${@:offset:num}指定从 offset 开始的 num 个位置参数。

${param/pattern/repl}用指定的字符串 repl 替换 param 的值中第一个匹配模式 pattern 的字符串，其值为替换后的 param 的值，但 param 本身的值不变。

${param//pattern/repl}用指定的字符串 repl 替换 param 的值中所有匹配模式 pattern 的字符串，其值为替换后的 param 的值，但 param 本身的值不变。

${param/#pattern/repl}，如果 param 的值以模式 pattern 开头，那么它将被 repl 替换，其值为替换后的 param 的值，但 param 本身的值不变。

${param/%pattern/repl}，如果 param 的值以模式 pattern 结尾，那么它将被 repl 替换，其值为替换后的 param 的值，但 param 本身的值不变。

（7）约束函数。

约束函数是与特定的变量关联的函数。它允许你每次定义和调用用于引用、设置或取消变量的函数。这些函数采用 varname.function 的格式，varname 是变量名，而 function 是约束函数。预定义的约束函数有 get、set 和 unset。

varname.get 函数在每次变量 varname 被引用时调用。如果在这个函数中设置了特殊变量.sh.value，那么变量 varname 的值会变为此特殊变量的值。例如，我们在命令行定义了一个函数 time.get：

```
$ function time.get
> {
>   .sh.value=$(date +%r)
> }
```

接下来我们打印变量 time 的值，将得到类似如下的结果：

```
$ echo $time
07:14:49 PM
$ echo $time
07:15:12 PM
```

varname.set 函数在每次变量 varname 被设置时调用。如果在这个函数中设置了特殊变量.sh.value，那么变量 varname 的值会被赋值为此特殊变量的值。例如，我们在命令行定义了一个函数 adder.set：

```
$ function adder.set
> {
>   let .sh.value="${.sh.value} + 1"
> }
```

接下来我们在命令行下给变量 adder 赋值，再查看变量 adder 的值，将会得到类似如下的结果：

```
$ adder=0
$ echo $adder
1
$ adder=$adder
$ echo $adder
2
```

varname.unset 函数在每次变量 varname 被取消时调用。

（8）函数环境的不同。

用“function funcname”格式声明的函数运行在一个单独的函数环境并支持本地变量。

用“funcname()”格式声明的函数与父 Shell 运行在同一环境下。

（9）命令返回值。

- 如果执行的命令没找到，返回值是 127。
- 如果执行的命令为不可执行文件，返回值是 126。
- 如果命令是可执行的，但被信号终结，返回值是 256 加信号值。

（10）PATH 搜索规则。

特殊内部命令首先被搜索，然后是所有函数（包括那些在 FPATH 目录中的），然后是其他内部命令。

（11）新增的内部命令。

ksh93 中增加了如下内部命令。

- builtin 命令：列出所有可用的内部命令。
- printf 命令：工作原理与 C 库例程 printf()类似。
- disown 命令：阻止 Shell 发送 SIGHUP 信号到指定的命令。
- getconf 命令：其工作原理与命令/usr/bin/getconf 类似。
- read 命令增加了如下选项：

read -d {char}允许你指定一个字符分隔符替代默认的换行符。

read -t {seconds}允许你指定一个时间限制，在几秒钟之后，read 命令会超时。如果 read 命令超时，它将返回 FALSE。

- exec 命令增加了如下选项：

exec -a {name} {cmd}指定使用 name 替换命令 cmd 的参数 0。

exec -c {cmd}让 exec 在执行 cmd 之前清除环境。

- kill 命令增加了如下选项：

kill -n {signum}用于指定发送到进程的信号值。

kill -s {signame}用于指定发送到进程的信号名。

- whence 命令增加了如下选项：

-a 选项显示所有匹配。

-f 选项跳过函数的搜索。

- 所有常规内部命令都识别-?选项，用于显示指定命令的语法。

（12）ksh93 与 ksh 的其他不同。

- 在 ksh93 中，你不能使用内部命令 typeset -fx 来导出函数。
- 在 ksh93 中，你不能使用 alias -x 内部命令来导出别名。
- 在 ksh93 中，一个美元符号后跟一个单引号（$'）被解释为一个 ANSI C 字符串。你必须用双引号将美元符号括起（\"$\"）来得到旧版本的 ksh 的结果。
- ksh93 内部命令的参数解析逻辑已经被改变。ksh 中可用的内部命令的非法参数组合在 ksh93 中将无法工作。例如，在 ksh 中，typeset -4i 的执行与 typeset -i4 类似，但在 ksh93 中，typeset -4i 就无法工作。
- ksh93 移除了 ERRNO 变量。
- 在 ksh93 中，对于非交互式 Shell，重定向符号后的文件名不会被扩展。
- 在 ksh93 中，你必须使用 alias 内部命令的-t 选项显示跟踪别名。
- 在 ksh93 中的 emacs 编辑模式下，Ctrl+T 组合键会互换当前和前一个字符。而在

ksh 的 emacs 模式下，Ctrl+T 组合键会互换当前和后一个字符。

- 在 ksh93 中，不允许不对称的括号。例如，${name-(}需要一个转义字符${name-\(}。
- 在 ksh93 中，kill -1 命令只显示信号名，不会显示数值。

## 15.3　小　　结

下面我们总结一下本章所学的主要知识。

C Shell（简称 csh）是由 Bill Joy 在 1979 年开发的。它是一种比 Bourne Shell（sh）更适于编程的 Shell。C Shell 的总体风格看起来更像 C 语言并且看起来可读性更好。

在很多系统中（例如，Mac OS X 和 RedHat Linux），csh 实际上是 tcsh，tcsh 是 csh 的改进版。在这些系统中，csh 和 tcsh 都链接到包含 tcsh 可执行程序的同一个文件，所以它们都调用同一个 C Shell 的改进版。

我们知道 Unix（以及 Linux 和类 Unix）系统几乎完全由 C 语言写成，所以 C Shell 作为命令语言的首要目标就是在文体上尽量与系统其他部分保持一致。C Shell 的关键字、使用的圆括号、内部表达式语法和对数组的支持都是受 C 语言强烈的影响。

C Shell 的主要目标之一是更好地用于交互式使用。它引入了很多使它更简单快速的新特性，以及更友好地在终端打印命令。这些特性中最重要的是历史记录、编辑机制、别名、目录堆栈、波浪号、cdpath、作业控制和路径散列法。

C Shell 脚本的第一行为“#!/bin/csh”（如此路径与你系统中的路径不符，请替换）。

tcsh 在 csh 基础上的主要增强功能包括：命令行编辑器、编辑指令、补全和列表、拼写校正、目录堆栈替换、自动及定期和定时事件、本地语言系统支持、终端管理和新增的变量。

Korn Shell 简称为 ksh，它是由 David Korn 于 20 世纪 80 年代初期在贝尔实验室开发的，并于 1983 年 7 月 14 日在由高级计算机系统协会（USENIX）赞助的 USENIX 年度技术大会上宣布。自 2005 年年初的 93q 版本开始，它便在通用公共许可证（CPL）之下，作为 AT&T 软件技术的开源软件集合的一部分。

最初的 ksh（ksh88）的功能被作为 POSIX.2（Shell 和实用程序）标准中命令解释器的基础。

ksh 从 C Shell 中引入的新特性包括：作业控制、别名、函数和命令历史。

ksh 自身与 Bourne Shell 相比的主要新特性包括：命令行编辑、集成编程特性、控制结构、调试功能、正则表达式、增强的 I/O 工具、新的选项和变量、提高了性能及安全特性。

ksh、sh 和 csh 都有的控制结构是 if/else、for、case 和 while。而 select 控制结构是在 ksh 中新加入的。

ksh 的特权模式是一个优秀的安全特性。它解决了原来环境文件的概念第一次出现在 C Shell 中时所引入的问题。

ksh93 是 Korn Shell 的最新版本，这个增强版本不仅向前兼容旧版本的 ksh（ksh88），还包括了一些在 ksh88 中不可用的额外特性。

与 ksh88 相比，ksh93 引入的新特性包括：算法改进、支持复合变量、支持复合赋值、支持联合数组、支持变量名引用、增强的参数扩展、约束函数、函数环境的不同、命令返回值、PATH 搜索规则及一些新增的内部命令和选项等。

[General Information]
书名=Linux Shell命令行及脚本编程实例详解
SS号=13708609