1. set –e //这行代码之后的任何代码，如果返回一个非0的值，那么整个脚本立即退出，官方的说明是为了防止错误出现滚雪球的现象
2. set -o pipefail //告诉 bash 返回从右到左第一个以非0状态退出的管道命令的返回值，如果所有命令都成功执行时才返回0
3. 格式: echo "/033[字背景颜色;字体颜色m字符串/033[控制码"

如果单纯显示字体颜色可以固定控制码位0m。

格式: echo "/033[字背景颜色;字体颜色m字符串/033[0m"

字背景颜色范围:40 - 49

40:黑 41:深红 42:绿 43:黄色 44:蓝色 45:紫色 46:深绿 47:白色

字颜色:30 - 39

30:黑 31:红 32:绿 33:黄 34:蓝色 35:紫色 36:深绿 37:白色

ANSI控制码

\33[0m 关闭所有属性 \33[01m 设置高亮度

\33[04m 下划线 \33[05m 闪烁

\33[07m 反显 \33[08m 消隐

\33[30m -- \33[37m 设置前景色 \33[40m -- \33[47m 设置背景色

\33[nA 光标上移n行 \33[nB 光标下移n行

\33[nC 光标右移n行 \33[nD 光标左移n行

\33[y;xH设置光标位置 \33[2J 清屏

\33[K 清除从光标到行尾的内容 \33[s 保存光标位置

\33[u 恢复光标位置 \33[?25l 隐藏光标

\33[?25h 显示光标

如：echo -e "\033[34m\033[05m 请确认您的操作,输入 [Y/N]\033[0m"

1. 条件表达式

if ....; then

....

elif ....; then

....

Else

....

fi

文件表达式：

if [ -f file ] 如果文件存在

if [ -d ... ] 如果目录存在

if [ -h file ] 如果 FILE 存在且是一个符号连接则为真。

if [ -s file ] 如果文件存在且非空

if [ -r file ] 如果文件存在且可读

if [ -w file ] 如果文件存在且可写

if [ -x file ] 如果文件存在且可执行

if [ -e file ] 文件或目录名存在

整数变量表达式：

if [ int1 -eq int2 ] 如果int1等于int2

if [ int1 -ne int2 ] 如果不等于

if [ int1 -ge int2 ] 如果>=

if [ int1 -gt int2 ] 如果>

if [ int1 -le int2 ] 如果<=

if [ int1 -lt int2 ] 如果<

字符串变量表达式：

If [ $a = $b ] 如果string1等于string2

字符串允许使用赋值号做等号

if [ $string1 != $string2 ] 如果string1不等于string2

if [ -n $string ] 如果string 非空(非0），返回0(true)

if [ -z $string ] 如果string 为空

if [ $sting ] 如果string 非空，返回0 (和-n类似)

如：

If空[空-z "$BARCELONA\_ROOT"空];then //空格特别重要,黄色代表空格

echo "Errot:BARCELONA\_ROOT is not defined!"

exit 1

fi

if [ ! -f "$1" ]; then

printf "Error: file does not exist.\n"

exit 1

fi

**2）最常用的简化if语句：**

// && 如果是“前面”，则“后面”

[ -f /var/run/dhcpd.pid ] && rm /var/run/dhcpd.pid //检查文件是否存在，如果存在就删掉

// || 如果不是“前面”，则后面

[ -f /usr/sbin/dhcpd ] || exit 0 //检验文件是否存在，如果存在就退出

3）以 test 条件表达式 作为if条件

if test $num -eq 0 等价于 if [ $num –eq 0 ]

///test 表达式,没有 [ ]

if test $num -eq 0

then

echo "try again"

else

echo "good"

fi

1. 变量说明:

$$  
Shell本身的PID（ProcessID）  
$!  
Shell最后运行的后台Process的PID  
$?  
最后运行的命令的结束代码（返回值）  
$-  
使用Set命令设定的Flag一览  
$\*  
所有参数列表。如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。  
$@  
所有参数列表。如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。  
$#  
添加到Shell的参数个数  
$0  
Shell本身的文件名  
$1～$n  
添加到Shell的各参数值。$1是第1参数、$2是第2参数…。

1. 整数比较

-eq 等于,如:if ["$a" -eq "$b" ]

-ne 不等于,如:if ["$a" -ne "$b" ]

-gt 大于,如:if ["$a" -gt "$b" ]

-ge 大于等于,如:if ["$a" -ge "$b" ]

-lt 小于,如:if ["$a" -lt "$b" ]

-le 小于等于,如:if ["$a" -le "$b" ]

< 小于(需要双括号),如:(("$a" < "$b"))

<= 小于等于(需要双括号),如:(("$a" <= "$b"))

> 大于(需要双括号),如:(("$a" > "$b"))

>= 大于等于(需要双括号),如:(("$a" >= "$b"))

1. $(( ))的用途

用来作整数运算。在 bash 中，$(( ))的整数运算符号大致有这些：+ - \* / 加、减、乘、%

$ echo $(((a+b)/c))

1. 字符串操作

1)${ }的一些特异功能：

假设我们定义了一个变量为：

file=/dir1/dir2/dir3/my.file.txt

我们可以用${ }分别替换获得不同的值：

${file#\*/}：拿掉第一条/及其左边的字符串：dir1/dir2/dir3/my.file.txt

${file##\*/}：拿掉最后一条/及其左边的字符串：my.file.txt

${file#\*.}：拿掉第一个. 及其左边的字符串：file.txt

${file##\*.}：拿掉最后一个. 及其左边的字符串：txt

${file%/\*}：拿掉最后条/及其右边的字符串：/dir1/dir2/dir3

${file%%/\*}：拿掉第一条/及其右边的字符串：(空值)

${file%.\*}：拿掉最后一个. 及其右边的字符串：/dir1/dir2/dir3/my.file

${file%%.\*}：拿掉第一个. 及其右边的字符串：/dir1/dir2/dir3/my

记忆的方法为：

#是去掉左边(在鉴盘上#在$之左边)

%是去掉右边(在鉴盘上%在$之右边)

单一符号是最小匹配﹔两个符号是最大匹配。

${file:0:5}：提取最左边的5个字节：/dir1

${file:5:5}：提取第5个字节右边的连续5个字节：/dir2

2) 从左边第几个字符开始以及字符的个数，用法为:start:start:len

${file:1:5} ==> dir1/

从左边第几个字符开始一直到结束，用法为:start

${file:5} ==> /dir2/dir3/my.file.txt

从右边第几个字符开始以及字符的个数，用法:0-start:len

${file:0-8:8} ==> file.txt

从右边第几个字符开始一直到结束，用法:0-start

${file:0-5} ==>e.txt

1. 字符串操作（长度，读取，替换）

test=” c:/windows/boot.ini”

1. ${#string} //$string的长度
2. ${string/substring/replacement}

//使用$replacement, 来代替第一个匹配的$substring

echo ${test/\//\\} ==> c:\windows/boot.ini

1. oth
2. cd -是返回到上一次的工作目录
3. shell读文件的方法

1）管道的方式：

cat /sites/linuxpig.com.txt |while read LINE

do

echo $LINE

done

2）利用重定向符<

while read LINE

do

echo $LINE

done < /sites/linuxpig.com.txt

3）for line in `cat file(待读取的文件)`

do

    echo $line

done

1. 数值计算

方法1：declare –i

#declare -i sum=100+300+50

# echo $sum ==>#结果为：450

方法2，var=$ (( expression ))

foo=1

foo=$(($foo+1))

1. return=`[linux命令](https://www.baidu.com/s?wd=linux%E5%91%BD%E4%BB%A4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4PH9WuH-BnvDsmhf1PWmz0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtzPW0Yn1RsPWf)` 会将[linux命令](https://www.baidu.com/s?wd=linux%E5%91%BD%E4%BB%A4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4PH9WuH-BnvDsmhf1PWmz0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtzPW0Yn1RsPWf)的结果赋值给return  
    return=$([linux命令](https://www.baidu.com/s?wd=linux%E5%91%BD%E4%BB%A4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4PH9WuH-BnvDsmhf1PWmz0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtzPW0Yn1RsPWf))也是将linux命令的结果赋值给return的一种方式
2. Linux系统脚本的三种执行方式

1)、source命令用法：   source FileName

作用:在当前bash环境下读取并执行FileName中的命令。该filename文件可以无"执行权限"

注：该命令通常用命令“.”来替代。

如：source bash\_profile   . bash\_profile两者等效。

2)、sh和bash命令用法： sh FileName bash FileName

3)、./的命令用法：   ./FileName

作用:打开一个子shell来读取并执行FileName中命令。 该文件必须有可执行的权限。

source filename 与 sh filename 及./filename执行脚本的区别在那里呢？

1)、当shell脚本具有可执行权限时，用sh filename与./filename执行脚本是没有区别得。./filename是因为当前目录没有在PATH中，所有"."是用来表示当前目录的。

2)、sh filename 重新建立一个子shell，在子shell中执行脚本里面的语句，该子shell继承父shell的环境变量，但子shell新建的、改变的变量不会被带回父shell，除非使用export。

3)、source filename：这个命令其实只是简单地读取脚本里面的语句依次在当前shell里面执行，没有建立新的子shell。那么脚本里面所有新建、改变变量的语句都会保存在当前shell里面。

1. 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1，原来的$5现在变成$2等等，原来的$1、$2、$3丢弃，$0不移动。不带参数的shift命令相当于shift 1。

若用户要求 Shell 在不知道位置变量个数的情况下，还能逐个的把参数一一处理，也就是在 $1 后为 $2,在 $2 后面为 $3 等。在 shift 命令执行前变量 $1 的值在 shift 命令执行后就不可用了。

until [ $# -eq 0 ]

do

echo "第一个参数为: $1 参数个数为: $#"

shift

done

执行以上程序x：$./shift.sh 1 2 3 4

第一个参数为: 1 参数个数为: 4

第一个参数为: 2 参数个数为: 3

第一个参数为: 3 参数个数为: 2

第一个参数为: 4 参数个数为: 1

1. EOF … End of File

EOD … End of Document

EOT … End of Text

EOS … End of String

1. for循环

1）数字段形式

for i in {1..10}

do

echo $i

done

2）详细列出（字符且项数不多）

for File in 1 2 3 4 5

do

echo $File

done

3） 对存在的文件进行循环

for shname in `ls \*.sh`  
do   
          name=`echo "$shname" | awk -F. '{print $1}'`             
          echo $name  
done

4）查找循环（ls数据量太大的时候也可以用这种方法）  
for shname in `find . -type f -name "\*.sh"`  
do   
          name=`echo "$shname" | awk -F/ '{print $2}'`            
          echo $name  
done

5）((语法循环--有点像C语法，但记得双括号  
for((i=1;i<100;i++))  
do  
    if((i%3==0))  
    then  
        echo $i  
        continue  
    fi  
done

6）seq形式 起始从1开始  
for i in `seq 100`  
do  
    if((i%3==0))  
    then  
        echo $i  
        continue  
    fi  
done

1. while循环

1）while循环注意为方括号[],且注意空格  
min=1  
max=100  
while [ $min -le $max ]  
do  
    echo $min  
    min=`expr $min + 1`  
done

2）双括号形式，内部结构有点像C的语法，注意赋值：i=$(($i+1))  
i=1  
while(($i<100))  
do  
    if(($i%4==0))  
    then  
        echo $i  
    fi  
    i=$(($i+1))  
done

3）从配置文件读取，并可以控制进程数量  
MAX\_RUN\_NUM=8  
cat cfg/res\_card\_partition.cfg |grep -v '^$'|grep -v "#" | grep -v grep |while read partition   
do

       nohup sh inv\_res\_card\_process.sh $partition >log/resCard$partition.log 2>&1 &                 
        while [ 1 -eq 1 ]  
        do  
               psNum=`ps -ef | grep "inv\_res\_card\_process" | grep -v "grep" | wc -l`  
               if [ $psNum -ge $MAX\_RUN\_NUM ]  
                then  
                            sleep 5  
                else  
                           break  
                 fi                                          
         done                  
done

1. case in

//选择启动功能

 case $1 in   
         start | begin)   
           echo "start something"   
         ;;   
         stop | end)   
           echo "stop something"   
         ;;   
         \*)   
           echo "Ignorant"   
         ;;   
 esac

//判断系统

 SYSTEM=`uname -s`   
 case $SYSTEM in   
     Linux)   
         echo "My system is Linux"   
         echo "Do Linux stuff here..."   
     ;;   
     FreeBSD)   
         echo "My system is FreeBSD"   
         echo "Do FreeBSD stuff here..."   
     ;;   
     \*)   
         echo "Unknown system : $SYSTEM"   
         echo "I don't what to do..."   
     ;;   
 esac

1. Shell中调用、引用、包含另一个脚本文件的三种方法

脚本 first (测试示例1)

first

#!/bin/bash

echo 'your are in first file'

方法一:使用source

#!/bin/bash

echo 'your are in second file'

source first

方法二:使用.

#!/bin/bash

echo 'your are in second file'

. first

方法三:使用sh

#!/bin/bash

echo 'your are in second file'

sh first

1. 重定向

1. command > file 2>&1

这个命令中实现重定向的关键系统调用序列是：

open(file) == 3

dup2(3,1)

dup2(1,2)

2. command 2>&1 >file

这个命令中实现重定向的关键系统调用序列是：

dup2(1,2)

open(file) == 3

dup2(3,1)

3. 定向输出

cat >&2 <<EOF //输出至错误

test\_test\_test

EOF

4、sub shell重定向

return=$(systemctl status netatalk.service)

while read line; do

echo $line

done <<< "$return"

1. array数组

用括号来表示数组，数组元素用“空格”符号分割开。

1、length=${#array\_name[@]} 或者 length=${#array\_name[\*]} // 取得数组元素的个数

2、unset array[0] //删除第一个元素

3、直接通过 ${数组名[@或\*]:起始位置:长度} 切片原先数组，返回是字符串，中间用“空格”分开，因此如果加上”()”，将得到切片数组，上面例子：c 就是一个新数据

$ a=(1 2 3 4 5)

$ echo ${a[@]:0:3} ==> 1 2 3

$ echo ${a[@]:1:4} ==> 2 3 4 5

4、${数组名[@或\*]/查找字符/替换字符} 该操作不会改变原先数组内容

$ a=(1 2 3 4 5)

$ echo ${a[@]/3/100} ==> 1 2 100 4 5

$ echo ${a[@]} ==> 1 2 3 4 5

$ a=(${a[@]/3/100})

$ echo ${a[@]}  ==> 1 2 100 4 5

5、遍历数组：

filename=(`ls`)  
for var in ${filename[@]};do  
echo $var  
done

1. 多行注释

1、 : << !

语句1

语句2

!

2、: '

语句1

语句2

'

3、 if false; then

语句1

fi

1. 使用getopts处理shell中的输入参数

几个重要参数:

OPTIND：getopts使用OPTIND作为索引，来处理下一个需要处理的参数，记录当前的状态。

OPTARG：在上面的循环中**，a，b两个参数后面各有一个冒号，冒号表示该输入的参数后面还有一个参数值，当getopts发现冒号后，会处理用户输入的参数值，这个参数值被保存在OPTARG中。**

OPTSTRING：也就是上例中的 :a:b:cdefg，getopts需要处理的参数。注意，**最前面的冒号“：”用于指定getopts工作于silent mode，**在silent模式下，当用户输入的参数不满足OPTSTRING时，不会讲illegal option这样的错误信息打印出来，使代码看起来更加专业。如果想要工作在verbose模式下，可以去掉最前面的冒号

function create-apps() {  
    **local OPTIND**    cmd="rhc app create -p $OPENSHIFT\_PASSWD"  
    **while** getopts a:t:sn x  
    **do**  
        **case** $x **in**  
            a) cmd=$cmd" -a $OPTARG" ;;  
            t) cmd=$cmd" -t $OPTARG" ;;  
            s) cmd=$cmd" -s" ;;  
            n) cmd=$cmd" -n --no-dns" ;;  
            \?) **echo** Invalid Params ;;  
        **esac**  
    **done**  
    **echo** $cmd  
}

1. perl正则表达式

. 匹配除换行符以外的所有字符   
x? 匹配 0 次或一次 x 字符串   
x\* 匹配 0 次或多次 x 字符串，但匹配可能的最少次数   
x+ 匹配 1 次或多次 x 字符串，但匹配可能的最少次数   
.\* 匹配 0 次或一次的任何字符   
.+ 匹配 1 次或多次的任何字符   
{m} 匹配刚好是 m 个 的指定字符串   
{m,n} 匹配在 m个 以上 n个 以下 的指定字符串   
{m,} 匹配 m个 以上 的指定字符串   
[] 匹配符合 [] 内的字符   
[^] 匹配不符合 [] 内的字符   
[0-9] 匹配所有数字字符   
[a-z] 匹配所有小写字母字符   
[^0-9] 匹配所有非数字字符   
[^a-z] 匹配所有非小写字母字符   
^ 匹配字符开头的字符   
$ 匹配字符结尾的字符   
d 匹配一个数字的字符，和 [0-9] 语法一样   
d+ 匹配多个数字字符串，和 [0-9]+ 语法一样   
D 非数字，其他同 d   
D+ 非数字，其他同 d+   
w 英文字母或数字的字符串，和 [a-zA-Z0-9] 语法一样   
w+ 和 [a-zA-Z0-9]+ 语法一样   
W 非英文字母或数字的字符串，和 [^a-zA-Z0-9] 语法一样   
W+ 和 [^a-zA-Z0-9]+ 语法一样   
s 空格，和 [ f] 语法一样   
s+ 和 [ f]+ 一样   
S 非空格，和 [^ f] 语法一样   
S+ 和 [^ f]+ 语法一样

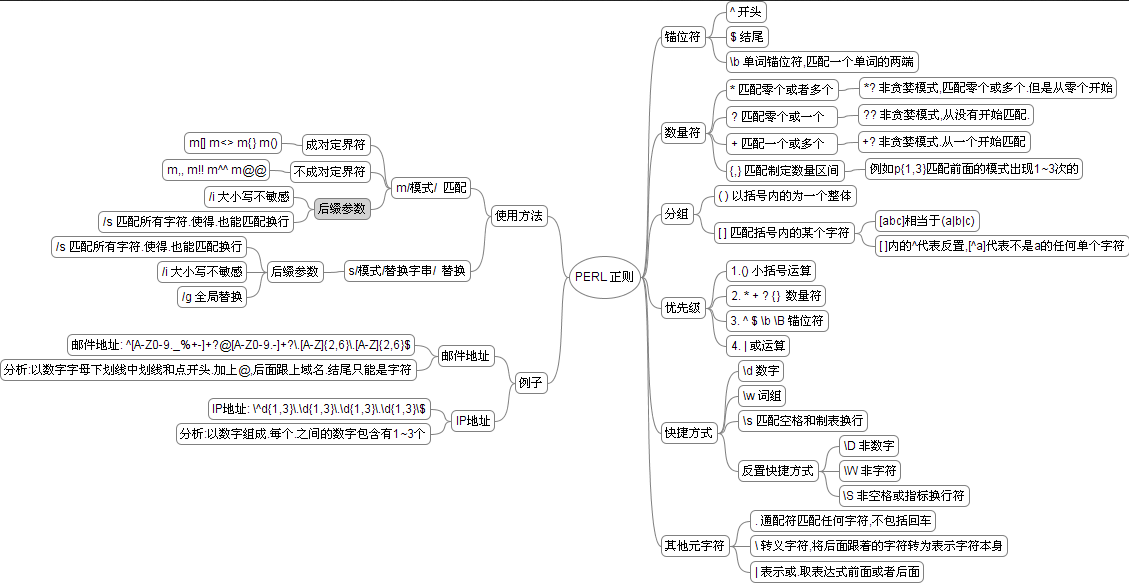
a|b|c 匹配符合a字符 或是b字符 或是c字符 的字符串

匹配：m// （还可以简写为 // ，略去 m）

/i不区分大小写   /s匹配任何字符    /x添加空格  /o 不重复编译正则表达示

替换：s///

 /g    进行全局替换           \U     大小写转换,例如s/(fred|barney)/\U$1/gi;  
\L    转换小写                   \E    影响到剩余的(替换的)字符串  
\l 和\u    写形式时(\l 和\u),只作用于下一个字符

转化：tr///

1. other