Linux操作系统

6 Shell 变量

主讲：杨东平
中国矿大计算机学院

Shell 变量的分类

- 用户自定义变量
- 环境变量
  - 主要用来保存和系统环境相关的数据，在程序中常用的有 PATH，LOGNAME，HOME，PWD，MAIL等 env 命令
- 位置参数变量
  - 出现在命令行上的位置确定的参数称为位置参数变量，主要用来向脚本中传递参数或数据，变量名不能自定义，变量的作用是固定的
- 预定义变量
  - Shell 中已经定义好的变量，变量名不能自定义，作用也是固定的
  - 位置参数变量实际是预定义变量的一种

用户自定义变量

- 用户自定义变量也叫本地变量，用户可以自定义变量的名称，也可以给变量赋值，还可修改变量的值，甚至删除变量

用户自定义变量的命名规则

- (1) 变量的名称可以用字母、数字和下划线组成，但不能以数字开头
- (2) 变量的值的类型默认是字符串型，如果要进行数值运算，就需指定变量类型为数值型
- (3) 变量可以用等号 = 来赋值，等号左右两侧不能有空格
- (4) 变量的值如果包含空格，需要用单引号或双引号括起来
- (5) 在变量的值中，可以使用 \ 转义符
- (6) 如果需要增加变量的值，可以进行变量值的叠加。不过，变量需要用双引号包含"变量名"或用{变量名}包含
- (7) 如果要把命令的结果作为值赋给变量，则需要使用反引号或 $() 包含命令
- (8) 环境变量的名称建议大写，便于区分
- (9) 变量名的长度不得超过 255 个字符
- (10) 变量名在有效范围内必须是唯一的

用户自定义变量(续)

- 定义变量
  - 格式： 变量名=变量的值
  - 例： name="super man"
    x=5 注意：5是字符，不是数字
  - 说明：= 左右不能加空格
- 调用变量
  - 格式： $变量名
  - 例： echo $name
  - 注意：在调用变量时，最好用双引号括起来。如："num"、"name$i"。否则，很容易出现下面的错误信息。
    - too many arguments
    - unary operator expected

用户自定义变量(续)

- 查看变量
  - 格式： set [选项]
  - 选项：
    - -u 使用此选项时，调用未声明变量时会报错(默认无任何提示)
  - 功能：查看系统中所有的变量(包括自定义变量和环境变量)
- 删除变量
  - 格式： unset 变量名
  - 说明：删除变量时不需要在变量名前加 $ 符号

```
[root@localhost ~]# unset a
[root@localhost ~]# echo $a

[root@localhost ~]# _
```

用户自定义变量(续)

- 只读变量
  - 使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变
  - 例：

```
myUrl="http://www.google.com"
readonly myUrl
myUrl="http://www.runoob.com"
```

```
[root@localhost ~]# myUrl="http://www.google.com"
[root@localhost ~]# readonly myUrl
[root@localhost ~]# myUrl="http://www.runoob.com"
-bash: myUrl: readonly variable
```


Shell 字符串

- 单引号字符串
  - 例： str='this is a string'
  - 单引号字符串的限制：
    - 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的
    - 单引号字串中不能出现单独一个的单引号(对单引号使用转义符后也不行)，但可成对出现，作为字符串拼接使用

Shell 字符串

- 双引号字符串
  - 例：

```
your_name='runoob'
str="Hello, I know you are \"$your_name\"! \n"
echo $str
```

结果：Hello, I know you are "runoob"!

  - 双引号的优点：
    - 双引号里可以有变量
    - 双引号里可以出现转义字符

用户自定义变量(续)

- 叠加变量(拼接字符串)：指将变量的值和其它字符串拼接起来
  - 例1：

```
a=123
a="$a"456
a=${a}789
echo $a
```

```
[root@localhost ~]# a=123
[root@localhost ~]# a="$a"456
[root@localhost ~]# a=${a}789
[root@localhost ~]# echo $a
123456789
```

  - 例2：

```
your_name="runoob"
# 使用双引号拼接
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting  $greeting_1
```

结果：hello, runoob ! hello, runoob !

  - 例2(续)：

```
# 使用单引号拼接
greeting_2='hello, '$your_name' !'
greeting_3='hello, ${your_name} !'
echo $greeting_2  $greeting_3
```

结果： hello, runoob ! hello, ${your_name} !

环境变量

- 环境变量包括：
  - 系统级环境变量：每个登录到系统的用户都要读取的系统变量
    - /etc/environment：登录系统时读取的第一个文件，用于为所有进程设置环境变量，内容格式：KEY=VALUE
      - 在此文件中定义新的 PATH 环境变量，只需加入一行形如：PATH=$PATH:/xxx/bin 的代码即可
    - /etc/profile：登录系统时执行的第二个文件，可以用于设定针对全系统所有用户的环境变量，该文件一般调用 /etc/bash.bashrc 文件
    - /etc/bash.bashrc：为每一个运行 bash 的用户执行此文件。此文件会在用户每次打开 shell 时执行一次
    - 注意：
      - /etc/environment 是设置整个系统的环境，而 /etc/profile 是设置所有用户的环境，前者与登录用户无关，后者与登录用户有关
      - 这两个文件修改后一般都要重启系统才能生效

环境变量(续)

- 环境变量包括(续)：
  - 用户级环境变量：是登录用户使用系统时加载的环境变量，匹配文件在家目录下
    - ~/.profile:：当前登录用户的 profile 文件，用于定制当前用户的个人工作环境
      - 每个用户都可使用该文件输入专用于自己使用的 shell 信息，当用户登录时，该文件仅仅执行一次！默认情况下，他设置一些环境变量，执行用户的 .bashrc 文件
    - ~/.bashrc：当前登录用户的 bash 初始化文件，当用户每次打开 shell 时，系统都会执行此文件一次
- 总结：以上文件的执行先后顺序是

/etc/enviroment -> /etc/profile -> ~/.profile -> /etc/bash.bashrc -> ~/.bashrc

环境变量(续)

- 环境变量与用户自定义变量的区别
  - 环境变量是全局变量，用户自定义变量是局部变量
  - 环境变量在当前 Shell 和这个 Shell 的所有子 Shell 中生效，用户自定义变量只在当前的 Shell 中生效
  - 对系统生效的环境变量名和变量作用是固定的

环境变量(续)

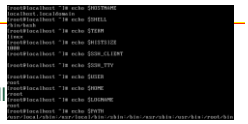
- 设置环境变量
  - 设置 shell 下临时使用的环境变量
    - 格式： export 变量名=变量值
      或 变量名=变量值
      export 变量名
    - 说明：
      - 也可用 set 设置
      - 只能在当前 shell 脚本下可用，切换到另一个终端就会失效

环境变量(续)

- 查看环境变量
  - 查看用户定义的所有变量
    - 格式： set
    - 说明：显示普通变量和环境变量
  - 查看环境变量
    - 格式： env
    - 说明：只显示环境变量

环境变量(续)

- 常用环境变量



HOSTNAME	主机名
SHELL	当前使用的 shell
TERM	终端环境
HISTSIZE	历史命令条数
SSH_CLIENT	当前操作环境是用ssh连接的，这里记录客户端 ip
SSH_TTY	ssh连接的终端时 pts/1
USER	当前登录的用户
HOME	当前登录用户的主工作目录
PATH	命令的搜索路径
LOGNAME	当前用户的登录名

环境变量(续)

- 添加环境变量路径
  - 格式：PATH="$PATH":新路径
  - 说明：一旦退出当前系统，需要重新定义
  - 例： PATH="$PATH":/root/sh/
    #用拼接的方法增加 PATH 变量的值

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@localhost ~]# PATH="$PATH":/root/
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root/
[root@localhost ~]# PATH="$PATH":/root/sh
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root/:/r
oot/sh
```


环境变量(续)

- 命令行提示符修改(环境变量PS1)

符号	含义
\d	显示日期，格式为"星期 月 日"，如："Mon Aug 1"
\H	显示完整的主机名称。如："默认主机名"localhost.localdomain"
\h	仅取主机的第一个名字，如上例中的"localhost"，其后内容被省略
\t	显示时间为24小时格式：HH:MM:SS
\T	显示时间为12小时格式
\A	显示时间为24小时格式：HH:MM
\u	显示当前用户名
\v	显示 BASH 版本信息
\w	显示当前所在目录的完整名称。家目录会以 ~ 代替
\W	显示当前所在目录的最后一个目录，它是利用basename取得工作目录名称，所以只会列出最后一个目录
\#	下达的第几个命令
\$	提示符，如果是root用户则提示符为"#"，普通用户则提示符为"$"

默认的 PS1 定义为：PS1='[\u@\h \W]\$ '
所以默认提示符为： [root@localhost ~]#

```
[root@localhost ~]# echo $PS1
[\u@\h \W]\$
[root@localhost ~]# _
```

位置参数变量

- 位置参数是一种在调用 Shell 程序的命令行中按照各自的位置决定的变量，是在程序名之后输入的参数，它们分别标识了用户输入的整个命令行中以空格分隔开的字符串，主要用来向脚本中传递参数或数据

位置参数变量	作用
$#	传递给命令的总的参数数目
$*	传递给命令的所有参数组成的字符串
$n	表示第 n 个参数，$1 表示第一个参数，$2 表示第二个参数…，十个以上的参数需要用大括号包含，如 ${10}
$0	当前程序的名称
$@	以“参数1”、“参数2”……保存所有的参数

位置参数变量(续)

- 例1：shart.sh代码

```
#!/bin/bash
#sharg.sh
echo “Command received $# params.”
echo “Command:$0”
echo “Arg1:$1”
echo “Arg2:$2”
echo “Arg3:$3”
```

- # chmod 755 sharg.sh
- # ./sharg.sh 80 f1 f2
- 视频：3 位置参数变量1

```
[root@localhost ~]# chmod 755 sharg.sh
[root@localhost ~]# ./sharg.sh 80 f1 f2
Command received 3 params.
Command:./sharg.sh
Arg1:80
Arg2:f1
Arg3:f2
```


位置参数变量(续)

- 例2：sharg1.sh 源代码

```
#! /bin/bash
num1=$1
num2=$2
sum=$(($num1+$num2))
#变量sum的和是num1加num2
echo $sum
$打印变量sum的值
```

- # chmod 755 sharg1.sh
- # ./sharg2.sh 5 6
- 视频：4 位置参数变量2

```
[root@localhost ~]# ./sharg2.sh 5 6
11
```


位置参数变量(续)

- $* 与 $@ 的区别
  - 代码1(视频：5 位置参数变量3)：

```
#!/bin/bash
#sharg3.sh
for i in “$*”
# $* 中的所有参数看成是一个整体，所以这个for循环只会循环一次
do
   echo “The Parameters is:$i”
done
```

```
[root@localhost ~]# chmod 755 sharg3.sh
[root@localhost ~]# ./sharg3.sh 1 2 3 a b c
The Parameters is:1 2 3 a b c
```

  - 代码2(视频：6 位置参数变量4)：

```
#!/bin/bash
#sharg4.sh
for y in “$@”
# $@ 中的每个参数看成是独立的，所以“$@”中有几个参数就会循环几次
do
   echo “The Parameters:$y”
done
```

```
[root@localhost ~]# chmod 755 sharg5.sh
chmod: cannot access `sharg5.sh': No such file or directory
[root@localhost ~]# chmod 755 sharg4.sh
[root@localhost ~]# ./sharg4.sh 1 2 3 a b c
The Parameters:1
The Parameters:2
The Parameters:3
The Parameters:a
The Parameters:b
The Parameters:c
```


预定义变量

预定义变量	作用
$?	上一个代码或者 Shell 程序在 Shell 中退出的情况，如正常退出则返回 0，否则返回非 0 值(这个非 0 值由程序自己决定)
$$	当前进程的进程号 PID (进程ID)
$!	后台运行的最后一个进程的进程号 PID

接收键盘输入

- 格式： read [选项] [变量名]
- 选项：

-p “提示信息”	在等待read输入时，输出的提示信息
-t 秒数	read 命令会一直等待用户输入，此选项可以指定等待时间
-n 字符数	read 命令只接收指定的字符数，然后执行
-s	隐藏输入的数据，适用于机密信息的输入

接收键盘输入(续)

- 例(视频：7接收键盘输入)：

```
#!/bin/bash
#keyinput.sh

# 提示用户输入并等待30秒，并将输入结果存入 name 变量中
read -t 30 -p "Please input your name: " name
echo -e "\n"
echo "Name is $name"

# 加上 -s 以后当输入 age 的时候将隐藏起来
read -s -t 30 -p "Please input your age: " age
echo -e "\n"
echo "Age is $age"

# -n 1 代表只接收一个字符
read -n 1 -t 30 -p "Please input gender M/F :" gender
echo -e "\n"
echo "Genger is $gender"
```

- [root@localhost ~]# chmod 755 keyinput.sh
- [root@localhost ~]# ./keyinput.sh

```
Please input your name: xiaol

Name is xiaol
Please input your age:

Age is 12
Please input gender M/F :M

Genger is M
```