

大数据Hadoop高薪直通车课程

Shell 基础编程

讲师：轩宇（北风网版权所有）

课程大纲

- Shell 基础编程
 - Shell 介绍及基本案例
 - 变量、控制、循环基本语法

Shell 程序

- ◆ 以文件形式存放**批量的Linux命令集合**，该文件能够被Shell解释执行，这种文件就是Shell脚本程序。
- ◆ 通常由**一段Linux命令、Shell命令、控制语句以及注释语句**构成
- ◆ Shell 脚本的编写
 - Shell 脚本是**纯文本文件**，可以使用任何文本编辑器编写
 - Shell 脚本通常是以 **.sh** 作为后缀名

Shell 程序

- ◆ 第一行：指定用哪个程序来编译和执行脚本。

```
#!/bin/bash
```

```
#!/bin/sh
```

- ◆ 注释行：使用（#）符号

变量

➤ 变量命名

- 变量名必须以字母或下划线开头，后面可以跟字母、数字或下划线。任何其它字符都标志变量名的结束。
- 变量名关于大小写敏感。

➤ 变量类型

- 根据变量的作用域，变量可以分为本地变量和环境变量
- 本地变量只在创建它们的 shell 程序中可用。而环境变量则在Shell中的所有用户进程中可用，通常也称为全局变量。

变量

◆ 变量赋值

- 等号两边不能有空格
- 如果要给变量赋空值，可以在等号后面跟一个换行符

◆ 显示变量的值

```
echo $variable 或 echo ${variable}
```

◆ 清除变量

```
unset variable
```

◆ 显示所有变量

```
set
```

环境变量

◆环境变量称为全局变量，按照惯例需要大写

export LANG

- 注意：可被所有的Shell环境下访问；
- 如果父Shell进程产生了子Shell进程，则环境变量可被“继承”并复制

位置参量

- ◆ 位置参量是一组特殊的内置变量，通常被 `shell` 脚本用来从命令行接受参数，或被函数用来保存传递给它的参数。
- ◆ 执行 `shell` 脚本时，用户可以通过命令行向脚本传递信息，跟在脚本名后面的用空格隔开的每个字符串都称为位置参量。
- ◆ 在脚本中使用这些参数时，需通过位置参量来引用。例如：
`$1` 表示第一个参数，`$2` 表示第二个参数，以此类推。`$9` 以后需要用花括号把数字括起来，如第 `10` 个位置参量以 `${10}` 的方式来访问。

位置参量列表

\$0	当前脚本的文件名
\$1-\$9	第 1 个到第 9 个位置参量
\${10}	第 10 个位置参量，类似地，有 \${11} ，...
\$#	位置参量的个数
\$*	以单字符串显示所有位置参量
\$@	未加双引号时与 \$* 含义相同，加双引号时有区别
\$\$	脚本运行的当前进程号
#!	最后一个后台运行的进程的进程号
\$?	显示前面最后一个命令的退出状态。 0 表示没有错误，其他任何值表示有错误。

位置参量

#例1: shell-test.sh

```
echo "the count of parameters:$#"
```

```
echo "first param=$1"
```

```
echo "second param=$2"
```

```
echo "params' string=$@"
```

- ◆ 给SHELL程序传递参数

```
shell-test.sh This is Peter
```

- ◆ 如果位置参量中含有空格，则需要使用双引号

```
shell-test.sh This is "Peter Piper"
```

退出码

◆ 任何命令进行时都将返回一个退出状态。

◆ 查看命令：**echo \$?**

◆ 应用中通常会在关键步骤后判定\$?，已确定关键步骤的执行是否正常。
尤其调度系统里需要监控sh返回码。

◆ shell 脚本的返回码取决于最后一个命令的返回码！

◆ 程序控制返回码：**exit N**

➤ 退出状态0 成功，无错误。

➤ 退出状态大于0，失败，某处有错误。

数组

◆数组定义

```
arr=(math english chinese)
```

◆数组初始化

```
arr=(math english chinese)
```

◆数组引用

- 引用变量: `${arr[0]}`
- 数组个数: `${#arr[*]}`
- 所有元素: `${arr[*]}`

◆数组赋值

```
arr[0]=chemical
```

date 命令

如果在文字接口中想要知道目前 Linux 系统的时间，那么就直接在指令列模式输入 date 即可显示：

```
[vbird@www ~]$ date  
Mon Aug 17 17:02:52 CST 2009
```

上面显示的是：星期一，八月十七日，17:02 分，52 秒，在 2009 年的 CST 时区！台湾在 CST 时区中啦！请赶快动手做做看呦！好了，那么如果我想要让这个程序显示出『2009/08/17』这样的日期显示方式呢？那么就使用 date 的格式化输出功能吧！

```
[vbird@www ~]$ date +%Y/%m/%d  
2009/08/17  
[vbird@www ~]$ date +%H:%M  
17:04
```

3. 开始利用 date 指令来取得所需要的档名了；

```
date1=$(date --date='2 days ago' +%Y%m%d) # 前两天的日期
```

```
date2=$(date --date='1 days ago' +%Y%m%d) # 前一天的日期
```

```
date3=$(date +%Y%m%d) # 今天的日期
```

cal 命令

那如果我想要列出目前这个月份的月历呢？呵呵！直接给他下达 cal 即可！

```
[vbird@www ~]$ cal
```

```
[vbird@www ~]$ cal 2009
```

基本上 cal 这个指令可以接的语法为：

```
[vbird@www ~]$ cal [month] [year]
```

所以，如果我想要知道 2009 年 10 月的月历，可以直接下达：

```
[vbird@www ~]$ cal 10 2009
```

判断

1. 关于某个档名的『文件类型』判断，如 `test -e filename` 表示存在否

-e	该『档名』是否存在？(常用)
-f	该『档名』是否存在且为档案(file)？(常用)
-d	该『文件名』是否存在且为目录(directory)？(常用)

2. 关于档案的权限侦测，如 `test -r filename` 表示可读否 (但 root 权限常有例外)

-r	侦测该档名是否存在且具有『可读』的权限？
-w	侦测该档名是否存在且具有『可写』的权限？
-x	侦测该档名是否存在且具有『可执行』的权限？

判断

4. 关于两个整数之间的判定，例如 `test n1 -eq n2`

<code>-eq</code>	两数值相等 (equal)
<code>-ne</code>	两数值不等 (not equal)
<code>-gt</code>	n1 大于 n2 (greater than)
<code>-lt</code>	n1 小于 n2 (less than)
<code>-ge</code>	n1 大于等于 n2 (greater than or equal)
<code>-le</code>	n1 小于等于 n2 (less than or equal)

5. 判定字符串的数据

<code>test -z string</code>	判定字符串是否为 0？若 <code>string</code> 为空字符串，则为 <code>true</code>
<code>test -n string</code>	判定字符串是否非为 0？若 <code>string</code> 为空字符串，则为 <code>false</code> 。 注： <code>-n</code> 亦可省略
<code>test str1 = str2</code>	判定 <code>str1</code> 是否等于 <code>str2</code> ，若相等，则回传 <code>true</code>
<code>test str1 != str2</code>	判定 <code>str1</code> 是否不等于 <code>str2</code> ，若相等，则回传 <code>false</code>

判断

💡 利用判断符号 []

除了我们很喜欢使用的 `test` 之外，其实，我们还可以利用判断符号 `[[]]` (就是中括号啦) 来进行数据的判断呢！举例来说，如果我想要知道 `$HOME` 这个变量是否为空的，可以这样做：

```
[root@www ~]# [ -z "$HOME" ]; echo $?
```

使用中括号必须要特别注意，因为中括号用在很多地方，包括通配符与正规表示法等等，所以如果要在 `bash` 的语法当中使用中括号作为 `shell` 的判断式时，必须要注意中括号的两端需要有空格符来分隔喔！假设我空格键使用 `[c]` 符号来表示，那么，在这些地方你都需要有空格键：

```
[ "$HOME" == "$MAIL" ]
```

if 判断

- 单层、简单条件判断式

如果你只有一个判断式要进行，那么我们可以简单的这样看：

```
if [ 条件判断式 ]; then
```

当条件判断式成立时，可以进行的指令工作内容：

```
fi
```

==将 if 反过来写，就成为 fi 啦！结束 if 之意！

至于条件判断式的判断方法，与前一小节介绍相同啊！较特别的是，如果我有多个条件要判别时，除了 `sh06.sh` 那个案例所写的，也就是『将多个条件写入一个中括号内的情况』之外，我还可以有多个中括号来隔开喔！而括号与括号之间，则以 `&&` 或 `||` 来隔开，他们的意义是：

- `&&` 代表 AND ；
- `||` 代表 or ；

if 判断

```
if [ "$yn" == "Y" ] || [ "$yn" == "y" ]; then
    echo "OK, continue"
    exit 0
fi
if [ "$yn" == "N" ] || [ "$yn" == "n" ]; then
    echo "Oh, interrupt!"
    exit 0
fi
```

if 判断

- 多重、复杂条件判断式

在同一个数据的判断中，如果该数据需要进行多种不同的判断时，应该怎么做？举例来说，上面的 `sh06.sh` 脚本中，我们只要进行一次 `$yn` 的判断就好（仅进行一次 `if`），不想要作多次 `if` 的判断。此时你就得要知道底下的语法了：

```
# 一个条件判断，分成功进行与失败进行 (else)
if [ 条件判断式 ]; then
    当条件判断式成立时，可以进行的指令工作内容；
else
    当条件判断式不成立时，可以进行的指令工作内容；
fi
```

if 判断

如果考虑更复杂的情况，则可以使用这个语法：

```
# 多个条件判断 (if ... elif ... elif ... else) 分多种不同情况执行
```

```
if [ 条件判断式一 ]; then
```

```
    当条件判断式一成立时，可以进行的指令工作内容；
```

```
elif [ 条件判断式二 ]; then
```

```
    当条件判断式二成立时，可以进行的指令工作内容；
```

```
else
```

```
    当条件判断式一与二均不成立时，可以进行的指令工作内容；
```

```
fi
```

你得要注意的是，elif 也是个判断式，因此出现 elif 后面都要接 then 来处理！但是 else 已经是最后的没有成立的结果了，所以 else 后面并没有 then 喔！好！我们来将 sh06-2.sh 改写成这样：

for 循环

```
for var in con1 con2 con3 ...  
do  
    程序段  
done
```

以上面的例子来说，这个 \$var 的变量内容在循环工作时：

1. 第一次循环时，\$var 的内容为 con1；
2. 第二次循环时，\$var 的内容为 con2；
3. 第三次循环时，\$var 的内容为 con3；
4.

```
for animal in dog cat elephant  
do  
    echo "There are ${animal}s...."  
done
```

for 循环

```
for (( 初始值; 限制值; 执行步阶 ))  
do  
    程序段  
done
```

这种语法适合于数值方式的运算当中，在 for 后面的括号内的三单内容意义为：

- 初始值：某个变量在循环当中的起始值，直接以类似 $i=1$ 设定好；
- 限制值：当变量的值在这个限制值的范围内，就继续进行循环。例如 $i \leq 100$ ；
- 执行步阶：每作一次循环时，变量的变化量。例如 $i=i+1$ 。

```
s=0  
for (( i=1; i=$nu; i=i+1 ))  
do  
    s=$((s+i))  
done
```

while 循环

🔔 while do done, until do done (不定循环)

一般来说，不定循环最常见的就是底下这两种状态了：

```
while [ condition ] <==中括号内的状态就是判断式
do      <==do 是循环的开始！
    程序段落
done    <==done 是循环的结束
```

while 的中文是『当...时』，所以，这种方式说的是『当 condition 条件成立时，就进行循环，直到 condition 的条件不成立才停止』的意思。还有另外一种不定循环的方式：

```
until [ condition ]
do
    程序段落
done
```

这种方式恰恰与 while 相反，它说的是『当 condition 条件成立时，就终止循环，否则就持续进行循环的程序段。』是否刚好相反啊～我们以 while 来做个简单的练习好了。假设我要让使用者输入 yes 或者是 YES 才结束程序的执行，否则就一直进行告知用户输入字符串。

while read line

从文件或命令中逐行读取

```
cat file | while read line
do
echo $line
done
```

或

```
cat `ls ./*.txt` | while read line
do
echo $line
done
```

本课程版权归北风网所有

欢迎访问我们的官方网站

www.ibeifeng.com