目录：

目录

[1.1、shell脚本简介 2](#_Toc440624722)

[1.1.1 什么是shell？ 2](#_Toc440624723)

[1.1.2 什么是shell脚本？ 2](#_Toc440624724)

[1.1.3 shell脚本在运维工作中的作用地位？ 4](#_Toc440624725)

[1.1.4 脚本语言的种类 5](#_Toc440624726)

[1.1.5 常用操作系统的默认shell 7](#_Toc440624727)

[1.1.6 shell脚本的建立和执行 8](#_Toc440624728)

[1.2 shell变量基础及深入 14](#_Toc440624729)

[1.2.1 什么是变量？ 14](#_Toc440624730)

[1.2.2 变量类型 15](#_Toc440624731)

[1.2.3 环境变量 15](#_Toc440624732)

[1.2.4 自定义环境变量（全局变量） 16](#_Toc440624733)

[1.2.5 显示与取消环境变量 18](#_Toc440624734)

[1.2.6 局部变量 44](#_Toc440624735)

[1.3 条件测试与比较 51](#_Toc440624736)

[1.3.1 条件测试与比较 51](#_Toc440624737)

[1.3.2 文件测试表达式 54](#_Toc440624738)

[1.3.3 字符串测试表达式 59](#_Toc440624739)

[1.3.4 整数二元比较操作符 62](#_Toc440624740)

[1.3.5 逻辑操作符 64](#_Toc440624741)

[1.3.6 shell变量的输入 69](#_Toc440624742)

[1.4 分支与循环结构 71](#_Toc440624743)

[1.4.1 if条件句 71](#_Toc440624744)

[1.5 shell函数 76](#_Toc440624745)

[1.5.1 为什么要使用shell函数 76](#_Toc440624746)

[1.5.2 shell函数语法 77](#_Toc440624747)

[1.5.3 shell函数的执行 78](#_Toc440624748)

[1.5.4 shell函数范例： 79](#_Toc440624749)

[1.6 case结构条件句 81](#_Toc440624750)

[1.6.1 case结构条件句语法 81](#_Toc440624751)

[1.6.2 case结构条件句范例： 82](#_Toc440624752)

[1.7 当型循环和直到型循环 87](#_Toc440624753)

[1.7.1 和直到型循环语法 88](#_Toc440624754)

[1.7.2 当型和直到型循环基本范例： 88](#_Toc440624755)

[1.8 for循环结构\*\*\*\*\* 98](#_Toc440624756)

[1.8.1 for循环结构语法 98](#_Toc440624757)

[1.8.2 for循环结构基础例子 99](#_Toc440624758)

[1.8.3 企业面试重点题目： 101](#_Toc440624759)

[1.9 break continue exit return 104](#_Toc440624760)

[1.9.1 break continue exit对比 104](#_Toc440624761)

[1.9.2 break、continue、exit范例 105](#_Toc440624762)

[1.9.3 生产范例 106](#_Toc440624763)

[1.10 shell数组 107](#_Toc440624764)

[1.10.1 数组介绍 107](#_Toc440624765)

[1.10.2 数组定义与增删改查 108](#_Toc440624766)

[1.10.3 数组实践实战例子 111](#_Toc440624767)

**前文：**

**学习shell编程到底需要什么样的基础呢？**

shell脚本是实现Linux系统及运维自动化管理的重要且必要的工具，几乎每一个合格的Linux系统管理员或高级运维工程师，都需要熟练shell脚本语言的编写，只有这样才能提升运维工作的效率，解决工作中的重复劳动。那么，shell脚本编程难不难呢？学历shell编程到底需要什么样的Linux基础呢？

下面，就和大家说下，我认为的学习shell编程的基础，这里提到的基础并不是一定具备了才可以学下去，而是，具备了如下的基础，可以把shell学习的更好，领悟的更深。

学好shell编程并实现通过shell脚本自动化管理系统的必备基础：

1、vi/vim编辑器的熟练使用，SSH终端机“.vimrc”的设置等等需要搞熟练了。

2、命令基础：Linux150个常用命令的熟练使用。

3、Linux正则表达式以及三剑客（awk、grep、sed）要熟练。

4、常见Linux网络服务部署、优化及排错。例如：crond、NFS、rsync、inotify、lanmp、sersync、ssh、memcached等。

## 1.1、shell脚本简介

### 1.1.1 什么是shell？

shell是一个**命令解释器**，它在操作系统的**最外层**，负责直接与用户对话，把用户的输入解释给操作系统，并处理各种各样的操作系统的输出结果，输出到屏幕返回给用户。这种对话方式可以是交互的方式（从键盘输入命令，可以立即得到shell的回应），或非交互（脚本）的方式。

下图中的黄色部分就是命令解释器shell处于操作系统中的位置形象图解。



### 1.1.2 什么是shell脚本？

当Linux命令或语句不在命令行下执行（严格说，命令行执行的语句也是shell脚本）而是通过一个程序文件执行时，该程序就被称为shell脚本或shell程序，shell程序很类似DOS系统下的批处理程序（扩展名\*.bat）。用户可以在shell脚本中敲入一系列的命令及命令语句组合。这些命令、变量和流程控制语句等有机的结合起来就形成了一个功能强大的shell脚本。

下面是在Windows下利用批处理程序bat开发的备份网站及数据库数据的脚本。

echo off

set date=%date:~0,4%-%date:~5,2%-%date:~8,2%

mysqldump -uroot -poldboy -A -B > D:\bak\"%date%".sql

rar.exe a -k -r -s -ml D:\bak\"%date%".sql.rar D:\bak\"%date%".sql

del D:\bak\\*.sql

rar.exe a -k -r -s -ml D:\bak\"%date%"htdocs.rar D:\work\PHPnow\htdocs

范例1.清除/var/log下messages日志文件的简单命令脚本

#把所有命令放在一个文件里堆积起来就形成了脚本，下面就是一个最简单的命令堆积形成的shell脚本。

#要使用root身份来运行这个脚本。

#清楚日志脚本，版本1

cd /var/log

cat /dev/null > message

echo "Logs cleaned up."

提示： /var/log/messages 是系统的日志文件，很重要。

上述脚本的问题：

1、如果不是root用户就无法执行清理日志。

2、没有恩和流程控制语句，简单的说就是顺序操作，没有成功判断和逻辑严密性。

范例2. 包含命令、变量和流程控制语句的清楚/var/log下messages日志文件的shell脚本。

LOG\_DIR=/var/log

ROOT\_UID=0 #$UID为0的时候，用户才具有root用户的权限

#要使用root用户来运行.

if ["$UID" -ne "$ROOT\_UID"]

then

echo "Must be root to run this script."

exit 1

fi

cd $LOG\_DIR || {

echo "Cannot change to necessary directory." >&2

exit 1

}

cat /dev/null > messages && echo "Logs cleaned up."

exit 0

#退出之前返回0表示成功. 返回1表示失败。

类似打游戏的环节：

1、第一关，必须是root才能执行脚本，否则退出。

2、第三关，成功切换目录（cd /var/log），否则提出。

3、第三关，清理日志（cat /dev/null >messages），判断成功。

4、第四关，通关了。。。（echo输出）。

拓展：清空日志及文件内容的三种方法：

[root@db01 ~]# echo >test.log

[root@db01 ~]# > test.log

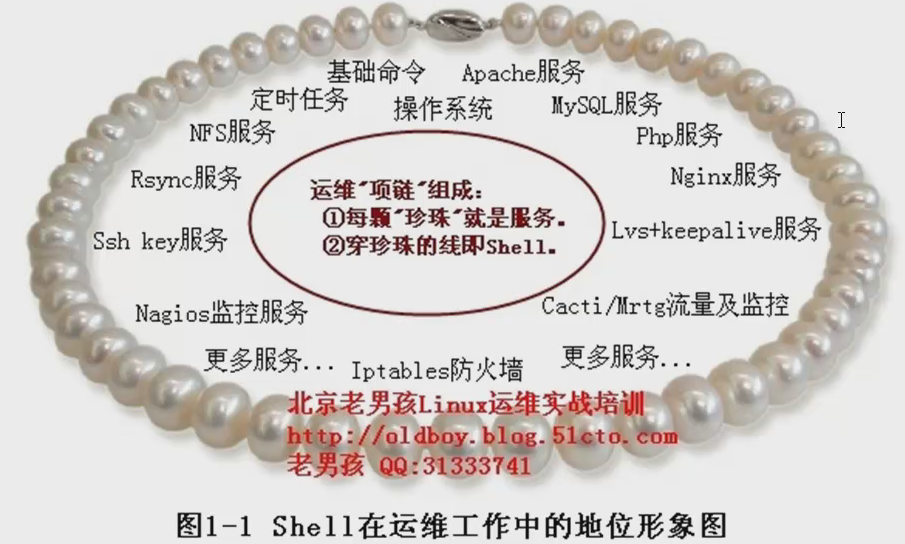
[root@db01 ~]# cat /dev/null > test.log

应用场景：保留文件，情况内容。

### 1.1.3 shell脚本在运维工作中的作用地位？

shell脚本很擅长处理**纯文本类型的数据**，而Linux系统中几乎所有的配置文件、文件（如nfs,rsync,httpd,nginx,lvs等）、多数启动文件都是纯文本类型的文件。因此，学号shell脚本语言，就可以利用它在Linux系统中发挥巨大的作用。

下面是shell脚本在运维工作中的作用地位形象图：



### 1.1.4 脚本语言的种类

#### 1.1.4.1 shell脚本语言的种类

在Unix/Linux中主要由两大类shell

Bourne shell（包括sh,ksh,andbash）

Bourne shell (sh)

Korn shell (ksh)

Bourne Again shell (bash)

POSIX shell (sh)

C shell （包括 csh and tcsh）

C shell （csh）

TENEX/TOPS C shell (tcsh)

shell脚本语言是弱类型语言，较为通用的shell有标准的Bourne shell（sh）和C shell（csh）。其中Bourne shell（sh）已经被bash shell取代，但是我们还是习惯称之为sh。

查看系统的shell：

[root@db01 ~]# cat /etc/shells

/bin/sh

/bin/bash

/sbin/nologin

/bin/dash

/bin/tcsh

/bin/csh

Linux系统中的主流shell是bash，它是Bourne again shell的缩写，bash是由Bourne shell发展而来的，但bash与sh稍有不同，它还包含了csh和ksh的特色，但大多数脚本都可以不加修改地在bash上运行。

#### 1.1.4.2 其他常用的脚本语言种类

1）PHP

PHP是**网页程序**，也是脚本语言。是一款更**专注于web页面开发（前端展示）的脚本语言**，例如：dedecms,discuz。PHP程序也可以处理系统日志，配置文件等，PHP也可以调用系统命令\*

2） Perl

Perl脚本语言。比shell脚本强大很多，2010年以前很火，语法灵活、复杂，实现方式很多，不易读，团队协作困难，但仍不失很好的脚本语言，存世大量的程序软件，运维人员了解就好了，无需学习这个，MHA高可用Perl写的。

3） Python

Python是近几年很火的语言，不但可以做脚本程序开发，也可以实现web程序以及软件的开发。近两年越来越多的公司多要求会Python。

#### 1.1.4.3 shell脚本与PHP/perl/python语言的区别和优势？

shell脚本的优势在于处理操作系统底层的业务（Linux系统内部的应用都是shell脚本完成），因为有大量的Linux系统命令为它做支撑，2000多个命令都是shell脚本编程的有力支撑，特别是grep、awk、sed等。例如：一键软件安装、优化，加农报警脚本。常规的业务应用，shell开发更简单快速，符合Linux运维的简单、一用、高效原则。

PHP，Python优势在于开发运维工具以及web界面的管理工具，web的开发\*处理一键软件安装、优化，报警脚本，常规的业务应用等PHP/Python也是能够租到\*是开发效率和复杂度比用shell就差很多了。我们使用软件就是要根据业务需求来选\*长避短。

### 1.1.5 常用操作系统的默认shell

Linux是Bourne again shell （bash）

Solaris和FreeBSD缺省的是Bourne shell （sh）。

AIX下是korn shell（ksh）。

HP-UX缺省的是POSIX shell （sh）。

提示：这里我们将重点讲Linux系统环境下的Bourne again shell （bash），centos5-6 下面的bash。

企业考试题一例：centos Linux系统默认的shell是（bash）

查看方法1：

[root@db01 ~]# echo $SHELL

/bin/bash

法2：

[root@db01 ~]# grep root /etc/passwd

root:x:0:0:root:/root:/bin/bash:

**更改shell默认的配置文件**

[root@db01 ~]# cat /etc/default/useradd

# useradd defaults file

GROUP=100

HOME=/home

INACTIVE=-1

EXPIRE=

SHELL=/bin/bash

SKEL=/etc/skel

CREATE\_MAIL\_SPOOL=yes

### 1.1.6 shell脚本的建立和执行

#### 1.1.6.1 shell脚本的建立

在Linux系统中，shell脚本（bash shell程序）通常是在编辑器（如vi/vim）中编写，由Unix/Linux命令、bash shell命令、程序结构控制语句和注释等内容组成，这里推荐用vim编辑器编写，可以实现做一个别名alias vi=’vim’。

[root@db01 ~]# echo "alias vi='vim'" >>/etc/profile

[root@db01 ~]# tail -1 /etc/profile

alias vi='vim'

[root@db01 ~]# . /etc/profile

脚本开头（第一行）

一个规范的shell脚本在脚本第一行会支出由哪个程序（解释器）来执行脚本中\*这一行内容在Linux bash编程中一般为：

#!/bin/bash

或

#!/bin/sh ===>255个字符以内

其中开头的“#！”字符又称为幻数，在执行bash脚本的时候，内核会根据“#！”解释器来确定改用哪个程序解释这个脚本中的内容。注意：这一行必须在每个脚本中的第一行，如果不是第一行则为脚本注释行，例如下面的例子。

[root@db01 ~]# cat test1.sh

#!/bin/bash

echo "oldboy start"

#!/bin/bash <==写到这里就是注释了

#!/bin/sh

echo "oldboy end"

sh和bash的区别

早起的bash与sh稍有不同，bash它还包含了csh和ksh的特色，但大多数脚本都可以 不加修改的在sh上运行。

[root@db01 ~]# ll /bin/sh

lrwxrwxrwx. 1 root root 4 12月 27 22:41 /bin/sh -> bash

[root@db01 ~]# ll /bin/bash

-rwxr-xr-x 1 root root 941720 7月 24 02:55 /bin/bash

提示：sh为bash的软链接，这里推荐用标准写法#！/bin/bash

提示：当使用/bin/sh执行脚本不正常的时候，可以使用/bin/bash执行。

bash的版本

[root@db01 ~]# bash --version

GNU bash, version 4.1.2(1)-release (x86\_64-redhat-linux-gnu)

Copyright (C) 2009 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

下面是Linux中常用脚本语言开头的编码写法，不同语言脚本的开头一般都要加相应语言的开头标识内容。

1、 #！/bin/sh

2、 #！/bin/bash

3、 #!/usr/bin/awk

4、 #!/bin/sed

5、 #!/usr/bin/tcl

6、 #!/usr/bin/expect

7、 #!/usr/bin/perl

8、 #!/usr/bin/env python

bash是GUN/Linux默认的shell，和Bourne shell（sh）兼容，bash采用了Korn shell和C shell（csh）的特色。符合IEEE POSIX P1003.2、ISO 9945.2 shell and tools标准。

**如果脚本的开头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。

如果是Python脚本，就用Python test.py执行。

如果是expect脚本，就用expect test.exp执行。

其他的脚本程序几乎都是类似的执行方法。

**脚本注释：**

在shell脚本中，跟在（#）井号后面的内容表示注释，用来对脚本进行注释说明注释不得呢不会被当做程序执行，仅仅是给用户看的，系统解释器是看不到的更不会执行，解释可自成一行，也可以跟在脚本命令后面与命令在同一行。开发脚本是，如果没有注释，团队里的其他人就很难理解脚本究竟在做什么，如果时间长了自己也会忘记。。因此，要尽量养成为所开发的shell脚本书写注释的习惯，书写注意不光是方便别人，也是方便自己。的否则，写完一个脚本后也许几天就既不器脚本的用途了，需要时自重新阅读脚本很浪费很多宝贵时间。特别是影响团队的写作的效率，以及给后来接受维护的人带来一定的困难，注释尽量不用中文。

#### 1.1.6.2 shell脚本的执行

当shell脚本以非交互的方式（文件方式）运行时，它会先查找系统环境变量ENV，变量指定了环境文件（通常是.bashrc, .bash\_profile, /etc/bashrc, /etc/profile等），然后该环境变量文件开始执行脚本，当读取了ENV的文件后，shell才会开始执行shell脚本中的内容。

**特殊技巧：设置crond任务时，最好把系统环境变量在定时任务脚本中重新定义，否则，一些系统环境变量将不被加载，这个问题要注意！！！**

**shell脚本的执行通常可以采用以下几种方式：**

1. bash script-name或sh script-name （推荐使用：不需要权限）
2. path/script-name 或./script-name （当前路径下执行脚本：必须要给权限）
3. source script-name 或 . script-name #注意“.”点号。
4. sh<script-name或cat scripts-name|sh (同样适合bash)

例子：

[root@db02 scripts]# /bin/sh test.sh   
I am oldboy  
[root@db02 scripts]# /bin/bash test.sh     
I am oldboy  
[root@db02 scripts]# /server/scripts/test.sh  
-bash: /server/scripts/test.sh: 权限不够  
[root@db02 scripts]# LANG=en  
[root@db02 scripts]# /server/scripts/test.sh  
-bash: /server/scripts/test.sh: 权限不够  
[root@db02 scripts]# export LANG=en  
[root@db02 scripts]# /server/scripts/test.sh  
-bash: /server/scripts/test.sh: 权限不够  
[root@db02 scripts]# ./tesh.sh  
-bash: ./tesh.sh: 没有那个文件或目录  
[root@db02 scripts]# ./test.sh  
-bash: ./test.sh: 权限不够  
[root@db02 scripts]# source test.sh   
I am oldboy  
[root@db02 scripts]# . test.sh          
I am oldboy  
[root@db02 scripts]# sh <test.sh   
I am oldboy  
[root@db02 scripts]# cat test.sh |sh   
I am oldboy

某互联网公司Linux运维职位实际面试笔记填空题：

1、已知如下命令返回结果，请问echo $user的返回结果为（**空**）。

[root@db02 ~]# cat test.sh   
user=`whoami`  
[root@db02 ~]# sh test.sh    
[root@db02 ~]# echo $user

问：执行echo $user命令的结果是什么？

[root@db01 backup]# sh test.sh

[root@db01 backup]# echo $user

[root@db01 backup]# source test.sh

[root@db01 backup]# sh test.sh

[root@db01 backup]# echo $user

root

**结论：**

父亲shell不能直接继承儿子shell的变量等，反之可以。

儿子shell直接继承父亲shell的变量、函数等，反之不可以。

如果希望反过来继承，用source或者点号执行。

#### 1.1.6.3 shell脚本基本规范及习惯

1） 脚本第一行指定脚本解析器

#!/bin/sh 或 #！/bin/bash

2） 脚本开头加版本权等的信息

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为扩展名命名。

例：script-name.sh

5） 代码书写优秀习惯技巧

1. 成对的符号内容尽量一次写出来，防止遗漏。如：

{}、[]、''、``、""

2. []中括号两端要有空格，书写时即可留出空格[ ]，然后在退格书写内容，先书写一对中括号，然后退一格，然后在输入两个空格，再退一个格。双[[]]也是如此。

3. 流程控制语句一次书写完，在添加内容，如：

if语句格式一次完成：

if 条件内容

then

内容

fi

for循环格式一次完成：

for

do

内容

done

提示：while 和until，case等语句也是一样。

6） 通过缩进让代码更易读

if 条件内容

then

内容

fi

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。

#### 1.1.5.4 shell帮助与资料推荐

http://www.gnu.org/software/bash/manual/bash.html

## 1.2 shell变量基础及深入

### 1.2.1 什么是变量？

在小学初高中时，我们都知道数学方程式的例子，例如：已知x=1，y=x+1，那么y的等于 答案：y=1+1=2 这简直太简单了。

上述内容等号左边的x和y就是变量，等号右边的1和x+1就是变量的内容。

通过上面的例子我们可以得出一个变量的概念小结论：**简单的说，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），代替更多更复杂的内容，这个内容可能还会包含变量和路径，字符串等其他的内容，变量的定义存在内存中的。**使用变量的最大好处就是方便，当然，除了方便外，很多时候在编程中使用变量也是必须的，否则就无法完成相关的开发工作。

2、shell的变量特性

在bash shell中默认情况下是不会区分变量内容的类型，例如：整数、字符串、小数等，这一点和其他强类型语言是由区别的，例如：Java/c语言，当然，如果读者需要指定shell变量内容类型的，可以使用declare显示指定定义变量的类型。

### 1.2.2 变量类型

变量可分为两类：环境变量（全局变量）和普通变量（局部变量）。

环境变量也可称为全局变量，可以在创建他们的shell及其派生出来的任意子进程中使用。局部变量只能在创建他们的shell函数或脚本中使用。还有一些变量是用户创建\*其他的则是专用shell变量。

普通变量也可以成为局部变量，只能在\*\*\*\*

### 1.2.3 环境变量

环境变量用于定义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（crond任务要注意）。

环境变量可以在命令行中设置，但用户退出时这些变量值也会丢失，因此最好在家目录下的.bash\_profile文件中或全局配置/etc/bashrc，/etc/profile文件或者/etc/profile.d中定义。将环境变量放入上述的文件中，每次用户登录时这些变量值都将被初始化一次。

传统上，所有**环境变量格式均为大写。**环境变量应用于用户进程程序前，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**exportOLDGIRL=1**。

环境变量可用在创建他们的shell和从该shell派生的任意子shell或进程中。他们\*被称为全局变量以区别局部变量。通常，环境变量应该大写。环境变量是已经用export内置命令导出的变量。

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。通常环境变量定义并保存在用户家目录下的.bash\_profile文件或者全局的配置文件/etc/profile中。具体的环境变量说明如下表：

|  |  |
| --- | --- |
| 变量名 | 含义 |
| \_= | 上一条命令的最后一个参数 |
| BASH=/bin/bash | 调用bash实例时使用的全路径 |
| DIRSTACK=() | 代表目录栈的当前内容 |
| EUID=0 | 为在shell启动时被初始化的当前用户\*效ID |
| GROUPS=() | 当前用户所属的组 |
| HISTFILE=/root/.bash\_history | 历史记录文件的全路径 |

### 1.2.4 自定义环境变量（全局变量）

设置环境变量

如果想设置环境变量，就要在给变量赋值之后或设置变量时使用export命令。带\*项的declare内置命令也可完成同样的功能。（注意：输出变量时不要在变量名前加\*表export命令和选项。

|  |  |
| --- | --- |
| 选项 | 值 |
| -- |  |
| -f |  |
| -n |  |
| -p |  |

格式：

1） export 变量名=value

2） 变量名=value; export变量名

3） declare –x 变量名=value

提示：以上为三种设置环境变量的方法

例：

export NAME=oldboy

declare –x NAME=oldboy

NAME=oldboy;export NAME

自定义全局变量实例：

[root@db01 scripts]# tail -1 /etc/profile

export OLDBOY='I am oldboy'

[root@db01 scripts]# source /etc/profile

[root@db01 scripts]# echo $OLDBOY

I am oldboy

[root@db01 scripts]# cat test.sh

echo $OLDBOY

[root@db01 scripts]# sh test.sh

I am oldboy

环境变量设置的常用文件及区别

用户的环境变量配置：

[root@db01 scripts]# ls /root/.bashrc

/root/.bashrc

[root@db01 scripts]# ls /root/.bash\_profile

/root/.bash\_profile

当前用户：家目录下面

全局环境变量的配置

[root@db01 scripts]# ls /etc/profile

/etc/profile

/etc/bashrc

/etc/profile.d/

全局：在/etc/下

需要登录后显示加载内容可以把脚本文件放在/etc/profile.d/下，设置可执行即可。

[root@db01 ~]# cat /etc/motd #字符串内容

welcome to L L

[root@db01 ~]# cat /etc/profile.d/oldboy.sh #脚本内容

echo "this is oldboy training"

自定义环境变量生产环境Java环境配置实例：tomcat，resin.csvn，Hadoop。

export JAVA\_HOME=/application/jdk

export CLASSPATH=$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib

export PATH=#JAVA\_HOME/bin:$JAVA\_HOME/jre/bin:$PATH:$HOMR/bin

export RESIN\_HOME=/application/resin

常见放在：/etc/profile

如果写一个JAVA的脚本，还要把Java环境变量放入脚本内，特别是定时任务。

### 1.2.5 显示与取消环境变量

* 通过echo或printf命令打印环境变量

$HOME 用户登录时进入的目录。

$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。

$SHELL 当前SHELL

……省略若干。

提示：在写shell脚本时，可以直接使用上面的系统默认的环境变量

**环境变量小结：**

1、变量名通常大写。

2、可以在自身的shell以及子shell中使用。

3、通过export来定义环境变量。

4、输出用$+变量名，取消用unset+变量名（没有$）。

5、书写定时任务要注意环境变量，最好在脚本中重新定义。

6、如果希望永久生效，可以放在用户环境变量文件或者全局环境变量文件里。

用env（printenv）或set显示默认的环境变量

例：

[root@db01 ~]# env

HOSTNAME=db01

SHELL=/bin/bash

TERM=linux

HISTSIZE=1000

USER=root

MAIL=/var/spool/mail/root

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

PWD=/root

LANG=zh\_CN.UTF-8

MODULEPATH=/usr/share/Modules/modulefiles:/etc/modulefiles

LOADEDMODULES=

OLDBOY=I am oldboy

HISTCONTROL=ignoredups

SHLVL=1

HOME=/root

LOGNAME=root

CVS\_RSH=ssh

MODULESHOME=/usr/share/Modules

LESSOPEN=||/usr/bin/lesspipe.sh %s

G\_BROKEN\_FILENAMES=1

BASH\_FUNC\_module()=() { eval `/usr/bin/modulecmd bash $\*`

}

\_=/bin/env

提示：

1、用set显示所有本地变量。

2、也可以自定义全局环境变量，见如下例子。

* 用unset消除本地变量和环境变量

例：

[root@db01 ~]# echo $USER

root

[root@db01 ~]# unset USER

[root@db01 ~]# echo $USER

#--此处为输出空行了。

#### 1.2.5.1 shell特殊变量\*\*\*\*\*\*

1、位置变量

$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。

$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shell脚本所有传参的参数，将所有的参数视为单个字符串，相当于“$1$2$3“…注意与$#的区别

$# 获取当前执行的shell脚本后面接的参数的总个数

$@ 这个程序所有参数"$1""$2""$3""…"这是将参数传递给其他程序的最佳方式，因为他会保留所有内嵌在每个参数里的任何空白。"$@"和"$\*"都要加双引号。

提示：$\*和$@的区别？了解！

2、进程状态变量

$$ 获取当前shell脚本的进程号（PID）

$! 执行上一个指令的PID

$? 获取执行上一个指令的返回值（0为成功，非零为失败） #这个变量很常用。

$\_ 在此之前执行的命令或脚本的最后一个参数。

提示：查找知识方法：man bash，然后搜如下关键字 Special Parameters.

举例（通过脚本举例，可以一行行执行测试）：

[root@db01 oldboy\_2016-01-05]# cat etiantian.sh

echo '$0 获取当前执行的shell脚本的文件名;' $0

echo '$n 获取当前执行的shell脚本的第n个参数值，n=1..9;' '$1'=$1 '$2'=$2 "\$3=$3"

echo '$\* 获取当前shell的所有参数"$1 $2 $3 ...注意与$#的区别;"' $\*

echo '$# 获取当前shell命令行中参数的总个数;' $#

echo '$$ 获取当前shell的进程号（PID）;' $$

sleep 2 &

echo '$! 执行上一个指令的PID;' $!

echo '$? 获取执行的上一个指令的返回值;' $?

echo '$@ 这个程序的所有参数 "$1"$2"$2""...";' $@

echo '$\_ 在此之前执行的命令或脚本的最后一个参数;' $\_

下面是etiantian.sh脚本的执行结果：

[root@db01 oldboy\_2016-01-05]# sh etiantian.sh 参数1 参数2 参数3

$0 获取当前执行的shell脚本的文件名; etiantian.sh

$n 获取当前执行的shell脚本的第n个参数值，n=1..9; $1=参数1 $2=参数2 $3=参数3

$\* 获取当前shell的所有参数"$1 $2 $3 ...注意与$#的区别;" 参数1 参数2 参数3

$# 获取当前shell命令行中参数的总个数; 3

$$ 获取当前shell的进程号（PID）; 15302

$! 执行上一个指令的PID; 15303

$? 获取执行的上一个指令的返回值; 0

$@ 这个程序的所有参数 "$1"$2"$2""..."; 参数1 参数2 参数3

$\_ 在此之前执行的命令或脚本的最后一个参数; 参数3

举例：

[root@db01 oldboy\_2016-01-05]# sh etiantian.sh xiaogang taotao maoge

$0 获取当前执行的shell脚本的文件名; etiantian.sh

$n 获取当前执行的shell脚本的第n个参数值，n=1..9; $1=xiaogang $2=taotao $3=maoge

$\* 获取当前shell的所有参数"$1 $2 $3 ...注意与$#的区别;" xiaogang taotao maoge

$# 获取当前shell命令行中参数的总个数; 3

$$ 获取当前shell的进程号（PID）; 15304

$! 执行上一个指令的PID; 15305

$? 获取执行的上一个指令的返回值; 0

$@ 这个程序的所有参数 "$1"$2"$2""..."; xiaogang taotao maoge

$\_ 在此之前执行的命令或脚本的最后一个参数; maoge

（1）$1 $2...$9 ${10} ${11}

范例1：$n的例子

[root@db01 oldboy\_2016-01-05]# cat p.sh

echo $1

[root@db01 oldboy\_2016-01-05]# sh p.sh oldboy #传一个字符串参数

oldboy

[root@db01 oldboy\_2016-01-05]# sh p.sh oldboy oldgirl #传两个字符串参数，第二个参数脚本不会接收，参数默认是空格分隔。

oldboy

[root@db01 oldboy\_2016-01-05]# sh p.sh "lifen shadiao" #加引号括起来标识为一个字符串参数。

lifen shadiao

范例2：修改脚本设置$1，$2用来接收两个参数

[root@db01 oldboy\_2016-01-05]# cat p.sh

echo $1 $2

[root@db01 oldboy\_2016-01-05]# sh p.sh lifen mazi

lifen mazi

[root@db01 oldboy\_2016-01-05]# sh p.sh "lifen mazi" gongs

lifen mazi gongs

范例3：设置15个$n接收15个参数

[root@db01 oldboy\_2016-01-05]# echo \${1..15} >qq.sh

[root@db01 oldboy\_2016-01-05]# sh qq.sh {a..z}

a b c d e f g h i a0 a1 a2 a3 a4 a5

[root@db01 oldboy\_2016-01-05]# cat qq.sh

echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12 $13 $14 $15 #数字大于9后，输出内容错误的问题

[root@db01 oldboy\_2016-01-05]# cat q.sh

echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12} ${13} ${14} ${15} #数字大于9，加括号后输出正确结果。

[root@db01 oldboy\_2016-01-05]# sh q.sh {a..z}

a b c d e f g h i j k l m n o #数字大于9，加括号后输出正确内容。

（2） $0 取脚本的名称，也包括路径。

范例1：取脚本的名称及路径

[root@db01 oldboy\_2016-01-05]# cat n.sh

echo $0

不带路径执行脚本

[root@db01 oldboy\_2016-01-05]# sh n.sh

n.sh # $0就是这个n.sh

带全路径执行脚本

[root@db01 oldboy\_2016-01-05]# sh /server/scripts/oldboy\_2016-01-05/n.sh

/server/scripts/oldboy\_2016-01-05/n.sh # $0显示的是全路径的名称

提示：当要执行的脚本为全路径时，$0也会带着路径，此时如果希望取出名称或路径，用下面的方法。

范例2：扩展技术，只取脚本名称或者只取脚本路径

[root@db01 backup]# cat p.sh

dirname $0

basename $0

[root@db01 backup]# sh /server/backup/p.sh

/server/backup

p.sh

系统脚本案例：

[root@db01 oldboy\_2016-01-05]# tail -6 /etc/init.d/rpcbind

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

RETVAL=2

;;

esac

exit $RETVAL

（3）$#作用获取传参的个数

范例1：$#作用是获取传参的个数

[root@db01 oldboy\_2016-01-05]# cat q.sh

echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12} ${13} ${14} ${15}

echo $#

[root@db01 oldboy\_2016-01-05]# sh q.sh {a..z}

a b c d e f g h i j k l m n o

26

范例2：企业案例：控制用户命令行脚本传的参数个数

一个脚本练习$0,$1,$#号

[root@db01 oldboy\_2016-01-05]# cat q1.sh

[ $# -ne 2 ] && {

echo "$0 muse two"

exit 1

}

echo oldgirl

[root@db01 oldboy\_2016-01-05]# sh q1.sh

muse two

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

[root@db01 oldboy\_2016-01-05]# cat tejiang5.sh

#no.1

if [ $# -ne 2 ]

then

echo "USAGE:/bin/sh $0 arg1 arg2"

exit 1

fi

#no.2

echo $1 $2

范例3：$#的用法：

[root@db02 oldboy26]# /server/scripts/oldboy26/t1.sh  
-bash: /server/scripts/oldboy26/t1.sh: 权限不够  
[root@db02 oldboy26]# chmod +x /server/scripts/oldboy26/t1.sh  
[root@db02 oldboy26]# /server/scripts/oldboy26/t1.sh           
USAGE /server/scripts/oldboy26/t1.sh arg1 arg2  
[root@db02 oldboy26]# /server/scripts/oldboy26/t1.sh ddd  fff ggg  
USAGE /server/scripts/oldboy26/t1.sh arg1 arg2  
[root@db02 oldboy26]# /server/scripts/oldboy26/t1.sh ddd  fff   
ddd fff  
[root@db02 oldboy26]# cat /server/scripts/oldboy26/t1.sh  
#!/bin/sh  
#no.1  
if [ $# -ne 2 ]  
then  
  echo "USAGE $0 arg1 arg2"  
  exit 1  
fi  
  
#no.2  
echo $1 $2

（4）$？的用法

|  |  |
| --- | --- |
| 0 | 表示运行成功； |
| 2 | 权限拒绝;  [root@db01 oldboy\_2016-01-05]# su - oldboy  this is oldboy training  [oldboy@db01 ~]$ ll /root/  ls: 无法打开目录/root/: 权限不够  [oldboy@db01 ~]$ echo $?  2 |
| 1~125 | 表示运行失败，脚本命令、系统命令错误或参数传递错误;  [oldboy@db01 ~]$ su -root  su：无效选项 -- r  请尝试执行"su --help"来获取更多信息。  [oldboy@db01 ~]$ echo $?  125 |
| 126 | 找到该命令了，但是无法执行； |
| 127 | 未找到要运行的命令；  [oldgirl@db01 ~]$ oldboy  -bash: oldboy: command not found  [oldgirl@db01 ~]$ echo $?  127 |
| 128 | 命令被系统强制结束 |

提示：

在脚本中一般用exit 0，在执行脚本后，返回值给$?，函数里return数字，返回值给$?。

范例1：通过脚本控制命令及脚本执行后的返回值。

[root@db01 oldboy\_2016-01-05]# cat q1.sh

[ $# -ne 2 ] && {

echo "$0 muse two"

exit 111 #赋值给当前shell的$?变量了

}

exit 112

[root@db01 oldboy\_2016-01-05]# sh q1.sh

q1.sh muse two

[root@db01 oldboy\_2016-01-05]# echo $?

1

[root@db01 oldboy\_2016-01-05]# sh q1.sh li li

112

[root@db01 oldboy\_2016-01-05]# echo $?

0

返回值的企业场景案例用法：

1、判断命令或脚本，是否执行成功

2、通过在脚本里调用执行exit数字，则脚本返回数字给$?，

3、如果是函数里return数字 ，则函数返回这个数值给$?

提示：

1、 查看执行命令后的返回值，可以判断命令是否成功执行。

2、在脚本中一般用exit +数字，在执行脚本后，返回值给$?，进而查看脚本是不是成功执行。

3、 在脚本中一般用exit 0，在执行脚本后，返回值给$?，函数里return数字，返回值给$?。

$？系统的案例：

[root@db02 oldboy26]# sed -n '50,73p' /etc/init.d/rpcbind    
        echo -n $"Starting $prog: "  
        daemon $prog $1 "$RPCBIND\_ARGS"  
        RETVAL=$?  
        echo  
        if [ $RETVAL -eq 0 ] ; then  
                touch /var/lock/subsys/$prog  
                [ ! -f /var/run/rpcbind.pid ] &&  
                        /sbin/pidof $prog > /var/run/rpcbind.pid  
        fi  
        return $RETVAL  
}  
  
  
stop() {  
        echo -n $"Stopping $prog: "  
        killproc $prog  
        RETVAL=$?  
        echo  
        [ $RETVAL -eq 0 ] && {  
                rm -f /var/lock/subsys/$prog  
                rm -f /var/run/rpcbind\*  
        }  
        return $RETVAL  
}

（5）$\*$@的区别例子

q $\* 将命令行所有的命令行所有参数视为单个字符串，等同于“$1$2$3”,“$\*”

q $@ 将命令行每个参数视为单独的字符串，等同于“$1”“$2”“$3”这是将参数传递给其他程序的最佳方式，因为它会保留所有内嵌在每个参数里的任何空

注意：上述区别仅在于加双引号的时候，即“$\*”与“$@”

例子：

[root@db02 ~]# set -- "i am" handsome oldboy.  #通过set传入三个参数  
[root@db02 ~]# echo $#       ##输出参数个数，共三个参数  
3  
[root@db02 ~]# for i in "$\*";do echo $i;done  
##在有双引号的情况下"$\*"，参数里引号内内容当作一个参数输出了!  
i am handsome oldboy.  
[root@db02 ~]# for i in "$@";do echo $i;done   
i am  
handsome  
oldboy.

（6）$$特殊变量的案例

范例1：获取脚本执行的进程PID

[root@db02 oldboy26]# cat pid.sh  
echo $$ >/tmp/a.pid  
sleep 300  
[root@db02 oldboy26]# sh pid.sh &  
[1] 9846  
[root@db02 oldboy26]# cat /tmp/a.pid   
9846  
[root@db02 oldboy26]# grep 9846  
^C  
[root@db02 oldboy26]# grep 9846 /tmp/a.pid   
9846  
[root@db02 oldboy26]# kill `cat /tmp/a.pid`  
[root@db02 oldboy26]# ps -ef|grep 9846  
root      9865  8416  0 17:50 pts/1    00:00:00 grep --color=auto 9846  
[1]+  已终止               sh pid.sh

#### 1.2.5.2 bash内部变量命令

bash命令解释套装程序包含了一些内部命令。有些内部命令在目录列表时是看不见的，他们由shell本身提供。常用的内部命令有：echo,eval,exec,export,readonly,read,shift,wait和点（.）。下面简单介绍其命令格式和功能。

**（1）echo \*\*\*\*\***

echo - display message on screen

命令格式：echo args #可以是字符串和变量的组合。

功能：将echo命令后面args指定的字符串及变量等显示到标准输出。

**（2） eval\*\*\*\*\***

命令格式：eval args

功能：当shell程序执行到eval语句时，shell读入参数args，并将它们组合成一个新的命令，然后执行。

**（3） exec\*\*\*\*\***

命令格式：exec命令参数

功能：当shell执行到exec语句时，不会去创建新的子进程，而是转去执行指定的命令，当指定的命令执行完时，该进程（也就是最初的shell）就终止了，所以shell程序中exec后面的语句将不再被执行。

**（4） export\*\*\*\*\***

命令格式：export 变量名 或：export变量名=变量值

功能：shell可以用export把它的变量向下带入子shell，从而让子进程继承父进程中的环境变量。但子shell不能用export把它的变量向上带入父shell。

注意：不带任何变量名的export语句将显示出当前所有的export变量。

**（5） readonly （很少用）**

readonly – Mark variables/functions as readonly

命令格式：readonly 变量名。

功能：将一个用户定义的shell变量标识为不可变。不带任何参数的readonly命令将显示出所有只读的shell变量。

**（6） read\*\*\*\*\***

命令格式：read 变量名表

功能：从标准输入读字符串等信息，传给shell程序内部定义的变量。

可以在函数中用local变量名的方式申明局部变量。

**（7） shift语句\*\*\*\*\***

shift – Shift positional parameters

功能：shift语句按如下方式重新命名所有的位置参数变量，即$2成为$1，$3成为$2…在程序中没使用一次shift语句，都使所有位置参数依次向左移动一个位置，并使位置参数$#减1，直到减到0为止。

**（8） wait**

功能：使shell等待在后台启动的所有子进程结束。wait的返回值总的是真。

**（9） exit\*\*\*\*\***

exit –Exit the shell

功能：退出shell程序。在exit之后可有选择地指定一个数位作为返回状态。

**（10） “.” \*\*\*\*\***

命令格式：. shell程序文件名

命令格式：.shell程序文件名

功能：使shell读入指定的shell程序文件并以次执行文件中的所有语句

#### 1.2.5.3 变量子串的常用操作（了解）

|  |  |
| --- | --- |
| 表达式 | 说明 |
| ${#string} | 返回$string的长度 |
| ${string:position} | 在$string中，从位置position之后开始提取子串 |
| ${string:position:length} | 在$string中，从位置position之后开始提取长度为length的子串 |
| ${string#substring} | 从$string开头开始删除最短匹配substring的子串 |
| ${string##substring} | 从$string开头开始删除最长匹配substring的子串 |
| ${string%substring} | 从$string结尾开始删除最短匹配substring的子串 |
| ${string%%substring} | 从$string结尾开始删除最长匹配substring的子串 |
| ${parameter/pattern/string} | 使用string，来代替第一个匹配的pattern |
| ${parameter/#pattern/string} | 从开头匹配string变量中的pattern，就用string来替换匹配的pattern。 |
| ${parameter/%pattern/string} | 从结尾开始匹配string变量中的pattern，就用string来替换匹配的pattern。 |
| ${parameter//pattern/string} | 使用string，来代替所有匹配的pattern |

依次举例说明：

定义OLDBOY变量，内容为“I am oldboy”

[root@db01 ~]# OLDBOY="I am oldboy"

[root@db01 ~]# echo $OLDBOY

I am oldboy

[root@db01 ~]# echo ${OLDBOY}

I am oldboy

1） 返回字符串OLDBOY变量字符串的长度

[root@db01 ~]# echo ${#OLDBOY}

11

[root@db01 ~]# echo $OLDBOY|wc -m

12

[root@db01 ~]# echo $OLDBOY|wc -L

11

[root@db01 ~]# echo $OLDBOY|wc -c

12

[root@db01 ~]# expr length "$OLDBOY"

11

面试题：打印下面语句中字符数小于6的单词。

I am oldboy linux,welcome to our training.

2）截取OLDBOY变量字符串从第2个字符之后开始取，默认取后面的字符的全部，第2个字符不包含在内。也可以理解为删除前面的多少个字符。

[root@db01 ~]# echo ${OLDBOY:2}

am oldboy

3）截取OLDBOY变量字符串从第2个字符之后开始取，取两个字符。

[root@db01 ~]# echo ${OLDBOY:2:2}

am

[root@db01 ~]# echo ${OLDBOY}|cut -c 3-4

am

4）从变量$OLDBOY开头开始删除最短匹配“a\*C”子串

[root@db01 ~]# OLDBOY=abcABC123ABCabc

[root@db01 ~]# echo ${OLDBOY}

abcABC123ABCabc

[root@db01 ~]# echo ${OLDBOY#a\*C}

123ABCabc

5）从变量$OLDBOY开头开始删除最长匹配子串

[root@db01 ~]# echo ${OLDBOY##a\*C}

abc

6）从变量$OLDBOY结尾开始删除最短匹配oldboy子串

[root@db01 ~]# OLDBOY=abcABC123ABCabc

[root@db01 ~]# echo ${OLDBOY%a\*c}

abcABC123ABC

[root@db01 ~]# echo ${OLDBOY%%a\*c}

7）从变量$OLDBOY结尾开始删除最长匹配boy子串

小结：

#开头删除匹配最短。

##开头删除匹配最长。

%结尾删除匹配最短。

%%结尾删除匹配最长。

8）使用etiantian字符串，来代替变量$OLDBOY第一个匹配的oldboy字符串

[root@db01 ~]# OLDBOY="I am oldboy oldboy"

[root@db01 ~]# echo $OLDBOY

I am oldboy oldboy

[root@db01 ~]# echo ${OLDBOY/oldboy/etiantian}

I am etiantian oldboy

[root@db01 ~]# echo ${OLDBOY//oldboy/etiantian}

I am etiantian etiantian

9）使用etiantian字符串，来代替从变量$OLDBOY结尾开始匹配的oldboy字符串

[root@db01 ~]# echo ${OLDBOY/%oldboy/etiantian}

I am oldboy etiantian

10）使用He is字符串，来代替从变量$OLDBOY开头开始匹配的I am 字符串

[root@db01 ~]# echo ${OLDBOY/#I/lifen}

lifen am oldboy oldboy oldboy

其他的变量替换：

|  |  |
| --- | --- |
| 运算符号 | 替换 |
| ${value：-word} | 如果变量名存在且非null，则返回变量的值。否则，返回word字符串。  用途：如果变量我定义，则返回默认值。  范例：${value:-word}，如果value未定义，则表达式的值为word。 |
| ${value:=word} | 如果变量名存在且非null，则返回变量值。否则，设置这个变量值为word，并返回其值。  用途：如果变量未定义，则设置变量为默认值，并返回默认值。  范例：${value:=word}，如果value未定义，则设置value值为word，返回表达式的值也为word。 |
| ${value:?”not defined”} | 如果变量名存在且非null，则返回变量的值。否则显示变量名：message，并退出当前的命令或脚本。  用途：用于捕捉由于变量为定义而导致的错误，并退出程序。  范例：${value:?”not defined”}如果value未定义，则显示-bash：value：not defined并退出。 |
| ${value:+word} | 如果变量名存在且非null，则返回word。否则返回null。  用途：测试变量是否存在。  范例：${value:+word} 如果value已经定义，返回word（也就是真）。 |
| **更多内容，请执行 man bash 查看帮助。Parameter Expansion** | |

每个运算符内的冒号都是可选的。如果省略冒号，则将每个定义中的“存在且非null”部分改为“存在”，也就是说，运算符仅用于测试变量是否存在。

下面依次举例说明：

1） ${value:-word}

当变量为定义或者值为空时，返回值为word内容，否则返回变量的值。

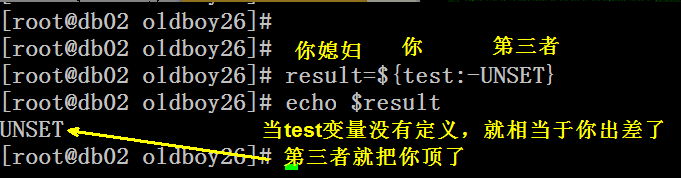
[root@db02 oldboy26]# result=${test:-UNSET}  
[root@db02 oldboy26]# echo $result  
UNSET

[root@db02 oldboy26]# echo $test  
###这里为空

结论：当test变量没有内容时，就返回了后面的UNSET。  
[root@db02 oldboy26]# test='oldboy'  
[root@db02 oldboy26]# echo $result   
UNSET  
[root@db02 oldboy26]# result=${test:-UNSET}  
[root@db02 oldboy26]# echo $result           
oldboy

提示：这个变量功能可以用来判断变量是否没有定义。

结论：当test变量有内容时，就返回了test变量的内容。



2）${value:=word}

[root@db01 ~]# unset test

[root@db01 ~]# unset lifen

[root@db01 ~]# lifen=${test:=UNSET}

[root@db01 ~]# echo $lifen

UNSET

[root@db01 ~]# echo $test

UNSET

[root@db01 ~]# test='mazi'

[root@db01 ~]# lifen=${test:=UNSET}

[root@db01 ~]# echo $lifen

mazi

#### 1.2.5.4 变量的数值（整数）计算

变量的数值计算常见的有如下几个命令：

（（））、let、expr、bc（小数）、$[]，其他都是整数。

##### 1. （（））用法：（此法很常用）

如果要执行简单的整数运算，只需将特定的算术表达式用“$((“和”))”括起来即可。

shell的算术运算符号常置于“$((“......”))”的语法中。这一语法如同双引号功能，除了内嵌双引号无需转义。

|  |  |
| --- | --- |
| 运算符 | 意义 |
| ++-- | 增加及减少，可前置也可放在结尾 |
| +-！~ | 一元的正号与负号；逻辑与位的取反。 |
| \* / % | 乘法、除法、与取余。 |
| +- | 加法、减法。 |
| <<= > >= | 比较符号。 |
| == ！= | 相等于不相等，一个“=”赋值。 |
| << >> | 向左位移，向右位移。 |
| & | 位的AND |
| ^ | 位的异或 |
| | | 位的或 |
| && | 逻辑的AND（make && make install） |
| || | 逻辑的OR（make || make install） |
| ？: | 条件表达式 |
| = += -= \*= /= %= &= ^=  <<= >>= |= | 赋值运算符a+=1 都相当a=a+1 |
| \*\* | 幂运算 |

加减乘除

[root@db02 oldboy26]# ((a=1+2\*\*3-4%3))  
[root@db02 oldboy26]# echo $a  
8  
[root@db02 oldboy26]# echo $((1+2\*\*3-4%3))  
8

[root@db02 oldboy26]# ((a=1+2\*\*3-4%3))  
[root@db02 oldboy26]# echo $a  
8  
[root@db02 oldboy26]# echo $((1+2\*\*3-4%3))  
8  
[root@db02 oldboy26]# echo $a  
8  
[root@db02 oldboy26]# echo $((a+=1))  
9  
[root@db02 oldboy26]# echo $a  
9  
[root@db02 oldboy26]# echo $((a-=3))  
6  
[root@db02 oldboy26]# echo $a  
6  
[root@db02 oldboy26]# echo $((a++))   
6  
[root@db02 oldboy26]# echo $a  
7  
[root@db02 oldboy26]# echo $((a++))  
7  
[root@db02 oldboy26]# echo $a  
8  
[root@db02 oldboy26]# echo $((a++))  
8  
[root@db02 oldboy26]# echo $((a--))  
9  
[root@db02 oldboy26]# echo $a  
8  
[root@db02 oldboy26]# echo $((a--))  
8  
[root@db02 oldboy26]# echo $a  
7

[root@db02 oldboy26]# myvar=99    
[root@db02 oldboy26]# echo $(($myvar+1))  
100  
[root@db02 oldboy26]# echo $(($myvar + 1))  
100  
[root@db02 oldboy26]# echo $((   $myvar + 1  ))  
100  
[root@db02 oldboy26]# echo $(($myvar+1))         
100  
[root@db02 oldboy26]# echo $((100+5))     
105  
[root@db02 oldboy26]# echo $((100-5))  
95  
[root@db02 oldboy26]# echo $((100\*5))  
500  
[root@db02 oldboy26]# echo $((100\*\*5))  
10000000000  
[root@db02 oldboy26]# echo $((100/5))   
20  
[root@db02 oldboy26]# echo $((100%5))  
0  
[root@db02 oldboy26]# echo $((100%3))  
1

提示：

1、 表达式执行不需要$符号，直接（（100 % 5）），如果需要输出，就加$，例如：echo$((100 % 5))

2、 （（））里的所有字符之间有没有1个或多个空格都不影响结果。

提示：

1. \*\*为幂运算：%为取模运算（就是除法当中取余数），加减乘除就不细说了。

2. 上面设计到的参数变量必须为整数（整列）。不能是小数（符点数）或者字符串。

后面的bc命令可以进行浮点数运算，但一般较少用到，可以直接在shell脚本中使用上述命令进行计算。

3． echo$((a++))和echo$((a--))标识先输出a自身的值，然后在进行++--的运算，echo$((++a))和echo$((--a))标识先进行++--的运算，在输出a自身的值。

记忆方法：++，--

变量a在前，表达式的值为a，然后a自增或自减，变量a在符号后，表达式值自增或自减，然后a值自增或自减。

范例：--，++在变量前后的举例理解：

[root@db01 oldboy\_2016-01-05]# a=10

[root@db01 oldboy\_2016-01-05]# echo $((a++)) #先输出a的值，因为a为10，所以输出10

10

[root@db01 oldboy\_2016-01-05]# echo $a #上面输出a的值后，a自增1，因此为11

11

范例：各种（（））运算的shell脚本例子

[root@db01 oldboy\_2016-01-05]# cat laohao.sh

#!/bin/bash

a=6

b=4

echo "a-b=$(($a - $b))"

echo "a+b=$(($a + $b))"

echo "a\*b=$(($a \* $b))"

echo "a/b=$(($a / $b))"

echo "a\*\*b=$(($a \*\* $b))"

echo "a%b=$(($a % $b))"

执行结果如下：

[root@db01 oldboy\_2016-01-05]# sh laohao.sh

a-b=2

a+b=10

a\*b=24

a/b=1

a\*\*b=1296

a%b=2

[root@db02 oldboy26]# cat test.sh   
#!/bin/bash  
  
#no.1  
[ $# -ne 2 ] &&{  
 echo "USAGE $0 NUM1 NUM2"  
 exit 1  
}  
#no.2   
a=$1  
b=$2  
expr $a + $b + 1 &>/dev/null  
if [ $? -ne 0 ]  
 then  
   echo "you must input two nums."  
   exit 2  
fi  
  
#no.3  
echo "a-b =$(( $a - $b ))"  
echo "a+b =$(( $a + $b ))"  
echo "a\*b =$(( $a \* $b ))"  
echo "a/b =$(( $a / $b ))"  
echo "a\*\*b =$(( $a \*\* $b ))"  
echo "a%b =$(( $a % $b ))"

expr（evaluate（求值）expressions（表达式））命令的用法：

expr命令一般用于整数值，但也可用于字符串，用来求表达式变量的值，同时expr也是一个手工命令行计算器。

1、计算

语法：expr expression

[root@db01 oldboy\_2016-01-05]# expr 2+2 #符号“+”前后一定要有空格，否则不会计算。

2+2

[root@db01 oldboy\_2016-01-05]# expr 2 + 2

4

[root@db01 oldboy\_2016-01-05]# expr 2 - 2

0

[root@db01 oldboy\_2016-01-05]# expr 2 \* 2

expr: 语法错误

[root@db01 oldboy\_2016-01-05]# expr 2 \\* 2 # 符号\*用/号来转义。

4

提示：expr用法：

1、注意：运算符及计算的数字左右都有至少一个空格。

2、使用乘号时，必须用反斜线屏蔽其特定含义，因为shell可能会吴杰星号的含义。

增量计数

expr在循环中可用于增量计算，首先，循环初始化为0，然后循环值加1，反引号的用法命令替代，最基本的一种是从（expr）命令接收输出并将之放入循环变量。

例：给自变量i加1

[root@db01 oldboy\_2016-01-05]# i=0

[root@db01 oldboy\_2016-01-05]# i=`expr $i + 1`

[root@db01 oldboy\_2016-01-05]# echo $i

1

expr $[$a+$b]表达式形式，其中$a $b可为整数值。

[root@db01 oldboy\_2016-01-05]# expr $[2+3]

5

[root@db01 oldboy\_2016-01-05]# expr $[2\*3]

6

[root@db01 oldboy\_2016-01-05]# expr $[2\*\*3]

8

[root@db01 oldboy\_2016-01-05]# expr $[2/3]

0

[root@db01 oldboy\_2016-01-05]# expr $[2%3]

2

expr特殊用法：判断文件扩展命名是否符合要求

expr用法ssh-copy-id脚本

if expr "$1" : ".\*\.pub" ; then

#expr id\_dsa.pub : ’.\*\.pub’，匹配\*.pub格式的文件如果是则为真。例：

[root@db01 oldboy\_2016-01-05]# cat jiude.sh

#!/bin/sh

if expr "$1" : ".\*\.pub" &>/dev/null

then

echo "you are using $1"

else

echo "pls use \*.pub file"

fi

[root@db01 oldboy\_2016-01-05]# sh jiude.sh oldboy.pub

you are using oldboy.pub

[root@db01 oldboy\_2016-01-05]# sh jiude.sh oldboy.log

pls use \*.pub file

expr特殊用法：判断是否为整数的例子：

[root@db01 oldboy\_2016-01-05]# cat laohao.sh

#!/bin/bash

a=$1 b=$2

#no.1

[ $# -ne 2 ] && {

echo "USAGE $0 arg1 arg2"

exit 1

}

#no.2

expr $a +$b +1 &>/dev/null

[ $? -ne 0 ] && {

echo "USAGE $0 请输入正确的整数值"

exit 22

}

#no.3

echo "a-b=$(($a - $b))"

echo "a+b=$(($a + $b))"

echo "a\*b=$(($a \* $b))"

echo "a/b=$(($a / $b))"

echo "a\*\*b=$(($a \*\* $b))"

echo "a%b=$(($a % $b))"

expr 特殊用法：通过expr计算字符串的长度

[root@db01 oldboy\_2016-01-05]# chars=`seq -s " " 100`

[root@db01 oldboy\_2016-01-05]# echo ${#chars}

291

[root@db01 oldboy\_2016-01-05]# echo $(expr length "$chars")

291

[root@db01 oldboy\_2016-01-05]# echo $chars|wc -L

291

[root@db01 oldboy\_2016-01-05]# echo $chars|awk '{print length($0)}'

291

压力测试

[root@db01 oldboy\_2016-01-05]# chars=`seq -s " " 100`

[root@db01 oldboy\_2016-01-05]# echo ${#chars}

291

[root@db01 oldboy\_2016-01-05]# echo "$chars"|awk '{print length($0)}'

291

[root@oldboy ~]# time for i in $(seq 11111);do count=${#chars};done; ç最快  
real    0m0.723s  
user    0m0.706s  
sys     0m0.010s  
[root@oldboy ~]# time for i in $(seq 11111);do count=`echo expr length "${chars}"`;done; ç其次  
real    0m7.095s  
user    0m1.894s  
sys     0m5.168s  
[root@db02 oldboy26]# time for i in $(seq 11111);do count=`echo "$chars"|awk '{print length($0)}'`;done;  
real    0m21.711s  
user    0m1.076s  
sys     0m2.631s  
[root@oldboy ~]# time for i in $(seq 11111);do count=`echo ${chars}|wc -L`;done; ç最慢  
real    0m24.761s  
user    0m6.486s  
sys     0m18.144s

我们看到速度相差几十到上百倍，一般情况调用外部命令处理，与内置功能操作性能相差较大，在shell编程中，我们应尽量用内置操作或函数完成。

压力测试小结：

1、内置功能最佳。

2、学习压力测试。

##### （2） bc命令的用法（可以整数也可以小数）：

bc是Unix下的计算器，它可以用在命令行下面：

例：

[root@db02 ~]# i=2  
[root@db02 ~]# i=`echo $i+1|bc`  
[root@db02 ~]# echo $i  
3

小结：shell的数值运算方法：

expr、(())、let、bc、$[]、awk、typeset

### 1.2.6 局部变量

#### 1.2.6.1 定义本地变量

本地变量在用户当前的shell生存期的脚本中使用。例如，本地变量oldboy取值为ett098，这个值只在用户当前shell生存期中有意义。如果在shell中启动另一个进程或退出。本地变量oldboy值将无效。

1、普通字符串变量定义\*\*\*\*\*

变量名=value

变量名= 'value '

变量名= "value"

2、shell中变量名及变量内容的要求

一般是由字母，数字，下划线组成。以字母开头，例如：

oldboy，oldboy123，oldboy\_training.

变量的内容，可以使用单引号或双引号引起来，或不加引号。

3、普通字符串变量定义例子

例1：下面的例子，在命令行输入如下内容会输出什么结果？

a=192.168.1.2

b= ' 192.168.1.2'

c="192.168.1.2"

echo "a=$a"

提示：

1） $c和${c}在这里等同。

2） 需要在命令行实践以上内容。

思考：想一想a,b,c各是什么结果？

答案：

[root@db01 ~]# a=192.168.1.2

[root@db01 ~]# echo $a

192.168.1.2

[root@db01 ~]# b='192.168.1.2'

[root@db01 ~]# echo $b

192.168.1.2

[root@db01 ~]# c="192.168.1.2"

[root@db01 ~]# echo $c

小结：连续普通字符串内容，赋值给变量，不管用什么引号，或者不用引号，内容是什么，打印变量就都输出什么。

例2：接着上述例1的结果，在命令行继续输入如下内容，想一想，a,b,c又各是什么结果？

a=192.168.1.2-$a

b='192.168.1.2-$a'

c="192.168.1.2-$a"

第一种定义a变量的方式是直接定义变量内容，内容一般为简单连续的数字、字符串、路径名等。

第二种定义b变量的方式是通过单引号定义变量。这个定义方式的特点是：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（反引起来）也会把他们原样输出。此法比较适合于定义显示纯字符串，既不希望解析变量，命令等。

第三种定义c变量方式是通过双引号定义变量。这个定义方式的特点是：输出变量时引号里的变量及命令会经过解析后才输出该变量的内容，而不是把引号中变量名及命令（反引起来）原样输出，适合于字符串中附带有变量及命令的内容想解析后输出的变量定义。

**常规使用习惯及使用建议：数字内容变量定义不加引号，其他字符串等定义最好加上双引号。**

**5. 把一个命令作为变量定义的方法**

需要获取命令结果的变量内容写法：

变量名=`ls` 反引号

变量名=$(ls)

范例1：

[root@db01 backup]# ls

oldboy test

[root@db01 backup]# CMD=`ls`

[root@db01 backup]# echo $CMD

oldboy test

[root@db01 backup]# CMD1=$(pwd)

[root@db01 backup]# echo $CMD1

/server/backup

生产环境常见应用：

范例2：对站点按天打包生成不同的文件名

[root@db01 backup]# CMD=$(date +%F)

[root@db01 backup]# echo $CMD

2016-01-04

[root@db01 backup]# echo $CMD.tar.gz

2016-01-04.tar.gz

[root@db01 backup]# echo `date +%F`.tar.gz

2016-01-04.tar.gz

[root@db01 backup]# tar zcvf etc\_$(date +%F).tar.gz /etc

[root@db01 backup]# H=$(uname -n)

[root@db01 backup]# echo $H

db01

6. 变量定义小结：

**普通变量内容的定义：**

1） a=1连续的数字字符串，可以不加引号，例如：a=123.

2） 变量内容很多，哈希望解析变量，加双引号，例如：a=”/etc/rc.local $USER”，会对内容中的$USER解析输出。

3） 希望原样输出，就用单引号。例如：a=’$USER’原样输出。

**希望变量内容是命令结果的定义：**

1） 反引号括起来，例如：a=`ls`

2） 或者$()括起来，例如：a=$(ls)

**变量的输出方法：**

echo $变量名

printf

7． 变量定义的小结技巧集锦：

1、“CMD=`ls`”注意命令变量前后的字符``反引号，不是单引号。

2、在变量名前加$，可以取得此变量的值，使用echo或printf命令可以显示变量的值，$A和${A}的写法不同，但功能一样，推荐使用后者的语法或“${A}”的用法。

3、用echo等命令输出的时候，也有单引号、双引号、反引号的形式，用法和前面一致。

4、${WEEK}day若变量后面有其他字符连接的时候，就必须给变量加上大括号{}。例如：$dbname\_tname就要改成${dbname}\_tname。

5、养成将所有**字符串变量**用**双引号括起来**使用的习惯，将会减少很多编程时遇到的怪异的错误。具体使用方法如：”$A”或”${A}”的用法，特别比较判断时候。

例子：

grep实例：

[root@db02 ~]# echo testchars >grep.log  
[root@db02 ~]# echo oldboy >>grep.log    
[root@db02 ~]# cat grep.log   
testchars  
oldboy  
[root@db02 ~]# grep "$OLDBOY" grep.log   
testchars  
[root@db02 ~]# grep $OLDBOY grep.log     
testchars  
[root@db02 ~]# grep '$OLDBOY' grep.log   
[root@db02 ~]# vim grep.log   
[root@db02 ~]# grep '$OLDBOY' grep.log   
$OLDBOY  
[root@db02 ~]# sed -n /$OLDBOY/p grep.log   
testchars  
[root@db02 ~]# sed -n /"$OLDBOY"/p grep.log   
testchars  
[root@db02 ~]# sed -n /'$OLDBOY'/p grep.log      
$OLDBOY

awk实例：

[root@db02 ~]# ETT=123  
[root@db02 ~]# awk 'BEGIN {print "$ETT"}'  
$ETT  
[root@db02 ~]# awk 'BEGIN {print '$ETT'}'  
123  
[root@db02 ~]# awk 'BEGIN {print $ETT}'

[root@db02 ~]# ETT=123  
[root@db02 ~]# awk 'BEGIN {print "$ETT"}'  
$ETT  
[root@db02 ~]# awk 'BEGIN {print '$ETT'}'  
123  
[root@db02 ~]# awk 'BEGIN {print $ETT}'  
  
[root@db02 ~]# ETT='abc'  
[root@db02 ~]# awk 'BEGIN {print "$ETT"}'  
$ETT  
[root@db02 ~]# awk 'BEGIN {print '$ETT'}'  
  
[root@db02 ~]#  awk 'BEGIN {print "'$ETT'"}'  
abc

**有关awk调用shell变量读者还可以参考老男孩的博文：**一道实用linux运维问题的9种shell解答方法（http://oldboy.blog.51cto.com/2561410/760192）

**变量名及变量内容定义小结：**

1、 变量名只能为字母，数字，下划线，只能字母或下划线开头。

2、 规范的变量名写法定义：见名知意。

1） OldboyAge=1 #每个单词的首字母大写

2） oldboy\_age=1 #单词之间用”\_”

3） oldboyAgeSex=1 #驼峰语法：首个单词的首字母小写，其余单词首字母大写

3、简单的内容可以不加引号。

4、=号的知识

**a=1 中的，等号是赋值的意思，比较是不是相等，为“==“，字符串比较也可用=等号。**

**5、 打印变量，变量名前接$符号，变量名后面紧接着字符的时候，要用大括号将变量单独括起来，防止金庸新著的问题。**

**6、 打印输出或使用变量内容时**，一般用双引号或者不加引号，如果是字符串变量最好加双引号，希望原样输出，使用单引号。

7、 注意变量内容引用方法，一般用双引号，简单连续字符串可以不加引号，希望原样输出，使用单引号。

8、 变量内容是命令，要用反引号或者$()把变量括起来使用。

#### 1.2.6.2 定义变量单引号、双引号与不加引号

1、有关shell变量邓毅及变量输出单引号、双引号与不加引号的简要说明如下：

|  |  |
| --- | --- |
| 名称 | 解释 |
| 单引号 | 所见即所得：即将单引号内的所有内容都原样输出，或者描述为单引号里**看到的是什么就会输出什么。** |
| **双引号（建议）** | **把双引号内人所有内容都输出出来；如果内容中有命令（要反引下）、变量、特殊转义符等，会先把变量、命令、转义字符解析出结果，然后在输出最终内容来。** |
| 无引号 | 把内容输出出来，会将含有空格的字符串视为一个整体输出，如果内容中有命令（要反引下）、变量等，会先把变量、命令解析出结果，然后在输出最终内容来，如果字符串中带有空格等特殊字符，则不能完整的输出，需要改加双引号，一般连续的字符串，数字，路径等可以不加任何引号，不过无引号的情况最好用双引号替代之。 |
| 反引号 | 一般用于引命令，执行时命令会被执行。不能与单引号配合。相当于$() |
| 提示：这里的结论仅为经验型的结论，可能对于某些语言不适合，例如：awk内部就有特殊。 | |

建议：

1、在脚本中定义普通字符串变量，尽量把变量的内容用双引号引起来。

2、单纯数字内容可以不加引号。

3、希望变量内容原样输出加单引号。

4、希望引用命令并获取命令的结果就用反引号。

一道使用Linux运维问题的9中shell解答方法（

<http://oldboy.blog.51cto.com/2561410/760192>）

## 1.3 条件测试与比较

### 1.3.1 条件测试与比较

#### 1.3.1.1 条件测试方法总括

在bash的各种流程控制结构中通常要进行各种测试，然后根据测试结果执行不同的操作，有时也会通过与if等条件语句相结合，更方便的完成判断。

条件测试通常有如下3中语法形式：

语法格式1：test<测试表达式>

**语法格式2：test[ <测试表达式> ]**

语法格式3：test[[ <测试表达式> ]]

说明：

1、 上述语法格式1和语法格式2的写法是等价的。语法格式3为扩展test命令，建议使用语法格式2.

2、 在[[]]中可以使用通配符进行模式匹配。&&、||、>、<等操作符可以应用于[[]]中，但不能应用于[]中。

3、 对于整数的关系运算，也可以使用shell的算术运算符（（））。

#### 1.3.1.2 test条件测试语法及示例：

test条件测试的语法格式：test<测试表达式>

范例1：test命令-f选项（文件存在且为普通文件则表达式成立）测试文件是否存在

[root@db01 3306]# test -f file && echo true||echo false

##file文件存在并且是普通文件为真，因为file文件不存在，所以返回false

false

[root@db01 3306]# touch file ##创建不存在的普通文件file

[root@db01 3306]# test -f file && echo true||echo false ##文件存在，所以返回true

true

范例2：test命令-z选项（字符串长度为0，则表达式成立）测试字符串oldboy

[root@db01 3306]# test -z "oldboy" &&echo 1||echo 0

0

[root@db01 3306]# char="oldboy"

[root@db01 3306]# test -z "$char" && echo 1||echo 0

0

提示：

1、更多test测试表达式，请执行man test查看test测试命令的帮助。

2、字符串一定要加双引号。

#### 1.3.1.3 []条件测试语法及示例

[]条件测试的语法格式：[ <测试表达式> ]，注意：中括号里面的两端要有空格

范例1：[]命令-f选项（文件存在且为普通文件则表达式成立）测试文件

[root@db01 3306]# [ -f /etc/hosts ] && echo 1||echo 0

1

[root@db01 3306]# [ -f /etc/oldboy.txt ] && echo 1||echo 0

0

#### 1.3.1.4 [[]]条件测试语法及示例

[[]]条件测试的语法格式：[[<测试表达式>]]，注意：双中括号里的两端要有空格

格式3： [[<测试表达式>]]

范例：[[]]

[root@db01 3306]# [[ -f /etc/hosts ]] && echo 1||echo 0

1

[root@db01 3306]# [[ -f /etc ]] && echo 1||echo 0

0

[[]]表达式与[]和test的用法的选项部分是相同的，最大的差别就是使用逻辑符号多条件判断写法，以及整数比较字符方面。

重大区别1：使用逻辑符号多条件判断方面。

例如：&&符号在[[]]中多条件表达式用法：

[root@db01 3306]# [[ -e /etc/hosts && -f /etc/hosts ]] && echo 1||echo 0

1

而改用&&符号在[]中表达式同样用法就会报错

[root@db01 3306]# [ -e /etc/hosts && -f /etc/hosts ] && echo 1||echo 0

-bash: [: missing `]'

0

在[]中多条件表达式正确用法为将&&用-a替换

[root@db01 3306]# [ -e /etc/hosts -a -f /etc/hosts ] && echo 1||echo 0

1

建议使用[]语法。

### 1.3.2 文件测试表达式

#### 1.3.2.1 文件测试表达式用法

在学习测试表达式之前，先举一个生活中的例子：如果你要找我去台球厅，你一定不会先去台球厅，而是会先打电话给我，问下有没有时间一起去打球，同样的道理，编程时需要处理一个对象时，徐璈对对象进行测试，只有符合要求，采取操作\*组偶读好处就是避免程序出错以及无谓的消耗系统资源，这个要测试的对象可以\*字符串，数字等。

在书写文件测试表达式时，通常可以使用下表中的文件测试操作符

|  |  |
| --- | --- |
| **常用文件测试操作符号** | **说明** |
| **-f文件，file** | 文件存在且为普通文件则真，即测试表达式成立。 |
| **-d文件，directory** | 文件存在且为目录文件则真，即测试表达式成立。 |
| **-s文件，size** | 文件存在且文件大小不为0则真，及测试表达式成立。 |
| **-e文件，exist** | 文件存在则真，及测试表达式成立。只要有文件就行。  区别“-f” |
| -r文件，read | 文件存在且可读则真，及测试表达式成立。 |
| -w文件，write | 文件存在且可写则真，即测试表达式成立。 |
| -x文件，executable | 文件存在且可执行则真，即测试表达式成立。 |
| -L文件，link | 文件存在且为链接文件则真，即测试表达式成立。 |
| f1 –nt f2，new erthan | 文件f1比文件f2新则真，即测试表达式成立。 |
| f1 –ot f2，old erthan | 文件f1比文件f2旧则真，即测试表达式成立。 |

特别说明：这些操作符号对于[[]]、[]、test几乎是通用的。

#### 1.3.2.2 文件测试表达式举例

普通文件测试表达式举例：

普通文件（测试文件类型）

[root@db01 ~]# touch oldboy

[root@db01 ~]# ls -l oldboy

-rw-r--r-- 1 root root 0 1月 7 13:42 oldboy

[root@db01 ~]# [ -f oldboy ] && echo 0||echo 1

0

目录文件：（测试文件类型）

[root@db01 ~]# mkdir oldgirl

[root@db01 ~]# [ -d oldgirl ] && echo 0||echo 1

0

[root@db01 ~]# [ -f oldgirl ] && echo 0||echo 1

1

[root@db01 ~]# [ -e oldgirl ] && echo 0||echo 1

0

测试文件属性举例：

[root@db01 ~]# ls -l oldboy

-rw-r--r-- 1 root root 0 1月 7 13:42 oldboy

[root@db01 ~]# [ -r oldboy ] && echo 0 ||echo 1

0

[root@db01 ~]# [ -w oldboy ] && echo 0 ||echo 1

0

[root@db01 ~]# [ -x oldboy ] && echo 0 ||echo 1

1

测试shell变量举例：

首先我们定义file1和file2两个变量，并分别赋予两个系统文件路径及文件名的\*

[root@db01 ~]# file1=/etc/services ;file2=/etc/rc.local

[root@db01 ~]# echo $file1 $file2

/etc/services /etc/rc.local

范例1：对单个文件变量的测试：

[root@db01 ~]# [ -f "$file1" ] && echo 1||echo 0 #文件存在且为普通文件所以为真（1）

1

[root@db01 ~]# [ -d "$file1" ] && echo 1||echo 0 #文件存不是目录所以为假（0）

0

[root@db01 ~]# [ -s "$file1" ] && echo 1||echo 0 #文件存在且大小不为0，所以为真（1）

1

[root@db01 ~]# [ -e "$file1" ] && echo 1||echo 0 #文件存在所以为真（1）

1

范例2：对单个目录变量的测试

[root@db01 ~]# dir1=/etc

[root@db01 ~]# echo $dir1

/etc

[root@db01 ~]# [ -e "$dir1" ] && echo 1 || echo 0

1

[root@db01 ~]# [ -w "$dir1" ] && echo 1 || echo 0

1

**特殊例子：如果变量不加双引号，测试结果可能不正确：**

[oldboy@db01 ~]$ echo $file7

[oldboy@db01 ~]$ [ -f "$file7" ] && echo 1 || echo 0

0

[oldboy@db01 ~]$ [ -f $file7 ] && echo 1 || echo 0

1

#明明不存在，却返回正确结果。

#变量一定要加双引号。

范例3：把变量内容换成实体文件路径测试

[root@db01 ~]# [ -f /etc/services ] && echo 1 ||echo 0

1

[root@db01 ~]# [ -f "/etc/service" ] && echo 1 ||echo 0

0

[root@db01 ~]# [ -f "/etc/services" ] && echo 1 ||echo 0

1

提示：如果是文件实体路径加引号与不加引号结果是一样的。

范例4（生产）：生产环境系统NFS启动脚本的条件测试内容

# Source networking configuration.

[ -f /etc/sysconfig/network ] && . /etc/sysconfig/network

#如果/etc/sysconfig/network文件存在就加载文件

# Check for and source configuration file otherwise set defaults

范例5：简易高效的文件判断例子：

在做测试判断时，不一定非要按照前面的方法。直接用后者做测试判断有时更简洁。例如：

[root@db01 ~]# file1="/etc/rc.local"

[root@db01 ~]# [ -f "$file1" ] && echo 1

1

[root@db01 ~]# [ -f "$file3" ] || echo 0 #file3没有定义变量，所以返回错误的结果（0）

0

系统范例6：/etc/init.d/nfs

[ -x /usr/sbin/rpc.nfsd ] || exit 5

[ -x /usr/sbin/rpc.mountd ] || exit 5

[ -x /usr/sbin/exportfs ] || exit 5

#### 1.3.2.3 特殊文件测试表达式写法案例

范例7：条件表达式判断条件后面执行多条命令语法写法。

范例7.1：当条件1成立时，同时执行命令1、2、3.

用法：

[ 条件1 ]&& {

命令1

命令2

命令3

}

示例：

[root@db01 ~]# [ -f /etc/hosts ] && { echo 1;echo 2;echo 3; }

1

2

3

上面判断相当于下面if语句的效果

if [条件1]

then

命令1

命令2

命令3

fi

范例8：当条件不成立时，执行多条语句，使用逻辑操作符“||”

[root@db01 ~]# [ -d /etc/hosts ] || { echo 1;echo 2;echo 3; }

1

2

3

或者使用脚本：

[root@db01 2016-01-07]# cat 11.sh

#!/bin/bash

[ -d /etc/hosts ] || {

echo 1

echo 2

echo 3

}

[root@db01 2016-01-07]# sh 11.sh

1

2

3

另一种写法：

[root@db01 2016-01-07]# cat 22.sh

[ 3 -ne 3 ] ||{

echo "I am oldboy"

echo "I am oldgirl"

exit 1

}

[root@db01 2016-01-07]# sh 22.sh

I am oldboy

I am oldgirl

如果写在一行里面，里面的每个命令还需要用分号结尾，如下所示：

[root@db01 2016-01-07]# cat 22.sh

[ 3 -ne 3 ] ||{ echo "I am oldboy";echo "I am oldgirl";exit 1; }

[root@db01 2016-01-07]# sh 22.sh

I am oldboy

I am oldgirl

提示：本例的用法很简洁，但是理解起来不如if条件句容易，因此，请视情况而定。

### 1.3.3 字符串测试表达式

#### 1.3.3.1 字符串测试操作符

字符串测试操作符的作用：比较两个字符串是否相同、字符串长度是否为零，字符串是否为null（注：bash区分零长度字符串和空字符串）等。

“=”比较两个字符串是否相同，与==等价，如if[“$a”=”$b”]，其中$a这样的变量最好用””括起来，因为如果中间有空格，\*等符号就可能出错了，当然更好的方法就是[ “${a}”=”${b}”]。“！=”比较两个字符串是否相同，不同则为“是”。

在书写测试时，可以使用下表中的字符串测试操作符。

下表为字符串测试操作符

|  |  |
| --- | --- |
| **常用字符串测试操作符** | **说明** |
| -z “字符串” | 若串长度为0则真，-z可以理解为zero |
| -n “字符串” | 若串长度不为0则真，-n可以理解为 no zero |
| “串1”“=”“串2” | 若串1等于串2则真，可以使用“==”代替“=”。 |
| “串1”“！=”“串2” | 若串1不等于串2则真，可以使用“==”代替“=”。 |
| **特别注意：**  **1、以上表格中的字符串测试操作符号务必要用“”引起来。**  **2、比较符号两端有空格。** | |

字符串测试操作符提示：

1） -n比较字符串长度是否不为0，如果不为0则为真，如：[ -n “$myvar” ]

2） -z比较字符串长度是否等于0，如果等于0则为真，如：[ -z “$myvar” ]

**特别注意，对于以上表格中的字符串测试操作符号，如：[ -n “$myvar” ]，要把字符串用“”引起来。**

注意事项：

1、字符串或字符串变量比较都要加双引号之后在比较。

2、字符串或字符串变量比较，比较符号两端最好都有空格。

参考系统脚本：

[root@db01 2016-01-07]# sed -n '30,31p' /etc/init.d/network

# Check that networking is up.

[ "${NETWORKING}" = "no" ] && exit 6

举例演示：

[root@db01 2016-01-07]# [ -n "abc" ] && echo 1 || echo 0

1

[root@db01 2016-01-07]# [ -n "" ] && echo 1 || echo 0

0

[root@db01 2016-01-07]# [ -z "" ] && echo 1 || echo 0

1

[root@db01 2016-01-07]# [ -z "abc" ] && echo 1 || echo 0

0

[root@db01 2016-01-07]# [ "abc" != "abc" ] && echo 1 || echo 0

0

[root@db01 2016-01-07]# [ "abc" = "abc" ] && echo 1 || echo 0

1

[root@db01 2016-01-07]# [ "abc" = "abcd" ] && echo 1 || echo 0

0

[root@db01 2016-01-07]# test="abc"

[root@db01 2016-01-07]# test1="fds"

[root@db01 2016-01-07]# echo $test $test1

abc fds

[root@db01 2016-01-07]# [ "$test" = "$test1" ] && echo 1|echo 0

[root@db01 2016-01-07]# [ "$test" = "$test1" ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ "$test" != "$test1" ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ "${#test}" = "${#test1}" ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ "${#test}" = "${#test1}" -a "abc" = "abc" ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [[ "${#test}" = "${#test1}" -a "abc" = "abc" ]] && echo 1||echo 0

-bash: syntax error in conditional expression

-bash: syntax error near `-a'

[root@db01 2016-01-07]# [[ "${#test}" = "${#test1}" && "abc" = "abc" ]] && echo 1||echo 0

1

变量字符串演示：

[root@db01 2016-01-07]# lifen="mazi"

[root@db01 2016-01-07]# [ -n "$lifen" ] && echo 1 || echo 0

1

[root@db01 2016-01-07]# [ -z "$lifen" ] && echo 1 || echo 0

0

不带双引号的例子：

[root@db01 2016-01-07]# [ -n abc ] && echo 1 ||echo 0

1

[root@db01 2016-01-07]# [ -n ] && echo 1 ||echo 0

1

[root@db01 2016-01-07]# [ -z abc ] && echo 1 ||echo 0

0

[root@db01 2016-01-07]# [ -z ] && echo 1 ||echo 0

1

[root@db01 2016-01-07]# [ -n $lifen ] && echo 1 || echo 0

1

[root@db01 2016-01-07]# [ -z $lifen ] && echo 1 || echo 0

0

### 1.3.4 整数二元比较操作符

在书写测试表达式时，可以使用下表中的整数二元比较操作符

下表中整数二元比较操作符

|  |  |  |
| --- | --- | --- |
| **在[]以及test中使用的比较符** | **在(())和[[]]中使用的比较符** | **说明** |
| -eq | ==或= | Equal的缩写，相等。 |
| -ne | != | Not equal的缩写，不相等。 |
| -gt | > | 大于greater than |
| -ge | >= | 大于等于greater equal |
| -lt | < | 小于，类似less than |
| -le | <= | 小于等于，less equal |

提示：

1） “<”符号意思是小于，if[[“$a”< “$b”]]，if[“$a” \< “$b”]。在单[]中需要转义，因为shell也用<和>重定向。

2） “>”符号意思是大于，if[[“$a”> “$b”]]，if[“$a” \> “$b”]。在单[]中需要转义，因为shell也用<和>重定向。

3) “=”符号意思是等于，if[[“$a”= “$b”]]，if[“$a” = “$b”]。在单[]中不需要转义。

特别提示：

经过实践，“=”和“！=”在[]中使用不需要转义，包含“>”和“<”的符号在[]中使用需要转义，对于数字不转义的结果未必会报错，但是结果可能不会对。

范例1：二元数字比较

[root@db01 2016-01-07]# [ 2 > 1 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ 2 < 1 ] && echo 1||echo 0

1 ###这里的结果逻辑不对，条件不成立，应该返回0.

[root@db01 2016-01-07]# [ 2 \< 1 ] && echo 1||echo 0

0 ###转义后是正确的。

[root@db01 2016-01-07]# [ 2 \> 1 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ 2 -gt 1 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ 2 -lt 3 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ 2 -eq 3 ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ 2 -le 3 ] && echo 1||echo 0

1

**实际测试结果结论：**

1、 整数加双引号也是对的。

2、 [[]]用-eq等的写法也是对的。

3、 []用>号写法语法没错，逻辑结果不对。

工作中：推荐使用[]的-eq的用法。

#### 1.3.4.1 整数变量测试举例：

范例1：整数变量条件测试举例

[root@db01 2016-01-07]# a1=10;a2=13

[root@db01 2016-01-07]# [ $a1 = $a2 ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ $a1 \> $a2 ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ $a1 \< $a2 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ $a1 -eq $a2 ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ $a1 -gt $a2 ] && echo 1||echo 0

0

[root@db01 2016-01-07]# [ $a1 -lt $a2 ] && echo 1||echo 0

1

[root@db01 2016-01-07]# [ $a1 -le $a2 ] && echo 1||echo 0

1

小结：整数比较推荐下面用法：

[ $num1 –eq $num2 ] #==注意空格，和比较符号。

(($num1>$num2)) #无需空格，常规数学比较符号。

系统脚本例子：

### 1.3.5 逻辑操作符

在书写测试表达式时，可以使用下表的逻辑操作符实现复杂

下表为逻辑连接符

|  |  |  |
| --- | --- | --- |
| **在[]和test中使用的逻辑操作符** | **在[[]]中使用的逻辑操作符** | **说明** |
| -a | && | And与，两端都为真，则真。 |
| -o | || | Or或，两端有一个为真则真。 |
| ! | ! | Not非，相反则为真。 |

提示：

! 中文意思是反：与一个逻辑值相反的逻辑值。

-a 中文意思是与（and&&）：两个逻辑值都为“真”，返回值才为“真”，反之为“假”。

-o 中文意思是或（or||）：两个逻辑值只要有一个为“真”，返回值就为“真”。

逻辑操作符运算规则：

**结论：-a和&&的运算规则：只有两端都是1才为真。**

真true1 假false0

------------

and 1\*0=0 假

and 0\*1=0 假

**and 1\*1=1 真**

and 0\*0=0 假

只有两端都是1才为真，and为交集。

**结论：-o或||两端都是0才为假，任何一段不为0都是真。**

------------

or 1+0=1 真

or 1+1=2 真

or 0+1=1 真

**or 0+0=1 假**

两端都是0才为假，不为0就是真。or为并集。

#### 1.3.5.1 逻辑操作符的举例

例子演示：

[root@db01 2016-01-07]# f1=/etc/rc.local ;f2=/etc/services

[root@db01 2016-01-07]# [ -f "$f1" && -f "$f2" ]&& echo 1||echo 0

-bash: [: missing `]'

0

[root@db01 2016-01-07]# [ -f "$f1" -a -f "$f2" ]&& echo 1||echo 0

1

[root@db01 2016-01-07]# [ -n "$f1" -a -z "$f2" ]&& echo 1||echo 0

0

[root@db01 2016-01-07]# [[ -f "$f1" && -f "$f2" ]]&& echo 1||echo 0

1

[root@db01 2016-01-07]# [[ -f "$f1" || -f "$f2" ]]&& echo 1||echo 0

1

范例2：多文件单重括号[]与或非测试

可用与（-a和&&）、或（-o和||）、非（！）将多个条件表达式连接起来，接着上面的变量测试。

[root@db01 2016-01-07]# [ -f "$f1" -o -f"$f2" ]&& echo 1||echo 0

1

[root@db01 2016-01-07]# [ -f "$f1" -a -f"$f2" ]&& echo 1||echo 0

1

[root@db01 2016-01-07]# [[ -f "$f1" && -f"$f2" ]]&& echo 1||echo 0

1

[root@db01 2016-01-07]# [[ -f "$f1" || -f"$f2" ]]&& echo 1||echo 0

1

提示：

1、 “a”和“o”逻辑操作符号用于[]中使用。

2、 “&&”和“||”逻辑操作符号用于[[]]中使用。

3、 注意括号两端，必须要有空格。

系统案例：

[root@db01 2016-01-07]# sed -n '87,90p' /etc/init.d/nfs

[ "$NFSD\_MODULE" != "noload" -a -x /sbin/modprobe ] && {

/sbin/modprobe nfsd

[ -n "$RDMA\_PORT" ] && /sbin/modprobe svcrdma

}

**小结：逻辑操作符使用总结**

[]中用-a，-o,！

[[]]中用&&，||，！

test用法和[]相同。

多个[]之间以及多个[[]]之间，或者任意混合中间逻辑操作符都是&&或||。

案例：利用条件表达式完成如下考试题：

通过传参两个参数，比较两个整数大小

[root@db01 2016-01-07]# cat 33.sh

#!/bin/bash

#no.1

[ $# -ne 2 ]&&{

echo "USAGE $0 num1 num2"

exit 11

}

#no.2

expr $1 + 1 &>/dev/null

RETVAL1=$?

expr $2 + 1 &>/dev/null

RETVAL2=$?

[ $RETVAL1 -ne 0 -a $RETVAL2 -ne 0 ]&&{

echo "pls input again."

exit 22

}

[ $RETVAL1 -ne 0 ]&&{

echo "the first num is not int,pls input again."

exit 33

}

[ $RETVAL2 -ne 0 ]&&{

echo "the second num is not int,pls input again."

exit 44

}

#no.3

[ $1 -lt $2 ]&&{

echo "$1<$2"

exit 0

}

[ $1 -eq $2 ]&&{

echo "$1=$2"

exit 0

}

[ $1 -gt $2 ]&&{

echo "$1>$2"

exit 0

}

##### 综合实战例：开发shell脚本分别实现以定义变量，脚本传参以及read读入的方式比较2个整数大小。用条件表达式（禁止if）进行判断并以屏幕输出的方式提醒用户比较结果。注意：一共是开发3个脚本。当用脚本传参以及read读入的方式需要对变量是否为数字、并且传参个数不对给予提示。

#!/bin/sh

#no.1

read -t 10 -p "Pls input the num you want:" a b

[ -z "$a" -a -z "$b" ]&&{

echo "Pls input two num"

exit 1

}

#no.2

expr $a + 1 &>/dev/null

[ $? -ne 0 ]&&{

echo "the first input int"

exit 2

}

expr $b + 1 &>/dev/null

[ $? -ne 0 ]&&{

echo "the second input int"

exit 3

}

#no.3

[ "$a" -eq "$b" ]&&{

echo "$a = $b"

exit 0

}

[ "$a" -lt "$b" ]&&{

echo "$a < $b"

exit 0

}

[ "$a" -gt "$b" ]&&{

echo "$a > $b"

exit 0

}

### 1.3.6 shell变量的输入

变量的输入3种方式：

1、定义a=1

2、传参方式$1

3、read交互式读入

shell变量除了可以**直接赋值**或**脚本传参**外，还可以使用read命令，read命令为内置命令，通过help read查看帮助。

【语法格式】

read [参数] [变量名]

【常用参数】

-p prompt:设置提示信息。

-t timeout:设置输入等待的时间，单位默认为秒。

范例1：read的基本读入：

[root@db01 2016-01-07]# read -p "Pls input a character" a

Pls input a character haha

脚本示例：

[root@db01 2016-01-07]# cat read.sh

#!/bin/sh

read -t 5 -p "Pls input a character" a

echo "your input is:$a"

[root@db01 2016-01-07]# sh read.sh

Pls input a character hehe

your input is:hehe

综合实例：打印选择菜单，一键安装web服务：

[root@db01 2016-01-07]# cat menu.sh

1. [install lamp]

2. [install lnmp]

3. [exit]

pls input the num you want:

要求：

1、当用户按照要求输入1时，输出“start installing lamp.”然后执行/server/scripts/lamp.sh，脚本内容输出“lamp is installed”后退出脚本；

2、当用户按照要求输入2时，输出“start installing lnmp.”然后执行/server/scripts/lnmp.sh，脚本内容输出“lnmp is installed”后退出脚本；

3、当输入3时，退出当前菜单及脚本。

4、当输入任何其他字符，给出提示“input error”后退出脚本。

5、要对执行的脚本进行相关条件判断，例如：脚本是否存在，是否可执行等。

例子：

[root@db01 2016-01-07]# cat menu.sh

#!/bin/sh

#no.1 menu

cat << LL

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

1. [install lamp]

2. [install lnmp]

3. [exit]

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

LL

#no.2

read -t 20 -p "Pls input the num you want" num

[ "$num" != "1" -a "$num" != "2" -a "$num" != "3" ]&&{

echo "input error"

exit

}

#no.3

[ -f /server/scripts/lamp.sh ]||{

echo " Please input the right script"

exit 1

}

[ $num -eq 1 ]&&{

echo "install lamp"

/bin/sh /server/scripts/lamp.sh

exit

}

[ -f /server/scripts/lnmp.sh ]||{

echo " Please input the right script"

exit 1

}

[ $num -eq 2 ]&&{

echo "install lnmp"

/bin/sh /server/scripts/lnmp.sh

exit

}

[ $num -eq 3 ]&&{

echo "bey!"

exit

}

## 1.4 分支与循环结构

if语句时实际生产工作中最重要且最常用的语句，所以，必须掌握牢固。

### 1.4.1 if条件句

#### 1.4.1.1 if条件句语法

1、但分支结构

语法：

if [ 条件 ]

then

指令

fi

或

if [ 条件 ];then

指令

fi

if但分支条件中文编程语法：

如果 [ 你有房 ]

那么

我就嫁给你

果如

提示：分号相当于命令换行，上面两种语法等同。

特殊写法：if[ -f “$file1” ];then echo1;fi 相当于：[ -f “$fil1” ]&& echo 1

if[ -f “$fil1” ];then

echo 1

fi

2、双分支结构

语法：

if 条件

then

指令集

else

指令集

fi

特殊写法：if[ -f “$file1” ];then echo1;else echo 0;fi 相当于：

[ -f “$file1” ]&& echo 1 ||echo 0

if双分支中文编程语法：

如果 [ 你有房 ]

那么

我就嫁给你。

否则

good bye！

果如

3、多分支结构

语法：

if 条件1

then

指令1

elif 条件2

then

指令2

else

指令3

fi

-----------------多个elif-----------------

if 条件

then

指令

elif 条件

then

指令

……

else

指令

fi

提示：

1） 注意多分支elif的写法elif条件;then，不要落下了then。

2） 结尾的else后面没有then。

多分支if语句中文编程语法：

如果 [ 你有房 ]

那么

我就嫁给你

**或者如果 [ 你爸是李刚 ]**

**那么**

**我就嫁给你。**

**或者如果 [ 你的活儿好 ]**

**那么**

**我们可以先谈男女朋友**

否则

不鸟你

果如

1.4.1.2 但分支if条件句举例

下面举几个使用if条件句的例子

范例1：请把下面文件条件表达式判断语句改成if语句

[root@db01 ~]# [ -f /etc/hosts ]&& echo 1

1

解答：

[root@db01 2016-01-07]# cat 44.sh

#!/bin/sh

if [ -f /etc/hosts ]

then

echo 1

fi

[root@db01 2016-01-07]# sh 44.sh

1

范例2：开发shell脚本判断系统剩余内存大小，如果低于100M就邮件报警，并且加入系统定时任务没3分钟执行一次检查。

解答：重视解决问题的过程（第一关，第二关，第三关）

1、如何去内存，去内存哪个值（buffers）

[root@db01 2016-01-07]# free -m|awk 'NR==3 {print $NF}'

796

[root@db01 2016-01-07]# free -m|awk -F " " 'NR==3 {print $4}'

796

2、发邮件mail，mutt。

[root@db01 2016-01-07]# echo -e "set from=znthskddf12@163.com smtp=smtp.163.com smtp-auth-user=znthskddf12 smtp-auth-password=789wynz123 smtp-auth=login" >>/etc/mail.rc

[root@db01 2016-01-07]# /etc/init.d/postfix start

启动 postfix： [确定]

[root@db01 2016-01-07]# echo "oldboy"|mail -s "test" znthskddf12@163.com

[root@db01 2016-01-07]# echo oldboy >/tmp/test.txt

[root@db01 2016-01-07]# mail -s "test" znthskddf12@163.com </tmp/test.txt

实战操作：

1、先在命令行把条件取出来

[root@db01 ~]# free -m|awk 'NR==3 {print $4}'

839

[root@db01 ~]# free -m|grep buffers\/

-/+ buffers/cache: 141 839

2、编写脚本

[root@db01 ~]# cat 11.sh

#!/bin/sh

cur\_free=$(free -m|awk 'NR==3 {print $4}')

chars="current memory is $cur\_free"

if [ $cur\_free -lt 1000 ]

then

echo $chars|mail -s "$chars" znthskddf12@163.com

fi

提示：

1） 测试时，没办法把内存调小，可以把阀值调大，例如100M改成1000M。

2） 注意开启邮件服务。PC虚拟机测试非自己公司的邮箱可能会遇到无法收到报警邮件的问题。

做一个测试mysql数据库状态的脚本：

[root@db01 2016-01-08]# cat mysql.sh

#!/bin/sh

#no.1

. /etc/init.d/functions

mysql=$(ss -lntup|grep 3309|wc -l)

if [ $mysql -eq 1 ]

then

echo "mysql is runing..."

else

echo "mysql is stop"

echo "mysql Ready to start"

sleep 3

/data/3309/mysql start &>/dev/null

[ $? -ne 0 ]&&echo "mysql start error "||{

action "mysql start " /bin/true

}

fi

**回顾下监控MySQL数据库是否异常的多种方法**

1、根据MySQL端口号监控MySQL（本地）。

此处是本地监控，端口在，服务可能不正常，例如：负载很高，CPU很高，连接数满了，也可以远程监控。

本地：ss,netstat,lsof  
远程：telnet,nmap,nc  
参考博文：http://oldboy.blog.51cto.com/2561410/942530

2、根据MySQL进程监控MySQL

ps –ef|grep mysql|grep –v grep|wc –l

注意：脚本名字里不能含有过滤字符串mysql，否则，不准。

3、通过MySQL客户端命令及用户账户连接MySQL

mysql –u –p –e “select version();” &>/dev/null

4、通过php/Java程序URL方式监控MySQL

查看远端的端口是否通畅3个简单实用案例！  
http://oldboy.blog.51cto.com/2561410/942530

小结：老男孩linux Web服务监控手段：  
1、端口  
本地：ss,netstat,lsof  
远程：telnet,nmap,nc  
echo -e "\n"|telnet baidu.com 80|grep Connected  
nmap 10.0.0.5 -p 3307|grep open|wc –l   
nc -w 2 10.0.0.5 3306 &>/dev/null  
查看远端的端口是否通畅3个简单实用案例！  
http://oldboy.blog.51cto.com/2561410/942530  
2、本地进程数  
3、header(http code) curl -I web地址返回200就OK。  
掌握技术思想比解决问题本身更重要  
http://oldboy.blog.51cto.com/2561410/1196298  
4、URL（wget,curl）web地址，模拟用户的方式。  
5、php,java写一个程序，模拟用户的方式监控（让开发提供）。

范例：

面试及实战考试题：监控web站点目录（/var/html/www）下所有文件是否被恶意篡改，如果有就打印改动的文件名（发邮件），定时任务没3分钟执行一次（10分钟时间完成）

## 1.5 shell函数

### 1.5.1 为什么要使用shell函数

讲解函数前，回顾下别名的作用：

[root@db01 www]# alias nginx='/application/nginx/nginx'

[root@db01 www]# pkill nginx

[root@db01 www]# nginx

提示：输入nginx指令就相当于执行/application/nginx/nginx

函数也是具有和别名类似的功能：

简单地说，函数的作用就是把程序里多次调用相同代码部分定义成一份，然后为这一份代码起个名字，其他所有的重复调用这部分代码就都只调用这个名字就可以了。当需要修改这部分重复代码时，只需要改变函数体内的一份代码即可实现所有调用修改。

**使用函数的优势：**

1、 把相同的程序段定义成函数，可以减少整个程序的代码量。

2、可以让程序代码结构更清晰。

3、 增加程序的可读、易读型。以及可管理性。

4、 可以实现程序功能模块化，不同的程序使用函数模块化。

强调： 对于shell来说，Linux系统的2000个命令都可以说是shell的函数。

### 1.5.2 shell函数语法

语法格式：

简单语法格式：

函数名(){

指令...

return n

}

规范语法格式：

**function** 函数名(){

指令...

return n

}

提示：shell的返回值是exit输出返回值，而函数里用return输出返回值。

### 1.5.3 shell函数的执行

调用函数

1） 直接执行函数名即可（不带括号）。

函数名

注意：

a. 执行函数时，函数后的小括号不要带了。

b. 函数定义及函数体必须在要执行的函数名的前面定义，先定义函数，在进行使用。

2） 带参数的函数执行方法：

函数名 参数1 参数2

提示：函数的传参和脚本的传参类似，只是脚本名换成函数名即可。

【函数后接的参数说明】

* shell的位置参数（$1、$2、$3、$4、$#、$\*、$?以及$@）都可以是函数的参数。
* 此时父脚本的参数临时的被函数参数所掩盖或隐藏。
* **$0比较特殊，它仍然是父脚本的名称。**
* 当函数完成时，原来的命令行脚本的参数即恢复。
* 在shell函数里面，return命令功能与shell里的exit类似，作用是跳出函数。
* 在shell函数体里使用exit会退出整个shell脚本，而不是shell函数。
* return语句会返回一个退出值（返回值）给调用函数的程序。
* 函数的参数变量是在函数体里面定义，如果是普通变量一般会使用local i定义。

### 1.5.4 shell函数范例：

范例1：开发脚本建立两个简单函数并调用执行

[root@db01 2016-01-09]# cat fun01.sh

#!/bin/sh

#定义两个函数，名字为oldboy和oldgirl

oldboy(){

echo "I am oldboy!"

}

function oldgirl(){

echo "I am oldgirl"

}

oldboy #在一个脚本里执行函数名调用函数。

oldgirl #在一个脚本里执行函数名调用函数。

[root@db01 2016-01-09]# sh fun01.sh

I am oldboy!

I am oldgirl

范例2：把函数体和执行的脚本分离（更规范的方法）

1） 首先建立函数库脚本（默认不会执行函数）

[root@db01 2016-01-09]# cat fun01.sh

#!/bin/sh

#定义两个函数，名字为oldboy和oldgirl

oldboy(){

echo "I am oldboy!"

}

function oldgirl(){

echo "I am oldgirl"

}

[root@db01 2016-01-09]# ll fun01.sh

-rwxr-xr-x 1 root root 155 1月 9 23:41 fun01.sh

2） 开发执行脚本调用上述函数

[root@db01 2016-01-09]# cat test.sh

#!/bin/sh

[ -x /server/scripts/2016-01-09/fun01.sh ]&& . . /server/scripts/2016-01-09/fun01.sh|| exit

#提示：可以用source或.来加载脚本fun01.sh中的命令变量参数等。

oldboy

oldgirl

提示：

1） 注意调用函数的函数名，及函数脚本路径的写法，不带括号。

2） 函数一定要在调用之前定义。

3） 执行

[root@db01 2016-01-09]# sh test.sh

I am oldboy!

I am oldgirl

范例3：函数传参转成脚本命令行传参，对任意指定URL判断是否异常。

分步解答：

1、 脚本传参检查web URL是否正常。

2、 检查的功能写成函数。

3、 函数传参转成脚本命令行传参，对任意指定URL判断是否异常。

方法1：

[root@db01 2016-01-10]# cat 1.sh

#!/bin/sh

#no.1

USAGE(){

echo "UGAGE: $0 arg1"

exit

}

#no.2

cheak(){

curl -s $1 &>/dev/null

if [ $? -ne 0 ]

then

echo "web $1 is not ok.."

else

echo "web $1 is turn..."

fi

}

#no.3

web(){

if [ $# -ne 1 ]

then

USAGE

fi

cheak $1

}

web $\*

法2：

[root@db01 2016-01-12]# cat 3.sh

#!/bin/sh

[ -f /etc/init.d/functions ]&& . /etc/init.d/functions

usage(){

echo "USAGE:$0 ar1"

exit 1

}

checkUrl(){

RETVAL=0

wget -T 10 --spider -t 2 $1 &>/dev/null

RETVAL=$?

if [ $RETVAL -eq 0 ]

then

action "$1 url" /bin/true

else

action "$1 url" /bin/false

fi

return $RETVAL

}

main(){

if [ $# -ne 1 ];then

usage

fi

checkUrl $1

}

main $\*

## 1.6 case结构条件句

### 1.6.1 case结构条件句语法

case就是一个多分支的if语句

case “字符串变量” in

值1） 指令1...

;;

值2） 指令2...

;;

\*） 指令...

esac

中文编程语法：

case “找女朋友条件”in

身材好） 娶你...

;;

很漂亮） 娶你...

;;

活儿好） 可以考虑先谈朋友..

;;

\*) bye ！！..

esac

提示：case语句相当于一个if的多分支结构语句。

### 1.6.2 case结构条件句范例：

范例1：根据用户的输入判断是哪个数字（case-1.sh）

如果用户输入1或2或3，则输出对应输入的数字，如果是其他内容，返回不正确，退出。

[root@db01 2016-01-10]# cat 2.sh

#!/bin/sh

read -t 10 -p "pleases input one num:" int

case "$int" in

1) echo "the num you input is 1"

;;

2) echo "the num you input is 2"

;;

[3-9])

echo "the num you input is $int"

;;

\*)

echo "the num you input must be less 9."

exit;

esac

if 语句实现：

[root@db01 2016-01-10]# cat 3.sh

#!/bin/sh

read -t 10 -p "please input noe int:" num

if [ $num -eq 1 ]

then

echo "the num you input is 1"

elif [ $num -eq 2 ]

then

echo "the num you input is 2"

elif [ $num -ge 3 -a $num -le 9 ]

then

echo "the num you input is $num"

else

echo "USAGE: $0 please input shu zi !!!"

exit

fi

范例2：执行脚本打印一个水果菜单如下：

1. apple

2. pear

3. banana

4. cherry

当用户选择水果的时候，打印告诉它选择的水果是什么，并给水果单词加上一种颜色。

要求用case语句实现。

解答：

热身颜色例子：

[root@db01 2016-01-10]# cat yanse.sh

#!/bin/sh

RED\_COLOR='\E[1;31m'

GREEN\_COLOR='\E[1;32m'

YELLOW\_COLOR='\E[1;33m'

BLUE\_COLOR='\E[1;34m'

RES\_COLOR='\E[0m'

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

echo -e "$YELLOW\_COLOR gongli $RES"

方法1：

[root@db01 2016-01-10]# cat 4.sh

#!/bin/sh

. /server/scripts/2016-01-10/yanse.sh

cat << EOF

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

1. apple

2. pear

3. banana

4. cherry

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

EOF

read -t 10 -p "please Choose a kind of fruit:" fruit

RED\_COLOR='\E[1;31m'

GREEN\_COLOR='\E[1;32m'

YELLOW\_COLOR='\E[1;33m'

BLUE\_COLOR='\E[1;34m'

RES\_COLOR='\E[0m'

case "$fruit" in

1)

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

;;

2)

echo -e "$GREEN\_COLOR this is pear $RES\_COLOR"

;;

3)

echo -e "$YELLOW\_COLOR this is banana $RES\_COLOR"

;;

4)

echo -e "$BLUE\_COLOR this is banana $RES\_COLOR"

;;

\*)

echo "Please select right num{1|2|3|4}"

exit

esac

提示：

1、 输出带颜色的句子使用 echo -e

带函数版本：

[root@db01 2016-01-10]# cat 4.sh

#!/bin/sh

add(){

RED\_COLOR='\E[1;31m'

GREEN\_COLOR='\E[1;32m'

YELLOW\_COLOR='\E[1;33m'

BLUE\_COLOR='\E[1;34m'

RES\_COLOR='\E[0m'

case "$1" in

red|RED)

echo -e "$RED\_COLOR $2 $RES\_COLOR"

;;

green|GREEN)

echo -e "$GREEN\_COLOR $2 $RES\_COLOR"

;;

yellow|YELLOW)

echo -e "$YELLOW\_COLOR $2 $RES\_COLOR"

;;

blue|BLUE)

echo -e "$BLUE\_COLOR $2 $RES\_COLOR"

;;

\*)

echo "please input num.."

exit

esac

}

menu(){

cat << EOF

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

1. apple

2. pear

3. banana

4. cherry

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

EOF

}

frest(){

read -t 10 -p "please Choose a kind of fruit:" fruit

case "$fruit" in

1)

add red apple

;;

2)

add green pear

;;

3)

add yellow banana

;;

4)

add blue cherry

;;

\*)

echo "Please select right num{1|2|3|4}"

exit

esac

}

main(){

menu

frest

}

main

使用if实现：

[root@db01 2016-01-12]# cat 4.sh

#!/bin/sh

RED\_COLOR='\E[1;31m'

GREEN\_COLOR='\E[1;32m'

YELLOW\_COLOR='\E[1;33m'

BLUE\_COLOR='\E[1;34m'

RES\_COLOR='\E[0m'

usage(){

echo "USAGE:$0 {red|green|yellow|prink}" contents

exit 1

}

color(){

if [ "$1" = "red" ];then

echo -e "${RED\_COLOR}$2${RES\_COLOR}"

elif [ "$1" = "green" ];then

echo -e "$GREEN\_COLOR $2 $RES\_COLOR"

elif [ "$1" = "yellow" ];then

echo -e "$YELLOW\_COLOR $2 $RES\_COLOR"

else

usage

fi

}

main(){

if [ $# -ne 2 ];then

usage

fi

color $1 $2

}

main $\*

语句小结：

1、 case主要是写启动脚本，范围较窄。

2、 if取值判断、比较、应用更广。

case语句小结：

1. case 语句相当于多分支if语句。case语句优势更规范，易读  
2. case语句适合变量的值少，且为固定的或数字或字符串集合。（1,2,3）或（start，stop）  
3. 系统服务启动脚本传参的判断多用case语句  
  
1. 所有的case语句都可以用if实现，但是case更规范清晰一些  
2. case一般适合服务的启动脚本  
3. case的变量的值如果一直固定的start/stop/restart元素的时候比价适合一些

## 1.7 当型循环和直到型循环

while循环工作中使用的不多，一般是守护进程程序或始终循环执行场景，其他循环计算，替换while。

### 1.7.1 和直到型循环语法

1. while 条件句

语法：

while 条件

do

指令

done

提示：

手机充值：发短信扣费，先充值100，每次扣1角5，当费用低于一角，就不能发了。

2． until条件句

语法：

until 条件

do

指令...

done

提示：until应用场合不多见，了解就好。

### 1.7.2 当型和直到型循环基本范例：

休息命令：sleep1休息1秒，usleep 1000000休息1秒，达到一分就用定时任务。

下面举几个while和until条件句的例子

范例1：每隔两秒记录一次系统负载情况

法1：每隔两秒屏幕输出负载值

[root@db01 2016-01-10]# cat 7.sh

#!/bin/sh

while true

do

uptime

sleep 2

done

法2：追加到log里，使用微秒单位。

[root@db01 2016-01-10]# cat 8.sh

#!/bin/sh

while [ 1 ] #==>条件这里和上面有区别

do

uptime >>./uptime.log

usleep 1000000 #==>这里是以微秒为单位。

done

[root@db01 2016-01-10]# sh 8.sh & #后台执行方法

[1] 38228

提示：在后台永久执行，我们就称之为守护进程模式。

[root@db01 2016-01-10]# tail -f uptime.log

21:56:51 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:52 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:53 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:54 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:55 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:56 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:57 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:58 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:56:59 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

21:57:00 up 1 day, 5:52, 2 users, load average: 0.00, 0.00, 0.00

防止脚本执行中断的方法：

1） sh 8.sh &

2） nohup /server/scripts/uptime.sh &

3） screen，保持会话，总结此命令。

脚本在后台执行知识拓展：

a. 功能和用途见下表格：

|  |  |
| --- | --- |
| **功能** | **用途** |
| Sh while-1-1.sh & | 把脚本while-1.sh放到后台执行 |
| Ctrl+c | 停止执行当前脚本或任务 |
| Ctrl+z | 暂停执行当前脚本或任务 |
| Bg | 把当前脚本或任务放到后台执行，background |
| Fg | 当前脚本或任务拿到前台执行，如果有多个任务，可fg 1 frontground |
| Jobs | 查看当前执行的脚本或任务 |
| Kill | [root@db01 2016-01-10]# jobs  [1]- Running sh 8.sh &  [3]+ Running sh 8.sh &  [root@db01 2016-01-10]# kill %1  [root@db01 2016-01-10]# jobs  [1]- 已终止 sh 8.sh  [3]+ Running sh 8.sh & |

拓展资料：

进程管理：（16个）总结这些命令

bg：后台运行

fg：挂起程序

jobs：显示后台程序

kill,killall,pkill：杀掉进程

crontab：设置定时

ps：查看进程

pstree：显示进程状态树

top：显示进程

nice：改变优先权

nohup：用户退出系统之后继续工作

pgrep：查找匹配条件的进程

**strace：跟踪一个进程的系统调用情况。**

ltrace：跟踪进程调用库函数的情况。

vmstat：报告虚拟内存统计信息。

范例1：

手机充值10元，每发一次短信（输出当前余额）话费1角5分钱，当余额低于1角5分钱不能发送短信，提示余额不足，请充值（可以允许用户充值继续发短信），请用while语句实现。

解答：

单位换算。统一单位，统一成整数

法1：二麻版本：

[root@db01 2016-01-12]# cat mo.sh

#!/bin/sh

TOTAL=1000

MSG=500

IS\_NUM(){

expr $1 + 1 &>/dev/null

if [ $? -ne 0 -a "$1" != "-1" ];then

echo "INVALID INPUT"

return 1

fi

return 0

}

while [ $TOTAL -ge $MSG ]

do

read -p "Pls input your msg:" TXT

read -p "Are you to send?{y|n}" OPTION

case "$OPTION" in

[yY]|[yY][eE][sS])

echo "Send successfully!"

((TOTAL=TOTAL-MSG))

echo "Your have $TOTAL lift!"

;;

[nN]|[nN][oO])

echo "Canceled"

;;

\*)

echo "Invalid Input this msg doesnt send out"

;;

esac

if [ $TOTAL -lt $MSG ];then

read -p "Money is not enough,Are you want to charge?{y|n}" OPT2

case "$OPT2" in

y|Y)

while true

do

read -p "How much are you want to charge?[INT]" CHARGE

IS\_NUM $CHARGE&&break||{

echo "INVALD INPUT"

exit 100

}

done

((TOTAL+=CHARGE)) && echo "you have $TOTAL money."

;;

n|N)

exit 101

;;

\*)

echo "INVALID INPUT!"

exit 102

esac

fi

done

法2：老师版本：

[root@db01 2016-01-12]# cat erma.sh

#!/bin/sh

RED\_COLOR='\E[1;31m'

GREEN\_COLOR='\E[1;32m'

YELLOW\_COLOR='\E[1;33m'

BLUE\_COLOR='\E[1;34m'

PINK='\E[1;35m'

RES='\E[0m'

TOTAL=1000

MSG\_FEE=499

color(){

case "$2" in

red|RED)

echo -e "${RED\_COLOR}$1${RES}"

;;

yellow|YELLOW)

echo -e "${YELLOW\_COLOR}$1${RES}"

;;

green|GREEN)

echo -e "${GREEN\_COLOR}$1${RES}"

;;

blue|BLUE)

echo -e "${BLUE\_COLOR}$1${RES}"

;;

pink|PINK)

echo -e "${PINK\_COLOR}$1${RES}"

;;

\*)

echo "Usage $0 content {red|yellow|blue|green}"

exit

esac

}

function IS\_NUM(){

expr $1 + 1 &>/dev/null

if [ $? -ne 0 -a "$1" != "-1" ];then

return 1

fi

return 0

}

function consum(){

read -p "Pls input your msg:" TXT

read -p "Are you to send?[y|n]" OPTION

case $OPTION in

[yY]|[yY][eE][sS])

color "Send successfully!" yellow

echo $TXT >>/tmp/consum.log

((TOTAL=TOTAL-MSG\_FEE))

color "Your have $TOTAL left!" yellow

;;

[nN]|[nN][oO])

echo "Canceled"

;;

\*)

echo "Invalid Input,this msg doesnt send out"

;;

esac

}

function charge(){

if [ $TOTAL -lt $MSG\_FEE ];then

color "Money is not enough,Are U want to charge?[y|n]" red

read OPT2

case $OPT2 in

y|Y)

while true

do

read -p "How much are you want to charge?[INT]" CHARGE

IS\_NUM $CHARGE&&break||{

echo "INVALID INPUT"

exit 100

}

done

((TOTAL+=CHARGE)) && echo "you have $TOTAL money."

if [ $TOTAL -lt $MSG\_FEE ];then

charge

fi

;;

n|N)

#exit 101

charge

;;

\*)

#echo "INVALID INPUT!"

#exit 102

charge

;;

esac

fi

}

main(){

while [ $TOTAL -ge $MSG\_FEE ]

do

consum

charge

done

}

main

10元=1000分，1角5分=15分

[root@db01 2016-01-11]# cat 1.sh

#!/bin/sh

sun=1000

i=15

while ((sun >= i))

do

((sum=sum - i))

[ $sum -lt $i ] && break

echo "send message,left $sum"

done

echo "money is not enough:$sum"

范例2：

猜数字游戏：首先让系统随机生成一个数字，给这个数字定一个范围（数字前50及后50），让用户输入猜的数字，对输入判断，如果不符合数字就给予高与低的提示，猜对后给下猜对用的次数，请用while语句实现。

范例3：

从1加，加到100

[root@db01 2016-01-11]# cat 4.sh

#!/bin/sh

sum=0

i=1

while [ $i -le 100 ]

do

((sum=sum+i))

((i++))

done

echo $sum

[root@db01 2016-01-11]# sh 4.sh

5050

范例4：实战分析Apache日志例子:

问题1：计算Apache一天的日志access\_2010-12-8.log中所有行的日志各元素的访问字节的总和。给出实现程序。联系日志：见目录下access\_2010-12-8.log，页可以用自己的Apache日志。请用while循环实现。

提示：web日志里有一列记录了访问资源的大小，把这些资源大小相加：

[root@db01 2016-01-11]# cat 5.sh

#!/bin/sh

sum=0

while read line

do

value=`echo $line|awk '{print $10}'`

expr $value + 1 &>/dev/null

[ $? -ne 0 ]&& continue

((sum+=value))

done<access\_2010-12-8.log

echo $sum

while读文件的命令

拓展：while按行读文件的方式：  
方式1：

exec <FILE  
sum=0  
while read line  
do  
    cmd  
done

方式2：

cat ${FILE\_PATH} | while read line  
do  
    cmd  
done

方式3：

while read line  
do  
    cmd  
done<FILE

##### while循环小结：

1、 while循环的特长是**执行守护进程**以及我们希望**循环不退出持续执行**的情况，用于频率小于1分钟循环处理（crond），其他的while循环几乎都可以被我们即将要讲的**for循环替代**。

2、 case语句可以if语句替换，一般在系统启动脚本**传入少量固定规则字符串**，用case语句，其他普通判断多用if。

3、 一句话，if，for语句最常用，其次while（守护进程），case（服务启动脚本）。

##### 各个语句使用场景：

条件表达式，简短的判断（文件是否存在，字符串是否为空等）。

if：取值判断，不同值数量较少的情况。

for：正常的循环处理，最常用。

while：守护进程，死循环（sleep）。

case：服务启动脚本、菜单。

函数：逻辑清晰，减少重复语句。

## 1.8 for循环结构\*\*\*\*\*

### 1.8.1 for循环结构语法

#### 1. for循环结构：

语法：

**for** 变量名 **in** 变量取值列表

**do**

指令...

**done**

提示：在此结构中“in变量取值列表”可省略，省略时相当于in “$@”使用for i就相当于使用 for I in “$@”.

#### 2． C语言型for循环结构

语法：

for((exp1;exp2;exp3))

do

指令...

done

例： for和while对比

[root@db01 2016-01-11]# cat 6.sh

for((i=1;i<=5;i++))

do

echo $i

done

[root@db01 2016-01-11]# cat 7.sh

i=1

while((i<=5))

do

echo $i

((i++))

done

说明：

1） 程序持续运行多用while，包括守护进程，还有配合read读入处理。

2） 有限次循环多用for，工作中for使用更多。

学习方法：记下面的内容。

for 男人 in 世界

do

if [有房] && [有车] && [存款] && [会做家务] && [帅气] && [温柔] && [体贴] && [逛街];then

echo “我喜欢”

else

rm –f 男人

fi

done

很好的方法：

#!/bin/sh

for n in {0..10}

do

echo ssh 1.0.0.$n

done

### 1.8.2 for循环结构基础例子

下面举几个for循环语句的例子

范例1：直接列出变量列表所有元素，打印5、4、3、2、1

法1:直接列元素的方法：

[root@db01 2016-01-11]# cat 6.1.sh

#!/bin/sh

#直接列出变量列表所有元素，打印5、4、3、2、1

for num in 5 4 3 2 1 #==提示：5 4 3 2 1需要空格隔开。

do

echo $num

done

for ip in 10.0.0.18 10.0.0.19

do

echo $ip

done

范例2：获取当前目录下的目录或文件名作为变量列表打印输出

[root@db01 2016-01-11]# cat ls.sh

for filename in `ls`

do

echo $filename

done

[root@db01 2016-01-11]# sh ls.sh

1.sh

2.sh

3.sh

范例3：用for循环批量修改图片扩展名（请把JPG改成GIF）

首先做测试环境：

touch stu\_102999\_1\_finished.jpg

touch stu\_102999\_2\_finished.jpg

touch stu\_102999\_3\_finished.jpg

touch stu\_102999\_4\_finished.jpg

touch stu\_102999\_5\_finished.jpg

[root@db01 1]# ll

总用量 4

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_1\_finished.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_2\_finished.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_3\_finished.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_4\_finished.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_5\_finished.jpg

开始进行操作：

[root@db01 1]# cat 1.sh

#!/bin/sh

file=\*.jpg

for file in `ls \*`

do

rename "\_finished" "" \*.jpg

done

[root@db01 1]# sh 1.sh

[root@db01 1]# ll

总用量 4

-rw-r--r-- 1 root root 81 1月 11 15:28 1.sh

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_1.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_2.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_3.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_4.jpg

-rw-r--r-- 1 root root 0 1月 11 15:30 stu\_102999\_5.jpg

法2：

ls $file|awk -F "\_finished" '{print "mv "$0" " $1 $2}'|bash

法3：

[root@db01 1]# file=stu\_102999\_1\_finished.jpg

[root@db01 1]# echo $file|sed 's#\_finished##g'

stu\_102999\_1.jpg

[root@db01 1]# mv $file `echo $file|sed 's#\_finished##g'`

[root@db01 1]# ll

-rw-r--r-- 1 root root 0 1月 11 15:36 stu\_102999\_1.jpg

[root@db01 1]# cat 2.sh

#!/bin/sh

for file in `ls \*.jpg`

do

mv $file `echo $file|sed 's#\_finished##g'`

done

范例4：计算从1加到100（用C语言型for循环实现）

for((i=1;i<=100;i++))

do

((j=j+i))

done

echo $j

[root@db02 oldboy26]# cat for5.sh  
for((i=1;i<=100;i++))  
do  
  let sum+=i  
done  
echo $sum  
  
for n in `seq 100`  
do  
   ((sum1+=n))  
done  
echo $sum1  
  
echo $((100\*(100+1)/2))

**企业精品shell面试题案例及专家解答精讲  
http://edu.51cto.com/course/course\_id-1511.htm**

### 1.8.3 企业面试重点题目：

问题1：使用for循环在/oldboy目录下批量创建10个文件，名称依次为：

oldboy-1.html

...

oldboy-10.html

解答：

[root@db01 1]# cat 4.sh

#!/bin/sh

[ ! -d /oldboy ] && mkdir /oldboy

for n in `seq 10`

do

touch /oldboy/oldboy-$n.html

done

问题2：用for循环实现将以上文件名中的oldboy全部改成Linux，并且扩展名改为大写。要求for循环的循环体不能出现oldboy字符串。

法1：

[root@db01 1]# cat 5.sh

#!/bin/sh

cd /oldboy

for n in `ls \*.html`

do

rename=`echo $n|awk -F "[-.]" '{print $2}'`

mv $n linux-$rename.HTML

done

法2：

[root@db01 1]# cat 6.sh

#!/bin/sh

cd /oldboy

for n in `ls \*.html`

do

rename=`echo $n|sed -nr 's#^.\*y-(.\*)\.html.\*$#linux-\1.HTML#gp'`

mv $n $rename

done

问题3：批量创建10个系统账号oldboy01-oldboy10并设置密码（密码不能相同）。

**提示：$RANDOM随机数，可以生产0-32767之间的数字。**

[root@db01 2016-01-13]# cat user.sh

#!/bin/sh

> /lifen/11.txt

for n in `seq -w 10`

do

pass=`echo $RANDOM`

useradd oldboy$n

echo $pass|passwd --stdin oldboy$n

echo -e "oldboy$n \t $pass" >> /lifen/11.txt

done

提示：

1） 注意随机数的字符串要定义成变量，否则，每次结果就会不相同。

2） RANDOM产生随机数。提示：8为随机数$((RANDOM+10000000))

问题4：批量创建10个系统账号oldboy01-oldboy10并设置密码（密码为随机8为字符串）。

[root@db01 2016-01-13]# cat user1.sh

#!/bin/sh

[ -f /etc/init.d/functions ]&& . /etc/init.d/functions

> /lifen/11.txt

for n in `seq -w 10`

do

pass=`echo $RANDOM|md5sum|cut -c 2-9`

useradd oldboy$n

action "The user to create a successful" /bin/true

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

action "Password is changed" /bin/true

echo -e "oldboy$n \t $pass" >> /lifen/11.txt

done

拓展小知识：产生随机数的方法：

法1：

[root@db01 2016-01-13]# echo $RANDOM|md5sum

1bf202838933b899ef17e663a03d5f0b -

法2：通过openssl产生随机数

[root@db01 2016-01-13]# openssl rand -base64 65

QY02Nnaoth/5GUul2TQOtwHlT8QPdNkerIJ5MlyWmO18r4GLG2AwBfLGOb5kUrCo

SJVIfTP9Y53MgKBV7a2jPnQ=

法3：通过时间获得随机数（date）

[root@db01 2016-01-13]# date +%s%N

1452679516643235010

法4：使用urandom

[root@db01 2016-01-13]# head /dev/urandom|cksum

2279738077 2390

法5：UUID

[root@db01 2016-01-13]# cat /proc/sys/kernel/random/uuid

c49eef6c-d35e-4d3a-a82d-2d832bbe0342

法6：expect（使用前，需安装expect）

[root@db01 2016-01-13]# yum install -y expect

[root@db01 2016-01-13]# mkpasswd -l 8

vkRXs18<

使用for循环测试随机数的唯一性

[root@db01 2016-01-13]# for n in `seq 20`;do date +%s%N|md5sum|cut -c 1-9;done|sort|uniq -c|sort -rn -k1

[root@db01 2016-01-13]# for n in `seq 20`;do echo $RANDOM|md5sum|cut -c 1-9;done|sort|uniq -c|sort -rn -k1

## 1.9 break continue exit return

### 1.9.1 break continue exit对比

break、continue、exit一般用于循环结构中控制循环（for，while，if）的走向。

|  |  |
| --- | --- |
| **命令** | **说明** |
| break n | n表示跳出循环的层数，如果省略n表示**跳出整个循环**。 |
| continue n | N表示退到第n层循环，如果省略n标识跳过本次循环，忽略本次循环的剩余代码，进入循环的下一次循环。 |
| exit n | 退出当前shell程序，n为返回值。n也可以忽略，再下一个shell里通过 $?接收这个n的值。 |
| return n | 用于在函数里，作为函数的返回值，用于判断函数执行是否正确。 |

### 1.9.2 break、continue、exit范例

下面举几个break、continue、exit的例子

范例1：break跳出整个循环，执行循环下面的其他程序

测试脚本：

[root@db01 2016-01-13]# cat break.sh

#!/bin/sh

for((i=0;i<=5;i++))

do

if [ $i -eq 3 ];then

#continue;

#break;

exit

fi

echo $i

done

echo "ok"

当使用break时，返回结果为：

[root@db01 2016-01-13]# sh break.sh

0

1

2

ok

提示：可以看到i等于3及以后的循环没有执行，但循环外的echo执行了。

当使用continue时，返回结果为：

[root@db01 2016-01-13]# sh break.sh

0

1

2

4

5

ok

提示：可以看到只有i等于3这层循环没有执行，其它循环全部执行了，循环外的echo也执行了。

当使用exit是，返回结果为：

[root@db01 2016-01-13]# sh break.sh

0

1

2

### 1.9.3 生产范例

范例（生产场景）：开发shell脚本实现给服务器临时配置多个别名IP，并可以随时撤销配置的所有IP。IP地址为：10.0.2.1-10.0.2.16，其中10.0.2.10不能配置。

配置IP命令（ifconfig/IP）提示：

ifconfig eth0:0 10.0.2.10/24 up

ifconfig eth0:0 10.0.2.10/24 down

ip addr add 10.0.2.1/24 dev eth0:0

ip addr del 10.0.2.1/24 dev eth0:0

实战开始：

[root@db01 2016-01-13]# cat ip.sh

#!/bin/sh

USAGE(){

if [ $# -ne 1 ];then

echo "$0 {start|stop}"

fi

}

start(){

for i in `seq 16`

do

if [ $i -eq 10 ];then

continue

fi

ip addr add 10.0.2.$i/24 dev eth0 label eth0:$i

done

}

stop(){

for i in `seq 16`

do

if [ $i -eq 10 ];then

continue

fi

ip addr del 10.0.2.$i/24 dev eth0 label eth0:$i

done

}

main(){

case "$1" in

start)

start

;;

stop)

stop

;;

\*)

USAGE

esac

}

main $\*

## 1.10 shell数组

### 1.10.1 数组介绍

平时定义a=1;b=2;c=3，变量如果多了，再一个一个定义很费劲，并且去变量的也费劲。

**简单的说，数组就是各种数据类型的元素按一定顺序排列的集合。**

数组就是把有限个元素变量或数据用一个名字命名，然后用编号区分它们的变量的集合。这个名字称为数组名，编号成为数组下标。组成数组的各个变量成为数组的分量，也称为数组的元素，有时也称为下标变量。

如果有过用其他语言编程的经历，那么想必会熟悉数组的概念。由于有了数组可以用相同名字引用一系列变量，并用数字（索引）来识别它们。在许多场合。使用数组可以缩短和简化程序开发，因为可以利用索引值设计一个循环，高效处理多种情况。

**数组的下标是从0开始**

### 1.10.2 数组定义与增删改查

方法1：array=（value1 value2 value3...）

1） 数组定义

[root@db01 2016-01-13]# array=(1 2 3) ##一对括号表示是数组，数组元素用“空格”符号分割开。

2） 获取数组的长度

[root@db01 2016-01-13]# echo ${#array[@]} 用${#数组名[@或\*]}可以得到数组长度

3

[root@db01 2016-01-13]# echo ${#array[\*]}

3

3） 打印数组元素

[root@db01 2016-01-13]# echo ${array[0]} ##打印数组元素用${数组名[下标]} 下标是从0开始

1

[root@db01 2016-01-13]# echo ${array[1]}

2

[root@db01 2016-01-13]# echo ${array[2]}

3

[root@db01 2016-01-13]# echo ${array[\*]} ##下标是：\*或者@，得到整个数组内容。

1 2 3

[root@db01 2016-01-13]# echo ${array[@]}

1 2 3

使用for循环打印数组的例子：

法1：

[root@db01 2016-01-13]# cat shuzu.sh

#!/bin/sh

array=(1 2 3)

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

do

echo ${array[i]}

done

法2：

[root@db01 2016-01-13]# cat arr.sh

#!/bin/sh

array=(1 2 3)

for i in ${array[@]}

do

echo $i

done

4） 数组赋值

直接通过数组名[下标]就可以对其进行引用赋值，如果下标不存在，自动添加新一个数组元素，如果\*覆盖原来的值。

[root@db01 2016-01-13]# echo ${array[@]}

1 2 3

[root@db01 2016-01-13]# array[3]=4 ##增加数组元素

[root@db01 2016-01-13]# echo ${array[@]}

1 2 3 4

[root@db01 2016-01-13]# echo ${array[@]} ##修改数组元素

1 2 3 4

[root@db01 2016-01-13]# array[0]=oldboy

[root@db01 2016-01-13]# echo ${array[@]}

oldboy 2 3 4

5） 数组删除

直接通过：unset数组[下标]可以清除相应的元素，不带下标，清除整个数据。

[root@db01 2016-01-13]# echo ${array[@]}

2 3

[root@db01 2016-01-13]# unset array ##删除整个数组

[root@db01 2016-01-13]# echo ${array[@]}

[root@db01 2016-01-13]# echo ${array[@]}

oldboy 2 3 4

[root@db01 2016-01-13]# unset array[3] ##删除某个数组元素

[root@db01 2016-01-13]# echo ${array[@]}

oldboy 2 3

[root@db01 2016-01-13]# unset array[0]

[root@db01 2016-01-13]# echo ${array[@]}

2 3

6） 数组内容的截取和替换

截取：

[root@db01 2016-01-13]# array=(1 2 3 4 5)

[root@db01 2016-01-13]# echo ${array[\*]}

1 2 3 4 5

[root@db01 2016-01-13]# echo ${array[\*]:1:3} ##截取1号到3号数组元素

2 3 4

[root@db01 2016-01-13]# echo ${array[\*]:3:2}

4 5

替换：

[root@db01 2016-01-13]# echo ${array[\*]}

1 2 3 4 5

[root@db01 2016-01-13]# echo ${array[\*]/5/6} ##把数组中的5替换成6，临时生效，元数组未被修改。

1 2 3 4 6

删除：

[root@db01 2016-01-13]# array1=(one two three four five)

[root@db01 2016-01-13]# echo ${array1[@]}

one two three four five

[root@db01 2016-01-13]# echo ${array1[@]#o} ##左边开始最短的匹配

ne two three four five

[root@db01 2016-01-13]# echo ${array1[@]#fo} ##左边开始最短的匹配

one two three ur five

[root@db01 2016-01-13]# echo ${array1[@]%t\*e}

one two four five

[root@db01 2016-01-13]# echo ${array1[@]%%t\*e}

one two four five

提示：数组也是变量，因此也适合于变量的子串处理的功能应用。

方法2：array=([1]=one[2]=two[3]=three) ##key-value键值对

[root@db01 2016-01-13]# array=([1]=one [2]=two [3]=three)

[root@db01 2016-01-13]# echo ${array[@]}

one two three

[root@db01 2016-01-13]# echo ${#array[@]}

3

方法3：array[0]=a array[1]=b array[2]=c

[root@db01 2016-01-13]# array[0]=a

[root@db01 2016-01-13]# array[1]=b

[root@db01 2016-01-13]# array[2]=c

[root@db01 2016-01-13]# array[3]=d

[root@db01 2016-01-13]# echo ${#array[@]}

4

[root@db01 2016-01-13]# echo ${array[@]}

a b c d

[root@db01 2016-01-13]# echo ${array[0]}

a

方法4：declare –a array

方法5：array=($(ls))

[root@db01 2016-01-13]# array=($(ls))

[root@db01 2016-01-13]# echo ${array[@]}

arr.sh break.sh gaiming.sh ip.sh jia.sh shu.sh shuzu.sh test.sh touch.sh user1.sh user.sh

### 1.10.3 数组实践实战例子

范例1：通过列举法打印数组元素

列举元素写法：

array=(red green blue yellow magenta)

array=(

oldboy

zhangyue

zhangyang

)

实战开始

法1：

[root@db01 2016-01-13]# cat shu.sh

#!/bin/sh

array=(

oldboy

zhangyue

zhangyang

)

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

do

echo "this is num $i,then content is ${array[$i]}"

done

echo ----------------------------

echo "array len:${#array[\*]}"

法2：

[root@db01 2016-01-13]# cat array.sh

#!/bin/sh

array=($(ls))

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

do

echo ${array[i]}

done

echo =====================

for i in ${array[\*]}

do

echo $i

done

Linux有关shell数组的重要知识小结：

1、定义：

静态数组array=(1 2 3)

动态数组array=($(ls))

2、打印：

${array[@]}或${array[\*]} 打印所有元素。

${#array[@]}或${#array[\*]} 打印数组长度。

${array[i]} 打印单个元素，i是数组下标。

3、循环打印

[root@db01 2016-01-13]# cat hehe.sh

#!/bin/sh

arr=(

10.0.0.11

10.0.0.22

10.0.0.33

)

for ((i=0;i<${#arr[\*]};i++))

do

echo "${arr[$i]}"

done

echo ===================

for n in ${arr[\*]}

do

echo "$n"

done

[root@db01 2016-01-13]# sh hehe.sh

10.0.0.11

10.0.0.22

10.0.0.33

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

10.0.0.11

10.0.0.22

10.0.0.33

企业面试题：批量检查多个网站地址是否正常

要求：

1、shell数组方法实现，检测策略尽量模拟用户访问。

2、每10秒做一次所有的检测，无法访问的输入报警。

3、待检测的地址如下：

http://www.etiantian.org

http://www.taobao.com

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

http://10.0.0.7

分步实现：

1、把URL定义成数组，然后while打印出来。

[root@db01 2016-01-13]# cat check\_url.ori.sh

#!/bin/sh

[ -f /etc/init.d/functions ]&& . /etc/init.d/functions

url\_list=(

http://www.etiantian.org

http://www.taobao.com

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

http://10.0.0.7

http://www.suibddddddian123.com

)

check\_url(){

[ $# -ne 1 ] && exit 1

curl -o /dev/null -s --connect-timeout 5 -w "%{http\_code}" $1

}

main(){

for ((i=0;i<${#url\_list[\*]};i++))

do

code=`check\_url ${url\_list[i]}`

if [ "$code" != "200" -a "$code" != "301" ];then

action "${url\_list[i]}" /bin/false

else

action "${url\_list[i]}" /bin/true

fi

done

}

main

**学习完shell之后，需要掌握的技能：**

1、各类监控脚本，文件、内存、磁盘、端口，URL监控报警。  
2、如何监控网站目录文件是否被篡改，以及站点目录批量被篡改后如何恢复。  
3、如何开发各类服务rsync、nginx、mysql等的启动及停止专业脚本（使用chkconfig管理）。  
4、如何开发MySQL主从复制监控报警以及自动处理不复制的脚本。  
5、一键配置mysql多实例、一键配置mysql主从，N多一键部署脚本  
6、监控http/mysql/rsync/nfs/memcached等服务是否异常的生产脚本。  
7、一键软件安装及优化，lanmp,linux一键优化，一键数据库安装，优化，配置主从。  
8、MySQL多实例启动脚本，分库、分表自动备份脚本。  
9、根据网络连接数以及根据web日志PV封IP的脚本。  
10、监控网站的pv以及流量，并且对流量信息进行统计。  
11、检查web服务器多个URL地址是否异常，要可以批量及通用。  
12、系统的基础优化一键优化的脚本。  
13、清理系统垃圾及文件（过期备份，clientmquene目录等）脚本。  
14、tcp连接状态统计。  
15、批量创建用户并设置随机8位密码。  
16、批量获取服务器信息、批量分发文件。