Shell 变量和 **Shell** 环境

内容提要

1. 学会使用自定义变量和环境变量
2. 掌握变量替换及输出方法
3. 区分 ' '、" " 和 ` `的作用
4. 理解 Shell 变量的作用域
5. 了解不同工作环境文件的作用
6. 学会设置用户自己的工作环境

Shell 变量简介

作为一种编程语言，Shell 支持变量、数组、流程控制、函数等编程语言的基本要素。简单的说，Shell 变量分为自定义变量和环境变量。本节讲述 Shell 变量的使用。

用户自定义变量

定义 Shell 变量

Shell 支持具有字符串值的变量。Shell 变量不需要专门的定义和初始化语句。一个没有初始化的 Shell 变量被认为是空字符串。通常通过赋值语句完成变量说明并予以赋值，并且可以给一个变量多次赋值以改变其值。

在Shell中，变量的赋值有下列句法格式：

```
name=string
```

其中：

- name ： 是变量名，变量名是以字母或下划线开头的字母、数字和下划线字符序列。用户自定义变量按照惯例使用小写字母命名。
- "=" ：是赋值符号。两边不能直接跟空格，否则**Shell**将视为命令。
- string ： 是被赋予的变量值。若**string**中包含空格、制表符和换行符，则**string**必须用 **'string'** 或 **"string"** 的形式，即用单（双）引号将其括起来。双引号内允许变量替换而单引号则不可以。

例如：

```
$ v1=CentOS
$ v2='CentOS 5'
$ v3="CentOS 5 $HOSTTYPE"
$ v4=1234
```

- 当变量的值仅仅包含数字时才允许进行数值计算。
- 在较新的 bash 中，可是使用 declare 或 typeset 命令声明变量及其属性，但一般不需要声明。而且为了使脚本兼容于不同的 shell，在没有必要的情况下尽量不使用变量声明。

Shell 变量替换和输出

通过在变量名（name）前加$字符，即 $name 引用变量的值，引用的结果就是用字符串 string 代替 $name。此过程也称为变量替换。在字符串连接过程中为了界定变量名、避免混淆，变量替换也可以使用 ${name} 的形式。

变量输出可使用 Shell 的内置命令 echo（常用） 或 printf （用于格式化输出，类似 C 语言的 printf()）。

下面给出一个使用 echo 命令显示字面变量的例子，同时体会一下三种字符串界定符号（双引号、单引号、反引号）的用法。

```
$ echo who am i
who am i
$ echo 'who am i'
who am i
$ echo "who am i"
who am i
# 由于要输出的字符串中没有特殊字符（Shell的保留字），
# 所以""和''的效果一致，不用""相当于使用""
$ echo `who am i`
osmond pts/0 2007-12-09 11:08 (192.168.0.77)
$ echo "`who am i`"
osmond   pts/0        2007-12-09 11:08 (192.168.0.77)
$ who am i
osmond   pts/0        2007-12-09 11:08 (192.168.0.77)
# 若字符串恰好是可执行的命令，被括在``之中，将返回命令的执行结果
# "``" 将保留命令的原始输出，``会将命令结果中的制表符和回车替换为空格
$
$ echo Je t'aime.
> '
$
# 由于要输出的字符串中有特殊字符（'），
# 由于'不匹配，Shell认为命令行没有结束，回车后出现系统第二提示符，
# 让用户继续输入命令行。键入 ‘ 或按Ctrl+c结束
$
# 为了解决这个问题，可以使用下面的两种方法
$ echo "Je t'aime."
Je t'aime.
$ echo Je t\'aime.
Je t'aime.
```

1. 注意反引号"``"和单引号"'"的区别，他们在功能上并不相同。
2. `` 和 $() 等效，即 `who am i` 与 $(who am i) 等效。通常称此为命令替换。

下面给出一个使用 echo 命令显示变量的例子。

```
$ v1=CentOS
$ echo $v1
CentOS
# 中间有空格，用''括起来
$ v2='CentOS 5'
$ echo $v2
CentOS 5
# 要置换 HOSTTYPE 的值使用""括起来
$ v3="CentOS 5 $HOSTTYPE"
$ echo $v3
CentOS 5 i686
# $HOSTTYPE 在双引号内，置换了其值
$
# 同样地，单、双引号规则在字符串连接时也适用
$ echo I love $v1.
I love CentOS.
$ echo "I love $v1."
I love CentOS.
$ echo 'I love $v1.'
I love $v1.
```

```
# 单引号中的内容被原样输出
$ echo "I love \$v1."
# 在双引号中使用转义字符\，转义字符将其后的字符还原为字面本身
I love $v1.
echo "I love \$$v1."
I love $CentOS.
```

下面再给出一个使用 ${} 进行变量置换的例子。

```
# 设置一个动词前缀变量
$ verb_p=parl
$ echo je $verb_pe tu $verb_pes il $verb_pe \
> nous $verb_pons vous $verb_pez ils $verb_pent
je tu il nous vous ils
# 由于没有找到 verb_pe、verb_pes、verb_pons、verb_pez、verb_pent 的值，打印了空串
# 使用 ${verb_p} 置换变量的值
$ echo je ${verb_p}e tu ${verb_p}es il ${verb_p}e \
> nous ${verb_p}ons vous ${verb_p}ez ils ${verb_p}ent
je parle tu parles il parle nous parlons vous parlez ils parlent
# echo 加 -e 参数可以解析 "\n" 等字符
$ echo -e " je ${verb_p}e \n tu ${verb_p}es \n il ${verb_p}e \n \
> nous ${verb_p}ons \n vous ${verb_p}ez \n ils ${verb_p}ent"
 je parle
 tu parles
 il parle
 nous parlons
 vous parlez
 ils parlent
# 可见 "\n" 被解释为换行
```

- 用户还可以使用不带任何参数的 set 、declare、typeset 命令显示当前定义的所有变量（包括环境变量）。
- 要取消一个 Shell 变量的声明或赋值，可以使用 unset 命令，例如下面的命令取消变量 v4 的定义。

```
$ unset v4
```

Shell 变量的作用域

Shell变量有其规定的作用范围。Shell变量分为局部变量和全局变量。所有自定义变量默认都是局部变量；环境变量是全局变量。

- 局部变量的作用范围仅仅限制在其命令行所在的Shell或当前Shell脚本执行过程中；
- 全局变量的作用范围则包括定义该变量的Shell及其所有子Shell。

可以使用 export 内置命令将局部变量设置为全局变量。export 的常用格式为：

```
export <变量名1> [<变量名2> ...]          # 将指定的一个或多个局部变量设置为全局变量
export -n <变量名1> [<变量名2> ...]       # 将指定的一个或多个全局变量设置为局部变量
export <变量名1=值1> [<变量名2=值2> ...]  # 直接对一个或多个全局变量赋值
```

下面给出一个Shell变量作用域的例子。

```
# 1-为 var1 赋值
$ var1=UNIX
# 1-为 var2 赋值
$ var2=Linux
# 1-将变量 var2 的作用范围设置为全局
$ export var2
# 1-直接为全局变量 var3、var4 赋值
$ export var3=centos var4=ubuntu
# 1-在当前Shell中显示四个变量的值
$ echo $var1 $var2 $var3 $var4
UNIX Linux centos ubuntu
```

```
# 1-进入子Shell
$ bash
# 2-显示 var1 的值
# 2-由于var1在上一级Shell中没有被声明为全局，所以在子Shell里没有值
$ echo $var1

# 2-显示 var2、var3、var4 的值
# 2-由于这三个变量在上一级Shell中被声明为全局，所以在子Shell里仍有值
$ echo $var2 $var3 $var4
Linux centos ubuntu
# 2-在当前Shell中将 var2 设置为局部
$ export -n var2
# 2-在当前Shell中 var2 仍有值
$ echo $var2
Linux
# 2-进入孙子Shell
$ bash
# 3-由于 var2 在当前Shell的父Shell中已经设置为局部的，所以在孙子Shell里没有值
# 3-当然，var1 在当前Shell的祖父Shell中就是局部变量，所以在当前Shell里没有值
$ echo $var1 $var2

# 3-由于var3和var4 在当前Shell的祖父Shell中设置为全局，
# 3-在当前Shell的父Shell中又没有变更，所以在当前Shell里仍有值
$ echo $var3 $var4
centos ubuntu
# 3-返回父Shell
$ exit
# 2-显示当前Shell中变量的值
$ echo $var2 $var3 $var4
Linux centos ubuntu
# 2-修改变量 var3 的值
$ var3=centos5.1
# 2-显示变量 var3 的值
$ echo $var3
centos5.1
# 2-返回父Shell
$ exit
# 1-已在父Shell中
$ echo $var1 $var2 $var3 $var4
UNIX Linux centos ubuntu
$
```

由上面的例子可以看出：

- 在当前Shell中要想使用父辈Shell中的变量，至少要在当前Shell 的父Shell中设置为全局
- 变量在子Shell中值的修改不会传回父Shell

环境变量

环境变量定义 Shell 的运行环境，保证 Shell 命令的正确执行。Shell用环境变量来确定查找路径、注册目录、终端类型、终端名称、用户名等。所有环境变量都是全局变量（即可以传递给 **Shell** 的子进程），并可以由用户重新设置。下表列出了一些系统中常用的环境变量。

环境变量名	说明
BASH	Bash 的完整路径名
EDITOR	在应用程序中默认使用的编辑器
ENV	Linux 查找配置文件的路径
HISTFILE	用于储存历史命令的文件
HISTSIZE	历史命令列表的大小
USER	当前用户名
UID	当前用户的 UID
HOME	当前用户的用户目录

OLDPWD	前一个工作目录
PATH	bash寻找可执行文件的搜索路径
PWD	当前工作目录
IFS	Bash 用于分割命令行参数的分隔符
PS1	命令行的一级提示符
PS2	命令行的二级提示符
PPID	父进程的 PID
SECONDS	当前Shell开始后所流逝的秒数
TERM	当前用户的终端类型
LANG	主语言环境

这些是可写的，用户可以为他们赋以任何值。要使用自己的环境变量应该使用上面介绍的 export 命令。

- 用户还可以使用不带任何参数的 env、printenv 或 export 命令显示当前定义的所有环境变量。
- 要取消一个环境变量的声明或赋值，也可以使用 unset 命令。

用户工作环境

用户工作环境概述

用户登录系统时，Shell为用户自动定义唯一的工作环境并对该环境进行维护直至用户注销。该环境将定义如身份、工作场所和正在运行的进程等特性。这些特性由指定的环境变量值定义。

Shell环境与办公环境相似，在办公室中每个人所处环境的物理特性，如灯光和温度相似，但在办公环境中又有许多因素是个人特有的，如日常工作和个人工作空间，因此用户自己的工作环境就有别于其他同事的工作环境。正如用户的Shell环境不同于其他用户的Shell环境。

用户工作环境还有登录环境和非登录环境之分。登录环境是指用户登录系统时的工作环境，此时的Shell对登录用户而言是主Shell。非登录环境是指用户再调用子Shell时所使用的用户环境。

工作环境设置文件

用户并不需要每次登录后都对各种环境变量进行手工设置，通过环境设置文件，用户的工作环境的设置可以在登录的时候自动由系统来完成。环境设置文件有两种，一种是系统环境设置文件，另一种是个人环境设置文件。

1. 系统中的用户工作环境设置文件
   - 登录环境设置文件：/etc/profile
   - 非登录环境设置文件：/etc/bashrc
2. 用户设置的环境设置文件
   - 登录环境设置文件：$HOME/.bash_profile
   - 非登录环境设置文件：$HOME/.bashrc

1. 工作环境设置文件是Shell脚本文件。
2. 系统中的用户工作环境设置文件对所有用户均生效；用户设置的环境设置文件只对用户自身生效。
3. 用户可以修改自己的用户环境设置文件来覆盖在系统环境设置文件中的全局设置。例如：
   I. 用户可以将自定义的环境变量存放在$HOME/.bash_profile中。
   II. 用户可以将自定义的别名存放在$HOME/.bashrc，以便在每次登录和调用子Shell时生效。

- 显示源文件

- 登录