Unix Shell脚本编程知识点总结及范例

时间 2016-01-08 09:14:51 异步社区

原文 http://www.epubit.com.cn/article/363

主题 Shell

脚本编程语言与编译型语言：

```
脚本编程语言：(Bash)
    脚本编程语言通常是解释型(interpreted),主要由解释器(interpreter)读入程序代码，并将其转换成内部的形式
    优点：
        能够轻易处理文件与目录之类的对象。
    缺点：
        运行效率通常不如编译型语言
编译型语言：(C、C++、Java、Fortran、Ada、Pascal)
    编译型语言多半运作于底层，所处理的是字节、整数、浮点数或其它及其机器层经的对象。
```

SHELL脚本的基本语法格式：

```
脚本必须以#!开头：(# cat /etc/shells)
例如#!/bin/bash(解释器)
# 其中间可以添加一些注释信息，例如脚本的使用方法、脚本的功能、创建日期、作者等相关信息
然后赋予脚本具有执行权限，# chmod +x scripts.sh
执行则使用./scripts.sh ##也可以将此脚本的路径添加到PATH变量中，以后直接使用脚本名称直接运行。
脚本的测试工具bash：
-n：检查脚本是否有语法错误，有则显示错误信息，否则无信息(没有消息才是最好的消息)
```

-x：检查脚本在执行中的详细过程(排错时，经常会用到)
exit：退出脚本(其数值为0-255)
如果脚本没有明确定义退出码，那么在执行脚本结束前的退出码为此脚本的退出码。
echo $? ##查看上一个命令执行结果所显示的状态码

SHELL脚本的逻辑关系总结：

逻辑与：符号为&&：
如果其中一个为假，则结果一定为假
如果第一个条件结果为假，则第二个条件不用再判断，最终结果已显示
如果第一个条件结果为真，则第二个条件必须判断
范例：
useradd redhat && echo "redhat" | passwd --stdin redhat 解说：如果useradd redhat执行成功，则

逻辑或||：
如果其中一个条件结果为真，则结果一定为真，不用检查后面的语句
如果其中一个条件结果为假，则检查下一个条件语句
范例：
id redhat || useradd redhat
解说：如果redhat用户存在，就显示redhat用户相关信息，否则添加此账户。
逻辑与和逻辑或联合使用范例：
id redhat && echo “redhat already existing“ || useradd redhat
解说：如果redhat用户存在，则显示redhat用户已存在，否则添加此账户。

条件判断语句总结：

单分支if语句双分支if语句 多分支if语句case选择语句

```
if 判断条件 ;then    if 判断条件;then     if 判断条件;thencase $1 in
  statement            statement           statement                  string)
  ......               ........            .........                      statement;;
fi                    else                elif 判断条件;then             string2)
                        statement           statement                      statement;;
                        .....               .........                  ......)
                    fi                  elif 判断条件;then                 statement;;
                                                statement              esac
                                                .........
else
                                                statement
                                            fi
```

范例：

```
#!/bin/bash
#

if [ $1 == '--add' ]; then
  for I in `echo $2 | sed 's/,/ /g'`; do
    if id $I &> /dev/null; then
      echo "$I exists."
    else
      useradd $I
      echo $I | passwd --stdin $I &> /dev/null
      echo "add $I finished."
    fi
  done
elif [ $1 == '--del' ];then
  for I in `echo $2 | sed 's/,/ /g'`; do
    if id $I &> /dev/null; then
      userdel -r $I
      echo "Delete $I finished."
    else
      echo "$I NOT exist."
    fi
  done
elif [ $1 == '--help' ]; then
  echo "Usage: adminuser2.sh --add USER1,USER2,... | --del USER1,USER2,...| --help"
else
  echo "Unknown options."
fi
```

脚本分析：
主要功能：传递一个不同的参数，来完成用户的创建、添加密码、删除用户。
详细说明：
当我们传递--add参数给此脚本时，此脚本为完成指定用户的添加，如果添加的用户存在，则提示用户以存在，否则添加指
当我们传递--del参数给此脚本时，此脚本会删除我们指定的用户，如果存在则删除此用户，否则提示用户不存在。
当我们传递--help参数给此脚本时，此脚本会给我们现实脚本的使用方法。
当我们传递其它参数时，会提示无法识别的选项。

POSIX的结束状态总结：

```
0：       ##命令成功所显示的状态
>0:      ##在重定向或单词展开期间(~、变量、命令、算术展开及单词切割)失败
1-125    ##命令不成功所显示的状态。
126      ##命令找到了，但文件无法执行所显示的状态
127      ##命令找不到，所显示的状态
>128     ##命令因收到信号而死亡
```

替换运算符总结(变量的赋值)：

```
${varname:-word}
   如果varname存在且非null，则返回其值;否则，返回word;
   用途：如果变量未定义，则返回默认值
   范例：如果count未定义，则${count:-0}的值为0
${varname:=word}
   如果varname存在且非null，则返回其值;否则，将varname设置为word，并返回其值;
   用途：如果变量未定义，则设置变量为默认值
   范烈：如果count未定义，则${count:=0}的值为0
${varname:+word}
   如果varname存在且非null，则返回word;否则，返回null；
   用途：为测试变量的存在
   范例：如果count已定义，则${count:+1}的值为1
```

```
${varname:?message}
   如果varname存在且非null，则返回其值;否则显示varname:message,并退出当前命令或脚本;
   用途：为了捕捉由于变量未定义所导致的错误。
   范例：如果count未定义，${count:?"undefined!"}则显示count:undefined!
```

模式匹配运算符总结：

```
假设path变量的值为：/etc/sysconfig/network-scripts/ifcfg-eth0.text.bak
${variable#pattern}：
   如果模式匹配于变量值的开头处，则删除匹配的最短部分，并返回剩下的部分；
   范例：echo ${path#/*/}的值为etc/sysconfig/network-scripts/ifcfg-eth0.text.bak
${variable##pattern}
   如果模式匹配于变量值的开头处，则删除匹配的最长部分，并返回剩下的部分；
   范例：echo ${path##/*/}的值为ifcfg-eth0.text.bake
${variable%pattern}
   如果模式匹配于变量的结尾处，则删除匹配的最短部分，并返回剩下的部分;
   范例：echo ${path%.*}的值为/etc/sysconfig/network-scripts/ifcfg-eth0.text
${variable%%pattern}
   如果模式匹配于变量的结尾处，则删除匹配的最长部分，则返回剩余部分。
   范例：echo ${path%%.*}的值为/etc/sysconfig/network-scripts/ifcfg-eth0
${#variable}
   显示variable变量值的字符长度
```

Shell脚本常用的循环语句总结：

```
for循环                          while循环                  until循环
for 变量 in 列表 ;do    while condition(条件);do      until condition ;do
   command...               statements                      statements
done                     done                            done

while循环：只要condition满足条件，while会循环
until循环：只要condition不满足条件，until会循环
```

test命令

1. test命令可以处理shell脚本中的各类工作，它产生的不是一般输出，而是可使用退出状态，test接受各种不同的参数，可控制它要执行哪一种测试
2. 语法：
3. test [expression]
4. test [[expression]]
5. 用途：
6. 为了测试shell脚本里的条件，通过退出状态返回其结果。
7. 行为模式：
8. test用来测试文件的属性、比较字符串、比较数字

9. 主要选项与表达式：
10. string　　string不是 null
11. -b file　　file是块设备文件(-b)
12. -c file　　file是字符设备文件(-c)
13. -d file　　file为目录(-d)
14. -e file　　file是否存在
15. -f file　　file是一般文件(-)
16. -g file　　file有设置它的setgid位
17. -h file　　file是一个符号链接
18. -r file　　file是可读的
19. -s file　　file是socket
20. -w file　　file是可写的
21. -x file　　file是可执行的，或file是可被查找的目录
22. s1 = s2　　s1与s2字符串相同
23. s1 != s2　　s1与s2字符串不相同
24. n1 -eq n2　整数n1与n2相等
25. n1 -ne n2　整数n1与n2不相等
26. n1 -lt n2　整数n1小于n2
27. n1 -gt n2　整数n1大于n2
28. n1 -le n2　整数n1小于或等于n2
29. n1 -ge n2　整数n1大于或等于n2
30. -n string　string是非 null
31. -z string　string为 nul l特殊参数变量
32. 在bash shell中有些特殊变量，它们会记录命令行参数的个数。例如$#

33. 你可以只数一下命令行中输入了多少个参数，而不同测试每个参数。bash为此提供了一个特殊的变量，就是上面所提到的$#
34. $#的说明
    1. $#特殊变量含有脚本运行时就有的命令行参数的个数。你可以在脚本中任何地方来调用这个特殊变量来调用$#来计算参数的个数
35. 范例:
    1. vim Count.sh

       #!/bin/bash

       #Script Name: Count.sh

       # Count Parameters number

       echo There were $# parameters.

       chmod +x Count.sh

       ./Count.sh 1 2 3

       There were 3 parameters.

下面来说下${!#}的作用？

既然$#变量含有参数的总数量，那么${!#}可以调用最后一个参数的变量名称。

范例：

vim Count-1.sh

#!/bin/bash

#Script Name: Count-1.sh

Print last parameter

params=$#

echo "The last parameter is "$params"

echo "The last parameter is "${!#}"

:wq

chmod +x Count-1.sh

./Count-1.sh 1 2 3

The last parameter is 3

The last parameter is 3 原文出自 http://guodayong.blog.51cto.com/263451/1188606

分享

收藏 纠错

推荐文章

- 1. MIT 6.828 JOS学习笔记6. Appendix 1: 实模式(real mode)与保护模式..
- 2. 嵌入式系统硬件初始化
- 3. Linux下控制文件的生成的C代码实现
- 4. python使用smtplib库和smtp.qq.com邮件服务器发送邮件
- 5. dumb-init：一个Docker容器初始化系统
- 6. Oz沙盒技术详解

我来评几句

请输入评论内容...

登录后评论

已发表评论数()

相关站点

异步社区

+ 订阅

热门文章

- 1. MIT 6.828 JOS学习笔记6. Appendix 1: 实模式(real mode)与保护模式(protected..
- 2. 嵌入式系统硬件初始化
- 3. Linux下控制文件的生成的C代码实现
- 4. python使用smtplib库和smtp.qq.com邮件服务器发送邮件
- 5. dumb-init：一个Docker容器初始化系统
- 6. Oz沙盒技术详解

分享本文

收藏到推刊 创建推刊

收 藏 取消

已收藏到推刊！

推刊名(必填) 请填写推刊名

推刊描述

描述不能大于100个字符!

权限设置： 公开 仅自己可见

创建 取消

×

文章纠错

邮箱地址

错误类型 正文不准确

补充信息

提交

网站相关

- 关于我们
- 移动应用
- 建议反馈

关注我们


推酷网


tuicool2012

友情链接

人人都是产品经理 魔部网 PM256 品途网 移动信息化 行晓网 Code4App 智城外包网 LAMP人 安卓航班网 虎嗅 缘创派 IT耳朵 艾瑞网 创媒工场 雷锋网 经理人分享 市场部网 砍柴网 CocoaChina 北风网 云智慧 我赢职场 大数据时代 奇笛网 咕噜网 红联linux Win10之家 鸟哥笔记 爱游戏 投资潮 31会议网 极光推送 Teambition Cocos引擎中文官网 硅谷网 leangoo 更多链接>>