

# 3 Shell变量基础

变量：具有**标识符**的**临时存储空间**。

## 一、Shell变量概述

### 1.1 什么是Shell变量

变量是任何一种编程语言都必不可少的组成部分。

⚠ 脚本语言在定义变量时通常不需要指明类型，直接赋值即可使用，Shell 变量也遵循这个规则，在**声明变量**时并**不需要指定其变量类型**，而且也**不需要遵循C语言中“先声明再使用”的规定**，即使用**未定义**的变量**不会抛出异常**。（与Python非常相似）

⚠ 在 Bash shell 中，变量的值**默认类型**为**字符串**。即使你将整数和小数赋值给变量，它们也会被视为字符串，这一点**和大部分的编程语言不同**。

如果有必要，也可以使用 `declare` 命令显式定义变量的类型，但在一般情况下没有这个需求，在编写代码时注意值的类型即可。

### 1.2 Shell变量的类别

Shell变量分为三类，分别为自定义变量、环境变量和预定义变量。

- 根据工作要求临时定义的变量称为**自定义变量**
- **环境变量**一般是指用 `export` 内置命令导出的变量，用于定义Shell的运行环境，保证Shell命令的正确执行，如 `$0`、`$1`、`$#`
- **预定义变量**是在bash（Linux系统的默认Shell）中已有的变量，可以直接使用，`$#`、`$*`

## 二、自定义变量

自定义变量可以理解为**局部变量**或**普通变量**，**只能在创建它们的Shell函数或Shell脚本中使用**。

## 2.1 定义自定义变量

Shell 支持以下三种定义变量的方式。

- `variable=value`
- `variable='value'`
- `variable="value"`

`variable` 是变量名，`value` 是赋给变量的值。

如果 `value` **不包含**任何**空白符**（例如空格、Tab 缩进等），那么**可以不使用引号**。

如果 `value` **包含了空白符**，那么就必须使用**引号**包围起来。

⚠ 赋值号 `=` 的周围**不能有空格**，这可能和**大部分编程语言都不一样**。

### 案例：定义自定义变量

```
▼ Shell | 复制代码
1 [root@Shell ~]# a=2
2 # 变量赋值时如果有空白符，需要用引号包围起来
3 [root@Shell ~]# a=2 3
4 -bash: 3: command not found
5 [root@Shell ~]# a="2 3"
6 # 赋值时=两边不能有空格
7 [root@Shell ~]# a= 1
8 -bash: 1: command not found
9 [root@Shell ~]# a =1
10 -bash: a: command not found
```

## 2.2 自定义变量命名规则

Shell 变量的**命名规范和大部分编程语言都一样**。

- 区分大小写
- 只能由数字、字母、下划线组成
- 必须以字母或者下划线开头
- 不能使用 Shell 里的关键字（通过 `help` 命令可以查看内置命令列表）
- 变量的长度没有限制

### 案例：自定义变量命名规则

```
1  # 正确的变量命名
2  first
3  First
4  FIRST
5  _first
6  first_month
7  group1
8  # 错误的变量命名
9  1group
10 *group
11 for
```

## 2.3 使用自定义变量

使用一个已定义的变量，只要在**变量名前面加美元符号 \$** 即可，例如：

```
1 [root@Shell ~]# author="yb"
2 [root@Shell ~]# echo $author
3 yb
4 [root@Shell ~]# echo ${author}
5 yb
```

变量名外面的**花括号 { }** 是可选的，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
1 [root@Shell ~]# skill=python
2 [root@Shell ~]# echo "I am good at $skillScript"
3 I am good at
4 [root@Shell ~]# echo "I am good at ${skill}Script"
5 I am good at pythonScript
```

如果不给 `skill` 变量加花括号，解释器就会把 `$skillScript` 当成一个变量（该变量未定义，因此其值为空），代码执行结果就不是我们期望的样子了。

**⚠ 强烈建议给所有变量加上花括号 { }，这是良好的编程习惯。**

⚠ 注意！在Shell中，当**第一次使用某个变量名**时，实际上就**定义了这个变量**。如果没有给出变量值，则变量会被赋值为一个**空字符串**，而不是抛出异常。

Shell | 复制代码

```
1 [root@Shell ~]# echo ${abc}
2
```

💬 例如：Python中调用未定义的变量会抛出异常。

Bash | 复制代码

```
1 [root@shell ~]# python
2 Python 2.7.5 (default, Oct 14 2020, 14:45:30)
3 [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux2
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> a
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8   NameError: name 'a' is not defined
9 >>> quit()
```

💬 Bash可以**设置强制声明变量**，这样就要求脚本必须遵循“**先声明再使用**”的原则，否则使用未声明的变量的情况则立刻**提示错误**。

Shell | 复制代码

```
1 # 练习前请确保abc和ab变量未定义
2 # 设置强制变量声明
3 [root@Shell ~]# shopt -s -o nounset
4 [root@Shell ~]# echo ${abc}
5 -bash: abc: unbound variable
6 # 取消强制变量声明
7 [root@Shell ~]# shopt -u -o nounset
8 [root@Shell ~]# echo ${ab}
9
10 [root@Shell ~]#
```

💬 `shopt` 命令用于显示和设置shell中的行为选项。

## 2.4 修改自定义变量的值

已定义的变量，可以被重新赋值。**赋值方法与定义变量方法相同。**

```
1 [root@Shell ~]# a="abc"
2 [root@Shell ~]# echo ${a}
3 abc
4 [root@Shell ~]# a="abcd"
5 [root@Shell ~]# echo ${a}
6 abcd
```

⚠ 对变量赋值时不能在变量名前加 `$`，只有在使用变量时才能加 `$`。

## 2.5 取消自定义变量

取消变量指的是将变量从内存中释放，格式为 `unset 变量名`。

```
1 [root@Shell ~]# echo ${a}
2 abcd
3 [root@Shell ~]# unset a
4 [root@Shell ~]# echo ${a}
5
6 [root@Shell ~]#
```

## 2.6 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，也不能使用 `unset` 命令取消。**

💬 设置只读变量应当慎重，取消只读变量操作相对比较繁琐。

```
1 [root@Shell ~]# b=123
2 [root@Shell ~]# readonly b
3 [root@Shell ~]# echo ${b}
4 123
5 [root@Shell ~]# b=456
6 -bash: b: readonly variable
7 [root@Shell ~]# unset b
8 -bash: unset: b: cannot unset: readonly variable
```

## 三、环境变量

环境变量也可称为**全局变量**，可以在**创建它们的Shell及其派生出来的任意子进程Shell**中使用。

### 3.1 定义环境变量

使用 **export** 命令将**自定义变量导出**，那么该变量就在**Shell所有的子进程中也有效了**，这称为**环境变量**。

```
1 # 未导出环境变量
2 [root@Shell ~]# a=123
3 [root@Shell ~]# echo ${a}
4 123
5 # bash命令相当于创建了一个shell的子进程
6 [root@Shell ~]# bash
7 [root@Shell ~]# echo ${a}
8
9 # 导出环境变量
10 [root@Shell ~]# a=123
11 [root@Shell ~]# echo ${a}
12 123
13 [root@Shell ~]# export a
14 [root@Shell ~]# bash
15 [root@Shell ~]# echo ${a}
16 123
```

在**bash**中再运行 **bash** 命令会派生**bash**子进程。**exit** 命令可以返回主进程。

 请在课下验证**自定义变量**和**环境变量**在子进程Shell中是否生效。

验证bash自定义变量作用范围

Shell

 复制代码


```
1 [root@shell ~]# a=1
2 [root@shell ~]# echo ${a}
3 1
4 # 启动bash子进程
5 [root@shell ~]# bash
6 [root@shell ~]# echo ${a}
7 # 退出bash子进程
8 [root@shell ~]# exit
9 exit
10 [root@shell ~]# echo ${a}
11 1
```

## 3.2 操作环境变量

修改、使用、取消环境变量的方法与**自定义变量**相同。


▼

Shell

 复制代码

```
1 # 导出环境变量
2 [root@Shell ~]# export author="yb"
3 # 使用环境变量
4 [root@Shell ~]# echo ${author}
5 yb
6 # 修改环境变量
7 [root@Shell ~]# author=tom
8 [root@Shell ~]# echo ${author}
9 tom
10 #取消环境变量
11 [root@Shell ~]# unset author
12 [root@Shell ~]# echo ${author}
13
```

## 3.3 常用内置环境变量

 Bash中**默认**包含有一些**内置环境变量**，可以使用 `man bash` 查看man文件，在 `Shell Variables` 章节中可具体查看每个变量的含义。

常用的内置环境变量如下：

- `PATH` : 系统路径, 决定了Shell将到哪些目录中寻找命令或程序
- `HOME` : 当前用户主目录
- `UID` : 当前用户的UID (用户标识), 相当于 `id -u`
- `PWD` : 当前工作目录的绝对路径名
- `HOSTNAME` : 是指主机的名称, 许多应用程序如果要用到主机名的话, 通常是从这个环境变量中取得的
- `LOGNAME` : 是指当前用户的登录名
- `USER` : 当前用户
- `SHELL` : 当前SHELL
- `MAIL` : 是指当前用户的邮件存放目录
- `HISTSIZE` : 是指保存历史命令记录的条数
- `LANG/LANGUGE` : 是和语言相关的环境变量, 使用多种语言的用户可以修改此环境变量
- `PS1` : 是基本提示符, 对于root用户是#, 对于普通用户是\$
- `PS2` : 是附属提示符, 默认是“>”可以通过修改此环境变量来修改当前的命令符, 比如下列命令会将提示符修改成字符串“Hello,My NewPrompt :)”

可以使用 `set` 命令和 `env` 命令查看当前系统的**已定义的环境变量**。

- `set` 命令显示当前环境下所有变量 (自定义变量、环境变量)
- `env` 命令显示当前用户的环境变量。由于输出内容较多, 不再演示。

通过观察可知, **环境变量命名通常要大写**。



```
1 ▾ [root@Shell ~]# echo $PATH
2 /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
3 ▾ [root@Shell ~]# echo $HOME
4 /root
5 ▾ [root@Shell ~]# echo $USER
6 root
7 ▾ [root@Shell ~]# echo $LOGNAME
8 root
9 ▾ [root@Shell ~]# echo $UID
10 0
11 ▾ [root@Shell ~]# echo $HOSTNAME
12 Shell
13 ▾ [root@Shell ~]# echo $PWD
14 /root
```

### 3.4 通过配置文件设置永久环境变量

除了系统内置环境变量，其他环境变量的作用范围是临时的，一旦退出shell变量即失效。

如果想永久添加环境变量，常用的方法就是在Bash shell配置文件中定义环境变量。在登录Linux系统并启动一个Bash shell时，默认情况下Bash会在若干个文件中查找环境变量的设置，这些文件可统称为系统环境文件。

#### 3.4.1 Bsh Shell 的配置文件

Bash shell的配置文件主要有以下两类。

- 全局配置文件：影响所有用户
  - /etc/profile
  - /etc/profile.d/\*.sh
  - /etc/bashrc
- 个人配置文件：影响当前用户
  - ~/.bash\_profile
  - ~/.bashrc

由上可知，Bash 的配置文件分别为profile文件和bashrc文件：

- profile 类文件的功能。
  - 设定环境变量

- 运行命令或脚本（登录时运行的脚本）
- bashrc 类文件配置的功能。
  - 设定本地变量
  - 定义命令别名

### 3.4.2 Bash Shell 配置文件的读取流程（扩展内容）

Bash Shell检查的环境变量文件的情况取决于系统运行Shell的方式。

系统运行Shell的方式一般分为登录式Shell和非登录式Shell。

- 登录式 shell 读取配置 文件过程：

```
/etc/profile -> /etc/profile.d/*.sh -> ~/.bash_profile -> ~/.bashrc -> /etc/bashrc
```

- 非登录式 shell 读取配置 文件过程：

```
~/.bashrc -> /etc/bashrc -> /etc/profile.d/*.sh
```

#### 1. 登录式Shell

用户登录系统后首先会加载 `/etc/profile` 全局环境变量文件，**这是Linux系统上默认的Shell主环境变量文件**。系统上每个用户登录都会加载这个文件。有关重要的环境变量都定义在此，其中包括 `PATH`、`USER`、`LOGNAME`、`MAIL` 等。

当加载完 `/etc/profile` 文件后，会执行 `/etc/profile.d` 目录下的脚本文件，这个目录下的脚本文件有很多，例如：系统的字符集设置

（`/etc/sysconfig/i18n`）等。


之后开始运行 `~/.bash_profile`（用户环境变量文件），在这个文件中，又会去找 `~/.bashrc`（用户环境变量文件），如果有，则执行，如果没有，则不执行。在 `~/.bashrc` 文件中又会去找 `/etc/bashrc`（全局环境变量文件），如果有，则执行，如果没有，则不执行。

#### 2. 非登录式Shell

如果用户的Shell不是登录时启动的，那么这种非登录Shell只会加载 `~/.bashrc`（用户环境变量文件），并会去找 `/etc/bashrc`（全局环境变量文件）。

因此如果希望非登录Shell下也可读到的环境变量等内容，就需要将变量设定等写入 `~/.bashrc` 或者 `/etc/bashrc`，而不是 `~/.bash_profile` 或 `/etc/profile`。

le。

 通过查看上述配置文件了解配置文件的工作原理。课后请查询资料了解配置文件的详细解读。

补充：登录式 shell 和非登录式 shell 的运行形式如下：

- 登录式 shell：
  - 正常通过某终端登录的 shell。
  - `su - username`。
  - `su -l username`。
- 非登录式 shell：
  - `su username`。
  - 图形终端下打开的命令窗口。
  - 自动执行的 shell 脚本。

### 3.4.3 案例：永久添加JAVA环境变量

Shell | 复制代码

```
1 [root@Shell ~]# vi /etc/profile
2 JAVA_HOME=/usr/local/java
3 PATH=$JAVA_HOME/bin:$PATH
4 export JAVA_HOME PATH
5 "/etc/profile" 80L, 1825C written
6 [root@Shell ~]# echo $JAVA_HOME
7
8 # 必须重新执行/etc/profile后环境变量才生效
9 [root@Shell ~]# . /etc/profile
10 [root@Shell ~]# echo $JAVA_HOME
11 /usr/local/java
```

首先自定义 `JAVA_HOME` 变量，然后使用 `export` 将 `JAVA_HOME` 、 `PATH` 自定义变量转换为环境变量写在 `PATH` 中。**最后必须重新执行/etc/profile后环境变量才生效**。这样用户执行 `JAVA` 命令或使用 `JAVA` 环境时，系统将自动识别环境变量。

## 四、预定义变量

在Shell中还有一些**预先定义**的特殊变量，它们的**值只有在脚本运行时才能确定**。

预定义变量	说明
\$n	脚本参数，\$0表示脚本名，\$1-\$9 代表接收的第1~9个参数，\$10以上需要用{}括起来。
\$*	所有的参数
\$@	所有的参数
\$#	参数的个数
\$\$	当前进程的PID
\$_	上一个后台进程的PID
\$?	上一个命令的返回值 0表示成功

## 4.1 位置变量

`$n` 也被称为**位置变量**，用于在命令行、函数或脚本中**传递参数**，其**变量名不用自己定义，其作用也是固定的**。

`$0` 代表命令本身，`$1 - $9` 代表接收的第1~9个参数，`$10` 以上需要用 `{}` 括起来，如 `${10}` 代表接收的第10个参数。

Shell | 复制代码

```

1 ▾ [root@Shell ~]# vi test.sh
2  echo $0 $1 $2
3  echo $* $#
4 ▾ [root@Shell ~]# source test.sh a b
5  -bash a b
6  a b 2
7 ▾ [root@Shell ~]# bash test.sh a b
8  test.sh a b
9  a b 2
10 ▾ [root@Shell ~]# chmod +x test.sh
11 ▾ [root@Shell ~]# ./test.sh a b
12  ./test.sh a b
13  a b 2

```

## 4.2 脚本或命令返回值

`$?` 表示脚本或命令的**返回值**。

**⚠ 正常**退出的命令和脚本应该以**0**作为其返回值，任何**非0**的返回值都表示**命令未正确退出或未正常执行**。

Shell | 复制代码

```
1 [root@Shell ~]# pstree
2 bash: pstree: command not found
3 [root@Shell ~]# echo $?
4 127
5 [root@Shell ~]# echo 1
6 1
7 [root@Shell ~]# echo $?
8 0
```

## 小结

1. 变量概述：变量的类别
2. 自定义变量：=, \$
3. 环境变量：export
4. 预定义变量：\$n,\$?

## 课程目标

- 知识目标：了解Shell变量的概念，掌握Shell变量的基本操作。
- 技能目标：能够根据要求定义、使用、取消Shell变量。

## 课外拓展

- 进一步了解Shell变量的基础知识

## 参考资料

- bash帮助： `man bash`