Introduction

A Potted History

Perl was originally written by Larry Wall while he was working at NASA's Jet Propulsion Labs. Larry is an Internet legend: Not only is he well-known for Perl, but as the author of the UNIX utilities `rn`, which was one of the original Usenet newsreaders, and `patch`, a tremendously useful utility that takes a list of differences between two files and allows you to turn one into the other. The word 'patch' used for this activity is now widespread.

Perl started life as a 'glue' language, for the use of Larry and his officemates, allowing one to 'stick' different tools together by converting between their various data formats. It pulled together the best features of several languages: the powerful regular expressions from `sed` (the UNIX stream editor), the pattern-scanning language `awk`, and a few other languages and utilities. The syntax was further made up out of C, Pascal, Basic, UNIX shell languages, English and maybe a few other things along the way.

Version 1 of Perl hit the world on December 18, 1987, and the language has been steadily developing since then, with contributions from innumerable people. Perl 2 expanded the regular expression support, while Perl 3 allowed Perl to deal with binary data. Perl 4 was released so that the Camel Book (see the Resources section at the end of this chapter) could refer to a new version of Perl.

Perl 5 has seen some rather drastic changes in syntax and some pretty fantastic extensions to the language. Perl 5 is (more or less) backwardly compatible with previous versions of the language, but at the same time, makes a lot of the old code obsolete. Perl 4 code may still run, but Perl 4 style is definitely frowned upon these days.

At the time of writing, the current stable release of Perl is 5.6, which is what this book will detail. That said, the maintainers of Perl are very careful to ensure that old code will run, perhaps all the way back to Perl 1 – changes and features that break existing programs are evaluated extremely seriously. Everything you see here will continue to function in the future.

I say 'maintainers' because Larry no longer looks after Perl by himself – there is a group of 'porters' who maintain the language and produce new releases. The 'perl5-porters' mailing list is the main development list for the language, and you can see the discussions archived at http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/. For each release, one of the porters will carry the 'patch pumpkin' – the responsibility for putting together and releasing the next version of Perl.

Where is Perl going in the future? Well, we expect Perl to develop steadily up the 5.x release series, adding more useful features and steadily deprecating more and more of the accumulated old-fashionedness, making it harder for people to justify the myth that Perl 4 is still alive and well.

There is at least one existing project to rewrite Perl from scratch: Chip Salzenberg is heading up a team called the Topaz project, which aims to produce a faster, more efficient Perl. Topaz is being written in C++, rather than C, but hopes to remain compatible with Perl 5. At the moment, the Topaz team isn't planning to add any new features to the language, but I'm sure that as the project gains momentum, more features will be added. You might sometimes hear Topaz referred to as Perl 6, but it'll only really become Perl 6 if Larry likes it – the way things are going, Topaz won't be in common use for quite some time yet, and I expect that Perl 6 will be the natural development of the current Perl.

Why Perl?

Just like the Basic programming language, the name 'Perl' isn't really an acronym. People like making up acronyms though, and Larry has two favorite expansions. According to its creator, perl is the **P**ractical **E**xtraction and **R**eport **L**anguage, or the **P**athologically **E**clectic **R**ubbish **L**ister. Either way, it doesn't really matter. Perl is a language for doing what you want to do.

The Perl motto is 'There's More Than One Way To Do It', emphasizing both the flexibility of Perl and the fact that Perl is about getting the job done. We can say that one Perl program is faster, or more idiomatic, or more efficient than another, but if both do the same thing, Perl isn't going to judge which one is 'better'. It also means that you don't need to know every last little detail about the language in order to do what you want with it. You'll probably be able to achieve a lot of the tasks you might want to use Perl for after the first four or five chapters of this book.

Perl has some very obvious strengths

- It's very easy to learn, and learning a little Perl can get you a long way.
- Perl was designed to be easy for humans to write, rather than easy for computers to understand. The syntax of the language is a lot more like a human language than the strict, rigid grammars and structures of other languages, so it doesn't impose a particular way of thinking upon you.
- Perl is very portable; That means what it sounds like – you can pick up a Perl program and carry it around between computers. Perl is available for a huge variety of operating systems and computers, and properly written programs should run almost anywhere that Perl does without any change.
- Perl talks text. It thinks about words and sentences, where other languages see the character at a time. It also thinks about files in terms of lines, not individual bytes. Its 'regular expressions' allow you to search for and transform text in innumerable ways with ease and speed.
- Perl is what is termed a 'high-level language'. Some languages like C concern you with unnecessary, 'low-level' details about the computer's operation: making sure you have enough free memory, making sure all parts of your program are set up properly before you try to use them, and leaving you with strange and unfriendly errors if you don't do so. Perl cuts you free from all this.

However, since Perl is so easy to learn and to use, especially for quick little administrative tasks, 'real' Perl users tend to write programs for small, specific jobs. In these cases, the code is meant to have a short lifespan and is for the programmer's eyes only. The problem is, these programs may live a little longer than the programmer expects and be seen by other eyes too. The result is a cryptic one-liner that is incomprehensible to everyone but the original programmer. Because of the proliferation of these rather concise and confusing programs, Perl has developed a reputation for being arcane and unintelligible – one that I hope we can dispel during the course of this book.

This reputation is unfair. It's possible to write code that is tortuous and difficult to follow in any programming language, and Perl was never meant to be difficult. In fact, Perl is one of the easiest languages to learn, especially given its scope and flexibility.

Throughout this book you will learn how to avoid the stereotypical 'spaghetti code' and how to write programs that are both easy to write and easy to follow. Let's work to kill off this negative image.

It's Free

Larry started (and indeed, continued) Perl with the strong belief that software should be free – freely available, freely modifiable, and freely distributable. Perl is developed and maintained by the porters, who are volunteers from the Perl user community, all of whom strive to make Perl as good as possible.

This has a few nice side effects – the porters are working for love, rather than merely because it's their job, so they're motivated solely by their desire to see a better Perl. It also means Perl will continue to be free to use and distribute.

This doesn't mean that Perl is part of the GNU suite of utilities. The GNU ("**G**NU's **N**ot **U**NIX") project was set up to produce a freely usable, distributable, and modifiable version of the UNIX operating system and its tools. It now produces a lot of helpful, free utilities. Perl is included in distributions of GNU software, but Perl itself is not a product of the Free Software Foundation, the body that oversees GNU.

While Perl can be distributed under the terms of the **G**NU **P**ublic **L**icense (which you can find at http://www.gnu.org/), it can also be distributed under the Artistic License (found either with the perl sources or at http://www.opensource.org/licenses/), which purports to give more freedom to users and more security to developers than the GPL. You may judge for yourself – we've included these licenses in Appendix G.

Of course, those wanting to use Perl at work might be a little put off by this – managers like to pay money for things and have pieces of paper saying that they can get irate at someone if it all stops working. There's a question in the Perl FAQ (Frequently Asked Questions) about how to get a commercial version or support for Perl, and we'll see how you can find out the answer for yourself pretty soon.

What Is Perl Used For?

Far and away the most popular use of Perl is for CGI programming – that is, dynamically generating web pages. A whole chapter is devoted to introducing CGI programming in Perl. Perl is the power behind some of the most popular sites on the web: Slashdot (http://www.slashdot.org/), Amazon (http://www.amazon.com/), and Deja (http://www.deja.com/), and many others besides are almost entirely Perl-driven. We'll also look at some of the more recent extensions to the Perl/CGI concept: PerlScript, `mod_perl` and `HTML::Mason`, which are becoming widely used.

Of course Perl is still widely used for its original purpose: extracting data from one source and translating it to another format. This covers everything from processing and summarizing system logs, through manipulating databases, reformatting text files, and simple search-and-replace operations, to something like `alien`, a program to port Linux software packages between different distributors' packaging formats. Perl even manages the data from the Human Genome Project, a task requiring massive amounts of data manipulation.

For system administrators, Perl is certainly the 'Swiss Army chainsaw' that it claims to be. It's great for automating administration tasks, sending automatically generated mails and generally tidying up the system. It can process logs, report information on disk usage, produce reports on resource use and watch for security problems. There are also extensions that allow Perl to deal with the Windows registry and run as a Windows NT service, not to mention functions built into that allow it to manipulate UNIX `passwd` and `group` file entries.

However, as you might expect, that's not all. Perl is becoming the de facto programming language of the Internet its networking capabilities have allowed it to be used to create clients, servers, and proxies for things such as IRC, WWW, FTP, and practically every other protocol you wish to think of. It's used to filter mail, automatically post news articles, mirror web sites, automate downloading and uploading, and so on. In fact, it's hard to find an area of the Internet in which Perl isn't used.

Windows, UNIX, and Other Operating Systems

Perl is one of the most portable, if not *the* most portable programming languages around. It can be compiled on over 70 operating systems, and you can get binary distributions for most common platforms. Over the course of the book, we'll be looking at programs that can run equally well on almost any operating system.

When we're setting up Perl and running our examples, we'll concentrate particularly on UNIX and Windows. By UNIX, I mean any commercial or free UNIX-like implementation – Solaris, Linux, Net-, Free- and OpenBSD, HP/UX, A/IX, and so on. Perl's home platform is UNIX, and 90% of the world uses Windows. That said, the Perl language is the same for everyone. If you need help with your particular platform, you will probably be able to find a README file for it in the Perl source distribution. We'll see how to get hold of that in the next chapter.

While we're talking about operating system specifics, we'll use the filename extension `.plx` for our examples. Traditionally, UNIX programs take no extension, and Windows files take a three-letter extension to indicate their type. `.plx` is used by ActiveState to indicate a Perl program. Since UNIX isn't fussy, we'll use that idiom. You may also see the extension `.pl` in use for Perl programs (and, in fact, I use it myself from time to time to remind me that a given program is in fact a Perl one), but to be really pedantic, that's more properly used for Perl 4 libraries. These have, for the most part, been replaced by Perl 5 modules, which generally have the extension `.pm`. To avoid confusion, we won't use the `.pl` extension.

You can also get more information on portable Perl programming from the `perlport` documentation. Again, we'll see how to access this documentation very soon.

The Prompt

If you're primarily using your computer in a graphical environment like Windows or X, you may not be familiar with using the command line interface, or 'shell'. Before these graphical environments came into common use, users had to start a new program, not by finding its icon and clicking on it but by typing its name. The 'shell' is the program that takes the name from you. The 'shell prompt' (or just 'prompt') refers specifically to the text that prompts you to enter a new program name, and more generally, to working with the shell instead of using a graphical interface. Some people still find working with the shell much easier, and sophisticated shells have developed to simplify common tasks. In fact, on UNIX, the shell is programmable, and Perl takes some of its inspiration from standard 'Bourne Shell' programming practices.

To get to a prompt in Windows, look for Command Prompt or DOS Prompt in the Start Menu. UNIX users should look for a program called something like `console`, `terminal`, `konsole`, `xterm`, `eterm` or `kterm`. You'll then be faced with a usually black screen with a small amount of text that may say:

```
$
%
C:\>
#
bash$
```

For the purposes of this book, however, we'll use a prompt that looks like this:

```
>
```

We'll show text that you type in is bold. The text the computer generates is in a lighter typeface, like this:

```
> perl helloworld.plx
Hello World!
```

The command line may look scary at first, but you'll quickly get used to it as we go through the following examples and exercises. Note that ActiveState Perl will allow you to click on Perl programs and run them directly from the GUI if they have a `.pl` or `.plx` extension. (Later in the introduction, we'll show how you can manually configure Windows to do this.) However, the window containing the output will disappear as soon as the program has finished (try it!), and you won't be able to see what's happened, so I encourage you to use the shell instead.

What Do I Need To Use This Book?

As we've said, Perl is available for almost any kind of computer that has a keyboard and a screen, but we will be concentrating on perl for Windows and UNIX. Perl 5.6 will run on Windows 95 and 98 as well as NT and 2000. It'll run on more or less any UNIX, although you may find compilation is difficult if you don't have the latest C libraries. Any 2.x Linux kernel should be fine, likewise Solaris 2.6 or higher.

As well as Perl itself, you'll need a text editor to write and edit Perl source files. We look at a couple of options in Chapter 1.

To get the most out of some chapters, you'll also need to have an Internet connection.

For the chapter on CGI, you'll need a web server that supports CGI scripting. Apache is a good bet on UNIX machines (and it's included in most Linux distributions). Windows users could also use Apache, or alternatively, Microsoft's Personal Web Server (for 95 and 98). Internet Information Server (for NT and 2000) can be configured to run Perl CGIs. To use mod_perl, you'll have to use Apache, which you can obtain from http://www.apache.org.

How Do I Get Perl?

Perl has been ported to many, many platforms. It will almost certainly build and run on anything that looks like (or pretends to be) UNIX, such as Linux, Solaris, A/IX, HP/UX, FreeBSD, or even the Cygwin32 UNIX environment for Windows. Most other current operating systems are supported: Windows 95, 98, NT, and 2000, OS/2, VMS, DOS, BeOS, the Apple MacOS, and AmigaOS to name but a few.

- ❑ You can get the source to the latest stable release of Perl from http://www.perl.com/CPAN-local/src/stable.tar.gz.
- ❑ Binary distributions for some ports will appear in http://www.perl.com/CPAN-local/ports. These ports may differ in implementation from the original sources.
- ❑ You can get binary packages of Perl for Linux, Solaris, and Windows from ActiveState at http://www.activestate.com/ActivePerl/download.htm.
- ❑ Linux users should be able to get binary packages from the contrib section of their distributor's FTP site.

Installing on Linux/UNIX

As I said, Perl is freely available. If you're running a Linux system, then you probably got Perl packaged with your distribution. Type `perl -v` from a shell prompt to check this. If you see something that starts with the text `This is perl`, then congratulations – you already have Perl. It should, however, go on to give you a version number. If that's less than `v5.6.0` then you'll need to upgrade to a newer version to run the code as we've written it in this book. A few minor tweaks will get it running in earlier versions of Perl, but there's nothing like starting with the most up-to-date version of a toy, is there?

If you are running a package-based Linux system, such as Red Hat, SuSE, or Debian, then you have the choice of installing Perl using your system package manager, which makes upgrading and uninstalling simple. However, at the time of writing, this was complicated by the lack of availability of Perl 5.6 binary packages. ActiveState (http://www.activestate.com) makes packages in both RPM and Debian format, and if you don't already have Perl installed, these are fine. However, you may find it difficult to upgrade an existing Perl installation to ActivePerl using the package manager. In this case, installation from source may be your only option. The major distributors should, however, be making Perl 5.6 packages available from their FTP sites soon, which will allow you to upgrade.

Installing/Upgrading an RPM Installation

If you are installing the ActivePerl RPM from ActiveState, you need to type:

```
> rpm --prefix=/usr/local -Uvh ActivePerl-5.6.0.613.rpm
ActivePerl          ############################
```

The # marks appear to show the installation's progress. Using the `--prefix` option shown tells RPM to install the perl binaries in `/usr/local/bin`, libraries in `/usr/local/lib`, and so on, rather than their default locations under `/usr/local/perl-5.6`. If you already have a Perl package installed with your distribution, RPM won't let you overwrite the files with ActiveState's versions, though.

Once you've installed ActivePerl in this way, you may find it useful to add a soft link, or shortcut, from `/usr/bin/perl` to the `/usr/local/bin/perl` executable, since some scripts assume the perl interpreter is located there. To do this, you need to type:

```
> ln -sf /usr/local/bin/perl /usr/bin/perl
```

If you have obtained an RPM from your distributor, then you should be able to upgrade your existing perl installation using:

```
> rpm -Uvh perl-5.6.0.613.rpm
perl                ########################
```

Building Perl from Source

If none of these apply, you may have to build Perl from source. To do this, you need to obtain the `stable.tar.gz` file from any CPAN mirror. One such location is http://www.perl.com/CPAN-local/src/stable.tar.gz.

The build process on most UNIX systems, and especially for relatively current versions of Linux, is simple. Extract the archive and untar it:

```
> gunzip stable.tar.gz
> tar -xvf stable.tar
> cd perl-5.6.0
```

Now we need to run the `Configure` program. By supplying the `-d` switch, we tell `Configure` to probe our system and work out default settings for us. The `e` tells the `Configure` program not to bother us with any questions:

```
> ./Configure -de

Sources for perl5 found in "/root/perl-5.6.0".

Beginning of configuration questions for perl5.

...
```

There will now be a considerable amount of text scrolling up the screen, which shouldn't stop until the following appears:

```
...

Now you must run a make.

If you compile perl5 on a different machine or from a different object
directory, copy the Policy.sh file from this object directory to the
new one before you run Configure -- this will help you with most of
the policy defaults.
```

So, we do what the program says, and we run `make`.

```
> make
      AutoSplitting perl library
./miniperl -Ilib -e 'use AutoSplit; \

...
```

The build process itself will take the longest of all the steps. Once it's finished, it is worth running the built-in diagnostics with `make test`, as follows:

```
> make test
```

Finally, running `make install` puts all the files in the correct places.

```
> make install
      AutoSplitting perl library
./miniperl -Ilib -e 'use AutoSplit; \

...
```

If you need or want finer control about how Perl should be compiled, then run `./Configure` with no switches instead. The installer will ask you a few questions. If you don't know the answer at any stage, you can just hit Return, and let the system guess.

After the interrogation, you should now run `make`, or `make test` if you prefer, and then type `make install`. On most modern systems, Perl should compile and install within the space of a lunch break.

Now, if we type `perl -v`, we should see something like:

```
This is perl, v5.6.0 built for i686-linux
```

Installing on Windows

Installing ActivePerl is quite straightforward. Download ActiveState's Perl 5.6 installer for Windows/Intel from http://www.activestate.com/ActivePerl/download.htm. You'll need the latest version of Windows Installer from Microsoft as well, unless you're running Windows 2000.

On Windows NT or 2000, you should make sure you are logged in as an administrator, as the installer needs administrator privileges to set up your Perl installation.

Simply double-click the installer and follow the instructions. You can elect to install documentation and examples, as well as the Perl language itself. You can also choose anywhere on your system to install the Perl programs.

The only options that might cause some confusion are those related to installing Perl support into IIS (Internet Information Server) or PWS (Personal Web Server), if you have either of them installed. Setting up script mapping and ISAPI associations will enable you to run Perl programs within the web server. For development purposes, you should check all the boxes. We'll look at how to use Perl as a web scripting language in Chapter 12.

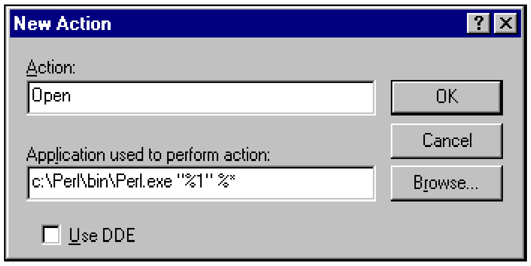
You can also run the installer program to modify or remove Perl at a later date.

Windows Troubleshooting

If you're following this book from beginning to finish, this may not have troubled you yet, but in true Windows style, it may be true that while installing, Perl was unable to associate itself with the `.plx` extension. Or, in English, when you double-click on a perl file icon in Windows, nothing happens. Similarly, you may have not noticed the pearl icon beside your perl files in Windows Explorer. If this is the case, don't panic. Just follow these instructions:

For Windows 9x users...

1. Open Windows Explorer and choose Folder Options... from the View menu.
2. When the Folder Options dialog box appears, click on the File Types tab. Now click on New Type.
3. In the Add New File Type dialog, add 'PLX file' to the Description Of Type box, and '.plx' to the Associated Extension text box. Then select the text/plain option for Content Type(MIME):
4. Now click New, and type Open in the Action text box, and `c:\Perl\bin\Perl.exe "%1" %*` (or whatever location you chose when installing Perl) under Application used to perform action as shown below.

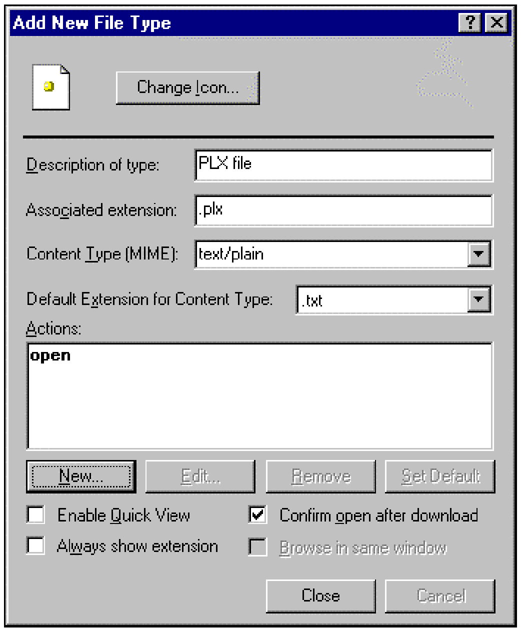


Action: Open

Application used to perform action: c:\Perl\bin\Perl.exe "%1" %*

☐ Use DDE

5. Click on OK to exit the New Action dialog, and now your screen will look something like this:



Description of type: PLX file

Associated extension: .plx

Content Type (MIME): text/plain

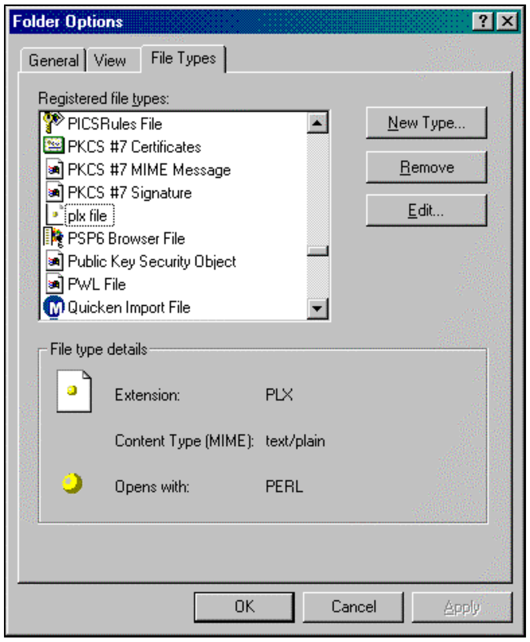
Default Extension for Content Type: .txt

Actions: open

☐ Enable Quick View ☑ Confirm open after download

☐ Always show extension ☐ Browse in same window

6. Finally, close the Add New File Type dialog, and you should now be able to see the following window:



Registered file types:
PICSRules File
PKCS #7 Certificates
PKCS #7 MIME Message
PKCS #7 Signature
plx file
PSP6 Browser File
Public Key Security Object
PWL File
Quicken Import File

File type details
Extension: PLX
Content Type (MIME): text/plain
Opens with: PERL

Notice that your `.plx` file displays the pearl icon, which means that we are finished, and everything will work according to plan.

For Windows NT / 2000 Users...

1. Open the Start menu and choose Control Panel from the Settings menu. Double click on the Folder Options control panel.

2. Select the File Types tab and hit New.

3. In the Create New Extension dialog, type PLX as your new extension.

4. Finally, select Advanced >> and Perl File from the drop-down list that eventually appears.

5. Hit OK, and for confirmation, the extension, along with a pearl icon and the associated File Type, 'Perl File' should have appeared in the main list box. Hit Close to leave the control panel.

How To Get Help

Perl comes with an excellent set of documentation. The interface to this system is through a command, itself a Perl program, called `perldoc`. UNIX users can also use the `man` command to get at the same information, but `perldoc` allows you to do interesting things, as you're about to see.

Perldoc

Typing `perldoc perl` from a command prompt will get you the table of contents and some basic information about Perl. The pages you're probably going to use the most are the Perl FAQ and 'perlfunc', which describes the built-in functions.

Because of this, `perldoc` has a special interface to these two pages. `perldoc -f` allows you to see information about a particular function, like this:

```
> perldoc -f print
print FILEHANDLE LIST
print LIST
print   Prints a string or a comma-separated list of strings.  Returns TRUE if successful. . . .
```

Similarly, `perldoc -q` allows you to search the Perl Frequently Asked Questions (FAQ) for any regular expression or keyword.

```
> perldoc -q reverse
Found in /usr/lib/perl5/5.6.0/pod/perlfaq4.pod
  How do I reverse a string?
    Use reverse() in scalar context, as documented in the reverse
    entry in the perlfunc manpage

    $reversed = reverse $string;
```

Now see if you can find that question about commercial Perl support that I mentioned earlier.

As well as the documentation pages for the language itself, whose names all start 'perl', there's an awful lot of other documentation out there, too. The reason for this is modules: files containing Perl code that can be used to help with a certain task. Later on we'll examine what modules are available and what they can help us do, but you should know that each Perl module, whether a core module that comes with the Perl package or one you download from the Internet, should contain its own documentation. We'll see how that's constructed later – for now though, know that you can use `perldoc` to get at this too. Here's the beginning of the documentation for the `Text::Wrap` module, which is used to wrap lines into paragraphs.

```
> perldoc Text::Wrap

5.6.0::Text  User Contributed Perl Documentation  Text::Wrap(3)


NAME
     Text::Wrap - line wrapping to form simple paragraphs

...
```

The pages are written in a special mark-up language called 'POD' (which sounds mysterious, but in fact stands for 'Plain Old Documentation'). The `perldoc` utility attempts to translate this into ordinary text when you view it, but if for some reason it cannot, you may need to specify the `-t` option to `perldoc`. If your documentation ends up looking like this:

```
> perldoc -q reverse
=head1 Found in /usr/lib/perl5/5.6.0/pod/perlfaq4.pod
=head2 How do I reverse a string?
Use reverse() in scalar context, as documented in
•<perlfunc/reverse>
   $reversed = reverse $string;
```

then you will need to run `perldoc -t -q reverse` instead.

Manpages

As well as the `perldoc` system, perl may well have installed its documentation in some other places as well. UNIX people can get at the standard documentation as `man` pages (providing the `MANPATH` environment variable includes the correct location), and ActiveState users should be able to find the documentation under ActivePerl | Online Documentation on the Start menu.

There's an exorbitant wealth of knowledge in these pages, and some are well beyond the scope of this book. Here then is a list of those relevant to this book, in roughly the order we touch on the topics in the book, plus one or two others that are handy and may satisfy your curiosity.

Documentation Page	**Subject**
`perl`	Introduction to Perl, and 'cover sheet'
`perltoc`	Table of contents – what's in the other pages
`perlfaq`	Index to the Frequently Asked Questions
`perlfaq1, perlfaq2… perlfaq9`	The Perl Frequently Asked Questions
`perlpod`	Plain Old Documentation and how to write it
`perlbook`	Information on Perl books
`perlstyle`	Perl style guide
`perllexwarn`	A guide to the new `use warnings` feature of Perl.
`perlsyn`	Perl's syntax rules
`perldata`	Perl's data types
`perlvar`	Perl's special variables
`perlop`	Perl's built-in operators
`perlunicode`	Perl's support for Unicode
`perlre`	Regular expression reference
`perlopentut`	Tutorial on opening files
`perlreftut`	Tutorial on using references
`perllol`	Lists of lists using references
`perlref`	Perl references

Documentation Page	Subject
`perlfunc`	Perl's built-in functions
`perlsub`	Creating subroutines
`perlrun`	Run-time options to perl
`perlmod`	Perl modules – what they are
`perlmodinstall`	How to install Perl modules
`perlmodlib`	Guide to the standard modules
`perlboot`	Randal Schwartz's Object Oriented Tutorial
`perltoot`	Tom Christiansen's Object Oriented Tutorial
`perltootc`	Tom Christiansen's Object Oriented Tutorial on Classes
`perlobj`	Object oriented programming in Perl
`perlbot`	The Bag of Object Tricks
`perltie`	A walk through tied objects
`perlipc`	Talking to other programs or networks
`perldbmfilter`	Controlling how Perl writes to databases
`perldiag`	What the error messages mean
`perldebug`	Debugging Perl programs
`perltrap`	Traps for the unwary programmer
`perlhist`	Perl's development history

If the Perl FAQ and the various documentation pages don't help answer your question, it's time to look for other sources of information.

Perl Resources

There is a tremendous amount of Perl information available in books and on the Internet. Let's have a look at some of the more prominent ones.

Websites

On the web, the first port of call is http://www.perl.com/, the main Perl community site, run by the publisher O'Reilly. This contains some good articles of interest to the Perl community and news from Perl's major developers, as well as a wealth of links, tips, reviews, and documentation.

It is also home to CPAN, the Comprehensive Perl Archive Network, a collection of ready-made programs, documents, notably the latest edition of the FAQ, some tutorials, and the Far More Than Everything You Wanted To Know About (FMTEYWTKA) series of more technical notes. Most useful of all, this site contains a huge (and they don't call it comprehensive for nothing!) collection of those Perl modules mentioned above. We'll fully cover the use of modules and some of the best ones in a later chapter.

Because CPAN is a network of sites, there are mirror sites around the world – the CPAN multiplexer takes you to your nearest site. Find it at http://www.perl.com/CPAN (note: no trailing slash!)

Other important Perl sites are:

- http://www.perlclinic.com/ – Paul Ingram's Perl Clinic, providing commercial Perl support and training
- http://www.perlfaq.com/ – an alternative, and very comprehensive, FAQ site
- http://www.tpj.com/ – the home of the Perl Journal
- http://www.activestate.com/ – the home of Perl on Windows
- http://www.perl.org/ – Perl Mongers, a worldwide umbrella organisation for Perl user groups
- http://www.perlarchive.com/ - another great source of articles, tutorials and information

Newsgroups

Perl is so cool it has its own Usenet hierarchy, comp.lang.perl.*. The groups in it are:

- comp.lang.perl.announce for Perl-related announcements: new modules, new versions of Perl, conferences and so on.
- comp.lang.perl.misc for general Perl chat and questions.
- comp.lang.perl.moderated, which requires prior registration before posting, but is excellent for sensible questions and in-depth discussion of Perl's niggly bits.
- comp.lang.perl.modules, for discussion and queries relating to creating and using Perl modules.
- comp.lang.perl.tk, for discussion and queries relating to the Tk graphical extensions.

IRC

If you've got a more urgent mindbender, or just want to hang around like-minded individuals, come join #perl on Efnet (See http://www.efnet.org/). Make sure you read the channel rules (at http://pound.perl.org/RTFM/) and the Perl documentation **thoroughly** first, though. Asking questions about CGI or topics covered in the FAQ or the perldoc documentation is highly inflammatory behavior.

If that hasn't put you off, come over and say hi to me. (I have no imagination, so my nick is usually Simon.)

Books

Of course, reading stuff from the net is a great way to learn, but I can't curl up in bed with a good web site. Not until I get myself a laptop, anyway.

In the meantime, there are a few good treeware resources available, too. O'Reilly has published some of the definitive books on Perl – *Learning Perl* (the Llama book), *Programming Perl* (the Camel book), and the *Perl Cookbook* are well known and well respected in the Perl community. Check out the book reviews pages housed at the http://www.perl.com/ and http://www.perl.org/ sites.

As for the best book for teaching yourself Perl, just keep reading...

Conventions

We have used a number of different styles of text and layout in the book to help differentiate between the different kinds of information. Here are examples of the styles we use and an explanation of what they mean:

Try It Out – A 'Try It Out' Example

'Try It Out' is our way of presenting a practical example.

How It Works

Then the 'How It Works' section explains what's going on.

Advice, hints and background information come in an indented, italicized font like this.

Important bits of information that you really shouldn't ignore come in boxes like this!

- **Important Words** are in a bold typeface.
- Words that appear on the screen in menus like the File or Window menu are in a similar font to what you see on screen.
- Keys that you press on the keyboard, like *Ctrl* and *Enter*, are in italics.

Perl code has two fonts. If it's a word that we're talking about in the text, for example, when discussing the `sub greeting {...}` subroutine, it's in a distinctive font. If it's a block of code that you can type in as a program and run, then it's shown in a gray box like this:

```
sub greeting {
        print "Hello, world!\n";
}
```

Sometimes you'll see code in a mixture of styles, like this:

```
sub greeting {
        print "Hello, world!\n";
}

&greeting();
```

This is meant to draw your attention to code that's new or relevant to the surrounding discussion (in the gray box), while showing it in the context of the code you've seen before (on the white background).

Where we show text to be entered at the command prompt, this will be shown as follows:

```
> perl helloworld.plx
```

And the output from the program will be shown in the same font, but lighter:

```
Hello World!
```

Downloading the Source Code

As you work through the examples in this book, you might decide that you prefer to type all the code in by hand. Many readers prefer this, because it's a good way to get familiar with the coding techniques that are being used.

Whether you want to type the code in or not, we have made all the source code for this book available at our web site, at the following address:

http://www.wrox.com

If you're one of those readers who likes to type in the code, you can use our files to check the results you should be getting. They should be your first stop if you think you might have typed in an error. If you're one of those readers who doesn't like typing, then downloading the source code from our web site is a must! Either way, it'll help you with updates and debugging.

Exercises

At the end of each of the first eleven chapters, you'll find a number of exercises. It is highly recommended you work through them. This book will give you the knowledge you need - but it is only through practice that you will hone your skills and get a true feel for what Perl can help you achieve. You can find our suggested solutions to the exercises in Appendix H at the back of the book and also for download from http://www.wrox.com, but remember that there's more than one way to do it, so they're not the only ways to solve the exercises.

Errata

We've made every effort to make sure that there are no errors in the text or the code. However, to err is human, and as such we recognize the need to keep you informed of any mistakes as they're spotted and corrected. Errata sheets are available for all our books at http://www.wrox.com/. If you find an error that hasn't already been reported, please let us know, by emailing support@wrox.com.

Our web site acts as a focus for other information and support, including the code from all our books, sample chapters, previews of upcoming titles, news of Wrox conferences, and articles and opinion on related topics. For a more in-depth look at our online support and errata, turn to Appendix J.

Customer Support

Our commitment to our readers doesn't stop when you walk out of the bookstore. We want you to get the most out of this book, and we provide a selection of support services for all our readers. See Appendix J for information about our support process and our community P2P mailing lists.

We've tried to make this book as accurate and enjoyable as possible, but what really matters is what the book actually does for you. Please let us know your views, either by returning the reply card in the back of the book, or by emailing us at feedback@wrox.com.