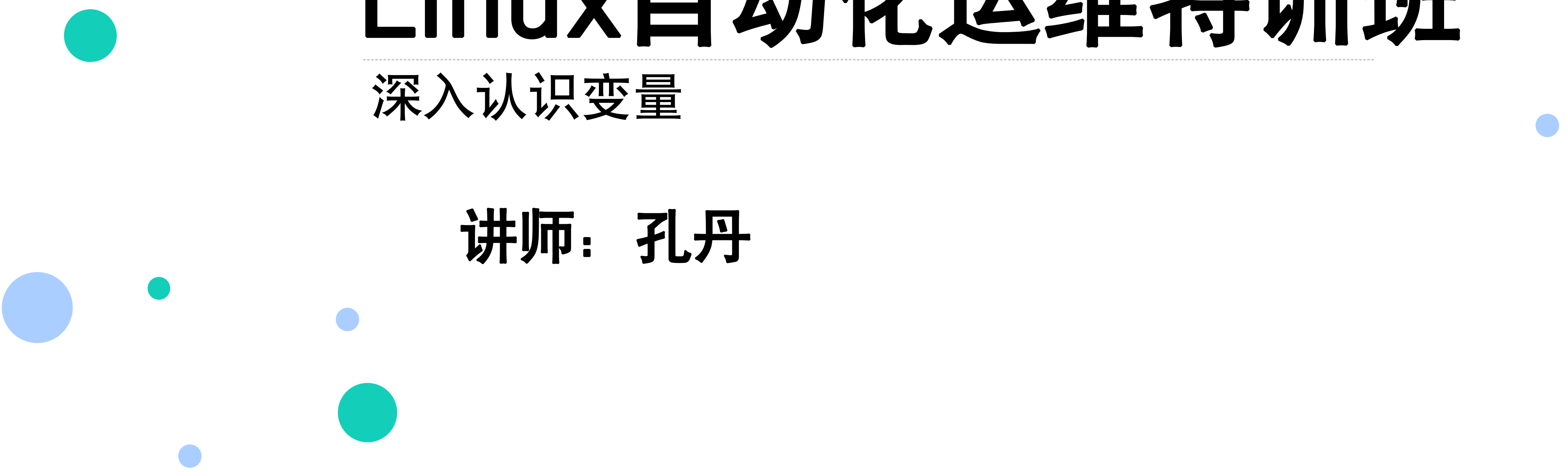




Linux自动化运维特训班

深入认识变量

讲师：孔丹



大纲

- 深入认识变量
- 变量的赋值和替换
- 变量和引用
- 变量进阶

深入认识变量

- 在程序设计语言中，变量是一个非常重要的概念。也是初学者在进行Shell程序设计之前必须掌握的一个非常基础的概念。只有理解变量的使用方法，才能设计出良好的程序。本节将介绍Shell中变量的相关知识。

什么是变量

- 顾名思义，变量就是程序设计语言中的一个可以变化的量，当然，可以变化的是变量的值。变量几乎所有的程序设计语言中都有定义，并且其涵义也大同小异。从本质上讲，变量就是在程序中保存用户数据的一块内存空间，而变量名就是这块内存空间的地址。
- 在程序的执行过程中，保存数据的内存空间的内容可能会不断地发生变化，但是，代表内存地址的变量名却保持不变。

变量的类型

- Shell是一种动态类型语言和弱类型语言，即在Shell中，变量的数据类型毋需显示地声明，变量的数据类型会根据不同的操作有所变化。准确地讲，Shell中的变量是不分数据类型的，统一地按照字符串存储。但是根据变量的上下文环境，允许程序执行一些不同的操作，例如字符串的比较和整数的加减等等

定义变量

- 在定义变量时，有一些规则需要遵守：
 - ✓ 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。如果变量名是“2name”则是错误的。
 - ✓ 在 Bash 中，变量的默认类型都是字符串型，如果要进行数值运算，则必修指定变量类型为数值型。
 - ✓ 变量用等号连接值，等号左右两侧不能有空格。
 - ✓ 变量的值如果有空格，需要使用单引号或双引号包括。如：“test=“hello world!””。其中双引号括起来的内容“\$”、“\”和反引号都拥有特殊含义，而单引号括起来的内容都是普通字符。
 - ✓ 在变量的值中，可以使用“\”转义符。
 - ✓ 如果需要增加变量的值，那么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}包含变量名。

变量的类型

➤ 1. 自定义变量

这种变量是最常见的变量，由用户自由定义变量名和变量的值

定义变量：变量名=变量值 变量名必须以字母或下划线开头，区分大小

写 ip1=192.168.2.115

引用变量：\$变量名 或 \${变量名}

查看变量：echo \$变量名 set(所有变量：包括自定义变量和环境变量)

取消变量：unset 变量名

作用范围：仅在当前shell中有效

变量的类型

➤2. 环境变量

这种变量中主要保存的是和系统操作环境相关的数据，比如当前登录用户，用户的家目录，命令的提示符等。

定义环境变量：

方法一 export back_dir2=/home/backup

方法二 export back_dir1 将自定义变量转换成环境变量

引用环境变量：\$变量名 或 \${变量名}

查看环境变量：echo \$变量名 env 例如env |grep

back_dir2

取消环境变量：unset 变量名

变量作用范围： 在当前shell和子shell有效

=====			
C语言	局部变量	vs	全局变量
SHELL	自定义变量	vs	环境变量
=====			

变量的类型

➤3. 位置变量

这种变量主要是用来向脚本当中传递参数或数据的，变量名不能自定义，变量作用是固定的。

\$1 \$2 \$3 \$4 \$5 \$6 \$7 \$8 \$9 \${10}

➤4. 预定义变量

是 Bash 中已经定义好的变量，变量名不能自定义，变量作用也是固定的。

\$0 脚本名

\$* 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

变量的类型

□ 示例1：位置参数和预定义变量

```
# vim test.sh
```

```
echo "第2个位置参数是$2"
```

```
echo "第1个位置参数是$1"
```

```
echo "第4个位置参数是$4"
```

```
echo "所有参数是: $*"
```

```
echo "所有参数是: $@"
```

```
echo "参数的个数是: $#"
```

```
echo "当前进程的PID是: $$"
```

```
echo '$1='$1
```

```
echo '$2='$2
```

```
echo '$3='$3
```

```
echo '$*='$*
```

```
echo '$@='$@
```

```
echo '$#='$#
```

```
echo '$$='$$
```

变量的类型

□ 了解\$*和\$@区别

\$* 与\$@ 区别：仅仅在加双引号时区别

\$* 获取当前shell的所有参数，将所有命令行参数视为单个字符串，相当于"\$1\$2\$3"

\$@ 将所有参数以"\$1" "\$2" "\$3" 形式，每个参数作为个体

□ 示例2:

```
# vim ping.sh
```

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

```
ping -c2 $1 &>/dev/null
```

```
if [ $? = 0 ];then
```

```
    echo "host $1 is ok"
```

```
else
```

```
    echo "host $1 is fail"
```

```
fi
```

```
# chmod a+x ping.sh
```

```
# ./ping.sh 192.168.2.25
```

变量的类型

□示例3： 设置java环境变量

#配置环境变量/etc/profile

export JAVA_HOME=/usr/java/jdk1.6.0_45

export

CLASSPATH=.:\$JAVA_HOME/lib/dt.jar:\$JAVA_HOME/lib/tools.j

ar

export PATH=\$PATH:\$JAVA_HOME/bin

#使配置生效 #source /etc/profile

#检测： # java -version

java version "1.6.0_45"

Java(TM) SE Runtime Environment (build 1.6.0_45-b06)

Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed
mode

#也可以脚本形式（比如java.sh）定义在/etc/profile.d/下面

变量的赋值

□ 1. 显式赋值

变量名=变量值

示例:

ip1=192.168.1.251

school="BeiJing uplooking"

today1=`date +%F`

today2=\$(date +%F)

□ 2. read 从键盘读入变量值

read 变量名

read -p "提示信息: " 变量名

read -t 5 -p "提示信息: " 变量名

read -n 2 变量名

read命令行中也可以不指定变量.如果不指定变量, 那么read命令会将接收到的数据放置在环境变量REPLY中。

变量和引号

- Shell语言中一共有3种引号，分别为
 - ✓ 单引号 (' ') 单引号括起来的字符都作为普通字符出现
 - ✓ 双引号 (" ") 双引号括起来的字符，除 "\$" 、 "\" 、 "'" 和 "" 这几个字符仍是特殊字符并保留其特殊功能外，其余字符仍作为普通字符对待，
 - ✓ 反引号 (` `) 。反引号括起来的字符串被Shell解释为命令，在执行时，Shell首先执行该命令，并以它的标准输出结果取代整个反引号（包括两个反引号）部分

符号	说明
双引号	除美元符号、单引号、反引号和反斜线之外，其他所有的字符都将保持字面意义
单引号	所有的字符都将保持字面意义
反引号	反引号中的字符串将被解释为Shell命令
反斜线	转义字符，屏蔽后的字符的特殊意义

只读变量

□ 只读变量

将变量配置成为 readonly 类型，该变量不可被更改内容，也不能 unset

定义方法：

方法一：

readonly [-fap] [变量定义]

-f 定义只读函数

-a 定义只读数组变量

-p 显示系统中全部的变量列表

方法二：

declare -r 变量定义

变量的运算

Shell 中常见的算术运算符号

算术运算符	意义 (* 表示常用)
+, -	加法 (或正号)、减法 (或负号) *
*, /、%	乘法、除法、取余 (取模) *
**	幂运算 *
++, --	增加及减少, 可前置也可放在变量结尾 *
!, &&、	逻辑非 (取反)、逻辑与 (and)、逻辑或 (or) *
<, <=, >, >=	比较符号 (小于、小于等于、大于、大于等于)
==, !=, =	比较符号 (相等、不相等, 对于字符串 “=” 也可以表示相当于) *
<<, >>	向左移位、向右移位
~, , &, ^	按位取反、按位异或、按位与、按位或
=, +=, -=, *=, /=, %=	赋值运算符, 例如 a+=1 相当于 a=a+1, a-=1 相当于 a=a-1 *

变量的运算

shell中常见的算术运算指令

运输操作符与运算指令	意义
(())	用于整数运算，效率很高
let	用于整数运算，类似于(())
expr	可用于整数运算，但还有其他功能
bc	Linux计算器程序
[\$]	用于整数运算
awk	既可作整数运算，也可做小数运算
declare	定义整数变量值和属性，-i可做运算

示例：

```
r=$((2+5*8))
r=`expr 4 + 2`
r=${1+2}
declare -i r=2+3
let r=3+4
# echo "2+5*8" | bc
42
# awk 'BEGIN {print 2+5*8}'
42
```

变量的进阶

取字符串

规则1: `${变量名:位置起点}`

含义: 由指定的位置起点开始, 截取子字符串到字符串结束, 起点由0开始

规则2: `${变量名:位置起点:长度}`

含义: 由指定的位置起点开始, 截取指定长度的字符串, 起点由0开始

计算子串长度

规则: `${#变量名称}`

含义: 表示返回变量名称的字符串长度

变量的进阶

对比样式:

`${变量#关键词}` 最短匹配, 从左到右

`${变量##关键词}` 最长匹配, 从左到右

`${变量%关键词}`

`${变量%%关键词}`

变量替换:

`${var:-word}`: 表示如果 `var` 已经被赋值, 则取它的值, 否则取 `word` 的值, 但 `var` 不改变。

`${var:=word}`: 表示如果 `var` 已经被赋值, 则取它的值, 否则取 `word` 的值, 同时将 `word` 赋给 `var`。

`${var:+word}`: 表示如果 `var` 已经被赋值, 则取它的值, 否则 `var` 变量置为空。

`${变量/旧字符串/新字符串}`

`${变量//旧字符串/新字符串}`

变量的进阶

示例:

返回变量长度

```
[root@localhost ~]# str1="hello world"
```

```
[root@localhost ~]# echo ${#str1}
```

变量截取

指定起始位置，一直到结束

```
[root@localhost ~]# echo ${str1:1}
```

```
ello world
```

指定长度，不指定起始位置默认从开头开始

```
[root@localhost ~]# echo ${str1::3}
```

```
hel
```

指定起始位置和长度

```
[root@localhost ~]# echo ${str1:1:3}
```

```
ell
```

从右边第几个字符开始，及字符的个数

```
[root@localhost ~]# echo ${str1:0-1:1}
```

```
d
```

输出右边的几个字符

```
[root@localhost ~]# echo ${str1:0-5}
```

```
world
```

```
[root@localhost ~]# echo ${str1: -5}
```

```
world
```

注意：下面写法会输出全部，以\${parameter:-default}方式,默认是提取完整地字符串

```
[root@localhost ~]# echo ${str1:-5}
```

```
hello world
```


变量的进阶

示例:

删除字符串:

// 获取后缀名tar.gz

```
[root@localhost ~]# filename=testfile.tar.gz
```

```
[root@localhost ~]# file=${filename#*.}
```

```
[root@localhost ~]# echo $file
```

tar.gz

// 获取后缀名gz

```
[root@localhost ~]# filename=testfile.tar.gz
```

```
[root@localhost ~]# file=${filename##*.}
```

```
[root@localhost ~]# echo $file
```

gz

//截取testfile.tar

```
[root@localhost ~]# filename=testfile.tar.gz
```

```
[root@localhost ~]# file=${filename%.*}
```

```
[root@localhost ~]# echo $file
```

testfile.tar

//截取testfile

```
[root@localhost ~]# filename=testfile.tar.gz
```

```
[root@localhost ~]# file=${filename%%.*}
```

```
[root@localhost ~]# echo $file
```

testfile

变量的进阶

示例:

-示例: 用法1, 如果parameter变量值为空或未赋值, 则返回word字符串代替变量的值
//变量未赋值, 输出为空。

```
[root@localhost ~]# echo $var1
```

```
[root@localhost ~]# var2=${var1:-hello}
```

```
[root@localhost ~]# echo $var2
```

```
hello
```

-示例: 用法2, 如果parameter变量值有值, 则变量返回parameter变量的值

```
[root@localhost ~]# var1="hello"
```

```
[root@localhost ~]# echo $var1
```

```
hello
```

```
[root@localhost ~]# var2=${var1:-world}
```

```
[root@localhost ~]# echo $var2
```

```
hello
```


总结

□ 变量定义

□ 变量类型

□ 变量引用

□ 变量进阶

作业

- 1.Shell脚本中, \$0 \$1 \$\$ \$* \$?分别代表了什么意思
- 2.通过一条命令查找test用户调用的php进程, 并将这些进程强制关闭



谢谢观看

更多好课，请关注[万门大学APP](#)

