调试

set -x/+x 打开关闭调试跟踪

脚本参数拆分

getopt, getopts

```
while getopts ":abcde:fg" Option
# Initial declaration.
# a, b, c, d, e, f, and g are the options (flags) expected.
# The : after option 'e' shows it will have an argument passed with it.
do
  case $Option in
    a ) # Do something with variable 'a'.
    b ) # Do something with variable 'b'.
    ...
    e)  # Do something with 'e', and also with $OPTARG,
        # which is the associated argument passed with option 'e'.
    ...
    g ) # Do something with variable 'g'.
  esac
done
shift $(($OPTIND - 1))
# Move argument pointer to next.

# All this is not nearly as complicated as it looks <grin>.
```

符号:

#

; 多命令分割

;; case语句结束

. 等同source

"部分引用，在使用中需要注意:与'的区别，包含的特殊符号转义

' 完全引用，引号内容会原样保留。

例如'$a', 打印还是$a，不会解析

` 命令替换

: 空命令 ，退出码是0，即真。 其它用法较多，

域分割符/usr/local/bin:/bin:/usr

和重定向操作符（>）连用, 可以把一个文件的长度截短为零，文件的权限不变。如果文件不存在，则会创建一个新文件。

! 取反

? 测试操作符

$* $@ 分隔后的参数，@表示被字符串包含，未再次解析。

$# 参数个数

$?退出码值的变量,

$$ 进程id

() 命令组。由子shell内执行

{xxx,yyy,zzz,…}扩展支持

cp file22.{txt,backup}

6 # 拷贝"file22.txt"内容为"file22.backup"

{} 代码块，相当于匿名函数，

[] 测试

[[]]测试

| 管道

>| 强迫重定向，会强迫覆盖一个文件

~ $HOME

~+ `pwd`

~- 之前的工作目录， $OLDPWD

=~ 使用正则匹配

^ 行首 $行尾

空白用做函数的分隔符,分隔命令或变量

\ 转义符；多行命令连续

操作符'!'放在一个命令前面会导致调用Bash的历史机制。

```
1 true  # ture命令是内建的.
2 echo "exit status of \"true\" = $?"    # 0
3
4 ! true
5 echo "exit status of \"! true\" = $?"   # 1
6 # 注意逻辑非字符"!"需要一个空格.
7 #   !true   会导致一个"command not found"（命令没有发现）的错误。
8 #
9 # 操作符'!'放在一个命令前面会导致调用Bash的历史机制。
10
11 true
12 !true
13 # 这次没有错误，也没有反转结果.
```

14 # 它只是重复执行上次的命令(true).
15
16 # 多谢, Stéphane Chazelas and Kristopher Newsome.

文件测试

-e
文件存在
-a
文件存在
这个和-e的作用一样. 它是不赞成使用的，所以它的用处不大。
-f
文件是一个普通文件(不是一个目录或是一个设备文件)
-s
文件大小不为零
-d
文件是一个目录
-b
文件是一个块设备(软盘, 光驱, 等等.)
-c
文件是一个字符设备(键盘, 调制解调器, 声卡, 等等.)
-p
文件是一个管道
-h
文件是一个符号链接
-L
文件是一个符号链接
-S
文件是一个socket
-t
文件(描述符)与一个终端设备相关
这个测试选项可以用于检查脚本中是否标准输入 ([-t 0])或标准输出([-t 1])是一个终端.
-r
文件是否可读 (指运行这个测试命令的用户的读权限)
-w
文件是否可写 (指运行这个测试命令的用户的读权限)
-x
文件是否可执行 (指运行这个测试命令的用户的读权限)

-g
文件或目录的设置-组-ID(sgid)标记被设置
如果一个目录的sgid标志被设置，在这个目录下创建的文件都属于拥有此目录的用户组，而不必是创建文件的用户所属的组。这个特性对在一个工作组里的同享目录很有用处。
-u
文件的设置-用户-ID(suid)标志被设置
一个root用户拥有的二进制执行文件如果设置了设置-用户-ID位(suid)标志普通用户可以以root权限运行。[1] 这对需要存取系统硬件的执行程序（比如说pppd和cdrecord）很有用。如果没有设置suid位，则这些二进制执行程序不能由非root的普通用户调用。

```
-rwsr-xr-t    1 root       178236 Oct  2  2000 /usr/sbin/pppd
```

被设置了suid标志的文件在权限列中以s标志表示.
-k
粘住位设置
-O
你是文件拥有者
-G
你所在组和文件的group-id相同
-N
文件最后一次读后被修改
f1 -nt f2
文件f1比f2新
f1 -ot f2
文件f1比f2旧
f1 -ef f2
文件f1和f2 是相同文件的硬链接
!
"非" -- 反转上面所有测试的结果(如果没有给出条件则返回真).

-z
字符串为"null"，即是指字符串长度为零。
-n
字符串不为"null"，即长度不为零.

计算操作符
+

加
-
减
*
乘
/
除
**
求幂

```
1 # Bash在版本2.02引入了"**"求幂操作符.
2
3 let "z=5**3"
4 echo "z = $z"  # z = 125
```

%
求模（它返回整数整除一个数后的余数）

位操作符
<<
位左移（每移一位相当乘以2）
<<=
"位左移赋值"
let "var <<= 2" 结果使var的二进制值左移了二位（相当于乘以4）
>>
位右移（每移一位相当除以2）
>>=
"位右移赋值"（和<<=相反）
&
位与
&=
"位于赋值"
|
位或
|=
"位或赋值"
~
位反
!
位非
^

位或
^=
"位或赋值"

数字表示法

一个前缀为0的数字是八进制数。
一个前缀为0x的数字是十六进制数。
一个数用内嵌的#来求值则看成BASE#NUMBER(有范围和符号限制)(译者注:
BASE#NUMBER即：基数#数值
BASE值在2和64之间.
NUMBER必须使用在BASE范围内的符号,看下面的示例.

```
let "b64 = 64#@_"
echo "base-64 number = $b64"            # 4031
# 这个符号只能工作在ASCII码值为2-64的范围限制.
# 10个数字+26个小写字母+26个大写字母+ @ + _
```

内置变量

$LINENO
$SECONDS脚本已运行的秒数.
$TMOUT
如果$TMOUT环境变量被设为非零值时间值time，那么经过time这么长的时间后，shell提示符会超时．这将使此shell退出登录.

内置字符串操作

*建议awk

字符串长度
expr length $string

匹配字符串开头的子串的长度
expr match "$string" '$substring'　　#$substring 是一个正则表达式.
expr "$string" : '$substring'　　#$substring 是一个正则表达式.
索引
expr index $string $substring

子串提取
${string:position}
把$string中从第$postion个字符开始字符串提取出来.
如果$string是"*"或"@"，则表示从位置参数中提取第$postion后面的字符串。[1]
${string:position:length}

```
1 stringZ=abcABC123ABCabc
2 #       0123456789.....     以0开始计算.
6 echo ${stringZ:1}                       # bcABC123ABCabc
9 echo ${stringZ:7:3}                      # 23A
10                                  # 提取的子串长为3
16 echo ${stringZ:-4}                      # abcABC123ABCabc
17 # 默认是整个字符串，就相当于${parameter:-default}.
18 # 然而. . .
20 echo ${stringZ:(-4)}                    # Cabc
22 # 这样,它可以工作了.
23 # 圆括号或附加的空白字符可以转义$position参数.
```

子串移动
${string#substring}
从$string左边开始，剥去最短匹配$substring子串.
${string##substring}
从$string左边开始，剥去最长匹配$substring子串.

${string%substring}
从$string结尾开始，剥去最短匹配$substring子串。
${string%%substring}
从$string结尾开始，剥去最长匹配$substring子串。
子串替换
${string/substring/replacement}
用$replacement替换由$substring匹配的字符串。
${string//substring/replacement}
用$replacement替换所有匹配$substring的字符串。
${string/#substring/replacement}
如果$string字符串的最前端匹配$substring字符串，用$replacement替换$substring.
${string/%substring/replacement}
如果$string字符串的最后端匹配$substring字符串，用$replacement替换$substring.

变量操作

${parameter-default}, ${parameter:-default}

如果变量没有被设置，使用默认值。

```
1 echo ${username-`whoami`}
2 # 如果变量$username还没有被设置，则把命令`whoami`的结果赋给该变量.
```

${parameter-default}和${parameter:-default}几乎是相等的。它们之间的差别是：当一个参数已被声明，但是值是NULL的时候两者不同.

${parameter=default}, ${parameter:=default}

如果变量parameter没有设置，把它设置成默认值.

${parameter+alt_value}, ${parameter:+alt_value}

如果变量parameter设置，使用alt_value作为新值，否则使用空字符串。

除了引起的当变量被声明且值是空值时有些不同外，两种形式几乎相等。

${parameter?err_msg}, ${parameter:?err_msg}

如果变量parameter已经设置，则使用该值，否则打印err_msg错误信息。

${!varprefix*}, ${!varprefix@}

匹配所有前面声明过的变量，并且变量名以varprefix开头.

变量声明

declare或typeset内建命令(它们是完全相同的)可以用来限定变量的属性.

declare/typeset 选项

-r 只读

-i 整数

-a -f -x

变量引用

假设一个变量的值是第二个变量的名字. 这样要如何才能从第一个变量处重新获得第二个变量的值?

```
eval var1=\$$var2
echo $var1 #print  var2的内容
```

$RANDOM是Bash的一个返回伪随机 [1]整数(范围为0 - 32767)的内部函数

```
RANDOM=33; #种子
```

C风格
用((...))结构来使用C风格操作符来处理变量.

循环结构
for arg in [list]
do
  command(s)...
done
在[list]中的参数加上双引号是为了防止单词被不合理地分割.

如果在for循环的[list]中有通配符(*和?),那将会产生文件名扩展,也就是文件名扩展
在一个for循环中忽略in [list]部分的话,将会使循环操作$@

 for ((a=1; a <= LIMIT ; a++)) # 双圆括号, 并且"LIMIT"变量前边没有 “$”.c风格

while
这种结构在循环的开头判断条件是否满足,如果条件一直满足,那就一直循环下去(0为退出码[exit status])
while [condition]
do
  command...
done

until
这个结构在循环的顶部判断条件,并且如果条件一直为false那就一直循环下去.(与while相反).

until [condition-is-true]
do
  command...
done

break命令可以带一个参数.一个不带参数的break循环只能退出最内层的循环,而break N可以退出N层循环.
continue N将会把N层循环剩余的代码都去掉,但是循环的次数不变.

控制程序分支
case (in) / esac

```
case "$variable" in

?"$condition1" )
?command...
?;;

?"$condition2" )
?command...
?;;

esac
```

菜单

提示用户输入选择的内容(比如放在变量列表中).注意:select命令使用PS3提示符[默认为(#?)],

select结构是建立菜单的另一种工具,这种结构是从ksh中引入的.

```
select variable [in list]
do
?command...
?break
done
```