# 简介

Linux将命令合并处理过程转为文件执行，不使用命令行，该文件内容就是shell脚本，它是非交互的。执行非交互式shell时，在/etc/bashrc和.bashrc文件中检索BASH\_ENV环境变量。

制作shell脚本第一行:#!/bin/bash称为“魔术数字”，向内核告知需要在脚本解析行的程序。

read命令：用于从终端或文件中读取输入的字符串，read命令会读取1行，直到发现newline为止，行尾的newline读取为null。

|  |  |
| --- | --- |
| read命令选项 | 含义 |
| read var | 从标准输入读取1行，并赋值给var |
| read var1 var2 ... | 从标准输入读取1行，以空格或newline为准，将第一个次赋给var1，以此类推。 |
| read –a array | 读取命名为array的数组 |
| read –p alert | 显示alert，将输入的内容保存到REPLY变量中 |
| read –r line | 允许输入包含反斜杠的内容 |

# 变量

变量类型分为局部变量和环境变量。

变量名必须以字母或“\_”字符开始，起始字符后可输入字母，数字，“\_”等。给变量赋值时“=”前后不可以出现空格。

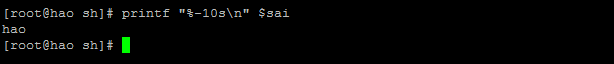
创建变量：变量名=值

显示变量值：echo内置命令通过标准输出显示参数。

|  |  |
| --- | --- |
| echo选项值 | 含义 |
| -e | 翻译字符串中有下一个反斜杠转义的字符 |
| -E | 在默认需要解析的位置也不能解析转义符 |
| -n | 末尾不显示换行符 |



printf命令：使用printf命令可以指定输出格式，使用方法和C语言相同。



删除变量：局部变量和环境变量若不是只读状态，可以使用unset命令清除设置。

declare，typeset内置命令创建变量，二者基本相同,通常使用declare创建变量。

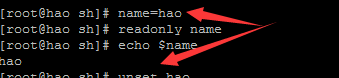
|  |  |
| --- | --- |
| declare选项值 | 含义 |
| -a | 将变量作为数组处理 |
| -f | 显示函数名和定义目录 |
| -F | 只显示函数名目录 |
| -i | 生成整数型类型变量 |
| -r | 生成只读变量 |
| -x | 向子shell传递变量名 |

只使用declare –i命令可以显示声明为整数型的变量目录和值。

echo $$：显示当前shell的进程号。

创建只读变量：readonly 变量名

declare –r 变量名=值



## 环境变量

环境变量又称全局变量，环境变量一般使用大写字母定义，可以使用export内置命令将局部变量转换从横全局变量。

父shell创建的环境变量可以传递个子shell，而子shell创建的环境变量不可以在父shell中使用。

创建环境变量：1.declare -x 变量名=值

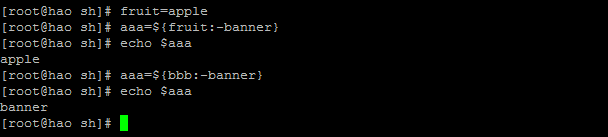
2.使用export命令。

|  |  |
| --- | --- |
| export选项值 | 含义 |
| -- |  |
| -f | 识别为函数而非变量 |
| -n | 因全局变量转为局部变量，故变量不能传递个子进程 |
| -p | 显示所有全局变量 |

## 变量扩展修饰符

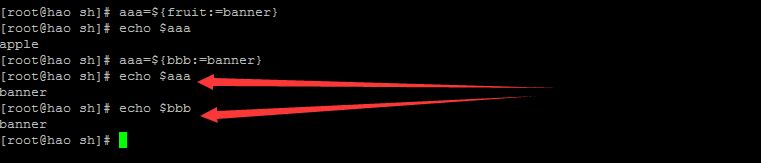
1. var=${variable:-word}

若variable存在并非空（null），则var=variable，否则，var=word

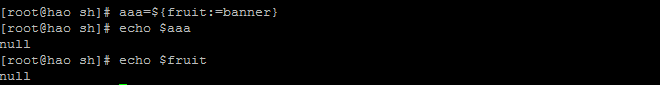


1. var=${variable:=word}

若variable非空，则var=variable，否则，var=word，并且创建变量variable。

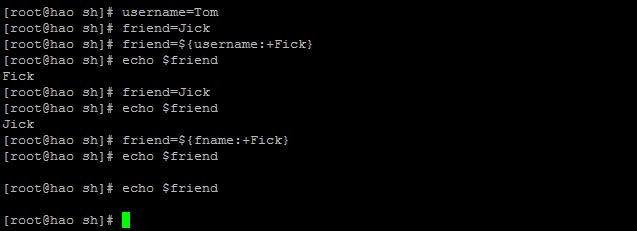


如果variable等于空（null），则var等于空（null）。



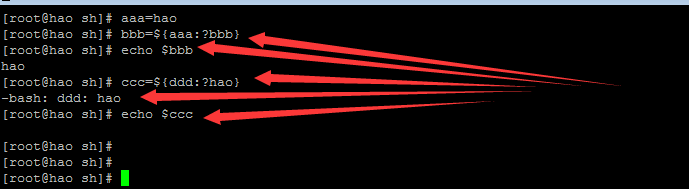
1. var=${variable:+word}

若variable变量非空且有值，则var=word，否则，如果var存在，就删除。



1. var={variable:?word}

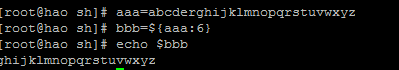
若variable存在且不为空（null），则var=variable，否则，word在variable后显示为标准错误，若是脚本则终止，如果word没有设置，则shell就会向标准错误发送默认消息。





1. var=${variable:offset}

字符串从0开始偏移，由偏移指定并以偏移开始的值替换变量值。



1. var=${variable:offset:length}

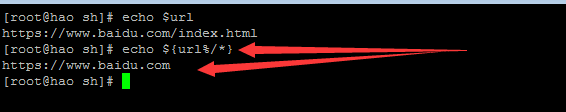
字符串从0开始偏移，由从起始偏移开始拥有指定字符长度的偏移的值替换变量值。



## 截取字符串的变量扩展名

1. ${variable%pattern}

从变量值后端开始，截取与%后模式匹配的最短值



1. ${variable%%pattern}

从变量值后端开始，截取与%后模式匹配的最长值

1. ${variable#pattern}

从变量值前端开始，截取与#后模式匹配的最短值

1. ${variable##pattern}

从变量值前端开始，截取与#后模式匹配的最长值

1. ${#variable}

替换分配给变量的值的字符数。若\*或@，那么个数等于位置参数的个数

## 特殊变量

Shell包含具有1个字符的特殊变量，$符号用于访问变量保存的值。

位置参数：

|  |  |
| --- | --- |
| 位置参数表达式 | 含义 |
| $0 | 0号参数，表示当前shell脚本名 |
| $1-$9 | 表示1-9号位置参数 |
| ${10} | 表示第10个位置参数值。两位数参数位置上使用({}) |
| $# | 表示参数的个数，常用于循环 |
| $\* | 评估所有位置参数值 |
| $@ | 除使用双引号(“”)的情况外，等同于$\* |
| “$\*” | 以“$1,$2,$3”格式评估 |
| “$@” | 以”$1” “$2” “$3”格式评估 |

$?： 表示最后一个命令退出的状态码（返回值）,范围是0~127，如果状态码时0，则表示命令正常执行。

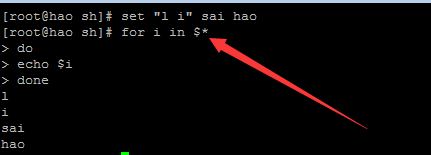
$$ ：表示当前进程编号

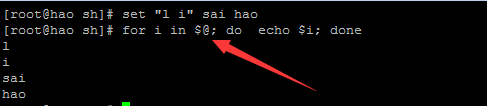
$!：表示后台正在执行的最后一个命令的PID值。

$\*和$@ 都表示参数列表

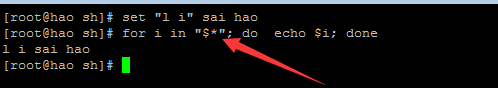
**$\*与$@区别：**

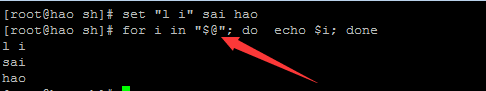
1. 不被双引号包含时：$\*和$@都以$! $2 ... $n的形式组成参数列表





1. 当它们被双引号包含时：“$\*”会将所有参数作为一个整体，以"$1$2$3...$n"的形式组成一个整体"$@"会将各个参数分开，以"$1" "$2" ... "$n"的形式组成一个参数列表





## 引用符号

引用符号在解析器中保护特殊元字符和参数扩展。

使用方式：

1.使用反斜杠（\）：将要转义的字符转化为引用符号或转义，反斜杠保护$,’，使用双引号保护反斜杠（\）。

2.使用单引号（’’）：单引号必须前后匹配，在解析器中保护所有元字符。使用双引号或反斜杠才能显示单引号。

3.使用双引号（””）：双引号必须前后匹配，允许使用命令和变量替换，保护特殊的元字符不被解析器解析。

元字符：

|  |  |
| --- | --- |
| 元字符 | 含义 |
| ; | 命令分隔符 |
| & | 后台处理 |
| () | 命令分组，创建子shell |
| {} | 命令分组，不创建子shell |
| | | 管道 |
|  |  |
|  |  |

# 运算符

**算数运算符：**

+ - \* / = == !=

**关系运算符:**

-eq(检测两个数是否相等，相等返回true)

-gt:检测左边的数是否大于右边的，如果是，则返回 true。

-ge：检测左边的数是否大等于右边的，如果是，则返回 true。

-lt：检测左边的数是否小于右边的，如果是，则返回 true。

-le：检测左边的数是否小于等于右边的，如果是，则返回 true。

-ne：检测两个数是否相等，不相等返回 true。

**布尔运算符：** !(非) -o(或) -a(与)

! 非运算， 表达式为true，则返回false，否则返回true

-o 或运算， 有一个表达式返回true，则返回true

-a 与运算， 两个表达式都为true，才返回true

**逻辑运算符：** && 和 ||

&& 相当于Java中的&&。

|| 相当于Java中的||。

**字符串运算符 ：**= != -z -n str

原生bash是不支持简单的数学运算符，可以借助其它的命令来实现，例如awk和expr。expr是一个表达式计算工具，也是最常用在表达式求值用途中的val=`expr 3 + 2` ## 用到反引号，表达式和运算符之间要有空格。

在相乘的时候需要把\*转义

**字符串运算符：**

= 检测两个字符串是否相等，相等返回true

!= 检测两个字符串是否相等，不相等返回true

-z 检测字符串长度是否为0，为0返回true

-n 检测字符串长度是否为0，不为0返回true

str 检测字符串是否为空，不为空返回true

**文件测试运算符：**用来检测文件的属性

-s file 检测文件是否为空（文件的大小是否大于0）

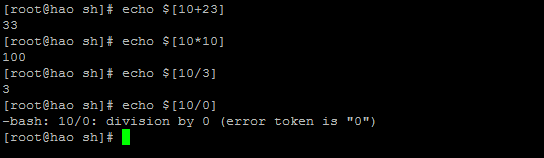
|  |  |
| --- | --- |
| -b filename | 文件存在且为块特殊文件就显示真 |
| -c filenaem | 文件存在且为字符特殊文件就显示真 |
| **-d filenaem** | 检测文件是否普通文件 |
| **-e filename** | 检测文件或目录是否存在 |
| -f filenaem | 文件存在且为一般文件就显示真 |
| -g filename | 文件存在且为set-group id就显示真 |
| -k filename | 文件设置沾滞位就显示真 |
| -L filename | 文件存在且为符号链接就显示真 |
| -p filename |  |
| -O filename | 文件存在且为所属有效用户ID就显示真 |
| **-r filename** | 文件存在且可读就显示真 |
| **-s filename** | 文件存在且大小大于0就显示真 |
| **-w filename** | 文件存在且可写就显示真 |
| **-x filename** | 文件存在且可执行就显示真 |

# 整数算数运算

Shell解析算法表达式时执行算数扩展。做除法运算时，从结果值中舍弃余数，只显示整数值。

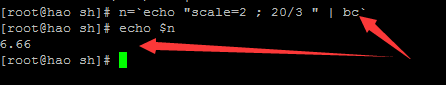
算数表达式：$[算数表达式]

$((算数表达式))



let命令：Bash的内置命令，执行整数型算数运算并测试数字表达式。

Bash只支持整数型运算，但使用bc，awk使用工具即可执行复杂的运算。





# 数组

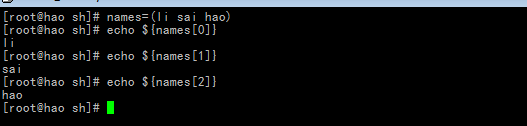
Bash Shell支持创建一维数组。数组可以使用内置函数declare –a以x[0]格式创建。

数组索引值从数值0开始，数组没有上限。

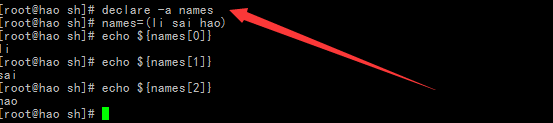
使用${数组名[index]}格式获取数组要素，使用declare命令-a ,-r选项创建只读数组。

定义数组：

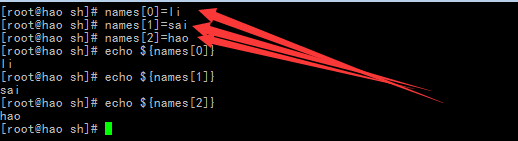
方法一：定义数组：数组名=(值1 值2 值3 ... 值n)



方法二：declare –a 变量名

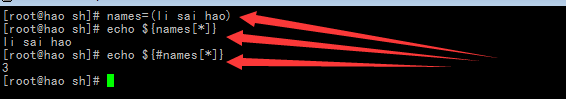


方法三：单独定义数组的各个元素



输出数组全部元素：${arr[\*]}

获取数组长度：${#arr[\*]}



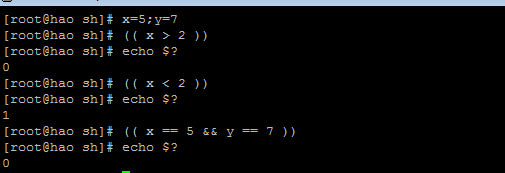
# 条件语句和分支语句

If命令是最简单的条件命令，还有if/else，if/elif/else命令。

Bash可测试2种状态，根据表达式的结果是true还是false测试命令执行成功还是失败。另一种，使用终止状态（exit status），若终止状态为0就表示成功并显示true，若显示0以外的数字就表示失败并显示false。终止状态变量“?”拥有终止状态数值。

**let命令：**

let命令等于在双括号(())使用表达式。



**test命令：**

test命令可以使用[]代替，test命令可以评估表达式。

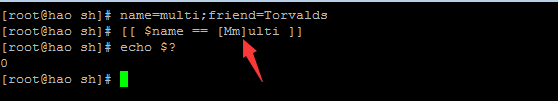
在Bash2.x版本以上双方括号（[[]]）可用来判断表达式，前方括号之后必须输入有空格。

字符串测试：

|  |  |
| --- | --- |
| 字符串测试 | 含义 |
| [ string1 = string2 ] | 测试字符串string1和字符串string2是否相等 |
| [ string1 != string2 ] | 测试字符串string1和字符串string2是否不相等 |
| [ string1 ] | 测试字符串string1是否非null |
| [ -z string] | 测试字符串的长度是否为0 |
| [ -n string ] | 测试字符串的长度是否非0 |
| [ -l string] | 字符串的长度 |

逻辑测试：

|  |  |
| --- | --- |
| 逻辑测试 | 含义 |
| [[ pattern1 && pattern2 ]] | 测试模式1和模式2是否都是true |
| [[ pattern || pattern2 ]] |  |
| [[ !pattern ]] |  |
| [string1 –a string2 ] |  |
| [string1 –o string2 ] |  |
| [ !string ] |  |



整数型测试：

|  |  |
| --- | --- |
| [ int1 –eq int2 ] |  |
| [ int1 –ne int2 ] |  |
| [ int1 –gt int2 ] |  |
| [ int1 –ge int2 ] |  |
| [int1 –lt int2 ] |  |
| [int1 –le int2 ] |  |

if命令：条件语句，执行if语句的条件返回终止状态，终止状态为0就表示命令成功执行，然后执行then之后的语句，若非0就忽略then之后的语句。

在Bash中，if条件语句有命令组合而成。

exit命令：终止脚本并返回命令行，exit命令参数可以使用0-255之内的数字，若显示exit0就正常终止程序，若参数非0就表示失败。

exit命令参数保存于shell的“？”变量中。

## 条件语句

**格式：**if 命令

then

command1

command2

fi

if.sh

|  |
| --- |
| #!/bin/bash  echo "Are you ok.(y/n)"  read var  if [ "$var" = Y -o "$var" = y ]  then  echo "you choice is y"  fi |

**格式：** if命令

then

command

else

command

fi

**格式：**if 命令

then

command

elif 命令

then

command

else

command

fi

## 分支语句

case命令：该命令可以代替if/then/elif/then/else命令，case变量值可以匹配value1，value2等值。匹配值执行可检索到的位置命令，该命令会一直执行直到发现双分号（;;）。case命令末尾使用esac，它是case的倒写。

若未匹配case变量，程序将一直执行直到发现\*)之后的;;或esac为止。case值可以使用shell通配符，也可以使用竖杠（|）

**格式：**

case 变量 in

value1)

command

;;

value2)

command

;;

\*)

Command

;;

esac

# 循环语句

## for循环语句命令

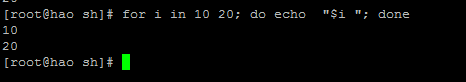
for命令后面输入用户自定义变量，in关键字和词目录。

格式：for 变量 in 词目录

do

comman

done



# 函数