1

What Is Perl?

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Getting Perl
- Learning about the community
- Understanding the Perl documentation
- Using a terminal
- Writing your first Hello, World! program

My goodness, where to start? To even begin to cover a language with such a rich history and huge influence over the world of computing and the web is a daunting task, so this chapter just touches on the highlights.

By the time you finish with this chapter, you'll have a good understanding of the history of Perl and where to go to get more help when you need to know more than this book offers. Learning how to find the answers to your questions is probably one of the most valuable skills you can develop.

Before you install Perl, a word about Perl terminology — information that you need to know to converse intelligently with other Perl users.

The name of the language is Perl. It is not PERL. Larry Wall, the creator of Perl, originally wanted a name with positive connotations and named the language Pearl, but before its release, he discovered another programming language named Pearl, so he shortened the name to Perl.

The name of the language causes a bit of confusion. When people write *Perl* (uppercase), they are referring to the programming language you learn in this book. When people write `perl` (lowercase), they are referring to the binary executable used to run Perl, the language.

So `perl` is the binary and Perl is the language. The former parses and runs the latter: `perl` parses and runs Perl. If someone writes PERL, you know immediately that they're not familiar with the Perl language. This is why sometimes you see experienced programmers use PERL to refer to poorly written Perl programs.

Due to the wording of the original documentation that shipped with Perl, many programmers assume that PERL is an acronym for *Practical Extraction and Report Language*. However `perlfaq1` — the documentation that shipped with Perl — sets the record straight:

```
... never write "PERL", because perl is not an acronym, apocryphal
 folklore and post-facto expansions notwithstanding.
```

Remember, there is no such thing as PERL. It's Perl, the language, or `perl`, the executable.

DYNAMIC PROGRAMMING LANGUAGES

Perl, Python, Ruby, and PHP are all examples of *dynamic programming languages*. In contrast to languages such as Java, C++, and other *static programming languages*, the dynamic languages often delay certain things until run time that static languages might decide at compile time, such as determining which class a method will dispatch to. Without going into detail beyond the scope of this book, dynamic languages tend to be rapid to develop in, but have certain kinds of errors that are less common in static languages.

Discussions about dynamic and static typing are about type theory, and the terms are poorly defined. However, there is one solid rule you should remember: Computer *scientists* have reasonable disagreements about type theory, whereas computer *programmers* have unreasonable ones. If you get into "static versus dynamic languages" debates, and you don't understand type theory, you're going to sound like a fool to those who do. So don't do that.

PERL TODAY

Today, Perl is widely used throughout the world. It comes standard on every major operating system apart from Windows and is still extensively used in web development, thus driving many websites. New startups choose Perl as their language of choice for data processing, system administration, web development, and other uses.

As of this writing, Ricardo Signes, a long time Perl hacker, is overseeing the development of Perl. Perl 6, a new language with roots in Perl 5, is being actively worked on with several interesting implementations, including a Niecza, which runs on Mono/.NET.

PERL JOB OPPORTUNITIES

A quick search of many job sites shows plenty of opportunities, but there are fewer competent developers vying for these roles. If a career in Perl interests you, you can also check out `http://jobs.perl.org/` for a website dedicated to listing jobs that have Perl as their major technology, compared to jobs where Perl is merely used incidentally.

This book mostly focuses on 5.8.*x* and 5.10.*x* versions of Perl, even though support for both of these has officially been discontinued. Why? This was a difficult decision to make, but there were several reasons for this decision. An important consideration is that surveys show most businesses still run these versions of Perl. It's a strange thing for a book author to stand up and say, "This book targets an unsupported version of the language," but you go to war with the Perl you have, not the Perl you want.

Fortunately, this choice isn't as bad as it might sound. The Perl 5 Porters (known as "P5P") work hard to keep new releases of Perl backward compatible. Perl 5.14.2 ships with almost half a million tests (455,832, to be exact) to ensure that Perl works exactly as intended. Thus, what you learn to write throughout this book generally works unmodified on later versions of Perl.

GETTING PERL

Obviously, it's difficult to program Perl if you don't have it installed on your computer; this section covers several methods for doing this. Even if you already have Perl installed, you should to read this section anyway because if your system depends on your Perl installation, you might want to install a separate version to avoid changing behavior that your system requires.

So how do you get Perl? Well, you're in luck. Almost every major operating system aside from Windows ships with Perl by default. This is often referred to as the *system Perl*. You can test whether you already have Perl installed by opening up a terminal and typing **`perl -v`** at the command line. Currently, on my MacBook Pro, this prints the following:

```
$ perl -v
This is perl 5, version 14, subversion 2 (v5.14.2) built for darwin-2level
Copyright 1987-2011, Larry Wall
Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.
Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.
```

Perl is supported on more than 100 platforms — did you even know there were that many? If you want a different version of Perl than what you already have installed, go to `http://www.perl.org/get.html`.

> ***NOTE*** *If you use OS X, you already have Perl installed. However, you will eventually build modules or install other modules. To do this, you need to install the Developer Tools found on your OS X install DVD or in Apple's AppStore. Only the UNIX Development Support tools are required, but there's no harm (other than disk space) in installing all of them. Why Apple built a wonderful computer for developers and made the development tools optional is one of life's many inscrutable mysteries.*

Working with Non-Windows Platforms: perlbrew

If you do not run Windows, check out `perlbrew` (`http://www.perlbrew.pl/`). This tool enables you to install multiple and run different versions of Perl.

Running different Perl installations is important because there's a good chance that some of your operating system depends on the behavior of the system Perl. Therefore, using `perlbrew` to install your own versions of Perl not only ensures that you don't need to worry about breaking your system Perl, but you also can play with different versions.

That being said, so long as you're not overwriting any modules that your system Perl uses, it's fine to use your system Perl for learning Perl. It's also usually fine to upgrade your system modules, but it's not recommended. If a core module your system depends on changes in an incompatible way, the results are unpredictable. Windows does not have this problem because it does not depend on Perl.

If your system has both `bash` and `curl` installed, you can try to install `perlbrew` with the following command-line command:

```
curl -kL http://xrl.us/perlbrewinstall | bash
```

If you don't have `curl` installed but you do have `wget`, you can install `perlbrew` with this:

```
wget --no-check-certificate -O - http://install.perlbrew.pl | bash
```

If that works on your system, it should enable you to easily install multiple versions of Perl without superuser (root, or administrator) permissions. It's then easy to switch between those versions. This has many benefits, including the following:

- It's easy to try new versions of Perl.
- You don't risk breaking your system's Perl.
- You don't need superuser permission to install *Comprehensive Perl Archive Network* (CPAN) modules.
- You can test production code on newer versions of Perl.

To install and use Perl version 5.14.2, type the following (but see the `perlbrew available` command below):

```
perlbrew install perl-5.14.2
perlbrew switch perl-5.14.2
```

The installation takes a while because `perlbrew` needs to download and build the version of Perl you're asking for.

After `perlbrew` installs, you can use the following commands:

- **`perlbrew help`:** Typing **`perlbrew help`** shows you how to use `perlbrew`. It's quite easy.
- **Installing an older Perl version:** If you want to install an older version of Perl, you can run the following:

  ```
  perlbrew install perl-5.8.3
  ```

- **Switching versions:** You can run `perlbrew list` to see which versions of Perl you have installed and can switch to a different version. Following is the author's setup:

  ```
  $ perlbrew list
    perl-5.10.1
    perl-5.12.3
    perl-5.14.0
  * perl-5.14.2
    perl-5.8.3
  ```

  The asterisk before the version indicates which version of Perl you're running at the moment.

- **Testing code:** To test your code against different versions of Perl, use the following:

  ```
  perlbrew exec myprogram.pl
  ```

  The author used this command extensively while writing this book because it's extremely useful when you want to find out if your code is compatible with different versions of Perl.

- **Available versions:** As of this writing, following are the Perl versions available to install on the author's computer. The `perlbrew available` command lists all available versions:

  ```
  $ perlbrew available
    perl-5.15.4
  i perl-5.14.2
    perl-5.12.4
  i perl-5.10.1
    perl-5.8.9
    perl-5.6.2
    perl5.005_04
    perl5.004_05
    perl5.003_07
  ```

  The leading `i` indicates which versions of Perl you have installed, and the list of available versions will grow over time.

If you can use `perlbrew`, it will make your programming life much more pleasant.

> ***NOTE*** *Using* `perlbrew` *is great, but it requires that you already have Perl 5.8 or newer installed on your system. However, as because version 5.8 was released in July of 2002 (see as shown in Table 1-1), this is generally not a problem.*

Using Windows

Windows is one of the few operating systems that does not include Perl by default. This makes things a bit more difficult, but you have a wide variety of options here. Your author recommends Strawberry Perl, but ActivePerl is also an excellent choice. Cygwin is only recommended only if you want a Linux emulation layer.

Strawberry Perl

Strawberry Perl (`http://strawberryperl.com`) is the newest option for Windows, but it's the one many developers prefer today. It's also free and it's the choice of Perl that Larry Wall utilizes when he uses Windows. Strawberry Perl does not offer commercial support. Like many open source projects, support is excellent — but on a volunteer basis.

When you install Strawberry Perl, the following software is installed with it:

- Mingw GCC C/C++ compiler
- `dmake` `make` tool
- `ExtUtils::CBuilder` and `ExtUtils::ParseXS`
- `Bundle::CPAN`
- `Bundle::LWP` (which provides more reliable HTTP CPAN repository support)
- `XML::Parser` and `XML::LibXML`, which enables most CPAN XML modules
- DBI and DBD drivers for SQLite, ODBC, MySQL, and Postgres
- Additional minor modules to enhance the stability of Win32 Perl.

Don't worry about what all this means for now. As you move further along in the book, these items will start to make sense. Just know that they make Perl on Windows easy enough to use that it rivals Perl on Linux for many tasks. Unless you have a particular reason to use another version of Perl, the author recommends Strawberry Perl. Some things to remember with Strawberry Perl follow:

- **Pros:** Strawberry Perl "just works." Almost everything you need to develop Perl is bundled with it, including many tools that are usually mandatory in a work environment.
- **Cons:** It's relatively new and companies that rely on Windows are sometimes uncomfortable with software that lacks commercial support.

ActiveState Perl

Another strong alternative for Windows is ActivePerl (`http://www.activestate.com/activeperl`). It's free, but commercial support is provided. ActivePerl has been available for more than a decade and is probably the most popular Perl for Windows. When considering ActivePerl, remember the following:

- **Pros:** ActivePerl has been around for more than a decade, and it is maintained by a company with a strong history of supporting Perl and dynamic languages. It's also often updated faster than Strawberry Perl. Additionally, some binary packages are easier to install with ActiveState than with Strawberry Perl.
- **Cons**: ActivePerl does not ship with the full set of tools with which Strawberry Perl ships. Further, it contains some non-open source utilities and, unlike Strawberry Perl, it cannot be embedded in other open source projects.

Cygwin

One way to run Perl on Windows is to install Cygwin, a free Linux emulator for Windows. You can download Cygwin from `http://www.cygwin.com/`. Click the Install Cygwin link for instructions.

By default, Cygwin does not install Perl. You can easily find instructions on the web for installing and running Perl under Cygwin, including many useful YouTube videos. If you go this route, make sure that when you install Cygwin, you select both Perl and the GCC/C++ packages from Development menu when you're given a choice on which packages to install. However, to get the most out of Perl on Cygwin, make sure you have the following packages installed:

- `perl`
- `gcc/C++`
- `gnupg`
- `gzip`
- `lynx`
- `make`
- `ncftp`
- `ncftpget`
- `tar`
- `unzip`
- `wget`

This list should cover most of what you need. Keep the following in mind:

- **Pros:** With Cygwin, you get a Linux environment, which means that most Perl programs can run unchanged.
- **Cons:** As an emulation layer, it tends to be a bit slow. It's also a bit difficult to install everything correctly if you're not used to it.

> ***NOTE*** *If you have issues getting Perl to run on Windows, go to* `http://win32.perl.org/`*. Your easiest (and best) options are to go with the ActiveState or Strawberry Perl options, but* `win32.perl.org` *gives you plenty of answers to questions you may encounter.*

THE PERL COMMUNITY

You didn't read detailed instructions on how to install Perl for Windows or how to install alternative versions of Perl on your operating system of choice. As mentioned previously, Perl is supported on more than 100 platforms, and although the author has tried writing instructions on how to do this in the past, the impossibility of handling that obscure error that someone inevitably has makes this difficult. Fortunately, Perl is easy to install on Windows, and the language has a strong community supporting it; this community can help you work through even the most unusual issues.

Because the Wrox "Programmer to Programmer" series targets experienced developers looking to expand their skills, you, the developer, will likely be familiar with software installation. If you're new to programming, you might need a bit more help. Either way, the following sections discuss a variety of resources to help you start.

> ***NOTE*** *Consult these sources regularly when you get stuck on a particular problem. This is one of the lovely things about the open source community: Quality help is widely available, and it's free. There's no need to struggle on your own when so many people can help you learn Perl.*

IRC

Internet Relay Chat (IRC) has been around since 1988, and it's often a great way to get questions answered "in real time." With IRC, you have several options:

- **mIRC (`http://www.mirc.net/`):** For Windows, this is probably the most popular IRC client, but it's shareware, and you can use it only for 30 days before paying.
- **KVIrc (`http://www.kvirc.net/`):** This is a good, free choice for a graphical IRC client, and it's available for most platforms.
- **Colloquy (`http://colloquy.info/`):** For OS X, the author uses this.
- **Chatzilla (`http://chatzilla.hacksrus.com/`):** If you use the Firefox browser, it has the capable Chatzilla add-on, which this should work regardless of which operating system you choose.
- **freenode:** You can also access `freenode` with any browser via `http://webchat.freenode.net/`.

Actually, any IRC client you're comfortable with is fine.

When you get on IRC, connect to the `irc.freenode.net` server and join `#perl`. The `#perl` channel generally has plenty of users, and you can get many Perl questions answered quickly and easily — or at least get told where to RTFM, which stands for *Read The Manual*. (the "F" is silent.)

If you're not familiar with IRC, hit you favorite search engine and search for **list of IRC commands**. You can also consult the Wikipedia page for IRC (`http://en.wikipedia.org/wiki/Irc`) for more information, including lists of other IRC clients.

PerlMonks

PerlMonks (`http://www.perlmonks.org/`) is a fantastically useful Perl site that's been around for more than a decade. Your author joined in 2000, unsurprisingly as "Ovid," and has been a regular contributor for years.

In the top right corner of the site, you see many useful links. Seekers of Perl Wisdom is probably the most useful when you need an answer to a problem. When you first post a question, it shows in Newest Nodes, and many people follow that to try to help answer the new questions. Fortunately, the regular users at PerlMonks generally don't suffer as much from the "first post" silliness you often find at other sites.

In addition to answering questions, PerlMonks has book reviews, *Meditations* (a section for people who just want to muse about Perl-related things), tutorials, Perl news, site discussion, and a *chatterbox* for those who just want casual conversation or have a quick question.

If you're serious about learning Perl, PerlMonks is a good place to start. Many of the top minds in Perl hang out there, and it's a good resource with plenty of history to search through. PerlMonks is "all Perl, all the time." Joe Bob says, "Check it out."

Perl Mongers

For those who like a bit of real-life interaction (and who doesn't?), there's also Perl Mongers (`http://www.pm.org/`). Founded by `brian d foy` in 1997, Perl Mongers is an organization of Perl hackers in different cities who meet periodically to, well, do whatever they want. Your author ran the Perl Mongers group in Portland, Oregon (`Portland.pm`) for several years, and has attended Perl Mongers meetings in a number of countries.

The local Perl Mongers user groups are Perl enthusiasts who enjoy hanging out together and talking about stuff. Sometimes that stuff is Perl. The `Portland.pm` group generally schedules technical talks followed by a "social" at a local pub, often the excellent Lucky Lab in Portland, Oregon. If you ever visit Portland, check out that pub.

There are Perl Mongers groups on every continent except Antarctica, but there was discussion of an Antarctica group starting up when Mongers found out there was a Perl programmer there. If you live near a major city, there's a good chance there's a Perl Mongers group close to you. If not, create one!

StackOverflow

StackOverflow (`http://stackoverflow.com/`) was created in 2008 by Joel Spolsky and Jeff Atwood as an "open" site for anyone to ask programming-related questions. It has spun off numerous related sites and has become extremely popular as the site where you can ask just about any technology question.

Perl questions are answered quickly with solid information, and you can easily see the "rating" of the users who respond to your questions. Because of how questions are tagged, it's easy to quickly drill down to questions that might be relevant to your situation.

LEARNING HOW TO ASK EFFECTIVE QUESTIONS

Quite often on PerlMonks, StackOverflow, or other sites, you see a question like "I tried to use module XYZ, but when I tried to print with it, it didn't work. What am I doing wrong?"

That's it. "Didn't work" isn't explained. No code sample is provided. Nothing.

Here's how to ask an effective question:

1. State what you're trying to do.
2. Explain how you tried to do it.
3. Explain what result you expected.
4. Explain what result you had instead.

"How you tried to do it" often involves posting a minimal code sample. Posting no code is just as bad as posting 500 lines of code. Just give people an idea of what you're trying to do, and answer any follow-up questions they have (if any).

It's also a good idea to indicate how you already tried to find an answer. People are often more helpful if it looks like you've already tried to find an answer to a basic question.

TRY IT OUT Register for a Free Account at PerlMonks

Every chapter, has "Try It Out" sections, but for this first chapter, there's not much to "try out." After the "Try It Out" sections, there is usually a "How It Works" section explaining what you've just done, but this first one is self-explanatory, so "How It Works" is skipped this time. Instead, this Try It Out is to nudge you to PerlMonks and get you started on your journey to Perl. Just do the following:

1. Go to `http://www.perlmonks.org/` and click Create a New User. (The link is on the right, below the login box.)
2. Read some of the useful information, such as "Don't create a username people can't type."
3. Fill out the small form and wait for your confirmation e-mail.

I encourage you to click the Newest Nodes or Seekers of Perl Wisdom links and read through some of the material there. Much, if not most, of the information might seem foreign to you, but by the time you finish this book, you'll be answering questions for newcomers. Or you should be: Answering questions is one of the best ways to learn new material.

USING PERLDOC

Now that you've installed Perl, the first thing you should do is get acquainted with the extensive Perl documentation that ships with the language. As this book covers various topics, a `perldoc` tip often prefixes sections, like this:

```
perldoc perlnumber
```

If you type **`perldoc perlnumber`** into your terminal, you receive an introduction to how numbers are used in Perl. If you prefer a web browser, go to `http://perldoc.perl.org/`, select your Perl version, and then go to: `http://perldoc.perl.org/perlnumber.html`.

By constantly reinforcing `perldoc` throughout this text, you get the tools to find answers to most questions yourself. This is one bit of advice the author would have liked to received when starting his Perl journey in the '90s. You don't need to memorize the material in the documentation, but as you become more familiar with it, you'll find it easier to remember where to look it up later.

Understanding the Structure of perldoc

The Perl documentation is written in POD, short for *Plain Old Documentation*. POD is an easy-to-learn markup language for documenting Perl. It's easy enough to learn (and you will in Chapter 11), but flexible enough, that many authors write their books in POD.

When you type **`perldoc <documentation name>`**, the program searches through parts of your system where it thinks the documentation may be found, looking for a file with a `.pod` or `.pm` extension. The `.pod` extension is preferred, and `.pm` is used if the file with the `.pm` extension has embedded POD and the `.pod` extension is not found. The program then formats the POD and displays it. For earlier versions of `perldoc`, you could add the `-v` switch to see where the `perldoc` command is looking for your POD:

```
perldoc -v perldoc
```

If your version of `perldoc` supports (see `perldoc perldoc`) this, use the `-D` switch to see where `perldoc` is looking for the documentation. The `-v` switch now displays the description of Perl's built-in variables:

```
perldoc -v '$_'
perldoc -v '@ARGV'
```

You can also type **`perldoc perlvar`** to see all of Perl's built-in variables.

You can read `perldoc perldoc` for more information about how to customize `perldoc` output or to see what other command-line switches it supports.

Getting Started with perldoc

The first thing you want to do is type **`perldoc perl`**. This gives you a brief description of some of what Perl can do and quickly directs you to

```
perldoc perlintro
```

That's a gentle introduction to Perl. If you're dedicated, you could start there and not buy this or any other Perl book. That approach works if you have lots of time and patience. This book presents what you need to know most, including where to get more information.

The `perlintro` is clear but terse. It assumes that you already know how to program and rushes through the basic features of the language. As a result, there are many bits and pieces you should be aware of but won't be. So to follow up on the `perlintro`, you'll want:

```
perldoc perltoc
```

As you might expect, that's the *Table of Contents* for the Perl documentation. For Perl 5.14.2, that Table of Contents is more than 20,000 lines! That's a huge amount of documentation. It's longer than many of the chapters in this book, and your author hopes his publisher doesn't notice. In contrast, Perl 5.8.3's Table of Contents weighs in at a measly 11,911 lines. However, this book mostly focuses on 5.8 and 5.10, and it won't actually talk (much) about what's in those newer documents.

Using Tutorials and FAQs

Perl comes bundled with many tutorials you can read with `perldoc`. Table 1-1 lists the tutorials that are some of the popular ones included in Perl version 5.8.3. You can type **`perldoc <tutorialname>`** to read these tutorials.

TABLE 1-1: perldoc Tutorials

TUTORIAL	DESCRIPTION
`perlreftut`	Tutorial on references
`perldsc`	Data structures cookbook
`perllol`	Data structures: arrays of arrays
`perlrequick`	Regular expression quickstart
`perlretut`	Regular expression tutorial
`perlboot`	Object Oriented (OO) Perl for beginners
`perltoot`	OO tutorial, part 1
`perltooc`	OO tutorial, part 2
`perlbot`	OO tricks and examples
`perlstyle`	Style guide
`perlcheat`	Cheat sheet
`perltrap`	Traps for the unwary
`perldebtut`	Debugger tutorial

> ***NOTE*** *The object oriented (OO) Perl documentation which ships with Perl 5.8 and 5.10 was very useful in its day but is now considered to be rather out of date. Its examples and recommended practices should be considered suspect. We'll be covering OO starting in chapter 12.*

Because the author had so much fun cutting and pasting from the documentation and padding the page count, Table 1-2 lists the Frequently Asked Questions (FAQs) that ship with Perl.

TABLE 1-2: perlfaq

FAQ	DESCRIPTION
`perlfaq`	Perl FAQs
`perlfaq1`	General questions about Perl
`perlfaq2`	Obtaining and learning about Perl
`perlfaq3`	Programming tools
`perlfaq4`	Data manipulation
`perlfaq5`	Files and formats
`perlfaq6`	Regexes (regular expressions)
`perlfaq7`	Perl language issues
`perlfaq8`	System interaction
`perlfaq9`	Networking

These FAQs are extensive. For example, the following are some of the questions addressed in `perlfaq2`:

- What machines support Perl? Where do I get Perl?
- How can I get a binary version of Perl?
- I don't have a C compiler on my system. How can I compile Perl?
- I copied the Perl binary from one machine to another, but scripts don't work. Why?

What's nice is that for any of these questions, you can type **`perldoc -q "something I'm looking for"`** and `perldoc` will spit out the sections from any FAQ that matches the term you give it. (Actually, `perldoc` matches against *regular expressions*, which aren't covering until Chapter 8, so pretend you didn't read that bit.)

A full reference manual also ships with the Perl documentation along with extensive information about the internals of Perl (not for the faint of heart), linking Perl to C and C++ programs, platform-specific information, and other things that aren't covered in this book.

Using the perldoc -f function

One of the most useful `perldoc` commands is `perldoc -f` . When you type **`perldoc -f`**, followed by a function name, you can see a complete description of the function in question and quite possibly far more than you ever need to know. For example, `perldoc -f my` displays the following:

```
my EXPR
my TYPE EXPR
my EXPR : ATTRS
my TYPE EXPR : ATTRS
        A "my" declares the listed variables to be local (lexically) to
        the enclosing block, file, or "eval".  If more than one value
        is listed, the list must be placed in parentheses.
        The exact semantics and interface of TYPE and ATTRS are still
        evolving.  TYPE is currently bound to the use of the "fields"
        pragma, and attributes are handled using the "attributes"
        pragma, or starting from Perl 5.8.0 also via the
        "Attribute::Handlers" module.  See "Private Variables via my()"
        in perlsub for details, and fields, attributes, and
        Attribute::Handlers.
```

It starts with the grammar for the function and then a brief (and sometimes verbose) explanation of that function. In the preceding example, the grammar could represent any of the following:

```
my $dog;
my Dog $spot;
my $dog : HasSpots;
my Dog $spot : HasSpots;
```

> ***NOTE*** *In real-world Perl, you almost always see the first form from the previous code,* `my $dog`*, and not the three that follow it. The semantics of the last three forms were never well defined and caused confusion, so people don't use them. This is an example where the docs show you what you can do, not what you should do.*

USING A TERMINAL WINDOW

You can skip this section if you already know how to use a terminal window. Otherwise, this section will explain the absolute minimum you need to know about opening and using a terminal window. As with a number of other languages, if you want to program in Perl, much of your professional life will be spent in a terminal window.

Using the Command Line

Perl comes from a UNIX background and, as a result, is often run from a terminal window. Unlike many graphical user interface (GUI) systems, terminals enable you to type commands directly into the system rather than clicking an icon on a screen or selecting items from menus. Getting used to the command line not only gives you all the power of a GUI system, but also leverages the considerable power of the command line. If you're not familiar with this method, hit your favorite search engine for how to use the command line on your system, but for now, this section concentrates on getting a terminal window open.

This isn't difficult, but ask a geek friend for help if you get stuck.

Working with the Terminal Window in Linux

If you're familiar with Linux, you probably already know about the terminal window. Unfortunately, because there are more than 100 Linux distributions and many different window managers, it's impossible to tell you how to do this on your system. However, following are some general tips:

1. Look for an icon on your desktop that looks like a computer screen. It may say Terminal or Console next to it. Try double-clicking that. You can also often right-click your desktop and look for `open terminal` or something similar.
2. In the menu system under the `System` folder, you may also find the `Konsole` or `Gnome Terminal` program.
3. Search your desktop menu for the words **terminal** or **console**. Many Linux systems have icons on their menus, and you may see a terminal icon there.

Working with the Terminal Window in Mac OS X

If you're on a Mac, you can follow these steps:

1. Go to your desktop and press Command-Shift-G (in other words, hit all those keys at the same time). This brings up a *Go to folder* dialog.
2. Type **`/Applications/Utilities`** in the text window, and click Go.
3. Scroll through the applications until you see the Terminal icon.
4. Drag this to the dock. You'll use the terminal a lot in this book, so you want to have this handy.

A quick check of a search engine for **mac command line** or **learning os x terminal** should bring you up to speed on some of the basic commands. When you use the Mac command line, you'll find that most standard UNIX/Linux commands operate the same way.

> ***NOTE*** *Alternatively, go to iTerm2 (`http://www.iterm2.com/`) to download their free terminal application. The author uses iTerm2, which is an excellent replacement for `Terminal.app` that is included with OS X.*

Working with the Terminal Window in Windows

For Windows, you have a couple options:

- If you've installed Cygwin, you can double-click the Cygwin desktop icon (not the installer!) and you'll automatically be at a command-line prompt ready to go.
- Press the Windows key and r at the same time. This should bring up a Run dialog box. Type **`cmd`** (short for command) into the box, click OK, and a terminal window pops up.
- You can bring up the Run dialog box by clicking Start; then you should see Run as one of the menu items. Click that and it's the same procedure: type **`cmd`** into that box and click OK.

If you don't like the standard terminal on Windows, some people prefer `console`, available via free download at `http://sourceforge.net/projects/console/`.

For Windows, the terminal window is sometimes referred to as a *DOS window*. DOS stands for *Disk Operating System* and earlier versions of Windows were based on DOS with a Window manager on top. Today, Windows is a GUI system, and the DOS window is an emulation layer, but the commands have not changed much over time.

If you're unfamiliar with the Windows command line, search the Internet for **list of DOS commands** to learn more about this environment.

Creating a Work Directory

Now that you have a terminal window open, you might want to find out where you are on your system. To see the current directory you are in, you can type **`pwd`** (print working directory) on Linux or OS X, or **`cd`** (with no arguments) on Windows. You can type **ls** on Linux or OS X to see a list of files in the current directory or **dir** if you're on Windows.

> ***NOTE*** *A folder in Windows or OS X is what most other operating systems refer to as a directory. This text says directory.*

Create a folder named `wroxperl` and change to it. For most major operating systems, type this:

```
mkdir wroxperl
cd wroxperl
```

You should now be in an empty directory, suitable for creating your sample programs. When you create them, make them in separate directories named `chapter1`, `chapter2`, and so on. This makes it easier to organize and refer back to them. So go ahead and create a `chapter1` directory now and change to it:

```
mkdir chapter1
cd chapter1
```

You won't need this until you get to the "Hello, World!" section (it's a law that all programming books start with this), but stay in the terminal for now to get used to the `perldoc` command.

INSTALLING THE PERLDOC COMMAND

You probably have `perldoc` installed. You can verify this by typing `perldoc -h` to bring up a help page for `perldoc`. Annoyingly enough, some systems that include Perl by default don't include the `perldoc` command even though it is installed by default when you install Perl manually. If your system uses `apt`, you can install `perldoc` with:

```
sudo apt-get install perl-doc
```

Unfortunately, that won't work on systems that don't use `apt`, and because Perl is available on more than 100 platforms, this book can't cover them all. Thus, in the event that you don't have `perldoc` installed, try hitting IRC, PerlMonks, StackOverflow, or your favorite search engine to find out how to install `perldoc`. Or ask your geek friend to do it for you. Pizza is a great payment.

TRY IT OUT Getting Used to perldoc

You don't want to just read about the command line; you must get used to it, too. You'll see a lot of Perl's internal documentation here. You don't actually have to read it right now, but you should be familiar enough with seeing it to know where to look for more information.

1. Open a terminal. Actually, you should already have one open by this time. To navigate, try the following commands by typing the following:
   - **`q`:** To exit (quit) `perldoc`
   - **Spacebar or the down arrow:** This enables you to scroll through the pages
   - **Forward slash (`/`) and some letters:** Enables you to search through the documentation

Unfortunately, most of those commands depend on you having a sane pager program, such as `less`. You can set the `PAGER` environment variable to your desired pager or just play around with your `perldoc` to see which commands it accepts.

2. See which `perldoc` version you're using.

```
perldoc -V
```

3. Read about what the `perldoc` command can do on your version of Perl.

```
perldoc perldoc
```

4. Read (skim) about Perl.

```
perldoc perl
```

5. Read the Table of Contents. (Actually, there's probably too much here to read).

```
perldoc perltoc
```

6. Search for information in the FAQs, which provide a wealth of information.

```
perldoc -q variable
perldoc -q file
```

7. Read about some Perl functions.

```
perldoc -f print
perldoc -f map
```

8. If your Perl is new enough (5.12 or better), you can read about some built-in Perl variables. Older versions of Perl use the `-v` to "verbosely" show you where `perldoc` is searching for your documentation. Newer versions of Perl use the `-D` switch for this.

```
perldoc -v '$_'
perldoc -v '@ARGV'
```

How It Works

The `perldoc` command searches all places where it thinks Perl documentation may be living and reads likely files it finds to determine if they contain the information you need. If you are curious to know, you can run the following command to see for yourself where it's (mostly) searching:

```
perl -le 'print join "\n", @INC, map {split /:/} @ENV{qw/PERL5LIB PATH/}'
```

If you understand that command and what it's doing, there's a good chance you don't need this book. By the time you're done with this book, you'll understand it.

CREATING HELLO, WORLD!

Now that you've read far too much documentation (who am I kidding? You skimmed it), it is time for that traditional rite (write?) of passage, "Hello, World!" As one friend explained to me, he was proud that he could write "Hello, World!" in 15 programming languages — though he could program in none. Try to avoid that, okay?

Writing Your First Program

First, open your terminal and type this:

```
perl -e 'print "Hello, Wrox!\n"'
```

Oh, wait. Sorry Windows people. You have to type this:

```
perl -e "print \"Hello, Wrox!\n\""
```

Except that it might not work, depending on your version of Windows. See `perldoc perlfaq3` and read the section "Why don't Perl one-liners work on my DOS/Mac/VMS system?" to understand why your life is difficult. If you have a Mac, the "Mac" section likely does not apply to you because OS X handles Perl and the command line quite well, thank you.

Aside from your author blatantly patronizing the publisher, the "Hello, Wrox!" snippet shows something common about Perl: running Perl from the command line. This won't be covered much in the book, but as you get more familiar with Perl, you'll see people doing things like this:

```
perl -pi.bak -e 's/version = 13/version = 14/' <list of files>
```

That changes all strings in `<list of files>` matching "`version = 13`" to "`version = 14`" and create backups of all those files with a `.bak` extension. That's more or less equivalent to the following Perl program that is also listed in `perldoc perlrun`. (Although it's been cleaned up to be "safer.")

```
#!/usr/bin/perl
my $extension = '.bak';
my $oldargv;
LINE: while (<>) {
    if ($ARGV ne $oldargv) {
        my $backup;
        if ($extension !~ /\*/) {
            $backup = $ARGV . $extension;
        }
        else {
            ($backup = $extension) =~ s/\*/$ARGV/g;
        }
        rename($ARGV, $backup);
        open(ARGVOUT, ">", $ARGV)
          or die "Cannot open '$ARGV' for writing: $!";;
        select(ARGVOUT);
        $oldargv = $ARGV;
    }
    s/version = 13/version = 14/;
}
continue {
    print;  # this prints to original filename
}
select(STDOUT);
```

As you can see, using Perl on the command line effectively gives you a lot of power to get things done quickly. You can read `perldoc perlrun` to understand some of this, but search for `perl` **one-liners** online to see what you can do if you're interested in this area.

Getting back to "Hello, World!", the general way you write a Perl program is to save a file with the program code and then type **`perl <programname>`**. The first line of the program is often the

shebang line, which you learn more about in a bit. After that is your program text. All you need to do to get a basic Perl program running is to type up your program, save it (usually with a `.pl` extension), and then type **`perl <programname>`**.

Listing 1-1 is a short Perl program that shows how a simple program may look. You learn more about the `strict`, `warnings`, and `diagnostics` in Chapter 3.

LISTING 1-1: Hello, World!

```
#!perl
use strict;
use warnings;
use diagnostics;
# this is a comment
print "Hello, World!\n"; # so is this
```

TRY IT OUT Your First Perl Program

This is a simple example to demonstrate writing a Perl program, saving it, and running it.

1. Type the following code into your favorite editor, and save it as `bonjour.pl`.

```
#!/usr/bin/perl
# "Hello world!, in French
print "Bonjour, tout le monde!\n";
```

2. From the command line type **`cd`** (change directory) into the directory where you saved your program, and type **`perl bonjour.pl`**. You should see this output:

```
Bonjour, tout le monde!
```

How It Works

On the command line, when you type `perl` followed by the name of a file containing a Perl program, Perl reads that file, parses the code, and executes it. The sharp (`#`) begins a comment. It can be on its own line or embedded in a line after some code.

> ***NOTE*** *People sometimes mistakenly refer to Perl as an* interpreted language*, but it's not quite a compiled one, either. Like many modern languages, it falls somewhere in between the two. When you run a program with* `perl programname.pl`*, Perl first compiles your Perl down to a set of* opcodes *and then executes those. Because there is generally no complicated compile/link phase for executing a Perl program, it's very easy to quickly make and test changes to programs.*

WINDOWS AND THE .PL EXTENSION

On Windows, when you install Perl, you'll often find that the `.pl` extension is associated with Perl in the registry. New Perl programmers on Windows often double-click a Perl program icon and then wonder why they see a brief flash of a console before it disappears, taking their program output with it. That's because Perl is usually run from the command line. One trick to work around this is to add the following code as the last line of your program:

```
<STDIN>;
```

That causes Perl to hang, waiting for you to enter some input, leaving the console up. Just pressing Enter makes the console disappear. This is explained more when you cover user interaction in Chapter 17, but for now do not use this trick. Get used to running Perl from the command line.

Shebang Lines

The first line of a Perl program often starts with a *shebang line*. A shebang line starts with "sharp" (`#`) and an exclamation point, also known as a "bang" (`!`), hence the term shebang. The line is followed by a path telling the shell where to find the interpreter that is used to execute the program.

On a system that understands the `chmod` command, you can type **`chmod +x programname`** to make the program directly executable. If it's in your path, you can then type **`programname`** to run the program. Otherwise, you can type the full or relative path to the program to execute it.

For example, if you're in `/Users/ovid/wroxperl/chapter1` and you create a program called `runme` in that directory, you could run it like this:

```
$ ./runme
$ /Users/ovid/wroxperl/chapter1/runme
```

For now, you can just type **`perl programname`** to run the programs.

The shebang line might take one of a number of different forms. On a Linux system, this often looks like one of the following:

```
#!/usr/bin/perl
#!/usr/local/bin/perl
#!/usr/bin/perl -w
#!/usr/bin/env perl
```

The first two lines point directly to the Perl executable that should run the program. The third line, with the `-w` switch, tells Perl to run the program with global warnings. The final line tells Perl to use

the `env` program to find out which `perl` is currently set as the default `perl` for your system. This is useful if you have different versions of Perl installed and want your program to always run with the Perl you're currently using.

Some people just do the following:

```
#!/perl
```

And that generally does what you want.

On Windows you might see the following:

```
#!C:\Perl\bin\perl.exe
#!C:\strawberry\perl\bin\perl.exe
```

The first line is often found when running with ActiveState Perl. The line version is found when running with Strawberry Perl.

When `perl` sees the shebang line, it attempts to run your program using whatever it finds after the `#!`. Generally, this isn't a problem, but if you want to run the script on more than one machine, even with the same architecture, you could have a problem if someone installs Perl in a different location.

Fortunately, there is one simple trick you can follow to ensure you don't have problems with shebang lines: Don't install modules and scripts by hand. Instead, package them as proper distributions and install them with the standard Perl toolchain (such as `cpan` or `cpanm`). You learn module installation in Chapter 2 and module writing in Chapter 11.

For the Perl code that can be downloaded with this book, you will not be using shebang lines because they tend not to be portable. You will need to run the programs by explicitly typing **`perl` *`programname`***.

SUMMARY

By this time you've learned a bit about the history of Perl, where to go to get more information, installing Perl, and running a simple Perl program. This isn't a huge amount of information, but it's the foundation you need to progress in Perl.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
History	The basic history of the Perl language, its releases, and common use.
Getting Perl	About system Perl and `perlbrew` for those who use UNIX-style systems. Cygwin, ActivePerl, and Strawberry Perl are compared as options for Windows users.
Community	Perlmonks, IRC, Perl Mongers, and StackOverflow are all valuable resources for learning Perl.
`perldoc`	Perl comes with extensive documentation. You learned the basic structure of the docs and how to look up basic information.
Using a terminal	You use a terminal extensively when programming Perl. You learned how to launch a terminal and run a program from the command line.