目 录

1 命令行参数 **2**
1.1 特殊的参数变量 3

2 特殊的参数变量 $# **3**
2.1 特殊的参数变量 $* 和 $@ 3

3 shift 移位命令 **4**

4 处理参数项 **4**
4.1 getopt 命令 5
4.2 getopts 命令 6

5 标准化选项 **7**

6 获取用户输入 **7**
6.1 read 命令 7
6.1.1 read 命令的各个选项 7
6.1.2 使用 read 命令读取文件 9

1 命令行参数

向 shell 脚本传递数据的最基本方式就是使用命令行参数。bash shell 将命令行参数赋值给一些特殊变量，这些变量称为位置参数。位置参数通过标准数字表示，其中 $0 为程序名称，$1 为第一个参数，$2 为第二个参数，以此类推，直到 $9 为第九个参数。

使用命令行参数的例子如下所示：

```
#!/bin/bash

factorial=1
for (( number=1; number <= $1; number++ ))
do
    factorial=$[ $factorial * $number ]
done
```

要使用这个程序，只要在命令行输入：

```
# test是文件名，5是程序的参数
./test 5
```

有时候可以利用程序名称，从而编写不同名称有不同功能的脚本，例子如下所示：

```
#!/bin/bash
# basename是一个命令，可以只返回程序的名称，不带路径
name=`basename $0`
if [ $name = "addem" ]
then
    total=$[ $1 + $2 ]
elif [ $name = "multem" ]
then
    total=$[ $1 * $2 ]
fi
```

然后只要将程序名称改为不同的名字，就有不一样的功能了，如下所示：

```
# test是原脚本名称
cp test addem
cp test multem
# 执行程序
./addem 2 5
./multem 2 5
```

需要注意的是，在使用参数之前，需要先使用-n 参数来检查命令行参数是否存在数据，例子如下：

```
#!/bin/bash
# 带双引号是考虑到了第一个参数有空格的情况
if [ -n "$1" ]
then
    echo "Hello $1, glad to meet you"
```

```
    else
        echo "Sorry, you didn't identify yourself"
    fi
```

1.1 特殊的参数变量

2 特殊的参数变量 $#

特殊变量 $# 中存储着执行脚本时的命令行参数个数，所以我们就可以利用它在使用参数前测试现有的参数个数，例子如下：

```
#!/bin/bash

if [ $# -ne 2 ]
then
    echo "Error"
else
    total=$[ $1 + $2 ]
    echo "The total is $total"
fi
```

通过这个变量还可以直接获取最后一个参数的值，不过此时必须用感叹号替代美元符号，格式如下：

```
$(!#)
```

2.1 特殊的参数变量 $* 和 $@

$* 和 $@ 这两个变量都包含了所有的命令行参数，$* 将所有命令行参数当作一个单词处理，而 $@ 把命令行参数当作多个单词处理，所以只有 $@ 可以被 for 命令遍历，例子如下所示：

```
#!/bin/bash
count=1
for param in "$*"
do
    echo "\$* Parameter #$count = $param"
    count=$[ $count + 1 ]
done

count=1
for param in "$@"
do
    echo "\$@ Parameter #$count: $param"
    count=$[ $count + 1 ]
done
```

3 shift 移位命令

shift 命令可以改变命令行参数的相对位置，将每个参数变量左移一位，$3 的值移给 $2，$2 移给 $1，原先 $1 的值被丢弃。需要注意的是，$0 的值保持不变。

这样子就可以先对第一个参数进行操作，然后对参数进行一次左移，再对第一个参数进行操作，如下例所示：

```
#!/bin/bash
count=1
while [ -n $1 ]
do
    echo "Parameter #$count = $1"
    count=$[ $count + 1]
    shift
done
```

shift 命令还可以带参数，指定左移几位，格式如下：

```
shift n
```

4 处理参数项

shell 程序的输入有时候是带参数项的，如下所示：

```
./testing -a -b test1 -c
```

例子如下所示：

```
#!/bin/bash
while [ -n "$1" ]
do
    case "$1" in
    -a)
        echo "Found the -a option";;
    -b)
        param=$2
        echo "Found the -b option, with parameter value $param"
        shift;;
    -c)
        echo "Found the -c option";;
    --)
        shift
        break;;
    *)
        echo "$1 is not an option";;
    esac
    shift
done
```

4.1 getopt 命令

getopt 命令可以接受任意形式的命令行选项和参数列表，并将这些选项和参数转换为适当的格式。

getopt 命令的格式如下所示：

```
getopt [options] <optstring> <parameters>
# optstring定义了命令行中的有效选项字母，并且定义了哪些选项字母需要参数值
# optstring由选项字母组成，需要参数值的选项字母后面需要放置一个冒号
# optstring的例子: ab:cd
```

getopt 的使用例子如下：

```
getopt ab:cd -a -b test1 -cd test2 test3
# 返回: -a -b test1 -c -d -- test2 test3
```

当 paramter 中的选项不在 optstring 中时，getopt 会产生错误消息，通过“-q”参数项可以忽略这个错误消息，例子如下：

```
getopt -q ab:cd -a -b test1 -cde test2 test3
```

通过 set 命令带“–”参数项可以将命令行参数项替换为 set 命令的输入值，set 命令和 getopt 的配合使用如下所示：

```
set -- `getopt -q ab:cd "$@"`
```

具体例子如下所示：

```
#!/bin/bash

set -- `getopt -q ab:c "$@"`
while [ -n "$1" ]
do
    case "$1" in
    -a)
        echo "Found the -a option";;
    -b)
        param="$2"
        echo "Found the -b option with the parameter $param"
        shift;;
    -c)
        echo "Found the -c option";;
    --)
        shift
        break;;
    *)
        echo "$1 is not an option";;
    esac
    shift
done
```

4.2 getopts 命令

getopts 命令的格式和 getopt 类似，如下所示：

```
getopts optstring variable
```

getopts 按顺序处理命令行参数，并给 variable 赋值。如果要禁止输出错误消息，需要在 optstring 前加上冒号“:”。

环境变量 OPTARG 包含需要参数值的选项要使用的值，环境变量 OPTIND 包含 getopts 停止处理时在参数列表中的位置。

使用 OPTARG 的例子如下所示：

```
#!/bin/bash

while getopts :ab:c opt
do
    case "$opt" in
    # getopts会把破折号-去掉，所以case选项中不需要破折号
    a)
        echo "Found the -a option";;
    b)
        echo "Found the -b option, with value $OPTARG";;
    c)
        echo "Found the -c option";;
    *)
        # 未知的参数项将变为问好?
        echo "Unknown option: $opt";;
    esac
done
```

可以利用 OPTIND 来跳过命令行参数的参数项，直接获得参数，例子如下所示：

```
#!/bin/bash

while getopts :ab:cd opt
do
    case "$opt" in
    a)
        echo "Found the -a option";;
    b)
        echo "Found the -b option, with value $OPTARG";;
    c)
        echo "Found the -c option";;
    d)
        echo "Found the -d option";;
    *)
        echo "Unknown option: $opt";;
    esac
done
# 利用shift和OPTIND来跳过命令行参数中的参数项
shift $[ $OPTIND - 1 ]

```

```
21    count=1
22    for param in "$@"
23    do
24        echo "Parameter $count: $param"
25        count=$[ $count + 1 ]
26    done
```

5 标准化选项

linux 世界中有一些字母选项具有某种标准含义，如果在 shell 脚本中使用这些选项，会使脚本变得更加便于使用。这些命令行选项如下所示：

选项	描述	选项	描述
-a	显示所有对象	-n	使用非交互式（批量）模式
-c	生成计数	-o	指定一个输出文件来重定向输出
-d	指定目录	-q	以 quiet 模式执行
-e	展开对象	-r	递归处理目录和文件
-f	指定读取数据的文件	-s	以 silent 模式执行
-h	显示命令的帮助信息	-v	生成 verbose 输出
-i	忽略大小写	-x	排除和拒绝
-l	生成长格式的输出	-y	设置所有提问的回答为 yes

6 获取用户输入

6.1 read 命令

read 命令可以将用户输入放入一个标准变量中，格式如下：

```
1    read variable
```

例子如下：

```
1    #!/bin/bash
2
3    echo -n "Enter your name: "
4    read name
5    echo "Hello $name, welcome to my program."
```

6.1.1 read 命令的各个选项

read 命令带“-p”选项时，可以直接在 read 命令中指定一个提示，如下例所示：

```
1    #!/bin/bash
2
```

```
read -p "Please enter your age: " age
days=$[ $age * 365 ]
echo "That makes you over $days days old!"
```

如果不指定变量，read 命令会把用户输入放到环境变量 REPLY 中，如下例所示：

```
#!/bin/bash

read -p "Enter a number: "
factorial=1
for (( count=1; count <= $REPLY; count++ ))
do
    factorial=$[ $factorial * $count ]
done
echo "The factorial of $REPLY is $factorial"
```

使用 read 命令时，为了防止脚本停下一直等待用户输入数据，可以带上“-t”选项，指定 read 命令等待输入的秒数。当计时器计时数满时，read 命令返回一个非零退出状态，例子如下：

```
#!/bin/bash

if read -t 5 -p "Please enter your name: " name
then
    echo "Hello $name, welcome to my script"
else
    # 换行
    echo
    echo "Sorry, too slow!"
fi
```

read 命令还可以带上“-n”选项，当用户输入字符达到预定数目时，自动退出，并将输入的数据赋值给变量，如下例所示：

```
#!/bin/bash

read -n1 -p "Do you want to continue [Y/N]? " answer
case $answer in
Y | y)
    echo
    echo "fine, continue on";;
N | n)
    echo
    echo "OK, goodbye"
    exit;;
esac
```

需要注意的是，当 read 命令自动退出时，不会自动换行。

read 命令带上“-s”选项时，不会让用户输入显示在终端上，例子如下：

```
#!/bin/bash

```

```
read -s -p "Enter your password: " pass
echo
echo "Is your password really $pass"
```

6.1.2 使用 read 命令读取文件

read 命令还可以读取文件，每一次 read 命令都会读取文件中的一行文本。当文件中没有可读的行时，read 命令将以非零退出状态退出。read 命令一般和 cat、管道相配合使用，cat 输出文件内容，再交给 read 命令读取，格式如下：

```
cat file | while read line
do
    commands
done
```

read 命令的使用如下例所示：

```
#!/bin/bash
count=1
cat test | while read line
do
    echo "Line $count: $line"
    count=$[ $count + 1 ]
done
```