前言

歡迎來到《*Perl 學習手冊*》第七版的天地，本書的內容已針對 Perl 5.24 以及它的最新功能最做過更新。如果你還在使用 Perl 5.8（但這是多年前發行的版本；你有升級的打算嗎？）也適合閱讀本書。

我們希望你在購買本書之前先閱讀這篇序言，因為歷史上的一個小小問題可能會引起一些混亂。還有另一個語言，Perl 6，最初係做為 Perl 5 的替代品，但後來卻在沒有更名的情況下獨自發行了。在我撰寫本文當時，Perl 5 可能是你想要使用的版本。當人們提到 Perl 時，他們所說的是被廣泛安裝和使用的程式語言。如果你不知道這一段為什麼會在這裡，因為這就是你想要知道的內容。

假如你正在尋找以 30 到 45 小時來學習 Perl 的最佳方式，那麼你已經找到了！在後面三百多頁裡，你將會看到我們精心安排的入門指引，介紹這個在 Internet（網際網路）之上擔負重任的程式語言；它也是最受全世界系統管理員、網站黑客及業餘程式員青睞的程式語言。這本書是根據我們所教的面授課程來設計的，所以我們將本書設定為一週的時間。

我們不能只花幾小時，就把 Perl 全部傳授給你。這樣保證的書大概都撒了一點謊。相對地，我們審慎選取了 Perl 之中實用的部分供你學習。這些材料足以讓你撰寫出 128 列以內的程式，通常 90% 的 Perl 程式都不會超過這個規模。當你準備做更深入的探討時，建議您翻翻《*Intermediate Perl*》，該書涵蓋了許多本書捨去不講的部分。此外，我們也納入了數個可供鑽研的切入點。

每一章的份量並不多，可在一兩個小時之內讀完。各章後面都有一系列的習題，可協助你練習剛學到的內容，附錄 A 還附有解答供你對照。因此，本書非常適合做為「Perl 入門」的課堂教材。我們對此有第一手的經驗，因為本書的內容幾乎是逐字逐句從「Learning Perl」課程的教材摘取來的；它是我們教過上千位學生的招牌課程。話雖如此，我們也將本書設計成適合自修閱讀。

雖然 Perl 是活生生的「Unix 工具箱」，但你並不需要是一個 Unix 大師，甚至也不必懂 Unix 就可以閱讀本書。除非特別註明，否則我們所提到的一切，都同樣可以應用到 ActiveState 的 Windows ActivePerl（*http://www.activestate.com/activeperl*）、Strawberry Perl（*http://www.strawberryperl.com/*），以及許許多多最新的其他 Perl 實作。

閱讀本書之前，雖然不需事先具備任何 Perl 的基礎，但我們還是衷心建議你先熟悉一些寫程式的基本概念，像是變數（variable）、迴圈（loop）、副常式（subroutine）和陣列（array），以及最重要的「用你最熟悉的文字編輯器來編輯原始碼」。我們不會浪費時間來說明這些概念。有些人生平所學的頭一個程式語言就是 Perl，並以本書學習成功，我們相當高興有這樣的例子，但是我們並不敢保證每個人都能有一樣的成果。

本書編排慣例

本書具有如下的字型慣例：

定寬字（`Constant width`）

套用於方法名稱（method name）、函式名稱（function name）、變數（variable）、屬性（attribute）以及程式範例。

定寬粗體字（**`Constant width bold`**）

套用於使用者所輸入的內容。

定寬斜體字（*`Constant width italic`*）

套用於程式碼中可被替換的項目（例如：*`filename`*，你可以將它代換成實際的檔名）。

斜體字（*Italic*）

套用於內文所提到的檔案名稱、網址（URL）、主機名稱以及命令。

第一章

簡介

歡迎蒞臨「駱馬書」（Llama book），這是我們對涉及 Perl 5 之書籍的親切稱呼！

本書從 1993 年迄今（第七版）已擁有超過百萬的讀者。我們最大的期盼是讀者喜歡本書的內容。可以確定的是，寫作當時我們是樂在其中的。至少，這是我們交出稿件，等了幾個月（在網路上）看到本書上架後的心情。

這是 Perl 6 發行後，我們最受歡迎之 Perl 5 書籍的第一個版本。Perl 6 以 Perl 為基礎開始了他的生命，但他現在過著自己的生活。不幸的是，這段歷史意味著，這兩個語言的名稱中都會有「Perl」，儘管它們之間只有輕微的關係。我們的設定是，你想要使用 Perl 5，所以你購買了這本書，除非你有其他想法。因此，本書中，Perl 意味著 Perl 5，也就是已經存在了二十年的 Perl。

答客問

或許你有一些關於 Perl 的問題，如果你已經快速翻閱過本書，大略知道了書中的內容，也可能會有一些關於本書的問題。現在，我們打算利用這一章來予以回應。

這本書適合你嗎？

這不是一本參考書，而是一本介紹 Perl 基礎知識的教材，本書的材料足以讓你建立出可供你自己使用的簡單程式。我們不會探討每個主題的每個細節，而會把幾個重要的主題分散在各章，這樣你就可以按照自己的需求選擇是否閱讀它們。

我們將讀者設定為懂得一點點程式設計，而且只需要學習 Perl 的人。我們會假設你至少會使用終端機、編輯檔案以及執行程式——不只是 Perl 程式。而且你已經知道變數和副常式之類的東西，但是你只需要瞭解 Perl 的運作方式。

這並不意味著，從未碰過終端機程式或撰寫過一列程式碼的初學者將會完全不知所云。首次閱讀本書的時候，你可能無法領會我們所說的一切，但是許多初學者在使用本書的時候所遇到的挫折並不大。訣竅在於不要擔心你可能會遺漏什麼，你只需要把焦點擺在我們所介紹的核心概念上。相較於有經驗的程式員，你可能需要花比較長的時間，但你必須從某處開始。

我們會假設你對 Unicode 已經有一點的瞭解，所以我們不會陷入細節之中，但我們在附錄 C 中做了一些說明。你可以在著手閱讀本書之前，先看一下附錄 C，以及在有需要的時候參考它。

此外，本書第七版首次納入了實驗性功能的附錄（附錄 D）。自從上一個版本以來，有一些令人興奮的新功能正等著你來使用，但我們不會強迫你使用它們。如果可能的話，我們將會試著向你展示，如何以無聊的老方式來進行同樣令人驚奇的事情。

你應該不會只閱讀這本與 Perl 有關的書。它只是一個教材，並非無所不包。它可以帶著你朝正確的方向出發，因此當你準備妥當後，你可以繼續閱讀我們所寫的其他書籍：《*Intermediate Perl*》和《*Mastering Perl*》，以及 Perl 的權威參考書《*Perl* 程式設計》，也被稱為「駱駝書」（Camel book）。

此外，儘管本書的內容涵蓋 Perl 5.24，但它仍適用於 Perl 之前的版本，即使是 15 年前推出的 Perl 5.8。你可能會因而錯過一些很酷的新功能，但是你仍舊能夠學習如何使用 Perl 的基本功能。

如前所述，有另一個名為 Perl 6 的語言，但它並非 Perl 5 的後繼者。不幸的是，由於歷史的因素（原本 Perl 6 有可能取代 Perl 5，但事實上並沒有），它們共享相同的名稱。除非你確定，你想要學習的是 Perl 6（而不是因為 6 大於 5），否則這便是一本適合你閱讀的書。

關於習題和解答？

每一章的末尾都收錄有習題，因為我們曾以相同的教材教過上千名學生。我們知道透過練習來犯錯是最好的學習方式。所以我們精心設計了這些習題，讓你也有機會犯錯。

並不是我們希望你犯錯，而是你需要有這樣的機會。因為你將會在你的 Perl 編程生涯中犯大部分的錯誤，還不如現在先體驗會比較好。由於前車之鑑，閱讀本書時你所犯下的任何錯誤，當你在趕限期交付 Perl 程式時，就不會重蹈覆轍了。另外，如果出現錯誤，我們會以附錄 A 來協助你：裡頭有我們對各習題所提供的解答，以及一些相關的說明。當你做完這些習題之後，可以核對一下答案。

在你努力嘗試克服問題之前，儘量先不要偷瞄答案。你自己完成習題，會學得比直接看答案來得好。就算一直想不出答案，也不要老是用頭撞牆。不用那麼在意，就先翻到下一章吧。

即使你沒有犯任何錯誤，在你做完習題之後也應該看一下解答；其中的文字會指出一些乍看之下或許並不明顯的重要細節。

當你看我們的解答時，請注意，你可以用不同的方式來完成習題，而且仍然是正確的。你不必像我們那樣做。某些情況下，我們會有多個解答。不僅如此，當你讀完本書之後，你可能會以不同的方式來完成相同的任務，因為我們的答案僅限於我們目前已介紹過的概念。稍後所介紹的其他功能或許可以縮短解決問題的時間。

每個習題之前都會有一個被方括號括起來的數字，看起來像這樣：

- [37] 當被方括號括起來的數字 37 出現在習題本文之前時，是什麼意思？

該數字是我們（非常粗略地）估計你在這個部分的練習，需要花幾分鐘的時間。那只是非常概略的估算，所以如果你已經全部完成（包括撰寫、測試和除錯）而只用了一半的時間，或是花了兩倍的時間還沒完成，都請不要太訝異。另一方面，如果你真的被難倒了，我們不會告訴別人，你的答案是偷瞄附錄 A 來的。

如果你需要其他的習題，可購買歐萊禮出版的《*Learning Perl Student Workbook*》，其中為本書的每一章提供了額外的習題。

如果我是 Perl 課程的講師？

若你是 Perl 課程的講師，想要以本書做為教材（多年來有許多人就是這樣做的），請留意我們對各章習題的設計，是儘量讓大部分的學生能在 45 分鐘到 1 小時之內完成，再留下一些休息的時間。有些章節的習題需要花的時間會少一些，而有些章節則需要多一些時間。會出現這種情況，是因為在填完方括號裡的數字後，我們發現自己竟然不會算加法（好在我們還知道要怎麼讓電腦幫我們做這件事）。

正如稍早所提到的，我們為本書出版了《*Learning Perl Student Workbook*》，其中為本書的每一章提供了額外的習題。如果你購買的是搭配本書之前版本的 workbook（作業簿），你應該依需要調整各章的次序。

「Perl」這個字是什麼意思？

Perl 有時被稱為「實用析取與報告語言」（Practical Extraction and Report Language），但它同時也被稱為「病態折衷式廢話製表器」（Pathologically Eclectic Rubbish Lister），還有其他不同的全名展開方式。Perl 是一個溯寫字（backronym）而非縮寫字（acronym）—— Perl 之父，Larry Wall，先想到 Perl 字，後來又提出展開的全名。這也就是為什麼「Perl」不會全部用大寫來表示的原因。我們無須爭辯這兩種全名哪一個才對：它們都受到 Larry 的認可。

你可能會在某些文章裡看到以小寫 p 來表示「perl」。一般說來，大寫 P 所表示的「Perl」指的是程式語言，而小寫 p 所表示的「perl」指的是翻譯和執行程式的直譯器。

Larry 為什麼要創造 Perl ？

1980 年代中期，Larry 試圖為一個瑕疵回報（bug-reporting）系統，從新聞群組式（Usenet-news-like）的檔案階層來產生報表；這超出了 awk 的能力範圍。身為懶惰的程式員，Larry 決定寫一支通用的工具，讓它不僅能解決這個問題，還能在別的地方使用。Perl 第零版於焉誕生。

我們說 Larry 懶惰，並不是在說他的壞話；懶惰其實是一種美德。正如 Larry 在《*Programming perl*》第一版中所寫的，不耐煩和傲慢也是如此。手推車是由懶得扛東西的人發明的；書寫是由懶得記憶的人發明的；Perl 是由若不創造一個全新的程式語言，就懶得把事情搞定的人發明的。

Larry 為什麼不用其他的語言就好了？

世界上不缺程式語言，不是嗎？但是在當時，Larry 卻找不到任何符合其需求的語言。如果時下的某種語言，在當年就能夠出現的話，Larry 或許就會使用它了。他當時需要的是，像 shell 或 *awk* 一樣能夠快速撰寫的語言，又具有和 *grep*、*cut*、*sort*、*sed* 類似的進階功能，而又不必回頭去使用 C 之類的語言。

Perl 試圖填補低階程式設計（如 C、C++ 或組合語言）和高階程式設計（如「shell」程式設計）之間的空隙。低階程式設計通常既難寫又難看，但是執行速度很快，而且不受限制；在任何機器上，都很難贏過寫得好之低階程式的執行速度。它們差不多什麼事都可以做。高階程式設計是另一個極端，它們通常速度緩慢、既難寫又難看，並且限制重重；如果系統上不提供執行必要功能的某個命令，你的 shell 或 batch 程式設計有很多事情都不能做。而 Perl 則相當容易、幾乎不受限制、速度通常很快，也有一點難看。

讓我們來看看上面這段話對 Perl 所做的四項描述：

首先，**Perl 很容易**。不過接下來你會發現，這說的是容易使用。學習 Perl 並不會特別容易。如果你會開車，你一定是花了好幾個星期或幾個月的時間來學習，最後開起來才會那麼容易。若你花在寫 Perl 程式的時間，和學開車所花的時間一樣多，那麼 Perl 對你而言就會很容易。

Perl 幾乎不受限制。幾乎沒什麼事是 Perl 辦不到的。雖然你大概不會想用 Perl 來撰寫「中斷 - 微核心 - 層次」（interrupt-microkernel-level）的裝置驅動程式（不過的確有人這麼做過），但一般人用來處理日常瑣事的程式，從用完即丟的小程式到產業級的大型應用程式，都很適合用 Perl 來撰寫。

Perl 的速度通常很快。那是因為 Perl 的開發者，同時也都是使用者——所以我們都希望它快。假設有人為 Perl 加上某個很酷的功能，可是會讓其他程式變慢，那麼 Perl 的開發者幾乎一定會拒絕加入這一項新功能，直到我們找出讓它變快的方法為止。

Perl 有一點醜陋；這是事實。Perl 的象徵是駱駝；來自於值得尊敬之「駱駝書」（也就是《*Perl 程式設計*》）的封面，而本「駱馬書」（以及姐妹作「羊駝書」）算是該書的表親。駱駝長得也有一點醜陋。但是牠們賣力工作，即使是在嚴酷的環境下也一樣。駱駝能在種種不利的條件下幫你把事情搞定，即使它們長相不美，而且氣味更糟，有時候還會對你吐口水。Perl 就有一點像這樣。

Perl 是容易還是難？

Perl 容易使用，但有時候不太好學。當然，這只是泛論而已。不過在 Larry 設計 Perl 的當時，他必須做出許多取捨。每當他有機會讓程式員感到方便、但是讓學 Perl 的人覺得不便時，他幾乎一定站在程式員這一邊。那是因為你只需要學一次 Perl，但卻會一直使用它。

如果你每週或每個月只花幾分鐘在程式設計上，容易學習的語言會比較合適，因為在你下一次使用時，可能就忘光了。Perl 是為每天至少花二十分鐘寫程式（而且以 Perl 程式為主）的程式員而設計的。

Perl 有不少捷徑，可以讓程式員節省時間。舉例來說，大部分的函式都具有預設行為；它通常就是你使用該函式時想採取的方式。因此，你會看到如下的 Perl 程式碼：

```
while (<>) {
  chomp;
  print join("\t", (split /:/)[0, 2, 1, 5] ), "\n";
}
```

現在還不用擔心這段程式碼的意思。如果不使用 Perl 的預設行為與簡寫，上面這段程式大概會增長十倍或十二倍；這樣一來，閱讀與撰寫的時間也會大幅增加。它會用到更多的變數，讓維護和除錯也變得比較困難。如果你已經看得懂一點 Perl，你將發現上面的程式裡沒有變數，這也是重點之一。其所用到的變數，都是以預設行為來達成。但是，讓程式員容易行事的這些功能，是要在學習時付出代價的；你得先學會這些預設行為與簡寫才行。

就好比說，在英語裡，大家常會使用縮寫，而不會覺得不妥。沒錯，「will not」跟「won't」這兩種寫法的意義一模一樣。但是大家都會說「won't」，而比較不會說「will not」。一來因為比較省時間，二來因為大家都熟悉這樣的模式，而且不無道理。同樣地，Perl 的「簡寫」縮寫了常用的「片語」，好讓維護人員能夠快速地「講」Perl，以及瞭解 Perl。

一旦熟悉 Perl 之後，你可以花較少的時間來搞定，如何正確使用 shell 的引號（或 C 語言的宣告），而花較多的時間來瀏覽網站；這是因為 Perl 能讓你事半功倍。Perl 簡明的語法，讓你能夠（毫不費力地）建立很酷而且略勝一籌的解決方案，或是用途廣泛的工具程式。由於 Perl 既跨平台又隨處可用，所以這一次所做的工具可以在下一次的任務裡沿用，讓你有更多的時間瀏覽網站、充實自己。

Perl 是非常高階的語言。這表示程式碼的密度相當高；Perl 程式碼的長度，大約是等效之 C 程式碼的 1/4 到 3/4 左右。這使得 Perl 程式碼的撰寫、閱讀、除錯以及維護所需要的時間會比較少。只需要寫過一點程式，便會發現當整個副常式小到能夠整個放進螢幕時，不需要來回捲動程式碼，你就可以知道目前的情況如何。此外，既然程式裡瑕疵（bug）的數量約與程式長度成正比（而不是與程式的功能成正比），較短的 Perl 程式碼，平均起來會含有較少的瑕疵。

像任何語言一樣，Perl 也可以變成「防讀保護」（write-only）——它可以寫出讓人完全看不懂的程式。但只要你稍用一點心，就可以避免這項常見的污名。沒錯，Perl 程式對門外漢來說，看起來可能像線路雜訊；但是對經驗老道的 Perl 程式員來說，它看來就是大交響樂團的總譜。你只要遵照本書的指引，就能寫出易於閱讀及維護的程式；它們大概贏不了 Perl 迷津程式大賽（Obfuscated Perl Contest）。

Perl 怎麼會有這麼多人在用？

Larry 只稍微玩一下 Perl，並在它各處加一點功能之後，就把它釋出給 Usenet 的讀者社群，也就是一般所謂的「網路」（the Net）。這群散居世界各處的（上萬名）使用者給了 Larry 回應，要求 Perl 做這個、做那個，其中有許多事情是 Larry 從來沒有想到，要讓他的小 Perl 去處理的。

結果，Perl 不斷持續成長。它的功能變多了，能夠執行它的平台也增加了。當年這個只能在少數幾種 Unix 系統上執行的小語言，而今長成了具有上千頁的線上自由文件、成打的書藉、數個主流的 Usenet 新聞群組（以及成打的小眾新聞群組與郵遞論壇）、新聞群組裡無以數計的讀者，還有時下幾乎所有系統皆可使用的版本——當然，也不要忘了這一本「駱馬書」。

Perl 目前的發展如何？

Perl 5 的開發經歷了驚人的復興，而大多數人都在等待它的後繼者 Perl 6。現在它們實際上是不同的語言，然而 Perl 5 仍舊佳績不斷。不幸的是，它們仍然共享相同的名稱。

從 v5.10 開始，Perl 開發出了一種在該語言中包含新功能的方法，此方法不會干擾到舊有的程式。我們將會向你示範，如何在有需要時獲取這些新功能，以及如何啟用你想要使用的實驗性功能。

Perl 5 Porters 團隊還採行了一種版本支援策略。經過 20 年輕率的版本支援策略後，他們決定了支援最後兩個穩定版本的策略。在本書英文版完成當時，預期是 v5.22 和 v5.24。點號後面的奇數編號保留給開發版本使用。

自本書上一個版本發行以來，出現了一些令人興奮的功能（也消失了一些功能）。你必須繼續閱讀才能瞭解這些內容。

哪些事情最適合用 Perl 來做？

Perl 很適合用來在三分鐘內寫出急就章（quick-and-dirty）的程式。Perl 也很適合用來撰寫用處廣泛、需要一打程式員花三年時間完成的大型程式。當然，你將發現自己所撰寫的許多程式，從構思到完整測試，只需要花一小時以內的時間。

Perl 擅長處理「約有 90% 與文字處理有關、10% 與其他事務有關」的問題。這似乎佔了當前程式處理需求的絕大部分。在理想的世界中，每個程式員都知道每一種語言；進行每項專案時，他們都能夠選擇最適合的語言。通常，他們會選擇 Perl。

哪些事情不適合用 Perl 來做？

所以，既然 Perl 能做的事情這麼多，哪些事不適合它做呢？嗯，如果你想做出「不透明的二元執行檔」（opaque binary），請不要使用 Perl。這指的是，取得或購得你程式的人，無法看到你在原始碼中的祕密演算法，也因此無法協助你進行程式碼的維護或除錯。當你把 Perl 程式給某人時，通常給的是原始碼，而非不透明的二元執行檔。

不過，如果你需要不透明的二元執行檔，我們必須告訴你其實沒這種東西。只要某人能安裝並執行你的程式，他就能將它還原成原始碼。的確，該原始碼可能和原本的不一樣，但它總是某種原始碼。要保護你的祕密演算法，真正的辦法只有一種：聘僱足夠數量的律師；他們能寫出一份授權條款，上面說『你可以用這一支程式做這件事，但不能做那件事。要是你違反了我們的規定，我們有足夠數量的律師，能讓你後悔莫及。』

如何取得 Perl ？

你的機器上可能已經有 Perl 了。至少，我們不管到哪裡都找得到 Perl。許多系統都會隨附 Perl，而系統管理員經常會在單位裡的每台機器上安裝 Perl。不過，就算在系統上找不到 Perl，你還是可以免費取得它。大多數的 Linux 或 *BSD 系統、Mac OS X 以及其他系統都會預先安裝 Perl。有些公司，像是 ActiveState（*http://www.activestate.com*），會針對數種平台（包括 Windows）提供預先建構和增強過的發行套件。Windows 平台之上還有 Strawberry Perl（*http://www.strawberryperl.com*）可供取用，它其實就是一般的 Perl 額外加上用於編譯和安裝第三方模組的工具程式。

Perl 的授權條款有兩種。對只是使用 Perl 的大部分人來說，這兩種條款並沒有什麼差別。不過，如果你想修改 Perl，請詳細閱讀這兩份授權條款，上面對散佈更動過的 Perl 有些限制。對於不想修改 Perl 的人而言，授權條款基本上是說：「這是自由軟體——你愛怎麼用都行。」

事實上，它不只是自由軟體，還能在幾乎所有自稱為 Unix、具有 C 語言編譯器的系統上順利執行。你只要下載它，鍵入一或兩道命令，Perl 就能自行完成設定與安裝。或者，更好的是，你可以讓套件管理程式替你完成此事。除了 Unix 和類似 Unix 的系統，對 Perl 上癮的人還將它移植到了別的平台，像是 VMS、OS/2，甚至還包括 MS/DOS 以及 Windows 的每一個現代的版本——在你看到這句話時，可能又增加了更多平台。這些 Perl 的移植版本，經常會附上安裝程式，甚至比 Unix 的安裝程序更容易使用。請參考 CPAN 上的「ports」專區。

在 Unix 系統上，從原始碼來編譯 Perl 幾乎總是比較好的做法。其他的作業系統可能沒有 C 編譯器，所以 CPAN 為它們提供了二元執行檔。當你使用本地端的套件管理程式，你將會改變你的作業系統用於進行其任務的 *perl*。這可能會搞得一團亂。我們建議你僅安裝供你自己使用的 *perl*，但是對本書來說並非必須。

CPAN 是什麼？

CPAN 就是「Perl 綜合典藏網」（Comprehensive Perl Archive Network），它是 Perl 的一站式賣場。裡面有 Perl 本身的原始碼、各種非 Unix 系統的安裝程式、範例程式、說明文件、延伸模組，以及跟 Perl 相關的庫存新聞。簡言之，CPAN 確實是包羅萬象。

CPAN 有數百個映射站台，分佈在世界各地；你可以從 CPAN（*http://search.cpan.org/*）或 metacpan（*http://www.metacpan.org*）開始瀏覽或搜尋整個典藏網。

如果你使用的是 ActivePerl，ActiveState 公司提供有 PPM（全名為 Perl Package Manager）。他們已經使用相同的編譯設定完成各種模組的編譯，並允許你透過 ppm 工具來安裝它們。

Perl 有任何形式的支援嗎？

當然有！我們最喜歡的方式之一，就是 Perl 推廣組（Perl Mongers）。它是 Perl 使用者的全球性組織；相關資訊請參考 *http://www.pm.org/*。你附近應該就有個分部，可以找到專家或認識專家的人。如果附近沒有分部，你很容易就可以自己成立一個。當然，請不要忽略 Perl 的說明文件；它們提供了立即的支援。

除了說明文件，CPAN、metacpan（*http://www.metacpan.org*）以及其他提供有 Perl 文件的網站（*http://perldoc.perl.org*）也有文件可看，你也可以到 *http://faq.perl.org* 查閱最新版的 perlfaq（Perl 常見問題集）。

另一個權威性的資料來源，則是《*Perl 程式設計*》這一本書，因為該書封面動物的關係，它通常被稱作「駱駝書」（就像本書叫作「駱馬書」一樣）。「駱駝書」裡包含了完整的參考資料、一些教學範例，以及一堆與 Perl 相關的雜項資訊。另外還有一本口袋大小的 Perl 5 參考手冊（O'Reilly 出版），作者是 Johan Vromans，很適合拿在手上閱讀（或是放進口袋裡）。

如果想找人問問題，可以到眾多的郵遞論壇（mailing lists）提出——有許多列在 *http://lists.perl.org*。此外，也可以到 Perl Monastery（*http://www.perlmonks.org*）和 Stack Overflow（*http://www.stackoverflow.com*）提出問題。無論何時，會有一個 Perl 專家在某個時區醒著回答問題—— Perl 是一個日不落帝國。這表示在你提出問題的幾分鐘後，通常就會有人提供解答。如果你沒有先查過說明文件和常見問題集，幾分鐘之內就會挨罵。

你也可以看看 *http://learn.perl.org* 以及相關的郵遞論壇 *beginners@perl.org*。許多著名的 Perl 程式員也會定期在自己的部落格上張貼與 Perl 有關的文章，你可以透過 Perlsphere 網站（*http://perlsphere.net/*）找到這些部落格。

如果你需要簽一份 Perl 的支援服務合約，有好幾家公司會願意收你的錢。通常，其他免費的支援管道就綽綽有餘了。

如果發現 Perl 有瑕疵，該怎麼辦？

身為一個新的 Perl 程式員，你可能會造成一個你認為 Perl 有問題的情況。你使用的是一個你還不知道的大語言，所以你不知道誰應該為這種異常行為負責。這是常有的事。

當你發現 Perl 有瑕疵（bug），請再次檢視說明文件。有時還得反覆查個兩三次。當我們在查閱說明文件對某個異常行為的解釋時，經常會發現新的 Perl 小細節。它們最後都進了簡報或是雜誌的文章裡。Perl 裡有許多的特殊功能和例外，所以你遇到的可能是一個功能，而不是瑕疵。另外，請確定目前所安裝的是最新版的 Perl；也許你碰到的問題，在新的版本中已經修正好了。

一旦你 99% 確定自己找到的是真正的瑕疵，請問問周遭的人。你可以問同事、在當地的 Perl 推廣組集會時發問，或是在 Perl 會議上提出。它很可能仍舊是一個功能，而不是瑕疵。

一旦你 100% 確定自己找到的是真正的瑕疵，請寫一個測試案例（如果還沒有這樣做的話）。理想的測試案例（test case）是一支自給自足的小程式，任何 Perl 使用者在執行它時，都可以看到你所發現的（不當）行為。在你做出能夠清楚顯示瑕疵的測試案例後，

請利用工具程式 *perlbug*（隨附在 Perl 裡）來回報這個瑕疵。這通常會寄出一份電子郵件給 Perl 的開發團隊，所以在做出測試案例之前請勿使用 *perlbug*。

如果沒出什麼差錯，一般在你送出瑕疵回報後幾分鐘之內就會有回應。通常，你可以使用一個簡單的修正檔（patch）來修正此瑕疵。當然，在最壞的狀況下，也許不會有人回應；Perl 的開發團隊對你並沒有義務，甚至不一定非讀你的瑕疵報告不可。但因為我們全都是 Perl 的愛好者，因此不會讓瑕疵逃過眾人的法眼。

我該如何撰寫 Perl 程式？

差不多是問這個問題的時候了（即使你還沒問）。Perl 程式是一個純文字檔；你可以用自己喜歡的文字編輯器來建立與修改它。你不需要使用任何特殊的開發環境，雖然也有廠商提供這種商用軟體。我們使用這類軟體的經驗不足無法推薦它們（但使用得夠久了，因此不想再使用它們）。此外，開發環境的使用其實是個人主觀的選擇。若你去詢問三個程式員，應該使用何種開發環境，你將會得到八種答案。

一般來說，你應該使用程式員專用的文字編輯器，而不是普通的編輯軟體。兩者有什麼不同呢？嗯，程式員專用的文字編輯器，能讓你進行撰寫程式時常用的操作，例如調整一塊程式碼的縮排，或是尋找成對的左右大括號。

在 Unix 系統下，最受歡迎的兩套程式員專用編輯器是 *emacs* 和 *vi*（以及它們的衍生版本）。BBEdit、TextMate 和 Sublime Text 則是 Mac OS X 系統上不錯的編輯器，在 Windows 上則有很多人推薦 UltraEdit、SciTE、Komodo Edit 與 PFE（Programmer's File Editor）。在 perlfaq3 文件中也列出了一些其他的編輯器。如果你不知道你的系統上有哪些文字編輯器可用，請就近找專家幫忙。

本書習題所需要撰寫的簡單程式，長度都不會超過 20 或 30 列，所以用哪一種文字編輯器其實都沒問題。

少數初學者會嘗試使用文書處理器（word processor），而不使用文字編輯器（text editor）。我們不建議這種做法——你可能會很不方便，或是根本無法使用。但是，我們不會阻止你。請讓文書處理器以「純文字」（text only）格式來儲存檔案；文書處理器的內定格式幾乎一定不能使用。幾乎所有的文書處理器都會提醒你，你的 Perl 程式拼字錯誤，並且過度使用分號。

某些情況下，你可能需要在一台機器上撰寫程式，再傳送到另一台機器上執行。這個時候，請使用“text”（文字）模式或“ASCII”模式來傳送程式，而不要使用“binary”

（二進制）模式。這是因為在不同的機器上，文字的格式也不一樣。如果不用文字模式傳送，執行的結果可能會有所不同——某些版本的 Perl 在偵測到換列方式（line ending）不一致時，甚至還會中斷執行。

一支簡單的程式

按照歷史悠久的慣例，任何根植於 Unix 文化的程式語言入門書，都會以印出「Hello, World」的程式做為開頭。所以，底下就是它在 Perl 裡的寫法：

```
#!/usr/bin/perl
print "Hello, world!\n";
```

假設你已經把上面那兩列程式碼鍵入文字編輯器。（暫且別管程式各部分是什麼意思。我們很快就會討論到。）一般來說，程式可以存成任何檔名。Perl 程式並不需要使用任何特殊的檔名或副檔名，而且通常不用副檔名比較好。不過，某些系統上，可能需要使用 *.plx*（意指 PerL eXecutable）之類的副檔名。

接下來，你可能還需要告訴系統，這檔案是一個可執行的程式（也就是一個命令）。在不同的系統上，需要的操作方式也會有所不同；也許只需要把程式檔儲存到某個地方就行了。（通常你的當前工作目錄就可以了。）在 Unix 系統上，你可以使用 *chmod* 命令，把程式標記成可執行，就像這樣：

```
$ chmod a+x my_program
```

其中，最前面的錢號（以及空白）代表命令列提示符號（shell prompt），在你的系統上或許會長得不太一樣。如果你習慣用 `755` 之類的數值，而不是 `a+x` 這樣的符號，來代表權限，那當然也沒有問題。不管使用哪種寫法，它都能告訴系統，這個檔案現在已經是一支程式了。

現在，你可以這樣執行它：

```
$ ./my_program
```

此命令開頭的點號（.）與斜線（/），表示要在當前工作目錄（current working directory）中尋找這個程式，而不是透過 `PATH` 來尋找。你不一定每次都得使用它們，但是在完全瞭解它們的用處之前，每次執行命令的時候，還是都加上去比較好。

你還可以透過明確指定 *perl* 的方式來執行此程式。如果你在 Windows 上，則必須在命令列之上指定 *perl*，因為 Windows 並不會猜測你想要執行的是什麼程式：

```
C:\> perl my_program
```

如果一切正常，那可真是個奇蹟。通常，你會發現程式裡有瑕疵。這時只要修改程式，再試一次就好了——但是不必每次都使用 *chmod*，因為第一次的設定應該會「附著」在檔案上。（當然，如果問題其實是 *chmod* 沒設對，那麼你的 shell 可能會顯示「permission denied」的錯誤訊息。）

在 Perl 5.10 或之後的版本中，這個簡單的程式可以有另一種寫法；也就是使用 `say`，而不是使用 `print`。`say` 如同 `print`，但可以少打點字。`say` 會自動替我們添加換列符（newline）；這意味著，若我們忘了加上換列符，它可以讓我們節省一點時間。因為它是一個新功能，所以我們會透過 `use v5.10` 這列陳述（statement）告訴 Perl，我們想要使用新功能：

```
#!/usr/bin/perl
use v5.10;
say "Hello World!";
```

此程式只能在 v5.10 或之後的版本下執行。本書中，當我們在介紹 v5.10 或之後版本的功能時，我們將在內文中明確指出這些是新功能，提醒你應該要使用 `use v5.10` 這列陳述。

通常，Perl 的最早期版本就可以提供我們需要用到的功能。因為本書的內容涵蓋 v5.24，所以當你用到了〈前言〉所提到的新功能時，別忘了加入如下的陳述：

```
use v5.24;
```

我們也可以在不使用 v 的情況下來指定這個版本需求，但是別忘了此時的次要編號（minor number）是三位數字：

```
use 5.024;
```

不過，本書採用的是使用 v 的形式。

程式裡寫的是什麼？

就像其他「形式自由」（free-form）的語言，Perl 通常可以隨意加上無關緊要的空白符（像是空格、跳格與換列符）讓程式碼更容易閱讀。不過，大部分 Perl 程式的格式都相當統一，就和本書中程式的格式相似。perlstyle 文件對此有提供一般的建議（並非必須遵守的規則！）。我們十分鼓勵你為你的程式進行適當的縮排，因為縮排能讓程式變得較好閱讀；好的文字編輯器會幫忙處理此事。適當的註解也有助於程式的閱讀。Perl 裡的註解，是從井字號（#）開始，到列尾結束。

Perl 裡沒有「區塊式註解」（block comment），不過，有許多模擬的方法。請參考 perlfaq 文件。

本書中的程式沒有用到太多的註解，因為前後的正文已經把它們的功能解釋得很清楚了，但是在你的程式裡，請儘量在需要時加上註解。

所以，同樣的「Hello, World」程式，也可以寫成這樣（我們必須說，這是一種相當奇怪的寫法）：

```
#!/usr/bin/perl
    print    # 這是一個註解
"Hello, world!\n"
;    # 請不要把 Perl 程式寫成這樣！
```

此程式碼的第一列，其實是一個具有特殊意義的註解。在 Unix 系統上，如果文字檔開頭的兩個字符是「#!」（發音為“sh-bang”或 SHəˈbaNG），則後面跟著的是用來執行這個檔案之程式的名稱。此範例中，該程式就存放在 */usr/bin/perl* 檔案中。

事實上，Perl 程式裡最缺乏可移植性的就是 #! 那一列了，因為你必須找出它在每台機器上分別要設成什麼。幸好，它幾乎總是 */usr/bin/perl* 或 */usr/local/bin/perl* 兩者之一。如果發現 Perl 沒有安裝在這兩處，你就得找找看系統上的 Perl 到底藏在什麼路徑，然後改用該路徑。在一些 Unix 系統上，或許可以寫成這樣，讓系統自動幫你找：

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

如果你在搜尋路徑中的任何目錄底下都找不到 *perl*，那麼可能就得就近請教一下你當地的系統管理員，或是使用同樣系統的朋友了。但請注意，儘管你在第一時間找到了 *perl*，不過它可能不是你想要的。

在 Unix 以外的系統上，傳統上第一列會寫成 #!perl（這也有其實用價值）。至少，它能讓維護人員在著手修正程式時，馬上知道這是一支 Perl 程式。

要是 #! 這一列設錯了，shell 通常會發出錯誤訊息。它的內容可能出乎意料，像是「File not found」（找不到檔案）或是「bad interpreter」（直譯器不良）。它並非找不到你的程式，而是找不到 */usr/bin/perl*。我們很想把這個訊息改得清楚一點，但它不是 Perl 發出的；發出抱怨的是 shell。

你還可能碰上的另一個問題，就是你的系統完全不支援 #! 列。這樣一來，你的 shell（或你的系統所使用的其他機制）也許會直接執行程式的內容，但是出來的結果大概會令人失望或感到意外。要是你搞不清某些奇怪的錯誤訊息在講什麼，可以到 perldiag 文件裡搜尋。

所謂的「主」程式（"main" program）完全由一般的 Perl 陳述（statement）組成（副常式裡的除外，這我們稍後再談）。和 C 或 Java 之類的語言不同，Perl 沒有「主」常式（"main" routine）。事實上，許多程式裡根本就沒有常式（副常式的形式）。

另外，程式裡也不必有變數宣告區段，這一點和某些其他的語言不同。如果你在過去都必須宣告所有變數，現在可能會有一點不安。不過，這個特性讓我們得以撰寫急就章（quick-and-dirty）的 Perl 程式。如果程式的長度只有兩列，將其中一列耗費在宣告變數上，似乎不太值得。如果你真的想宣告變數的話，那也不是壞事；我們在第 4 章〈副常式〉裡會討論怎麼做。

大部分的陳述，都是運算式（expression）後面接著一個分號。底下的陳述，我們已經看過好幾次了：

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

你只需要使用分號（;）來隔開陳述，而不需要終止它們。如果後面已經沒有陳述了（或者它是作用域中最後一個陳述）則可以不使用分號：

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

現在你大概已經猜到了，上面這列會印出「Hello, World!」訊息。接在訊息後面的是簡寫 \n，其含意對 C、C++ 或 Java 之類語言的使用者應該不陌生；它代表「換列符」（newline character）。當它被列印在訊息後面時，游標位置會從訊息的末端，移到下一列的開頭，這讓隨後出現的命令列提示符號得以自成一列，而不會緊貼在訊息之後。輸出結果的每一列都應該以一個換列符做結尾。下一章，我們將會看到更多關於「換列符」簡寫以及「倒斜線轉譯」（backslash escape）的介紹。

如何編譯我的 Perl 程式？

直接執行你的 Perl 程式就行了。對於使用者而言，只要一個步驟，perl 直譯器就能完成編譯和執行這兩個動作：

```
$ perl my_program
```

當你執行程式時，Perl 內部的編譯器會先處理過整個原始碼，將之轉換成內部的 bytecode（Perl 在內部用來表示程式的資料結構），再交由 Perl 的 bytecode 引擎來執行。所以，如果在第 200 列有一個語法錯誤，錯誤訊息在程式開始執行第二列之前就會出現。如果你的程式碼中出現了一個執行 5,000 次的迴圈，它只會被編譯一次；實際的迴圈會以最快的速度執行。除此之外，不論你用了多少註解和空白來讓程式變得容易閱讀，都不會影響「執行階段」（runtime）的速度。你甚至可以使用完全由常數（constant）組成的計算式，這樣它的值只會在程式開始時計算一次——而不會每一個迴圈都再計算一次。

不用說，編譯是要花時間的——撰寫一支連篇累牘的 Perl 程式，若只是為了完成一件簡單的小事（而不是一次做好幾件事），實在沒什麼效率，因為花在編譯上的時間會比執行階段還要長。但是 Perl 編譯器的執行速度非常快；通常編譯時間只占執行階段的極少百分比而已。

不過，有一個例外的狀況，如果你所撰寫的是一支透過 Web 執行的 CGI 命令稿，那麼它每分鐘可能會被叫用成千上百次。（這樣的使用頻率非常高；如果是每天被叫用成千上百次，就像在 Web 上執行的大部分程式那樣，或許就沒什麼好擔心了。）許多這類的程式，執行階段都很短，所以重複編譯所造成的問題，可能會比較明顯。如果你有這方面的問題，你必須在兩次執行之間，將程式保留在記憶體中。Apache web 伺服器的 `mod_perl` 延伸套件（*http://perl.apache.org*）與 `CGI::Fast` 之類的 Perl 模組都可以幫你解決這方面的問題。

若是把編譯過的 bytecode 儲存起來，能否避免編譯的開銷？或者更好的是，能否將 bytecode 轉換成另一個語言（例如 C 語言），再進行編譯？以上兩種做法都可行，但卻幾乎不會讓程式更容易使用、維護、除錯或安裝，甚至還會讓程式跑得更慢。

走馬看花一番

閱讀到這裡，你一定想看看真正有料的 Perl 程式長得什麼樣子？（若是你其實不想，麻煩暫時合作一下。）請看：

```
#!/usr/bin/perl
@lines = `perldoc -u -f atan2`;
foreach (@lines) {
  s/\w<([^>]+)>/\U$1/g;
  print;
}
```

如果無法使用 *perldoc*，通常是因為系統不支援命令列介面，或者你的特殊系統將其包含在不同的套件中。

現在，這樣的 Perl 程式碼，乍看之下可能有一點怪異。（事實上，每當你遇到這樣的 Perl 程式碼時，看起來可能都滿奇怪的。）不過，我們會逐列講解這一支範例程式，看看它到底做了些什麼。底下的介紹會非常簡略；畢竟這一節只是「走馬看花一番」。此程式裡用到的所有功能，在後面的章節中都會詳加說明。你不需要現在就把整支程式搞懂。

第一列就是我們之前介紹過的 `#!` 列。你也許需要依照前面所提到的步驟，把這一列改成系統能夠接受的樣子。

第二列使用了一對倒引號（`` ` `）來執行外部的命令。（倒引號的按鍵在全尺寸的美式鍵盤上，通常會放在數字 1 的左邊。請小心，別把倒引號和單引號「'」搞混了。）我們所使用的命令是 *perldoc -u -f atan2*；請在命令列上鍵入這道命令，試試看它會印出什麼訊息。在大部分的系統上，一般都可以使用 *perldoc* 這個命令，來閱讀和顯示 Perl 以及其延伸套件和工具程式的文件。這道命令執行時，會顯示關於三角函式 `atan2` 的一些資訊；在這裡，我們只是用它來示範，如何處理外部命令的輸出結果。

當倒引號裡的命令執行完畢後，輸出結果會一列列依序儲存在稱為 `@lines` 的陣列變數裡。下一列程式碼會開始一個迴圈，依序對每一列資料進行處理。迴圈裡的陳述經過縮排。雖然 Perl 並不會要求你這麼做，但好的程式員都會如此要求自己。

迴圈裡的第一列看起來最恐怖：`s/\w<([^>]+)>/\U$1/g;`。此處不會深入其細節；它的意思是，更改任何內含一對角括號（`< >`）標記的資料列，而在 *perldoc* 命令的輸出結果裡，應該至少有一列資料符合此條件。

迴圈裡的第二列，出乎眾人意料地，會把每一列資料的內容（有可能已遭到修改）印出來。最後的輸出結果，看起來應該和 *perldoc -u -f atan2* 的執行結果差不多，但是其中出現角括號標記的地方都會遭到修改。

因此，在短短數列程式碼中，我們執行了另一支程式、把它的輸出結果存進記憶體、更新記憶體裡的資料項，之後再列印出來。這種在資料格式之間進行轉換的程式，就是 Perl 常見的用途之一。

習題

一般來說，每章結束時都會有幾道習題，解答在附錄 A 可以找到。不過本章的習題答案，其實在前面都已經出現過了。

如果這些習題的解答在你的機器上無法運作，請先詳細檢查，再就近詢問專家。請注意，你可能需要照著前面所提到的方法，稍微修改每個程式的內容。

1. [7] 鍵入前面的「Hello, world」程式，想辦法讓它跑起來！程式檔可以取任何名稱，但是像 *ex1-1* 這樣簡單的名稱比較好，因為它代表著第 1 章的第一道習題。這是一支即使是有經驗的程式員都會寫的程式，主要用於測試系統的設置是否妥當。若你可以執行這支程式，代表你的 *perl* 已設置妥當。

2. [5] 在命令列提示符號之後鍵入 *perldoc -u -f atan2* 這一道命令，並記下它的輸出結果。如果無法執行，請就近詢問專家或參考你的 Perl 版本的文件，以找出如何調用 *perldoc* 或它的替代品。無論如何，下一道題目必須用到 *perldoc* 。

3. [6] 鍵入第二支範例程式（參閱上一節），看看它會印出什麼。提示：請仔細照著書上的標點符號鍵入，不要打錯了！你看得出來，它是如何改變 *perldoc* 命令的輸出結果嗎？