lifayi2008 new post c3a66a1 on Sep 13, 2016

1 contributor

460 lines (329 sloc) | 26.4 KB

Raw Blame History

| title | date | tags | categories |
| --- | --- | --- | --- |
| Bash文档6--Bash特性(译) | 2016-06-16 04:25:52 -0700 | bash | Bash |

本章描述bash的特性

调用**bash**

```
bash [long-opt] [-ir] [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] [argument …]
bash [long-opt] [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] -c string [argument …]
bash [long-opt] -s [-abefhkmnptuvxdBCDHP] [-o option] [-O shopt_option] [argument …]
```

所有内置命令set可以使用的单字符选项都可以在调用bash时使用。另外，还有几个多字符选项。这些多字符选项必须出现在单字符选项之前

**`--debugger`**
在shell开始之前执行debugger描述。打开扩展的debug模式（详细的信息可以查看shopt内置命令有关extdebug选项的描述）

**`--dump-po-strings`**
A list of all double-quoted strings preceded by `$` is printed on the standard output in the GNUgettext PO (portable object) file format. Equivalent to `-D` except for the output format.

**`--dump-strings`**
等同于`-D`

**`--help`**
在标准输出打印帮助信息，然成功退出

**`--init-file filename`**
**`--rcfile filename`**
在交互式shell中从filename中读取并执行命令，代替`~/.bashrc`文件

**`--login`**
等同于`-l`

**`--noediting`**
在交互式shell中不使用GNU Readline库读取命令行

**`--noprofile`**
当bash启动为一个login shell时不读取系统范围的startup file `/etc/profile`和个人的初始化配置文件`~/.bash_profile`、`~/.bash_login`、`~/.profile`文件

**`--norc`**
交互式shell中不读取`~/.bashrc`初始化文件。当shell使用sh调用时这是默认行为

**`--posix`**
Change the behavior of Bash where the default operation differs from the POSIX standard to match the standard. This is intended to make Bash behave as a strict superset of that standard. See Bash POSIX Mode, for a description of the Bash POSIX mode.

**`--restricted`**
使用restricted模式

**`--verbose`**
等同于`-v`选项。当shell读取时打印输入的内容

**`--version`**
打印当前bash实例的版本信息并且成功退出

也有几个单字符选项可以在shell启动时使用，这些选项不能使用set内置命令设置:

```
-c      处理完选项后读取并执行第一个非选项的参数，然后退出。所有其他的参数都作为命令的位置参数从$0开始
-i      强制shell交互式运行。交互式shell在后面章节中描述
-l      Make this shell act as if it had been directly invoked by login. When the shell is interact
-r      使shell成为restricted shell
-s      如果使用这个选项或者没有非选项的参数，则会从标准输入读取命令。使用本选项可以在交互式shell中设置位置参数
```

```
-D      在标准输出打印所有前置$的双引号引用的字符串列表。如果当前的locale不是C或者Posix则这些字符串会进行语言转换。
[-+]O [shopt_option]      shopt_option是shopt内置命令可以接收的选项之一。如果使用shopt_option，-o表示设置这个
--      表示选项的结束。任何在--之后的参数被作为文件名或者参数对待
```

登陆shell在系统进程中的名字的第一个字符是`-`，或者使用`--login`调用的shell

交互式shell表示没有任何非选项参数启动的shell，除非使用`-s`选项同时没指定`-c`选项，或者使用了`-i`选项；交互式shell的输入和输出都连接到终端

如果有非选项的参数，并且没有使用`-c`或者`-s`选项，则第一个非选项参数会被假定是包含shell命令的文件名。如果bash使用这种方式调用，则`$0`被设置为这个文件名，其他的参数则成为位置参数。bash从这个文件中读取并且执行命令，然后退出。bash的退出状态是文件中最后一个命令的退出状态，如果文件中没有任何命令则返回0

### Bash startup files

本节描述bash如何执行它的startup files。如果这些文件存在但是却不可读则bash会报告一个错误。下文中的Tilde会按照前面章节中描述的Tilde扩展规则进行扩展

交互式shell在后面章节中描述

作为一个交互式登陆**shell**调用，或者使用`--login`选项
当bash被作为一个交互式登陆shell调用或者非交互式shell调用时使用login选项，则它首先会从/etc/profile文件中读取并执行命令（如果文件存在）。当读取（执行）完这个文件后，bash会按~/.bash_profile、~~/.bash_login~~、/.profile的顺序读取并执行第一个找到的文件。--noprofile选项启动shell可以改变这种行为

当login shell退出时，如果`~/.bash_logout`文件存在，则bash会读取并执行这个文件的内容

作为一个交互式非登陆**shell**
当一个交互式非登陆shell启动时，如果`~/.bashrc`文件存在，bash会读取并执行这个文件。使用`--norc`选项可以改变这种行为。`--rcfile file`会让bash读取并执行file而不是`~/.bashrc`文件

一般在~/.bash_profile文件中会包含`if [ -f ~/.bashrc ]; then . ~/.bashrc; fi`所以登陆shell实际上也会执行~/.bashrc文件中的内容

非交互式调用
当非交互式启动bash时，比如运行一个脚本，bash会在环境中查找BASH_ENV变量，然后将变量的值作为文件名读取并执行这个文件。bash的行为就像执行下面的命令:

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

但是执行`$BASH_ENV`时不会再系统的PATH中寻找这个文件名

如果bash是非交互式但是使用`--login`选项启动时，会作为一个login shell对待，会读取并执行相应的startup files

使用**sh**名调用
如果使用sh名字来调用，则bash会尽可能模仿历史版本的sh的启动行为，同时也会遵循Posix标准

当作为一个交互式登陆shell调用或者非交互式shell调用时使用login选项，bash会首先按`/etc/profile`、`~/.profile`的顺序读取并执行这两个文件。`--noprofile`选项会改变这种行为。当以非登陆交互式shell调用时，bash会查找ENV变量，以变量值作为文件名读取并执行这个文件。除了上面提到的文件名或者变量外，bash不会读取其他的任何文件和变量，所以`--norc`选项不会有任何作用。如果以非交互式非登陆shell执行sh时，bash会读取任何文件

当使用sh调用时，bash在读取并执行完startup file后就进入Posix模式

以**POSIX**模式调用
当bash以Posix模式启动时，和在命令行选项使用`--postfix`一样，bash会按照POSIX标准的startup files执行。这种模式中shell查找ENV变量，并且将变量值作为一个文件名读取并执行这个文件，同事bash不会读取其他任何startup files

被远程的**shell**服务调用
bash会尝试识别是否在被调用的时候将标准输入连接到网络，比如被远程的shell服务执行，通常是`rshd`，或者shell服务`sshd`。如果bash确定是以这种方式执行，它会从`~/.bashrc`文件（如果文件存在并且可读）中读取并执行命令。如果使用sh命令调用则不会执行上述操作。`--norc`选项也会阻止这种行为，`--rcfile`则可以要读取的文件的位置，但是执行远程调用的应用`rshd sshd`通常不会提供给你设置参数的机会

使用不同于真实**UID/GID**的有效用户**ID**调用
如果调用bash的有效用户ID不同于当前真实用户ID，并且未提供`-p`选项，则不会读取任何启动文件，也不会从环境中继承shell函数，如果环境中有`SHELLOPTS` `BASHOPTS` `CDPATH` `GLOBIGNORE`变量也被忽略，并且有效用户ID会被设置为真实用户ID；如果使用`-p`选项，则有效用户ID不会被设置，其他的同上

## 交互式**shell**

### 什么是交互式**shell**

交互式shell是指调用bash时没有非选项的参数，或者指定`-s`选项同时有没有指定`-c`选项，并且bash的输入输出和错误输出都连接到一个终端，或者启动时使用`-i`选项

交互式shell通常从终端读取数据，并且向终端写入数据

当启动交互式shell时`-s`选项可以用来设置位置参数

### 如何判断一个交互式**shell**

要在一个startup file中判断当前启动的shell是否是交互式shell，可以测试`-`特殊变量的值。如果值包含`i字符则是交互式shell。比如：

```
case "$-" in
*i*)    echo This shell is interactive ;;
*)  echo This shell is not interactive ;;
esac
```

另外，在startup file中可以检查变量`PS1`；在非交互式shell中`PS1`未设置，交互式shell中则设置这个变量

```
if [ -z "$PS1" ]; then
        echo This shell is not interactive
else
        echo This shell is interactive
fi
```

交互式**shell**的行为

当shell以交互式运行时会有如下行为：

1. 按照前面章节描述的规则读取并执行startup files
2. 默认启用Job Control。如果Job Control生效，则bash会忽略键盘产生的作业控制信号`SIGTTIN`，`SIGTTOU`和`SIGTSTP`
3. 在读取第一个行命令之前bash扩展并打印`PS1`，如果命令有多行则会在接下来的第二行之前打印`PS2`
4. 在打印首要提示符`PS1`之前会将`PROMTP_COMMAND`变量对应的值作为命令执行 5.从用户终端读取命令时使用Readline特性
5. bash会根据`ignoreeof`选项的值来决定在读取命令时在标准输入接收到EOF符号的行为，默认是退出
6. 默认启用命令历史和history expansion。当一个启用history功能的shell退出时Bash会将命令保存到`$HISTFILE`变量指定的文件名中
7. 默认进行别名扩展
8. 未设置任何Trap的情况下，bash忽略`SIGTERM`信号
9. 未设置温和Trap的情况下，bash会捕捉`SIGINT`信号并且执行相应的handler。`SIGINT`会打断某些内部命令的执行
10. 如果启用huponexit选项，交互式登陆shell会在退出时发送一个`SIGHUP`信号给所有的作业

11. Bash的`-n`选项被忽略，并且`set -n`命令也无效
12. Bash会依照`MAIL`、`MAILPATH`和`MAILCHECK`变量值的设置周期性检查邮件的到来
13. 如果使用`set -u`命令，则对未分配值的变量的引用产生的错误不会导致shell退出
14. 如果变量替换表达式`${var:?word}`中的var为空或者未设置而导致的变量扩展错误不会使shell退出
15. Shell内置命令造成的重定向错误不会导致shell退出
16. 在Posix模式中，特定内置命令返回的错误状态也不会导致shell退出
17. 失败的`exec`命令会导致shell退出
18. 语法错误也不会导致shell退出
19. 默认启用对`cd`内置命令目录参数的简单拼写检查（详细信息可以查看`shopt`命令的`cdspell`选项）
20. Bash会检查`TMOUT`变量的值，并且在打印`PS1`之后TMOUT秒内没有读取到命令时退出

**Bash**条件表达式

条件表达式一般由`[[`组合命令结构或者`[ test`内置命令来使用

表达式可以是一元的或者二元的。一元的表达式通常用来检查一个文件的状态。也有字符串操作和数值对比操作符。如果file参数指代的操作对象是`/dev/fd/N`的形式，则文件描述符`N`被检查。如果file参数指代的对象是`/dev/stdin`，`/dev/stdout`或者`/dev/stderr`则文件描述符0，1，2被检查

在`[[`结构中`< >`操作符按本地语言字符顺序排序。而`test`命令则按`ascii`的顺序排序

除非特别指出，下面这些`file`如果是一个符号连接的话，都需要进行解引用（检查符号连接指代的对象，而非符号连接本身）：

**`-a file`**
**`-e file`**
如果文件存在则返回真

`-s file`
文件存在并且size大于零则返回真

`-r file`
`-w file`
`-x file`
文件存在并且可读/可写/可执行则返回真

`-c file`
`-b file`
文件存在并且是一个字符/块特殊设备则返回真

`-f file`
`-d file`
文件存在并且是一个常规文件/目录则返回真

`-h file`
`-L file`
文件存在并且是一个符号连接则返回真

`-p file`
`-S file`
文件存在并且是一个命名管道/socket则返回真

`-t fd`
`fd`指代的文件描述符已经打开并且指向一个终端则返回真

`-g file`
`-u file`

`-k file`
文件存在并且set user id/set group id/sticky被置位则返回真

`-O file`
`-G file`
文件存在并且为当前有效用户/用户组所有则返回真

`-N file`
文件存在并且自上次读之后已经被修改则返回真

`file1 -ef file2`
两个文件指代相同的设备和i-node号（互为硬链接）

`file1 -nt file2`
`file1 -ot file2`
两个文件都存在并且file1比file2新/旧（mtime），或者文件1/2存在但是文件2/1不存在则返回真

`-o optname`
如果optname选项启用则返回真

`-v varname`
如果shell变量varname已经被设置（可以为空值）则返回真

`-R varname`
如果shell变量varname已经被设置并且是一个name reference则返回真

`-z string`
如果字符串string长度为0则返回真

`-n string`
`string`

如果字符串string长度不为0则返回真

**`string1 == string2`**
**`string1 = string2`**
如果字符串相等则返回真，当在`[[`结构中使用时，则表示模式匹配

**`string1 != string2`**
两个字符串不等时返回真

**`arg1 OP arg2`**
OP可以是`-eq`, `-ne`, `-lt`, `-le`, `-gt`, or `-ge`.中的一个，用于操作数值操作数，两个操作数都可以是负值

**`string1 < string2`**
**`string1 > string2`**
True if string1 sorts before/after string2 lexicographically.

**shell算术运算**

使用shell扩展或者`(())`表达式或者内置命令`let`或者内置命令`declare -i`选项声明的变量都可以进行算术运算

> 注意：`(())`表达式和`let`命令也做算术运算但是结果是一个返回值（0或者1）而不是算术表达式计算的结果

运算使用固定宽度的整型数并且不进行溢出检查，但是如果有除0的操作的话会被捕获并且报错。操作符的优先级，结合性和C语言中的一样。下面的一系列操作符会按优先级分为几个等级。下面列出的操作符优先级依次降低

**`id++`** **`id--`**
变量后缀自增和自减

**`++id`** **`--id`**
变量前缀自增和自减

`-` `+`
一元的加和减

`!` `~`
逻辑取反和按位取反

`\*\*`
求幂

`\*` `/` `%`
乘、除、取模

`+` `-`
加、减

`<<` `>>`
按位左移和按位右移

`<=` `>=` `<` `>`
比较

`==` `!=`
等和不等

`&`
按位与

`^`
按位取反

`|`
按位或

`&&`
逻辑与

`||`
逻辑或

`expr ? expr : expr`
三元运算符，条件表达式

`= *= /= %= += -= <<= >>= &= ^= |=`
分配值

`expr1 , expr2`
逗号运算符

Shell variables are allowed as operands; parameter expansion is performed before the expression is evaluated. Within an expression, shell variables may also be referenced by name without using the parameter expansion syntax. A shell variable that is null or unset evaluates to 0 when referenced by name without using the parameter expansion syntax. The value of a variable is evaluated as an arithmetic expression when it is referenced, or when a variable which has been given the integer attribute using ‘declare -i’ is assigned a value. A null value evaluates to 0. A shell variable need not have its integer attribute turned on to be used in an expression.

以0开始的常量被当作是8进制的数值，以`0x`或者`0X`开始则被当作十六进制。否则数值常量使用`[base#]n`的形式，`base`是一个在2到64之间的数字表示进制，`n`是基于这个进制的数值。如果`base#`为空，则表示十进制。如果`base`大于10则使用`数字` `小写字母` `大写字母` `@` `_`字符（并且按这样的顺序）来表示每一个位置上的数字值。如果`base`小于36也可以在数字之后直接使用大写字母来表示10到35之间的数字值

操作符按照优先级的顺序进行计算，可以使用小括号来改变这种优先级

别名

Aliases可以让我们将简单命令中的第一个word进行字符串替换。shell维护一个包含所有aliases的列表，可以使用alias或者unalias内置命令来设置或者取消别名

简单命令的第一个word，如果未被引用，则会检查它是否是一个别名。如果是的话则会被别名的内容（别名值）代替。`/` `$` `` ` `` `=`和任何元字符以及引用字符都不能出现在别名名字中。而别名内容可以包括任何可以在bash中输入的字符，包括shell元字符而别名内容可以包括任何可以在bash中输入的字符，包括shell元字符。别名内容的第一个word会被再次检查是否是别名，但是如果第一个word等同于别名名称则不会再被替换。比如，`ls`命令是`ls -F`的别名，则不会进行递归的别名替换。如果别名值的最后一个字符是空白，则后面紧接着的word也会做别名检查

和csh一样，在别名值中不能使用参数。如果需要一个参数可以使用函数代替别名

如果shell是非交互式，则默认不会进行别名替换，除非使用`shopt`命令的`expand_aliases`选项

关于别名的定义和使用可能有些令人混淆的地方。Bash总是读取一个完整的行之后才会执行这一行的命令。而别名需要在命令被读取时进行替换，而不是执行时。所以，如果别名定义和其他命令在同一行时，直到下一次执行命令时别名才会生效。而同一行中别名定义之后的命令不能使用这个别名。这样的问题在函数定义过程中也存在，如果在函数定义中有别名定义，则只有在函数执行时这个别名才被设置。所以为了安全期间，别名定义通常放置在单独的一行，并且不在组合命令中使用别名

综合各种原因，应该优先使用函数而不是别名

数组

bash提供了一维的索引和关联数组变量。任何变量都可以作为一个数组变量使用；declare内置命令可以显式的声明一个数组。数组的元素个数和每个元素的大小都没有限制，成员的索引也不必是连续的数字或者字符。索引数组使用

整数数字（包括算术表达式）下标来引用，从0开始；关联数组的下标可以是任意字符串。除非特别指出，索引数组的下标都不能是负数

直接使用下面形式的语法就可以创建一个索引数组（空的value值也是一个合法值）（如果首先使用`declare -A`声明，然后subscript是一个字符串的话，则创建一个关联数组）：

```
$ name[subscript]=value
```

subscript被认为是一个算术表达式，并且计算结果必须是一个数字。也可以使用下面的语法显式声明一个索引：

```
declare -a name
declare -a name[subscript]
```

第二种方法也是正确的，但subscript在这里没有意义，所以被忽略

使用下面的语法可以声明一个关联数组：

```
$ declare -A name
```

可以使用内置命令declare和readonly来给数组变量指定一个属性。数组的属性被应用到每一个数组元素 也可以使用下面的组合命令语法来给数组赋值（这样可以创建索引数组，关联数组）：

```
$ name=(value1value2 … )
```

所有的value都是 `[subscript]=string` 的形式。索引数组赋值只需要string部分，*但是如果要给非连续的特定subscript赋值或者要创建一个关联数组则需要提供`[subscript]`，相应的string被赋给这个subscript*，否则string被分配给上一个`subscript+1`的下标

如果使用上面的这种赋值方式，在一个数组中可以出现数字和字符串的下标，但这种数组在遍历时会有意想不到的结果

declare内置命令也可以使用这种语法。单个元素可以使用 `name[subscript]=value` 的方式赋值

给一个索引数组赋值时，如果使用一个负值的下标 `-n`，则表示相对于这个数组最大元素的元素，所以负值表示从后面数第n个元素，-1表示最后一个元素

数组中的元素可以使用 `${name[subscript]}` 的形式引用。大括号是必须的否则可能会被shell解释为路径名扩展操作符。如果subscript是 `@` 或者 `*` 则参数扩展的结果是数组中所有元素。两种表示方法的区别只在于在双引号中扩展时：`${name[*]}` 扩展为一个单个的word，数组中所有元素使用 `IFS` 变量的第一个字符分割；而 `${name[@]}` 将数组中的每个元素都扩展为单个的word。当数组name中没有元素时，`${name[@]}` 扩展为空。如果上面的双引号扩展发生在一个word中时，数组的第一个元素会和数组前面的字符连接，而最后一个元素会和数组后面的字符连接。这和特殊参数 `@` 和 `*` 扩展的形式类似。`${\#name[subscript]}` 扩展为 `${name[subscript]}` 数组元素的长度；如果下标是 `@` 或者 `*` 的话则扩展为数组name中元素的个数。如果使用 `${name}` 的形式引用数组则表示数组的第一个元素（下标为0）

`${!name[@]}` 和 `${!name[*]}` 的形式用来获取数组所有元素的下标，两种方式在双引号中的不同扩展结果和上面的描述相似

`unset` 内置命令可以用来销毁一个数组变量。`unset name[subscripdangt]` 则用来销毁一个数组元素，这里也可以使用负值的下标。需要注意的是这种形式有可能被shell当作是路径扩展。当下标是 `*` 或者 `@` 时也表示销毁整个数组

declare local和readonly内置命令都可以接受 `-a` 参数或者 `-A` 参数来声明一个索引数组或者关联数组；如果两个参数都给出的话 `-A` 优先。read内置命令也接受一个 `-a` 选项表示将从标准输入获取的一系列word赋给一个数组变量。set和declare内置命令以可以重新作为输入的方式显示数组变量的元素

目录栈

目录栈是最近访问的目录的一个列表。pushd内置命令会将当前目录切换到由参数指定的目录并且将这个目录加入到目录栈；而popd命令则从目录栈移除指定目录然后切换到参数指定的目录。内置命令dirs可以显示当前的目录栈

目录栈的内容还可以使用DIRSTACK内置变量来显示

目录栈相关的内置命令

```
$ dirs [-clpv] [+N | -N]
显示当前系统记住的目录列表。可以使用pushd命令将目录添加到这个列表；popd命令从列表中移除目录

-c      删除目录栈中的所有目录
-l      使用绝对路径打印目录列表，默认情况下会使用`~`代替根目录
-p      打印的每个目录占一行
-v      打印的每个目录占一行，并且在前面显示行号
+N      打印第N个目录名，计数从左边开始第一个是0
-N      打印从右边数的第N个目录
```

```
$ popd [-n] [+N | -N]
从目录栈中移除顶端的一个目录，并且切换到新的顶端目录；顶端目录是dirs命令不带参数时最左边的一个目录；popd等同于popd -

-n      只从目录栈中移除最顶端的目录，而不会进行目录切换
+N      移除第N个目录名，计数从左边开始第一个是0（是否会切换目录要看最顶端目录是否变动）
-N      移除从右边数的第N个目录（是否会切换目录要看最顶端目录是否变动）
```

```
$ pushd [-n] [+N | -N | dir]
切换到dir指定的目录，并且添加到目录栈的顶端。如果没有指定目录则pushd交换最顶端的两个目录（而且会切换目录）

-n      只增加dir到目录栈的顶端，而不会进行目录切换
+N      将第N个目录（按照dirs的显示从左到右第一个为0）交换到目录栈的顶端并且，切换到这个目录
```

```
-N      将第N个目录（按照dirs的显示从右到左第一个为0）交换到目录栈的顶端并且，切换到这个目录
dir      指定要切换到的目录
```

控制命令提示符

Bash每次打印首要命令提示符时会检查`PROMPT_COMMAND`变量的值。如果这个变量被设置并且非空，则shell在打印提示符之前将这个变量的值作为命令执行

下面表示可以在命令提示符变量`PS1`到`PS4`中可以使用的特殊字符及代表的意义：

**`\a`** 响铃字符 **`\d`** 时间，以“星期 月 日”的格式 **`\D{format}`** 将format传递个strftime(3)系统函数，并将结果值插入PROMPT；如果format为空则使用地区相关的当前时间，注意大括号是必须的 **`\e`** 转义字符 **`\h`** 主机名，到第一个`.`的内容；`PS1`默认值包括这个特殊字符 **`\H`** 完整主机名包括域名 **`\j`** shell管理的作业数量 **`\l`** shell终端设备名的`basename` **`\n`** 换行 **`\r`** 回车 **`\s`** shell名，`$0`连接线后面的部分 **`\t`** 24小时，`HH:MM:SS`格式的时间 **`\T`** 12小时，`HH:MM:SS`格式的时间 **`\@`** 12小时，`am/pm`格式的时间 **`\A`** 24小时，`HH:MM`格式的时间 **`\u`** 当前登录的用户名；`PS1`默认包含这个特殊字符 **`\v`** Bash版本 **`\V`**
**`\w`** 当前工作目录，家目录缩写为一个`~` **`\W`** `$PWD`变量的`basename`，家目录缩写为一个`~` **`\!`** 命令的`history number` **`\#`** 命令的`command number` **`\$`** 如果当前有效id是0则显示为`#`，否则显示为`$`\`` ** \nnn ** \ ** 反斜线 **[ ** 开始一个非打印字符序列，This could be used to embed a terminal control sequence into the prompt. **]`** 结束一个非打印字符序列

command number和history number通常是不同的：command表示当前会话中执行的command的号码，history还包括其他会话执行的已经存储到history文件中的命令

当特殊字符替换完毕后，还要经过`parameter expansion`，`command substitution`，`arithmetic expansion`，和`quote removal`，并且要受到`promptvars` shell选项值的限制