第1章

背景知识

本章将简述UNIX系统的发展史。了解UNIX在何处开发、如何开发，以及它的设计动机。这有助于用户善加利用UNIX所提供的工具。此外，本章将介绍软件工具的设计原则。

1.1 UNIX简史

或许你对于UNIX的发展史已有些了解，并且已经有很多介绍UNIX完整发展历史的资料。这里，我们只想让你知道：UNIX是在何种环境下诞生的，以及它如何影响软件工具的设计。

UNIX最初是由贝尔电话实验室（Bell Telephone Laboratories，注1）的计算机科学研究中心（Computing Sciences Research Center）开发的。第一版诞生于1970年——也就是在贝尔实验室（Bell Labs）退出Multics项目不久。在UNIX广受欢迎的功能中，有许多便是来自Multics操作系统。其中最著名的有：将设备视为文件，以及特意不将命令解释器（command interpreter）或Shell整合到操作系统中。更完整的历史信息可在*http://www.bell-labs.com/history/unix*找到。

由于UNIX是在面向研究的环境下开发的，因而没有必须生产或销售成品的盈利压力。这使其具有下列优势：

- 系统由用户自行开发。他们使用这套系统来解决每天所遇到的计算问题。

注1：该名称至今已变更数次。本书从这里开始都以口语式名称“贝尔实验室”(Bell Labs)称呼它。

- 研究人员可以不受拘束地进行实验，必要时也可任意变换程序。由于用户群不大，若程序有必要整个重写，多半也不会太难。由于用户即为开发人员，发现问题时便能随即修正，有地方需要加强时，也可以马上就做。

  UNIX已历经数个版本，各个版本以字母V加上数字作为简称，如V6、V7，等等。（正式名称则是遵循发行的使用手册的修订次数编号来命名，例如First Edition、Second Edition，等等。这两种名称的对应其实很直接：V6 = Sixth Edition、V7 = Seventh Edition。和大多数有经验的UNIX程序员一样，这两种命名方式我们都会用到）。影响最深远的UNIX系统是1979年所发行的第7版（Seventh Edition），但是在最初的几年，它仅应用于学术教育机构领域。值得一提的是：第7版的系统同时提出了awk与Bourne Shell，这二者是POSIX Shell的基础，同时，第一本讨论UNIX的书也在此诞生。

- 贝尔实验室的研究人员都是计算机科学家。他们所设计的系统不单单是自己使用，还要分享给同事——这些人一样也是计算机科学家。因此，衍生出“务实”（no nonsense）的设计模式：程序会执行你所赋予的任务，但不会跟你对话，也不会问一堆“你确定吗？”之类的问题。

- 除了精益求精，在设计与问题解决上，他们也不断地追求“优雅”（elegance）。关于“优雅”有一个贴切的定义：简单就是力量（power cloaked in simplicity，注2）。贝尔实验室自由的环境，所造就的不仅是一个可用的系统，也是一个优雅的系统。

当然，自由同样也带来了一些缺点。当UNIX流传至开发环境以外的地方，这些问题也逐一浮现：

- 工具程序之间存在许多不一致的地方。例如，同样的选项字母，在不同程序之间有完全不一样的定义；或是相同的工作却需要指定不同的选项字母。此外，正则表达式的语法在不同程序之间用法类似，却又不完全一致，易产生混淆——这种情况其实可以避免。（直至正则表达式的重要性受到认可，其模式匹配机制才得以收录在标准程序库中。）

- 诸多工具程序具有缺陷，例如输入行（input lines）的长度，或是可打开的文件个数，等等。（现行的系统多半已经修正这些缺陷。）

- 有时程序并未经过彻底测试，这使得它们在执行的时候一不小心就会遭到破坏。这

注2： 我最初是在20世纪80年代从Dan Forsyth口中听到这个定义的。

可能会导致核心转储（core dumps，译注1），令用户不知所措。幸好，现行的UNIX系统极少会面临这样的问题。

- 系统的文档尽管大致上内容完备，但通常极为简单。使得用户在学习时很难找到所需要的信息（注3）。

本书之所以将重点放在文本（而非二进制）数据的处理与运用上，是由于UNIX早期的发展都源自于对文本处理的强烈需求，不过除此之外还有另外的重要理由（马上会讨论到）。事实上，贝尔实验室专利部门（Bell Labs Patent Department）在UNIX系统上所使用的第一套产品，就是进行文本处理和编排工作的。

最初的UNIX机器（Digital Equipment Corporation PDP-11s）不能运行大型程序。要完成复杂的工作，得先将它分割成更小的工作，再用程序来完成这些更小的工作。某些常见的工作（从数据行中取出某些字段、替换文本，等等）也常见于许多大型项目，最后就成了标准工具。人们认为这种自然而生的结果是件好事：由于缺乏大型的解决空间，因而产生了更小、更简单、更专用的程序。

许多人在UNIX的使用上采用半独立的工作方式，重复套用彼此间的程序。由于版本之间的差异，而且不需要标准化，导致许多日常工具程序的发展日渐分歧。举例来说，`grep`在某系统里使用`-i`来表示“查找时忽略大小写”，但在另一个系统中，却使用`-y`来代表同样的事！无独有偶，这种怪事也发生在许多工具程序上。还有，一些常用的小程序可能会取相同的名字，针对某个UNIX版本所编写的Shell程序，不经修改可能无法在另一个版本上执行。

最后，对常用标准工具组与选项的需求终于明朗化，POSIX标准即为最后的结果。现行标准IEEE Std. 1003.1-2004包含了C的库层级的主题，还有Shell语言与系统工具及其选项。

好消息是，在这些标准上所做的努力有了回报。现在的商用UNIX系统，以及可免费使

译注1：在UNIX系统中，常将“主内存”（main memory）称为核心（core），因为在使用半导体作为内存材料之前，便是使用核心（core）。而核心映像（core image）就是“进程”（process）执行当时的内存内容。当进程发生错误或收到“信号”（signal）而终止执行时，系统会将核心映像写入一个文件，以作为调试之用，这就是所谓的核心转储（core dump）。

注3：系统文档分成两个部分：参考手册与使用手册。后者是系统各功能的教学手册。虽然把整份文件读完就可能学会UNIX——事实上有许多人（包括作者）真的是这么做，不过现今的系统，已不再附上打印好的文件。

用的同类型产品，例如GNU/Linux与BSD衍生系统，都兼容POSIX。这样一来，学习UNIX变得更容易，编写可移植的Shell脚本也成为可能（详见第14章）。

值得注意的是：POSIX并非UNIX标准化的唯一成果，POSIX之外仍有其他标准。例如，欧洲计算机制造商协会自行发起了一套名为X/Open的标准。其中最受欢迎的是1988年首度出现的XPG4（X/Open Portability Guide，Fourth Edition）。另外还有XPG5，其更广为人知的名称为UNIX 98标准，或Single UNIX Specification。XPG5很大程度上把POSIX纳入为一个子集，同样深具影响力（注4）。

XPG标准在措辞上可能不够严谨，但其内容却较为广泛，其目标是将现存于UNIX系统上实际用到的各种功能正式生成文档。（POSIX的目的在于建立一套正式的标准，让从头开始的实践者有指导方针可以套用 —— 即便是在非UNIX的平台上。因此，许多UNIX系统上常见的功能，一开始就排除在POSIX标准之外）。2001 POSIX标准由于纳入了X/Open System Interface Extension（XSI）而有了双重身份，也叫做XPG6，这是它首度正式扩张POSIX版图。此文档的特色在于：让系统不只兼容POSIX，也兼容于XSI。所以，当你在编写工具或应用程序时，必须参考的正式文件只有一份（就叫做Single UNIX Standard）。

本书自始至终都把重点放在根据POSIX标准所定义的Shell语言与UNIX工具程序。重点部分也会加入XSI定义的说明，因为你很可能会用得到。

1.2 软件工具的原则

随着时间的流逝，人们开发出了一套设计与编写软件工具的原则。在本书用来解决问题的程序中，你将会看到这些原则的应用示例。好的软件工具应该具备下列特点：

一次做好一件事

在很多方面，这都是最重要的原则。若程序只做一件事，那么无论是设计、编写、调试、维护，以及生成文件都会容易得多。举例来说，对于用来查找文件中是否有符合样式的`grep`程序，不应该指望用它来执行算术运算。

这个原则的结果，自然就是会不断产生出更小、更专用于特定功能的程序，就像专业木匠的工具箱里，永远会有一堆专为特定用途所设计的工具。

处理文本行，不要处理二进制数据

文本行是UNIX的通用格式。当你在编写自己的工具程序时便会发现，内含文本行的数据文件很好处理，你可以用任何唾手可得的文本编辑器来编辑它，也可以让这

注4： X/Open的出版物列表可参见 *http://www.opengroup.org/publications/catalog/*。

些数据在网络与各种机器架构之间传输。使用文本文件更有助于任何自定义工具与现存的UNIX程序之间的结合。

使用正则表达式

正则表达式（regular expression）是很强的文本处理机制。了解它的运作模式并加以使用，可适度简化编写命令脚本（script）的工作。

此外，虽然正则表达式多年来在工具与UNIX版本上不断在变化，但POSIX标准仅提供两种正则表达式。你可以利用标准的库程序进行模式匹配的工作。这样就可以编写出专用的工具程序，用于与grep一致的正则表达式（POSIX称之为基本型正则表达式，Basic Regular Expressions，BRE），或是用于与egrep一致的正则表达式（POSIX称之为扩展型正则表达式，Extended Regular Expressions，ERE）。

默认使用标准输入/输出

在未明确指定文件名的情况下，程序默认会从它的标准输入读取数据，将数据写到它的标准输出，至于错误信息则会传送到标准错误输出（这部分将于第2章讨论）。以这样的方式来编写程序，可以轻松地让它们成为数据过滤器（filter），例如，组成部分的规模越大，越需要复杂的管道（pipeline）或脚本来处理。

避免喋喋不休

软件工具的执行过程不该像在“聊天”（chatty）。不要将“开始处理”（starting processing）、“即将完成”（almost done）或是“处理完成”（finished processing）这类信息放进程序的标准输出（至少这不该是默认状态）。

当你有意将一些工具串成一条管道时，例如：

```
tool_1 < datafile | tool_2 | tool_3 | tool_4 > resultfile
```

若每个工具都会产生“正处理中”（yes I'm working）这样的信息并送往管道，那么别指望执行结果会像预期的一样。此外，若每个工具都将自己的信息传送至标准错误输出，那么整个屏幕画面就会布满一堆无用的过程信息。在工具程序的世界里，没有消息就是好消息。

这个原则其实还有另外一个含义。一般来说，UNIX工具程序一向遵循“你叫它做什么，你就会得到什么”的设计哲学。它们不会问“你确定吗？”（are you sure?）这种问题，当用户键入rm somefile，UNIX的设计人员会认为用户知道自己在做什么，然后毫无疑问地rm删除掉要删除的文件（注5）。

注5：如果你真觉得这样不好，rm的-i选项可强制rm给你提示以做确认，这么一来，当你要求删除可疑文件时，rm便会提示确认它。一直以来，应该“永远不要提示”还是应该“永远得到提示”是个争议的话题，值得用户深思。

输出格式必须与可接受的输入格式一致

专业的工具程序认为遵循某种格式的输入数据，例如标题行之后接着数据行，或在行上使用某种字段分隔符等，所产生的输出也应遵循与输入一致的规则。这么做的好处是，容易将一个程序的执行结果交给另一个程序处理。

举例来说，netpbm程序集（注 5）是用来处理以 Portable BitMap（PBM）格式保存的图像文件（注 6）。这些文件内含 bitmapped 图像，并使用定义明确的格式加以绘制。每个读取PBM文件的工具程序，都会先以某种格式来处理文件内的图像，然后再以 PBM 的格式写回文件。这么一来，便可以组合简单的管道来执行复杂的图像处理，例如先缩放影像后，再旋转方向，最后再把颜色调淡。

让工具去做困难的部分

虽然 UNIX 程序并非完全符合你的需求，但是现有的工具或许已经可以为你完成 90% 的工作。接下来，若有需要，你可以编写一个功能特定的小型程序来完成剩下的工作。与每次都从头开始来解决各个问题相比，这已经让你省去许多工作了。

构建特定工具前，先想想

如前所述，若现存系统里就是没有需要的程序，可以花点时间构建满足所需的工具。然而，动手编写一个能够解决问题的程序前，请先停下来想几分钟。你所要做的事，是否有其他人也需要做？这个特殊的工作是否有可能是某个一般问题的一个特例？如果是的话，请针对一般问题来编写程序。当然，这么做的时候，无论是在程序的设计或编写上，都应该遵循前面所提到的几项原则。

1.3 小结

UNIX 原为贝尔实验室的计算机科学家所开发的产品。由于没有盈利上的压力，再加上 PDP-11 小型计算机的能力有限，因而程序都以小型、优雅为圭臬。也因为没有盈利上的压力，系统之间并非完全一致，学习上也不太容易。

随着 UNIX 持续地流行，各种版本陆续开发出来（尤其是衍生自 System V 和 BSD 的版本），Shell 脚本层次的可移植性也日益困难。幸好，POSIX 标准成熟后，几乎所有商用 UNIX 系统与免费的 UNIX 版本都兼容 POSIX。

注 6：这套程序并非 UNIX 工具集的标准配备，不过 GNU/Linux 与 BSD 系统上通常都会安装。其网站位于 *http://netpbm.sourceforge.net/*。可按照 Sourceforge 项目网页的指示，下载源代码。

注 7：有三种格式。若你的系统里有安装 netpbm，可参阅 *pnm*(5)手册页。

之所以会在这里指出软件工具的设计原则，主要是为了提供开发与使用UNIX工具集的指导方针。让软件工具的设计原则成为思考习惯，将有助于编写简洁的Shell程序和正确使用UNIX工具。