< 2015年4月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

公告

昵称：Frank Tan
园龄：5年3个月
粉丝：9
关注：0
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

C/C++(6)
Linux(3)
Linux Kernel
Python
Shell(4)
valgrind(1)
生活(1)

随笔档案

2012年4月 (1)
2011年2月 (1)
2011年1月 (3)
2010年12月 (4)
2010年3月 (3)
2010年1月 (1)

最新评论

1. Re:Bash Shell中命令行选项/参数处理
getopt 最牛的可以提取参数，被提取的参数可以随便放位置，getopts就不行，提取参数必须放在其他的

Bash Shell中命令行选项/参数处理

0.引言

写程序的时候经常要处理命令行参数，本文描述在Bash下的命令行处理方式。

选项与参数：

如下一个命令行：

```
./test.sh -f config.conf -v --prefix=/home
```

我们称-f为选项，它需要一个参数，即config.conf, -v 也是一个选项，但它不需要参数。

--prefix我们称之为一个长选项，即选项本身多于一个字符，它也需要一个参数，用等号连接，当然等号不是必须的，/home可以直接写在--prefix后面，即--prefix/home,更多的限制后面具体会讲到。
在bash中，可以用以下三种方式来处理命令行参数，每种方式都有自己的应用场景。

* 手工处理方式
* getopts
* getopt

下面我们依次讨论这三种处理方式。

1. 手工处理方式

在手工处理方式中，首先要知道几个变量，还是以上面的命令行为例：

* $0 ： ./test.sh,即命令本身，相当于C/C++中的argv[0]
* $1 ： -f,第一个参数.
* $2 ： config.conf
* $3, $4 ... ：类推。
* $# 参数的个数，不包括命令本身，上例中$#为4.
* $@ ：参数本身的列表，也不包括命令本身，如上例为 -f config.conf -v --prefix=/home
* $* ：和$@相同，但"$*" 和 "$@"(加引号)并不同，"$*"将所有的参数解释成一个字符串，而"$@"是一个参数数组。如下例所示：

参数前面才行

--linuxdog


```
#!/bin/bash

for arg in "$*"
do
    echo $arg
done

for arg in "$@"
do
    echo $arg
done

```

执行./test.sh -f config.conf -n 10 会打印：

-f config.conf -n 10 #这是"$*"的输出

-f #以下为$@的输出

config.conf

-n

10

所以，手工处理的方式即对这些变量的处理。因为手工处理高度依赖于你在命令行上所传参数的位置，所以一般都只用来处理较简单的参数。如

./test.sh 10

而很少使用./test -n 10这种带选项的方式。 典型用法为：

```
#!/bin/bash

if [ x$1 != x ]
then
    #...有参数
else
then
    #...没有参数
fi
```

为什么要使用 x$1 != x 这种方式来比较呢？想像一下这种方式比较：

if [-n $1] #$1不为空

但如果用户不传参数的时候，$1为空，这时 就会变成 [-n] ,所以需要加一个辅助字符串来进行比较。

手工处理方式能满足大多数的简单需求，配合shift使用也能构造出强大的功能，但在要处理复杂选项的时候建议用下面的两种方法。

2. getopts/getopt

处理命令行参数是一个相似而又复杂的事情，为此，C提供了getopt/getopt_long等函数，
C++的boost提供了Options库，在shell中，处理此事的是getopts和getopt.

getopts和getopt功能相似但又不完全相同，其中getopt是独立的可执行文件，而getopts是由Bash内置的。

先来看看参数传递的典型用法:

* ./test.sh -a -b -c ： 短选项，各选项不需参数
* ./test.sh -abc ： 短选项，和上一种方法的效果一样，只是将所有的选项写在一起。
* ./test.sh -a args -b -c ：短选项，其中-a需要参数，而-b -c不需参数。
* ./test.sh --a-long=args --b-long ：长选项

我们先来看getopts,它不支持长选项。

使用getopts非常简单：
代码

```
#test.sh

#!/bin/bash

while getopts "a:bc" arg #选项后面的冒号表示该选项需要参数
do
        case $arg in
             a)
                echo "a's arg:$OPTARG" #参数存在$OPTARG中
                ;;
             b)
                echo "b"
                ;;
             c)
```

```
                echo "c"
                ;;
            ?)  #当有不认识的选项的时候arg为?
                echo "unkonw argument"
        exit 1
        ;;
        esac
done
```

现在就可以使用：
./test.sh -a arg -b -c
或
./test.sh -a arg -bc
来加载了。
应该说绝大多数脚本使用该函数就可以了，如果需要支持长选项以及可选参数，那么就需要使用getopt.
下面是getopt自带的一个例子：

```
#!/bin/bash

# A small example program for using the new getopt(1) prog
ram.
# This program will only work with bash(1)
# An similar program using the tcsh(1) script language can
 be found
# as parse.tcsh

# Example input and output (from the bash prompt):
# ./parse.bash -a par1 'another arg' --c-long 'wow!*\?' -
cmore -b " very long "
# Option a
# Option c, no argument
# Option c, argument `more'
# Option b, argument ` very long '
# Remaining arguments:
# --> `par1'
# --> `another arg'
# --> `wow!*\?'

# Note that we use `"$@"' to let each command-
line parameter expand to a
# separate word. The quotes around `$@' are essential!
# We need TEMP as the `eval set --
' would nuke the return value of getopt.

#-o表示短选项，两个冒号表示该选项有一个可选参数，可选参数必须紧贴选项
#如-carg 而不能是-c arg
#--long表示长选项
```

```
#"$@"在上面解释过
# -n:出错时的信息
# -- ：举一个例子比较好理解：
#我们要创建一个名字为 "-f"的目录你会怎么办？
# mkdir -f #不成功，因为-f会被mkdir当作选项来解析，这时就可以使用
# mkdir -- -f 这样-f就不会被作为选项。

TEMP=`getopt -o ab:c:: --long a-long,b-long:,c-long:: \
     -n 'example.bash' -- "$@"`

if [ $? != 0 ] ; then echo "Terminating..." >&2 ; exit 1 ;
 fi

# Note the quotes around `$TEMP': they are essential!
#set 会重新排列参数的顺序，也就是改变$1,$2...$n的值，这些值在
getopt中重新排列过了
eval set -- "$TEMP"

#经过getopt的处理，下面处理具体选项。

while true ; do
        case "$1" in
                -a|--a-long) echo "Option a" ; shift ;;
                -b|--b-
long) echo "Option b, argument \`$2'" ; shift 2 ;;
                -c|--c-long)
                        # c has an optional argument. As w
e are in quoted mode,
                        # an empty parameter will be gener
ated if its optional
                        # argument is not found.
                        case "$2" in
                                "") echo "Option c, no arg
ument"; shift 2 ;;
                                *)  echo "Option c, argume
nt \`$2'" ; shift 2 ;;
                        esac ;;
                --) shift ; break ;;
                *) echo "Internal error!" ; exit 1 ;;
        esac
done
echo "Remaining arguments:"
for arg do
   echo '--> '"\`$arg'" ;
done
```

比如我们使用
./test -a -b arg arg1 -c
你可以看到,命令行中多了个arg1参数，在经过getopt和set之后，命令行会变为：
-a -b arg -c -- arg1

$1指向-a,$2指向-b,$3指向arg,$4指向-c,$5指向--,而多出的arg1则被放到了最后。

3.总结

一般小脚本手工处理也许就够了，getopts能处理绝大多数的情况，getopt较复杂，功能也更强大。
有问题请指出，不胜感激。

分类: Shell

Frank Tan
关注 - 0
粉丝 - 9
+加关注

7 0

(请您对文章做出评价)

« 上一篇：关于valgrind的 “Conditional jump or move depends on uninitialised value(s)”
» 下一篇：gtest测试重载接口

posted on 2010-03-01 21:32 Frank Tan 阅读(44208) 评论(3) 编辑 收藏

评论

#1楼 2013-07-17 16:04 指尖的跳动

写的不错，赞一个

支持(0) 反对(0)

#2楼 2013-12-05 10:44 风城

很清晰，不错！
自己写的小脚本，直接用 $1,$2
稍微复杂点的带两三个参数的，用了getopts
比较大的工程的install.sh，我看是用的getops

支持(0) 反对(0)

#3楼 2015-02-03 03:51 linuxdog

getopt 最牛的可以提取参数，被提取的参数可以随便放位置，getopts就不行，提取参数必须放在其他的参数前面才行

支持(0) 反对(0)