NAME

bash - GNU Bourne-Again SHell (GNU 命令解释程序 “Bourne 二世”)

概述(SYNOPSIS)

bash [options] [file]

版权所有(COPYRIGHT)

描述(DESCRIPTION)

Bash 是一个与 sh 兼容的命令解释程序，可以执行从标准输入或者文件中读取的命令。 Bash 也整合了 Korn 和 C Shell (ksh 和 csh) 中的优秀特性。

Bash 的目标是成为遵循 IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2，可移植操作系统规约： shell 和工具) 的实现。

选项(OPTIONS)

除 了在 set 内建命令的文档中讲述的单字符选项 (option) 之外，bash 在启动时还解释下列选项。

-c string 如果有 -c 选项，那么命令将从 string 中读取。如果 string 后 面有参数 (argument)，它们将用于给位置参数 (positional parameter ，以 $0 起始) 赋值。

-i 如果有 -i 选项，shell 将交互地执行 (interactive)。

-l 选项使得 bash 以类似登录 shell (login shell) 的方式启动 (参见下面的 启动(INVOCATION) 章节)。

-r 如果有 -r 选项，shell 成为受限的 (restricted) (参见下面的受限的 shell(RESTRICTED SHELL) 章节)。

-s 如果有 -s 选项，或者如果选项处理完以后，没有参数剩余，那么 命令将从标准输入读取。这个选项允许在启动一个交互 shell 时可以设置位置参数。

-D 向标准输出打印一个以 $ 为前导的，以双引号引用的字符串列表。这是 在当前语言环境不是 C 或 POSIX 时，脚本中需要翻译的字符串。这个选项隐含了 -n 选项；不会执行命令。

[-+]O [shopt_option]

shopt_option 是一个 shopt 内建命令可接受的选项 (参见 下 面 的 shell 内 建 命 令(SHELL BUILTIN COMMANDS) 章节)。如果有

shopt_option，-O 将设置那个选项的取值； +O 取消它。如果没有给出 shopt_option，shopt 将在标准输出上打印设为允许的选项的名称和值。如果启动选项是 +O，输出将以一种可以重用为输入的格式显示。

-- -- 标志选项的结束，禁止其余的选项处理。任何 -- 之后的参数将作为文件名和参数对待。参数 - 与此等价。

Bash 也解释一些多字节的选项。在命令行中，这些选项必须置于需要被识别的单字符参数之前。

--dump-po-strings
等价于 -D，但是输出是 GNU gettext po (可移植对象) 文件格式

--dump-strings
等价于 -D

--help 在标准输出显示用法信息并成功退出

--init-file file

--rcfile file
如 果 shell 是交互的，执行 file 中的命令，而不是标准的个人初始化文件 ~/.bashrc (参见下面的 启动(INVOCATION) 章节)

--login
等价于 -l

--noediting
如果 shell 是交互的，不使用 GNU readline 库来读命令行

--noprofile
不读取系统范围的启动文件 /etc/profile 或者任何个人初 始 化 文 件 ~/.bash_profile, ~/.bash_login, 或 ~/.profile 。默认情况下，bash 在作为登录 shell 启动时读取这些文件 (参见下面的 启 动(INVOCATION) 章节)

--norc 如果 shell 是交互的，不读取/执行个人初始化文件 ~/.bashrc 这个选项在 shell 以 sh 命令启动时是默认启用的

--posix
如果默认操作与 POSIX 1003.2 标准不同的话，改变 bash 的行为来符合标准 (posix mode)

--restricted

shell 成为受限的 (参见下面的 受限的 shell(RESTRICTED SHELL) 章节)

--rpm-requires

产生一个为使脚本运行，需要的文件的列表。这个选项包含了 -n 选项。它 是为了避免进行编译期错误检测时的限制－－ Backticks, [] tests, 还有 evals 不会被解释，一些依赖关系可能丢失

--verbose

等价于 -v

--version

在标准输出显示此 bash 的版本信息并成功退出。

参数(ARGUMENTS)

如果选项处理之后仍有参数剩余，并且没有指定 -c 或 -s 选项，第一个参数 将假 定 为一个包含 shell 命令的文件的名字。如果 bash 是以这种方式启动的，$0 将设置为这个文件的名字，位置参数将设置为剩余的其他参数。 Bash 从这个文 件中读取并执行命令，然后退出。 Bash 的退出状态是脚本中执行的最后一个命令的退出状态。如果没有执行命令，退出状态是 0。尝试的步骤是先试图打开在当 前目录中的这个文件，接下来，如果没有找到，shell 将搜索脚本的 PATH 环境变量中的路径。

启动(INVOCATION)

login shell 登录 shell，参数零的第一个字符是 - ，或者 启 动 时 指 定 了 --login 选项的 shell。

interactive 交互的 shell，是一个启动时没有指定非选项的参数，并且没有指定 -c 选项，标准输出和标准输入都连接到了终端 (在 isatty(3) 中 判 定) 的 shell ，或者启动时指定了 -i 选项的 shell。如果 bash 是交互的， PS1 环境变量将被设置，并且 $- 包含 i ，允许一个 shell 脚本或者一个启动文件 来检测这一状态。

下 列段落描述了 bash 如何执行它的启动文件。如果这些启动文件中的任一个存在但是不可读取， bash 将报告一个错误。文件名中的波浪号 (~,tilde) 将 像 EXPANSION 章节中 Tilde Expansion 段描述的那样展开。

当 bash 是作为交互的登录 shell 启动的，或者是一个非交互的 shell 但是指定了 --login 选项，它首先读取并执行 /etc/profile 中的命令，只要那个文件存 在 。 读 取 那 个 文 件 之 后 ， 它以如下的顺序查找 ~/.bash_profile, ~/.bash_login, 和 ~/.profile, 从存在并且可读的第一个文件中读取并执行 其

中的命令。 --noprofile 选项可以用来在 shell 启动时阻止它这样做。

当一个登录 shell 退出时， bash 读取并执行文件 ~/.bash_logout 中的命令，只要它存在。

当一个交互的 shell 但不是登录 shell 启动时， bash 从文件 ~/.bashrc 中读取 并 执行命令，只要它存在。可以用 --norc 选项来阻止它这样做。 --rcfile file 选项将强制 bash 读取并执行文件 file 中的命令，而不是 ~/.bashrc 中的。

当 bash 以非交互的方式启动时，例如在运行一个 shell 脚本时，它在环境中查找变量 BASH_ENV ，如果它存在则将它的值展开，使用展开的值作为一个文件 的名称，读取并执行。 Bash 运作的过程就如同执行了下列命令：

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

但是没有使用 PATH 变量的值来搜索那个文件名。

如 果 bash 以名称 sh 启动，它试图模仿 (mimic) sh 历史版本的启动过程，尽可能地相似，同时也遵循 POSIX 标准。当作为交互式登录 shell 启动时,或者是非 交 互 但 使 用 了 --login 选项启动的时候,它首先尝试读取并执行文件 /etc/profile 和 ~/.profile, 中的命令. 选项 --noprofile 用于避免这种 行为.当使用命令 sh 来启动一个交互式的 shell 时, bash 查找环境变量 ENV, 如果有定义的话就扩展它的值,然后使用扩展后的值作为要读取和执行的文件 的 名称. 由 于 使 用 sh 启动的 shell 不会读取和执行任何其他的启动文件,选项 --rcfile 没有意义.使用名称 sh 启动的非交互的 shell 不会读取任何其他启动文件.当以 sh 启动时, bash 在读取启动文件之后进入 posix 模式.

当 bash 以 posix 模式启动时,(和使用 --posix 命令行参数效果相同),它遵循 POSIX 标准. 这种模式下,交互式 shell 扩展 ENV 环境变量的值,读取并执行以扩展后值为文件名的配置文件. 不会读取其他文件.

Bash 试着检测它是不是由远程 shell 守护程序,通常为 rshd 启动的.如果 bash 发现它是由 rshd 启动的,它将读取并执行 ~/.bashrc 文件中的命令, 只要这 个文件存在并且可读.如果以 sh 命令启动,它不会这样做. 选项 --norc 可以用来阻止这种行为,选项 --rcfile 用来强制读取另一个文件,但是通常 rshd 不会 允许它们, 或者用它们来启动 shell.

如果 shell 是以与真实用户(组) id 不同的有效用户(组) id 来启动的, 并且没有 - 选项,那么它不会读取启动文件, 也不会从环境中继承 shell 函数. 环境变量中如果出现 SHELLOPTS，它将被忽略.有效用户 id 将设置为真实用户 id. 如果 启 动 时 给 出 了 -

选项,那么启动时的行为是类似的, 但是不会重置有效用户 id.

定义(DEFINITIONS)

下列定义在文档余下部分中通用.

blank 空白

一个空格或是 tab .

word 词

一个字符序列, shell 将它们视为一个结构单元. 也称为一个 token 片段。

name 名称

一个只由字母,数字和下划线构成的词,并且以字符或下划线起始. 也称为一个 word identifier 标识符.

metacharacter 元字符

一个字符, 如果不是引用的话, 将成为词的分隔符. 它是如下字符之一:

| & ; ( ) < > space tab

control operator 控制操作符

一个 token(标识), 拥有控制功能. 它是如下符号之一:

|| & && ; ;; ( ) | <newline>

保留字("RESERVED WORDS")

Reserved words(保留字) 是对 shell 有特殊意义的词. 下列词被识别为保留的, 如果不是引用, 并且不是一个简单命令的起始词 (参见 下 面 的 shell 语法("SHELL GRAMMAR")), 也不是 case 或者 for 命令的第三个词:

! case do done elif else esac fi for function if in select then until while { } time [[ ]]

shell 语法("SHELL GRAMMAR")

Simple Commands 简单命令

simple command(简单命令) 是(可选的)一系列变量赋值, 紧接着是 blank(空格)分隔的词和重定向, 然后以一个 control operator 结束. 第一个词指明了要执行的命令, 它被作为第 0 个参数. 其余词被作为这个命令的参数.

simple command 简单命令的返回值是它的退出状态, 或是 128+n, 如果命 令 被 signal(信号) n 结束的话.

Pipelines 管道

pipeline(管道) 是一个或多个命令的序列，用字符 | 分隔。管道的格式是这样:

[time [-p]] [!] command [| command2 ...]

命令 command 的标准输出通过管道连接到命令 command2 的标准输入。连接是在命令指定的任何重定向之前进行的(参见下面的 REDIRECTION 重定向)。

如 果保留字 ! 作为管道前缀，管道的退出状态将是最后一个命令的退出状态的逻辑非值。否则，管道的退出状态就是最后一个命令的。 shell 在返回退出状态值之前，等待管道中的所有命令返回。

如 果保留字 time 作为管道前缀，管道中止后将给出执行管道耗费的用户和系统时间。选项 -p 将使输出符合 POSIX 指定的格式。环境变量 TIMEFORMAT 可以设置 为 一 个格式字符串，指定时间信息应当如何显示；参见下面的 Shell Variables 环境变量中 TIMEFORMAT 的讲述。

管道中的每个命令都作为单独的进程来执行(即，在一个子 shell 中启动)。

Lists 序列

list(序列)是一个或多个管道，用操作符 ;, &, &&, 或 | | 分隔的序列, 并且可以选择用 ;, &, 或 <newline>新行符结束.

这些序列操作符中， && 和 | | 优先级相同，其次是 ; 和 &, 它们的优先级是相同的。

序列中可以有一个或多个新行符来分隔命令，而不是使用分号分隔。

如果一个命令是由控制操作符 & 结束的, shell 将在后台的子 shell 中执行 这个 命令。 shell 不会等待命令执行结束，返回状态总是 0。以分号 ; 分隔的命令会被顺序执行；shell 会等待每个命令依次结束。返回状态是最后执行的命 令的返回状态。

控制操作符 && 和 | | 分别代表 AND 和 OR 序列。一个 AND 序列的形式是

```
command1 && command2
```

command2 只有在 command1 返回 0 时才被执行。

一个 OR 序列的形式是

```
command1 | | command2
```

command2 只有在 command1 返回非 0 状态时才被执行。AND 和 OR 序列的返回状态是序列中最后执行的命令的返回状态。

Compound Commands 复合命令

compound command(复合命令) 是如下情况之一：

(list) list 序列将在一个子 shell 中执行。变量赋值和影响 shell 环境变 量的内建命令在命令结束后不会再起作用。返回值是序列的返回值。

{ list; }

list 序列将在当前 shell 环境中执行。序列必须以一个新行符或分号结束。这种做法也称为 group command(命令组)。返回值是序列的返回值。注意与元字符 (和 不同， { 和 } 是 reserved words(保留字)，必须出现在能够识别保留字的场合。由于它们不会产生断 词(cause a word break)，它们和序列之间必须用空格分开。

((expression))

表 达 式 expression 将被求值。求值规则在下面的 算术求值 (ARITHMETIC EVALUATION) 章节中描述。如果表达式的值非零，返回值就是 0；否则返回值是 1。这种做法和 let "expression" 等价。

[[expression]]

返 回 0 或 1，取决于条件表达式 expression 求值的情况。表达式是由下面 CONDITIONAL EXPRESSIONS 条件表达式章节中描 述 的 原 语(primaries) 组成。 [[和]] 中的词不会进行词的拆分和路径的扩展处理；而 tilde 扩展，参数和变量扩展，算术扩展，命令替换，函数替换和引用的去除则都将进行。

当使用 == 和 != 操作符时，操作符右边的字符串被认为是一个模式，根据下面 Pattern Matching(模式匹配) 章节中的规则进行匹配。如果匹配则返回值是 0，否则返回 1。模式的任何部分可以被引用，强制使它作为一个字符串而被匹配。

表达式可以用下列操作符结合起来。根据优先级的降序列出如下：

(expression)

返回表达式 expression 的值。括号可以用来提升操作符的优 先级。

! expression

返回真，如果表达式 expression 返回假。

expression1 && expression2

返回真，如果表达式 expression1 和 expression2 都返回真。

expression1 || expression2

返 回真，如果表达式 expression1 或者 expression2 二者之一返回真。

&&(与) 和 || 操作符不会对表达式 expression2 求值，如 果 expression1 可以决定整个条件表达式的返回值的话。

for name [in word] ; do list ; done

in 之后的一系列词会被扩展，产生一个项目列表。变量 name 被依次赋以这个列表中的每个元素，序列 list 每次都被执行。如果 in word 被忽略，那么 for 命令遍历已设置的位置参数(positional parameter，参见下面的 PARAMETERS 参数)，为每一个执行一次序列 list。返回值是最后一个命令的返回值。如果 in 之后的词扩展的结果是空列表，就不会执行任何命令，返回值是 0。

for ((expr1 ; expr2 ; expr3)) ; do list ; done

首先，算术表达式 expr1 被根据下面 算术求值 (ARITHMETIC EVALUATION) 中的规则进行求值。然后算术表达式 expr2 被循环求值，直到它等于 0。每次 expr2 结果非零时，序列 list 都被执行，算术 表 达 式 expr3 被求值。如果任何表达式被忽略，将被视为执行结果是 1。返回值是序列 list 中被执行的最后一个命令的返回值；或者是 false，如果任何表达式非法的话。

select name [in word] ; do list ; done

in 之后的一系列词会被扩展，产生一个项目列表。这个扩展后的词集合被输出到标准错误上，每个前面加上一个数字。如果 in word 被忽略，将输出位置参数 (参见下面的 PARAMETERS 参数章节)。 PS3 提示符将被显示出来，等待从标准输入得到一行输入。如果输入是一个数字且显示中有对应的词，那么变量 name 的值将设置为这个词。如果输入一个空行，那么词和提示符将再次显示出来。如果读入了一个 EOF，命令就结束。任何 其 他 值将设置变量 name 为空。读入的行保存为变量 REPLY. 序列 list 在每次选择之后都会执行，直到执行了一个 break 命令。 select 的退出状态是序列 list 中执行的最后一个命令的退出状态，如果没有执行命令就是 0。

case word in [[(] pattern [| pattern] ...) list ;;] ... esac

case 命令首先扩展 word, 然后依次试着用每个 pattern 来匹配它， 使用 与路径扩展相同的匹配规则(参见下面的 Pathname Expansion 路径扩

展章节)。如果找到一个匹配，相应的序列将被执行。找到一个匹配之 后，不会再尝试其后的匹配。如果没有模式可以匹配，返回值是 0。否则，返回序列中最后执行的命令的返回值。

if list; then list; [elif list; then list;] ... [else list;] fi

序列 if list 被执行。如果退出状态是 0，then list 将被执行。否 则，每个 elif 将被一次执行，如果退出状态是 0，相应的 then list 将被执行，命令结束。否则，else list 将被执行，如果存在的话。退出状态是最后执行的命令的退出状态，或者是 0，如果所有条件都不满足。

while list; do list; done

until list; do list; done

while 命令不断地执行序列 do list，直到序列中最后一个命令返回 0。until 命令和 while 命令等价，除了对条件的测试恰好相反；序 列 do list 执行直到序列中最后一个命令返回非零状态值。 while 和 until 命令的退出状态是序列 do list 中最后一个命令的退出状态，或者是 0，如果没有执行任何命令。

[function] name () { list; }

这样可以定义一个名为 name 的函数。函数体 body 是包含在 { 和 } 之间的命令序列 list。在指定将 name 作为一个命令运行的场合，这个 序列 将被执行。函数的退出状态是函数体最后执行的命令的退出状态(参见下面的 FUNCTIONS 函数章节)。

注释(COMMENTS)

在非交互的 shell 中或者使用内建命令 shopt 启用 了 interactive_comments 选 项的交互的 shell 中，以 # 起始的词使得这个词和所有同一行上所有剩余的字符都被忽略。没有启用 interactive_comments 选项的交互式 shell 不允许出现 注释。这个选项在交互式 shell 中是默认启用的 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

引用(QUOTING)

引用 Quoting 用来去掉特定字符或词的特殊意义。引用可以用来禁止对特殊字符的处理，阻止保留字被识别，还用来阻止参数的扩展。

上 面在 DEFINITIONS 定义中列出的每个元字符 metacharacters 对于 shell 都有特殊意义。如果要表达它的本义，必须引用它。

在使用命令行历史扩展功能时，history expansion 字符，通常是 !，必须被 引用，才不会进行历史扩展。

有三种引用机制：转义字符 (escape character), 单引号和双引号。

一 个未被引用的反斜杠 (\) 是转义字符 escape character。它保留其后下一个字符的字面意义，除非那是一个新行符。如果 \ 和新行符成对出现，并且反斜杠自 身没有被引用，那么 \<newline> 被视为续行标志 (意思是，它被从输入流中删除并忽略了)。

将字符放在单引号之中，将保留引用中所有字符的字面意义。单引号不能包含 在单引号引用之中，即使前面加上了反斜杠。

将字符放在双引号中，同样保留所有字符的字面意义，例外的情况是 $, ‘, 和 \ 。字符 $ 和 ‘ 在双引号中仍然具有特殊意义。反斜杠只有后面是下列字符时 才有 特殊意义： $, ‘, ", \, 或 <newline>. 双引号可以包含在双引号引用中，但要在前面加上一个反斜杠。

特殊的参数 * 和 @ 在双引号中有特殊意义(参见下面的 PARAMETERS 参数章 节) 。

形 式为 $’string’ 的词会被特殊处理。它被扩展为 string，其中的反斜杠转义字符被替换为 ANSI C 标准中规定的字符。反斜杠转义序列，如果存在的话， 将做如下转换：

\a alert (bell) 响铃
\b backspace 回退
\e an escape character 字符 Esc
\f form feed 进纸
\n new line 新行符
\r carriage return 回车
\t horizontal tab 水平跳格
\v vertical tab 竖直跳格
\\ backslash 反斜杠
\’ single quote 单引号
\nnn 一个八比特字符，它的值是八进制值 nnn (一到三个数字)。
\xHH 一个八比特字符，它的值是十六进制值 HH (一到两个十六进制数字)。
\cx 一个 ctrl-x 字符

扩展结果是单引号引用的，就好像 $ 符号不存在一样。

双引号引用字符串前面加上一个 $ 符号将使得这个字符串被根据当前语言 环 境

(locale) 来翻译。如果当前语言环境是 C 或者 POSIX，这个符号将被忽略。如果这个字符串被翻译并替换了，那么替换结果是双引号引用的。

参数(PARAMETERS)

一个参数 parameter 是一个储存值的实体。它可以是一个名称 name, 一个数字或者是下面 Special Parameters 特殊参数章节中列出的特殊字符之一。从 shell 的角度来看，一个变量 variable 是一个由名称 name 代表的参数。一个变量有一个值 value 以及零个或多个属性 attibutes。属性可以使用内建命令 declare 来设置(参见下面 shell 内建命令(SHELL BUILTIN COMMANDS) 章节中对 declare 的描述)。

如果给一个参数赋值，那么它就被定义了。空字符串是有效的值。一旦一个变量被定义了，它只能用内建命令 unset 来取消(参见下面 shell 内建命令(SHELL BUILTIN COMMANDS) 章节).

一个变量 variable 可以用这样的语句形式来赋值：

```
name=[value]
```

如果没有给出值 value，变量就被赋为空字符串。所有值 values 都经过了波浪线扩展，参数和变量扩展，命令替换，算术扩展和引用的删除(参见下面的 EXPANSION 扩展章节)。如果变量设置了 integer 整数属性，那么值 value 将进行算术扩展，即使没有应用 $((...)) 扩展 (参见下面的 Arithmetic Expansion 算术扩展章节)。不会进行词的拆分，除非是下面 Special Parameters 特殊参数中提到的 "$@"。不会进行路径的扩展。赋值语句也出现在下列内建命令中，作为它们的参数： declare, typeset, export, readonly, 和 local 。

Positional Parameters 位置参数

位置参数 positional parameter 是以一或多个数字代表的参数，除了 0。位置参数是在 shell 启动时，根据它的参数来赋值的，也可以用内建命令 set 来重新赋值。位置参数不能用赋值语句来赋值。在一个 shell 函数被执行的时候，位置参数会被暂时地替换掉 (参见下面的 FUNCTIONS 函数章节)。

当位置参数由两个以上的数字构成时，它必须放在括号内 (参见下面的 EXPANSION 扩展章节)。

Special Parameters 特殊参数

shell 对一些参数做特殊处理。这些参数只能被引用而不能被赋值。

* 扩展为位置参数，从 1 开始。如果扩展发生在双引号中，它扩展为一个词，值是各个参数，以特殊变量 IFS 的第一个字符分隔。也就是说

，"$*" 等价于 "$1c$2c..."，这里 c 是变量 IFS 的第一个字符。如果没有设置 IFS，那么参数将用空格分隔。 IFS

@ 扩展为位置参数，从 1 开始。如果扩展发生在双引号中，每个参数都 将扩展为一个词。也就是说， "$@" 等价于 "$1" "$2" ... 如果位置参数不存在，"$@" 和 $@ 扩展为空 (即，它们被删除了)。

扩展为位置参数的个数，以十进制表示。

? 扩展为最近执行的前台管道的状态。

- 扩展为当前选项标志。标志是在启动时或以内建命令 set 指定的，或 者是 shell 自身设置的 (例如选项 -i)。

$ 扩展为 shell 的进程 ID。在一个 () 子 shell 中，它扩展为当前 shell 的进程 ID 而不是子 shell 的。

! 扩展为最近一次执行的后台 (异步) 命令的进程号。

0 扩展为 shell 或者 shell 脚本的名称。这个变量是在 shell 初始化 时设置的。如果 bash 是执行脚本文件时启动的， $0 将设置为那个文件的名称。如果 bash 启动时的参数包含 -c，那么 $0 被设置为启动命令 行被 执行后的第一个参数，如果有的话。否则，它被设置为用来启动 bash 的文件名，就是参数 0。

_ shell 启动时，设置为 shell 或参数中被执行的 shell 脚本的绝对路径名。然后，在扩展时扩展为上一个命令的最后一个参数。它也被设置为被执行的每个命令的文件全名并且被设置到这个命令执行的环境当中。当检查邮件时，这个参数保存着正在检查的邮件文件的名称。

Shell Variables 变量

shell 定义了下列变量：

BASH 扩展为用来启动当前 bash 实例的文件全名。

BASH_VERSINFO

一个只读数组变量，成员保存着当前 bash 实例的版本信息。赋予数组元素的值是如下这些：

BASH_VERSINFO[0]	主版本号 (release).
BASH_VERSINFO[1]	次版本号 (version).
BASH_VERSINFO[2]	补丁版本
BASH_VERSINFO[3]	编译信息
BASH_VERSINFO[4]	发布时的状态 (例如, beta1).
BASH_VERSINFO[5]	MACHTYPE 平台类型

BASH_VERSION

扩展为一个字符串，描述了这个 bash. 实例的版本。

COMP_CWORD

${COMP_WORDS} 的索引，指向当前光标位置所在的词。这个变量只有在被可 编程补全功能 (参见下面的 Programmable Completion 章节) 调用的 shell 函数中才可用。

COMP_LINE
当前命令行。这个变量只有在被命令补全功能调用的 shell 函数和外 部命令中才可用。

COMP_POINT
相对于当前命令起始处的当前光标位置。如果当前光标位置是当前命令的末端，它的值就和 ${#COMP_LINE} 相等。这个变量只有在被命令补全 功能调用的 shell 函数和外部命令中才可用。

COMP_WORDS
一个数组变量 (参见下面的 Arrays(数组)一节)，由当前命令行的各个单词构成。这个变量只有在被命令补全功能调用的 shell 函数中才可用。

DIRSTACK
一个数组变量，包含当前目录栈的内容。栈中的目录排列的顺序就是用内建命令 dirs 显示时的顺序。对这个数组变量的成员赋值可以用来修改栈中已有的目录，但是要添加和删除目录就必须使用内建命令 pushd 和 popd。对它赋值不会改变当前目录。如果取消了 DIRSTACK 的定义，它就失去了它的特殊意义，即使后来重新定义它。

EUID 扩展为当前用户的有效用户 ID。它在 shell 启动时设置。它是只读的。

FUNCNAME
当前执行的 shell 函数名。这个变量只有在执行一个 shell 函数时存在。向 FUNCNAME 赋值没有效果并且返回一个错误。如果取消了 FUNCNAME 的定义，它就失去了特殊的意义，即使后来重新定义它。

GROUPS 一个数组变量，包含当前用户所属的组的列表。向 GROUPS 赋值没有效果并且返回一个错误。如果取消了 GROUPS 的定义，它就失去了特殊的意义，即使后来重新定义它。

HISTCMD
当 前命令的历史编号，或者历史列表中的索引。如果取消了 HISTCMD 的定义，它就失去了特殊的意义，即使后来重新定义它。

HOSTNAME

自动设置为当前的主机名。

HOSTTYPE
自动设置为一个字符串，唯一地标识着正在运行 bash 的机器类型。默认值是系统相关的。

LINENO 每次引用这个参数时，shell 将它替换为一个指示在脚本或函数中当前行号的十进制数字(从 1 开始)。如果不是在脚本或函数中，替换得到的 值不一定有意义。如果取消了 LINENO 的定义，它就失去了特殊的意义，即使后来重新定义它。

MACHTYPE
自动设置为一个字符串，完整的描述了正在运行 bash 的系统类型，格式是标准的 GNU cpu-company-system 格式。默认值是系统相关的。

OLDPWD 上一次命令 cd 设置的工作目录。

OPTARG 内建命令 getopts 处理的最后一个选项参数值 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

OPTIND 内建命令 getopts 将处理的下一个参数的索引 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

OSTYPE 自动设置的一个字符串，描述了正在运行 bash 的操作系统。默认值是系统相关的。

PIPESTATUS
一个数组变量 (参见下面的 Arrays 数组章节)，包含最近执行的前台 管道中的进程(可能只包含一个命令)的退出状态。

PPID shell 的父进程的进程号。这个变量是只读的。

PWD 由 cd 命令设置的当前工作目录。

RANDOM 每次引用这个参数时，都会产生一个 0 到 32767 之间的随机整数。可以通过向 RANDOM 赋值来初始化随机数序列。如果取消了 RANDOM 的定义，它就失去了特殊的意义，即使后来重新定义它。

REPLY 变量的值将作为内建命令 read 的输入，如果命令没有参数的话。

SECONDS

每 次引用这个参数时，返回 shell 自运行以来的秒数。如果向 SECONDS 赋值，此后对它的引用将返回自赋值时起的秒数加上所赋予的值。如果取消 SECONDS 的定义，它就失去了特殊的意义，即使后来重新定义它。

SHELLOPTS

一 个冒号分隔的被允许的 shell 选项列表。列表中每个词都是内置命令 set 的 -o 选项的有效参数。 SHELLOPTS 中出现的选项也是 set -o 显示为 on 的选项。如果 bash 启动时从环境中找到这个变量，那么在读取任何配置文件之前，列表中的每个选项都将被设置。这个变量是只读的。

SHLVL 每次启动一个 bash 的实例时都会增加。

UID 扩展为当前用户的 ID，在启动时初始化。这个变量是只读的。

下列变量被 shell 使用。有时 bash 会为变量赋默认值；这些情况在下面会标出。

BASH_ENV

如果 bash 在执行一个 shell 脚本时设定了这个变量，它的值将被解 释为一个文件名，包含着初始化 shell 用到的命令，就像 ~/.bashrc 中一样。 BASH_ENV 的值在被解释为一个文件名之前要经过参数扩展，命令替换和算术扩展。不会使用 PATH 来查找结果文件名。

CDPATH 命 令 cd 的搜索路径。这是一个冒号分隔的目录列表，shell 从中查找 cd 命令的目标目录。可以是这样： ".:~:/usr".

COLUMNS

用在内建命令 select 当中，用来判断输出选择列表时的终端宽度。自动根据 SIGWINCH 信号来设置。

COMPREPLY

一 个数组变量，bash 从中读取可能的命令补全。它是由命令补全功能调用的 shell 函数产生的。

FCEDIT 内建命令 fc 默认的编辑器。

FIGNORE

一个冒号分隔的后缀名列表，在进行文件名补全时被忽略 (参见下 面 的 READLINE 章节)。一个后缀满足其中之一的文件名被排除在匹配的文件名之外。可以是这样： ".o:~".

GLOBIGNORE

一个冒号分隔的模式列表，定义了路径名扩展时要忽略的文件名集合。如果一个文件名与路径扩展模式匹配，同时匹配 GLOBIGNORE 中的一个模式时，它被从匹配列表中删除。

HISTCONTROL

如果设置为 ignorespace, 以 space 开头的行将不会插入到历史列表 中。 如果设置为 ignoredups, 匹配上一次历史记录的行将不会插入。设置为 ignoreboth 会结合这两种选项。如果没有定义，或者设置为其他值，所有解释器读取的行都将存入历史列表，但还要经过 HISTIGNORE 处理。这个变量的作用可以被 HISTIGNORE 替代。多行的组合命令的第二和其余行都不会被检测，不管 HISTCONTROL 是什么，都会加入到历史中。

HISTFILE

保存命令历史的文件名 (参见下面的 HISTORY 历史章节)。默认值是 ~/.bash_history。如果取消定义，在交互式 shell 退出时命令历史将不会保存。

HISTFILESIZE

历史文件中包含的最大行数。当为这个变量赋值时，如果需要的话，历史文件将被截断来容纳不超过这个值的行。默认值是 500。历史文件在交互式 shell 退出时也会被截断到这个值。

HISTIGNORE

一个冒号分隔的模式列表，用来判断那个命令行应当保存在历史列表中。每个模式都定位于行首，必须匹配整行 (没有假定添加 ‘*’)。在 HIST-CONTROL 指定的测试结束后，这里的每个模式都要被测试。除了平常的 shell 模式匹配字符， ‘&’ 匹配上一个历史行。‘&’ 可以使用反斜杠 来转义；反斜杠在尝试匹配之前将被删除。多行的组合命令的第二行以及后续行都不会被测试，不管 HISTIGNORE 是什么，都将加入到历史中。

HISTSIZE

命令历史中保存的历史数量 (参见下面的 HISTORY 历史章节)。默认值是 500。

HOME 当前用户的个人目录；内建命令 cd 的默认参数。在执行波浪线扩展时也用到这个变量。

HOSTFILE

包含一个格式和 /etc/hosts 相同的文件名，当 shell 需要补全主机 名时要读取它。shell 运行过程中可以改变可能的主机名补全列表；改变之后下一次需要主机名补全时 bash 会将新文件的内容添加到旧列表中。如果 定义了 HOSTFILE 但是没有赋值，bash 将尝试读取 /etc/hosts 文件来获得可能的主机名补全列表。当取消 HOSTFILE 的定义时，主机名列表将清空。

IFS 内部字段分隔符 Internal Field Separator 用来在扩展之后进行分词，使用内部命令 read 将行划分成词。默认值是 ‘‘<space><tab><newline>’’。

IGNOREEOF

控制交互式 shell 接受到唯一一个 EOF 字符时的行为。如果有定义，值是需要在一行的开始连续输入 EOF 字符，直到可以使 bash 退出的字 符

个数。如果这个变量存在，但是值不是一个数字或者没有赋值，默认值是10。如果变量没有定义， EOF 标志着输入的结束。

INPUTRC

readline 的启动配置文件，而不是默认的 ~/.inputrc (参见下面的 READLINE 章节)。

LANG 用来决定没有特地用 LC_ 变量指定的语言环境项。

LC_ALL 这个变量超越了 LANG 和所有其他指定语言环境项的 LC_ 变量。

LC_COLLATE

这个变量决定了为路径扩展的结果排序时的字母顺序，决定了范围表达式的行为，等价类，和路径扩展中的归并顺序以及模式匹配。

LC_CTYPE

这个变量决定了字符的解释和路径扩展以及模式匹配中字符类的行为。

LC_MESSAGES

这个变量决定了翻译以 $ 前导的双引号字符串时的语言环境。

LC_NUMERIC

这个变量决定了格式化数字时的语言环境分类。

LINES 内建命令 select 用它来判断输出选择列表时的列宽度。在收到 SIGWINCH 信号时自动设置。

MAIL 如果这个参数设置为一个文件名，并且没有设置环境变量 MAILPATH 的话， bash 将在这个文件中通知用户有邮件到达。

MAILCHECK

指定 bash 检查邮件的频率是多少，以秒为单位。默认值是 60 秒。需要检查邮件的时候，shell 在显示提示符之前将进行检查。如果取消它的定义，或者设置为并非大于等于零的数值，shell 将禁止邮件检查。

MAILPATH

一个冒号分隔的文件名列表，从中检查邮件。当邮件到达某个特殊文件中时，输出的特定消息可以通过将文件名与消息以 ‘?’ 分隔来指定。在消息的文本中，$_ 扩展为当前邮件文件的文件名。例如：

MAILPATH=’/var/mail/bfox?"You have mail":~/shell-mail?"$_ has mail!"’

Bash 为这个变量提供默认值，但是它使用的用户邮件文件的位置是系统相关的 (例如，/var/mail/$USER)。

OPTERR 如果设置为 1， bash 显示内建命令 getopts 产生的错误消息 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。每次 shell 启动时或者一个 shell 脚本被执行时 OPTERR 被初始化为 1。

PATH 搜索命令的路径。它是一个冒号分割的目录列表，shell 从中搜索命令 (参见下面的 命令执行(COMMAND EXECUTION) 段落)。默认的路径是系统相关的，是由安装 bash 的系统管理员设置的。通常它的值是 ‘‘/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:.’’ 。

POSIXLY_CORRECT

如果 bash 启动环境中有这个变量，它将在读取启动配置文件之前进入 posix mode，就好像提供了 --posix 启动参数一样。如果 shell 运行过程中设置了它，bash 就启用 posix mode，就好像执行了 set -o posix 命令一样。

PROMPT_COMMAND

如果有定义，它的值将作为一个命令，每次显示主提示符之前都会执行。

PS1 这个参数的值被扩展 (参见下面的 PROMPTING 提示符段落)，用作主提示符字符串。默认值是 ‘‘\s-\v\$ ’’ 。

PS2 这个参数的值同 PS1 一起被扩展，用作次提示符字符串。默认值是 ‘‘> ’’ 。

PS3 这个参数的值被用作内建命令 select 的提示符 (参见上面的 SHELL GRAMMAR 语法章节)。

PS4 这个参数的值同 PS1 一起被扩展，在执行跟踪中在 bash 显示每个命令之前显示。需要的话， PS4 的第一个字符会被复制多次，来指示 indirection 的层数。默认值是 ‘‘+ ’’ 。

TIMEFORMAT

在前缀 time 保留字的管道中，这个参数的值用作格式字符串，指定计时信息如何显示。字符 % 引入的转义序列，被扩展为时间值或其他信息。转义序列和它们的含义如下所示；括号中是可选的成分。

%% 一个字面上的 %。

%[p][l]R 经历的时间，以秒计算。

%[p][l]U CPU 在用户模式下执行的秒数。

%[p][l]S CPU 在系统模式下执行的秒数。

%P CPU 使用率，算法是 (%U + %S) / %R。

可选的 p 是指定精度 (小数点后数字位数) 的数值。如果是 0 就不输出小数点或小数值。最多指定到小数点后三位；如果 p 大于 3 就会被改为 3。如果没有指定 p，默认使用 3。

可选的 l 指定了长格式，包含分钟，格式是 MMmSS.FFs。 p 的值决定了是不是包含小数位。

如果没有设置这个值，bash 假定它的值是 $’\nreal\t%3lR\nuser\t%3lU\nsys%3lS’ 。如果它是空值，就不会显示计时信息。显示格式字符串的时候，会加上一个前导的新行符。

TMOUT 如果设置为大于 0 的值，TMOUT 被当作内建命令 read 的默认超时等待时间。如果等待终端输入时， TMOUT 秒之后仍然没有输入， select 命令将终止。在交互的 shell 中，它的值被解释为显示了主提示符之后等待输入的秒数。如果经过这个秒数之后仍然没有输入， Bash 将退出。

auto_resume

这个变量控制了 shell 如何与用户和作业控制交互。如果设置了这个 变量，一个不包含重定向的单个词的简单命令，将作为恢复被中断的作业的指示。不允许出现模棱两可的情况；如果有多个作业都以这个词起始，将恢复最近运行的作业。在这种情形下，被中断的作业的 name 是用于启动它的命令行。如果值设置为 exact，给出的字符串必须精确匹配被中断的作业名；如果设置为 substring，给出的字符串需要匹配被中断的作业名的子串。值 substring 的功能与作业标识符 %? 功能类似 (参见下面的 JOB CONTROL 作业控制章节)。如果设置为任何其他值，给出的字符串必须是被中断的作业的前缀；这样做与作业标识符 % 功能类似。

histchars

两到三个字符，控制着历史扩展和分段 (tokenization，参 见 下 面 的 HISTORY EXPANSION 历史扩展章节)。第一个字符是 history expansion(历史扩展) 字符，这个字符表明了历史扩展的开始，通常是 ‘!’。第二个字符是 quick substitution(快速替换) 字符，它是重新运行上次输入的命令，但将命令中的字符串替换为另一个的简写，默认是 ‘^’。可选的第三个字符是指示如果作为一个词的开始，那么一行中剩余字符是注释。通常这个字符是 ‘#’。历史注释字符使得对一行中剩余字符在历史替换中被跳过。它不一定使 shell 解释器将这一行的剩余部分当作注释。

Arrays

Bash 提供了一维数组变量。任何变量都可以作为一个数组；内建命令 declare 可以显式地定义数组。数组的大小没有上限，也没有限制在连续对成员引用和 赋值时有什么要求。数组以整数为下标，从 0 开始。

如 果 变 量赋值时使用语法 name[subscript]=value，那么就会自动创建数组。subscript 被当作一个算术表达式，结果必须是大于等于 0 的值。要显式地定义一个数组，使用 declare -a name (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。也可以用 declare -a name[subscript] 这时 subscript 被忽略。数组变量的属性可以用内建命令 declare 和 readonly 来指定。每个属性对于所有数组元素都有效。

数组赋值可以使用复合赋值的方式，形式是 name=(value1 ... valuen)，这里每个 value 的形式都是 [subscript]=string。string 必须出现。如果出现了可选的括号和下标，将为这个下标赋值，否则被赋值的元素的下标是语句中上一次 赋值 的下标加一。下标从 0 开始。这个语法也被内建命令 declare 所接受。单独的数组元素可以用上面介绍的语法 name[subscript]=value 来赋值。

数组的任何元素都可以用 ${name[subscript]} 来引用。花括号是必须的，以 避免 和路径扩展冲突。如果 subscript 是 @ 或是 *，它扩展为 name 的所有成员。这两种下标只有在双引号中才不同。在双引号中，${name[*]} 扩展为一个词，由 所有数组成员的值组成，用特殊变量 IFS 的第一个字符分隔；${name[@]} 将 name 的每个成员扩展为一个词。如果数组没有成员，${name[@]} 扩展为空串 。这 种不同类似于特殊参数 * 和 @ 的扩展 (参见上面的 Special Parameters 段落)。${#name[subscript]} 扩展为 ${name[subscript]} 的长度。 如 果 subscript 是 * 或者是 @，扩展结果是数组中元素的个数。引用没有下标数组变量等价于引用元素 0。

内建命令 unset 用于销毁数组。unset name[subscript] 将销毁下 标 是 subscript 的元素。 unset name, 这里 name 是一个数组，或者 unset name[subscript], 这里 subscript 是 * 或者是 @，将销毁整个数组。

内建命令 declare, local, 和 readonly 都能接受 -a 选项，从而指定一个数组。 内建命令 read 可以接受 -a 选项，从标准输入读入一列词来为数组赋值。内建命令 set 和 declare 使用一种可以重用为输入的格式来显示数组元素。

扩展(EXPANSION)

命令行的扩展是在拆分成词之后进行的。有七种类型的扩展： brace expansion(花 括 号 扩 展), tilde expansion(波浪线扩展), parameter and variable expansion(参数和变量扩展), command substitution(命令替 换), arithmetic expansion(算术扩展), word splitting(词的拆分), 和 pathname expansion(路径扩展).

扩展的顺序是：brace expansion, tilde expansion, parameter, variable 和 arithmetic expansion 还有 command substitution (按照从左到右的顺序), word splitting, 最后是 pathname expansion.

还有一种附加的扩展：process subtitution (进程替换) 只有在支持它的系统中有效。

只 有 brace expansion, word splitting, 和 pathname expansion 在扩展前后的词数会发生改变；其他扩展总是将一个词扩展为一个词。唯一的例外是上面 提到的 "$@" 和 "${name[@]}" (参见 PARAMETERS 参数)。

Brace Expansion

Brace expansion 是一种可能产生任意字符串的机制。这种机制类似于 pathname expansion, 但是并不需要存在相应的文件。花括号扩展的模式是一 个 可 选 的 preamble(前导字符), 后面跟着一系列逗号分隔的字符串，包含在一对花括号中

，再后面是一个可选的 postscript(附言)。前导被添加到花括号中的每个字符串前面，附言被附加到每个结果字符串之后，从左到右进行扩展。

花 括号扩展可以嵌套。扩展字符串的结果没有排序；而是保留了从左到右的顺序。例如， a{d,c,b}e 扩展为 ‘ade ace abe’ 。

花括号扩展是在任何其他扩展之前进行的，任何对其他扩展有特殊意义的字符 都保 留在结果中。它是严格字面上的。 Bash 不会对扩展的上下文或花括号中的文本做任何语义上的解释。

正确的花括号扩展必须包含没有引用的左括号和右括号，以及至少一个没有引 用的 逗号。任何不正确的表达式都不会被改变。可以用反斜杠来引用 { 或 , 来阻止将它们识别为花括号表达式的一部分。为了避免与参数扩展冲突，字符 串 ${ 不被认为有效的组合。

这种结构通常用来简写字符串的公共前缀远比上例中为长的情况，例如：

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

或者：

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

花 括号扩展导致了与历史版本的 sh 的一点不兼容。在左括号或右括号作为词的一部分出现时， sh 不会对它们进行特殊处理，会在输出中保留它们。 Bash 将括号从花括号扩展结果的词中删除。例如，向 sh 输入 file{1,2} 会导致不变的输出。同样的输入在 bash 进行扩展之后，会输出 file1 file2 . 如果需要 同 sh 严格地保持兼容，需要在启动 bash 的时候使用 +B 选项，或者使用 set 命令加上 +B 选项来禁用花括号扩展 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

Tilde Expansion

如果一个词以没有引用的波浪线字符 (‘~’) 开始，所有在第一个没有引用的斜线 (‘/’) 之前的字符 (或者是这个词的所有字符，如果没有没引用的斜线的话) 都被认为是 tilde-prefix(波浪线前缀)。如果 tilde-prefix 中没有被引用的字符，那么波浪线之后的字符串被认为是 login name(登录名)。如果登录名是空字符串 ，波浪线将被替换为 shell 参数 HOME 的值。如果没有定义 HOME，将替换为执行此 shell 的用户的个人目录。否则，tilde-prefix 被替换为与指定登录 名相联系的个人目录。

如 果 tilde-prefix 是 ‘~+’ ，将使用 shell 变量 PWD 的值来替换。如果 tilde-prefix 是 ‘~-’ ，并且设置了 shell 变量 OLDPWD, 将使用这个变量值 来

替换。如果在 tilde-prefix 中，波浪线之后的字符串由一个数字 N 组成，前缀可选的 ‘+’ 或者 ‘-’，那么 tilde-prefix 将被替换为目录栈中相应的元素，就是将 tilde-prefix 作为参数执行内建命令 dirs 显示的结果。如果 tilde-prefix 中波浪线之后的字符是一个数字，没有前缀，那么就假定有一个 ‘+’。

如果登录名不合法，或者波浪线扩展失败，这个词将不会变化。

在变量赋值中，对于 : 或 = 之后的字符串会立即检查未引用的 tilde-prefix。这种情况下，仍然会进行波浪线扩展。因此，可以使用带波浪线的文件名来为 PATH, MAILPATH, 和 CDPATH 赋值，shell 将赋予扩展之后的值。

Parameter Expansion

字符 ‘$’ 引入了参数扩展，命令替换和算术扩展。要扩展的参数名或符号可能包含 在花括号中，花括号可选的，但是可以使得要扩展的变量不会与紧随其后的字符合并，成为新的名称。

使用花括号的时候，匹配的右括号是第一个 ‘}’，并且它没有被反斜杠引用或 包含 在一个引用的字符串中，也没有包含在一个嵌入的算术扩展，命令替换或是参数扩展中。

${parameter}

被替换为 parameter 的值。如果 parameter 是一个位置参数，并且数字多 于一位时；或者当紧随 parameter 之后有不属于名称一部分的字符时，都必须加上花括号。

如果 parameter 的第一个字符是一个感叹号，将引进一层间接变量。 bash 使用以 parameter 的其余部分为名的变量的值作为变量的名称；接下来新的变量被扩展，它的值用在随后的替换当中，而不是使用 parameter 自身的值。这也 称 为 indirect expansion(间接扩展). 例外情况是下面讲到的 ${!prefix*}。

下面的每种情况中，word 都要经过波浪线扩展，参数扩展，命令替换和算术扩展。如果不进行子字符串扩展，bash 测试一个没有定义或值为空的参数；忽略冒号的结果是只测试未定义的参数。

${parameter:-word}

Use Default Values(使用默认值)。如果 parameter 未定义或值为空，将替换为 word 的扩展。否则，将替换为 parameter 的值。

${parameter:=word}

Assign Default Values(赋默认值)。如果 parameter 未定义或值为空，word 的扩展将赋予 parameter. parameter 的值将被替换。位置参数和

特殊参数不能用这种方式赋值。

${parameter:?word}

Display Error if Null or Unset(显示错误，如果未定义或值为空)。如果 parameter 未定义或值为空，word (或一条信息，如果 word 不存在)的扩展将写入到标准错误；shell 如果不是交互的，则将退出。否 则 ，parameter 的值将被替换。

${parameter:+word}

Use Alternate Value(使用可选值)。如果 parameter 未定义或非空，不会进行替换；否则将替换为 word 扩展后的值。

${parameter:offset}

${parameter:offset:length}

Substring Expansion(子字符串扩展)。扩展为 parameter 的最多 length 个 字 符 ， 从 offset 指定的字符开始。如果忽略了 length，扩展为 parameter 的子字符串，从 offset 指定的字符串开始。length 和 offset 是算术表达式 (参见下面的 ARITHMETIC EVALUATION 算术求值段落) 。 length 必须是一个大于等于 0 的数值。如果 offset 求值结果小 于 0 ， 值将当作从 parameter 的值的末尾算起的偏移量。如果 parameter 是 @，结果是 length 个位置参数，从 offset 开始。如 果 parameter 是 一 个 数 组名，以 @ 或 * 索引，结果是数组的 length 个成员，从 ${parameter[offset]} 开始。子字符串的下标是从 0 开始的，除非使用位置参数时， 下标从 1 开始。

${!prefix*}

扩 展为名称以 prefix 开始的变量名，以特殊变量 IFS 的第一个字符分隔。

${#parameter}

替换为 parameter 的值的长度 (字符数目)。如果 parameter 是 * 或者是 @, 替换的值是位置参数的个数。如果 parameter 是一个数组名，下标是 * 或者是 @, 替换的值是数组中元素的个数。

${parameter#word}

${parameter##word}

word 被扩展为一个模式，就像路径扩展中一样。如果这 个 模 式 匹 配 parameter 的值的起始，那么扩展的结果是将 parameter 扩展后的值中 ，最短的匹配 (‘‘#’’ 的情况) 或者最长的匹配 (‘‘##’’ 的情况) 删除的结果。如果 parameter 是 @ 或者是 *, 则模式删除操作将依次施用于每个位置参数，最后扩展为结果的列表。如果 parameter 是一个数组变 量 ， 下标是 @ 或者是 *, 模式删除将依次施用于数组中的每个成员，最后扩展为结果的列表。

${parameter%word}
${parameter%%word}

word 被扩展为一个模式，就像路径扩展中一样。如果这 个 模 式 匹 配 parameter 扩展后的值的尾部，那么扩展的结果是将 parameter 扩展后的值中，最短的匹配 (‘ ‘%’ ’ 的情况) 或者最长的匹配 (‘ ‘%%’ ’ 的情 况) 删除的结果。如果 parameter 是 @ 或者是 *, 则模式删除操作将依次施用于每个位置参数，最后扩展为结果的列表。如果 parameter 是一个 数组 变量，下标是 @ 或者是 *, 模式删除将依次施用于数组中的每个成员，最后扩展为结果的列表。

${parameter/pattern/string}
${parameter//pattern/string}

patterm 被扩展为一个模式，就像路径扩展中一样。parameter 被扩展，其值中最长的匹配 pattern 的内容被替换为 string。在第一种形式中，只有第一个匹配被替换。第二种形式使得 pattern 中所有匹配都被替 换为 string。如果 pattern 以 # 开始，它必须匹配 parameter 扩展后值的首部。如果 pattern 以 % 开始，它必须匹配 parameter 扩展后值 的尾 部。如果 string 是空值，pattern 的匹配都将被删除， pattern 之后的 / 将被忽略。如果 parameter 是 @ 或者是 *, 则替换操作将依 次施 用于每个位置参数，最后扩展为结果的列表。如果 parameter 是一个数组变量，下标是 @ 或者是 *, 模式删除将依次施用于数组中的每个 成员，最后扩展为结果的列表。

Command Substitution

命 令替换 (Command substitution) 允许以命令的输出替换命令名。有两种形式：

$(command)

还有

‘command‘

Bash 进行扩展的步骤是执行 command，以它的标准输出替换它，并且将所有后续的 新行符删除。内嵌的新行符不会删除，但是它们可能会在词的拆分中被删除。命令替换 $(cat file) 可以用等价但是更快的方法 $(< file) 代替。

当使用旧式的反引号 (" ‘ ‘") 替换形式时，反斜杠只有其字面意义，除非后面 是 $, ‘, 或 者是 \. 第一个前面没有反斜杠的反引号将结束命令替换。当使用 $(command) 形式时，括号中所有字符组成了整个命令；没有被特殊处理的字符。

命 令替换可以嵌套。要在使用反引号形式时嵌套，可以用反斜杠来转义内层的反引号。

如果替换发生在双引号之中，结果将不进行词的拆分和路径扩展。

Arithmetic Expansion

算术扩展允许算术表达式的求值和结果的替换。算术扩展的格式是：

$((expression))

表达式 expression 被视为如同在双引号之中一样，但是括号中的双引号不会 被特 殊处理。表达式中所有词都经过了参数扩展，字符串扩展，命令替换和引用的删除。算术替换可以嵌套。

求值根据下面 算术求值 (ARITHMETIC EVALUATION) 章节中列出的规则进行。 如果表达式 expression 非法， bash 输出错误提示消息，不会进行替换。

Process Substitution

Process substitution (进程替换) 只有在支持命名管道 (FIFOs)，或者支持使用 /dev/fd 方式为打开的文件命名的系统中才可用。它的形式是 <(list) 或 者是 >(list)。进程 list 运行时的输入或输出被连接到一个 FIFO 或者 /dev/fd 中的文件。文件的名称作为一个参数被传递到当前命令，作为扩展的结果。如 果使 用 >(list) 形式，向文件写入相当于为 list 提供输入。如果使用 <(list) 形式，可以读作为参数传递的文件来获得 list 的输出。

如果可能的话，进程替换是与参数和变量扩展，命令替换和算术扩展同时发生 的。

Word Splitting

shell 检测不在双引号引用中发生的参数扩展，命令替换和算术扩展的结果，进行 word splitting(词的拆分)。

shell 将 IFS 的每个字符都作为定界符，根据这些字符来将其他扩展的结果分成词 。 如果 IFS 没有定义，或者它的值是默认的 <space><tab><newline>, 那么 IFS 字符的任何序列都将作为分界之用。如果 IFS 的值是默认之外的值，那么词开头和结尾的空白字符 space 和 tab 都将被忽略，只要空白字符在 IFS 的值之内 (即， IFS 包含空白字符)。任何在 IFS 之中但是不是 IFS 空白的字符， 以及 任何相邻的 IFS 空白字符，将字段分隔开来。 IFS 空白字符的序列也被作为分界符。如果 IFS 的值是空，不会发生词的拆分。

显式给出的空值参数 ("" 或 ’ ’) 将被保留。隐含的空值参数，来自于空值的 参数 扩展，如果没有引用则将被删除。如果空值的参数在双引号引用中扩展，结果是空值的参数，将被保留。

注意如果没有发生扩展，不会进行词的拆分。

Pathname Expansion

词的拆分之后，除非设置过 -f 选项， bash 搜索每个词，寻找字符 *, ?, 和 [. 如果找到了其中之一，那么这个词被当作一个 pattern(模式)，被替换为匹配这个模式的文件名以字母顺序排列的列表。如果没有找到匹配的文件名，并 且 shell 禁用了 nullglob 选项，这个词将不发生变化。如果设置了 nullglob 选项并且没有找到匹配，这个词将被删除。如果启用了 nocaseglob 选项，匹配 时将 不考虑字母的大小写。当模式用作路径名扩展时，字符 ‘ ‘.’ ’ 如果在一个名称的开始或者紧随一个斜杠之后，那么它必须被显式地匹配，除非设置 了 dotglob shell 选项。当匹配一个路径名时，斜杠符必须被显式地匹配。其他情况下，字符 ‘ ‘.’ ’ 不会被特殊对待。参见下面的 shell 内建命 令(SHELL BUILTIN COMMANDS) 中对 shopt 的介绍，其中有 shell 选项 nocaseglob, nullglob, 和 dotglob 的描述。

环境变量 GLOBIGNORE 可以用来限制匹配 pattern 的文件名集合。如果设 置 了 GLOBIGNORE ，每个匹配的文件名如果匹配 GLOBIGNORE 中任何一个模式的话将从匹配的列表中删除。文件名 ‘ ‘.’ ’ 和 ‘ ‘..’ ’ 总是被忽略， 即 使 设 置 了 GLOBIGNORE 。但是，设置 GLOBIGNORE 和启用 shell 选项 dotglob 效果是相同的，因此所有其他以 ‘ ‘.’ ’ 开头的文件名将被匹配。要得到原来的行为 (忽 略所 有以 ‘ ‘.’ ’ 开头的文件名)，可以将 ‘ ‘.*’ ’ 添加为 GLOBIGNORE 的模式之一。选项 dotglob 被禁用，如果 GLOBIGNORE 没有定义时。

Pattern Matching

任何模式中出现的字符，除了下面描述的特殊模式字符外，都匹配它本身。模 式中不能出现 NUL 字符。如果要匹配字面上的特殊模式字符，它必须被引用。

特殊模式字符有下述意义：

* 匹配任何字符串包含空串。
? 匹配任何单个字符。
[...] 匹配所包含的任何字符之一。用一个连字符 (‘-’) 分隔的一对字符意思是一个 range expression (范围表达式)；任何排在它们之间的字符，包含它们，都被匹配。排序使用当前语言环境的字符顺序和字符集。如果 [

之后的第一个字符是一个 ! 或是一个 ^ 那么任何不包含在内的字符 将被匹配。范围表达式中字符的顺序是由当前语言环境和环境变量 LC_COLLATE 的值 (如果设置了的话) 决定的。一个 - 只有作为集合中第一个或最 后一个字符时才能被匹配。一个] 只有是集合中第一个字符时才能被匹配。

在 [和] 中，character classes (字符类) 可以用 [:class:] 这样的语法来指定，这里 class 是在 POSIX.2 标准中定义的下列类名之一:
alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit
一个字符类匹配任何属于这一类的字符。word 字符类匹配字母，数字 和字符 _。

在 [和] 中，可以用 [=c=] 这样的语法来指定 equivalence class (等价类)。它匹配与字符 c 有相同归并权值 (collation weight，由当前语言环境定义) 的字符。

在 [和] 中，语法 [.symbol.] 匹配归并符号 (collating symbol) symbol。

如果使用内建命令 shopt 启用了 shell 选项 extglob，将识别另外几种模式 匹配 操作符。下面的描述中，pattern-list 是一个或多个模式以 | 分隔的列表。复合的模式可以使用一个或多个下列的子模式构造出来：

?(pattern-list)
匹配所给模式零次或一次出现
*(pattern-list)
匹配所给模式零次或多次出现
+(pattern-list)
匹配所给模式一次或多次出现
@(pattern-list)
准确匹配所给模式之一
!(pattern-list)
任何除了匹配所给模式之一的字串

Quote Removal
经过前面的扩展之后，所有未引用的字符 \, ’ , 以及并非上述扩展结果的字符 " 都被删除。

重定向(REDIRECTION)

在 命令执行前，它的输入和输出可能被 redirected (重定向)，使用一种 shell 可以解释的特殊记法。重定向也可以用于为当前 shell 执行环境 打开和关闭 文件。下列重定向操作符可以前置或者放在 simple command (简单命令) 之中的任何位置，或者放在 command 之后。重定向是以出现的顺序进行处理的，从左到右。

下 列描述中，如果文件描述符被忽略，并且第一个重定向操作符是 <, 那么重定向指的是标准输入 (文件描述符是 0)。如果重定向操作符的第一个字符是 >, 那么重定向指的是标准输出 (文件描述符是 1)。

下列描述中，重定向操作符之后的词如果没有特殊说明，都要经过 brace expansion, tilde expansion, parameter expansion, command substitution, arithmetic expansion, quote removal, pathname expansion, 还有 word splitting。如果扩展为多于一个词， bash 将报错。

注意重定向的顺序非常重要。例如，命令

```
ls > dirlist 2>&1
```

将标准输出和标准错误重定向到文件 dirlist, 而命令

```
ls 2>&1 > dirlist
```

只会将标准输出重定向到文件 dirlist, 因为在标准输出 被 重 定 向 到 文 件 dirlist 中之前，标准错误被复制为标准输出。

一些文件名在重定向中被 bash 特殊处理，如下表所示：

/dev/fd/fd
: 如果 fd 是一个合法的整数，文件描述符 fd 将被复制。

/dev/stdin
: 文件描述符 0 被复制。

/dev/stdout
: 文件描述符 1 被复制。

/dev/stderr
: 文件描述符 2 被复制。

/dev/tcp/host/port
: 如 果 host 是一个合法的主机名或 Internet 地址，并且 port 是一个整数端口号或服务名，bash 试图建立与相应的 socket (套接字) 的 TCP 连接。

/dev/udp/host/port

如果 host 是一个合法的主机名或 Internet 地址，并且 port 是一个整数端口号或服务名，bash 试图建立与相应的 socket (套接字) 的 UDP 连接。

打开或创建文件错误将导致重定向出错。

Redirecting Input

重定向输入使得以 word 扩展结果为名的文件被打开并通过文件描述符 n 读取，如果没有指定 n 那么就作为标准输入 (文件描述符为 0) 读取。

重定向输入的一般形式是：

[n]<word

Redirecting Output

重定向输出使得以 word 扩展结果为名的文件被打开并通过文件描述符 n 写入，如果没有指定 n 那么就作为标准输出 (文件描述符为 1) 写入。

重定向的一般形式是：

[n]>word

如果重定向操作符是 >，并且启用了内建命令 set 的 noclobber 选项，那么如果 word 扩展后得到的文件名存在并且是一个普通的文件，重定向将失败。如果重定向操作符是 >|, 或者重定向操作符是 > 并且没有启用内建命令 set 的 noclobber 选项，那么即使 word 得出的文件名存在，也会尝试进行重定向。

Appending Redirected Output (添加到重定向后的输出尾部)

这种方式的输出重定向使得以 word 扩展结果为名的文件被打开并通过文件描述符 n 从尾部添加。如果没有指定 n 就使用标准输出 (文件描述符 1)。如果文件不存在，它将被创建。

重定向的一般形式是：

[n]>>word

Redirecting Standard Output and Standard Error

Bash 允许使用这种结构将标准输出和标准错误 (文件描述符 1 和 2) 重定向到

以 word 扩展结果为名的文件中。

有两种重定向标准输出/标准错误的形式：

```
&>word
```

还有

```
>&word
```

两种形式中，推荐使用第一种。它与

```
>word 2>&1
```

在语义上等价。

Here Documents

这种重定向使得 shell 从当前源文件读取输入，直到遇到仅包含 word 的一行 (并且没有尾部空白，trailing blanks) 为止。直到这一点的所有行被用作命令的标准输入。

here-document 的格式是：

```
<<[-]word
        here-document
delimiter
```

不会对 word 进行 parameter expansion, command substitution, arithmetic expansion, 或者 pathname expansion。如果 word 中任何字符是引用的，delimiter 将是对 word 进行引用删除的结果，here-document 中的行不会被扩展。如果 word 没有被引用，here-documnet 中的所有行都要经过 parameter expansion, command substitution, 和 arithmetic expansion。在后一种情况下，字符序列 \<newline> 被忽略；必须用 \ 来引用字符 \, $, 和 ‘.

如果重定向操作符是 <<-, 那么所有前导的 tab 字符都被从输入行和包含 delimiter 的行中删除。这样使得 shell 脚本中的 here-document 可以被更好地缩进。

Here Strings

here-document 的变种，形式是

```
<<<word
```

word 被扩展，提供给命令作为标准输入。

Duplicating File Descriptors (复制文件描述符)

重定向操作符

[n]<&word

用于复制文件描述符。如果 word 扩展为一个或多个数字， n 代表的文件描述符将 成为那个文件描述符的复制。如果 word 中的数字并未指定一个被用于读取的文件描述符，将产生一个重定向错误。如果 word 扩展为 -, 文件描述符 n 将被关闭。如果没有指定 n，将使用标准输入 (文件描述符 0)。

类似的，操作符

[n]>&word

用于复制输出文件描述符。如果没有指定 n，将使用标准输出 (文件描述符 1)。如果 word 中的数字并未指定一个被用于输出的文件描述符，将产生一个重定 向错 误。特殊情况下，如果忽略了 n，并且 word 并非扩展为一个或多个数字，标准输出和标准错误将被重定向，和前面描述的一样。

Moving File Descriptors

重定向操作符

[n]<&digit-

将文件描述符 digit 移动为文件描述符 n, 或标准输入 (文件描述符 0)，如 果没有指定 n 的话。 digit 复制为 n 之后就被关闭了。

类似的，重定向操作符

[n]>&digit-

将 文件描述符 digit 移动为文件描述符 n, 或标准输出 (文件描述符 1)，如果没有指定 n 的话。

Opening File Descriptors for Reading and Writing

重定向操作符

[n]<>word

使得以 word 扩展结果为名的文件被打开，通过文件描述符 n 进行读写。如果没有指定 n 那么就使用文件描述符 0。如果文件不存在，它将被创建。

别名(ALIASES)

Aliases (别名机制) 允许将一个词来替换为一个字符串，如果它是一个简单命令的第一个词的话。shell 记录着一个别名列表，可以使用内建命 令 alias 和 unalias 来定义和取消 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。每个命令的第一个词，如果没有引用，都将被检查是否是一个别名。如果是， 这 个词将被它所指代的文本替换。别名和替换的文本可以包含任何有效的 shell 输入，包含上面列出的 metacharacters (元字符) ，特殊情况是别名中不能 包含 =。替换文本的第一个词也被检查是否是别名，但是如果它与被替换的别名相同，就不会再替换第二次。这意味着可以用 ls 作为 ls -F 的别名， bash 不 会递归地展开替换文本。如果别名的最后一个字符是 blank, 那么命令中别名之后的下一个词也将被检查是否能进行别名展开。

别名可以使用 alias 命令来创建或列举出来，使用 unalias 命令来删除。

在替换文本中没有参数机制。如果需要参数，应当使用 shell 函数 (参见下面的 FUNCTIONS (函数) 段落)。

如 果 shell 不是交互的，别名将不会展开，除非使用内建命令 shopt 设置了 expand_aliases 选项。

关于别名的定义和使用中的规则比较混乱。 Bash 在执行一行中的任何命令之 前， 总是读入至少完整一行的输入。别名在命令被读取时展开，而不是在执行的时候。因此，别名定义如果和另一个命令在同一行，那么不会起作用，除非读入 了下 一行。别名定义之后，同一行中的命令不会受新的别名影响。这种行为在函数执行时存在争议，因为别名替换是在函数定义被读取时发生的，而不是函数被 执行 的时候，因为函数定义本身是一个复合命令。结果，在函数中定义的别名只有当这个函数执行完才会生效。为了保险起见，应当总是将别名定义放在单独的 一行，不在复合命令中使用 alias。

不管什么情况下，别名都被 shell 函数超越 (be superseded)。

函数(FUNCTIONS)

一 个 shell 函数，以上面 SHELL GRAMMAR 中描述的方法定义，保存着一系列的命令，等待稍后执行。当 shell 函数名作为一个简单命令名使用时，这个函数名关联的命令的序列被执行。函数在当前 shell 的上下文环境中执行；不会创建新的进程来解释它们 (这与 shell 脚本的执行形成了对比)。当执行函数时，函 数

的参数成为执行过程中的位置参数 (positional parameters)。特殊参数 # 被更新以反映这个变化。位置参数 0 不会改变。函数执行时， FUNCNAME 变量被设置为函数的名称。函数和它的调用者在 shell 执行环境的所有其他方面都是一样的，特殊情况是 DEBUG 陷阱 (参见下面对内建函数 trap 的描述，在 shell 内 建命 令(SHELL BUILTIN COMMANDS) 章节中) 不会被继承，除非函数设置了 trace 属性 (参见下面对内建函数 declare 的描述)。

函数中的局部变量可以使用内建命令 local 来声明。通常情况下，变量和它们的值在函数和它的调用者之间是共享的。

如果函数中执行了内建命令 return，那么函数结束，执行从函数调用之后的下一个命令开始。函数结束后，位置参数的值以及特殊参数 # 都将重置为它们在函数执行前的值。

函 数名和定义可以使用内建命令 declare 或 typeset 加上 -f 参数来列出。如果在 declare 或 typeset 命令中使用 -F 选项将只列出函数名。函数可以使 用内建命令 export 加上 -f 参数导出，使得子 shell 中它们被自动定义。

函数可以是递归的。对于递归调用的次数没有硬性限制。

算术求值("ARITHMETIC EVALUATION")

在 一 定 的 环境下，shell 允许进行算术表达式的求值 (参见内建命令 let 和 Arithmetic Expansion (算术表达式))。求值使用固定宽度的整数，不检查是 否溢出，但是被零除会被捕获，标记为错误。操作数及其优先级和聚合程度与 C 语言中相同。下列操作数的列表按照相同优先级的操作数其级别来分组。列出的 级别顺序是优先级递减的。

id++ id--
　　变量自增/自减 (在后)
++id --id
　　变量自增/自减 (在前)
- + (单目的) 取负/取正
! ~ 逻辑和位取反
** 乘幂
* / % 乘，除，取余
+ - 加，减
<< >> 左/右位移
<= >= < >
　　比较
== != 相等/不等

& 位与 (AND)
^ 位异或 (exclusive OR)
| 位或 (OR)
&& 逻辑与 (AND)
|| 逻辑或 (OR)
expr?expr:expr
条件求值
= *= /= %= += -= <<= >>= &= ^= |=
赋值
expr1 , expr2
逗号表达式

shell 变量可以作为操作数；在表达式求值之前会进行参数扩展。在表达式中，可以用名称引用 shell 变量，不必使用参数扩展的语法。变量被引用时，其值被作为算术表达式来求值。 shell 变量用于表达式中时，不必启用整数属性。

以 0 为前导的常量被当作八进制数，以 0x 或 0X 作为前导表明是十六进制。其他情况下，数字的形式是 [base#]n，这里 base 是一个 2 到 64 的十进制数 值，作 为数字的基数，n 是在这个基数中数字的值。如果忽略了 base#，将以 10 为基数。大于 10 的数字依次以小写字母，大写字母，@ 和 _ 表示。如果 base 小于或等于 36，在表示 10 与 35 之间的数字时小写字母和大写字母可以互换。

操作符根据优先级顺序进行求值。圆括号中的子表达式被最先求值，可能会超 越上面的优先级规则。

条件表达式("CONDITIONAL EXPRESSIONS")

条件表达式用于 [[复合命令以及内建命令 test 和 [中，用来测试文件属性，进行字符串和算术比较。表达式使用下面的单目或二进制操作构造。如果某操 作的 任何 file 参数的形式是 /dev/fd/n，那么将检查文件描述符 n。如果某操作的 file 参数是 /dev/stdin， /dev/stdout 或者 /dev/stderr 之一，将分别检查文件描述符 0，1 和 2。

-a file
如果 file 存在则为真。
-b file
如果 file 存在且为块设备则为真。
-c file
如果 file 存在且为字符设备则为真。
-d file
如果 file 存在且是一个目录则为真。

-e file

如果 file 存在则为真。

-f file

如果 file 存在且为普通文件则为真。

-g file

如果 file 存在且是设置组 ID 的 (sgid) 则为真。

-h file

如果 file 存在且为符号链接则为真。

-k file

如果 file 存在且设置了 ‘‘sticky’’ 位 (粘滞位) 则为真。

-p file

如果 file 存在且是一个命名管道 (FIFO) 则为真。

-r file

如果 file 存在且可读则为真。

-s file

如果 file 存在且大小大于零则为真。

-t fd 如果文件描述符 fd 是打开的且对应一个终端则为真。

-u file

如果 file 存在且是设置用户 ID 的 (suid) 则为真。

-w file

如果 file 存在且可写则为真。

-x file

如果 file 存在且可执行则为真。

-O file

如果 file 存在且为有效用户 ID 所拥有则为真。

-G file

如果 file 存在且为有效组 ID 所拥有则为真。

-L file

如果 file 存在且为符号链接则为真。

-S file

如果 file 存在且为套接字则为真。

-N file

如果 file 存在且上次读取后被修改过则为真。

file1 -nt file2

如果 file1 比 file2 要新 (根据修改日期)，或者如果 file1 存在而 file2 不存在，则为真。

file1 -ot file2

如果 file1 比 file2 更旧，或者如果 file1 不存在而 file2 存在，则为真。

file1 -ef file2

如果 file1 和 file2 指的是相同的设备和 inode 号则为真。

-o optname

如 果启用了 shell 选项 optname 则为真。参见下面对内建命令 set 的 -o 选项的描述中的选项列表。

-z string

如果 string 的长度为 0 则为真。

-n string

string 如果 string 的长度非 0 则为真。

string1 == string2

如果字符串相等则为真。= 可以用于使用 == 的场合来兼容 POSIX 规 范 。

string1 != string2

如果字符串不相等则为真。

string1 < string2

如果 string1 在当前语言环境的字典顺序中排在 string2 之前则为真。

string1 > string2

如果 string1 在当前语言环境的字典顺序中排在 string2 之后则为真。

arg1 OP arg2

OP 是 -eq, -ne, -lt, -le, -gt, 或 -ge 之一。这些算术二进制操作返回真，如果 arg1 与 arg2 分别是相等，不等，小于，小于或等于，大于，大于或等于关系。 Arg1 和 arg2 可以是正/负整数。

简单命令扩展("SIMPLE COMMAND EXPANSION")

当执行一个简单命令时，shell 进行下列扩展，赋值和重定向，从左到右。

1. 解释器标记为与变量赋值 (在命令名之前的) 和重定向有关的词被保存等待随后处理。

2. 并非变量赋值或重定向的词被扩展。如果扩展后仍然有词保留下来，第一个词被作为命令名，其余词是参数。

3. 重定向按照上面 REDIRECTION 中讲到的规则进行。

4. 每个变量赋值中 = 之后的文本在赋予变量之前要经过波浪线扩展，参数扩展，命令替换，算术扩展和引用删除。

如果没有得到命令名，变量赋值影响当前 shell 环境。否则，变量被加入被执行的命令的环境中，不影响当前 shell 环境。如果任何赋值动作试图为只读变量赋值，将导致出错，命令以非零状态值退出。

如果没有得到命令名，重定向仍会进行，但是不影响当前 shell 环境。重定向出错将使命令以非零状态值退出。

如 果扩展后有命令名保留下来，那么执行过程如下所示。否则，命令退出。如果在任何扩展中包含命令替换，那么整个命令的退出状态是最后一个命令替换的 退出状态。如果没有进行命令替换，命令以状态零退出。

命令执行(COMMAND EXECUTION)

命 令被拆分为词之后，如果结果是一个简单命令和可选的参数列表，将执行下面的操作。

如果命令名中没有斜杠，shell 试图定位命令位置。如果存在同名的 shell 函数，函 数 将 被执行，像上面 FUNCTIONS 中讲到的一样。如果名称不是一个函数，shell 从 内建命令中搜索它。如果找到对应命令，它将被执行。

如果名称既不是 shell 函数也不是一个内建命令，并且没有包含斜杠， bash 搜索 PATH 的每个成员，查找含有此文件名 (可执行文件) 的目录。 Bash 使用散列表来储存可执行文件的全路径 (参见下面的 shell 内建命 令(SHELL BUILTIN COMMANDS) 中的 hash。只有在散列表中没有找到此命令，才对 PATH 进行完整的搜索。如果搜索不成功，shell 输出错误消息，返回退出状态 127。

如果搜索成功，或者命令中包含一个或多个斜杠，shell 在单独的执行环境中 执行这个程序。参数 0 被设置为所给名称；命令的其他参数被设置为所给的参数，如果有的话。

如果执行失败，因为文件不是可执行格式，并且此文件不是目录，就假定它是 一个 shell script (脚本)，一个包含 shell 命令的文件。此时将孵化 (spawn) 出一个子 shell 来执行它。子 shell 重新初始化自身，效果就好像是执行了 一个 新的 shell 来处理脚本一样，但是父 shell 保存的命令位置仍然被保留 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 中的 hash)。

如果程序是以 #!　开头的文件，那么第一行的其余部分指定了这个程序的解释器。 shell 执行指定的解释器，如果操作系统不会自行处理这种可执行文件格式的话。解释器的参数由下面三部分组成：程序第一行中解释器名称之后的可选的 一个参数，程序的名称，命令行参数，如果有的话。

命令执行环境(COMMAND EXECUTION ENVIRONMENT)

shell 有 execution environment (执行环境) 的概念，由下列内容组成：

- shell 启动时继承的打开的文件，例如在内建命令 exec 中使用重定向修改的结果
- 当前工作目录，使用 cd，pushd 或者 popd 设置，或是由 shell 在启动时继承得到
- 文件创建模式掩码，使用 umask 设置或是从 shell 的父进程中继承得到
- 当前陷阱，用 trap 设置
- shell 参数，使用变量赋值或者 set 设置，或者是从父进程的环境中 继承得到
- shell 函数，在执行中定义或者是从父进程的环境中继承得到
- 设为允许的选项，在执行时设置 (要么是默认允许的，要么是命令行给出的) 或者是用 set 设置
- 用 shopt 设为允许的选项
- 用 alias 定义的 shell 别名
- 各种进程号，包含后台作业的进程号，$$ 的值，以及 $PPID 的值

当并非 shell 函数或内置命令的简单命令执行时，它在一个由下述内容组成的单独的执行环境中启动。除非另外说明，值都是从 shell 中继承的。

- shell 打开的文件，加上对命令使用重定向修改和添加的文件
- 当前工作目录
- 文件创建模式掩码
- 标记为导出 (export) 的 shell 变量，以及传递到环境中为这个命令导出的变量
- shell 捕捉的陷阱被重置为从 shell 的父进程中继承的值，shell 忽 略的陷阱也被忽略

在单独的环境中启动的命令不能影响 shell 的执行环境。

命令替换和异步命令都在子 shell 环境中执行。子 shell 环境是原有 shell 环境的赋值，但 shell 捕捉的陷阱被重置为 shell 启动时从父进程中继承的值 。作 为 管道一部分来执行的内建命令也在一个子 shell 环境中执行。对子 shell 环境所作修改不能影响到原有 shell 的执行环境。

如果命令后面是 & 并且没有启用作业控制，命令的默认标准输入将 是 空 文 件 /dev/null 。否则，被执行的命令从调用它的 shell 中继承被重定向修改的文件描述符。

环境(ENVIRONMENT)

当一个程序执行时，它被赋予一个字符串数组，成为环境 environment。它是 一个名称-值对 (name-value) 的列表，形式是 name=value.

shell 提供了多种操作环境的方法。启动时，shell 扫描自身的环境，为每个找到的名字创建一个参数，自动地将它标记为 export (向子进程导出的)。被执 行的 命令继承了这个环境。 export 和 declare -x 命令允许参数和函数被加入到环境中或从环境中删除。如果环境中参数的值被修改，新值成为环境的一部分，替换了旧值。所有被执行的命令继承的环境包含 shell 的初始环境 (可能值被修改过)，减去被 unset 命令删除的，加上通过 export 和 declare -x 命令添 加的部分。

可 以在任何 simple command 或函数的环境中设定暂时有效的参数，只要将参数赋值放在命令前面就可以了，参见上面 PARAMETERS 的描述。这些赋值语句只 在这个命令的环境中有效。

如 果设置了内建命令 set 的 -k 选项， 所有的变量赋值都将放到命令的环境中，不仅是在命令名前面的那些。

当 bash 执行一个外部命令时，变量 _ 被设置为命令的文件全名，然后被传递到命令的环境之中。

退出状态("EXIT STATUS")

从 shell 的角度看，一个命令退出状态是 0 意味着成功退出。退出状态是 0 表明成功。非零状态值表明失败。当命令收到 fatal signal N 退出时， bash 使用 128+N 作为它的退出状态。

如 果没有找到命令，为执行它而创建的子进程返回 127。如果找到了命令但是文件不可执行，返回状态是 126。

如果命令由于扩展或重定向错误而失败，退出状态大于零。

shell 内建命令如果成功返回 0(true)，执行时出错则返回非零 (false)。所 有内建命令返回 2 来指示不正确的用法。

Bash 自身返回最后执行的命令的退出状态，除非发生了语法错误，这时它返回非零值。参见下面的内建命令 exit。

信号(SIGNALS)

如果 bash 是交互的，没有设定任何陷阱，它忽略 SIGTERM (这样 kill 0 不 会杀 掉交互的 shell)。 SIGINT 被捕获并处理 (从而使内建命令 wait 可以中断) 。在所有情况下， bash 忽略 SIGQUIT。如果正在使用作业控制， bash 忽 略 SIGTTIN, SIGTTOU, 和 SIGTSTP.

bash 开始的并行作业的信号处理句柄都设置为 shell 从父进程中继承的值。如果不是正在使用作业控制，异步命令还忽略 SIGINT 和 SIGQUIT 。作为命令替换结 果运行的命令忽略键盘产生的作业控制信号 SIGTTIN, SIGTTOU, 和 SIGTSTP.

如果收到信号 SIGHUP， shell 默认退出。在退出前，交互的 shell 向所有作业，运行的或停止的，发送 SIGHUP 信号。shell 向停止的作业发出 SIGCONT 信号来保证它们会收到 SIGHUP. 要阻止 shell 向特定的作业发送信号，应当使用内建 命 令 disown 将作业从作业表中删除 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节) 或者使用 disown -h 来标记为不接受 SIGHUP。

如果使用 shopt 设置了 shell 选项 huponexit，在交互的登录 shell 退 出 时 bash 向所有作业发出 SIGHUP 信号。

当 bash 等待命令执行结束时，如果收到已设置了陷阱的信号，陷阱 (trap) 将不会执行，直到命令结束。当 bash 通过内建命令 wait 等待异步命令时，如 果收到已设置了陷阱的信号，将使得内建命令 wait 立即以大于 128 的状态值返回。接着，陷阱将立即被执行。

作业控制("JOB CONTROL")

Job control (作业控制) 指的是可以选择停止 (suspend，挂起) 进程执行， 并且 可以在之后继续 (resume，恢复) 执行的能力。用户一般在交互的人机界面中使用这种功能。界面是由系统的终端驱动和 bash 共同提供的。

shell 将每个管道分配给一个 作业(job)。它保存一个当前运行的作业表，可 以用 jobs 命令来列出。当 bash 启动一个异步的作业时 (background，后台执行)

，它输出这样的一行：

[1] 25647

表明这个作业的作业号是 1，与作业相关连的管道中最后一个进程的 进 程 ID 是 15647 。管道中所有进程都是同一个作业的成员。 Bash 使用 作业(job) 概念作为作业控制的基础。

为简化作业控制的用户界面的实现，操作系统负责管理“当前终 端 的 进 程 组” (current terminal process group ID) 的概念。这个进程组的成员 (进程组 ID 是当前终端进程组 ID 的进程) 可以收到键盘产生的信号，例如 SIGINT. 这 些进程被称为 foreground(前台的)。 Background (后台的) 进程是那些进程组 ID 与终端不同的进程；这些进程不会收到键盘产生的信号。只有前台进程可以从 终端读或向终端写。后台进程试图读/写终端时，将收到终端驱动程序发送的 SIGT-TIN (SIGTTOU) 信号。这个信号如果没有加以捕捉，将挂起这个进程。

如果 bash 运行其上的操作系统支持作业控制， bash 会包含使用它的设施。 在一个进程正在运行的时候键入 suspend 挂起字符 (通常是 ^Z, Control-Z) 将使这个进程暂停，将控制权还给 bash. 输入 delayed suspend , 延时挂起字符 (通 常是 ^Y, Control-Y) 将使这个进程在试图从终端读取输入时暂停，将控制权还给 bash. 用户接下来可以控制此作业的状态，使用 bg 命令使它在后台继 续运行， fg 命令使它在前台继续运行，或 kill 命令将它杀死。^Z 会立即起作用，并且还有使等待中的 (pending) 输出和输入被忽略的附加副作用。

有很多方法来指代 shell 中的作业。字符 % 可以引入作业名。编号为 n 的作业可 以用 %n 的形式来指代。作业也可以用启动它的名称的前缀，或者命令行中的子字符串来指代。例如， %ce 指代一个暂停的 ce 作业。如果前缀匹配多于一个作 业 ， bash 报错。另一方面，使用 %?ce, 可以指代任何命令行中包含字符串 ce 的作业。如果子字符串匹配多于一个作业， bash 报错。符号 %% 和 %+ 指代 shell 意义上的 current job，当前作业，也就是前台被暂停的最后一个作业，或者是在后台启动的作业。 previous job ，前一作业可以使用 %- 来指代。 在有 关作业的输出信息中 (例如，命令 jobs 的输出)，当前作业总是被标记为 +, 前一作业标记为 -.

简单地给出作业名，可以用来把它放到前台： %1 是 ‘‘fg %1’’ 的同义词，将作业 1 从后台放到前台。类似的， ‘‘%1 &’’ 在后台恢复作业 1，与 ‘‘bg %1’’ 等价。

当某个作业改变状态时，shell 立即可以得知。通常， bash 等待直到要输出 一个 提示符时，才会报告作业的状态变化，从而不会打断其他输出。如果启用了内

建命令 set 的 -b 选项， bash 将立即报告这些变化。对 SIGCHLD 信号的陷 阱将在每个子进程退出时执行。

如 果在作业暂停时试图退出 bash， shell 打印一条警告消息。命令 jobs 可能被用来检查作业的状态。如果再次试图退出，中间没有其他命令，shell 不会 打印其他警告，暂停的作业将终止。

提示符(PROMPTING)

在 交互执行时， bash 在准备好读入一条命令时显示主提示符 PS1，在需要更多的输入来完成一条命令时显示 PS2。 Bash 允许通过插入一些反斜杠转义的特 殊字符来定制这些提示字符串，这些字符被如下解释：

\a 一个 ASCII 响铃字符 (07)

\d 日期，格式是 "星期 月份 日" (例如，"Tue May 26")

\D{format}

format 被传递给 strftime(3)，结果被插入到提示字符串中；空的 format 将使用语言环境特定的时间格式。花括号是必需的

\e 一个 ASCII 转义字符 (033)

\h 主机名，第一个 ‘.’ 之前的部分

\H 主机名

\j shell 当前管理的作业数量

\l shell 的终端设备名的基本部分

\n 新行符

\r 回车

\s shell 的名称， $0 的基本部分 (最后一个斜杠后面的部分)

\t 当前时间，采用 24 小时制的 HH:MM:SS 格式

\T 当前时间，采用 12 小时制的 HH:MM:SS 格式

\@ 当前时间，采用 12 小时制上午/下午 (am/pm) 格式

\A 当前时间，采用 24 小时制上午/下午格式

\u 当前用户的用户名 the username of the current user

\v bash 的版本 (例如，2.00)

\V bash 的发行编号，版本号加补丁级别 (例如，2.00.0)

\w 当前工作目录

\W 当前工作目录的基本部分

\! 此命令的历史编号

\# 此命令的命令编号

\$ 如果有效 UID 是 0，就是 #, 其他情况下是 $

\nnn 对应八进制数 nnn 的字符

\\ 一个反斜杠

\[一个不可打印字符序列的开始，可以用于在提示符中嵌入终端 控制序列

\]　一个不可打印字符序列的结束

命令编号和历史编号通常是不同的：历史编号是命令在历史列表中的位置，可能包含从历史文件中恢复的命令 (参见下面的 HISTORY 历史章节)，而命令编号是当前 shell 会话中执行的命令序列中，命令的位置。字符串被解码之后，它将进行扩展，要经过 parameter expansion, command substitution, arithmetic expansion 和 quote removal, 最后要经过 shell 选项 promptvars 处理 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节中，对命令 shopt 的描述)。

readline 库(READLINE)

这是在交互 shell 中处理读取输入的库，除非在 shell 启动时给出了 --noediting 选项。默认情况下，行编辑命令类似于 emacs 中的那些。也可以使用 vi 样式的行编辑界面。要在 shell 运行之后关闭行编辑，使用内置命令 set 的 +o emacs 或 +o vi 选项 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

Readline Notation

在这个小节中，将使用 emacs 样式的记法来表述按键。Ctrl 键记为 C-key，例如，C-n 意思是 Ctrl-N。类似的， meta 键记为 M-key，因此 M-x 意味着 Meta-x 。(在没有 meta 键的键盘上，M-x 意思是 ESC-x，也就是说，按下 Esc 键，然后按 x 键。这使得 Esc 成为 meta prefix。M-C-x 的组合意思是 Esc-Ctrl-x，也就是按 Esc 键，然后按住 Ctrl 键，同时按 x 键。)

readline 命令可以有数字的参数(arguments)，一般作为重复的计数。有些时候，它是重要参数的标记。给向前方进行的命令 (例如，kill-line) 传递负数参数，将使得命令向反方向进行。下面的命令如果接受参数时的行为与此不同，将另行说明。

当命令被描述为剪切 (killing) 文本时，被删除的文本被保存，等待将来使用 (粘贴，yanking)。被剪切的文本保存在 kill ring 中。连续的剪切使得文本被依次加入到一个单元中，可以一次被粘贴。不剪切文本的命令将 kill ring 中的文本分离。

Readline Initialization 初始化

readline 可以通过将命令放入初始化文件 (inputrc 文件) 来定制。文件名从变量 INPUTRC 的值中获取。如果没有设置这个变量，默认是 ~/.inputrc. 当使用 readline 库的程序启动时，将读取初始化文件，按键关联和变量将被设置。readline 初始化文件中只允许有很少的基本构造。空行被忽略。以 # 开始的行是注释。以 $ 开始的行指示了有条件的构造。其他行表示按键关联和变量设置。

默认的按键关联可以使用 inputrc 文件改变。其他使用这个库的程序可以添加它们自己的命令和关联。

例如，将

M-Control-u: universal-argument

或

C-Meta-u: universal-argument

放入 inputrc 将使得 M-C-u 执行 readline 命令 universal-argument.

可以识别下列字符的符号名称： RUBOUT, DEL, ESC, LFD, NEWLINE, RET, RETURN, SPC, SPACE, 和 TAB.

在命令名之外，readline 允许将按键与一个字符串关联，当按下这个键时，将插入这个字符串 (一个宏，macro)。

Readline Key Bindings

inputrc 文件中的控制按键关联的语法非常简单。需要的内容是命令名或宏，以及它应当关联到的按键序列。名称可以以两种方式指定：一个按键的符号名称，可能带有 Meta- 或 Control- 前缀，或者是一个按键序列。

当使用 keyname:function-name 或 macro 形式时， keyname 是按键以英文拼写的名称。例如：

Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> output"

在上述例子中， C-u 被关联到函数 universal-argument, M-DEL 被关联到函 数 backward-kill-word, 而 C-o 被关联为运行右边给出的宏 (意思是，将向行中插入 ‘‘> output’’)。

在第二种形式中，"keyseq":function-name 或 macro， keyseq 不同于上 面 的 keyname，表示整个按键序列的字符串可以通过将按键序列放在双引号引用中来指定。可以使用一些 GNU Emacs 样式的按键序列，如下例所示，但是不会识别按键的符号名称。

"\C-u": universal-argument
"\C-x\C-r": re-read-init-file

"\e[11~": "Function Key 1"

在上述例子中， C-u 被又一次关联到函数 universal-argument. C-x C-r 被关联到函数 re-read-init-file, 而 ESC [1 1 ~ 被关联为插入文本 ‘‘Function Key 1’’.

GNU Emacs 样式的转义序列的全集为：

\C- Ctrl 前缀
\M- Meta 前缀
\e 一个 Esc 字符
\\ 反斜杠
\" 字面上的 "
\’ 字面上的 ’

除了 GNU Emacs 样式的转义序列，还有一系列反斜杠转义序列可用：

\a 响铃
\b 回退
\d 删除
\f 进纸
\n 新行符
\r 回车
\t 水平跳格
\v 竖直跳格
\nnn 一个八比特字符，它的值是八进制值 nnn (一到三个八进制数字)。
\xHH 一个八比特字符，它的值是十六进制值 HH (一到两个十六进制数字)。

输 入宏的文本时，必须使用单引号或双引号引用来表明是宏的定义。没有引用的文本被当作函数名。在宏的定义体中，上述反斜杠转义被扩展。反斜杠将引用 宏文本中所有其他字符，包括 " 和 ’ 。

Bash 允许使用内建命令 bind 来显示和修改当前 readline 按键关联。在交互使用中可以用内建命令 set 的 -o 选项切换到编辑模式 (参见下面的 shell 内 建命令(SHELL BUILTIN COMMANDS) 章节)。

Readline Variables

readline 包含额外的可用于定制它的行为的变量。可以在 inputrc 文件中设置变量，使用如下形式的语句：

set variable-name value

除非另外说明，readline 变量的值总是 On 或 Off。变量和它们的默认值是：

bell-style (audible)

控制了当 readline 需要鸣终端响铃时的动作。如果 设 置 为 none ，readline 不会鸣铃。如果设置为 visible，readline 使用可视的响铃，如果可用的话。如果设置为 audible，readline 试着鸣终端响铃。

comment-begin (‘‘#’’)

这个字符串在执行 readline 命令 insert-comment 时被插入。这个命令在 emacs 模式下被关联为 M-#，在 vi 模式下是 #。

completion-ignore-case (Off)

如果设置为 On，readline 进行大小写不敏感的文件名匹配和补全。

completion-query-items (100)

这 个 变 量决定着何时向用户询问，是否查看由命令 possible-completions 产生的可能的补全数量。它可以设为任何大于或等于 0 的值。 如果可能的补全数量大于或等于这个变量的值，用户将被提示是否愿意查看它们；否则将直接在终端上列出它们。

convert-meta (On)

如果设置为 On，readline 将把设置了最高位的字符转换为 ASCII 按 键序 列，方法是去掉第八位，前缀一个转义字符 (实际上，使用 Esc 作为转义符 meta prefix)。

disable-completion (Off)

如果设置为 On，readline 将禁止词的补全。补全字符将被插入到行中，就好像它们被映射为 self-insert。

editing-mode (emacs)

控制 readline 的按键关联集合与 emacs 还是 vi 相似。 editing-mode 可以设置为 emacs 或 vi。

enable-keypad (Off)

如果设置为 On ，readline 在调用时将试图启用辅助键盘。一些系统 需要设置这个来启用方向键。

expand-tilde (Off)

如果设置为 On，readline 试图进行词的补全时会进行波浪线扩展。

history-preserve-point

如 果 设置为 On，历史代码试着在 previous-history 或 next-history 取回的每个历史行的相同位置中加点。

horizontal-scroll-mode (Off)

如果设置为 On，将使得 readline 使用单行来显示，如果它比屏幕宽 度要长，就在单一的屏幕行上水平滚动输入行，而不是自动回绕到新行。

input-meta (Off)

如果设置为 On，readline 将允许八比特输入 (也就是说，它不会将它读入的字符中最高位删除)，不管它能支持什么样的 终 端 要 求 。 名 称 meta-flag 与此变量同义。

isearch-terminators (‘‘C-[C-J’’)

用于终止增量的搜索，不再将字符当作命令执行的字符串。如果这个变量没有赋值，字符串 Esc 和 C-J 将终止增量的搜索。

keymap (emacs)

设置当前 readline 键盘映射。有效 的 键 盘 映 射 名 称 是 emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-command，还有 vi-insert。 vi 等价于 vi-command; emacs 等价于 emacs-standard 。默认值是 emacs; editing-mode 的值也会影响默认的键盘映射。

mark-directories (On)

如果设置为 On，补全的目录名会添加一个斜杠。

mark-modified-lines (Off)

如果设置为 On，已被修改的历史行将显示为前缀一个星号 (*)。

mark-symlinked-directories (Off)

如 果设置为 On，补全的名称如果是到目录的符号链接，则将添加一个斜杠 (与 mark-directories 的值同样处理)。

match-hidden-files (On)

这个变量，如果设置为 On，将使得 readline 在进行文件名补全时， 匹配以 ‘.’ 开头的文件 (隐藏文件)，除非用户在要补全的文件名中给出了前导的 ‘.’。

output-meta (Off)

如果设置为 On，readline 将直接显示设置了第八位的字符，而不是转化为一个带 meta 前缀的转义序列。

page-completions (On)

如果设置为 On，readline 将使用内建的类似 more 的分页程序，来每次显示一屏可能的补全。

print-completions-horizontally (Off)

如果设置为 On，readline 将匹配的补全按字母表顺序排序，然后水平排列显示出来，而不是在屏幕上竖直排列显示。

show-all-if-ambiguous (Off)

这 将调整补全函数的默认行为。如果设置为 on, 拥有多于一个可能的补全的词将立即列出所有匹配，而不是鸣响铃。

visible-stats (Off)

如果设置为 On，在列出可能的补全时，将在文件名后面添加一个表示 文件类型的字符，文件类型由 stat(2) 报告。

Readline Conditional Constructs

readline 实现了一种功能，本质上与 C 预处理器进行条件编译的功能类似，允

许根据测试的结果进行键盘关联和变量设置。其中使用了四种解释器指令。

$if $if 结构允许根据编辑模式，正在使用的终端，使用 readline 的应用程序来设定按键关联。测试的文本包括一行，直到行尾；不必用字符来隔离它。

mode $if 结构的 mode= 形式用于测试 readline 处于 emacs 还是 vi 模式。这可以与命令 set keymap 结合使用，例如，设置 emacs-standard 和 emacs-ctlx 键盘映射，仅当 readline 以 emacs 模式启动。

term term= 形式用于包含与终端相关的按键关联，也许是将按键序列输出与终端的功能键相关联。等号 = 右边的词被同终端的全名和名称中第一个 - 前面的一部分相比较。例如，允许 sun 同时匹配 sun 和 sun-cmd。

application

application 结构用于包含应用程序相关的设置。每个使用 readline 的程序都设置 application name，初始化文件可以测试它的值。它可用于将一个按键序列与对特定的程序有用的功能相关联。例如，下列命令添加了一个按键序列，用以引用 bash 中当前的词或前一个词

```
$if Bash
# Quote the current or previous word
"\C-xq": "\eb\"\ef\""
$endif
```

$endif 上例中的这个命令，结束了一个 $if 命令。

$else 如果测试失败，$if 指令中这个分支的命令将被执行。

$include

这个指令使用单个文件名作为参数，从文件中读取命令和按键关联。例如，下列指令将读取 /etc/inputrc：

```
$include /etc/inputrc
```

Searching

readline 提供了从命令历史中搜索包含给定字符串的行的命令 (参见下面的

HISTORY 历史章节)。有两种搜索模式： incremental 和 non-incremental.

增量的搜索在用户结束输入搜索字符串时开始。在搜索字符串的每个字符被输 入的 同时， readline 显示与已输入的字符串匹配的下一个历史条目。增量的搜索只要求输入能找到期望的历史条目所需的那么多字符。 isearch-terminators 变量中的字符用来终止一次增量的搜索。如果这个变量没有被赋值，Esc 和 Ctrl-J 字符串将结束一次增量的搜索。 Ctrl-G 将取消一次增量的搜索，恢复初始的 行 。当搜索终止时，包含搜索字符串的历史条目成为当前行。

要从历史列表中找到其他匹配的条目，适当地键入 Ctrl-S 或 Ctrl-R。这样将在历史中向前/向后搜索下一个匹配已输入的搜索字符串的条目。其他关联到 某 个 readline 命令的按键序列将终止搜索并执行关联的命令。例如， newline 将终止搜索，接受当前行，从而执行历史列表中的命令。

readline 可以记住上次增量搜索的字符串。如果键入两次 Ctrl-R，中间没有 输入任何字符来定义一个新的搜索字符串，那么将使用已记住的搜索字符串。

非 增量的搜索将整个搜索字符串读入，然后才开始搜索匹配的历史条目。搜索字符串可以由用户输入，或者是当前行的内容的一部分。

Readline Command Names

下面列出的是命令的名称以及默认情况下它们关联的按键序列。命令名称如果 没有对应的按键序列，那么默认是没有关联的。在下列描述中，点 (point) 指当前光标位置，标记 (mark) 指命令 set-mark 保存的光标位置。point 和 mark 之间的文本被称为范围 (region)。

Commands for Moving 移动

beginning-of-line (C-a)

移动到当前行的开始。

end-of-line (C-e)

移动到当前行的结尾。

forward-char (C-f)

向前移动一字。

backward-char (C-b)

向后移动一字。

forward-word (M-f)

向前移动到下一词尾。词由字符 (字母和数字) 组成。

backward-word (M-b)

向后移动到当前或上一词首。

clear-screen (C-l)

清除屏幕，保留当前行在屏幕顶端。有参数时，刷新当前行，不清屏。

redraw-current-line

刷新当前行。

Commands for Manipulating the History 操纵历史行

accept-line (Newline, Return)

接 受这一行，不管光标在什么位置。如果行非空，将根据变量 HISTCONTROL 的状态加入到历史列表中。如果行是修改过的历史行，将恢复该 历史行到初始状态。

previous-history (C-p)

从历史列表中取得前一个命令，从列表中向后移动。

next-history (C-n)

从历史列表中取得后一个命令，从列表中向前移动。

beginning-of-history (M-<)

移动到历史中的第一行。

end-of-history (M->)

移动到输入历史行的末尾，也就是当前输入的行的末尾。

reverse-search-history (C-r)

从当前行开始向后搜索，按照需要在历史中向“上”移动。这是一个增量的搜索。

forward-search-history (C-s)

从当前行开始向前搜索，按照需要在历史中向“下”移动。这是一个增量的搜索。

non-incremental-reverse-search-history (M-p)

从当前行开始向后，使用非增量搜索来查找用户给出的字符串。

non-incremental-forward-search-history (M-n)

从当前行开始向前，使用非增量搜索来查找用户给出的字符串。

history-search-forward

从 当前行开始向前搜索历史，查找从当前行首到 point 之间的字符串。这是一个非增量的搜索。

history-search-backward

从当前行开始向后搜索历史，查找从当前行首到 point 之间的字符串 。这是一个非增量的搜索。

yank-nth-arg (M-C-y)

将 前 一个命令的第一个参数 (通常是上一行的第二个词) 插入到 point 位置。有参数 n 时，将前一个命令的第 n 个词 (前一个命令中的词从 0 开始计数) 插入到 point 位置。负数参数则插入前一个命令倒数第 n 个词。

yank-last-arg (M-., M-_)

插入前一个命令的最后一个参数 (上一历史条目的最后一个词)。有参 数

时，行为类似于 yank-nth-arg。后继的 yank-last-arg 调用将从历史列表中向后移动，依次将每行的最后一个参数插入。

shell-expand-line (M-C-e)

扩展行，像 shell 做的那样。其中包含别名和历史扩展，还有 所 有 的 shell 词的扩展。参见下面的 HISTORY EXPANSION 中关于历史扩展的描述。

history-expand-line (M-^)

在当前行进行历史扩展。参见下面的 HISTORY EXPANSION 中关于历史 扩展的描述。

magic-space

在 当前行进行历史扩展，并插入一个空格。参见下面的 HISTORY EXPANSION 中关于历史扩展的描述。

alias-expand-line

在当前行进行别名扩展，参见上面的 ALIASES 中关于别名扩展的描述。

history-and-alias-expand-line

在当前行进行历史和别名扩展。

insert-last-argument (M-., M-_)

与 yank-last-arg 同义。

operate-and-get-next (C-o)

接受当前行，加以执行，从历史中取出相对当前行的下一行进行编辑。任何参数都被忽略。

edit-and-execute-command (C-xC-e)

启动一个编辑器，编辑当前命令行，将结果作为 shell 命令运行。 Bash 将依次试着运行 $FCEDIT, $EDITOR, 和 emacs 作为编辑器。

Commands for Changing Text 改变文本

delete-char (C-d)

删除 point 处的字符。如果 point 在行首，行中没有字符，最后一次输入的字符没有被关联到 delete-char，将返回 EOF.

backward-delete-char (Rubout)

删 除 光 标 之后的字符。当给出一个数值的参数时，保存删除的文本到 kill ring 中。

forward-backward-delete-char

删除光标下的字符，除非光标在行尾，此时删除光标后的字符。

quoted-insert (C-q, C-v)

将输入的下一字符保持原样添加到行中。例如，可以用它来插入类似 C-q 的字符。

tab-insert (C-v TAB)

插入一个跳格符号。

self-insert (a, b, A, 1, !, ...)

插入键入的字符。

transpose-chars (C-t)

将 point 之前的字符向前移动，越过 point 处的字符，同时也改变 point 的位置。如果 point 在行尾，将调换 point 之前的两个字符。负数参数没有作用。

transpose-words (M-t)

将 point 之前的词向前移动，越过 point 处的词，同时也改变 point 的位置。如果 point 在行尾，将调换行中的最后两个词。

upcase-word (M-u)

将当前 (或下一个) 词变成全大写。有负值的参数时，将前一个词变为大写，但是不移动 point。

downcase-word (M-l)

将当前 (或下一个) 词变成全小写。有负值的参数时，将前一个词变为小写，但是不移动 point。

capitalize-word (M-c)

将当前 (或下一个) 词变为首字大写。有负值的参数时，将前一个词变为首字大写，但是不移动 point。

overwrite-mode

控 制插入/改写模式。给出一个正整数参数时，切换为改写模式。给出一个非正数参数时，切换为插入模式。这个命令只影响 emacs 模式；vi 模式的改写与此不同。每个对 readline() 的调用都以插入模式开始。在改写模式下，关联到 self-insert 的字符替换 point 处的字符，而不是将它 推 到右边。关联到 backward-delete-char 的字符以空格替换 point 前的字符。默认情况下，这个命令没有关联。

Killing and Yanking 剪切和粘贴

kill-line (C-k)

剪切从 point 到行尾的文本。

backward-kill-line (C-x Rubout)

反向剪切到行首。

unix-line-discard (C-u)

反向剪切到行首。与 backward-kill-line 没有什么区别。剪切的文本被保存于 kill-ring 中。

kill-whole-line

剪切当前行中所有字符，不管 point 在什么位置。

kill-word (M-d)

剪切从 point 到当前词尾，或者如果 point 在词之间，那么剪切到下一词尾。

backward-kill-word (M-Rubout)

剪切 point 之后的词。词的边界与 backward-word 使用的相同。

unix-word-rubout (C-w)

剪切 point 之后的词，使用空白作为词的边界。剪切的文本被 保 存 于 kill-ring 中。

delete-horizontal-space (M-\)

删除 point 两边的所有空格和跳格。

kill-region

剪切当前 region 的文本。

copy-region-as-kill

将 region 的文本复制到剪切缓冲区中。

copy-backward-word

将 point 前面的词复制到剪切缓冲区中。词的边界与 backward-word 使用的相同。

copy-forward-word

将 point 之后的词复制到剪切缓冲区中。词的边界与 backward-word 使用的相同。

yank (C-y)

将 kill-ring 顶部的内容粘贴到 point 处的缓冲区中

yank-pop (M-y)

轮转 kill-ring，粘贴新的顶部内容。只能在 yank 或 yank-pop 之后使用。

Numeric Arguments 数值参数

digit-argument (M-0, M-1, ..., M--)

将这个数字加入已有的 (already accumulating) 参数中，或者开始新的参数。 M-- 开始一个否定的参数。

universal-argument

这是指定参数的另一种方法。如果这个命令后面跟着一个或多个数字，可能还包含前导的负号，这些数字定义了参数。如果命令之后跟随着数字，再次执行 universal-argument 将结束数字参数，但是其他情况下被忽略。有一种特殊情况，如果命令之后紧接着一个并非数字或负号的字符，下一命令的参数计数将乘以 4。参数计数初始是 1，因此第一次执行这个函数，使得参数计数为 4，第二次执行使得参数计数为 16，以此类推。

Completing 补全

complete (TAB)

试着对 point 之前的文本进行补全。 Bash 依次试着将文本作为一个 变量 (如果文本以 $ 开始)，一个用户名 (如果文本以 ~ 开始)，主机名 (如果文本以 @ 开始)，或者命令 (以及别名和函数) 来补全。如果这些都没有匹配，将尝试文件名补全。

possible-completions (M-?)

列出 point 之前的文本可能的补全。

insert-completions (M-*)

插入 possible-completions 已产生的 point 之前的文本所有的补全。

menu-complete

与 complete 相似，但是使用可能的补全列表中的某个匹配替换要补全的词。重复执行 menu-complete 将遍历可能的补全列表，插入每个匹配 。到达补全列表的结尾时，鸣终端响铃 (按照 bell-style 的设置来做) 并恢复初始的文本。参数 n 将在匹配列表中向前移动 n 步；负数参数可以用 于在列表中向后移动。这个命令应当与 TAB 键关联，但是默认情况下是没有关联的。

delete-char-or-list

删除光标下的字符，如果不是在行首或行尾 (类似 delete-char)。如 果在行尾，行为与 possible-completions 一致。这个命令默认没有关联。

complete-filename (M-/)

尝试对 point 之前的文本进行文件名补全。

possible-filename-completions (C-x /)

列出 point 之前的文本可能的补全，将它视为文件名。

complete-username (M-~)

尝试对 point 之前的文本进行补全，将它视为用户名。

possible-username-completions (C-x ~)

列出 point 之前的文本可能的补全，将它视为用户名。

complete-variable (M-$)

尝试对 point 之前的文本进行补全，将它视为 shell 变量。

possible-variable-completions (C-x $)

列出 point 之前的文本可能的补全，将它视为 shell 变量。

complete-hostname (M-@)

尝试对 point 之前的文本进行补全，将它视为主机名。

possible-hostname-completions (C-x @)

列出 point 之前的文本可能的补全，将它视为主机名。

complete-command (M-!)

尝试对 point 之前的文本进行补全，将它视为命令名。命令补全尝试 着将此文本依次与别名，保留字，shell 函数，shell 内建命令，最后是可执行文件名进行匹配。

possible-command-completions (C-x !)

列出 point 之前的文本可能的补全，将它视为命令名。

dynamic-complete-history (M-TAB)

尝试对 point 之前的文本进行补全，将此文本与历史列表中的行相比 较来查找可能的补全匹配。

complete-into-braces (M-{)

进行文件名补全，将可能的补全列表放在花括号中插入，使得列表可以被

shell 使用 (参见上面的 Brace Expansion 花括号扩展)。

Keyboard Macros 宏

start-kbd-macro (C-x ()

开始保存输入字符为当前键盘宏。

end-kbd-macro (C-x))

停止保存输入字符为当前键盘宏，保存宏定义。

call-last-kbd-macro (C-x e)

重新执行上次定义的键盘宏，即显示出宏中的字符，好像它们是从键盘输入的一样。

Miscellaneous

re-read-init-file (C-x C-r)

读入 inputrc 文件的内容，合并其中的按键关联和变量赋值。

abort (C-g)

取消当前编辑命令，鸣终端响铃 (按照 bell-style 的设置来做)。

do-uppercase-version (M-a, M-b, M-x, ...)

如 果有 Meta 前缀的字符 x 是小写的，那么与命令相关连的是对应的大写字符。

prefix-meta (ESC)

将输入的下一个字符加上 Meta 前缀。 ESC f 等价于 Meta-f.

undo (C-_, C-x C-u)

增量的撤销，分别记住每一行。

revert-line (M-r)

撤销这一行的所有修改。这与执行命令 undo 足够多次的效果相同，将这一行恢复到初始状态。

tilde-expand (M-&)

对当前词进行波浪线扩展。

set-mark (C-@, M-<space>)

在 point 处设置 mark。如果给出了数值的参数，标记被设置到那个位置。

exchange-point-and-mark (C-x C-x)

交换 point 和 mark。当前光标位置被设置为保存的位置，旧光标位置被保存为 mark。

character-search (C-])

读入一个字符，point 移动到这个字符下一次出现的地方。负数将搜索上一个出现。

character-search-backward (M-C-])

读入一个字符，point 移动到这个字符上一次出现的地方。负数将搜索下面的出现。

insert-comment (M-#)

没有数值的参数时，readline 变量 comment-begin 的值将被插入到当前行首。如果给出一个数值的参数，命令的行为类似于一个开关：如果行首字符不匹配 comment-begin 的值，将插入这个值，否则匹配 comment-begin 的字符将被从行首删除。在两种情况下，这一行都被接受，好像输入了新行符一样。comment-begin 的默认值使得这个命令将当前行变成一条 shell 注释。如果数值参数使得注释字符被删除，这一行将被 shell 执行。

glob-complete-word (M-g)

point 之前的词被当作路径扩展的一个模式，尾部暗含了一个星号。这个模式被用来为可能的补全产生匹配的文件名列表。

glob-expand-word (C-x *)

point 之前的词被当作路径扩展的一个模式，匹配的文件名的列表被插入，替换这个词。如果给出一个数值参数，在路径扩展之前将添加一个星号。

glob-list-expansions (C-x g)

显示 glob-expand-word 可能产生的扩展的列表，重绘当前行。如果给出一个数值参数，在路径扩展之前将添加一个星号。

dump-functions

向 readline 输出流打印所有的函数和它们的按键关联。如果给出一个数值参数，输出将被格式化，可以用作 inputrc 文件一部分。

dump-variables

向 readline 输出流打印所有可设置的 readline 函数。如果给出一个数值参数，输出将被格式化，可以用作 inputrc 文件一部分。

dump-macros

向 readline 输出流打印所有关联到宏的 readline 按键序列以及它们输出的字符串。如果给出一个数值参数，输出将被格式化，可以用作 inputrc 文件一部分。

display-shell-version (C-x C-v)

显示当前 bash 实例的版本信息。

Programmable Completion 可编程补全

当试图对一个命令的参数进行词的补全时，如果已经使用内建命令 complete 定义了这个命令的补全规则 (compspec)，将启动可编程补全功能 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。

首先，命令名被确认。如果针对这个命令有补全规则的定义，那么将使用规则来产生可能的词的补全的列表。如果命令词是一个路径全名，将首先搜索针对这个路径全名的规则。如果针对这个路径全名没有找到规则，将尝试查找针对最后一个斜杠后面的部分的规则。

一旦找到了一个规则，它将用作产生匹配的词。如果没有找到，将进行上面 Completing 中描述的 bash 默认的补全。

首 先，将执行规则指定的动作。只有以被补全的词开始的匹配词才会被返回。当在文件或目录名补全中使用 -f 或 -d 选项时，shell 变量 FIGNORE 将用于对匹配进行过滤。

接下来，将产生所有由-G 选项给出的文件名扩展模式指定的补全。模式产生的词不必匹配要补全的词。shell 变量 GLOBIGNORE 不会用于过滤匹配结果，但是 变量 FIGNORE 会被使用。

接 下来，将考虑 -W 选项的参数指定的字符串。这个字符串首先被划分，用特殊变量 IFS 中的字符作为分隔符。shell 引用被当作一个词。接下来，每个词被扩展 ， 使 用 上 面 EXPANSION 中描述的 brace expansion, tilde expansion, parameter 和 variable expansion, command substitution, arithmetic expansion, 以及 pathname expansion 规则处理。对于结果，再使用上面 Word Splitting 中描述的规则划分成词。扩展的结果与要补全的词进行前部一致的 比较，匹配的词成为可能的补全。

在这些匹配被产生后，任何由 -F 和 -C 选项指定的 shell 函数和命令将被执行 。当命令或函数被执行时，变量 COMP_LINE 和 COMP_POINT 被赋值，使用 上 面 Shell Variables 中 的 规 则 。 如 果 要执行 shell 函数，还将设置变量 COMP_WORDS 和 COMP_CWORD 当函数或命令被执行时，第一个参数是等待参数被补全 的命令的名称，第二个参数是要补全的词，第三个参数是当前命令行中，要补全的词前面的词。对要补全的词产生的补全不会进行任何过滤；函数或命令在 产生匹配时有完全的自由。

任何 -F 指定的函数将被首先执行。函数可以使用任何 shell 功能，包含内建命令 compgen，来产生匹配。它必须将可能的补全放到数组变量 COMPREPLY 中。

接下来，任何 -C 选项指定的命令将被执行，其执行环境与命令替换的环境相 同 。 它应当向标准输出打印一个补全的列表，每行一个。反斜杠可以用来转义一个新行符，如果需要的话。

所有可能的补全都产生之后，将对列表进行 -X 选项指定的任何过滤。过滤器 是一个模式，和路径名扩展中的一样；模式中的 & 替换为要补全的词。字面上的 & 可以用反斜杠转义；反斜杠在进行匹配时被删除。任何匹配这个模式的补全将 从列表中删除。前导的 ! 将使模式含义相反；这种情况下，任何不匹配这个模式的补全将被删除。

最后，B-P 和 -S 指定的任何前缀和后缀被添加到补全列表的每个成员后面， 结果返回给 readline 补全代码，作为可能的补全列表。

如果先前执行的动作没有产生任何匹配，并且在定义 compspec 规则时，为 complete 命令提供了 -o dirname 选项，将尝试目录名补全。

默认情况下，如果找到了一个规则，它产生的任何东西都被返回给补全代码， 作为可能的补全的全集。不再尝试默认的 bash 补全，readline 默认的文件名补全也会禁止。如果定义规则时，为 complete 命令提供了 -o default 选项，在 规则没有产生匹配时将进行 readline 默认的补全处理。

当 一个规则指出期望目录名补全时，可编程补全函数强制 readline 在补全的名称后面添加一个斜杠，如果它是一个到目录的符号连接。然后还要经过 readline 变量 mark-directories 的值处理， 不管 readline 变量 mark-symlinked-directories 的值是什么。

历史(HISTORY)

当启用内建命令 set 的 -o history 选项时，shell 允许访问 command history ， 以前输入的命令的列表。 HISTSIZE 的值用作命令列表中保存的命令数量。过去 HISTSIZE 个 (默认为 500) 命令将被保存。shell 将每条命令在进行参数和变量扩展之前保存到历史列表中 (参见上面的 EXPANSION 段落)，但是是在历史扩展进行之后，并且要经过 shell 变量 HISTIGNORE 和 HISTCONTROL 处理。

在启动时，历史根据以变量 HISTFILE 的值为名的文件 (默认 是 ~/.bash_history) 进行初始化。如果需要的话，以 HISTFILE 为名的文件将被截断，来包含不超过变量 HISTFILESIZE 的值指定的行数。当交互 shell 退 出 时 ， 最 后 $HISTSIZE 行被从历史列表中复制到 $HISTFILE 文件中。如果启用了 shell 选项 histappend (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节中对 内建命令 shopt 的描述)，这些行被追加到历史文件中，否则历史文件被覆盖。如果 HISTFILE 被取消定义，或者如果历史文件不可写，历史将不会保存。 保存历史之后，历史文件被截断，以包含不超过 HISTFILESIZE 行。如果 HISTFILESIZE 被取消定义，不会进行截断操作。

内建命令 fc (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节) 可以用来列出或修改并重新执行历史列表中的一部分。内建命令 history 可以用来显示或修改历史列表，操作历史文件。当使用命令行编辑时，每种编辑模式都 有搜索命令，提供对历史列表的访问。

shell 允许控制哪些命令被保存到历史列表中。可以设置 HISTCONTROL 和

HISTIGNORE 变量，来使得 shell 只保存输入命令的一个子集。shell 选项 cmdhist 如果被启用，将使得 shell 将多行的命令的每一行保存到同一个历史条目中，在需要的地方添加分号来保证语义的正确性。shell 选项 lithist 使得 shell 保存命令时，保留嵌入的新行而不是用分号代替。参见下面 shell 内建命令(SHELL BUILTIN COMMANDS) 中，内建命令 shopt 的描述，有关设置和取消 shell 选项的信息。

历史扩展("HISTORY EXPANSION")

shell 支持历史扩展机制，类似于 csh 中历史扩展。这一节描述了可用的语法特征。在交互的 shell 中这一机制被默认启用，可以使用内建命令 set 的 -H 选项来禁用它 (参见下面的 shell 内建命令(SHELL BUILTIN COMMANDS) 章节)。非交互的 shell 默认不进行历史扩展。

历史扩展将历史列表中的词引入输入流中，使得可以方便地重复已执行命令， 在当前输入行中为前一个命令插入新的参数，或者快速修正前一个命令中的错误。

历史扩展在读入一整行后，在 shell 将它拆分成词之前立即进行。它由两部分组成。首先是判断替换中使用历史列表中哪一行。其次是选择那一行中要包含到 当前 行中的部分。从历史中选择的行称为 event，从那一行中选择的部分是 words 。可以用多种多样的 modifiers 来操纵所选的词。在读入输入时，行被按照同样方式分解成词，因此多个以 metacharacter 分隔的词，如果被引号包含，就被当成一个词。历史扩展由历史扩展字符引入，默认是 !。只有反斜杠 (\) 和单引号可以引用历史扩展字符。

内 建命令 shopt 可以设定多个选项值，来调整历史扩展的行为。如果 shell 选项 histverify 被启用 (参见内建命令 shopt 的描述)，并且正在使用 readline ， 历 史替换不会被立即传给 shell 解释器。与此相对，扩展后的行被重新载入 readline 编辑缓冲区，进行进一步的修改。如果正在使用 readline，并且启 用了 shell 选项 histreedit，失败的历史替换将被重新载入到 readline 编辑缓冲区，进行改正。内建命令 history 的 -p 选项可以用来在执行之前查看历史扩展将如何进行。内建命令 history 的 -s 选项可以用来在历史列表末尾添加命令 ，而不真正执行它们，从而在接下来的调用中可以使用它们。

shell 允许控制历史扩展机制使用的多种字符 (参见上面的 Shell Variables 中 histchars 的描述)。

Event Designators

事件指示器 (event designator) 是一个对历史列表中某个命令行条目的引用。

! 开始一个命令替换，除非后面跟随的是 blank, newline, = 或是 (.

!n 引用命令行 n.
!-n 引用当前命令行减去 n.
!! 引用上一条命令。这是 ‘!-1’ 的同义词。
!string
引用最近的以 string 开始的命令。
!?string[?]
引 用 最近的包含 string 的命令。尾部的 ? 可以被忽略，如果 string 之后紧接着一个新行符。
^string1^string2^
快速替换。重复上一条命令，将 string1 替 换 为 string2. 与 ‘‘!!:s/string1/string2/’’ 等价 (参见下面的 修饰符 (Modifiers))。
!# 到此为止输入的整个命令行。

Word Designators
词指示器 (word designator) 用于从 event 中选择期望的词。 : 分 隔 event 规 则与 word 指示器。它可以忽略，如果词指示器以 ^, $, *, -, 或 % 开始。词被从行首开始编号，第一个词被表示为 0。插入当前行中的词以单个空格分 隔 。

0 (zero)
第 0 个词。对 shell 来将，这是命令名。
n 第 n 个词。
^ 第一个参数。也就是，第 1 个词。
$ 最后的参数。
% 最近一次搜索 ‘?string?’ 匹配的词。
x-y 一组词；‘-y’ 是 ‘0-y’ 的简写。
* 所有词，除了第 0 个。这是 ‘1-$’ 的同义词。如果 event 中只有一个词，使用 * 也不是错误；这种情况下将返回空字符串。
x* x-$ 的简写。
x- -$ 的简写，就像 x* 一样，但是忽略最后一个词。

如果给出了一个 word 指示器，没有给出 event 规则，前一个命令将用作 event 。

修饰符 (Modifiers)
可选的 word 指示器之后，可以出现一个或多个下述 modifiers 的序列，每一个都前缀有 ‘:’。

h 删除文件名组成的尾部，只保留头部。
t 删除文件名组成中前面的成分，保留尾部。

r 删除 .xxx 形式中尾部的后缀成分，保留基本名称部分。

e 删除所有内容，保留尾部的后缀。

p 打印新的命令，但是不执行它。

q 引用替换所得的词，使它不再进行替换。

x 引用替换所得的词，类似与 q, 但是会根据 blanks，空白和新行符分 解为词。

s/old/new/

将 事件行中出现的第一个 old 替换为 new。任何分隔符都可以用来代替 /，最后一个分隔符是可选的，如果它是事件行的最后一个字符。 old 和 new 中的分隔符可以用一个反斜杠来引用。如果 & 出现在 new 中，它将替换为 old。可以用单个反斜杠来引用 &。如果 old 为空，它将设置 为最 后 替 换 的 old ， 或 者 ， 如果前面没有发生过历史替换，就是 !?string[?] 搜索中的最后一个 string。

& 重复上一次替换。

g 使得改变被整个事件行所接受。用于与 ‘:s’ 或 ‘:&’ 结 合 (例 如 ， ‘:gs/old/new/’) 。如果与 ‘:s’ 结合使用，任何分隔符都可以用来代替 /，最后一个分隔符是可选的，如果它是事件行的最后一个字符。

shell 内建命令(SHELL BUILTIN COMMANDS)

除非另外说明，这一章介绍的内建命令如果接受 - 引导的选项，那么它也 接 受 -- 作为参数，来指示选项的结束

: [arguments]

没 有效果；这个命令除了扩展 arguments 并且作任何指定的重定向之外 ，不做任何事。退出时返回 0。

. filename [arguments]

source filename [arguments]

读取并在当前 shell 环境中执行 filename 中的命令，返 回 filename 中最后一个命令的返回状态。如果 filename 中不包含斜杠 (slash)，系统将在 PATH 中查找包含 filename 的目录。在 PATH 中搜索的文件不必是 可执行的。如果 bash 不是运行于 posix mode，当 PATH 中找不到文件时会在当前目录搜索。如果 shopt 内建命令的 sourcepath 选项被 关闭， PATH 将不会被搜索。如果有任何 arguments ，它们成为 filename 的位置参数 (positional parameters)，否则位置参数不发生变化。返回状 态是脚本中最后一个命令退出时的状态。没有执行命令则返回 0，没有找到或不能读取 filename 时返回 false。

alias [-p] [name[=value] ...]

Alias 不带参数或者带 -p 参数运行时将在标准输出以这样的格式 alias name=value 给出别名列表。如果有参数，将创建提供了 value 的 name

的别名。 value 中尾部的空格使得别名被扩展时，下一个词做别名替 换。对于参数列表中的每一个 name，如果 value 没有给出，这个别名的名称和值会被打印出来。 Alias 返回 true 除非 name 没有定义为别名。

bg [jobspec]

使挂起的程序 jobspec 在后台继续执行，就好像它是用 & 启动的一样。如果没有指定 jobspec，shell 意义上的 current job 当前作业 将被使用。 bg jobspec 返回 0，除非当前禁止了作业控制，或者在允许作业 控制，但是没有找到 jobspec ，或者它不是在作业控制下启动的时候。

bind [-m keymap] [-lpsvPSV]
bind [-m keymap] [-q function] [-u function] [-r keyseq]
bind [-m keymap] -f filename
bind [-m keymap] -x keyseq:shell-command
bind [-m keymap] keyseq:function-name
bind readline-command

显示当前 readline 键和功能的，将一个按键序列和一个 readline 功能或宏进行关联，或者设置一个 readline 变量。每一个在非选项的参数都是一个命令，好像它是在 .inputrc 中出现的一样。但是每个关联或者命令必须作 为 单 独 的 参 数 传 递 ； 也 就 是 这 样 ’ "\C-x\C-r": re-read-init-file’。如果有参数，它们有如下的意义：

-m keymap

使 用 keymap 作为随后的关联的 keymap。可选的 keymap 名称是 emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command，还有 vi-insert。 vi 和 vi-command 等价; emacs 和 emacs-standard 等价。

-l 列出所有的 readline 功能。

-p 以程序可读的方式显示 readline 功能名称和关联

-P 列出当前 readline 功能名称和关联。

-v 以程序可读的方式显示 readline 变量名称和值

-V 列出当前 readline 变量和值。

-s 以程序可读的方式显示 readline 键序列和对应的宏

-S 显示 readline 宏对应的键序列和他们输出的字符串

-f filename

从 filename 中读取键序列

-q function

查询那些键将执行 function。

-u function

取消所有关联到 function 的键。

-r keyseq

取消当前任何 keyseq 的关联。

-x keyseq:shell-command

使 shell-command 在 keyseq 按下时被执行。

返回值是 0，除非给出了一个不能识别的选项或是产生了一个错误。

break [n]

从一个 for, while, until, 或者 select 循环退出。如果指定了 n ，就 跳出 n 层循环。 n 必须 ≥ 1。如果 n 比当前循环层数还要大，将跳出所有循环。返回值是 0，除非执行 break 的时候 shell 不是在执行 一个循环。

builtin shell-builtin [arguments]

执行指定的 shell 内建命令，传递 arguments ，返回命令的返回值。这在定义了一个和 shell 内建命令同名的函数时很有用，在那个函数中 使用 它 来 执行相应的功能。cd 命令常以这种方式重新定义。返回状态是 false，如果指定的 shell-builtin 并不是一个 shell 内建命令。

cd [-L|-P] [dir]

改变当前路径到 dir。这个变量的默认值是 HOME 目 录 。 环 境 变 量 CDPATH 定义了包含 dir 的搜索路径。在 CDPATH 中可选的路径名以冒号(:) 分隔。 CDPATH 中的空路径名与当前路径相同，就是 ‘ ‘.’ ’ . 如果 目录名以斜杠 (/,slash) 起始，那么 CDPATH 不会被使用。 -P 选项 是说使用物理路径结构而不是跟随符号链接，(参见 set 命令中的 -P 选项); -L 选项强制跟随符号链接。另外，选项 - 与 $OLDPWD 是相同的。返回值是 true ，如果成功地改变了目录；否则是 false。

command [-pVv] command [arg ...]

运行 command ，使用 args 作为参数，禁止通常的查找 shell 函数的过程。只有内建命令或者 PATH 中包含的命令可以执行。如果给出 -p 参数 ， command 的查找是以 PATH 的默认值进行的。这样可以保证找到所 有的 标准工具。如果给出 -V 或者 -v 选项，关于 command 的说明将被打印出来。 -v 选项使得表述这个命令的词，或者要执行 command 需要 执行 的文件显示出来； -V 选项给出更详细的描述。如果给出 -V 或者 -v 选项，退出状态在找到了 command 的情况下 0，没找到就是 1。如果没 有提供选项，并且产生了错误或者 command 没有找到，退出状态就是 127。否则， command 内建命令的退出状态是 command 的退出状态。

compgen [option] [word]

根据 option 为 word 产生可能的补全。option 是内建命 令 complete

接受的任何选项，除了 -p 和 -r，将匹配结果写到标准输出。当使用 -F 或 -C 选项时，可编程补全功能所设置的多数 shell 变量如果存在， 其值将不再有用。

产生的匹配与可编程补全代码根据补全规则加上相同的标志直接产生的结果相同。如果指定了 word，只有匹配 word 的补全结果将被显示出来。

返回值为真，除非提供了非法的选项，或者没有产生匹配。

complete [-abcdefgjksuv] [-o comp-option] [-A action] [-G globpat] [-W wordlist] [-P prefix] [-S suffix]
[-X filterpat] [-F function] [-C command] name [name ...]
complete -pr [name ...]

指定每个 name 的参数应当如何被补全。如果给出了 -p 选项，或者没有选项给出，现有的补全规则将被显示出来，以一种可以重用为输入的格式显示。-r 选项将一个针对每个 name 的补全规则删除。或者，如果没 有给出 name，将删除所有补全规则。

尝试词的补全时，应用这些补全规则的过程在上面 Programmable Completion(可编程补全) 中详述。

其他选项，如果给出的话，具有下列意义。-G, -W, 和 -X 选项的参数 (如果需要的话，还包括 -P 和 -S 选项) 应当被引用，避免在执行内建命令 complete 之前被扩展。

-o comp-option

comp-option 控制着 compspec 除了简单地产生补全之外的多种行为。 comp-option 可以是如下之一：

default 使用 readline 的默认文件名补全，如果 compspec 没有得到匹配。

dirnames

进行目录名补全，如果 compspec 没有得到匹配。

filenames

告诉 readline，compspec 产生了文件名，使它可以进行任何文件名专用的处理 (例如，给目录名加上斜杠或消除尾部空白)。主要用于 shell 函数。

nospace 告诉 readline 不要向补全的词在行的最后添加一个空格 (这是默认行为)。

-A action

action 可以是下列之一，来产生一系列可能的补全结果：

alias 起别名。也可以用 -a 指定。

arrayvar
数组变量名。
binding Readline 按键关联。
builtin shell 内建命令的名称。也可以用 -b 指定。
command 命令名。也可以用 -c 指定。
directory
目录名。也可以用 -d 指定。
disabled
被禁用的内建命令名称。
enabled 启用的内建命令名称。
export 被导出的 shell 变量名称。也可以用 -e 指定。
file 文件名。也可以用 -f 指定。
function
shell 函数的名称。
group 组名。也可以用 -g 指定。
helptopic
内建命令 help 接受的帮助主题。
hostname
主机名，从环境变量 HOSTFILE 指定的文件中得到。
job 作业名，如果作业控制被激活的话。也可以用 -j 指定
。
keyword shell 保留字。也可以用 -k 指定。
running 正在运行的作业名，如果作业控制被激活的话。
service 服务名。也可以用 -s 指定。
setopt 内建命令 set 的 -o 选项的有效参数。
shopt 内建命令 shopt 接受的 shell 选项名。
signal 信号名。
stopped 停止的作业名，如果作业控制被激活的话。
user 用户名。也可以用 -u 指定。
variable
shell 变量的名称。也可以用 -v 指定。

-G globpat
文件名扩展模式 globpat 被扩展，产生可能的补全。

-W wordlist
wordlist 被使用 IFS 特殊变量中的字符作为定界符来拆分，每
个结果的词被扩展。可能的补全是结果列表中匹配要补全的词的
那一些。

-C command
command 将在一个子 shell 环境中执行，它的结果用作可能 的
补全。

-F function

shell 函数 function 将在当前 shell 环境中执行。当它结束时，可能的补全可以从数组元素 COMPREPLY 中得到。

-X filterpat

filterpat 是一个模式，用于文件名扩展。所有前面的选项和参数产生的可能的补全都要经过这一步处理，每一个匹配 filterpat 的补全都被从列表中删除。为 filterpat 加上前导 ! 使模式 意义相反；这种情况下，所有不匹配 filterpat 的模式被删除。

-P prefix

在所有其他选项都处理过之后，prefix 被加到每个可能的补 全前面。

-S suffix

在 所有其他选项都处理过之后，suffix 被加到每个可能的补全后面。

返回值为真，除非给出了非法的选项，给出除 -p 和 -r 之外的某个选项时没有给出 name 参数，试图删除一条 name 的补全规则但是规则不存在，或者添加补全规则时出错。

continue [n]

复位到外层 for, while, until, 或 select 循环的下一次开始。如果指定 了 n，复位到向外第 n 层循环的开始。 n 必须 ⩾ 1。如果 n 比外部循环的层数要多，将复位到最外层的循环 (‘‘top-level’’ loop，顶层循环)。返回值是 0，除非执行 continue 时，shell 不是在循环之中。

declare [-afFirtx] [-p] [name[=value]]

typeset [-afFirtx] [-p] [name[=value]]

声 明变量且/或设置它们的属性。如果没有给出 name 则显示变量的值。选项 -p 将显示每个名称 name 的属性和值。当使用 -p 时，其他选项被忽略。选项 -F 禁止显示函数定义；只有函数名和属性会被显示。 -F 选项暗含 -f. 下列选项可用来限制只输出具有指定属性的变量，或者为变量设置属性：

-a 每个 name 都是数组变量 (参见上面的 Arrays 段落)。

-f 只使用函数名。

-i 变量被当作一个整数；当变量被赋值时将进行算术运算 (参见 算术求值 (ARITHMETIC EVALUATION) 章节)。

-r 使得 name 只读。这些名称不能再被后续的赋值语句赋值或取 消定义。

-t 设置每个 name 的 trace(跟踪) 属性。被跟踪的函数继承了调用

者 shell 的 DEBUG 陷阱。trace 属性对变量没有特殊意义。

-x 标记 name 为可以通过环境导出给后续命令。

使用 ‘+’ 代替 ‘-’ 将关闭属性，特殊情况是 +a 不能用于销毁一个数组变量。当用于函数中时，它使得每个 name 成为局部的，就像使用了 local 命令。返回值是 0，除非遇到了非法的选项，试图使用 ‘‘-f foo=bar’’ 定义函数，试图向只读变量赋值，试图向数组变量赋值但没有使用复合的赋值语法 (参见上面的 Arrays 段落)，name 之一不是有效的 shell 变量名，试图将数组变量的数组状态关闭，或者是试图使用 -f 显示一个不存在的函数。

dirs [-clpv] [+n] [-n]

没有选项时显示当前保存的目录。默认输出为一行，目录名用空格分开。可以使用 pushd 命令将目录添加到列表，popd 命令将列表中的条目删除。

+n 显示 dirs 在不带选项执行时显示的列表的第 n 个条目，从 0 开始自左算起。

-n 显示 dirs 在不带选项执行时显示的列表的第 n 个条目，从 0 开始自右算起。

-c 删除所有条目，清空目录栈。

-l 产生长列表；默认列表格式使用波浪线来表示个人目录。

-p 输出目录栈，一行一个。

-v 输出目录栈，一行一个，每个条目前面加上它在栈中的位置索引。

返回值是 0，除非给出了非法的参数，或者 n 索引超出了目录栈的范围。

disown [-ar] [-h] [jobspec ...]

没有选项时，每个 jobspec 被从正在运行的作业表中删除。如果给出了 - 选项，每个 jobspec 并不从表中删除，而是被标记，使得在 shell 接到 SIGHUP 信号时，不会向作业发出 SIGHUP 信号。如果没有给出 jobspec，也没有给出 -a 或者 -r 选项，将使用当前作业 (current job)。如果没有给出 jobspec，选项 -a 意味着删除或标记所有作业；选项 -r 不带 jobspec 参数时限制操作只对正在运行的作业进行。返回值是 0，除非 jobspec 不指定有效的作业。

echo [-neE] [arg ...]

输出 arg，以空格分开，最后加一个新行符。返回值总是 0。如果指定了 -n，将不在尾部添加新行符。如果给出了 -e 选项，将允许解释下列反斜

杠转义的字符。 -E 选项禁止这些转义字符的解释，即使在默认解释它们的系统中也是如此。 shell 选项 xpg_echo 可以用来在运行时判断 echo 是否默认展开这些转义字符。 echo 不将 -- 作为选项的结束。 echo 解释下列转义序列：

\a alert (bell) 响铃
\b backspace 回退
\c suppress trailing newline 删除尾部新行符
\e an escape character 字符 Esc
\f form feed 进纸
\n new line 新行符
\r carriage return 回车
\t horizontal tab 水平跳格
\v vertical tab 竖直跳格
\\ backslash 反斜杠
\0nnn 一个八比特字符，它的值是八进制值 nnn (零到三个八进制数字)。
\nnn 一个八比特字符，它的值是八进制值 nnn (一到三个八进制数字)。
\xHH 一个八比特字符，它的值是十六进制值 HH (一到两个十六进制数字)。

enable [-adnps] [-f filename] [name ...]

允许或禁止 shell 内建命令。禁止一个内建命令使得磁盘上的与内建 命令 同名的文件得以运行，不必使用它的全路径，即使 shell 一般在搜索磁盘上的命令之前搜索内建命令。如果使用了 -n 选项，每个 name 都被禁止；否则， name 被允许。例如，要使用 PATH 中搜索到的 test 命令而不是 shell 内建的那一个，可以运行 ‘‘enable -n test’’. 选项 -f 意 味着从共享库 filename 中加载新的内建命令 name，如果系统支持动态加载的话。选项 -d 将删除曾经用 -f 加载的内建命令。如果没有给出 name 参数，或者给出了 -p 选项，将显示 shell 内建命令的列表。如果没有其他选项参数，这个列表只包含所有被允许的 shell 内建命令； 如果给出了 -n，将只显示被禁止的内建命令；如果给出了 -a，显示的列表中包含所有内建命令，还有命令是否被允许的指示；如果给出了 -s， 输出 被 限 制为 POSIX special 内建命令。返回值是 0，除非 name 不是 shell 内建命令，或者从共享库中加载新的内建命令时出错。

eval [arg ...]

arg 被读取并连结为单一的命令。这个命令然后被 shell 读取并执行 ，它 的退出状态被作为 eval 的值返回。如果没有 args，或仅仅包含空参数， eval 返回 0。

exec [-cl] [-a name] [command [arguments]]

如果指定了 command，它将替换 shell。不会产生新的进程。 arguments 成为 command 的参数。如果给出了 -l 选项，shell 将在传递给 command 的第 0 个参数前面加上一个连字符 (dash, ‘-’)。这样做和 login(1) 相同。选项 -c 使得命令 command 在一个空环境中执行。如果给出了 -a， shell 会将 name 作为第 0 个参数传递给要执行的命令。如果由于某种原因 as the zeroth argument to the executed command. If command 不能被执行，非交互的 shell 将退出，除非 shell 选项 execfail 被设置为允许，这种情况下它返回失败。如果命令不能执行，交互的 shell 返回失败。如果没有指定 command 任何重定向对当前 shell 发生作用，返回值是 0。如果发生重定向错误，返回状态是 1。

exit [n]

使得 shell 以状态值 n 退出。如果忽略了 n，退出状态是最后执行的命令的退出状态。在 shell 终止前，对 EXIT 的陷阱将被执行。

export [-fn] [name[=word]] ...

export -p

给出的名称 names 被标记为自动地导出到后续执行的命令的环境中。如果给出了 -f 选项，名称 names 指的是函数。如果没有给出 names，或者如果给出了 -p 选项，将打印在这个 shell 中被导出的所有名字的列表。选项 -n 使得以此为名的变量的导出属性被删除。 export 返回 0，除非遇到了非法的选项，name 之一不是有效的 shell 变量名，或者给出了 -f 选项，而 name 不是一个函数。

fc [-e ename] [-nlr] [first] [last]

fc -s [pat=rep] [cmd]

命令修复。第一种形式中，历史列表中从 first 到 last 范围内的命令都被选取。 First 和 last 可以指定为字符串 (可以定位最后一个以此字符串开始的命令) 或者数字 (历史列表中的索引，负数被当作相对当前命令号的偏移)。如果没有指定 last，它在列举时被设为当前命令 (因此 ‘‘fc -l -10’’ 将输出最后 10 条命令)，其他情况下被设为 first。如果没有指定 first，它在编辑时被设为前一个命令，列举是设为 -16。

选项 -n 使得列举时不显示命令号码。选项 -r 将命令顺序进行掉换。如果给出了 -l 选项，命令将列举在标准输出上。否则，将启动 ename 给出的编辑器，编辑包含这些命令的文件。如果没有给出 ename，将使用变量 FCEDIT 的值，如果 FCEDIT 没有定义就使用 EDITOR 的值。如果仍然没有定义，将使用 vi。编辑结束后，被编辑的命令将回显并执行。

第二种形式中，command 在每个 pat 的实例被 rep 替换后都被重新执行。使用这种特性时可以起一个有用的别名：‘‘r=fc -s’’, 这样输入 ‘‘r cc’’ 将运行最后的以 ‘‘cc’’ 开头的命令，输入 ‘‘r’’ 将重新执行上一个命令。

如果使用第一种形式，返回值是 0，除非遇到了非法的选项，或 first 或 last 指定的历史行数超出了范围。如果给出了 -e 选项，返回值是最后执行的命令的返回值，或着是失败，如果临时文件中的命令执行出错。如果使用第二种形式，返回状态是重新执行的命令，除非 cmd 没有指定一个有效的历史行，这种情况下 fc 返回失败。

fg [jobspec]

将 jobspec 恢复至前台，使它成为当前作业。如果 jobspec 不存在，将使用 shell 意义上的当前作业 current job。返回值是被放到前台的命令的状态，或者是失败，如果在禁用作业控制时运行，或者在启用作业控制时运行，但 jobspec 没有指定有效的作业，或 jobspec 指定了没有使用作业控制的作业。

getopts optstring name [args]

getopts 由 shell 程序用来处理位置参数。optstring 包含要识别的选项字符；如果某个字符跟随着冒号，那么这个选项需要一个参数，需要用空白和它隔离开。冒号和问号字符不能用作选项字符。每次它执行时，getopts 将下一个选项放在 shell 变量 name 中，如果 name 不存在就初始化它；下一个要处理的参数的索引放在变量 OPTIND 中。每次 shell 或 shell 脚本被执行的时候 OPTIND 被初始化为 1。当某个选项需要参数时，getopts 将那个参数放到变量 OPTARG 中。shell 不会自动重置 OPTIND；在相同的 shell 中，如果要使用新的参数集合而需要多次调用 getopts 时，必须手动重置它。

当遇到选项结束的时候，getopts 以大于 0 的值退出。OPTIND 被设置为第一个非选项的参数的索引，name 被设置为 ?。

getopts 通常解释位置参数，但是如果 args 中给出了更多参数，getopts 将解释它们。

getopts 能以两种方式报告错误。如果 optstring 的第一个字符是冒号，将使用 silent 安静的错误报告。通常的操作中，遇到非法选项或缺少选项的参数时将打印出诊断信息。如果变量 OPTERR 被设置为 0，不会显示错误消息，即使 optstring 的第一个字符不是冒号。

如果发现了一个非法的选项， getopts 向 name 中置入 ?，并且如果 不是 安 静模式的话，打印错误消息并取消 OPTARG 的定义。如果 getopts 是安静模式，找到的选项字符将置入 OPTARG，不会打印诊断消息。

如果没有找到需要的参数，并且 getopts 不是安静模式，将向 name 置入一个问号 (?)，取消 OPTARG 的定义，打印出诊断消息。如果 getopts 是安静模式，那么将向 name 置入一个冒号 (:) 并且 OPTARG 将设置 为找到的选项字符。

getopts 返回真，如果找到了指定的/未被指定的选项。它返回假，如果遇到了选项结束或者发生了错误。

hash [-lr] [-p filename] [-dt] [name]

对于每个 name, 通过搜索 $PATH 中的目录，找到命令的全路径名并记录它。如果给出了 -p 选项，不会进行路径搜索，直接将 filename 作为命令的全路径名。选项 -r 使得 shell 忘记所有已记录的位置。选 项 -d 使 得 shell 忘记已记录的 name 的位置。如果给出了 -t 选项，每个 name 对应的全路径名被打印出来。如果给出多个 name 作为 -t 的参 数 ，name 将在已记录的全路径名之前被打印出来。选项 -l 使得输出以一种可以重用为输入的格式显示。如果没有给出参数，或者只给出了 -l 选项，已记录的命令的信息将被打印出来。返回真，除非 name 没有找到或给出了非法的选项。

help [-s] [pattern]

显示关于内建命令的有用的信息。如果指定了 pattern (模式) ， help 给 出关于所有匹配 pattern 的命令的详细帮助；否则所有内建命令的帮助和 shell 控制结构将被打印出来。选项 -s 限制信息显示为简短的 用法概要。返回 0，除非没有匹配 pattern 的命令。

history [n]
history -c
history -d offset
history -anrw [filename]
history -p arg [arg ...]
history -s arg [arg ...]

不 带选项的话，显示带行号的命令历史列表。列出的行中含有 * 的已经被修改过。参数 n 使得只显示最后 n 行。如果给出了 filename，它 被用做历史文件名；没有的话，将使用 HISTFILE 的值作为历史文件名。选项如果给出，则具有下列意义：

-c 清空历史列表，删除所有条目。

-d offset

删除 offset 位置的历史条目。

-a 将 ‘‘新’’ 的历史条目 (自当前 bash 会话开始输入的历史命令) 追加到历史文件中。

-n 将尚未从历史文件中读取的历史条目读入当前历史列表。这些行是当前 bash 会话开始之后，才追加到历史文件中的行。

-r 读取历史文件的内容，使用它们作为当前历史。

-w 将当前历史列表写入历史文件，覆盖历史文件的原有内容。

-p 对后续的 args 进行历史替换，在标准输出上显示结果。不会 将结 果存入历史列表。每个 args 都必须被引用，来禁止普通的命令扩展。

-s 将 args 保存到历史列表中，作为单独的条目。历史列表中的 最后一个命令在添加 args 之前被删除。

返 回 0，除非遇到了非法的选项，读/写历史文件发生错误，在 -d 的参数中给出了无效的 offset，或者对 -p 的后续参数进行历史扩展失败。

jobs [-lnprs] [jobspec ...]

jobs -x command [args ...]

第一种形式列出正在运行的作业。选项具有下列意义：

-l 普通信息之外，列出进程 ID。

-p 只列出作业的进程组 leader 的进程 ID。

-n 只显示从上次用户得知它们的状态之后，状态发生改变的作业 的信息。

-r 限制只输出正在运行的作业。

-s 限制只输出停止的作业。

如 果给出了 jobspec 输出被限制为仅此作业的信息。返回 0，除非遇到了非法的选项或给出了非法的 jobspec。

如果给出了 -x 选项，作业 jobs 将 command 或 args 中的任 何 jobspec 替换为相应的进程组 ID，执行 command，传递参数 args 给它并返回它的退出状态。

kill [-s sigspec | -n signum | -sigspec] [pid | jobspec] ...

kill -l [sigspec | exit_status]

向以 pid 或 jobspec 为名的进程发送名为 sigspec 或 signum 的信 号 。 sigspec 可以是一个信号名称，类似 SIGKILL 或信号编号； signum 是一个信号编号。如果 sigspec 是一个信号名称，那么可以有，也可 以

没有 SIG 前缀。如果没有给出 sigspec，那么假设是 SIGTERM。参数 -l 将列出所有信号的名称。如果给出 -l 时还有任何参数，将列出参数对应的 信号名称，返回状态 0。 -l 的 exit_status 参数是一个数字，指定了一个信号编号或被信号终止的进程的退出状态值。 kill 返回真，如果至少成功发送了一个信号，或者返回假，如果发生了错误或遇到了非法的选项。

let arg [arg ...]

每个 arg 都是要求值的算术表达式 (参见 算术求值 (ARITHMETIC EVALUATION) 章节)。如果最后一个参数 arg 求值结果是 0， let 返回 1；否则返回 0。

local [option] [name[=value] ...]

对每个参数将创建一个名为 name 的局部变量并赋予值 value。 option 可以是任何 declare 接受的值。当 local 用于函数内部时，它使得变量 name 作用域局限于函数和它的子进程。没有操作数时， local 将局部变量 的列表写到标准输出。不在函数内部使用 local 会导致出错。返回 0 ，除非在函数之外使用了 local，给出了非法的 name，或者 name 是 一个只读的变量。

logout 退出登录 shell。

popd [-n] [+n] [-n]

从目录栈中删除条目。没有参数的话，从栈中删除顶层目录，执行 cd 切换到新的顶层目录。如果给出了参数，有下列的含义：

+n 删除 dirs 给出的列表中从左数第 n 个条目 (从 0 算起)。例如：‘‘popd +0’’ 删除第一个目录，‘‘popd +1’’ 第二个。

-n 删除 dirs 给出的列表中从右数第 n 个条目 (从 0 算起)。例如：‘‘popd -0’’ 删除最后一个目录，‘‘popd -1’’ 删除倒数 第二个。

-n 阻止从栈中删除目录之后改变目录，这时只对栈进行操作。

如 果命令 popd 成功，还要执行一个 dirs，返回 0。 popd 返回假，如果遇到了非法的选项，目录栈为空，指定了目录栈中不存在的条目，或者改变目录失败。

printf format [arguments]

在 format 控制下将格式化的 arguments 写到标准输出。 format 是一个字符串，包含三种类型的对象：普通字符，被简单地复制到标准输出，转义字符，被转换并复制到标准输出，格式说明，每一个都使得相邻的下

一个 argument 被打印出来。在标准的 printf(1) 格式之外，%b 使得 printf 展开相应 arguments 中的反斜杠转义序列，%q 使得 printf 将相应的 argument 以一种可以重用为 shell 输入的格式输出。

format 在需要时被重用，以处理所有的 arguments。如果 format 需要比所提供的更多的 arguments，多出的格式说明视为已经提供了相应的 0 值或空字符串。成功的话返回值是 0，失败则是非 0 值。

pushd [-n] [dir]
pushd [-n] [+n] [-n]

将目录推入目录栈，或者轮换栈中的内容，使栈的顶部成为当前工作目录。没有参数时，交换顶部两个目录，返回 0，除非目录栈为空。如果给出了参数，它们有如下含义：

+n 轮换栈中内容，使得 dirs 给出的列表中从左数第 n 个目录 (从 0 数起) 成为目录栈的顶部。

-n 轮换栈中内容，使得 dirs 给出的列表中从右数第 n 个目录 (从 0 数起) 成为目录栈的顶部。

-n 阻止向栈中添加目录之后改变目录，这时只对栈进行操作。

dir 添加 dir 到栈顶，使得它成为新的当前工作目录。

如果命令 pushd 成功，还要执行一个 dirs。如果使用第一种形式，pushd 返回 0，除非 cd 切换到目录 dir 失败。使用第二中形式时，pushd 返回 0，除非目录栈为空，指定了目录栈中不存在的元素，或者切换到指定的新的当前目录失败。

pwd [-LP]

打印当前工作目录的绝对路径名。如果给出了 -P 选项，或者设置了内建命令 set 的 -o physical 选项，打印出的路径名中不会包含符号链接。如果使用了 -L 选项，打印出的路径中可能包含符号链接。返回 0，除非在读取当前目录名时出错或给出了非法的选项。

read [-ers] [-u fd] [-t timeout] [-a aname] [-p prompt] [-n nchars] [-d delim] [name ...]

从标准输入读入一行，或从 -u 选项的参数中给出的文件描述符 fd 中读取，第一个词被赋予第一个 name, 第二个词被赋予第二个 name, 以此类推，多余的词和其间的分隔符被赋予最后一个 name. 如果从输入流读入的 词数比名称数少，剩余的名称被赋予空值。 IFS 中的字符被用来将行拆分成词。反斜杠字符 (\) 被用于删除读取的下一字符的特殊含义， 以及续行。如果给出了选项，将包含下列含义：

-a aname

词 被赋以数组变量 aname 的连续的下标，从 0 开始。在赋新值之前， aname 被取消定义。其他 name 参数被忽略。

-d delim

delim 的第一个字符被用于结束输入行，而不是新行符。

-e 如果标准输入来自终端，将使用 readline (参见上面 的 READ-LINE 章节) 来获得输入行。

-n nchars

read 读入 nchars 个字符后返回，而不是等待一整行输入。

-p prompt

读取任何输入之前，在标准错误显示提示 prompt，末尾没有新行符。提示只有在输入来自终端时才会显示。

-r 反斜杠不作为转义字符。反斜杠被认为行的一部分。特殊地， 一对反斜杠-新行符不作为续行。

-s 安静模式。如果输入来自终端，字符将不会回显。

-t timeout

使得 read 超时并返回失败，如果在 timeout 秒内没有读入完整的一行输入。如果 read 不是从终端或管道读取输入，那么这 个选项无效。

-u fd 从文件描述符 fd 中读取输入。

如果没有给出 names，读取的一行将赋予变量 REPLY。返回值是 0，除非遇到了 EOF，readP 超时，或给出了非法的文件描述符作为 -u 的参数。

readonly [-apf] [name ...]

给 出的 name 将被标记为只读的； names 的值不能被后来的赋值语句改变。如果给出了 -f 选项，names 对应的函数也被标记。选项 -a 限制变量只能是数组类型。如果没有给出 name 参数，或者如果给出了 -p 选项，将打印所有只读的名称。选项 -p 使得输出以一种可以被重新用作输入的 格式显示。返回值是 0，除非遇到了非法的选项， names 之一不是有效的 shell 变量名，或选项 -f 中给出的 name 不是一个函数。

return [n]

使得一个函数以指定值 n 退出。如果忽略了 n，返回状态是函数体中 执行 的 最后一个命令的退出状态。如果在函数外使用，但是是在一个以 . (source) 命令执行的脚本内，它使得 shell 中止执行脚本，返回 n 或 脚本中执行的最后一个命令的退出状态。如果在函数外使用，并且不是在以 . 执行的脚本内，返回状态是假。

set [--abefhkmnptuvxBCHP] [-o option] [arg ...]

不带选项时，shell 变量的名称和值将以一种可以重用为输入的格式显示

。输出根据当前语言环境进行排序。指定了选项的时候，它们设置或取消了 shell 的属性。处理完选项之后剩余的任何参数都被作为位置参数 的值 被赋值，分别赋予 $1, $2, ... $n. 如果给出了选项，那么具有以下含义：

-a 自动将被修改或创建的变量和函数标志为导出至后续命令的环境中。

-b 后台作业结束时立即报告状态，而不是在下次显示主提示符前报告。只有在启用作业控制时才有效。

-e 立即退出，如果 simple command (简单命令， 参 见 上 面 的 SHELL GRAMMAR 语法) 以非零值退出。shell 不会退出，如果失败的命令是 until 或 while 循环的一部分， if 语句的一部分 ， && 或 ｜｜ 序列的一部分，或者命令的返回值是由！ 翻转得到。针对 ERR 的陷阱，如果设置的话，将在 shell 退出前执行 。

-f 禁止路径扩展。

-h 在查找并执行命令时，记住它们的位置。这是默认启用的。

-k 所有以赋值语句形式出现的参数都被加入到命令执行的环境中，不仅是命令名前面那些。

-m 监视模式。作业控制被启用。在支持这个选项的系统中，它在交互 shell 中是默认启用的 (参见上面的 JOB CONTROL 作业控制)。后台进程在单独的进程组中运行，结束时将打印出包含 它们退出状态的一行信息。

-n 读取命令，但不执行。这可以用在检查 shell 脚本中的语法错误。交互 shell 中它被忽略。

-o option-name

option-name 可以是如下之一：

allexport
与 -a 相同。

braceexpand
与 -B 相同。

emacs 使用 emacs 样式的命令行编辑界面。这个选项在交 互 shell 中默认启用，除非 shell 以 --noediting 选项 启动。

errexit 与 -e 相同。

hashall 与 -h 相同。

histexpand
与 -H 相同。

history 允许记录命令历史，如上述 HISTORY 中的描述。这 个选项在交互 shell 中默认启用。

ignoreeof

它的效果是好像已经执行了 shell 命令
‘‘IGNOREEOF=10’’ 一样 (参见上面的 Shell Variables 变量)。

keyword 与 -k 相同。

monitor 与 -m 相同。

noclobber
与 -C 相同。

noexec 与 -n 相同。

noglob 与 -f 相同。 nolog 当前被忽略。

notify 与 -b 相同。

nounset 与 -u 相同。

onecmd 与 -t 相同。

physical
与 -P 相同。

posix 如果默认操作与 POSIX 1003.2 不同的话，改变 bash 的行为，来满足标准 (posix mode)。

privileged
与 -p 相同。

verbose 与 -v 相同。

vi 使用 vi 样式的命令行编辑界面。

xtrace 与 -x 相同。

如果给出了不带 option-name 的 -o 选项，当前选项的值将被打印出来。如果给出了不带 option-name 的 +o 选项，将在标准输出显示一系列可以重建当前选项设定的 set 命令。

-p 打开 privileged mode (特权模式)。在这个模式中，不会处理 $ENV 和 $BASH_ENV 文件，shell 函数不会从环境中继承，环境中如果有变量 SHELLOPTS，也将被忽略。如果 shell 启动时的有效用户(组) ID 与真实用户(组) ID 不同，并且没有给出 -p 选项，将执行这些操作，有效用户 ID 将设置为真实用户 ID 。如果启动是给出了 -p 选项，有效用户 ID 不会被重置。将这个选项关闭使得有效用户和组 ID 被设置为真实用户和组 ID。

-t 读取并执行一个命令之后退出。

-u 在进行参数扩展时，将未定义的变量作为错误。如果试图扩展未定义的变量，shell 将输出一条错误消息; 如果是非交互的 shell，shell 将以非零值退出。

-v 在读取输入的同时打印出来。

-x 扩展每个简单命令之后，显示 PS4 的值，接着显示命令和它扩展后的参数。

-B shell 执行花括号扩展 (参见上面的 Brace Expansion)。这是默认允许的。

-C 如果设置的话， bash 使用重定向操作符 >, >&, 和 <> 时，不会 覆盖已存在的文件。可以使用重定向操作符 >| 代替 > 来创建输出文件，从而绕过这个限制。

-H 允许 Enable ! 样式的历史替换。在交互 shell 中这个选项是默认启用的。

-P 如果设置的话，shell 在执行类似 cd 的，改变当前工作目录的命令时，不会跟随符号连接。它将使用物理的目录结构来代替。默认情况下， bash 在执行改变当前目录的命令时跟随路径的逻辑链。

-- 如果这个选项没有参数，将取消位置参数的定义。否则，位置参数将设置为 arg，即使它们以 - 开始。

- 通 知 信号的结束，使得所有剩余的 arg 被赋予位置参数。 -x 和 -v 选项被关闭。如果没有 arg，位置参数将不会改变。

这个选项默认是关闭的，除非另外说明。使用 + 而不是 - 使得这些选项被 关闭。选项都可以作为参数，在 shell 启动时指定。当前的选项集合可以从 $- 找到。返回值总是真，除非遇到了非法的选项。

shift [n]

从 n+1 ... 开始的选项被重命名为 $1 .... 从 $# 向下直 到 $#-n+1 的 选项被取消定义。 n 必须是非负整数，小于或等于 $#。如果 n 是 0 ，不会改变参数。如果没有给出 n，就假定它是 1。如果 n 比 $# 大，位 置参数不会改变。返回值大于 0，如果 n 比 $# 大或小于 0；否则返回 0。

shopt [-pqsu] [-o] [optname ...]

对于控制可选的 shell 行为的变量，改变它们的值。没有选项或者有 -p 选项时，将显示所有可设置的选项列表，以及它们是否已经设置的指示。-p 使得输出以一种可以被重用为输入的形式显示。其他选项有如下含 义 :

-s 允许(设置) 每个 optname。

-u 禁止(取消) 每个 optname。

-q 禁止通常的输出 (安静模式)；返回状态指示了 optname 是否被设置。如果对 -q 给出了多个 optname 参数，如果所有 optname 都被允许，返回值就是 0；否则返回非零值。

-o 限制 optname 的值为内建命令 set 的 -o 选项定义的值。

如 果使用 -s 或 -u 时没有给出 optname 参数，显示将分别限于被设置或被取消的选项。除非另外说明，shopt 选项默认被禁止(取消)。

返回值在列出选项时是 0，如果所有 optname 都被允许的话，否则是 非零 值 。 当 设 置或取消选项时，返回值是 0，除非 optname 是非法的 shell 选项。

shopt 选项的列表是：

cdable_vars
如果设置的话，内建命令 cd 的参数如果不是目录，就假定是一个变量，它的值是要切换到的目录名。

cdspell 如果设置的话， cd 命令中目录的细微拼写错误能够得以纠正。检查的错误包括字符错位，缺字符，重复输入同一字符。如果找到了正确的值，将打印正确的文件名，命令将继续。这个选项只能在交互 shell 中使用。

checkhash
如果设置的话，bash 在执行命令前检测散列表中的命令是否 存在。如果一个被散列的命令不再存在，将进行正常的路径搜索。

checkwinsize
如果设置的话，bash 在每条命令执行后检测窗口大小，如果 需要的话就更新 LINES 和 COLUMNS 的值。

cmdhist 如果设置的话， bash 试着将一个多行命令的所有行放到同一个历史条目中。这样使得多行命令可以容易地重新修改。

dotglob 如果设置的话， bash 会把以 ‘.’ 开始的文件名包含在路径 名扩展的结果中。

execfail
如 果设置的话，非交互的 shell 如果不能执行作为参数提供给内建命令 exec 的文件时将不会退出。交互的 shell 在 exec 失败时不会退出。

expand_aliases
如 果设置的话，别名被扩展，就像上面 ALIASES 中讲到的一样。这个选项在交互 shell 中是默认启用的。

extglob 如果设置的话，将允许上面 Pathname Expansion 中提到的扩展模式匹配特性。

histappend
如 果设置的话，在 shell 退出时，历史列表将追加到以 HIST-FILE 的值为名的文件之后，而不是覆盖文件。

histreedit
如果设置的话，并且正在使用 readline，用户可以重新修改 失败的历史替换。

histverify
如 果设置的话，并且正在使用 readline，历史替换的结果不会

立即传给 shell 解释器。结果行被加载到 readline 编辑缓 冲区，允许进行进一步的修改。

hostcomplete

如 果设置的话，并且正在使用 readline， bash 将试着对正在进行补全的包含 的词进行主机名补全 (参见上面的 READLINE 中的 Completing 段落)。这是默认允许的。

huponexit

如 果设置的话，在交互的登录 shell 退出时 bash 将向所有作业发出 SIGHUP 信号。

interactive_comments

如果设置的话，将允许在交互 shell 中遇到以 # 开头的词时忽略 这 个词和一行中所有剩余的字符 (参见上面的 COMMENTS 注释)。这个选项是默认允许的。

lithist 如果设置的话，并且允许了 cmdhist 选项，多行的命令在保 存到历史中时将包含新行符，而不是在可能的地方使用分号。

login_shell

如果 shell 作为登录 shell 启动，将设置这个选项 (参见上面的 启动(INVOCATION))。这个值不可修改。

mailwarn

如果设置的话，并且 bash 正在检测上次检测以来被存取过的邮件 ，将显示 ‘‘The mail in mailfile has been read’’ (mail-file 中的邮件已被读取)。

no_empty_cmd_completion

如果设置的话，并且正在使用 readline，试图在空行上执行 补全时， bash 不会搜索 PATH 来查找可能的补全。

nocaseglob

如果设置的话， bash 进行路径扩展时使用大小写不敏感方式匹配文件名(参见上面的 Pathname Expansion 路径扩展)。

nullglob

如果设置的话， bash 将允许不匹配任何文件的模式扩展为空字符 串而不是它们自身(参见上面的 Pathname Expansion 路径扩展)。

progcomp

如果设置的话，将启用可编程补全功能 (参 见 上 面 的 Programmable Completion)。这个选项是默认启用的。

promptvars

如 果设置的话，提示字符串要经过上面 PROMPTING 中描述的扩展，然后还要经过变量和参数扩展。这个选项是默认启用的。

restricted_shell

shell 设置这个选项，如果它是以受限模式启用的 (参见下面的

受 限的 shell(RESTRICTED SHELL) 章节)。这个值不能修改。在执行启动文件时，它不会被重置，使得启动文件可以得知 shell 是否是受限的。

shift_verbose
如 果设置的话，内建命令 shift 在偏移量超过位置参数的个数时打印一条错误消息。

sourcepath
如果设置的话，内建命令 source (.) 使用 PATH 中的值来查找包含作为参数给出的文件。这个选项默认是启用的。

xpg_echo
如果设置的话，内建命令 echo 默认扩展反斜杠转义序列。

suspend [-f]
挂起 shell 的执行，直到收到一个 SIGCONT 信号。选项 -f 表示如果这是一个登录 shell，那么不要提示，直接挂起。返回值是 0，除非 shell 是登录 shell 并且没有指定 -f，或者没有启用作业控制。

test expr
[expr]
返 回状态值 0 或 1，根据条件表达式 expr 的求值而定。每个操作符和操作数都必须是一个单独的参数。表达式使用上面 条件表达式 (CONDITIONAL EXPRESSIONS) 中的操作构造。

表达式可以用下列操作符结合，以优先级的降序列出。
! expr 值为真，如果 expr 为假。
(expr)
返回 expr 的值。括号可以用来超越操作符的一般优先级。
expr1 -a expr2
值为真，如果 expr1 和 expr2 都为真。
expr1 -o expr2
值为真，如果 expr1 或 expr2 为真。

test 和 [使用基于参数个数的一系列规则，对条件表达式进行求值。

0 arguments
表达式为假。
1 argument
表达式为真，当且仅当参数非空。
2 arguments
如 果第一个参数是 !，表达式为真，当且仅当第二个参数为空。如果第一个参数是上面 条件表达式 (CONDITIONAL EXPRESSIONS) 中 列出的单目条件运算符之一，表达式为真，当且仅当单目测试

为真。如果第一个参数不是合法的单目条件运算符，表达式为 假
。

3 arguments

如果第二个参数是上面 条件表达式 (CONDITIONAL EXPRESSIONS) 中列出的二进制条件操作符之一，表达式的结果是使用第一和 第三 个参数作为操作数的二进制测试的结果。如果第一个参数是 ! ，表达式值是使用第二和第三个参数进行双参数测试的结果取 反 。 如果第一个参数是 (，第三个参数是)，结果是对第二个参数 进行单参数测试的结果。否则，表达式为假。这种情况下 -a 和 -o 操作符被认为二进制操作符。

4 arguments

如 果第一个参数是 !，结果是由剩余参数组成的三参数表达式结 果取反。否则，表达式被根据上面列出的优先级规则解释并执 行
。

5 或更多 arguments

表达式被根据上面列出的优先级规则解释并执行。

times 对 shell 以及 shell 运行的进程，打印累计的用户和系统时间。返回状 态是 0。

trap [-lp] [arg] [sigspec ...]

当 shell 收到信号 sigspec 时， 命令 arg 将被读取并执行。如果没 有给 出 arg 或者给出的是 -, 所有指定的信号被设置为它们的初始值 (进入 shell 时它们的值)。如果 arg 是空字符串， sigspec 指定的信号被 shell 和它启动的命令忽略。如果 arg 不存在，并且给出了 -p 那么与每个 sigspec 相关联的陷阱命令将被显示出来。如果没有给出任何参 数 ， 或只给出了 -p， trap 将打印出与每个信号编号相关的命令列表。每个 sigspec 可以是 <signal.h> 定义的信号名，或是一个信号编号。 如果 sigspec 是 EXIT (0)，命令 arg 将在 shell 退出时执行。如果 sigspec 是 DEBUG, 命令 arg 将在每个简单命令 (simple command， 参见 上面的 SHELL GRAMMAR) 之后执行。如果 sigspec 是 ERR, 命令 arg 将在任何命令以非零值退出时执行。如果失败的命令是 until 或 while 循环的一部分， if 语句的一部分， && 或 | | 序列的一部分，或者命令的返回值是通过 ! 转化而来， ERR 陷阱将不会执行。选项 -l 使 得 shell 打印信号名和对应编号的列表。 shell 忽略的信号不能被捕捉或重置。在子进程中，被捕捉的信号在进程创建时被重置为初始值。返回值为假，如果 sigspec 非法；否则 trap 返回真。

type [-aftpP] name [name ...]

没有选项时，指示每个 name 将如何被解释，如果用作一个命令名。如果

使用了 -t 选项， type 打印一个字符串，内容是如下之 一： alias, keyword, function, builtin, 或 file ，如果 name 分别是一个别名 ，shell 保留字，函数，内建命令或磁盘文件。如果没有找到 name， 那么不会打印任何东西，返回退出状态假。如果使用了 -p 选项， type 返回如果 name 作为命令名，将被执行的磁盘文件名；或者返回空， 如 果 ‘‘type -t name’’ 不会返回 file. 选项 -P 选项强制对每个 name 搜索 PATH，即使 ‘‘type -t name’’ 不会返回 file. 如果命令在散列 中 ， -p 和 -P 将打印散列的值，而不是 PATH 中首先出现的那一个文件。 如果使用了 -a 选项， type 打印所有包含可执行的名称 name 的场合。 结果包括别名和函数，当且仅当没有同时使用 -p 选项。使用 -a 时不会 查找散列中的命令表。选项 -f 阻止 shell 进行查找，就像在内建命 令 command 中一样。 type 返回真，如果找到了任何参数。什么都没找到则 返回假。

ulimit [-SHacdflmnpstuv [limit]]

在支持它的系统上，对 shell 和它启动的进程，提供对可用资源的控 制 。选项 -H 和 -S 指定为所给资源设定的硬性和柔性限额。硬性限额在设 置后不能增加；柔性限额可以增加，直到与硬性限额相等。如果没有给出 -H 或 -S 选项，将同时设置硬性和柔性限额。 limit 的值可以是一个数 字，单位是指定资源的单元值，或者是特殊值 hard, soft, 或 unlim- ited 之一，意思分别是当前硬性限额，当前柔性限额和没有限额。如果 忽略了 limit，将打印出当前对资源的柔性限额值，除非给出了 -H 选项 。当指定多于一个资源时，限额名称和单位将在值之前打印出来。其他选 项按照如下意义解释：

-a 报告所有当前限额
-c core 文件的最大值
-d 进程数据段的最大值
-f shell 创建的文件的最大值
-l 内存中可以锁定的最大值
-m 常驻内存的最大值
-n 打开的文件描述符最大个数 (大多数系统不允许设置这个值)
-p 管道大小，以 512 字节的块为单位 (这个值可能不能设置)
-s 栈的最大值
-t cpu 时间总数的最大值，以秒计
-u 用户可以运行的最大进程数
-v shell 可用的虚拟内存总量的最大值

如果给出了 limit，它将是指定资源的新限额 (选项 -a 只显示它们) 。 如果没有给出选项，则假设有 -f。值的递增间隔是 1024 字节，除了 -t 单位是 秒， -p 单位是 512 字节的块个数， -n 和 -u 是不可调节的值

。返回 0，除非给出了非法的选项或参数，或者在设置新的限额时发生了错误。

umask [-p] [-S] [mode]

用户创建文件的掩码被设置为 mode. 如果 mode 以数字开始，它被解释为一个八进制数；否则被解释为类似于 chmod(1) 接受的符号形式的模式掩码。如果忽略了 mode，将打印当前掩码值。选项 -S 使得掩码以符 号形 式 打 印 ；默认输出是八进制数。如果给出了 -p 选项，并且忽略了 mode，输出将是一种可以重用为输入的形式。返回值是 0，如果成功改变了模式，或者没有给出 mode。其他情况返回假。

unalias [-a] [name ...]

从 已定义的别名列表中删除 name。如果给出了 -a 将删除所有别名定义。返回值是真，除非给出的 name 不是已定义的别名。

unset [-fv] [name ...]

将每个 name 对应的变量或函数删除。如果没有给出选项，或者给 出 了 -v 选项， name 仅包括 shell 变量。只读的变量不能被取消定义。如果给出了 -f 选项， name 仅包括 shell 函数，函数的定义将被删除。 每个被取消定义的变量或函数都被从后续命令的环境中删除。如果 RANDOM, SECONDS, LINENO, HISTCMD, FUNCNAME, GROUPS, 或者 DIRSTACK 中的任何一个被取消定义，它们将丧失特殊的属性，即使它们后来被重新定义。退出状态是真，除非 name 不存在或是只读的。

wait [n]

等待指定的进程，返回它的终止状态。 n 可以是进程 ID 或一个作业 号；如果给出的是作业号，将等待作业的管道中所有进程。如果没有给出 n，将等待所有当前处于激活状态的子进程，返回状态是 0。如果 n 指 定了不存在的进程或作业，返回状态是 127。否则，返回状态是所等待的最后一个进程或作业的退出状态。

受限的 shell(RESTRICTED SHELL)

如果 bash 以 rbash 名称启动，或者启动时使用了 -r 选项，那么它成为受限的 shell。受限的 shell 一般用来建立一个比标准的 shell 受到更多控制的环境。它的行为与 bash 一致，除了下列行为是不允许的 (disallowed) 或不会运行 的 (not performed)。

- 使用 cd 来改变路径；
- 设置或取消 SHELL, PATH, ENV, 或 BASH_ENV 变量的值；

- 指定的命令名中包含 / ；
- 指定包含 / 的文件名作为传递给内建命令 . 的参数；
- 指定包含斜杠 (slash) 的文件名作为 -p 选项的参数，传递给 hash 内建命令；
- 启动时从 shell 环境中导入 (import) 函数定义；
- 启动时解释 shell 环境中 SHELLOPTS 的值；
- 使用 >, >|, <>, >&, &>, 和 >> 等重定向操作符重定向输出；
- 使用 exec 内建命令来以另一个命令替换 shell；
- 使用 enable 内建命令的 -f 和 -d 选项来增加和删除内建命令；
- 使用 enable 内建命令来允许和禁止 shell 内建命令；
- 指定 command 内建命令的 -p 选项；
- 使用 set +r 或 set +o restricted 来关闭受限模式。

这些限制在所有启动文件读取之后才会生效。

当一个 shell 脚本作为一个命令执行时 (参见上面的 命令执行(COMMAND EXECUTION) 章节)， rbash 关闭为执行脚本而孵化 (spawn) 的 shell 的所有限制。

参见("SEE ALSO")

Bash Reference Manual, Brian Fox and Chet Ramey
The Gnu Readline Library, Brian Fox and Chet Ramey
The Gnu History Library, Brian Fox and Chet Ramey
Portable Operating System Interface (POSIX) Part 2: Shell and Utilities, IEEE
sh(1), ksh(1), csh(1)
emacs(1), vi(1)
readline(3)

文件(FILES)

/bin/bash
bash 可执行文件
/etc/profile
系统范围的初始化文件，登录 shell 会执行它
~/.bash_profile
个人初始化文件，登录 shell 会执行它
~/.bashrc
个人的每个交互式 shell 启动时执行的文件
~/.bash_logout
个人的登录 shell 清理文件，当一个登录 shell 退出时会执行它
~/.inputrc
个人的 readline 初始化文件

作者(AUTHORS)

Brian Fox, Free Software Foundation
bfox@gnu.org

Chet Ramey, Case Western Reserve University
chet@ins.CWRU.Edu

报告 BUGS (BUG REPORTS)

如 果你发现一个 bash 中的 bug，你应当报告它。但是首先，你应当确定它真的是一个 bug，并且它在你使用的最新版本的 bash 中存在。

一旦你认定存在那样一个 bug，使用 bashbug 命令来提交一个错误报告。如果你有固定住址，鼓励你用邮政的方式提交一份！建议和有关 bash “哲学”（‘philosophical’) 的 “错误报告” 可以寄给 bug-bash@gnu.org 或者贴到 Usenet 新闻组 gnu.bash.bug 之上。

所有错误报告应当包括：

bash 的版本号
硬件信息和操作系统
用来编译的编译器
对 bug 行为的描述
可以激活这个 bug 的一个短小的脚本或者什么 “秘诀” (recipe)

bashbug 会自动在它提供的错误报告模板中插入前三项。

关于这份手册页的评论和错误报告请直接提交到 chet@ins.CWRU.Edu.

BUGS

它太大了，并且有点慢。

bash 和传统版本的 sh 之间有一些细微的差别，大部分是因为 POSIX 规约的要求。

别名机制在一些应用中会混淆。

Shell 内建命令和函数不可终止/重新开始。

组合的命令和使用 ‘a ; b ; c’ 形式的命令序列在进程试图暂停时不能很好处理。 当一个进程中止，shell 会立即执行序列中的下一条命令。也可以将命令的序列放在圆括号中，来强制启动子 shell，这样就可以将它们作为一个单元中止 了。

在 $(...) 命令替换中的注释不会被解释，直到执行替换的时候。这将延迟报错，直到命令开始执行之后的一段时间。

数组变量还不能导出 (export)。

[中文版维护人]

袁乙钧 <bbbush@163.com>

[中文版最新更新]

2004.03.05

《中国 linux 论坛 man 手册页翻译计划》:

http://cmpp.linuxforum.net