16.5 File Globbing

'file globbing' refers to expanding a wildcard character (such as '*' or '?') into a list of valid file names for the local computer.

Display files with a '.txt' extension in the current folder

```
perl -e 'for (glob("*.txt")) { print $_."\n"}'
```

```
perl -e 'foreach (glob("*.txt")) { print $_."\n"}'      ~(the same)
```

```
perl -e 'foreach $f (glob("*.txt")) { print $f."\n"}'   ~(the same again
  ⇒ )
```

Section 17 — *Using Folders Or Directories*

Print the directory part of a file name

```
use File::Basename;
print dirname("/home/username/index.txt");
```

Section 18 — *Perl Standard Functions*

*

```
perldoc perlfunc
```

18.1 Writing Perl Functions

Append the result of a function to a scalar variable

```
$sOutput .= listData($configFile, $siteRoot, '');
```

18.2 Importing Other Files

Import the file "move.pl" which contains a function and is in the "lib" folder

```
require 'lib/move.pl';
```

Section 19 — *Text Files*

Change aaa for bbb and print each line

```
perl -p -e 's/aaa/bbb/' test.txt     ~(the file is not changed)
```

Change aaa for bbb and print each line

```
perl -pi -e 's/aaa/bbb/' test.txt     ~(the file IS changed)
```

Replace the word big with small in .txt files and backup to .bak

```
perl -p -i.bak -e 's/\bbig\b/small/g' *.txt
```

Recursive replacement of text in this and subdirectories

```
perl -p -i.bak -e 's/\bbig\b/small/g' $(find ./ -name "*.txt")
```

```
perl -p -i.bak -e 's/\bbig\b/small/g' $(grep -ril text *)
```

Insert one line in a text file

```
use Tie::File
```