# 第1章 变量

## 1.1 系统预定义变量

**1）常用系统变量**

$HOME、$PWD、$SHELL、$USER等

**2）案例实操**

（1）查看系统变量的值

[user1@hadoop101 datas]$ echo $HOME

/home/user1

（2）显示当前Shell中所有变量：set

[user1@hadoop101 datas]$ set

BASH=/bin/bash

BASH\_ALIASES=()

BASH\_ARGC=()

BASH\_ARGV=()

## 1.2 自定义变量

**1）基本语法**

（1）定义变量：变量名=变量值，注意=号前后不能有空格

（2）撤销变量：unset 变量名

（3）声明静态变量：readonly变量，注意：不能unset

**2）变量定义规则**

（1）变量名称可以由字母、数字和下划线组成，但是不能以数字开头，环境变量名建议大写。

（2）等号两侧不能有空格

（3）在bash中，变量默认类型都是字符串类型，无法直接进行数值运算。

（4）变量的值如果有空格，需要使用双引号或单引号括起来。

**3）案例实操**

（1）定义变量A

[user1@hadoop101 datas]$ A=5

[user1@hadoop101 datas]$ echo $A

5

（2）给变量A重新赋值

[user1@hadoop101 datas]$ A=8

[user1@hadoop101 datas]$ echo $A

8

（3）撤销变量A

[user1@hadoop101 datas]$ unset A

[user1@hadoop101 datas]$ echo $A

（4）声明静态的变量B=2，不能unset

[user1@hadoop101 datas]$ readonly B=2

[user1@hadoop101 datas]$ echo $B

2

[user1@hadoop101 datas]$ B=9

-bash: B: readonly variable

（5）在bash中，变量默认类型都是字符串类型，无法直接进行数值运算

[user1@hadoop102 ~]$ C=1+2

[user1@hadoop102 ~]$ echo $C

1+2

（6）变量的值如果有空格，需要使用双引号或单引号括起来

[user1@hadoop102 ~]$ D=I love banzhang

-bash: world: command not found

[user1@hadoop102 ~]$ D="I love banzhang"

[user1@hadoop102 ~]$ echo $D

I love banzhang

（7）可把变量提升为全局环境变量，可供其他Shell程序使用

export 变量名

[user1@hadoop101 datas]$ vim helloworld.sh

在helloworld.sh文件中增加echo $B

#!/bin/bash

echo "helloworld"

echo $B

[user1@hadoop101 datas]$ ./helloworld.sh

Helloworld

发现并没有打印输出变量B的值。

[user1@hadoop101 datas]$ export B

[user1@hadoop101 datas]$ ./helloworld.sh

helloworld

2

## 1.3 特殊变量

### 1.3.1 $n

**1）基本语法**

$n （功能描述：n为数字，$0代表该脚本名称，$1-$9代表第一到第九个参数，十以上的参数，十以上的参数需要用大括号包含，如${10}）

**2）案例实操**

[user1@hadoop101 datas]$ touch parameter.sh

[user1@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo '==========$n=========='

echo $0

echo $1

echo $2

[user1@hadoop101 datas]$ chmod 777 parameter.sh

[user1@hadoop101 datas]$ ./parameter.sh cls xz

==========$n==========

./parameter.sh

cls

xz

### 1.3.2 $#

**1）基本语法**

$# （功能描述：获取所有输入参数个数，常用于循环）。

**2）案例实操**

[user1@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo '==========$n=========='

echo $0

echo $1

echo $2

echo '==========$#=========='

echo $#

[user1@hadoop101 datas]$ chmod 777 parameter.sh

[user1@hadoop101 datas]$ ./parameter.sh cls xz

==========$n==========

./parameter.sh

cls

xz

==========$#==========

2

### 1.3.3 $\*、$@

**1）基本语法**

$\* （功能描述：这个变量代表命令行中所有的参数，$\*把所有的参数看成一个整体）

$@ （功能描述：这个变量也代表命令行中所有的参数，不过$@把每个参数区分对待）

**2）案例实操**

[user1@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo '==========$n=========='

echo $0

echo $1

echo $2

echo '==========$#=========='

echo $#

echo '==========$\*=========='

echo $\*

echo '==========$@=========='

echo $@

[user1@hadoop101 datas]$ ./parameter.sh a b c d e f g

==========$n==========

./parameter.sh

a

b

==========$#==========

7

==========$\*==========

a b c d e f g

==========$@==========

a b c d e f g

### 1.3.4 $？

**1）基本语法**

$？ （功能描述：最后一次执行的命令的返回状态。如果这个变量的值为0，证明上一个命令正确执行；如果这个变量的值为非0（具体是哪个数，由命令自己来决定），则证明上一个命令执行不正确了。）

**2）案例实操**

判断helloworld.sh脚本是否正确执行

[user1@hadoop101 datas]$ ./helloworld.sh

hello world

[user1@hadoop101 datas]$ echo $?

0

# 第2章 运算符

**1）基本语法**

“$((运算式))”或“$[运算式]”

**2）案例实操：**

计算（2+3）\* 4的值

[user1@hadoop101 datas]# S=$[(2+3)\*4]

[user1@hadoop101 datas]# echo $S

# 第3章 条件判断

**1）基本语法**

（1）test condition

（2）[ condition ]（注意condition前后要有空格）

注意：条件非空即为true，[ user1 ]返回true，[ ] 返回false。

**2）常用判断条件**

（1）两个整数之间比较

-eq 等于（equal） -ne 不等于（not equal）

-lt 小于（less than） -le 小于等于（less equal）

-gt 大于（greater than） -ge 大于等于（greater equal）

（2）按照文件权限进行判断

-r 有读的权限（read）

-w 有写的权限（write）

-x 有执行的权限（execute）

（3）按照文件类型进行判断

-e 文件存在（existence）

-f 文件存在并且是一个常规的文件（file）

-d 文件存在并且是一个目录（directory）

**3）案例实操**

（1）23是否大于等于22

[user1@hadoop101 datas]$ [ 23 -ge 22 ]

[user1@hadoop101 datas]$ echo $?

0

（2）helloworld.sh是否具有写权限

[user1@hadoop101 datas]$ [ -w helloworld.sh ]

[user1@hadoop101 datas]$ echo $?

0

（3）/home/user1/cls.txt目录中的文件是否存在

[user1@hadoop101 datas]$ [ -e /home/user1/cls.txt ]

[user1@hadoop101 datas]$ echo $?

1

（4）多条件判断（&& 表示前一条命令执行成功时，才执行后一条命令，|| 表示上一条命令执行失败后，才执行下一条命令）

[user1@hadoop101 ~]$ [ user1 ] && echo OK || echo notOK

OK

[user1@hadoop101 datas]$ [ ] && echo OK || echo notOK

notOK

# 第4章 流程控制

## 4.1 if判断

**1）基本语法**

（1）单分支

if [ 条件判断式 ];then

程序

fi

或者

if [ 条件判断式 ]

then

程序

fi

（2）多分支

if [ 条件判断式 ]

then

程序

elif [ 条件判断式 ]

then

程序

else

程序

fi

注意事项：

（1）[ 条件判断式 ]，中括号和条件判断式之间必须有空格

（2）if后要有空格

**2）案例实操**

输入一个数字，如果是1，则输出banzhang zhen shuai，如果是2，则输出cls zhen mei，如果是其它，什么也不输出。

[user1@hadoop101 datas]$ touch if.sh

[user1@hadoop101 datas]$ vim if.sh

#!/bin/bash

if [ $1 -eq 1 ]

then

echo "banzhang zhen shuai"

elif [ $1 -eq 2 ]

then

echo "cls zhen mei"

fi

[user1@hadoop101 datas]$ chmod 777 if.sh

[user1@hadoop101 datas]$ ./if.sh 1

banzhang zhen shuai

## 4.2 case语句

**1）基本语法**

case $变量名 in

"值1"）

如果变量的值等于值1，则执行程序1

;;

"值2"）

如果变量的值等于值2，则执行程序2

;;

…省略其他分支…

\*）

如果变量的值都不是以上的值，则执行此程序

;;

esac

注意事项：

（1）case行尾必须为单词“in”，每一个模式匹配必须以右括号“）”结束。

（2）双分号“**;;**”表示命令序列结束，相当于java中的break。

（3）最后的“\*）”表示默认模式，相当于java中的default。

**2）案例实操**

输入一个数字，如果是1，则输出banzhang，如果是2，则输出cls，如果是其它，输出renyao。

[user1@hadoop101 datas]$ touch case.sh

[user1@hadoop101 datas]$ vim case.sh

!/bin/bash

case $1 in

"1")

echo "banzhang"

;;

"2")

echo "cls"

;;

\*)

echo "renyao"

;;

esac

[user1@hadoop101 datas]$ chmod 777 case.sh

[user1@hadoop101 datas]$ ./case.sh 1

1

## 4.3 for循环

**1）基本语法1**

for (( 初始值;循环控制条件;变量变化 ))

do

程序

done

**2）案例实操**

从1加到100

[user1@hadoop101 datas]$ touch for1.sh

[user1@hadoop101 datas]$ vim for1.sh

#!/bin/bash

sum=0

for((i=0;i<=100;i++))

do

sum=$[$sum+$i]

done

echo $sum

[user1@hadoop101 datas]$ chmod 777 for1.sh

[user1@hadoop101 datas]$ ./for1.sh

5050

**3）基本语法2**

for 变量 in 值1 值2 值3…

do

程序

done

**4）案例实操**

（1）打印所有输入参数

[user1@hadoop101 datas]$ touch for2.sh

[user1@hadoop101 datas]$ vim for2.sh

#!/bin/bash

#打印数字

for i in cls mly wls

do

echo "ban zhang love $i"

done

[user1@hadoop101 datas]$ chmod 777 for2.sh

[user1@hadoop101 datas]$ ./for2.sh

ban zhang love cls

ban zhang love mly

ban zhang love wls

（2）比较$\*和$@区别

$\*和$@都表示传递给函数或脚本的所有参数，不被双引号“”包含时，都以$1 $2 …$n的形式输出所有参数。

[user1@hadoop101 datas]$ touch for1.sh

[user1@hadoop101 datas]$ vim for1.sh

#!/bin/bash

echo '=============$\*============='

for i in $\*

do

echo "ban zhang love $i"

done

echo '=============$@============='

for j in $@

do

echo "ban zhang love $j"

done

[user1@hadoop101 datas]$ chmod 777 for1.sh

[user1@hadoop101 datas]$ ./for1.sh cls mly wls

=============$\*=============

banzhang love cls

banzhang love mly

banzhang love wls

=============$@=============

banzhang love cls

banzhang love mly

banzhang love wls

当它们被双引号“”包含时，$\*会将所有的参数作为一个整体，以“$1 $2 …$n”的形式输出所有参数；$@会将各个参数分开，以“$1” “$2”…“$n”的形式输出所有参数。

[user1@hadoop101 datas]$ vim for4.sh

#!/bin/bash

echo '=============$\*============='

for i in "$\*"

#$\*中的所有参数看成是一个整体，所以这个for循环只会循环一次

do

echo "ban zhang love $i"

done

echo '=============$@============='

for j in "$@"

#$@中的每个参数都看成是独立的，所以“$@”中有几个参数，就会循环几次

do

echo "ban zhang love $j"

done

[user1@hadoop101 datas]$ chmod 777 for4.sh

[user1@hadoop101 datas]$ ./for4.sh cls mly wls

=============$\*=============

banzhang love cls mly wls

=============$@=============

banzhang love cls

banzhang love mly

banzhang love wls

## 4.4 while循环

**1）基本语法**

while [ 条件判断式 ]

do

程序

done

**2）案例实操**

从1加到100

[user1@hadoop101 datas]$ touch while.sh

[user1@hadoop101 datas]$ vim while.sh

#!/bin/bash

sum=0

i=1

while [ $i -le 100 ]

do

sum=$[$sum+$i]

i=$[$i+1]

done

echo $sum

[user1@hadoop101 datas]$ chmod 777 while.sh

[user1@hadoop101 datas]$ ./while.sh

5050

# 第5章 read读取控制台输入

**1）基本语法**

read (选项) (参数)

选项：

-p：指定读取值时的提示符；

-t：指定读取值时等待的时间（秒）。

参数

变量：指定读取值的变量名

**2）案例实操**

提示7秒内，读取控制台输入的名称

[user1@hadoop101 datas]$ touch read.sh

[user1@hadoop101 datas]$ vim read.sh

#!/bin/bash

read -t 7 -p "Enter your name in 7 seconds :" NAME

echo $NAME

[user1@hadoop101 datas]$ ./read.sh

Enter your name in 7 seconds : user1

user1

# 第6章 函数

## 6.1 系统函数

### 6.1.1 basename

**1）基本语法**

basename [string / pathname] [suffix] （功能描述：basename命令会删掉所有的前缀包括最后一个（‘/’）字符，然后将字符串显示出来。

选项：

suffix为后缀，如果suffix被指定了，basename会将pathname或string中的suffix去掉。

**2）案例实操**

截取该/home/user1/banzhang.txt路径的文件名称

[user1@hadoop101 datas]$ basename /home/user1/banzhang.txt

banzhang.txt

[user1@hadoop101 datas]$ basename /home/user1/banzhang.txt .txt

banzhang

### 6.1.2 dirname

**1）基本语法**

dirname 文件绝对路径 （功能描述：从给定的包含绝对路径的文件名中去除文件名（非目录的部分），然后返回剩下的路径（目录的部分））

**2）案例实操**

获取banzhang.txt文件的路径

[user1@hadoop101 ~]$ dirname /home/user1/banzhang.txt

/home/user1

## 6.2 自定义函数

**1）基本语法**

[ function ] funname[()]

{

Action;

[return int;]

}

**2）经验技巧**

（1）必须在调用函数地方之前，先声明函数，shell脚本是逐行运行。不会像其它语言一样先编译。

（2）函数返回值，只能通过$?系统变量获得，可以显示加：return返回，如果不加，将以最后一条命令运行结果，作为返回值。return后跟数值n(0-255)

**3）案例实操**

计算两个输入参数的和

[user1@hadoop101 datas]$ touch fun.sh

[user1@hadoop101 datas]$ vim fun.sh

#!/bin/bash

function sum()

{

s=0

s=$[$1+$2]

echo "$s"

}

read -p "Please input the number1: " n1;

read -p "Please input the number2: " n2;

sum $n1 $n2;

[user1@hadoop101 datas]$ chmod 777 fun.sh

[user1@hadoop101 datas]$ ./fun.sh

Please input the number1: 2

Please input the number2: 5

7

# 第7章 Shell工具

## 7.1 cut

**1）基本用法**

cut [选项参数] filename

说明：默认分隔符是制表符

**2）选项参数说明**

|  |  |
| --- | --- |
| 选项参数 | 功能 |
| -f | 列号，提取第几列 |
| -d | 分隔符，按照指定分隔符分割列，默认是制表符“\t” |
| -c | 指定具体的字符 |

**3）案例实操**

（1）数据准备

[user1@hadoop101 datas]$ touch cut.txt

[user1@hadoop101 datas]$ vim cut.txt

dong shen

guan zhen

wo wo

lai lai

le le

（2）切割cut.txt第一列

[user1@hadoop101 datas]$ cut -d " " -f 1 cut.txt

dong

guan

wo

lai

le

（3）切割cut.txt第二、三列

[user1@hadoop101 datas]$ cut -d " " -f 2,3 cut.txt

shen

zhen

wo

lai

le

（4）在cut.txt文件中切割出guan

[user1@hadoop101 datas]$ cat cut.txt | grep "guan" | cut -d " " -f 1

guan

（5）选取系统PATH变量值，第2个“：”开始后的所有路径：

[user1@hadoop101 datas]$ echo $PATH

/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/user1/.local/bin:/home/user1/bin

[user1@hadoop101 datas]$ echo $PATH | cut -d ":" -f 3-

/usr/local/sbin:/usr/sbin:/home/user1/.local/bin:/home/user1/bin

（6）切割ifconfig 后打印的IP地址

[user1@hadoop101 datas]$ ifconfig ens33 | grep netmask | cut -d "i" -f 2 | cut -d " " -f 2

192.166.4.101

## 7.2 awk

**1）基本用法**

awk [选项参数] ‘/pattern1/{action1} /pattern2/{action2}...’ filename

pattern：表示awk在数据中查找的内容，就是匹配模式

action：在找到匹配内容时所执行的一系列命令

**2）选项参数说明**

|  |  |
| --- | --- |
| 选项参数 | 功能 |
| -F | 指定输入文件折分隔符 |
| -v | 赋值一个用户定义变量 |

**3）案例实操**

（1）数据准备

[user1@hadoop101 datas]$ sudo cp /etc/passwd ./

（2）搜索passwd文件以root关键字开头的所有行，并输出该行的第7列。

[user1@hadoop101 datas]$ awk -F : '/^root/{print $7}' passwd

/bin/bash

（3）搜索passwd文件以root关键字开头的所有行，并输出该行的第1列和第7列，中间以“，”号分割。

[user1@hadoop101 datas]$ awk -F : '/^root/{print $1","$7}' passwd

root,/bin/bash

注意：只有匹配了pattern的行才会执行action

（4）只显示/etc/passwd的第一列和第七列，以逗号分割，且在所有行前面添加列名user，shell在最后一行添加"dahaige，/bin/zuishuai"。

[user1@hadoop101 datas]$ awk -F : 'BEGIN{print "user, shell"} {print $1","$7} END{print "dahaige,/bin/zuishuai"}' passwd

user, shell

root,/bin/bash

bin,/sbin/nologin

。。。

user1,/bin/bash

dahaige,/bin/zuishuai

注意：BEGIN 在所有数据读取行之前执行；END 在所有数据执行之后执行。

（5）将passwd文件中的用户id增加数值1并输出

[user1@hadoop101 datas]$ awk -v i=1 -F : '{print $3+i}' passwd

1

2

3

4

**4）awk的内置变量**

|  |  |
| --- | --- |
| 变量 | 说明 |
| FILENAME | 文件名 |
| NR | 已读的记录数（行号） |
| NF | 浏览记录的域的个数（切割后，列的个数） |

**5）案例实操**

（1）统计passwd文件名，每行的行号，每行的列数

[user1@hadoop101 datas]$ awk -F : '{print "filename:" FILENAME ",linenum:" NR ",col:"NF}' passwd

filename:passwd,linenum:1,col:7

filename:passwd,linenum:2,col:7

filename:passwd,linenum:3,col:7

。。。

（2）查询ifconfig命令输出结果中的空行所在的行号

[user1@hadoop101 datas]$ ifconfig | awk '/^$/{print NR}'

9

18

26

（3）切割IP

[user1@hadoop101 datas]$ ifconfig ens33 | grep netmask | awk -F "inet" '{print $2}' | awk -F " " '{print $1}'

192.166.4.101

## 7.3 sort

**1）基本语法**

Sort (选项) (参数)

|  |  |
| --- | --- |
| 选项 | 说明 |
| -n | 依照数值的大小排序 |
| -r | 以相反的顺序来排序 |
| -t | 设置排序时所用的分隔字符 |
| -k | 指定需要排序的列 |

参数：指定待排序的文件列表

**2）案例实操**

（1）数据准备

[user1@hadoop101 datas]$ touch sort.txt

[user1@hadoop101 datas]$ vim sort.txt

bb:40:5.4

bd:20:4.2

xz:50:2.3

cls:10:1.5

ss:30:1.6

（2）按照“：”分割后的第三列倒序排序。

[user1@hadoop101 datas]$ sort -t : -nrk 3 sort.txt

bb:40:5.4

bd:20:4.2

cls:10:1.5

xz:50:2.3

ss:30:1.6

## 7.4 wc

**1）基本语法**

wc [选项参数] filename

|  |  |
| --- | --- |
| 选项参数 | 功能 |
| -l | 统计文件行数 |
| -w | 统计文件的单词数 |
| -m | 统计文件的字符数 |
| -c | 统计文件的字节数 |

**2）案例实操**

统计/etc/profile文件的行数、单词数、字节数！

[user1@hadoop101 datas]$ wc -l /etc/profile

[user1@hadoop101 datas]$ wc -w /etc/profile

[user1@hadoop101 datas]$ wc -m /etc/profile

# 第8章 正则表达式入门

正则表达式使用单个字符串来描述、匹配一系列符合某个语法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。在Linux中，grep，sed，awk等命令都支持通过正则表达式进行模式匹配。

## 8.1 常规匹配

一串不包含特殊字符的正则表达式匹配它自己，例如：

[user1@hadoop101 datas]$ cat /etc/passwd | grep user1

就会匹配所有包含user1的行

## 8.2 常用特殊字符

**1）特殊字符：^**

^ 匹配一行的开头，例如：

[user1@hadoop101 datas]$ cat /etc/passwd | grep ^a

会匹配出所有以a开头的行

**2）特殊字符：$**

$ 匹配一行的结束，例如

[user1@hadoop101 datas]$ cat /etc/passwd | grep t$

会匹配出所有以t结尾的行

**思考：^$ 匹配什么？**

**3）特殊字符：.**

. 匹配一个任意的字符，例如

[user1@hadoop101 datas]$ cat /etc/passwd | grep r..t

会匹配包含rabt,rbbt,rxdt,root等的所有行

**4）特殊字符：\***

\* 不单独使用，他和上一个字符连用，表示匹配上一个字符0次或多次，例如

[user1@hadoop101 datas]$ cat /etc/passwd | grep ro\*t

会匹配rt, rot, root, rooot, roooot等所有行

**思考：.\* 匹配什么？**

**5）特殊字符：[ ]**

[ ] 表示匹配某个范围内的一个字符，例如

[6,8]------匹配6或者8

[0-9]------匹配一个0-9的数字

[0-9]\*------匹配任意长度的数字字符串

[a-z]------匹配一个a-z之间的字符

[a-z]\* ------匹配任意长度的字母字符串

[a-c, e-f]-匹配a-c或者e-f之间的任意字符

[user1@hadoop101 datas]$ cat /etc/passwd | grep r[a,b,c]\*t

会匹配rt,rat, rbt, rabt, rbact,rabccbaaacbt等等所有行

**6）特殊字符：\**

\ 表示转义，并不会单独使用。由于所有特殊字符都有其特定匹配模式，当我们想匹配某一特殊字符本身时（例如，我想找出所有包含 '$' 的行），就会碰到困难。此时我们就要将转义字符和特殊字符连用，来表示特殊字符本身，例如

[user1@hadoop101 datas]$ cat /etc/passwd | grep a\$b

就会匹配所有包含 a$b 的行。

## 8.3 其他特殊字符

见参考资料的正则表达式语法。