尚硅谷大数据技术之Shell

(作者：尚硅谷大数据研发部)

版本：V3.1

# 第1章 Shell概述



**1）Linux提供的Shell解析器有：**

[atguigu@hadoop101 ~]$ cat /etc/shells

/bin/sh

/bin/bash

/sbin/nologin

/bin/dash

/bin/tcsh

/bin/csh

**2）bash和sh的关系**

[atguigu@hadoop101 bin]$ ll | grep bash

-rwxr-xr-x. 1 root root 941880 5月 11 2016 bash

lrwxrwxrwx. 1 root root 4 5月 27 2017 sh -> bash

**3）Centos默认的解析器是bash**

[atguigu@hadoop102 bin]$ echo $SHELL

/bin/bash

# 第2章 Shell脚本入门

**1）脚本格式**

脚本以#!/bin/bash开头（指定解析器）

**2）第一个Shell脚本：helloworld**

（1）需求：创建一个Shell脚本，输出helloworld

（2）案例实操：

[atguigu@hadoop101 datas]$ touch helloworld.sh

[atguigu@hadoop101 datas]$ vi helloworld.sh

在helloworld.sh中输入如下内容

#!/bin/bash

echo "helloworld"

（3）脚本的常用执行方式

第一种：采用bash或sh+脚本的相对路径或绝对路径（不用赋予脚本+x权限）

sh+脚本的相对路径

[atguigu@hadoop101 datas]$ sh helloworld.sh

Helloworld

sh+脚本的绝对路径

[atguigu@hadoop101 datas]$ sh /home/atguigu/datas/helloworld.sh

helloworld

bash+脚本的相对路径

[atguigu@hadoop101 datas]$ bash helloworld.sh

Helloworld

bash+脚本的绝对路径

[atguigu@hadoop101 datas]$ bash /home/atguigu/datas/helloworld.sh

Helloworld

第二种：采用输入脚本的绝对路径或相对路径执行脚本（必须具有可执行权限+x）

（a）首先要赋予helloworld.sh 脚本的+x权限

[atguigu@hadoop101 datas]$ chmod +x helloworld.sh

（b）执行脚本

相对路径

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

Helloworld

绝对路径

[atguigu@hadoop101 datas]$ /home/atguigu/datas/helloworld.sh

Helloworld

注意：第一种执行方法，本质是bash解析器帮你执行脚本，所以脚本本身不需要执行权限。第二种执行方法，本质是脚本需要自己执行，所以需要执行权限。

# 第3章 变量

## 3.1 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

[atguigu@hadoop101 datas]$ echo $HOME

/home/atguigu

（2）显示当前Shell中所有变量：set

[atguigu@hadoop101 datas]$ set

BASH=/bin/bash

BASH\_ALIASES=()

BASH\_ARGC=()

BASH\_ARGV=()

## 3.2 自定义变量

**1）基本语法**

（1）定义变量：变量=值

（2）撤销变量：unset 变量

（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

[atguigu@hadoop101 datas]$ A=5

[atguigu@hadoop101 datas]$ echo $A

5

（2）给变量A重新赋值

[atguigu@hadoop101 datas]$ A=8

[atguigu@hadoop101 datas]$ echo $A

8

（3）撤销变量A

[atguigu@hadoop101 datas]$ unset A

[atguigu@hadoop101 datas]$ echo $A

（4）声明静态的变量B=2，不能unset

[atguigu@hadoop101 datas]$ readonly B=2

[atguigu@hadoop101 datas]$ echo $B

2

[atguigu@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

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

[atguigu@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

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

-bash: world: command not found

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

[atguigu@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

[atguigu@hadoop101 datas]$ vim helloworld.sh

在helloworld.sh文件中增加echo $B

#!/bin/bash

echo "helloworld"

echo $B

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

Helloworld

发现并没有打印输出变量B的值。

[atguigu@hadoop101 datas]$ export B

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

helloworld

2

## 3.3 特殊变量

### 3.3.1 $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**

[atguigu@hadoop101 datas]$ touch parameter.sh

[atguigu@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo "$0 $1 $2"

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

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

./parameter.sh cls xz

### 3.3.2 $#

**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**

[atguigu@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo "$0 $1 $2"

echo $#

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

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

parameter.sh cls xz

2

### 3.3.3 $\*、$@

**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）

注意：如果想让$\*和$@ 体现区别必须用双引号括起来才生效

**2）案例实操**

[atguigu@hadoop101 datas]$ vim parameter.sh

#!/bin/bash

echo "$0 $1 $2"

echo $#

echo $\*

echo $@

[atguigu@hadoop101 datas]$ bash parameter.sh 1 2 3

parameter.sh 1 2

3

1 2 3

1 2 3

### 3.3.4 $？

**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

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

hello world

[atguigu@hadoop101 datas]$ echo $?

0

# 第4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）X4的值

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

[atguigu@hadoop101 datas]# echo $S

# 第5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，[ atguigu ]返回true，[] 返回false。

**2）常用判断条件**

（1）两个整数之间比较

== 字符串比较

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

-eq 等于（equal） -gt 大于（greater than）

-ge 大于等于（greater equal） -ne 不等于（Not 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

-f 文件存在并且是一个常规的文件（file）

-e 文件存在（existence） -d 文件存在并是一个目录（directory）

**3）案例实操**

（1）23是否大于等于22

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

[atguigu@hadoop101 datas]$ echo $?

0

（2）helloworld.sh是否具有写权限

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

[atguigu@hadoop101 datas]$ echo $?

0

（3）/home/atguigu/cls.txt目录中的文件是否存在

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

[atguigu@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）

[atguigu@hadoop101 ~]$ [ condition ] && echo OK || echo notok

OK

[atguigu@hadoop101 datas]$ [ condition ] && [ ] || echo notok

notok

# 第6章 流程控制（重点）

## 6.1 if判断

**1）基本语法**

if [ 条件判断式 ];then

程序

fi

或者

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。

[atguigu@hadoop101 datas]$ touch if.sh

[atguigu@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

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

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

banzhang zhen shuai

## 6.2 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。

[atguigu@hadoop101 datas]$ touch case.sh

[atguigu@hadoop101 datas]$ vim case.sh

!/bin/bash

case $1 in

"1")

echo "banzhang"

;;

"2")

echo "cls"

;;

\*)

echo "renyao"

;;

esac

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

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

1

## 6.3 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

[atguigu@hadoop101 datas]$ touch for1.sh

[atguigu@hadoop101 datas]$ vim for1.sh

#!/bin/bash

s=0

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

do

s=$[$s+$i]

done

echo $s

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

[atguigu@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

[atguigu@hadoop101 datas]$ touch for2.sh

[atguigu@hadoop101 datas]$ vim for2.sh

#!/bin/bash

#打印数字

for i in $\*

do

echo "ban zhang love $i "

done

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

[atguigu@hadoop101 datas]$ bash for2.sh cls xz bd

ban zhang love cls

ban zhang love xz

ban zhang love bd

（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。

[atguigu@hadoop101 datas]$ touch for.sh

[atguigu@hadoop101 datas]$ vim for.sh

#!/bin/bash

for i in $\*

do

echo "ban zhang love $i "

done

for j in $@

do

echo "ban zhang love $j"

done

[atguigu@hadoop101 datas]$ bash for.sh cls xz bd

ban zhang love cls

ban zhang love xz

ban zhang love bd

ban zhang love cls

ban zhang love xz

ban zhang love bd

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。

[atguigu@hadoop101 datas]$ vim for.sh

#!/bin/bash

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

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

[atguigu@hadoop101 datas]$ chmod 777 for.sh

[atguigu@hadoop101 datas]$ bash for.sh cls xz bd

ban zhang love cls xz bd

ban zhang love cls

ban zhang love xz

ban zhang love bd

## 6.4 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

[atguigu@hadoop101 datas]$ touch while.sh

[atguigu@hadoop101 datas]$ vim while.sh

#!/bin/bash

s=0

i=1

while [ $i -le 100 ]

do

s=$[$s+$i]

i=$[$i+1]

done

echo $s

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

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

5050

# 第7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

[atguigu@hadoop101 datas]$ touch read.sh

[atguigu@hadoop101 datas]$ vim read.sh

#!/bin/bash

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

echo $NAME

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

Enter your name in 7 seconds xiaoze

xiaoze

# 第8章 函数

## 8.1 系统函数

### 8.1.1 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/atguigu/banzhang.txt路径的文件名称

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

banzhang.txt

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

banzhang

### 8.1.2 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

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

/home/atguigu

## 8.2 自定义函数

**1）基本语法**

**#定义函数**

[ function ] funname[()]

{

Action;

[return int;]

}

#调用函数

funname

**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

[atguigu@hadoop101 datas]$ touch fun.sh

[atguigu@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;

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

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

Please input the number1: 2

Please input the number2: 5

7

# 第9章 Shell工具

## 9.1 cut

cut的工作就是“剪”，具体的说就是在文件中负责剪切数据用的。cut 命令从文件的每一行剪切字节、字符和字段并将这些字节、字符和字段输出。

**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 |
| -c | 指定具体的字符 |

**3）案例实操**

（1）数据准备

[atguigu@hadoop101 datas]$ touch cut.txt

[atguigu@hadoop101 datas]$ vim cut.txt

dong shen

guan zhen

wo wo

lai lai

le le

（2）切割cut.txt第一列

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

dong

guan

wo

lai

le

（3）切割cut.txt第二、三列

[atguigu@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

[atguigu@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：

[atguigu@hadoop101 datas]$ echo $PATH

/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/atguigu/bin

[atguigu@hadoop102 datas]$ echo $PATH | cut -d: -f 2-

/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/atguigu/bin

（6）切割ifconfig 后打印的IP地址

[atguigu@hadoop101 datas]$ ifconfig eth0 | grep "inet addr" | cut -d: -f 2 | cut -d" " -f1

192.168.1.102

## 9.2 awk

一个强大的文本分析工具，把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行分析处理。

**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

[atguigu@hadoop102 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。

[atguigu@hadoop102 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。

[atguigu@hadoop102 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"。

[atguigu@hadoop102 datas]$ awk -F : 'BEGIN{print "user, shell"} {print $1","$7} END{print "dahaige,/bin/zuishuai"}' passwd

user, shell

root,/bin/bash

bin,/sbin/nologin

。。。

atguigu,/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。

**4）awk的内置变量**

|  |  |
| --- | --- |
| 变量 | 说明 |
| FILENAME | 文件名 |
| NR | 已读的记录数（行数） |
| 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

[atguigu@hadoop102 datas]$ awk -F: '{print "filename:" FILENAME ", linenumber:" NR ",columns:" NF}' passwd

filename:passwd, linenumber:1,columns:7

filename:passwd, linenumber:2,columns:7

filename:passwd, linenumber:3,columns:7

（2）切割IP

[atguigu@hadoop102 datas]$ ifconfig | grep netmask | awk -F "inet " '{print $2}' | awk -F " " '{print $1}'

192.168.1.102

## 9.3 sort

sort命令是在Linux里非常有用，它将文件进行排序，并将排序结果标准输出。

**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

[atguigu@hadoop102 datas]$ touch sort.sh

[atguigu@hadoop102 datas]$ vim sort.sh

bb:40:5.4

bd:20:4.2

xz:50:2.3

cls:10:3.5

ss:30:1.6

（2）按照“：”分割后的第三列倒序排序。

[atguigu@hadoop102 datas]$ sort -t : -nrk 3 sort.sh

bb:40:5.4

bd:20:4.2

cls:10:3.5

xz:50:2.3

ss:30:1.6

# 第10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句法规则的字符串。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。

## 10.1 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：

[atguigu@hadoop102 datas]$ cat /etc/passwd | grep atguigu

就会匹配所有包含atguigu的行

## 10.2 常用特殊字符

**1）特殊字符：^**

^ 匹配一行的开头，例如：

[atguigu@hadoop102 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

[atguigu@hadoop102 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

[atguigu@hadoop102 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，他和左边第一个字符连用，表示匹配上一个字符0次或多次，例如

[atguigu@hadoop102 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

[a-z]------匹配一个a-z之间的字符

[a-z]\*-----匹配任意字母字符串

[a-c, e-f]-匹配a-c或者e-f之间的任意字符

[atguigu@hadoop102 datas]$ cat /etc/passwd | grep r[a,b,c]\*t

会匹配rat, rbt, rabt, rbact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

[atguigu@hadoop102 datas]$ cat /etc/passwd | grep a\$b

注意：直接匹配 $ 字符，需要进行转义并且加上单引号

就会匹配所有包含 a$b 的行。

## 10.3 其他特殊字符

见参考资料的正则表达式语法。