学习shell

书籍视频：《跟老男孩学Linux运维：Shell编程实战》

# 一、 Shell编程基础

前期需要掌握的基础知识：

1. vi/vim编辑器的命令，.vimrc设置等等搞清楚
2. 命令基础，100多个命令
3. 基础，高端的网络服务都要会。nfs，rsync，inotify，lanmp，sersync，ssh key批量分发管理。

**主要内容**

（1）变量

本地变量、环境变量、位置参量

（2）输入输出

read / echo或printf

（3）条件测试

整数测试、逻辑测试、字符串测试

（4）控制语句

条件/循环/分支/

（5）函数

## 1、什么叫Shell

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（外部命令）、shell命令（内部命令）、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

## 2、shell第一行

一个规范的Shell脚本的第一行必须是如下格式：

#!/bin/bash

或

#!/bin/sh #<==255个字符以内。

符号 #! 字符又称为幻数，用来指定该脚本文件的解析程序。

注意：这一行必须位于每个脚本顶端的第一行，如果不是第一行则为脚本注释。

如果在脚本开头的第一行不指定解释器，那么就要用对应的解释器来执行脚本：

如果是Shell脚本，就用bash test.sh执行test.sh。

如果是Python脚本，就用python test.sh执行test.py。

如果是expect脚本，就用expect test.exp执行test.exp。

## 3、注释

跟在#后面的内容标识注释。

注释可自成一行，也可跟在命令后与命令同一行。

## 4、基本执行流程

**Centos：**

当shell脚本运行时，它会先查找系统环境变量ENV，该变量指定了环境文件（加载顺序通常是/etc/profile、~/.bash\_profile、~/.bashrc、（家目录下，为用户环境变量）/etc/bashrc），在加载了上述环境变量文件后，shell就开始执行shell脚本中的内容。

只要在/etc/profile、/etc/profile.d/这几个文件或目录下中定义的环境变量，就是全局的，在脚本就能直接取到。如果在该用户下执行脚本，当然首先是在该用户下读取家目录中的全局环境变量。

Shell脚本是**从上至下、从左至右一次执行每一行的命令及语句**，即执行完了一个命令后再执行下一个，如果在shell脚本中遇到子脚本（即**脚本嵌套）时，就会先执行子脚本的内容，完成后再返回父脚本继续执行脚本内后续的命令及语句。**

通常情况下，在执行shell脚本时，会**向系统内核请求启动一个新的进程**，以便在该进程中执行脚本的命令及子shell脚本，如图：



## 5、三种执行方式

Shell脚本的执行通常可以采用以下三种方式：

1. bash script-name或sh script-name（推荐使用的方式）
2. path/script-name或./script-name（当前路径下执行脚本）

需要先将脚本文件的权限修改为可执行（chmod +x script-name），再在脚本绝对路径或相对路径下执行。

1. source script-name或. script-name（”**.**”后有空格）

使用source或“.”（点号）读入或加载指定的shell脚本文件script-name，然后，依次执行script-name中的所有语句。**这些语句将在当前父shell脚本father.sh进程中运行**（**其他几种模式都会启动新的进程执行子脚本**）。因此，使用source或“.”可以将script-name自身脚本中的变量值或函数等的返回值或函数等的返回值传递到当前父shell脚本father.sh中使用。这是它和其他几种方法的最大区别。

source或“.”命令的功能是：在当前shell中执行source或“.”加载并执行的相关脚本文件中的命令及语句，而**不是**产生一个子shell来执行文件中的命令。

例：source

czha@czha-virtual-machine:~/test\_shell$ cat 3

usercurdir='pwd'

czha@czha-virtual-machine:~/test\_shell$ source 3

czha@czha-virtual-machine:~/test\_shell$ pwd

/home/czha/test\_shell

czha@czha-virtual-machine:~/test\_shell$ $usercurdir

/home/czha/test\_shell

例：已经如下命令及返回结果，请问echo $user的返回的结果为（）

czha@czha-virtual-machine:~$ echo "user='whoami'" > test.sh

czha@czha-virtual-machine:~$ cat test.sh

user='whoami'

czha@czha-virtual-machine:~$ sh test.sh

参考的选择项如下：

1. 当前用户
2. czha
3. 空（无内容输出）

*czha@czha-virtual-machine:~$*

如果执行以下命令，则有：

czha@czha-virtual-machine:~$ source test.sh

czha@czha-virtual-machine:~$ $user

czha

通过以上面试题得出如下的结论：

1. 儿子shell脚本会直接继承父亲shell脚本的变量、函数（就好像是儿子随父亲姓，基因也会继承父亲的）等，反之则不可以。
2. 如果希望反过来继承（就好像是让父亲随儿子姓，让父亲的基因也继承儿子的），就要用source或者“. ”在父亲shell脚本中事先加载儿子shell脚本。

## 6、创建方式

Shell：单条命令（交互式）和shell脚本（编程）。

## 7、Shell脚本开发规范

1）开头（第一行）指定脚本解释器，通常为：

#!/bin/bash 或 #！bin/sh

2）开头加版本版权信息

#Date: 21:30 2013-07-20

#Author: Created by xxx

#Mail: xxx@xxx.com

#Function: This scripts function is ......

# Version: GNU bash, version 3.2.25(1)-release

可配置vim编辑文件时自动加上以上信息的功能，方法是可修改“~/,vimrc”配置文件。

3）脚本中不用中文注释

尽量用英文注释，防止本机或切换系统环境后中文乱码的困扰。

4）脚本以.sh为扩展名

例如：script-name.sh

5）shell脚本应存放在固定的路径下

例如：/server/scripts

**代码书写良好习惯**

1. 成对的符号应尽量一次性写出来，然后退格在符号里增加内容，以防止遗漏。这些成对的符号包括：

**{ }**、**[ ]**、**‘ ’**、**` `**、**” ”**

1. 中括号（[]）两端至少要有一个空格，因此，键入中括号时即可留出[ ],然后再退格键入中间的内容，并确保两端都至少有一个空格，即先键入一对中括号，然后再退1格，输入2个空格，再退1格，双中括号（[ [ ] ]）的写法也是如此。

例如: [ abcde ]

1. 对于流程控制语句，应一次性将格式写完，再添加内容。

比如，一次性完成if语句的格式，应为：

if 条件内容

then

内容

fi

一次性完成for循环语句的格式，应为：

for

do

内容

done

1. 使用**缩进**让代码更易读

if 条件内容

then

内容

fi

1. 对于常规变量的字符串定义变量值应加双引号，并且等号前后不能有空格，需要强引用的（指所见即所得的字符引用），则用单引号（**’’**），如果是命令的引用，则用反引号（``）。例如：

OLDBOTY\_FILE=“test.txt”

1. 脚本中的单引号、双引号及反引号必须为应英文状态下的符号，其实所有的Linux字符及符号都应该是英文状态下的符号，这点需要特别注意

**良好的习惯可以让我们避免很多不必要的麻烦，提升很多的工作效率！**

## 8、内部命令和外部命令

echo 等是shell的内部命令，ps、cp等是shell外部命令（linux命令），用type命令即可区分。

用法：type 命令

**示例：**

[czha@localhost sys]$ type cp

cp 是 /usr/bin/cp

[czha@localhost sys]$ type echo

echo 是 shell 内嵌

[czha@localhost sys]$ type printf

printf 是 shell 内嵌

**常用shell程序内置指令**

declare / export / trap等

# 二、变量概述

## 1、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。

例如：oldboy、oldboy123、oldboy\_training

## 2、变量类型

### （1）环境变量和局部变量

* 根据变量的作用域，变量可以分别本地变量和环境变量
* **环境变量**

也称为全局变量，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，环境变量又可分为自定义环境变量和bash内置的环境变量。

* **本地变量**

也称为局部变量，只能在创建它们的shell函数或shell脚本中使用。普通变量一般由开发者在开发脚本程序时创建。

### （2）环境变量说明

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**。

Shell通过环境变量来确定 登录用户名、命令路径、终端类型、登录目录等，所有的环境变量都是系统全局变量，可用于所有子进程中，这包括编辑器、shell脚本和各类应用。

环境变量可以在命令行中设置和创建，但用户退出命令行时这些变量值就会丢失，因此，如果希望永久保存环境变量，可在用户家目录的.bash\_profile或,bashrc（非用户登录模式特有，例如远程SSH）文件中，或者全局配置/etc/bashrc（非用户登录模式特有，例如远程SSH）、/etc/profile或者/etc/profile.d/文件中定义。在将环境变量放入上述文件后，**每次用户登录时这些变量都将被初始化。**

按照系统规范，所有**环境变量的名字均采用大写形式（虽然不是必须的，规范）**。在将环境变量应用于用户进程程序之前，**都应该用export命令导出定义**，例如：正确的环境变量定义方法为export OLDGIRL=1。

例如：查看/ect/profile脚本文件

……

HOSTNAME=`/usr/bin/hostname 2>/dev/null`

HISTSIZE=1000

if [ "$HISTCONTROL" = "ignorespace" ] ; then

export HISTCONTROL=ignoreboth

else

export HISTCONTROL=ignoredups

fi

***#用export导出 可以一起导，也可以一个个***

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL

……

有一些环境变量，比如HOME、PATH、SHELL、UID、USER等，在用户登录之前就已经被/bin/login程序设置好了。通常环境变量被定义并保存在用户家目录下的.bash\_profile文件或全局的配置文件/etc/profile中。

例如：

[czha@localhost /]$ echo $PS1

[\u@\h \W]\$

u: 表示user用户

h: hostname主机名

W：路径

如果想让这个显示调整，可以设置

### （3）环境变量与局部变量的继承

**子进程仅会继承父进程的环境变量，不会继承父进程的自定义变量（普通变量）。**

被export后的变量，我们可以称它为“环境变量”。环境变量可以被子进程所引用，但是其他的自定义变量内容就不会存在于子进程中。

例如：

[root@localhost scripts]# cat m2.sh

echo $OLDBOY

echo $TEST

[root@localhost scripts]# export TEST="命令行中export是否会继承"

[root@localhost scripts]# ./m2.sh

I am oldboy.

命令行中export是否会继承

[root@localhost scripts]#

从上可以看出，执行m2.sh脚本后，虽开启了新的进程，但是export后的TEST变量，还是继承到了m2.sh脚本中。

如果取消export的定义，仅仅将TEST定义成局部变量，则TEST不会输出，如下：

[root@localhost scripts]# unset TEST

[root@localhost scripts]# echo $TEST

[root@localhost scripts]# TEST="命令行中export是否会继承"

[root@localhost scripts]# ./m2.sh

I am oldboy.

[root@localhost scripts]#

为什么环境变量的数据可以被子进程所引用呢？这是因为内存配置的关系。理论上是这样的：

* 当启动一个shell，操作系统会分配一记忆块给shell使用，此内存内的变量可让子进程取用；
* 若在父进程利用export功能，可以让自定义变量的内容写到上述的记忆块当中（环境变量）；
* 当加载另一shell时（即启动子进程，而离开原本的父进程了），子shell可以将父shell的环境变量所在的的记忆块导入自己的环境变量当中。

## 2、变量赋值

varible=value

* 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。

### （1）环境变量自定义

export命令 和 带-x选项的declare内置命令

1. export 变量名=value
2. 变量名=value; export 变量名
3. declare -x 变量名=value

### （2）普通变量的定义

为定义的普通变量赋值，一般有以下三种写法：

1） 变量名=value #<==赋值时不加引号

2） 变量名=‘value’ #<==赋值时加单引号

3） 变量名=“value” #<==赋值时加双引号

**变量的输出**

**$变量名**，表示输出变量，可以用**$c和${c}**两种方式。

### （3）举例

（1）环境变量测试

[czha@localhost /]$ su root

Password:

[root@localhost /]# vim etc/profile

[root@localhost /]# tail -l /etc/profile

fi

done

#just for test

export OLDBOY="I am oldboy."

unset i

unset -f pathmunge

[root@localhost /]# echo $OLEBOY

[root@localhost /]# . /etc/profile （.后有空格 == source）

[root@localhost /]# echo $OLDBOY

I am oldboy.

[root@localhost /]# mkdir server

[root@localhost /]# mkdir /server/scripts

[root@localhost /]# cd server/scripts/

[root@localhost scripts]# vi m2.sh

[root@localhost scripts]# bash m2.sh

I am oldboy.

[root@localhost scripts]# sh m2.sh

I am oldboy.

[root@localhost scripts]# cat m2.sh

echo $oldboy

### （4）普通变量定义的基本技巧总结

a=192.168.1.2 #无引号

b='192.168.1.2' #单引号

c="192.168.1.2" #双引号

变量的内容可以用单引号或双引号引起来，也可不加引号，但是这三者的含义是不同的。

1. **不加任何引号直接定义变量的内容**

例如：a=192.168.1.2-$a

定义变量a的方式是不加任何引号直接定义变量的内容，当内容为**简单连续的数字、字符串、路径名时**，可以这样用，例如：a=1, b=oldboy等。不加引号时，值里有变量的会被**解析后再输出**，上述变量定义中因为$a的值被解析为192.168.1.2，因此，新的a值就是192.168.1.2-192.168.1.2。

1. **单引号方式定义**

例如: b=’ 192.168.1.2-$a’

定义变量b的方式是通过单引号定义。这种定义方式的特点是：输出变量内容时，单引号里是什么就输出什么，即使内容中有变量和命令（命令需要反引起来）也会把它们原样输出（**不解析**）。这种方式比较适合于定义显示纯字符串的情况，即不希望解析变量、命令等的场景。因此，对于这里变量b的值，定义时看到的是什么就输出什么，即192.168.1.2-$a。

1. **双引号方式定义**

例如: c=”192.168.1.2-$a”

定义变量c的方式是通过双引号定义。这种定义方式的特点是：输出变量内容时，引号里的变量及命令会经过**解析后再输出内容**，而不是把双引号中的变量名及命令（命令需要反引起来）原样输出。这种方式比较适合于字符串中附带有变量及命令且想将其解析后再输出的变量定义。

* **老男孩经验：**

数字内容的变量定义可以不加引号，其他没有特别要求的字符串等定义最好都加上双引号，如果真的需要原样输出就加单引号，定义变量加双引号是最常见的使用场景。

**4）普通变量的定义及输出示例**

例1：在命令行中输入一下变量，并逐一打印：

a=192.168.1.2

b='192.168.1.2'

c="192.168.1.2"

echo "a=$a"

echo "b=$b"

echo "c=${c}"

在命令行中输入，我将其放入脚本文件中，一样的效果。

**[root@localhost scripts]# ls**

m2.sh varTmp\_1.sh varTmp\_2.sh

**[root@localhost scripts]# cat varTmp\_1.sh**

a=192.168.1.2

b='192.168.1.2'

c="192.168.1.2"

echo "a=$a" # echo a=$a也能达到相同效果

echo "b=$b"

echo "c=${c}"

**[root@localhost scripts]# sh varTmp\_1.sh**

a=192.168.1.2

b=192.168.1.2

c=192.168.1.2

可见，将连续的**普通字符串**的内容赋值给变量，不管用不用引号，或者不管用什么引号，它的内容是什么，打印变量时就会输出什么。

例2：

**[root@localhost scripts]# cat varTmp\_2.sh**

a=192.168.1.2

b='192.168.1.2'

c="192.168.1.2"

a=192.168.1.2-$a #如果改为a=192.168.1.2-a**（1）**

b='192.168.1.2-$a' **#单引号，不是反引号； 此时的a为上一步中的a，不管怎样，由于是单引号，所以**'192.168.1.2-$a'

c="192.168.1.2-$a"

echo "a=$a"

echo "b=$b"

echo "c=${c}"

**[root@localhost scripts]# ./varTmp\_2.sh**

a=192.168.1.2-192.168.1.2 #条件1修改，该句输出为a=192.168.1.2-a，没有$，则变量a不会被解析

b=192.168.1.2-$a

c=192.168.1.2-192.168.1.2-192.168.1.2

### （5）将命令的结果给变量赋值

对需要获取命令结果的变量内容赋值的常见方法有两种：

（1）变量名=`ls` #<==把命令用反引号引起来，**不推荐**，因为容易与单引号混淆

（2）变量名=$(ls) #<==把命令用$()括起来，**推荐**

例：用两种方法把命令的结果赋值给变量。

[root@localhost scripts]# ls

m2.sh varTmp\_1.sh varTmp\_2.sh

[root@localhost scripts]# CMD=`ls` #<==``为反引号

[root@localhost scripts]# echo $CMD

m2.sh varTmp\_1.sh varTmp\_2.sh

[root@localhost scripts]# CMD1=$(pwd)

[root@localhost scripts]# echo $CMD1

/server/scripts #<==`打印当前用户所在的目录

**注：生产场景中把命令的结果作为变量的内容进行赋值的方法在脚本开发时很常见。**

例：按天打包网站的站点目录程序，生成不同的文件名（此为企业实战案例）

[root@localhost scripts]# CMD=$(date +%F) #<==将当前日期（格式为2017-09-07）赋值给CMD变量

[root@localhost scripts]# echo $CMD #<==输出变量的值

2017-09-07

[root@localhost scripts]# echo $(date +%F).tar.gz #<==直接输出时间命令的结果

2017-09-07.tar.gz

[root@localhost scripts]# tar zcf etc\_$(date +%F).tar.gz /etc #<==将时间作为压缩包名打包

tar: Removing leading `/' from member names

[root@localhost scripts]# ls

**etc\_2017-09-07.tar.gz**  m2.sh varTmp\_1.sh varTmp\_2.sh

RANDOM

这是随机数变量。在/dev/random这个文件中，可以通过这个随机数文件的变量（$RANDOM）来随机取得随机数值。在BASH环境下，这个RANDOM变量的内容介于0~32768之间。

[root@localhost czha]# declare -i number=$RANDOM\*10/32768; echo $number <==此时会随机取出0~9之间的值。

0

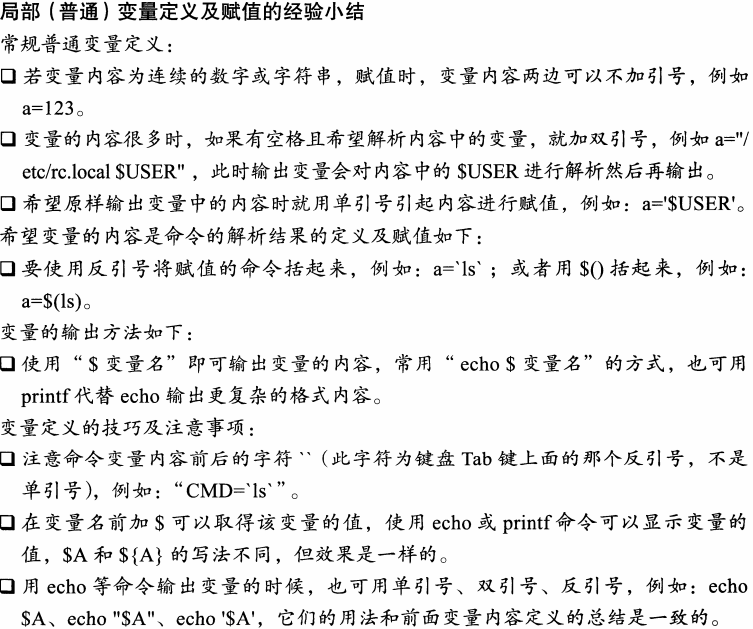
[root@localhost czha]# declare -i number=$RANDOM\*10/32768; echo $number

6

[root@localhost czha]# declare -i number=$RANDOM\*10/32768; echo $number

1

### （6）总结





## 设置环境变量

### （1）在哪儿配置环境变量

1. **用户环境变量配置**

.bash\_profile、

bashrc （推荐在此文件中优先设置）

**2）全局（系统）环境变量配置**

/etc/profile

/etc/bashrc （推荐在此文件中优先设置）

/etc/profile.d/

### （2）设置登录提示的两种方式

（1）在/etc/motd文件中增加（原文件中无任何内容）提示的字符串，如下：

[root@localhost etc]# cat motd

Welcome to czha's linux for shell training!!

（2）在/etc/profile.d/下面增加如下脚本：

[root@localhost etc]# cat /etc/profile.d/oldboy.sh

echo “Here is oldboy training”

### 显示与取消环境变量

1. **显示变量的值**

**echo $varible 或者 echo ${ varible }**

**2）清除变量**

**unset varible**

例如取消myname的设置：unset myname

**3）显示设置的变量**

在查看设置的变量时，**有3个命令可以显示变量的值：set、env和declare**（替代早期的typeset）。

* set命令显示所有的变量，包括全局变量和局部变量；
* env命令只显示全局变量；
* declare命令输出所有的变量、函数、整数和已经导出的变量。

set -o命令显示bash Shell的所有参数配置信息。

printenv也可查看

**3）举例**

用set可以查看环境变量，用unset可以取消本地变量和环境变量

[czha@localhost ~]$ echo $OLDBOY

I am oldboy.

[czha@localhost ~]$ **unset OLDBOY**

[czha@localhost ~]$ echo $OLDBOY

#什么都没有显示

[czha@localhost ~]$ tail /etc/profile

fi

done

#just for test

export OLDBOY="I am oldboy."

unset i

unset -f pathmunge

## 5、基础补充

### （1）变量前的$是什么意思?

A1：就是一个规定的标识吧，告诉shell，这个时候该取$后面这个变量的值了。:加个$表示这引用这个变量的值。

A2：**$变量名： 表示引用变量，可以用$c和${c}两种方式**。

[czha@localhost shell]$ echo $A

100

[czha@localhost shell]$ echo $AB

[czha@localhost shell]$ echo ${A}B

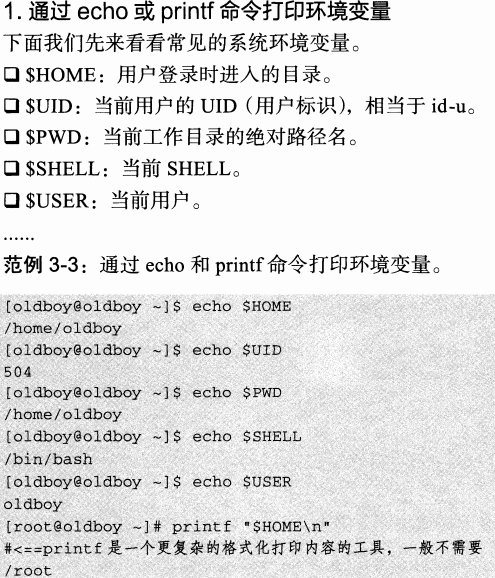
100B

[czha@localhost shell]$

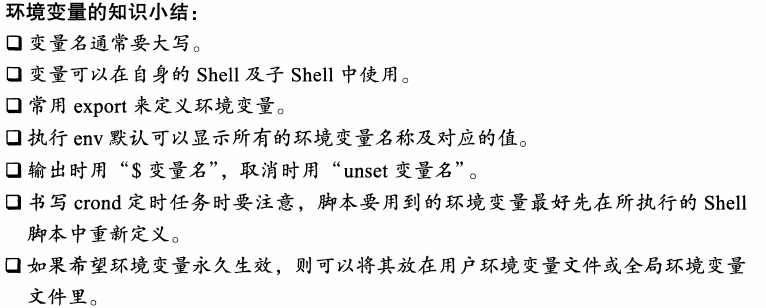
### （2）什么是shell子进程呢？

A1：就是说在我目前这个shell的情况下，去打开另一个新的shell（不是新的终端窗口），新的那个shell就是子进程。

### （3）有哪些常用环境变量？



### （4）环境变量知识总结



### （5）局部变量和环境变量练习

[czha@localhost shell]$ cat 1.sh

#!/bin/bash

echo 'hello world'

echo $A

[czha@localhost shell]$ A=100

[czha@localhost shell]$ ./1.sh # ./1.sh这个进程的父进程就是bash shell这个进程

hello world

#此时A什么值都没打印出来，上面定义的变量A仅为局部变量

[czha@localhost shell]$ export A=100

[czha@localhost shell]$ ./1.sh

hello world

100

如果此时CTR + ALT +T新开启一个终端，此时输入echo $A 会出现什么结果？

[czha@localhost ~]$ echo $A

[czha@localhost ~]$

说明：新开启的这个终端为另一个独立进程，不是上面的终端的子进程。

[czha@localhost shell]$ source 1.sh

hello world

100

[czha@localhost shell]$ . 1.sh

hello world

100

[czha@localhost shell]$

### （6）只读变量

是指不能被清除或重新赋值的变量。定义方式如下：

readonly myvar

[czha@localhost scripts]$ **readonly** AA=200

[czha@localhost scripts]$ AA=300 #无法修改

bash: AA: 只读变量

[czha@localhost scripts]$ unset AA #无法清除

bash: unset: AA: 无法反设定: 只读 variable

[czha@localhost scripts]$

## 6、位置参量（命令行参数）

**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。

$# 是传给脚本的参数个数

$0 是脚本本身的名字

$1 是传递给该shell脚本的第一个参数

$2 是传递给该shell脚本的第二个参数

$@ 是传给脚本的所有参数的列表

代表”$1”、 ”$2”、 ”$3”、 ”$4”之意，每个变量是独立的（用双括号括起来）。

$\* 是以一个单字符串显示所有向脚本传递的参数，与位置变量不同，参数可超过9个

代表“ ”$1c$2c$3c$4” ”，其中c为分隔符，默认为空格键，即代表“ “$1 $2 $3 $4” ”之意。

$$ 是脚本运行的当前进程ID号

$? 是显示最后命令的退出状态，0表示没有错误，其他表示有错误

如果Shell脚本程序后面加上**exit** number，执行完后，执行$?，会显示number

$! 最后一个后台运行进程的进程号。

eg：#wc &

#echo $!

## 7、数组

### （1）数组的定义

arr =("one" "two" "three");

arr =('one' 'two' 'three');

arr=(one two three);

arr =(1 two 3);

shell数组为弱类型，成员可随便定义。

### （2）数组的操作

　　我们使用数值类型数组arr=(1 2 3 4 5)做为源数组进行相关的讲解：获取数组长度，读取某个下标的值，对某个下标赋值，删除、分配和替换以及遍历。提一个知识点，我们在shell里面想获取某个变量的值，使用$符开头，如：$a或者${a}即可。

**1）获取数组的长度**

arr\_length=${#arr[\*]}或${#arr[@]}均可，即形式：${#数组名[@/\*]} 可得到数组的长度。

**2）读取某个下标的值**

　　arr\_index2=${arr[2]}，即形式：${数组名[下标]}

　　对某个下标赋值，这里需要提出两个问题：

　　　　第一个问题是如果该下标元素已经存在，会怎么样？

　　　　答：会修改该下标的值为新的指定值。

　　　　例如：arr[2]=100，数组被修改为(1 2 100 4 5)

　　　　第二个问题是如果指定的下标已经超过当前数组的大小，如上述的arr的大小为5，指定下标为10或者11或者大于5的任意值会如何？

　　　　答：新赋的值被追加到数组的尾部。

　　　　例如：arr\_number[13]=13，数组被修改为(1 2 100 4 5 13)

**3）删除操作**

　　　　清除某个元素：unset arr[1]，这里清除下标为1的数组；

清空整个数组：unset arr\_number;

**4）分片访问**

　　　　分片访问形式为：${数组名[@或\*]:开始下标:结束下标}，注意，不包括结束下标元素的值。

　　　　例如：${arr[@]:1:4}，这里分片访问从下标为1开始，元素个数为4。

**5）模式替换**

　　　　形式为：${数组名[@或\*]/模式/新值}

　　　　例如：${arr[@]/2/98}

**6）数组的遍历**

　　　　数组遍历我们使用for语句来演示：

　　　　for v in ${arr [@]}; do

　　　　　　echo $v;

　　　　done

输出数组的所有元素可以用如下方法：

echo ${arr[\*]}

**7）示例**

[czha@localhost scripts]$ arr=(1 2 3 4 5)

[czha@localhost scripts]$ echo ${#arr[\*]}

5

[czha@localhost scripts]$ arr[2]=100

[czha@localhost scripts]$ echo ${arr[2]}

100

[czha@localhost scripts]$ arr[13]=13

[czha@localhost scripts]$ for v in ${arr[@]};do \

> echo $v;\

> done

1

2

100

4

5

13 //虽然位置看起来像是下标[5]，实则是13，如下

[czha@localhost scripts]$ echo ${arr[5]}

[czha@localhost scripts]$ echo ${arr[13]}

13

[czha@localhost scripts]$ echo ${#arr[\*]}

6

# 三、输入输出

read

echo

echo输出颜色与光标定位

## 1、输入——read命令

有如下几种常用形式

|  |  |
| --- | --- |
| 格式 | 说明 |
| read var | 从标准输入读取一行并赋值给变量var |
| read | 标准输入读取一行并赋值给内置变量REPLY |
| read -a arr | 读取一组词，依次赋值给数组arr |
|  |  |

-p：表示一个提示符

-t：表示超时时间

-a：表示将标准输入的内容赋值给数组变量

要读取来自键盘输入的变量，就是用read这个命令。这个命令最常被用在shell script的编写当中。

范例一：

[czha@localhost scripts]$ read var

abc

[czha@localhost scripts]$ echo $var

abc

[czha@localhost scripts]$ read

abc

[czha@localhost scripts]$ echo $REPLY

abc

[czha@localhost scripts]$ read -a arr

1 2 3

[czha@localhost scripts]$ echo ${arr[\*]}

1 2 3

[czha@localhost scripts]$ read -p "please input 3 digits:" -t 10 -a arr

please input 3 digits:1 2 3 4

[czha@localhost scripts]$ echo ${arr[\*]}

1 2 3 4

范例二：提示用户30秒内输入自己的大名，将该输入字符串作为名为named的变量内容

[root@localhost scripts]# read -p "Please keyin your name:" -t 30 named

Please keyin your name:zhajin

[root@localhost scripts]# echo $named

zhajin

[root@localhost scripts]#

## 2、echo

-n：表示在字符输出字符后面不会换行

-e：会解析字符

[czha@localhost scripts]$ echo hello

hello

[czha@localhost scripts]$ echo "hello"

hello

[czha@localhost scripts]$ echo $var

abc

[czha@localhost scripts]$ echo ${var}

abc

[czha@localhost scripts]$ echo ${var}D

abcD

[czha@localhost scripts]$ echo $varD

[czha@localhost scripts]$ echo "hello world"

hello world

[czha@localhost scripts]$ echo **-n** "hello world"

hello world[czha@localhost scripts]$ //注意，这里由于-n的选项输出没有换行

[czha@localhost scripts]$ echo "\t"

\t

[czha@localhost scripts]$ echo \t

t

[czha@localhost scripts]$ echo **-e** "\t"A

A //注意，这里由于-e的选项，\t表示TAB，解析后\t变成TAB符

[czha@localhost scripts]$

## 3、echo输出颜色与光标定位

设置前景色

设置背景色

设置光标位置

博客：<http://blog.chinaunix.net/uid-26495963-id-3189345.html>

**范例：俄罗斯方块，输出不同颜色的方块**

# 四、算术扩展

## 1、单引号、双引号、反引号的区别

见本文档、第二章——2——（4）、（5）

## 2、命令替换

alias CD cd

## 3、算术扩展

中括号和双扩号都可以。

expr

eg： expr 4 + 5 #+两边必须有空格

eg： let n=n+1 #表达式时间不能有空格

[czha@localhost scripts]$ n=5

[czha@localhost scripts]$ echo n+1

n+1

[czha@localhost scripts]$ echo $n+1

5+1

[czha@localhost scripts]$ echo $[$n+1]

6

[czha@localhost scripts]$ echo $[$n \* 2]

10

[czha@localhost scripts]$ echo $(($n \* 2))

10

[czha@localhost scripts]$ echo $(($n + 1))

6

[czha@localhost scripts]$ ((n+=1))

[czha@localhost scripts]$ echo $n

6

[czha@localhost scripts]$ r=$[$n+1]

[czha@localhost scripts]$ echo $r

7

[czha@localhost scripts]$ expr 4 + 5

9

[czha@localhost scripts]$ expr 4 \\* 5

20

[czha@localhost scripts]$ expr 4 \/ 5

0

[czha@localhost scripts]$ expr 4 \/ 4

1

[czha@localhost scripts]$ let n+=1

[czha@localhost scripts]$ echo $n

7

[czha@localhost scripts]$ n=2

[czha@localhost scripts]$ let n+=1

[czha@localhost scripts]$ echo $n

3

[czha@localhost scripts]$ let n += 1

bash: let: +=: 语法错误: 期待操作数 （错误符号是 "+="）

# 五、条件测试

可以看<<鸟哥的私房菜>P379，善用判断式

## 1、条件测试

条件测试可以根据某个特定条件是否满足，来执行相应的任务。

Bash中允许测试两种类型的条件：

命令成功或失败

表达式为真或假

任何一种测试中，都要有退出状态（返回值），退出状态为0表示命令成功或表达式为真，非0则表示命令失败或表达式为假。

状态变量$?中保存命令退出状态的值

grep study /etc/passwd

echo $?

或者

grep study /etc/passwd; ehco $?

## 2、表达式测试

表达式测试包括：字符串测试、整数测试和文件测试。

内置测试命令**test**

[czha@localhost scripts]$ x=5;y=10

[czha@localhost scripts]$ test $x -gt $y

[czha@localhost scripts]$ echo $?

1

**中括号**

[czha@localhost scripts]$ [ $x -gt $y ] []中两边必须有空格，否则出现语法错误，如下。

[czha@localhost scripts]$ echo $?

1

[czha@localhost scripts]$ [ $x -gt $y]

bash: [: 缺少 `]'

[czha@localhost scripts]$

test与单中括号方式是一样的。

2.x版本以上的Bash中可以用**双中扩号**来测试表达式的值，此时可以使用**通配符**（相对于上面单中括号）进行**模式匹配**。

[czha@localhost scripts]$ name=Tom

[czha@localhost scripts]$ [[ $name = [Tt]?? ]] 双中括号可以匹配

[czha@localhost scripts]$ echo $?

0

[czha@localhost scripts]$ [ $name = [Tt]?? ] 单中括号无法匹配

[czha@localhost scripts]$ echo $?

1

[czha@localhost scripts]$

字符串测试

整数测试

逻辑测试

文件测试

## 3、括号总结

|  |  |
| --- | --- |
| 括号形式 | 功能说明 |
| ${…} | 获取变量值 |
| $(…) | 命令替换 |
| $[…] | 让无类型的变量参数算术运算 |
| $((…)) | 同上 |
| (( … )) | 算术运算（同let命令） |
| […] | 条件测试（等价于test命令） |
| [[…]] | 条件测试，支持模式匹配通配符 |

# 六、条件与分支语句

## 1、利用if…then判断

if [条件判断一]; then #分号必须有，因为shell语句一行有多个语句，必须用分号分隔

……

elif [条件判断式二]; then

……

else

……

fi

## 2、利用case…esac判断

case $变量名称 in

“第一个变量名称”)

程序段

;; #就是break的意思

“第二个变量名称”)

程序段

;;

\*)

“其他程序执行段”

exit 1

；；

esac

# 七、循环

## 1、for循环语句

**语法结构**

for variable in list #每一次循环，依次把列表list中的一个值赋给循环变量

do #循环开始的标志

commands #循环变量每取一次值，循环体就执行以便

done #循环结束的标志

**几点说明**

列表list可以是 命令替换、变量名替换、字符串和文件名列表（可包含通配符）

for循环执行的次数取决于列表list中的单词个数

for循环体中一般要出现循环变量，但也可以不出现

**循环执行的过程**

执行第一轮循环时，将list中的第一个词赋值给循环变量，并把该词从list中删除，然后进入循环体，执行do和done之间的命令。下一次进入循环体时，则将第二个词赋给循环变量，并把该词从list中删除，再往后的循环也以此类推。当list中的词全部被移走后，循环就结束了。

[czha@localhost scripts]$ cat loop\_01.sh

#!/bin/bash

for i in 1 2 3 4 5

do

echo the value of i is $i

done

[czha@localhost scripts]$ chmod +x loop\_02.sh

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1

2

3

**参量列表的使用**

位置参量的使用：$\*、”$\*”、$@、”$@”

[czha@localhost scripts]$ cat loop\_02.sh

#!/bin/bash

#for i #参量加"",将""内的内容当作一个参量 （1）

#for i in $\* #参量加"",仍当作没有""的情况 （2）

#for i in “$\*” #参量加"",仍当作没有""的情况 （3）

#for i in $@ #参量加"",仍当作没有""的情况 （4）

#for i in "$@" #参量加"",将""内的内容当作一个参量 （5）

do

echo $i

done

**针对（1）for i**

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1

2

3

[czha@localhost scripts]$ ./loop\_02.sh "1 2" 3

1 2

3

**针对（2）for i in $\***

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1

2

3

[czha@localhost scripts]$ ./loop\_02.sh "1 2" 3

1

2

3

**针对（3）for i in “$\*”**

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1 2 3

[czha@localhost scripts]$ ./loop\_02.sh "1 2" 3

1 2 3

**针对（4）for i in $@**

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1

2

3

[czha@localhost scripts]$ ./loop\_02.sh "1 2" 3

1

2

3

**针对（5）for i in "$@"**

[czha@localhost scripts]$ ./loop\_02.sh 1 2 3

1

2

3

[czha@localhost scripts]$ ./loop\_02.sh "1 2" 3

1 2

3

**对\*进行通配，统计目录下有多少个文件**。

[czha@localhost scripts]$ cat file\_list.sh

#!/bin/bash

count=0;

for f in \*

do

count=$[$count+1]

done

echo There are $count files in $(pwd)

[czha@localhost scripts]$ ./file\_list.sh

There are 21 files in /home/czha/smbroot/server/scripts

## 2、类似于C语言的for循环

for((初始值;限制值;执行步长))

do

程序段

done

当前行： i

总行数： $1

**公式：**

**空格数=$1-$i-1**

**星号数=2\*$i+1**

**范例：**

[czha@localhost scripts]$ cat 05for.sh

#!/bin/bash

if [ $# -ne 1 ]; then

echo 'usage:$0 <n>'

exit 1

fi

if [ $1 -lt "5" -o $1 -gt "15" ]; then

echo 'usage:$0 <n>'

echo ' where 5<=n<=15'

exit 1

fi

for ((i=0;i<$1;i++))

do

for ((j=0; j<$[$1-$i-1];j++))

do

echo -n " "

done

for ((j=0;j<$[2\*$i+1];j++))

do

echo -n "\*"

done

echo -ne '\n'

done

## 3、while循环语句

while [ condition ] #condition就是判断式

do

程序段

done

**范例：**打印\*号

[czha@localhost scripts]$ cat 06while.sh

#!/bin/sh

if [ $# -ne "1" ]

then

echo 'usage:$0 <n>'

exit 1

fi

if [ $1 -lt "5" -o $1 -gt "15" ]

then

echo 'usage:$0 <n>'

echo "where 5<=n<<15"

exit 1

fi

allLine=$1

curLine=0

while [ $curLine -lt $allLine ]

do

nSpace=$[$allLine-$curLine-1]

while [ $nSpace -gt "0" ]

do

nSpace=$[$nSpace-1]

echo -n " "

done

nStar=$[2\*$curLine+1]

while [ $nStar -gt "0" ]

do

nStar=$[$nStar-1]

echo -n '\*'

done

echo

curLine=$[$curLine+1]

done

## 4、until循环语句

until [ condition ] #condition就是判断式

do

程序段

done

当条件满足时，就退出循环。

**范例：**打印3次AAA，退出

[czha@localhost scripts]$ cat 07until.sh

#!/bin/bash

count=0

until [ $count -eq "3" ]

do

echo "AAA"

sleep 1

#count=$[$count+1]

count=$(($count+1)) #同上一句屏蔽了的语句，(())和[]都可以进行运算处理

done

## 5、break、continue、exit、sleep

## 6、select循环与菜单的使用

select varible in list

do

commands

done

list:

1. A 选择1对应A
2. B
3. C

**示例：**

[czha@localhost scripts]$ cat 08select.sh

#!/bin/bash

PS3="favorite pets? : "

select var in dogs cats birds

do

case $var in

"dogs")

echo dogs are my favorite pet

break

;;

"cats")

echo cats are my favorite pet

;;

"birds")

echo birds are my favorite pet

;;

"\*") #仅\*也可以,变量赋值

echo none of my favorite pet

;;

esac

#break

done

# 八、函数

三种定义方式：

function fname() { #荐

commands

}

function fname {

commands

}

fname() {

commands

}

* 只需输入函数名即可调用该函数
* 函数必须在调用之前定义

## 1、简单调用函数

**范例：**最简单的shell函数

[czha@localhost scripts]$ cat 01fun.sh

#!/bin/bash

#fun\_a() {

# echo "Enterning function"

# echo "Exit function"

#}

#fun\_a #调用函数fun\_a

function fun\_a() {

echo "Enterning function"

echo "Exit function"

}

fun\_a #调用（即执行）函数fun\_a

## 2、函数参数传递

**示例：**fun\_a函数传递两个参数，并将该函数名、参数列表打印出来。

[czha@localhost scripts]$ cat 02fun.sh

#!/bin/bash

function fun\_a() {

echo "the pareameter's count: $#"

echo "the first parameter :$1"

echo "the second parameter :$2"

}

fun\_a a b #调用函数fun\_a，并传入两个参数a、b

## 3、调用其他文件中的函数

**范例：**调用其他文件中的函数

#上一级目录中定义函数文件

[czha@localhost scripts]$ cat ../tool

test() {

echo "hello world!"

}

#调用定义的test函数

[czha@localhost scripts]$ cat 03fun.sh

#!/bin/bash

**.** ../tool #.号后面一定要有一个空格，后面指定具体目录下的原函数文件 即**include头文件**

test

# 九、正则表达式

## 1、字符串操作

|  |  |
| --- | --- |
| ${#var} | 返回字符串变量var的长度 |
| ${var:m} | 返回${var}中从**下标m**到最后的部分的字符（下标从0开始，类似数组） |
| ${var:m:len} | 返回${var}中**从下标m开始，长度为len**的部分 |
| ${var#pattern} | 删除${var}中**开头部分**与pattern匹配的最小部分 |
| ${var##pattern} | 删除${var}中**开头部分**与pattern匹配的最大部分 |
| ${var%pattern} | 删除${var}中**结尾部分**与pattern匹配的最小部分 |
| ${var%%pattern} | 删除${var}中**结尾部分**与pattern匹配的最大部分 |
| ${var/old/new} | 用new替换${var}中第一次出现的old |
| ${var//old/new} | 用new替换${var}中所有的old（全局替换） |
|  |  |

注：pattern，old中可以使用通配符

**示例:**

[czha@localhost scripts]$ var=abcdefg

[czha@localhost scripts]$ echo ${#var}

7

[czha@localhost scripts]$ echo ${var:2}

cdefg

[czha@localhost scripts]$ echo ${var:2:3}

cde

[czha@localhost scripts]$ cat 04string.sh

#!/bin/bash

var="/aa/bb/cc"

result1=${var#\*/}

result2=${var##\*/}

result3=${var%/\*}

result4=${var%%/\*}

echo $var

echo '${var#\*/}'=$result1

echo '${var##\*/}'=$result2

echo '${var%/\*}'=$result3

echo '${var%%/\*}'=$result4

[czha@localhost scripts]$ ./04string.sh

/aa/bb/cc

${var#\*/}=aa/bb/cc

${var##\*/}=cc

${var%/\*}=/aa/bb

${var%%/\*}=

[czha@localhost scripts]$ var=testaaatest

[czha@localhost scripts]$ echo ${var**/**test/TEST}

TESTaaatest

[czha@localhost scripts]$ var=testaaatest

[czha@localhost scripts]$ echo ${var**//**test/TEST}

TESTaaaTEST

### 随机数的产生

生成随机数的特殊变量

echo $**RANDOM**

### expr命令

通用表达式计算命令

### shif命令

移动位置参数

### eval命令

# 附：

## 通配符与特殊符号

|  |  |
| --- | --- |
| 符号 | 意义 |
| \* | 代表0到无穷多个任意字符 |
| ? | 代表一定有一个任意字符 |
| [] | 同样代表一定有一个在中括号内的字符（非任意字符）。例如[abcd]代表一定有一个字符，可能是a，b，c，d这四个中任何一个 |
| [-] | 若有减号在中括号内，代表在编码顺序内的所有字符。例如[0-9]代表0到9之间的所有数字，因为数字的语系编码是连续的。 |
| [^] | 若中括号内的第一个字符为指数符号（^），那表示原向选择，例如[^abc]代表一定有一个字符，只要是非abc的其他字符就接受的意思 |

## 特殊符号

除了通配符之外，bash环境中的特殊符号还有哪些？

常用测试标志

见《鸟哥的私房菜》P380-381