

# LINUX



- ❖ 从基本概念着手，通过丰富、实用的范例，深入浅出地讲解 Shell 编程语言
- ❖ 配有全程录像视频光盘，方便读者学习

## Shell 编程 从入门到精通

张昊 编著



# LINUX



附赠全程视频教学光盘

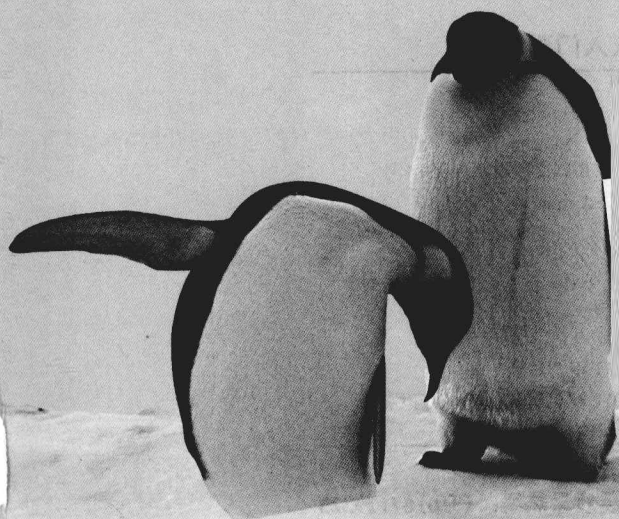


人民邮电出版社  
POSTS & TELECOM PRESS

# LINUX

## Shell 编程 从入门到精通

张昊 编著



# Linux

人民邮电出版社  
北京

## 图书在版编目 (C I P) 数据

Linux Shell编程从入门到精通 / 张昊编著. -- 北京: 人民邮电出版社, 2011.5  
ISBN 978-7-115-24645-5

I. ①L… II. ①张… III. ①Linux操作系统—程序设计 IV. ①TP316.89

中国版本图书馆CIP数据核字(2011)第026184号

## 内 容 提 要

本书由浅入深、循序渐进地详细讲解了 Linux Shell 编程的基本知识。这些基本知识包括 Shell 编程的基本知识、文本处理的工具和方法、正则表达式、Linux 系统知识等。

本书旨在通过理清 Linux Shell 编程的脉络,从基本概念着手,以丰富、实用的范例作为辅助,使读者能够深入浅出的学习 Linux Shell 编程。

本书的每一章节都配有综合案例,这些综合案例不仅可以使读者复习前面所学的知识,还可以增加开发项目的经验。这些案例实用性很强,许多代码可以直接应用到 Linux 系统管理中。

本书配有全程录像的视频讲解光盘,读者可以将视频和书结合起来学习,可以更快、更好地掌握 Linux Shell 编程技术。

本书适合于 Linux Shell 编程的初学者,有一定 Linux Shell 编程基础,但还需要进一步提高的人阅读。本书对于有一定编程经验的程序员也有很好的参考价值。



## Linux Shell 编程从入门到精通

- ◆ 编 著 张 昊  
责任编辑 蒋 佳
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号  
邮编 100061 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
大厂聚鑫印刷有限责任公司印刷
- ◆ 开本: 787×1092 1/16  
印张: 20.75  
字数: 519 千字  
印数: 1-3 000 册

ISBN 978-7-115-24645-5

定价: 49.00 元 (附光盘)

读者服务热线: (010) 67132692 印装质量热线: (010) 67129223  
反盗版热线: (010) 67171154

# 前 言

大学一年级的時候，我剛剛接觸 Linux。那時候的我，沉醉於 Linux 華麗的用戶界面，沉醉於用 Beryl（現在叫做 Compiz Fusion）帶來的圖形效果，並且自我感覺良好：能夠在姑娘們面前炫耀她們從沒有見過的操作系统，讓她們知道我是一個軟件高手。

直到某一天，參加一個學長（此人現在為南京大學高性能計算研究所老師）做的一個 Linux 講座。他使用最簡陋的圖形界面（gnome 默認配置），用兩台運行 Ubuntu Linux 的機器和一個攝像頭，各自打開一個命令行，然後，他的演示讓我目瞪口呆。

他首先將兩台機器配置成連網狀態，再將一台機器（A 機器）接上攝像頭，對着我們，另一台機器（B 機器）接着投影儀，打到大屏幕上。

然後，他在 A 機器的命令行中鍵入了一串長長的命令，又在 B 機器的命令行中鍵入另一串長命令，回車。赫然，我們發現現場觀眾的實時動畫被投影到大屏幕上！

一片嘩然！

他解釋道：主要是管道。我在 A 機器上用讀取命令將圖像從攝像頭中讀取出來，通過管道連接到壓縮程序，壓縮程序將一幀一幀的圖像壓縮，再傳輸到管道中，此時管道就通過無線局域網連接到 B 機器上。B 機器上的解壓程序從管道出口將壓縮幀解壓縮，通過流媒體播放器播放出來，再投放大屏幕上！

管道！

從此我愛上了黑乎乎的命令行，我知道它更強大，並且更有利於程序間的交互。這似乎有去除表象抓住實質的感受。後來再見到 Linux 用戶炫耀他華麗的圖形界面時，我的腦海中總會立刻蹦出一個單詞：fish（菜鳥）。

的確，Linux 命令行就是 Linux 的靈魂。而用戶界面只是運行在靈魂上的皮囊而已。和 Windows 的命令行不同，Linux 命令行的確是一個強大的操縱系統的工具，你可以在命令行里完成幾乎一切日常操作，並且比圖形界面高效和強大得多。

有許多人用了 Linux 好多年，還是在當 Windows 使用，這樣的人大約是得了命令行恐懼症，認為那個黑乎乎的交互界面應該是一些計算機 Geek 才用的；另許多人用了多年的 Linux 命令行，還僅僅會 ls、cp、mv 等簡單的命令，如果他的老板讓他寫一個 Linux Shell 腳本完成某批處理任務，就直接見光死了；而真正的 Linux 高手同時都是 Linux Shell 編程強人，能夠駕馭複雜的命令



行和 Shell 语言，完成神奇的工作。

下面，跟我一起走进 Linux Shell 编程的世界吧！

这本书讲的是什么？

这本书是 Linux Shell 编程的入门书籍。与市场上许多介绍 Linux 的书籍不同，这本书偏重于 Linux Shell 编程，实实在在将 shell 当作一门语言来讲，而不像其他书籍只有一两章提到 shell。而实际上，一两章是绝对不够介绍 shell 编程的，只能蜻蜓点水而已。

这本书涵盖了 Linux Shell 编程的方方面面，具体内容如下所述。

第 1 章介绍了 shell 的一些背景知识。我们从如何运行一个 shell 程序开始讲起，让你先有一个小小的成就感。然后介绍了 shell 的一些背景知识，如 shell 运行的环境变量、shell 的本质。最后，我们对 shell 语言的优势进行了探讨。

第 2 章是一个类似于总括的章节，讲 shell 编程的基础。包括 shell 脚本参数的传递方式、shell 中命令的重形象与管道、基本文本检索的方法，以及 UNIX/Linux 系统的设计思想。在这里提到了 KISS 原则。

第 3 章讲编程的基本元素。语言所特有的性质，在 shell 中大部分都支持。例如变量、函数、条件控制和流程控制，以及非常重要的循环。在本章中，你将会对这些元素的使用有初步的认识。

第 4 章跳出 shell 本身的范畴，介绍了正则表达式。当然，shell 如此强大的原因之一在于文本处理，而正则表达式又是文本匹配的利器。关于正则表达式，除了介绍基本知识外，还以两个案例给出了具体的应用场景，当然是在 Linux Shell 中完成。

第 5 章开始讲文本处理。由于大部分 Linux Shell 脚本都与文本处理相关，因此这是委实重要的一章。主要围绕在一些文本处理的功能上，包括：排序、去重、统计、打印、字段处理和文本替换。最后，以一个综合性的例子结束了本章的介绍。

第 6 章再次停顿一下，讲解文件和文件系统。似乎文件系统和 shell 编程没什么关系？其实不然。shell 脚本无法推卸的职责就是系统管理，怎么可能和文件与文件系统无关呢？这一章介绍了文件的查看、寻找与比较，还介绍了文件系统的定义与选择。

第 7 章介绍 sed。sed 也称为流编辑器，它对整行文本流进行处理。这一章和下一章关系紧密，sed 和 awk 常常在一起使用，是一对工作的好伙伴。

第 8 章介绍 awk。与 sed 不同，awk 往往更善于对字段进行处理。awk 也是一门紧凑的语言，几乎包括语言的所有常见属性。我们以一些 awk 的例子结束本章。

第 9 章是进程相关的一些知识。Linux 中的进程很多，此处介绍了进程的查看与管理、进程间的通信。此处举了两个例子，一个是 Linux 中的第一个进程 init，另一个是 Linux 系统中进程间

管道的实现。然后介绍了 Linux 任务管理工具。最后，拿 Linux 中的进程和线程做比较，分析不同的应用场景。

第 10 章主要介绍 Linux 中的工具。包括不同的 shell、远程登录的工具 SSH、管理多个终端的工具 Screen，以及文本编辑工具 Vim。

第 11 章通过日志清理程序和系统监控程序两个实例介绍了应用 Linux Shell 编程的方法。

至此，几乎涵盖了 Linux Shell 编程的方方面面，更多的高级技巧就需要在实战中慢慢摸索。

### 谁适合读这本书？

虽然这是一本 Linux Shell 编程的入门书籍，但是我仍然希望读者有一定的基础：

能够正常使用 Linux，并且的确有使用终端的需求；

无论是 C、C++ 还是 Java，甚至 VB，至少了解一门计算机语言，当然，如果你有其他脚本语言的基础会更好。

祝你在这本书里获取到想要的知识。

### 这本书能帮助你什么？

相信你已经对这本书介绍的内容有了一定了解。这本书的目标在于，帮助一个 Linux Shell 新手掌握 Linux Shell 脚本编程，从而改变他与 Linux 系统的交互方式：更加有效、准确、简单、完美。

当然，仅仅靠这本书还是不够的，你需要勤加练习。必要时，还要求助于 Mr Google。毕竟，人类的知识，Mr Google 几乎都知道。

### 如何联系作者？

如果你有任何意见或建议，可以通过邮箱联系到作者：ollir@live.com。我会在第一时间给你回复。

### 感谢

感谢我曾经的导师和学校（南京大学），是他们系统地教会我 Linux 上的 Shell 编程与实用技巧。

感谢我大学阶段参与创建的一个 Linux 社团 Open Association(<http://njuopen.com>)，这个社团促我成长，带领我走进 Linux 的广袤世界。

感谢我的女朋友，她做出了一定的牺牲，让我周末有时间写稿，而不是陪她逛街。

感谢马泽民、逯永广、林丹、李辉、田芳、王建国、赵海峰、刘勇、徐超、周建军、徐兵、



黄飞、林海、马建华、孙明、高峰、郑勇、刘建、李彬、彭丽、许小荣等同志，他们也参与了本书的编写和最终的整理。

感谢出版社，他们对稿件的编校和发行做出了极大的努力。没有他们，这本书不可能呈现在读者的面前。

编者  
2011.3

## 目 录

第 1 章 第一个 Shell 程序.....1	第 3 章 编程的基本元素..... 39
1.1 第一道菜.....2	3.1 再识变量..... 40
1.2 如何运行程序.....2	3.1.1 用户变量..... 41
1.2.1 选婿：位于第一行的#!.....2	3.1.2 位置变量..... 46
1.2.2 找茬：程序执行的差异.....4	3.1.3 环境变量..... 48
1.2.3 shell 的命令种类.....4	3.1.4 启动文件..... 49
1.3 Linux Shell 的变量.....6	3.2 函数..... 51
1.3.1 变量.....6	3.2.1 函数定义..... 52
1.3.2 用 echo 输出变量.....8	3.2.2 函数的参数和返回值..... 53
1.3.3 环境变量的相关操作.....9	3.3 条件控制与流程控制..... 54
1.3.4 shell 中一些常用环境变量.....12	3.3.1 if/else 语句..... 54
1.4 Linux Shell 是解释型语言.....12	3.3.2 退出状态..... 54
1.4.1 编译型语言与解释型语言.....12	3.3.3 退出状态与逻辑操作..... 56
1.4.2 Linux Shell 编程的优势.....13	3.3.4 条件测试..... 56
1.5 小结.....14	3.4 循环控制..... 61
第 2 章 Shell 编程基础.....15	3.4.1 for 循环..... 61
2.1 向脚本传递参数.....16	3.4.2 while/until 循环..... 62
2.1.1 Shell 脚本的参数.....16	3.4.3 跳出循环..... 63
2.1.2 参数的用途.....17	3.4.4 循环实例..... 63
2.2 I/O 重定向.....20	3.5 小结..... 65
2.2.1 标准输入、标准输出与标准错误.....20	第 4 章 正则表达式..... 66
2.2.2 管道与重定向.....22	4.1 什么是正则表达式..... 67
2.2.3 文件描述符.....23	4.1.1 正则表达式的广泛应用..... 67
2.2.4 特殊文件的妙用.....24	4.1.2 如何学习正则表达式..... 67
2.3 基本文本检索.....28	4.1.3 如何实践正则表达式..... 68
2.4 UNIX/Linux 系统的设计与 shell 编程.....31	4.2 正则基础..... 69
2.4.1 一切皆文件.....31	4.2.1 元字符..... 69
2.4.2 UNIX 编程的基本原则.....34	4.2.2 单个字符..... 72
2.5 小结.....37	4.2.3 单个表达式匹配多个字符..... 73
	4.2.4 文本匹配锚点..... 74
	4.2.5 运算符优先级..... 74
	4.2.6 更多差异..... 75





4.3 正则表达式的应用.....	76	6.1.1 列出文件.....	119
4.3.1 扩展.....	76	6.1.2 文件的类型.....	122
4.3.2 案例研究: 罗马数字.....	77	6.1.3 文件的权限.....	123
4.3.3 案例研究: 解析电话号码.....	82	6.1.4 文件的修改时间.....	131
4.4 小结.....	86	6.2 寻找文件.....	133
<b>第5章 基本文本处理.....</b>	<b>87</b>	6.2.1 find 命令的参数.....	133
5.1 排序文本.....	88	6.2.2 遍历文件.....	137
5.1.1 sort 命令的行排序.....	90	6.3 比较文件.....	138
5.1.2 sort 命令的字段排序.....	92	6.3.1 使用 comm 比较排序后文件.....	138
5.1.3 sort 小结.....	93	6.3.2 使用 diff 比较文件.....	139
5.2 文本去重.....	94	6.3.3 其他文本比较方法.....	141
5.3 统计文本行数、字数以及字符数.....	96	6.4 文件系统.....	142
5.4 打印和格式化输出.....	97	6.4.1 什么是文件系统.....	143
5.4.1 使用 pr 打印文件.....	97	6.4.2 文件系统与磁盘分区.....	143
5.4.2 使用 fmt 命令格式化文本.....	99	6.4.3 Linux 分区格式的选择与安全性.....	145
5.4.3 使用 fold 限制文本宽度.....	101	6.4.4 文件系统与目录树.....	147
5.5 提取文本开头和结尾.....	102	6.4.5 文件系统的创建与挂载.....	151
5.6 字段处理.....	104	6.5 小结.....	154
5.6.1 字段的使用案例.....	104	<b>第7章 流编辑.....</b>	<b>155</b>
5.6.2 使用 cut 取出字段.....	105	7.1 什么是 sed.....	156
5.6.3 使用 join 连接字段.....	107	7.1.1 挑选编辑器.....	156
5.6.4 其他字段处理方法.....	110	7.1.2 sed 的版本.....	156
5.7 文本替换.....	110	7.2 sed 示例.....	156
5.7.1 使用 tr 替换字符.....	110	7.2.1 sed 的工作方式.....	156
5.7.2 其他选择.....	113	7.2.2 sed 工作的地址范围.....	158
5.8 一个稍微复杂的例子.....	114	7.2.3 规则表达式.....	159
5.8.1 实例描述.....	114	7.3 更强大的 sed 功能.....	161
5.8.2 取出记录的 ip 字段和 id 字段.....	114	7.3.1 替换.....	162
5.8.3 将记录按照 IP 顺序排序.....	115	7.3.2 地址范围的迷惑.....	163
5.8.4 使用 uniq 统计重复 IP.....	115	7.4 组合命令.....	164
5.8.5 根据访问次数进行排序.....	116	7.4.1 组合多条命令.....	164
5.8.6 提取出现次数最多的 100 条.....	116	7.4.2 将多条命令应用到一个地址范围.....	166
5.9 小结.....	117	7.5 实际的例子.....	166
<b>第6章 文件和文件系统.....</b>	<b>118</b>	7.6 sed 实践.....	167
6.1 文件.....	119	7.6.1 第一步 替换名字.....	168
		7.6.2 第二步 删除前 3 行.....	168

7.6.3	第三步 显示 5~10 行	169	8.6.3	检验 passwd 格式的正 确性	214
7.6.4	第四步 删除包含 Lane 的行	169	8.6.4	sed/awk 单行脚本	215
7.6.5	第五步 显示生日在 November-December 之间 的行	170	8.7	小结	222
7.6.6	第六步 把 3 个星号 (***) 添加到以 Fred 开头的行	170	<b>第 9 章 进程</b>		223
7.6.7	第七步 用 JOSE HAS RETIRED 取代包含 Jose 的行	171	9.1	进程的含义与查看	224
7.6.8	第八步 把 Popeye 的生日 改成 11/14/46	172	9.1.1	理解进程	224
7.6.9	第九步 删除所有 空白行	173	9.1.2	创建进程	224
7.6.10	第十步 脚本	174	9.1.3	查看进程	225
7.7	小结	175	9.1.4	进程的属性	229
<b>第 8 章 文本处理利器 awk</b>		176	9.2	进程管理	230
8.1	来个案例	177	9.2.1	进程的状态	230
8.2	基本语法	178	9.2.2	shell 命令的执行	232
8.2.1	多个字段	178	9.2.3	进程与任务调度	233
8.2.2	使用其他字段分隔符	179	9.3	信号	239
8.3	AWK 语言特性	181	9.3.1	信号的基本概念	239
8.3.1	AWK 代码结构	181	9.3.2	产生信号	242
8.3.2	变量与数组	184	9.4	Linux 的第一个进程 init	244
8.3.3	算术运算和运算符	186	9.5	案例分析: Linux 系统中管道的 实现	247
8.3.4	判断与循环	188	9.6	调度系统任务	249
8.3.5	多条记录	192	9.6.1	任务调度的基本介绍	249
8.4	用户自定义函数	194	9.6.2	调度重复性系统 任务 (cron)	250
8.4.1	自定义函数格式	194	9.6.3	使用 at 命令	256
8.4.2	引用传递和值传递	196	9.7	进程的窗口/proc	259
8.4.3	递归调用	197	9.7.1	proc——虚拟文件系统	260
8.5	字符串与算术处理	199	9.7.2	查看/proc 的文件	260
8.5.1	格式化输出	199	9.7.3	从 proc 获取信息	261
8.5.2	字符串函数	201	9.7.4	通过/proc 与内核交互	263
8.5.3	算术函数	206	9.8	Linux 的线程简介	264
8.6	案例分析	210	9.8.1	Linux 线程的定义	264
8.6.1	生成数据报表	210	9.8.2	Pthread 线程的使用场合	264
8.6.2	多文件联合处理	212	9.8.3	Linux 进程和线程的 发展	265
			9.9	小结	265
			<b>第 10 章 超级工具</b>		267
			10.1	不同的 shell	268



10.1.1	修改登录 shell 和 切换 shell .....	268	10.4.5	鼠标的移动 .....	294
10.1.2	选择 shell .....	270	10.4.6	基本编辑指令 .....	295
10.2	SSH .....	273	10.4.7	复制 (yank) .....	299
10.2.1	SSH 的安全验证机制 .....	273	10.4.8	搜寻、替换 .....	301
10.2.2	使用 SSH 登录远程 主机 .....	274	10.5	小结 .....	303
10.2.3	OpenSSH 密钥管理 .....	276	<b>第 11 章</b>	<b>Linux Shell 编程实战</b> .....	<b>304</b>
10.2.4	配置 SSH .....	281	11.1	日志清理 .....	305
10.2.5	使用 SSH 工具套装拷贝 文件 .....	282	11.1.1	程序行为介绍 .....	305
10.3	screen 工具 .....	283	11.1.2	准备函数 .....	305
10.3.1	任务退出的元凶: SIGHUP 信号 .....	284	11.1.3	日志备份函数 .....	309
10.3.2	开始使用 screen .....	285	11.1.4	定时运行 .....	310
10.3.3	screen 常用选项 .....	287	11.1.5	代码回顾 .....	311
10.3.4	实例: ssh+screen 管理远程 会话 .....	289	11.2	系统监控 .....	312
10.4	文本编辑工具 Vim .....	289	11.2.1	内存监控函数 .....	313
10.4.1	为什么选择 Vim .....	290	11.2.2	硬盘空间监控函数 .....	314
10.4.2	从何处获取 Vim .....	290	11.2.3	CPU 占用监控函数 .....	315
10.4.3	Vim 的工作模式 .....	292	11.2.4	获取最忙碌的进程信息 .....	319
10.4.4	首次接触: step by step .....	293	11.2.5	结合到一起 .....	320
			11.2.6	代码回顾 .....	320
			11.3	小结 .....	322

# LINUX

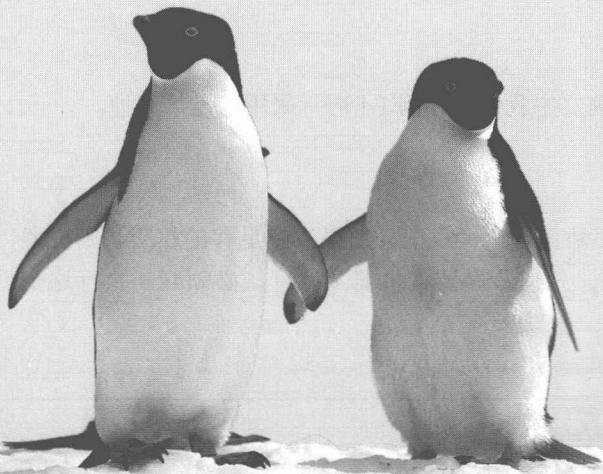
## 第1章 第一个Shell程序

欢迎来到 Linux Shell 编程世界。让我们开始吧。

在本章中，你将会学习如下知识：

- (1) 编译型语言与解释型语言的差异，Linux Shell 编程的优势；
- (2) 如何编写和运行 Linux Shell 程序；
- (3) Linux Shell 运行在环境变量中，环境变量的设置。

本章涉及的 Linux 命令有 sh、bash、echo、pwd、chmod、source、rm、more、set、unset、export、env。





# 1.1 第一道菜

尽管许多的 UNIX 书籍要从各种 UNIX 版本和分支讲起，洋洋洒洒罗哩罗嗦，但是还是让我们跳过这部分吧。

先来看一个实例——echo.sh。

实例：echo.sh

```
#!/bin/sh
cd /tmp
echo "hello world!"
```

这是一个完整的、可执行的 Linux Shell 程序。

它可能是世界上最简单的程序了，以至于你只要一眼，就能看出它葫芦里卖的是什么药（如果你知道 echo 命令的话）。别急着往后跳，因为程序并不是这一章的重点。

现在运行一下这个程序，看看结果是什么（见例 1.1）。

## 例 1.1 运行实例 echo.sh

```
$> pwd #查看当前工作目录
/home/prince #返回当前工作目录
$> chmod +x echo.sh #将 echo.sh 文件权限变为可执行
$> ./echo.sh #运行 echo.sh 文件
hello world! #返回运行结果
$> pwd
/home/prince #工作目录没改变
```

好，程序已经发挥作用了。很简单，不是吗？别高兴得太早，现在我要抛出几个问题，接招吧：

- (1) 程序第一行“#!/bin/sh”是什么意思？
- (2) 如何运行程序？

# 1.2 如何运行程序

运行 Linux 程序有 3 种方法：

- (1) 使文件具有可执行权限，直接运行文件；
- (2) 直接调用命令解释器<sup>①</sup>执行程序；
- (3) 使用 source 执行文件。

第 3 种方法运行结果和前两种是不同的。前文中我们运行例 1.1 时，采用的是第一种。

## 1.2.1 选婿：位于第一行的#!

当命令行 shell 执行程序时，首先判断程序是否有执行权限。如果没有足够的权限，则机器会告诉用户：“权限不够”。从安全的角度考虑，任何程序要在机器上执行，必须判断执行这个

<sup>①</sup> 参见 1.4，Linux Shell 是解释型语言。

程序的用户是否具有相应的权限。在第一种方法中，我们直接执行文件，则需要文件具有可执行权限。

`chmod` 命令可以修改文件的权限。`+x` 参数使程序文件具有可执行权限。

命令行 `shell` 接收到我们的执行命令，并且判定我们有执行权限后，则调用 Linux 内核命令新建 (`fork`) 一个进程，在新建的进程中调用我们指定的命令。如果这个命令文件是编译型的（二进制文件），则 Linux 内核知道如何执行文件。不幸的是，我们的 `echo.sh` 程序文件并不是编译型的，而是文本文件，内核并不知道如何执行，于是，内核返回“not executable format file”（不是可执行的文件类型）错误。`shell` 收到这个信息时，说：“内核不知道怎么运行，我知道，这一定是个脚本！”。

`shell` 知道这是个脚本后，启动了一个新的 `shell` 进程来执行这个程序。但是现在的 Linux 系统往往拥有好几个 `shell`，到底挑选哪个“女婿”呢？这就要看脚本中意哪个了：在第一行，脚本通过“`#!/bin/sh`”告诉命令行：“我只和他好，让他来执行吧！”。

这种“选婿”方法有助于执行方式的通用化。用户在编写脚本时，在程序的第一行通过 `#!` 来设置：告诉运行 `shell` 创建一个什么样的进程来执行此脚本。在我们的 `echo.sh` 中，`shell` 创建了一个 `/bin/sh`（标准 `shell`）进程来执行脚本。

命令行在扫过第一行，发现 `#!` 时，试图读取 `#!` 之后的字符，搜寻解释器的完整路径。如果在第一行中的解释器也有参数，则一并读取。例如，可以这样来引用我们的解释器：

```
#!/bin/bash -l
```

这样，命令行 `shell` 会启用一个新的 `bash` 进程来执行程序的每一行。并且，`-l` 参数使得这个 `bash` 进程的反应与登录 `shell` 相似。

这种“选婿”方法，使得我们可以调用任何解释器，并不局限于 Linux Shell。例如，可以创建这样一个 `python`<sup>①</sup> 程序：

```
#!/usr/bin/python
print "hello world"!
```

当这个文件被赋予可执行权限，并且用第一种方式运行时，就像调用了 `python` 解释器来执行一样。

#### NOTE

填写完整的解释器路径。如果不知道某解释器的完整路径，使用 `whereis` 命令查询

```
$> whereis bash
/bin/bash
```

每个脚本的头都指定了一个不同的命令解释器，为了帮助你打破 `#!` 的神秘性，我们可以这样来写一个脚本。见例 1.2。

#### 例 1.2 自删除脚本

```
#!/bin/rm
# 自删除脚本
# 当你运行这个脚本时，基本上什么都不会发生...当然这个文件消失不见了
WHATEVER=65
echo "This line will never print!"
exit $WHATEVER # 不要紧，脚本是不会在这退出的
```

<sup>①</sup> 参见 <http://www.python.org>。

当然,你还可以试试在一个 README 文件的开头加上一个 `#!/bin/more`,并让它具有执行权限。结果将是文件中自动列出自己的内容。

## 1.2.2 找茬: 程序执行的差异

在 3 种程序执行方法中,如果 `#!` 中指定的 shell 解释器和第二种指定的 shell 解释器相同的话,这两种的执行结果就是相同的。我们来看看第 3 种方法的执行过程:

### 例 1.3

```
$> pwd                #查看当前工作目录
/home/prince         #返回当前工作目录
$> source echo.sh    #运行 echo.sh 文件
hello world!        #返回运行结果
$> pwd
/tmp                #工作目录改变
```

细心的你一定发现了不同!是的,当前目录发生了改变!

我们再来看例 1.4。

### 例 1.4

```
$> pwd                #查看当前工作目录
/home/prince
$> cd /tmp           #改变当前工作目录
$> pwd
/tmp                #工作目录改变
```

为什么例 1.3 和 1.4 的 `cd` 命令可以改变工作目录,而例 1.1 中的工作目录并没有改变呢?这个问题的答案,我们将在 1.2.3 节揭晓。

## 1.2.3 shell 的命令种类

Linux Shell 可执行的命令有 3 种:内建命令、shell 函数和外部命令。

(1) 内建命令就是 shell 程序本身包含的命令。这些命令集成在 shell 解释器中,例如,几乎所有的 shell 解释器中都包含 `cd` 内建命令来改变工作目录。部分内建命令的存在是为了改变 shell 本身的属性设置,在执行内建命令时,没有进程的创建和消亡;另一部分内建命令则是 I/O 命令,例如 `echo` 命令。

(2) Shell 函数是一系列程序代码,以 shell 语言写成,它可以像其他命令一样被引用。我们在后面将详细介绍 shell 函数。

(3) 外部命令是独立于 shell 的可执行程序。例如 `find`、`grep`、`echo.sh`。命令行 shell 在执行外部命令时,会创建一个当前 shell 的复制进程来执行。在执行过程中,存在进程的创建和消亡。外部命令的执行过程如下。

① 调用 POSIX 系统 `fork` 函数接口,创建一个命令行 shell 进程的复制(子进程)。

② 在子进程的运行环境中,查找外部命令在 Linux 文件系统的位置。如果外部命令给出了完全路径,则略过查找这一步。

③ 在子进程里,以新程序取代 shell 拷贝并执行(`exec`),此时父进程进入休眠,等待子进程执行完毕。

④ 子进程执行完毕后,父进程接着从终端读取下一条命令,过程如图 1-1 所示。

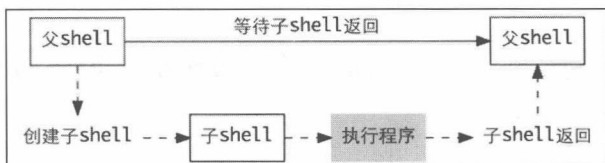


图 1-1 创建进程

**NOTE**

- (1) 子进程在创建初期和父进程一模一样，但是子进程不能改变父进程的参数变量。
- (2) 只有内建命令才能改变命令行 shell 的属性设置（环境变量）。

我们回到例 1.1。在这个例子中，我们使用 `cd`（内建命令）试图改变工作目录，但是未获成功。为了理解失败的原因，来看图 1-2，它说明了执行的过程。

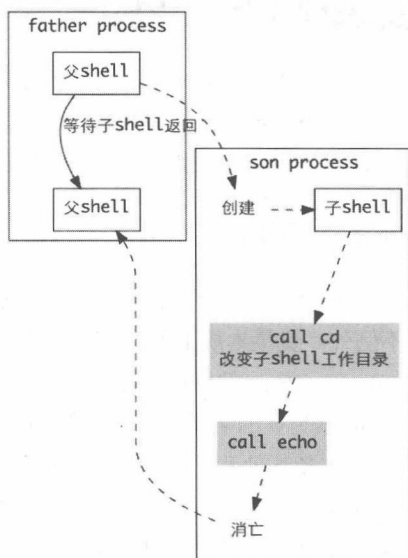
在我们运行 shell 程序的 3 种方法中，前两种方法的执行过程都可以用图 1-2 解释，如下所述。

(1) 父进程接收到命令“`/echo.sh`”或“`/bin/sh echo.sh`”，发现不是内建命令，于是创建了一个和自己一模一样的 shell 进程，来执行这个外部命令。

(2) 这个 shell 子进程用 `/bin/sh` 取代自己，`sh` 进程设置自己的运行环境变量，其中包括 `$PWD` 变量（标识当前工作目录）。

(3) `sh` 进程依次执行内建命令 `cd` 和 `echo`，在此过程中，`sh` 进程（子进程）的环境变量 `$PWD` 被 `cd` 命令改变。注意：父进程的环境变量并没有改变。

(4) `sh` 子进程执行完毕，消亡。一直在等待的父进程醒来继续接收命令。

图 1-2 `echo.sh` 的执行过程

这样，例 1.1 中 `cd` 命令失效的原因就可以理解了！

聪明的你，一定猜到了例 1.3 中使用 `source` 命令，为什么可以改变命令行 shell 的环境变量了吧！

这也是在例 1.1 中目录没有改变的原因：父进程的当前目录（环境变量）无法被子进程改变！

**NOTE**

使用 `source` 执行 shell 脚本时，不会创建子进程，而是在父进程中直接执行！

**source**

## 语法

```
source file
. file
```



**描述**

使用 shell 进程本身执行脚本文件。source 命令也被称为“点命令”，通常用于重新执行刚修改的初始化文件，使之立即生效。

**行为模式**

和其他运行脚本不同的是，source 命令影响 shell 进程本身。在脚本执行过程中，并没有进程创建和消亡。

**警告**

当需要在程序中修改当前 shell 本身的环境变量时，使用 source 命令。

## 1.3 Linux Shell 的变量

你一定想知道，shell 是如何记忆工作目录的改变呢？当 shell 接收到外部命令时，在庞大的文件系统中，如何迅速定位到命令文件呢？如何设定 Linux Shell 的环境变量？甚至如何使这黑乎乎的命令行看起来更漂亮？这些都是这一节要解决的问题。

### 1.3.1 变量

变量 (variable) 在许多程序设计语言中都有定义，与变量相伴的有使用范围的定义。Linux Shell 也不例外。变量本质上就是一个键值对。例如，str = “hello”，就是将字符串值(value) “hello” 赋予键(key) str。在 str 的使用范围内，我们都可以用 str 来引用 “hello” 值，这个操作叫做变量替换。

shell 变量的名称以一个字母或下划线符号开始，后面可以接任意长度的字母、数字或下划线。和许多其他程序设计语言不同的是，shell 变量名称字符并没有长度限制。Linux Shell 并不对变量区分类型。一切值都是字符串，并且和变量名一样，值并没有字符长度限制。神奇的是，bash 也允许比较操作和整数操作。其中的关键因素是：变量中的字符串值是否为数字。如例 1.5 所示。

#### 例 1.5 Linux Shell 中的变量

```
$> long_str="linux_shell_programming"
$> echo $long_str
linux_shell_programming
$> add_1=100
$> add_2=200
$> echo $(( $add_1+$add_2 ))
300
```

由例 1.5 可见，虽然 Linux Shell 中的变量都是字符串类型的，但是同样可以执行比较操作和整数操作，只要变量字符串值是数字。

变量赋值的方式为：

变量名称 = 值

其中 = 号两边不要有任何空格。当你想使用变量名称来获得值时，在名称前加上 \$ 符号。例如 \$long\_str。当赋值的内容包含空格时，请加引号，例如：