|  |
| --- |
| 文章标题  shell |
| **作者：归档：学习笔记** |
| **快捷键：**  Ctrl + 1 标题1  Ctrl + 2 标题2  Ctrl + 3 标题3  Ctrl + 4 实例  Ctrl + 5 程序代码  Ctrl + 6 正文 |
| **格式说明：**  蓝色字体：注释  黄色背景：重要  绿色背景：注意 |

****

**老男孩教育教学核心思想6重：重目标、重思路、重方法、重实践、重习惯、重总结**

**学无止境，老男孩教育成就你人生的起点！**

# shell 介绍

## shell开篇介绍

Shell脚本语言是实现Linux/UNIX系统管理及自动化运维所必备的重要工具，Linux/UNIX系统的底层及基础应用软件的核心大部分涉及Shell脚本的内容。每一个合格的Linux系统管理员或运维工程师，都需要熟练的编写Shell脚本语言，并能够阅读系统及各类软件附带的Shell脚本内容。只有这样才能提升运维人员的工作效率，适应日益复杂的工作环境，减少不必要的重复工作，从而为个人的职场发展奠定较好的基础。

### 面试题

2017最新企业Shell面试题及企业运维实战共30道案例

<http://oldboy.blog.51cto.com/2561410/1867160>

### 学好Shell编程所需的基础知识

能够熟练使用 vim编辑器,熟悉 SSH终端及“.vimrc”等的配置。

有一定的 Linux命令基础,至少需要掌握80个以上 Linux常用命令,并能够熟练使用它们。

要熟练掌握Linux正则表达式及三剑客命令(grep、 sed、 awk)。

熟悉常见的Linux网络服务部署、优化、日志分析及排错。

### 如何学好Shell编程

学好Shell编程的核心：多练→多思考→再练→再思考，坚持如此循环即可！

新手大忌：不可拿来主义，可以模仿，但是要自己嚼烂了再吃下去，否则会闹肚子的。

格言：你觉得会了并不一定会了，你认为对的并不一定对的。

大家要勤动手，自行完成学习笔记和代码的书写。通过每一个小目标培养自己的兴趣以及成就感。

## 使用的命令解释器

### 什么是shell

命令解释器，翻译官，人机交互的桥梁

linux脚本默认使用的命令解释器是 /bin/bash

#### 查看系统支持的shell

[root@oldboyedu ~]# cat /etc/shells

/bin/sh

/bin/bash

/sbin/nologin

/usr/bin/sh

/usr/bin/bash

/usr/sbin/nologin

当前系统命令解释器

CentOS默认的Shell是（bash）

[root@oldboyedu ~]# echo $SHELL

/bin/bash

## Shell脚本的建立和执行

### 脚本放在放在统一的目录

mkdir -p /server/scripts/

cd /server/scripts/

### 创建Shell脚本，使用vim编辑，文件名规范：名字要有意义，结尾以.sh结束

vim oldboyedu.sh

### 规范的Shell脚本第一行会指出由哪个程序(解释器)来执行脚本中的内容。

#!/bin/bash

或

#!/bin/sh

注意：这一行必须在每个脚本顶端的第一行，如果不是第一行则为脚本注释行。

注释以#开始，除了第一行

sh是bash的软链接，推荐标准写法#!/bin/bash

### 注释

在Shell脚本中，跟在#后面的内容表示注释。注释部分不会被执行，仅给人看。注释可以自成一行，也可以跟在命令后面，与命令同行。要养成写注释的习惯，方便自己与他人。

最好不用中文注释，因为在不同字符集的系统会出现乱码。

### 版权信息

在家目录放置.vimrc文件，新创建sh结尾的脚本文件

#### Shell脚本开发的规范和习惯

1) 放在统一的目录

2) 脚本以.sh为扩展名

3) 开头指定脚本解释器。

4) 开头加版本版权等信息，可配置~/.vimrc文件自动添加。

5) 脚本不要用中文注释，尽量用英文注释。

6) 代码书写优秀习惯

### 脚本书写注意事项

a、成对的内容一次性写出来，防止遗漏，如[ ]、' '、" "等

b、[]两端要有空格，先输入[],退格，输入2个空格，再退格写。

c、流程控制语句一次书写完，再添加内容。

if 条件

then

内容

fi

d、通过缩进让代码易读。

f、脚本中的引号都是英文状态下的引号，其他字符也是英文状态。

好的习惯可以让我们避免很多不必要的麻烦，提高工作效率。

### 执行脚本的方法

#### 方法一：常用方法

sh oldboyedu.sh

#### 方法二：加执行权限，直接执行

chmod +x /server/scripts/oldboyedu.sh

直接执行

/server/scripts/oldboyedu.sh

#### 方法三：在当前 shell 中执行一个文件中的命令

source /server/scripts/oldboyedu.sh

或者

. /server/scripts/oldboyedu.sh

### 在哪里执行了Shell脚本？

#### 写一个交单脚本

[root@oldboyedu scripts]# vim oldboyedu.sh

#!/bin/bash

##############################################################

# File Name: oldboyedu.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-10-26 10:50:22

# Description:

##############################################################

word="Hello world"

echo $word

#### 执行脚本

[root@oldboyedu scripts]# sh oldboyedu.sh

Hello world

[root@oldboyedu scripts]# echo $word

[root@oldboyedu scripts]# source oldboyedu.sh

Hello world

[root@oldboyedu scripts]# echo $word

Hello world

#### 结论：

sh命令在当前Shell窗口（A）执行脚本会先创建一个新的Shell进程（窗口，B），然后在执行脚本内容，最后将脚本的最终结果传给当前的Shell窗口（A）。

source直接在当前Shell窗口（A）执行脚本，所以脚本的变量在当前Shell窗口也生效了。

是否需要脚本中的变量？

什么时候用？ source /etc/profile

其他 sh

## Shell全局变量

### 变量

变量就是用一个固定的字符串(也可能是字符数字等的组合)，替代更多更复杂的内容，这个内容里可能还会包含变量和路径，字符串等其他内容。变量的定义是存在内存中

### 变量类型

#### 变量分为2类：

全局变量(也可称为环境变量):

可以在创建他们的Shell及其派生出来的子Shell中使用。环境变量又可以分为自定义环境变量和bash内置的环境变量。

局部变量（普通变量）：

只能在创建他们的Shell函数或Shell脚本中使用，还记得前面的$user? 我们创建的一般都是普通变量。

### 环境变量

#### 环境变量查看，设置

所有环境变量均为大写。

env # 查看环境变量

export # 设置环境变量

unset #用unset取消本地变量和环境变量

#### 临时设置环境变量

[root@oldboyedu-39-nb scripts]# lidao=lidao.blog.51cto.com

[root@oldboyedu-39-nb scripts]# echo $lidao

lidao.blog.51cto.com

#### 想要处处生效，写入环境变量配置文件。

全局生效优选/etc/bashrc，当前用户生效优选~/.bashrc。

全局环境变量配置文件：/etc/profile、/etc/bashrc、/etc/profile.d/

用户环境变量配置文件：~/.bash\_profile、~/.bashrc

换件变量配置文件开机启动顺序

/etc/profile

~/.bash\_profile

~/.bashr

/etc/bashrc

### 局部变量：只能在当前窗口用，

定义变量，变量名以字母、数字、下划线组成，必须以字母开头

#### 规范的变量名写法定义：见名知意。

OldboyAge=1 <==每个单词首字母大写

oldboy\_age=1 <==每个单词之间用"\_"

oldboyAgeSex=1 <==驼峰语法：首个单词字母小写，其余单词首字母大写。

#### 普通变量的作用：精简代码，容易更改，但只能在当前窗口生效

[root@oldboy-01-nb ~]# OLDBOY=lidao

[root@oldboy-01-nb ~]# echo $OLDBOY

lidao

[root@oldboy-01-nb ~]# vim /server/scripts/lidao.sh

#!/bin/bash

echo $OLDBOY

~

~

~

"/server/scripts/lidao.sh" 2L, 25C written

[root@oldboy-01-nb ~]# sh /server/scripts/lidao.sh

[root@oldboy-01-nb ~]#

#### $与${} 用法 将变量放入{}中，可以让系统认为只有{}中的才是变量

[root@oldboyedu-39-nb ~]# week=10

[root@oldboyedu-39-nb ~]# echo $week

10

[root@oldboyedu-39-nb ~]# echo $weekday

[root@oldboyedu-39-nb ~]# echo ${week}day

10day

#### 把一个命令作为变量

变量名=`ls` <==反引号

变量名=$(ls)

#### 自定义环境变量

建议所有环境变量名均为大写

必须用export命令定义

export OLDBOY=1

永久生效

/etc/profile

与用户环境变量有关的文件 目录

全局环境变量配置文件

/etc/profile、/etc/bashrc、/etc/profile.d/(用户登录到系统 会运行这个目录下面的脚本)

用户环境变量配置文件

~/.bash\_profile、~/.bashrc

## 特殊的变量

### $0

获取当前执行的shell脚本的文件名，如果执行脚本带路径那么就包括脚本路径

#### 使用场景：

echo $"Usage: $0 {start|stop|status|restart|reload|force-reload}"

### $n

获取当前执行的shell脚本的第n个参数值，n=1..9，当n为0时表示脚本的文件名，如果n大于9用大括号括起来{10}，参数以空格隔开。

### $#

获取当前执行的shell脚本后面接的参数的总个数

#### 使用场景：判断用户传入的参数个数，决定是否让他使用脚本。

[root@oldboy-01-nb ~]# cat /server/scripts/jisuan.sh

#!/bin/bash

a=$1

b=$2

echo $((a+b))

echo $#

[root@oldboy-01-nb ~]# sh /server/scripts/jisuan.sh 2 3

5

2 ###参数

### $\*

获取当前shell的所有传参的参数，不加引号同$@;如果给$\*加上双引号，例如: “$\*”,则表示将所有的参数视为单个字符串，相当于“$1$2$3”

### $@

获取当前shell的所有传参的参数，不加引号同$\*;如果给$@加上双引号，例如: “$@”,则表示将所有参数视为不同的独立字符串，相当于“$1” “$2” “$3” “……”，这是将参数传递给其他程序的最佳方式，因为他会保留所有内嵌在每个参数里的任何空白。

当“$\*”和“$@”都加双引号时，两者有区别，都不加双引号时，两者无区别。

常用"$@"

### $? 显示上一个是否执行成功

如果命令执行成功 会显示0 如果命令执行失败，则会显示不是0

########## 如果是0，成功了 如果不是0就失败了。

[root@oldboy-01-nb ~]# echo 233

233

[root@oldboy-01-nb ~]# echo $?

0

[root@oldboy-01-nb ~]# cho 344

-bash: cho: command not found

[root@oldboy-01-nb ~]# echo $?

127

### $1 $2 $数字，是脚本与命令行桥梁，把命令的内容传入到脚本 $1第一个参数，$2第二个参数

#### 使用变量省事 方便

[root@oldboy-01-nb ~]# vim /server/scripts/jisuan.sh

#!/bin/bash

a=$1 变量a=第一个参数

b=$2 变量b=第二个参数

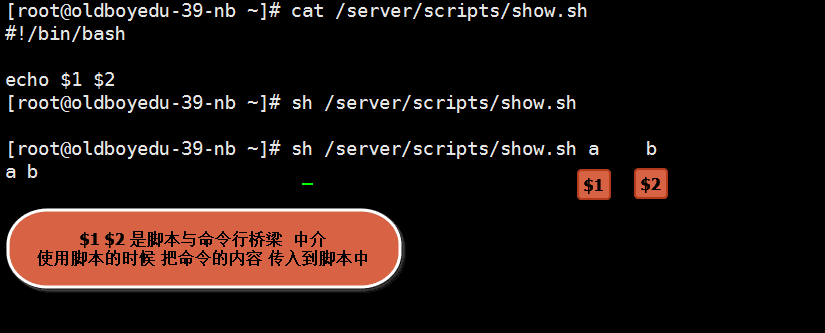
echo $((a+b))

[root@oldboy-01-nb ~]# sh /server/scripts/jisuan.sh 3 4 ###必须是数字

7

[root@oldboy-01-nb ~]# sh /server/scripts/jisuan.sh 20 50

70



#### 要求：给cal2.sh脚本传入三个参数 ，显示三个参数相乘的结果

#sh /server/scripts/cal2.sh 10 20 30

[root@oldboy-01-nb ~]# vim /server/scripts/ca12.sh

#!/bin/bash

a=$1

b=$2

c=$3

echo $((a\*b\*c))

[root@oldboy-01-nb ~]# sh /server/scripts/ca12.sh 2 3 5

30

### $? 显示上一个是否执行成功

如果命令执行成功 会显示0 如果命令执行失败，则会显示不是0

########## 如果是0，成功了 如果不是0就失败了。

[root@oldboy-01-nb ~]# echo 233

233

[root@oldboy-01-nb ~]# echo $?

0

[root@oldboy-01-nb ~]# cho 344

-bash: cho: command not found

[root@oldboy-01-nb ~]# echo $?

127

## 定义变量的3种方式

### 直接赋值(直观，但是死板，需要改变时必须编辑脚本文件)

x=1

dir=/etc/

file=/etc/hosts

### 传参（传递参数，借助位置变量）

好处：灵活，脚本无需变动，但是通过传入参数的变化实现不同的结果

### 交互式设置变量（使用read命令实现）

read

-p ： 添加提示语句

-t ： 设置超时时间

-s ： 关闭回显（返回显示）

#### 练习

#read -t 10 -p "请输入密码：" oldboy

read -s -p "请输入密码：" oldboy

echo

echo $oldboy

#### 练习题

执行脚本后显示

请输入你的银行卡账号：

请输入你的密码：

你的卡号为：

密码为：

[root@centos7 scripts]# sh ceshi.sh

请输入银行卡账号:12345

请输入你的密码：

你的卡号为:12345

你的密码为: 123

查看脚本内容

[root@centos7 scripts]# cat ceshi.sh

#!/bin/bash

##############################################################

# File Name: ceshi.sh

# Version: V1.0

# Author: yyc

# Organization: www.oldboyedu.com

# Created Time : 2017-10-27 09:25:04

# Description:

##############################################################

read -p "请输入银行卡账号:" oldboy

read -s -p "请输入你的密码：" old

echo

echo 你的卡号为:$oldboy

echo 你的密码为: $old

## 变量的子串（了解）

### ${#a} 获取变量值的长度。

#### wc统计个数

[root@oldboyedu scripts]# a=`uuidgen`

[root@oldboyedu scripts]# echo $a|wc -L

36

[root@oldboyedu scripts]# time echo $a|wc -L

36

#### ${#a} 获取变量值的长度

[root@oldboyedu scripts]# echo $a

3cd480cd-4480-4cbd-8d74-79d634ff4b2e

[root@oldboyedu scripts]# echo ${a}

3cd480cd-4480-4cbd-8d74-79d634ff4b2e

[root@oldboyedu scripts]# echo ${#a}

36

[root@oldboyedu scripts]# time echo ${#a}

36

查看命令执行速度

real 0m0.000s

user 0m0.000s

sys 0m0.000s

## 数值计算

### 整数运算

#### $(()) 计算

[root@oldboyedu ~]# echo $((1+1\*3/5%2-4))

-3

[root@oldboyedu ~]# ((a=1+2\*\*3-4%3))

[root@oldboyedu ~]# echo $a

8

[root@oldboyedu ~]# b=$((1+2\*\*3-4%3))

[root@oldboyedu ~]# echo $b

8

[root@oldboyedu ~]# echo $b

8

[root@oldboyedu ~]# echo $((a++))

8

[root@oldboyedu ~]# echo $a

9

[root@oldboyedu ~]# echo $((a++))

9

[root@oldboyedu ~]# echo $a

10

[root@oldboyedu ~]# echo $((a--))

10

[root@oldboyedu ~]# echo $((a--))

9

[root@oldboyedu ~]# echo $((a--))

8

[root@oldboyedu ~]# echo $a

7

[root@oldboyedu ~]# echo $((++a))

8

#### let （整数）

[root@oldboyedu ~]# i=1

[root@oldboyedu ~]# i=i+1

[root@oldboyedu ~]# echo $i

i+1

[root@oldboyedu ~]# i=1

[root@oldboyedu ~]# let i=i+1

[root@oldboyedu ~]# echo $i

2

#### expr（整数运算） 判断变量是否为整数

[root@oldboyedu ~]# expr 1 + 1

2

[root@oldboyedu ~]# expr 1 - 1

0

[root@oldboyedu ~]# expr 1 \* 1

expr: 语法错误

[root@oldboyedu ~]# expr 1 \\* 1

1

[root@oldboyedu ~]# expr 1 / 1

1

[root@oldboyedu ~]# expr 1+1

1+1

判断变量是否为整数

[root@oldboyedu ~]# expr 1 + a

expr: 非整数参数

[root@oldboyedu ~]# echo $?

2

[root@oldboyedu ~]# expr 1 + 1

2

[root@oldboyedu ~]# echo $?

0

[root@oldboyedu ~]# expr a + a

#### $[] 计算

[root@oldboyedu ~]# echo $[1+1]

2

[root@oldboyedu ~]# echo $[1+3]

4

#### bc计算

[root@oldboyedu ~]# echo 1 + 1|bc

2

[root@oldboyedu ~]# echo 1+1|bc

2

[root@oldboyedu ~]# echo 1+1.1|bc

2.1

[root@oldboyedu ~]# echo 1/3|bc

0

[root@oldboyedu ~]# echo 'scale=2;1/3'|bc

.33

[root@oldboyedu ~]# echo 'scale=5;1/3'|bc

.33333

习题：输出结果1+2+3+4+5+6+7+8+9+10=55

[root@oldboyedu ~]# seq -s + 1 10

1+2+3+4+5+6+7+8+9+10

[root@oldboyedu ~]# echo `seq -s + 1 10`|bc

55

#### awk计算数字

[root@zyops ~]# echo "7.7 3.8"|awk '{print ($1-$2)}'

3.9

[root@zyops ~]# echo "358 113"|awk '{print ($1-3)/$2}'

3.14159

[root@zyops ~]# echo "3 9"|awk '{print ($1+3)\*$2}'

54

[root@backup scripts]# awk BEGIN'{print 1.2+3.3}'

4.5

#### python 计算

[root@oldboyedu ~]# python

Python 2.7.5 (default, Nov 20 2015, 02:00:19)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> 2+2

4

>>> 5-6

-1

## 条件表达式

### 整数判断

大于 -gt great than

大于等于 -ge great equal

等于 -eq equal

小于等于 -le less equal

小于 -lt less than

不等于 -ne not equal

[root@oldboyedu ~]# [ 1 -eq 1 ] && echo "表达式成立" || echo "表达式失败"

表达式成立

[root@oldboyedu ~]# [ 1 -ge 1 ] && echo "表达式成立" || echo "表达式失败"

表达式成立

[root@oldboyedu ~]# [ 1 -le 1 ] && echo "表达式成立" || echo "表达式失败"

表达式成立

[root@oldboyedu ~]# [ 1 -lt 1 ] && echo "表达式成立" || echo "表达式失败"

表达式失败

[root@oldboyedu ~]# [ 1 -lt 2 ] && echo "表达式成立" || echo "表达式失败"

表达式成立

[root@oldboyedu ~]# [ 1 -gt 2 ] && echo "表达式成立" || echo "表达式失败"

表达式失败

[root@oldboyedu ~]# [ 3 -gt 2 ] && echo "表达式成立" || echo "表达式失败"

表达式成立

### 字符串判断

字符串需要加双引号

空格非空

-z 若字符串的长度为0，则为真

-n 若字符串的长度不为0，则为真

[root@oldboyedu ~]# a=

[root@oldboyedu ~]# [ -n "$a" ] && echo "表达式成立" || echo "表达式失败"

表达式失败

[root@oldboyedu ~]# a=' '

[root@oldboyedu ~]# [ -n "$a" ] && echo "表达式成立" || echo "表达式失败"

表达式成立

### 文件判断

#### 判断 /data/oldboy.txt 这个文件是否存在

[root@oldboy-01-nb ~]# [ -f /data/oldboy ]

[root@oldboy-01-nb ~]# echo $?

1

[root@oldboy-01-nb ~]# [ -f /oldboy/oldboy.txt ]

[root@oldboy-01-nb ~]# echo $?

0

#### 判断 /data/ 这个目录是否存在

[root@oldboy-01-nb ~]# [ -d /data ]

[root@oldboy-01-nb ~]# echo $?

0

[root@oldboy-01-nb ~]# [ -d /da ]

[root@oldboy-01-nb ~]# echo $?

1

#### 判断你输入的目录是否存在，如果不存在就创建他

[root@oldboy-01-nb ~]# vim /server/scripts/if.sh

#!/bin/bash

#filename:dir.sh

dir=$1

if [ ! -d $dir ];then

mkdir -p $dir

echo $dir create

fi

[root@oldboy-01-nb ~]# sh /server/scripts/if.sh /yy

/yy create

小结：

dir=$1

if [ ! -d $dir ];then ###如果这个目录不存在 那么

mkdir -p $dir ###创建这个目录

echo $dir create ###显示已经创建好了 目录 dir

fi

### 逻辑符号

与 -a

或 -o

非 !

# if语句

条件表达式和if语句可以互相转换

## 单分支语句

[ -f /etc/hosts ] && echo '文件存在'

if [ -f /etc/hosts ]

then

echo '文件存在'

fi

### if语句格式

if 条件

然后

命令

结束

## 双分支语句

[ -f /etc/hosts ] && echo '文件存在'||echo '文件不存在'&& echo '...' >>/test.log

if [ -f /etc/hosts ]

then

echo '文件存在'

else

echo '文件不存在'

echo '...' >>/test.log

fi

## 多分支语句

if [ -f /etc/hosts ]

then

echo 'hosts文件存在'

elif [ -f /etc/host ]

then

echo 'host文件存在'

elif [ -f /etc/my.cnf ]

then

echo 'my.cnf文件存在'

else

echo '你要的文件都没有'

fi

## 总结：

多分支语句：多个条件多个结果

双分支语句：一个条件2个结果

单分支语句：一个条件1个结果

## 企业案例练习题

### 两个数字比较大小

#### 传参的方式

#!/bin/bash

##############################################################

# File Name: lx.sh

# Version: V1.0

# Author: yyc

# Organization: www.oldboyedu.com

# Created Time : 2017-10-27 20:25:16

# Description:

##############################################################

if [ $# -ne 2 ]

then

echo "USAGE: sh $0 num1 num2"

exit 1

fi

num1=$1

num2=$2

expr 1 + $num1 &>/dev/null

if [ $? -eq 2 ]

then

echo "$num1不是整数"

exit 1

fi

expr 1 + $num2 &>/dev/null

if [ $? -eq 2 ]

then

echo "$num2不是整数"

exit 1

fi

if [ $num1 -gt $num2 ]

then

echo "$num1 > $num2"

elif [ $num1 -eq $num2 ]

then

echo "$num1 = $num2"

else

echo "$num1 < $num2"

fi

#### 交互式

#!/bin/bash

##############################################################

# File Name: lx2.sh

# Version: V1.0

# Author: yyc

# Organization: www.oldboyedu.com

# Created Time : 2017-10-28 06:02:38

# Description:

##############################################################

read -p "请输入第一个整数" num1

read -p "请输入第二个整数" num2

expr 1 + $num1 &>/dev/null

if [ $? -eq 2 ]

then

echo "$num1不是整数"

exit 1

fi

expr 1 + $num2 &>/dev/null

if [ $? -eq 2 ]

then

echo "$num2不是整数"

exit 1

fi

if [ $num1 -gt $num2 ]

then

echo "$num1 > $num2"

exit

elif [ $num1 -eq $num2 ]

then

echo "$num1 = $num2"

exit

else

echo "$num1 < $num2"

fi

### 系统内存低于100M邮件报警,加入计划任务，3分钟检查一次。

#### 对于开发程序而言，一般来说应该遵循下面的3步法则。

（1）分析需求

明白开发需求，是完成程序的大前提，因此，分析需求至关重要，一切不以需求为主的程序开发，都是耍流氓的！

（2）设计思路

设计思路就是根据需求，把需求进行拆解，分模块逐步实现，例如本题可以分为如下几步：

1）获取当前系统剩余内存的值（先在命令行实现）。

2）配置邮件报警（可采用第三方邮件服务器）。

3）判断取到的值是否小于100MB，如果小于100MB，就报警（采用if语句）。

4）编码实现Shell脚本。

5）加入crond定时任务，每三分钟检查一次。

（3）编码实现

编码实现就是具体的编码及调试过程，工作中很可能需要先在测试环境下调试，调试好了，再发布到生产环境中。

#### 老师版本

[root@oldboy scripts]# cat free.sh

#!/bin/bash

source /etc/profile

UsedMem=`free -m|awk -F "[ ]+" 'NR==3 {print $NF}'`

if [ $UsedMem -lt 100 ]

then

echo "mem is not enough,$UsedMem."|mail -s "memory warning $(date +%F)" 11111111@qq.com

fi

[root@oldboy scripts]# crontab -l

#memory check

\*/3 \* \* \* \* /bin/sh /server/scripts/free.sh &>/dev/null

#### 自己书写板

#!/bin/bash

##############################################################

# File Name: /server/scripts/free.sh

# Version: V1.0

# Author: yyc

# Organization: www.oldboyedu.com

# Created Time : 2017-10-28 06:44:07

# Description:

##############################################################

Free=`free -m|awk 'NR==2 {print $6}'`

CHARS="使用内存 $Free"

if [ $Free -lt 100 ]

then

echo $CHARS|tee /tmp/messages.txt

mail -s "`date +%F-%T`$CHARS" yyc1503@163.com </tmp/messages.txt

fi

#### 发邮件修改

[root@centos7 scripts]# tail -2 /etc/mail.rc

set from=yyc1503@163.com smtp=smtp.163.com

set smtp-auth-user=yyc1503 smtp-auth-password=yyc1503 smtp-autp=login

### 模拟启动nginx脚本（if语句）

目标：启动nginx脚本

#### 思路：

a. 启动nginx：先检查语法，语法出错也停止；再判断80端口是否存在，端口存在就提示错误信息；不存在，就启动即可，最后显示启动特效

b. 停止nginx pkill nginx

c. 重启 ，分为优雅重启： -s reload （先判断进程是否存在）

和强制重启 先停止再启动

d. 查看状态：看端口，看进程

#### 脚本

#!/bin/sh

. /etc/init.d/functions

start\_nginx=/application/nginx/sbin/nginx

USAGE(){

echo "USAGE $0 {start|stop|restart}"

exit 1

}

if [ $# -ne 1 ]

then

USAGE

fi

if [ "$1" == "start" ]

then

$start\_nginx

action "start nginx " /bin/true

elif [ "$1" == "stop" ]

then

pkill nginx

action "stop nginx " /bin/true

elif [ "$1" == "restart" ]

then

pkill nginx

sleep 2

$start\_nginx

action "restart nginx " /bin/true

else

USAGE

fi

#### 自写

#监控nginx

/usr/sbin/nginx -t

if [ $? -ne 0 ]

then

echo "语法错误，请检查语法"

exit 1

fi

if [ `ss -lntup|grep -w 80|grep -v grep|wc -l` -lt 2 ]

then

/usr/sbin/nginx

echo "[启动完成]"

exit

fi

if [ `ss -lntup|grep -w 80|grep -v grep|wc -l` -ge 2 ]

then

/usr/sbin/nginx -s reload

echo "[重启成功]"

fi

### 企业Shell面试题12：Web及MySQL服务异常监测案例

#### 题目要求

a.检测端口80/3306 ss、netstat、lsof、telnet、nmap、nc

b.通过进程判断 ps -ef|grep nginx ps -ef|grep mysql

注意：脚本名不能和进程名重复

c.模拟用户的方法

简单：通过返回值：

网页200，

mysql：mysql -uroot -poldboy123 -e "select version();"

精确：通过具体内容

网页：创建一个测试页面，访问测试页面的内容

数据库：随机插入数据，然后读出来。对比

#### 脚本

. /etc/init.d/functions

if [ `ss -lntup|grep -w 80|grep -v grep|wc -l` -lt 2 ]

then

action "nginx" /bin/false

else

action "nginx" /bin/true

fi

if [ `ss -lntup|grep -w 3306|grep -v grep|wc -l` -lt 1 ]

then

action "mysql" /bin/false

else

action "mysql" /bin/true

fi

# case语句

## 语法

case 如果，假设

case "字符串变量" in

值1)

指令1

;;

值2)

指令2

;;

\*)

指令

esac

### nginx启动脚本

case "$1" in

start)

/application/nginx/sbin/nginx

;;

stop)

pkill

;;

\*)

echo "usage: sh $0 start|stop"

esac

#注：case语句相当于一个if的多分支结构语句

### 值大小写

值1的选项

apple)

echo -e "$RED\_COLOR apple $RES"

;;

也可以这样写，输入2种格式找同一个选项

apple|APPLE)

echo -e "$RED\_COLOR apple $RES"

;;

## case与if什么时候用

if [ $num -eq `ps -ef|grep nginx|wc -l` ] 这种就不能用case

### case语句小结

 case语句就相当于多分支的if语句。case语句的优势是更规范、易读。

 case语句适合变量的值少，且为固定的数字或字符串集合。(1,2,3)或(start,stop,restart)。

 系统服务启动脚本传参的判断多用case语句，多参考rpcbind/nfs/crond脚本

菜单脚本也用case

## case应用场景

写服务启动脚本，菜单型脚本就用case

除此之外都用if语句

## 菜单练习题

### 水果菜单

#### 颜色选择

echo -e "\033[30m 黑色字oldboy trainning \033[0m"

echo -e "\033[31m 红色字oldboy trainning \033[0m"

echo -e "\033[32m 绿色字oldboy trainning \033[0m"

echo -e "\033[33m 黄色字oldboy trainning \033[0m"

echo -e "\033[34m 蓝色字oldboy trainning \033[0m"

echo -e "\033[35m 紫色字oldboy trainning \033[0m"

echo -e "\033[36m 天蓝字oldboy trainning \033[0m"

echo -e "\033[37m 白色字oldboy trainning \033[0m"

echo -e "\033[40;37m 黑底白字 welcome to old1boy\033[0m"

echo -e "\033[41;37m 红底白字 welcome to old2boy\033[0m"

echo -e "\033[42;37m 绿底白字 welcome to old3boy\033[0m"

echo -e "\033[43;37m 黄底白字 welcome to old4boy\033[0m"

echo -e "\033[44;37m 蓝底白字 welcome to old5boy\033[0m"

echo -e "\033[45;37m 紫底白字 welcome to old6boy\033[0m"

echo -e "\033[46;37m 天蓝白字 welcome to old7boy\033[0m"

echo -e "\033[47;30m 白底黑字 welcome to old8boy\033[0m"

#### 脚本书写

cat <<EOF

==================

1.apple

2.pear

3.banana

4.exit

==================

EOF

read -p "请选择你要的水果：" choice

case "$choice" in

1)

echo -e "\033[31m apple \033[0m"

;;

2)

echo -e "\033[32m peal \033[0m"

;;

3)

echo -e "\033[33m banana \033[0m"

;;

\*)

echo -e "\033[34m 其他 \033[0m"

esac

### 企业Shell面试题11：菜单自动化软件部署经典案例

[root@backup scripts]# cat 11.sh

#!/bin/bash

function menu(){

cat <<-EOF

1.[install lamp]

2.[install lnmp]

3.[exit]

EOF

}

function main(){

menu

read -p "pls input the num you want:" choose

case $choose in

1)

lamp=/server/scripts/lamp.sh

[ -f $lamp ] && sh $lamp

;;

2)

lnmp=/server/scripts/lnmp.sh

[ -f $lnmp ] && sh $lnmp

;;

3)

exit

;;

\*)

echo "Input error!" && exit 1

esac

}

main

### 企业Shell面试题15：开发Rsync服务启动脚本案例

. /etc/init.d/functions

USAGE(){

ehco "USAGE $0 {start|restart|stop}"

}

if [ $# -ne 1 ]

then

echo "USAGE"

fi

cat <<-EOF

start

restart

stop

EOF

port=`ss -lntup|grep -w 873|grep -v grep|wc -l`

read -p "选择状态：" state

case $state in

start)

rsync --daemon

if [ $port -ge 2 ]

then

echo "Started, no repetitive operation"

fi

;;

restart)

pkill rsync

sleep 2

rsync --daemon

if [ $port -lt 2 ]

then

echo "Stop status start first"

fi

;;

stop)

pkill rsync

if [ $port -lt 2 ]

then

echo "Stop status start first"

fi

;;

\*)

echo "USAGE $0 {start|restart|stop}"

esac

### 监控nginx状态

cat <<EOF

==========

start

restart

stop

==========

EOF

USAGE(){

echo "USAGE $0 {start|restart|stop}"

}

if [ $# -ne 1 ]

then

echo "USAGE"

fi

port=`ss -lntup|grep -w 80|grep -v grep|wc -l`

read -p "请输入需要执行Nginx的状态:" state

case $state in

start)

/usr/sbin/nginx &>/dev/null

if [ $port -ge 2 ]

then

echo "nginx已经启动，请勿重复启动"

fi

;;

restart)

pkill nginx

sleep 2

/usr/sbin/nginx

if [ $port -lt 2 ]

then

echo "nginx尚未启动，请先启动"

fi

;;

stop)

pkill nginx

if [ $port -lt 2 ]

then

echo "nginx处于停止状态,请先启动"

fi

;;

\*)

echo "USAGE $0 {start|restart|stop}"

esac

# for循环

## for循环语句

### 简写：for i in {1..10};do echo $i;done

for i in {1..10}

do

echo $i

done

### 格式

for i in 变量取值列表（有n个值，就给i赋值n次，同理循环n次）

do

执行命令

done

注：i 是 index（索引） 缩写

### 变量取值列表 写法：

a. 1 2 3 4

b. {1..4}

c.文件list.txt

1

2

3

4

for i in `cat list.txt`

d. 从命令执行结果赋值

for i in `ls`

## for 变量 in 列表

#### 显示1 2 3 4 5 .... 10

[root@oldboy-01-nb ~]# for num in 1 2 3 4 5 6 7 8 9 10

> do

> echo $num

> done

1

2

3

4

5

6

7

8

9

10

[root@oldboy-01-nb ~]# for num in {1..10}

> do

> echo $num

> done

1

2

3

4

5

6

7

8

9

10

#### 显示命令，跟啥就显示啥

[root@oldboy-01-nb ~]# for num in awk sed ls mkdir cp rm mv

> do

> echo $num

> done

awk

sed

ls

mkdir

cp

rm

mv

## 优化linux开机启动项目

### 第一个里程碑明确目标-关闭不必要的 除了network crond sshd rsyslog sysstat

[root@oldboy-01-nb ~]# chkconfig|egrep "sshd|crond|network|rsyslog|sysstat"

crond 0:off 1:off 2:on 3:on 4:on 5:on6:off

network 0:off 1:off 2:on 3:on 4:on 5:on6:off

rsyslog 0:off 1:off 2:on 3:on 4:on 5:on6:off

sshd 0:off 1:off 2:on 3:on 4:on 5:on6:off

sysstat 0:off 1:on 2:on 3:on 4:on 5:on6:off

### 第二个里程碑-使用循环之后 我们要的样子

[root@oldboy-01-nb ~]# chkconfig |egrep -v "sshd|crond|network|rsyslog|sysstat"

abrt-ccpp 0:off 1:off 2:off 3:on 4:off 5:on6:off

abrtd 0:off 1:off 2:off 3:on 4:off 5:on6:off

acpid 0:off 1:off 2:on 3:on 4:on 5:on6:off

atd 0:off 1:off 2:off 3:on 4:on 5:on6:off

auditd 0:off 1:off 2:on 3:on 4:on 5:on6:off

blk-availability 0:off 1:on 2:on 3:on 4:on5:on 6:off

cpuspeed 0:off 1:on 2:on 3:on 4:on 5:on6:off

haldaemon 0:off 1:off 2:off 3:on 4:on 5:on6:off

ip6tables 0:off 1:off 2:on 3:on 4:on 5:on6:off

iptables 0:off 1:off 2:off 3:off 4:off 5:of6:off

irqbalance 0:off 1:off 2:off 3:on 4:on 5:on6:off

kdump 0:off 1:off 2:off 3:on 4:on 5:on6:off

lvm2-monitor 0:off 1:on 2:on 3:on 4:on 5:on6:off

mdmonitor 0:off 1:off 2:on 3:on 4:on 5:on6:off

messagebus 0:off 1:off 2:on 3:on 4:on 5:on6:off

netconsole 0:off 1:off 2:off 3:off 4:off 5:of6:off

netfs 0:off 1:off 2:off 3:on 4:on 5:on6:off

nfs-rdma 0:off 1:off 2:off 3:off 4:off 5:of6:off

ntpd 0:off 1:off 2:off 3:off 4:off 5:of6:off

ntpdate 0:off 1:off 2:off 3:off 4:off 5:of6:off

postfix 0:off 1:off 2:on 3:on 4:on 5:on6:off

psacct 0:off 1:off 2:off 3:off 4:off 5:of6:off

quota\_nld 0:off 1:off 2:off 3:off 4:off 5:of6:off

rdisc 0:off 1:off 2:off 3:off 4:off 5:of6:off

rdma 0:off 1:on 2:on 3:on 4:on 5:on6:off

restorecond 0:off 1:off 2:off 3:off 4:off 5:of6:off

rngd 0:off 1:off 2:off 3:off 4:off 5:of6:off

saslauthd 0:off 1:off 2:off 3:off 4:off 5:of6:off

smartd 0:off 1:off 2:off 3:off 4:off 5:of6:off

svnserve 0:off 1:off 2:off 3:off 4:off 5:of6:off

udev-post 0:off 1:on 2:on 3:on 4:on 5:on6:off

### 第三个里程碑-取出服务的名字

[root@oldboy-01-nb ~]# chkconfig |egrep -v "sshd|crond|network|rsyslog|sysstat"|awk '{print $1}'

abrt-ccpp

abrtd

acpid

atd

auditd

blk-availability

cpuspeed

haldaemon

ip6tables

iptables

irqbalance

kdump

lvm2-monitor

mdmonitor

messagebus

netconsole

netfs

nfs-rdma

ntpd

ntpdate

postfix

psacct

quota\_nld

rdisc

rdma

restorecond

rngd

saslauthd

smartd

svnserve

udev-post

### 第四个里程碑-得到结果

[root@oldboy-01-nb ~]# for fu in $(chkconfig |egrep -v "sshd|crond|network|rsyslog|sysstat"|awk '{print $1}' )

> do

> # chkconfig $fu off

> echo $fu

> done

abrt-ccpp

abrtd

acpid

atd

auditd

blk-availability

cpuspeed

haldaemon

ip6tables

iptables

irqbalance

kdump

lvm2-monitor

mdmonitor

messagebus

netconsole

netfs

nfs-rdma

ntpd

ntpdate

postfix

psacct

quota\_nld

rdisc

rdma

restorecond

rngd

saslauthd

smartd

svnserve

udev-post

## 企业Shell面试题1：批量生成随机字符文件名案例

难点1： 生成随机字母

uuidgen tr sed

[root@oldboyedu scripts]# cat 30-1.sh

#!/bin/bash

##############################################################

# File Name: 30-1.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 10:15:10

# Description:

##############################################################

dir=/oldboy

[ -d $dir ]|| mkdir -p $dir

cd $dir &&\

for i in {1..10}

do

ranStr=`uuidgen|tr '0-9-' 'a-z'|cut -c -10`

touch ${ranStr}\_oldboy.html

done

## 企业Shell面试题2：批量改名特殊案例

[root@oldboyedu scripts]# cat 30-2.sh

#!/bin/bash

##############################################################

# File Name: 30-2.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 10:39:40

# Description:

##############################################################

dir=/oldboy

cd $dir || exit 1

for i in `ls \*html`

do

ranStr=`echo $i|cut -c -10`

mv $i ${ranStr}\_oldgirl.HTML

done

## 企业Shell面试题3：批量创建特殊要求用户案例

批量创建10个系统帐号oldboy01-oldboy10并设置密码（密码为随机数，要求字符和数字等混合）。

a. 使用for循环

b. useradd oldboy01

c. 随机密码 uuidgen

d. 记录密码！

e. 友好提示

[root@oldboyedu scripts]# cat 30-3.sh

#!/bin/bash

##############################################################

# File Name: 30-3.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 11:09:21

# Description:

##############################################################

. /etc/init.d/functions

dbFile=/tmp/userlist

for i in oldboy{11..20}

do

useradd $i &>/dev/null

if [ $? -eq 0 ]

then

action "$i" /bin/true

else

action "$i" /bin/false

continue

fi

pass=`uuidgen`

echo $pass|passwd --stdin $i &>/dev/null

echo "$i:$pass" >> $dbFile

done

### continue和exit

[root@oldboyedu scripts]# cat continue.sh

#!/bin/bash

#############################################################

# File Name: continue.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 11:22:33

# Description:

#############################################################

for i in {1..5}

do

if [ $i -eq 3 ] ;then

continue

fi

echo $i

done

echo "ok"

### exit

[root@oldboyedu scripts]# cat exit.sh

#!/bin/bash

##############################################################

# File Name: continue.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 11:22:33

# Description:

##############################################################

for i in {1..5}

do

if [ $i -eq 3 ] ;then

exit

fi

echo $i

done

echo "ok"

## 企业Shell面试题4：扫描网络内存活主机案例

### 串行

. /etc/init.d/functions

for i in 10.0.0.{1..254}

do

ping -c 1 -W 1 $i &>/dev/null

[ $? -eq 0 ] && action "$i" /bin/true

done

### 并行

. /etc/init.d/functions

for i in 10.0.0.{1..254}

do

ping -c 1 -W 1 $i &>/dev/null && action "$i" /bin/true &

done

## 企业Shell面试题8：筛选符合长度的单词案例

### 一般写法

[root@backup scripts]# cat 8.sh

#!/bin/sh

word="I am oldboy teacher welcome to oldboy trainingclass"

for i in $word

do

# [ ${#i} -le 6 ] && echo $i

# [ `echo $i|wc -L` -le 6 ] && echo $i

[ `expr length "$i"` -le 6 ] && echo $i

done

### 切割法

[root@db03 scripts]# cat 30-8.sh

#!/bin/bash

zimu="I am oldboy teacher welcome to oldboy trainingclass"

for i in $zimu

do

zifu=`echo $i|cut -c 7`

if [ -z "$zifu" ]

then

echo $i

fi

done

## 企业Shell面试题18：破解RANDOM随机数案例

### 方法1：生成密码字典，然后查字典

if [ ! -f /tmp/pass.txt ]

then

for i in `seq 0 32767`

do

a=`echo $i|md5sum`

echo "$i:$a" >>/tmp/pass.txt

done

fi

grep "$1" /tmp/pass.txt|awk -F ":" '{print $1}'

### 方法2：实时加密实时对比

#!/bin/bash

##############################################################

# File Name: ranDOM.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-01 23:55:54

# Description:

##############################################################

for i in {1..32767}

do

for o in 21029299 00205d1c a3da1677 1f6d12dd 890684b

do

if [ `echo $i |md5sum |grep $o|wc -l` -ne 0 ]

then

echo "$i:$o"

fi

done

done

### 自写方法

read -p "输入字符串:" a

txt=/tmp/md5.txt

if [ -f $txt ]

then

echo `grep $a $txt`

else

for i in {0..32767}

do

num="$i `echo $i|md5sum`"

echo $num>>$txt

done

echo `grep $a $txt`

fi

## 企业Shell面试题28：51CTO博文爬虫案例

难点1：html语法，从文件中提取url地址

难点2：中文乱码 iconv -f gb2312 -t utf-8 -c index.html

# while循环语句

## ;语法

while 条件表达式

（表达式成立，继续执行循环，表达式不成立，立即退出循环）

do

命令

done

## 死循环 守护进程

while true

do

uptime

done

### 作用：

a. 补充定时任务功能，执行小于1分钟的定时任务

b. 循环执行某些操作，比如我们的水果菜单。

## 数学运算

sum=0

i=1

while [ $i -le 100 ]

do

((sum=sum+i))

((i++))

done

echo $sum

## 手机充值10元，每发一条短信(输出当期余额)花费1角5分，当余额低于1角5分则不能发短信，提示余额不足，请充值。

a=1000

i=15

while [ $a -gt 15 ]

do

printf "

1. 发短信

2. 查询余额

3. 退出

"

read -p "选择业务：" num

case $num in

1)

echo "发送成功"

((a=a-i))

[ $a -lt 15 ] && echo "余额不足"

;;

2)

echo "余额为：$a"

;;

3)

exit

;;

esac

done

## 防止脚本执行中断的方法

screen 分离ctrl+a+d 查看 screen -ls 进入screen -r num

[root@oldboyedu ~]# screen -ls

There is a screen on:

5938.pts-2.oldboyedu (Detached)

1 Socket in /var/run/screen/S-root.

[root@oldboyedu ~]# screen -r 5938

## 获取文件中的行、单词和字符

### 迭代文件中的每一行

#### 使用while循环从标准输入获取

while read line

do

echo $line

done < file.txt

#### 使用子shell方式

cat file.txt|while read line

do

echo $line

done

#### exec

exec < file.txt

while read line

do

echo $line

done

### 案例：迭代文件每一行，一行中每一个单词，每个单词每个字母

while read line

do

echo $line

sleep 2

for word in $line

do

echo $word

sleep 1

for((i=0;i<${#word};i++))

do

echo ${word:i:1}

done

done

done </etc/hosts

#### 老师版本

遍历hosts文件

[root@oldboyedu scripts]# cat diedai.sh

#!/bin/bash

##############################################################

# File Name: diedai.sh

# Version: V1.0

# Author: zhang yao

# Organization: www.oldboyedu.com

# Created Time : 2017-11-02 11:12:15

# Description:

##############################################################

file=/etc/hosts

[ -f $file ]|| exit 1

i=1

while read line

do

echo "第$i行：$line"

m=1

for x in $line

do

echo "第$i行第$m个单词：$x"

echo $x|grep -o .

((m++))

done

((i++))

done < $file

## break /continue/ exit/ return 区别

break exit return 都有中断的功能：

break只能在循环中起作用

其作用是：跳出整个循环，继续循环外后面的程序执行。

continue

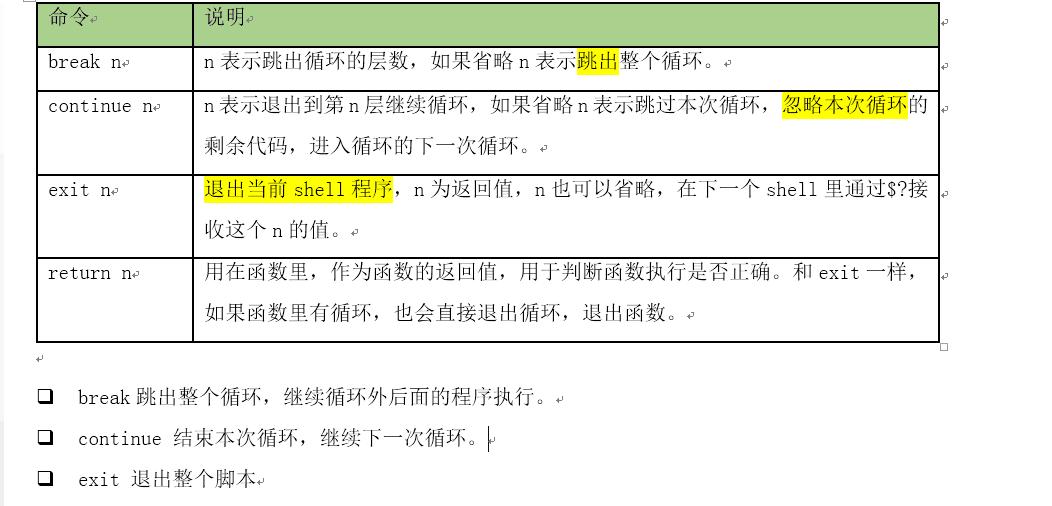
结束本次循环，继续下一次循环。

exit 在任何地方都能生效

退出整个脚本

return只能在函数中生效

退出函数



### break

执行脚本

for i in {1..5}

do

if [ $i -eq 3 ] ;then

break

fi

echo $i

done

echo "ok"

结果：结束当前循环

[root@centos7 ~]# sh break.sh

1

2

ok

### continue

执行脚本

for i in {1..5}

do

if [ $i -eq 3 ] ;then

continue

fi

echo $i

done

echo "ok"

结果：退出当前参数，执行下一个参数

[root@centos7 ~]# sh continue.sh

1

2

4

5

ok

### exit

执行脚本

for i in {1..5}

do

if [ $i -eq 3 ] ;then

exit

fi

echo $i

done

echo "ok"

结果：exit 直接退出

[root@centos7 ~]# sh exit.sh

1

2

### return

退出当前函数

## 读取进度条脚本

b=''

i=0

while [ $i -le 100 ]

do

printf "progress:[%-50s]%d%%\r" $b $i

sleep 0.1

i=`expr 2 + $i`

b=#$b

done

echo

# 数组、函数

## Shell数组

### 定义数组

#### # 一对括号表示是数组，数组元素用“空格”符号分隔开

[root@oldboyedu ~]# array=(1 2 3 4 5)

#### 查看数组长度

[root@oldboyedu ~]# echo ${#array[@]}

5

### 查看数组内容 <==数组下标从0开始，${数组名[下标]}

#### 查看数组所有内容

[root@oldboyedu ~]# echo ${array[\*]}

1 2 3 4 5

[root@oldboyedu ~]# echo ${array[@]}

1 2 3 4 5

#### 查看数组内第几个数字，系统默认从0位开始

[root@oldboyedu ~]# echo ${array[2]}

3

[root@oldboyedu ~]# echo ${array[0]}

1

[root@oldboyedu ~]# echo ${array[43]} 如果查看的位数不存在就显示空

[root@oldboyedu ~]# echo ${array[4]}

5

### 遍历数组

#### 方法1：

[root@oldboyedu ~]# for i in ${array[\*]};do echo $i;done

1

2

3

4

5

#### 方法2：

[root@oldboyedu ~]# for i in `eval echo {1..${#array[@]}}`;do echo ${array[i-1]};done

1

2

3

4

5

注：eval echo {1..${#array[@]}} 输出 1 2 3 4 5

${array[i-1]} i-1 ==》 0 1 2 3 4

==》${array[0]} ${array[1]}${array[2]}${array[3]} ${array[4]}

#### 方法3：

sz=(a b c d e f g h i j)

for i in `seq ${#sz[\*]}`

do

((i--))

echo ${sz[$i]}

done

#### 方法4：

for ((i=0;i<${#array[\*]};i++)); do echo ${array[i]}; done

### 定义值与删除数组元素

#### 定义固定数组

[root@oldboyedu ~]# array[5]=10

[root@oldboyedu ~]# echo ${array[5]}

10

系统默认从0位开始，所以第5位是10

[root@oldboyedu ~]# echo ${array[@]}

1 2 3 4 5 10

#### 删除数组

[root@centos7 scripts]# echo ${array[\*]}

1 2 3 4 5 10

[root@oldboyedu ~]# unset array[2]

[root@oldboyedu ~]# echo ${array[@]}

1 2 4 5 10

[root@oldboyedu ~]# echo ${array[2]} # 删除元素的坑依然保留 #仍然将原来的位置空出来

[root@oldboyedu ~]# array[2]=10

[root@oldboyedu ~]# echo ${array[@]}

1 2 10 4 5 10

[root@oldboyedu ~]# unset array

[root@oldboyedu ~]# echo ${array[@]}

##此处为空位

### 数组内容的截取和替换

[root@oldboy ~]# array=(1 2 3 4 5)

#### 截取 array[@]:起始值：截取长度

[root@oldboy ~]# echo ${array[@]:1:3}

2 3 4

[root@oldboy ~]# echo ${array[@]:3:2}

4 5

[root@backup logs]# echo ${array1[@]::2}

a b

[root@backup logs]# echo ${array1[@]:2}

c d e

#### 替换 array[@]/数组原有内容/想要替换的内容

[root@oldboy ~]# echo ${array[@]/5/6}

1 2 3 4 6

[root@oldboy ~]# echo ${array[@]}

1 2 3 4 5

[root@oldboy ~]# echo ${array[@]/5/zy}

1 2 3 4 zy

#### 替换操作不会改变原先数组内容，如需修改，则要重新定义数组

[root@oldboy ~]# array1=${array[@]/5/zy}

[root@oldboy ~]# echo ${array1[@]}

1 2 3 4 zy

#### 数组也是变量，适合于变量的子串处理功能

#帮助 man bash 搜寻Arrays

### 将命令的结果赋值给数组

[root@oldboyedu ~]# array=(`ls`)

[root@oldboyedu ~]# echo ${array[@]}

作用：将文件内容全部读取到内存中

[root@oldboyedu ~]# log=(`cat /etc/hosts`)

[root@oldboyedu ~]# echo ${log[@]}

### 数组定义格式

array=(red green blue yellow magenta)

或

array=(

oldboy1

oldboy2

oldboy3

)

### 数组重要知识小结

数组本质就是一个变量，只是这个变量存储多个值

#### 定义

array=(1 2 3)

array=($(ls))

#### 打印

${array[@]} 所有元素

${#array[@]} 数组长度

${array[i]}单个元素，i是下标

### 企业Shell面试题19：批量检查多个网站地址是否正常

. /etc/init.d/functions

arry=(

http://blog.oldboyedu.com

http://oldboy.blog.51cto.com

)

while true

do

for i in ${arry[\*]}

do

a=`curl -s -I $i|awk 'NR==1{print $2}'`

if [ $a -eq 200 ]

then

action "url" /bin/true

else

action "url" /bin/false

fi

done

sleep 10

done

## Shell函数

### 函数简介

函数能够让你的脚本更加简洁，清晰明了，更有逼格

函数的作用就是把程序里多次调用相同代码的部分定义成一份，然后起个名字，所有的调用都只用这个名字就可以了。修改代码时，只需要改变函数体内的代码即

### Shell函数语法

#### 简单的语法

函数名(){

指令...

return n

}

oldgirl(){

echo "Hello,boy"

}

#### 规范的语法

function 函数名(){

指令...

return n

}

function oldboy(){

echo 'Hello World'

}

### 函数执行

#### 直接执行函数名即可

oldboy

oldgirl

注意：

执行函数时，不要带小括号。

函数定义及函数体必须在要执行的函数名的前面定义，Shell的执行是从上到下的。

#### 函数传参

[root@oldboyedu scripts]# cat fun1.sh

function oldboy(){

echo "Hello $1 World $2"

}

oldgirl(){

echo "Hello,boy"

}

echo $1 $2

oldboy action true

oldboy $1 $2

"fun1.sh" 25L, 555C 已写入

[root@oldboyedu scripts]# sh fun1.sh arg1 arg2

arg1 arg2

Hello action World true

Hello arg1 World arg2

### 脚本加载自定义函数库原理

funFile=/server/scripts/fun1.sh

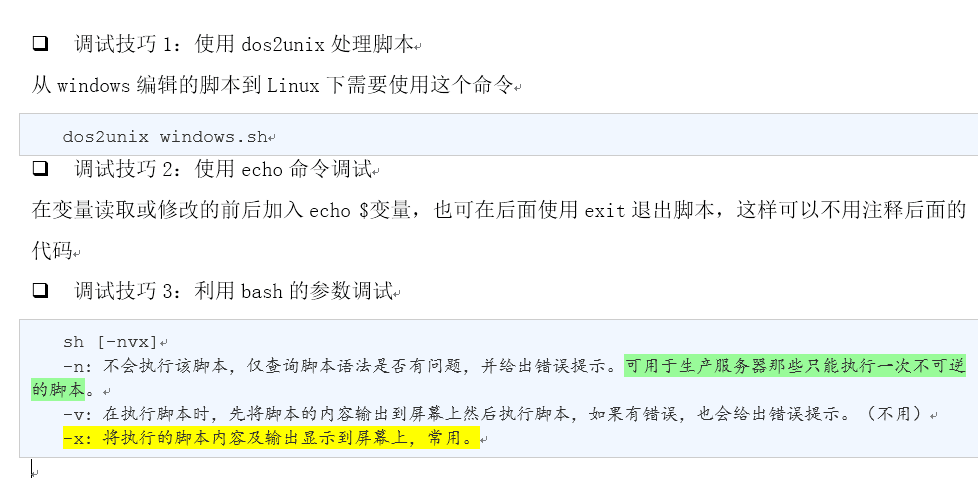
[ -f $funFile ] && . $funFile

oldboy

### 怎样能够在脚本中使用函数

需要将脚本的功能分析，拆开，模块化

# shell脚本的调试



## dos2unix命令处理在Windows下开发的脚本

在windos下开发的脚本，明明检查没有发现问题，但就是执行时会出现莫名其妙的语法错误。这时最好执行dos2unix 格式化一下。

## 使用echo命令调试

在变量读取或者修改操作前后加入echo命令，并紧挨着退出命令exit

## 使用bash命令参数调试

参数说明：

-n ：不会执行该脚本，仅查询脚本语法是否有问题，并给出错误提示

-v ：在执行脚本时，现将脚本的内容输出到屏幕上，然后执行脚本，如果有错误，也会给出错误提示

-x ：将执行的脚本内容及输出显示到屏幕上，这是对调试很有用的参数

### 参数-n的测试

#### 查看脚本内容

[root@centos7 ~]# cat sh.sh

#!/bin/bash

echo "hello $USER"

echo "today is $(date +%F)"

#### sh -n 检查语法

[root@centos7 ~]# sh -n sh.sh

[root@centos7 ~]#

没有输出，语法没错

#### 修改后检查

[root@centos7 ~]# cat sh.sh

#!/bin/bash

echo "hello $USER 将此处双引号去掉，再检查

echo "today is $(date +%F)"

检查语法

[root@centos7 ~]# sh -n sh.sh

sh.sh:行11: 寻找匹配的 `"' 是遇到了未预期的文件结束符

sh.sh:行12: 语法错误: 未预期的文件结尾

### -v 参数

[root@centos7 ~]# sh -v sh.sh

#!/bin/bash

echo "hello $USER

echo "today is $(date +%F)"

sh.sh:行11: 寻找匹配的 `"' 是遇到了未预期的文件结束符

sh.sh:行12: 语法错误: 未预期的文件结尾

### -x参数

[root@centos7 ~]# sh -x sh.sh

sh.sh:行11: 寻找匹配的 `"' 是遇到了未预期的文件结束符

sh.sh:行12: 语法错误: 未预期的文件结尾

## 使用set命令调试

set -n :读命令但并不执行

set -v: 显示读取的所有行

set -x: 显示所有命令及参数

通过set -x 开启调试，set +x 关闭调试

