循环与分支

笨办法学

概述

- 循环 Loops
- 嵌套循环 Nested Loops
- 循环控制 Loop Control
- 测试与分支 Testing and Branching

循环　Loops

- `for` 循环
- `while` 循环
- `until` 循环

for 循环

```
for arg in [list]
do
   command(s)...
done
```

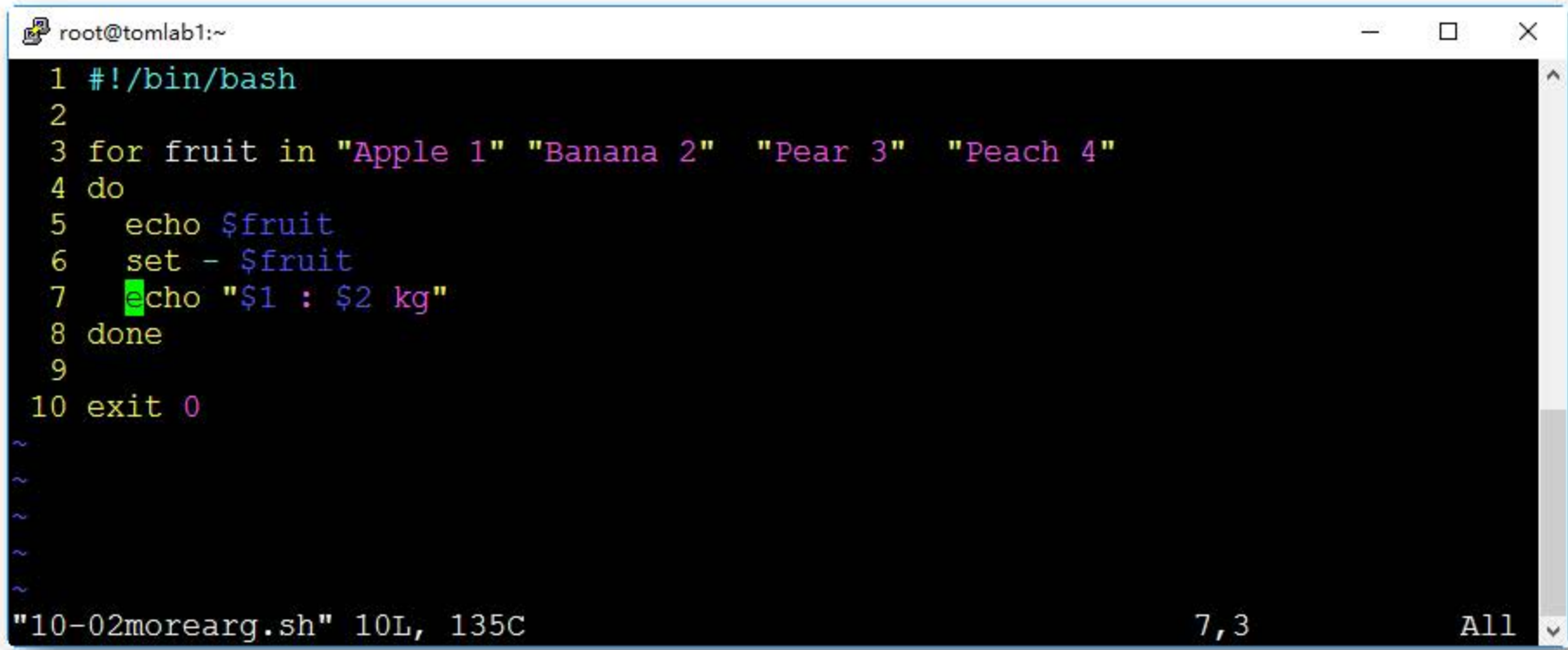
- 在循环的每次执行中， [list]中的值按顺序赋值给变量*arg*
- [list]中的参数
  - 可以加上双引号，阻止单词分割
  - 可以使用通配符
  - 可以使用命令替换
  - ……

示例1：一个简单的for循环

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 for fruit in Apple Banana Pear Peach
 4 do
 5   echo $fruit
 6 done
 7
 8 echo
 9
10 for fruit in "Apple Banana Pear Peach"
11 do
12   echo $fruit
13 done
14
15 exit 0
"10-01simpleforloop.sh" 15L, 153C                    12,3          All
```

示例2：每个[list]元素中都带有两个参数的for循环



示例3：对包含在变量中的文件列表进行操作

root@tomlab1:~

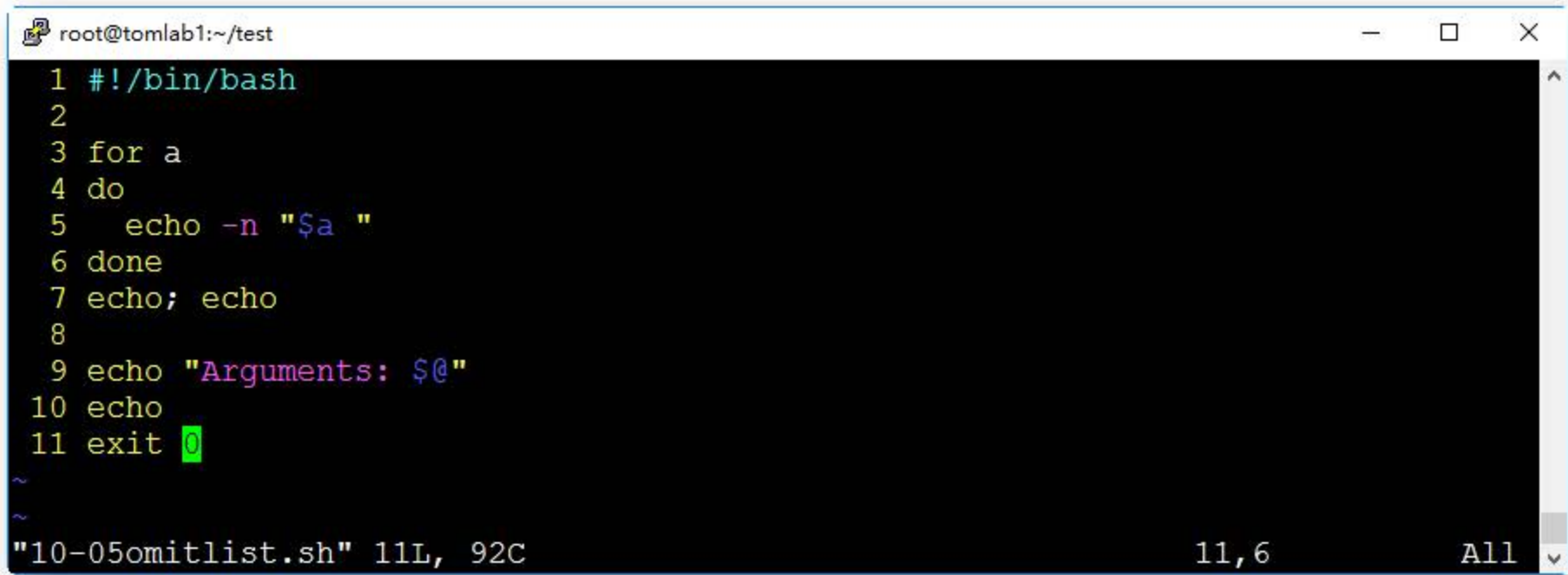
```
 1 #!/bin/bash
 2
 3 FILES="/etc/fstab
 4 /usr/bin/gawk
 5 /usr/bin/fakefile"
 6
 7 for strFile in $FILES
 8 do
 9   if [ ! -e "$strFile" ]; then
10     echo "$strFile does not exits."; echo
11     continue
12   fi
13
14   ls -l $strFile | awk '{print $9 " size: " $5}'
15   echo
16 done
17 exit 0
18
~
"10-03fileinfo.sh" 18L, 251C                                  17,6          All
```

示例4：在for循环中操作文件

root@tomlab1:~

```
 1 #!/bin/bash
 2 for file in *
 3 do
 4   ls -l "$file"
 5 done
 6
 7 echo;
 8
 9 for file in [ct]*
10 do
11   # rm -f $file
12   echo "Remove file \"$file\"."
13 done
14 exit 0
~
~
~
~
~
"10-04list-glob.sh" [New] 14L, 141C written                    14,6          All
```

示例5：在for循环中省略in [list]



示例6：使用命令替换来产生for循环的[list]

root@tomlab1:~/test

```
 1 #!/bin/bash
 2
 3 NUMBERS="1 2 3 4.5"
 4
 5 for number in `echo $NUMBERS`
 6 do
 7   echo -n "$number "
 8 done
 9
10 echo
11 exit 0
~
~
"10-06for-loopcmd.sh" 11L, 108C                    11,6          All
```

示例7：grep二进制文件

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 E_BADARGS=65
 4 E_NOFILE=66
 5
 6 if [ $# -ne 2 ]; then
 7   echo "Usage: `basename $0` searching_string filename"
 8   exit $E_BADARGS
 9 fi
10
11 if [ ! -f "$2" ]; then
12   echo "File \"$2\" does not exists."
13   exit $E_NOFILE
14 fi
15
16 IFS=$'\012'
17 for word in $( strings "$2" | grep "$1" )
18 do
19   echo $word
20 done
21 exit 0
"10-07bin-grep.sh" 21L, 303C                                  21,6          All
```

示例8：列出系统上的所有用户

root@tomlab1:~

```
 1 #/bin/bash
 2
 3 PASSWORD_FILE=/etc/passwd
 4 n=1
 5
 6 for UserName in $(awk 'BEGIN{FS=":"} {print $1}' < "$PASSWORD_FILE")
 7 do
 8   echo "USER #$n = $UserName"
 9   let "n += 1"
10 done
11
12 exit 0
~
~
"10-08userlist.sh" 12L, 173C                                  12,6          All
```

```
awk 'BEGIN{FS=":"} {print $1}' < "$PASSWORD_FILE"
```

示例9：在目录的所有文件中查找字符串

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 directory=/usr/bin
 4 fstring="Free Software Foundation"
 5
 6 for strFile in $(find $directory -type f -name '*' | sort)
 7 do
 8  strings -f $strFile | grep "$fstring" | sed -e "s%$directory%%"
 9 done
10
11 exit 0
~
"10-09findstring.sh" 11L, 208C                                11,6          All
```

语法：s/regexp/replacement/

作业：改进此脚本，通过命令行参数来指定目录和搜索字符串

示例10：列出目录中所有的符号链接

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 directory=${1-`pwd`}
 4
 5 echo "Symbolic links in directory \"$directory\""
 6
 7 for strFile in "$( find $directory -type l)"
 8 do
 9   echo "$strFile"
10 done | sort
11
12 exit 0
~
"10-10symlinks.sh" 12L, 173C                          12,6          All
```

示例11：将目录中所有的符号链接文件名保存到文件中

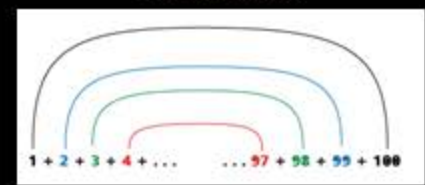
root@tomlab1:~

```
#!/bin/bash
# 列出目录中所有的符号文件，并保存到一个文件中

OUTFILE=symlinks.list.txt
directory=${1-`pwd`}

echo "Symbolic links in directory \"$directory\"" > "$OUTFILE"
echo "----------------------------------------" >> "$OUTFILE"

for strFile in "$( find $directory -type l)"
do
  echo "$strFile"
done | sort >> "$OUTFILE"

exit 0
~
~
"10-11symlinks.sh" 15L, 358C                                  15,6          All
```

示例12：类C语言风格的for循环

root@tomlab1:~

```
 1 #!/bin/bash
 2 # 循环指定数的再种方法
 3
 4 # 方法1：传统的办法
 5 n=0
 6 for a in 1 2 3 4 5 6 7 8 9 10; do
 7   let "n=$n+$a"
 8 done
 9 echo $n
10
11 # 方法：C语言风格
12 n=0
13 for ((a=1; a<=100; a++)); do
14   let "n=$n+$a"
15 done
16 echo $n
17
18 exit 0
~
"10-12gauss.sh" 18L, 238C                                    18,6          All
```



((...)) 双圆括号结构

- 与let命令很相似, 允许算术扩展和赋值.

```
a=$(( 5 + 3 ))
```

- 在Bash中，使用C语言风格变量操作的一种处理机制

```
(( a = 23 ))   # 变量赋值, "="两边允许有空格.
(( a++ ))      # 后置自加
(( a-- ))      # 后置自减
(( ++a ))      # 前置自加
(( --a ))      # 前置自减
```

◆循环　Loops

- ~~`for` 循环~~
- `while` 循环
- `until` 循环

while 循环

```
while [ condition ]
do
 command(s)...
done
```

```
while [[ condition ]]
do
 command(s)...
done
```

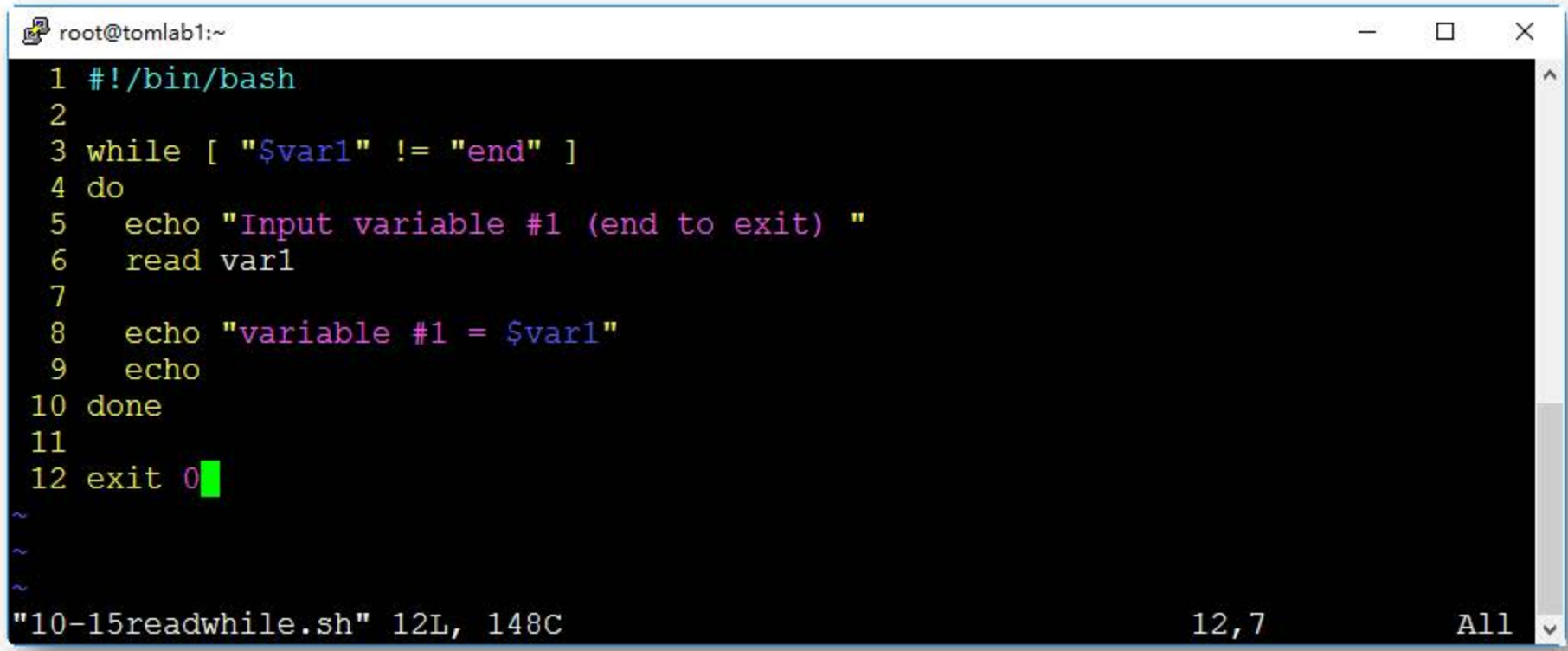
- 首先判断条件是否满足，如果满足，就执行command(s)
- 如果条件一直满足，那么就一直循环下去

示例1：简单的while循环

root@tomlab1:~

```
 1 #!/bin/bash
 2 n=0
 3 var=1
 4 LIMIT=100
 5
 6 while [ "$var" -le "$LIMIT" ]
 7 do
 8   if [ "$var" -eq "1" ]; then
 9     echo -n "$var"
10   else
11     echo -n "+$var"
12   fi
13   let "n=$n+$var"
14   let "var += 1"      # var=`expr $var + 1` 或  var=$((var+1))
15 done
16
17 echo -n "=$n"
18 echo;echo;
19 exit 0
~
"10-14simplewhile.sh" 19L, 267C                          19,7          All
```

示例2：在while循环中接受用户输入



示例3：多个条件的判断的while循环

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 var1=unset
 4 previous=$var1
 5
 6 while echo "previsour-variable = $previous"
 7       echo
 8       previous=$var1
 9       [ "$var1" != "end" ]
10 do
11   echo "Input variable #1 (end to exit)"
12   read var1
13   echo "variable #1 = $var1"
14 done
15
16 exit 0
~
"10-16multicond.sh" 16L, 243C                                 16,7          All
```

示例4： C风格的while循环

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 # C语言网络的while循环
 4 ((LIMIT = 10))
 5 ((a = 1))
 6
 7 while (( a<=LIMIT ))
 8 do
 9   echo -n "$a "
10   ((a += 1))
11 done
12
13 echo
14 exit 0
~
"10-17wh-loopc.sh" 14L, 140C                    14,6          All
```

示例5：使用函数作为while的条件

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 t=0
 4 condition()
 5 {
 6   ((t++))
 7   if [ $t -lt 5 ]; then
 8     return 0    # true
 9   else
10     return 1    # false
11   fi
12 }
13
14 while condition
15 do
16   echo "Still going: t = $t"
17 done
18
19 exit 0
"10-001wh-func.sh" 20L, 222C                          19,7          All
```

示例6：while与read命令结合进行文件的分析

root@tomlab1:~

```
 1 #!/bin/bash
 2 # 方法1
 3 cat /etc/hosts |
 4 while read line1
 5 do
 6   echo "$line1"
 7 done
 8
 9 # 方法2
10 echo "============================"
11
12 while read line2
13 do
14   echo "$line2"
15 done < /etc/hosts
16
17 exit 0
                                                    1,6           All
```

循环　Loops

- ~~`for` 循环~~
- ~~`while` 循环~~
- `until` 循环

until 循环

```
until [ condition ]
do
 command(s)...
done
```

- 首先判断条件是否满足，如果不满足，就执行command(s)
- 如果条件一直为false, 那么就一直循环下去

示例：until 循环

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 END_CONDITION=end
 4
 5 until [ "$var1" = "$END_CONDITION" ]
 6 do
 7   echo "Input vaiable #1 ($END_CONDITION to exit)"
 8   read var1
 9   echo "variable #1 = $var1"
10   echo
11 done
12
13 exit 0
~
~
"10-18until.sh" 13L, 185C                                  13,6          All
```

◆ 循环　Loops

- ~~`for` 循环~~
- ~~`while` 循环~~
- ~~`until` 循环~~

◆ 嵌套循环 Nested Loops

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 outer=1             # 外部循环计数器
 4 for a in 1 2 3 4 5
 5 do
 6   echo "Pass $outer in outer Loop."
 7   echo "----------------------"
 8   inner=1           # 重置内部循环计数器
 9   for b in 1 2 3 4 5
10   do
11     echo "Pass $inner in inner Loop."
12     let "inner+=1"
13   done
14
15   let "outer+=1"
16   echo
17 done
18
19 exit 0
"10-19nested-loop.sh" 19L, 305C                19,1          All
```

概述

- ~~循环 Loops~~
- ~~嵌套循环 Nested Loops~~
- 循环控制 Loop Control
- 测试与分支 Testing and Branching

循环控制 Loop Control

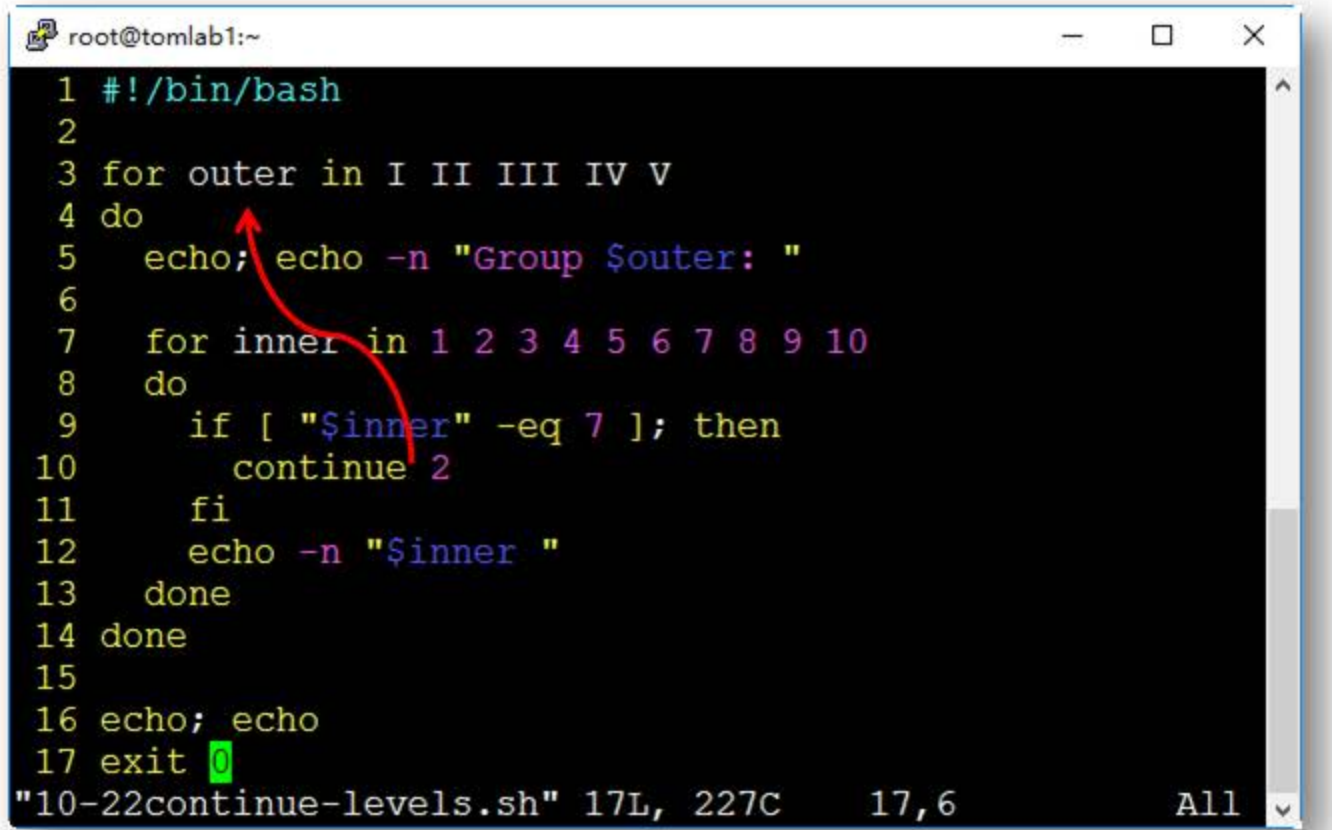
- **`break`**：跳出循环
- **`continue`**：忽略本次循环剩余的代码，跳到循环的头部
- **`break`**和 **`continue`**均可通过参数指定级别

示例：多层循环的break退出

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 for outerloop in 1 2 3 4 5
 4 do
 5   echo -n "Group $outerloop: "
 6
 7   for innerloop in 1 2 3 4 5
 8   do
 9     echo -n "$innerloop "
10     if [ "$innerloop" -eq 3 ]; then
11       break 2
12     fi
13   done
14   echo
15 done
16
17 echo
18 exit 0
~
"10-21break-levels.sh" 18 lines --5%--                    1,7           All
```

示例：多层循环的continue继续



概述

- ~~循环 Loops~~
- ~~嵌套循环 Nested Loops~~
- ~~循环控制 Loop Control~~
- 测试与分支 Testing and Branching

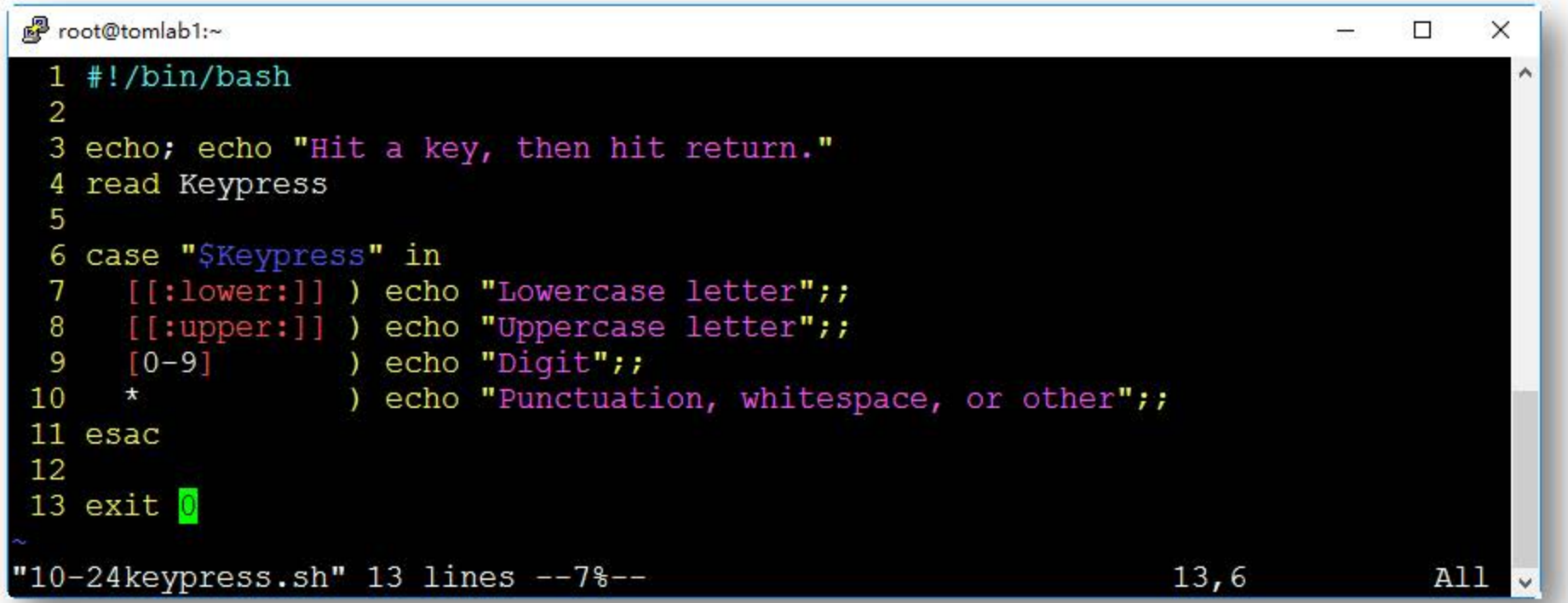
测试与分支 Testing and Branching

- `case...esac`
- `select`

case...esac

```
case "$variable" in ①

  "$condition1" )②
    commands...
    ;; ③

  "$condition2" )
    commands...
    ;;

  ...
esac④
```

示例1：简单的case结构



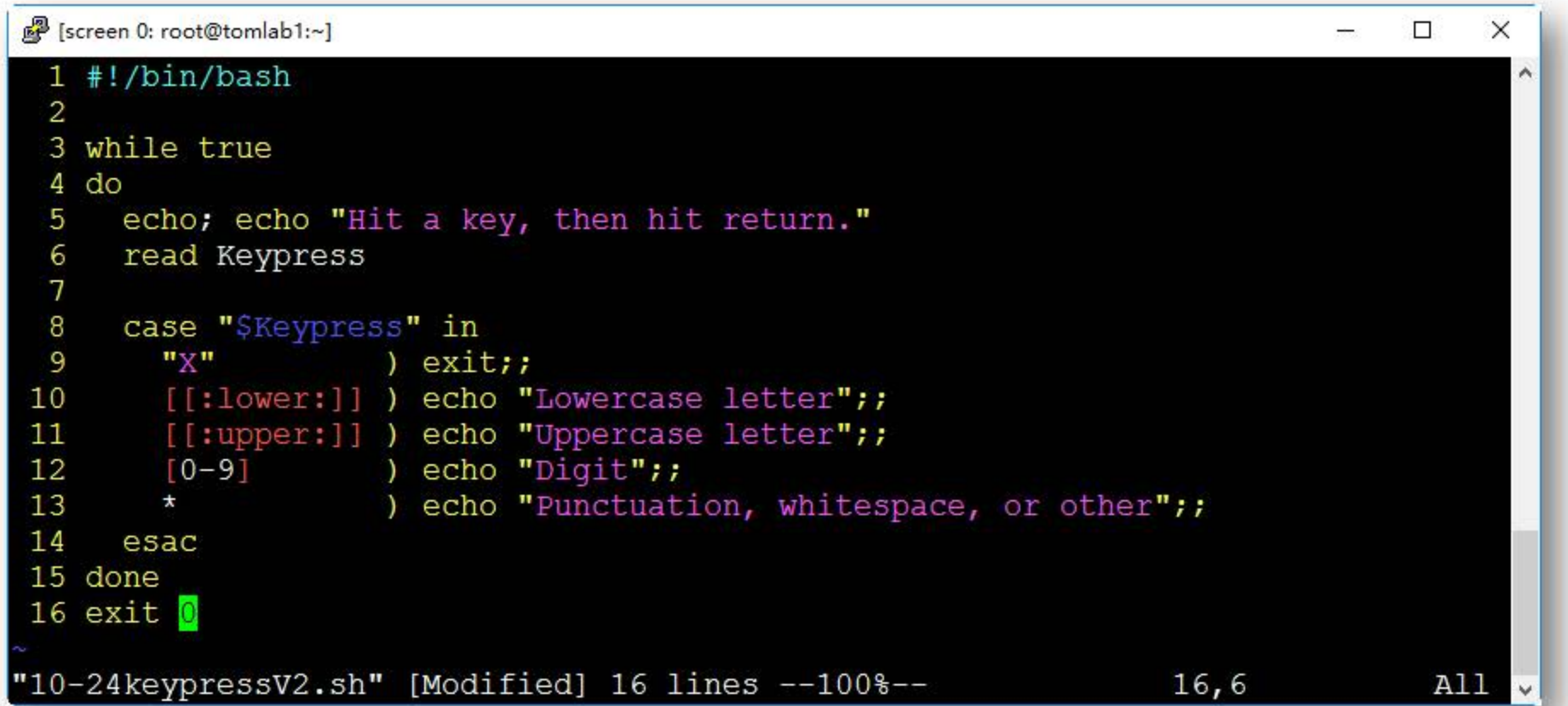
作业：修改此脚本，达到以下目的：

1、允许用户反复输入字符
2、echo每次输入的字符，当输入“X”时，才能结束脚本

查遗补缺：通配符

- `*` 匹配任意长度的任意字符
- `?` 匹配任意单个字符
- `[]` 匹配指定范围内的任意单个字符
- `[^]` 匹配指定范围之外的任意单个字符
- `[:space:]` 空白字符
- `[:punct:]` 标点符号
- `[:lower:]` 小写字母
- `[:upper:]` 大写字母
- `[:alpha:]` 大小写字母
- `[:digit:]` 数字
- `[:alnum:]` 数字和大小写字母

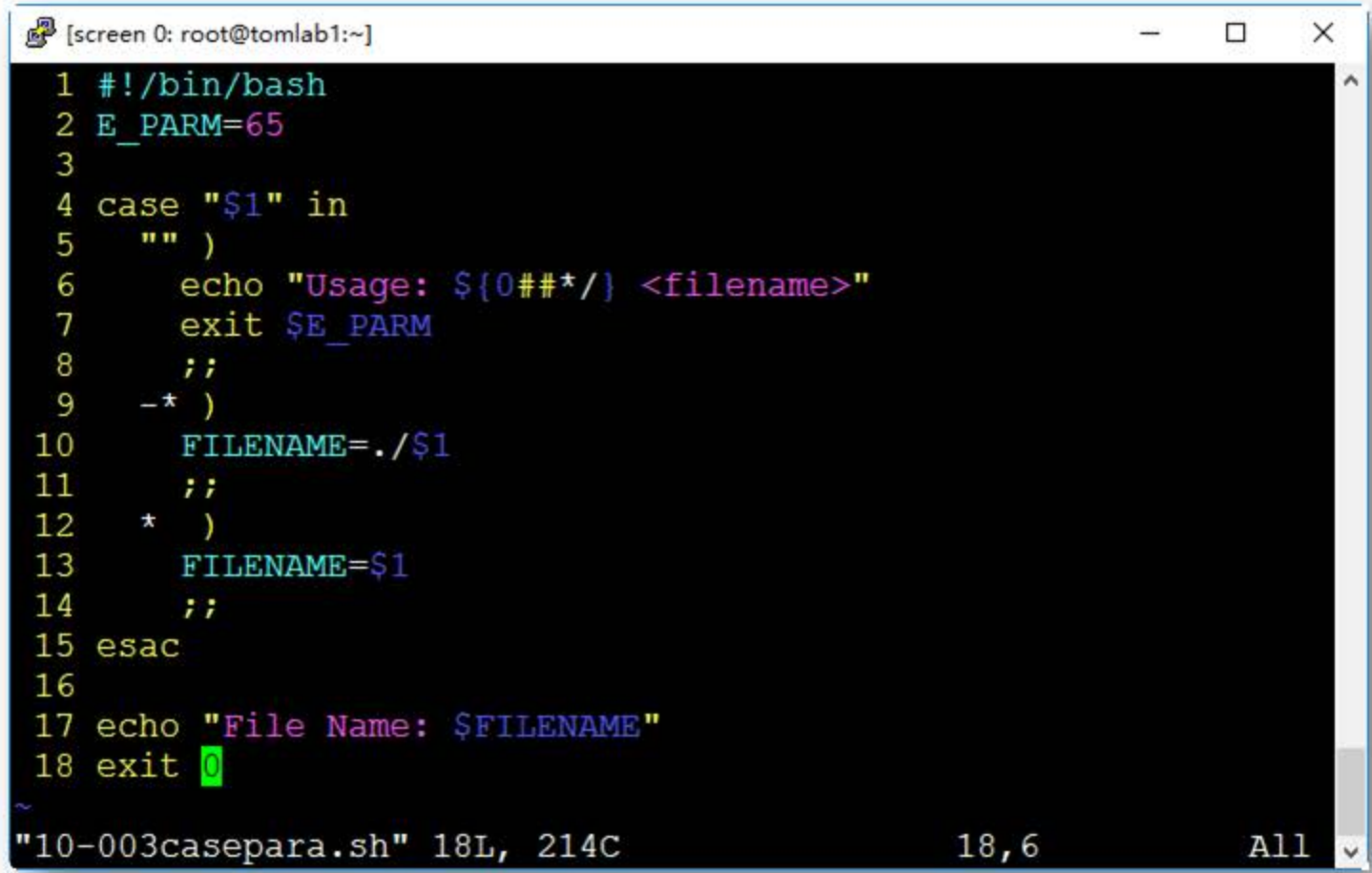
示例1练习的参考答案


[screen 0: root@tomlab1:~]
1 #!/bin/bash
2
3 while true
4 do
5 echo; echo "Hit a key, then hit return."
6 read Keypress
7
8 case "$Keypress" in
9 "X") exit;;
10 [[:lower:]]) echo "Lowercase letter";;
11 [[:upper:]]) echo "Uppercase letter";;
12 [0-9]) echo "Digit";;
13 *) echo "Punctuation, whitespace, or other";;
14 esac
15 done
16 exit 0
~
"10-24keypressV2.sh" [Modified] 16 lines --100%-- 16,6 All

示例2：使用case来创建菜单

```
$ ./10-25casemenu.sh

Contact List
------------

[E]vans, Roland
[J]ones, Mildred
[S]mith, Julie

输入E
Evans, Roland
(010)6123456
```

示例3：用case来测试命令行参数（1）



示例4：用case来测试命令行参数（2）

root@tomlab1:~

```
 1 #!/bin/bash
 2 # 假设传给脚本的参数有：-d|--debug -f|--conf ConfigureFile等
 3 while [ $# -gt 0 ]
 4 do
 5   case "$1" in
 6     -d | --debug )
 7       DEBUG=1    # 有-d或--debug参数
 8       ;;
 9     -f | --conf )
10       CONFFILE="$2"
11       shift
12       if [ ! -f $CONFFILE ]; then
13         echo "Error: Supplied file doesn't exist!"
14         exit $E_CONFFILE
15       fi
16       ;;
17   esac
18   shift          # 检查剩余参数
19 done
20
21 exit 0
"10-004casewhileshift.sh" 21L, 417C                    21,6          All
```

示例5： 使用命令替换来产生case变量

示例6：简单的字符串匹配

root@tomlab1:~

```
#!/bin/bash

match_string ()
{
  MATCH=0
  NOMATCH=90
  PARAMS=2       # 此函数需要2个参数
  BAD_PARAMS=091

  [ $# -eq $PARAMS ] || return $BAD_PARAMS

  case "$1" in
    "$2" ) return $MATCH ;;
    *    ) return $NOMATCH ;;
  esac
}
```

```
a=one
b=two
c=three
d=two

match_string $a; echo $?          # 参数个数错误 91
match_string $a $b; echo $?       # 不匹配 90
match_string $b $d; echo $?       # 匹配 0
exit 0
```

示例7：case结构中的通配符

```
1  # 检查字符串的第一个字符是否是字母表上的字符
2  isalpha ()
3  {
4    if [ -z "$1" ] ; then
5      return $FAILURE
6    fi
7
8    case "$1" in
9      [a-zA-Z]* ) return $SUCCESS ;;       # 以字母开头
10     *         ) return $FAILURE ;;
11   esac
12 }
13
14
15 # 测试整个字符串是否都是字母表上的字符
16 isalpha2 ()
17 {
18   [ $# -eq 1 ] || return $FAILURE        # 另一种判断传递参数的作法
19
20   case "$1" in
21     *[!a-zA-Z]* | "" ) return $FAILURE ;;
22     *                ) return $SUCCESS ;;
23   esac
24 }
```

测试与分支 Testing and Branching

- ~~case...esac~~
- select

select

```
select variable [in list]
do
 commands...
 break
done
```

示例1：使用select来创建菜单

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 PS3='Choose your favorite vegetable: '
 4 echo
 5
 6 select vegetable in "beans" "carrots" "potatoes"
 7 do
 8   echo
 9   echo "Your favorite veggie is $vegetable."
10   echo
11   break
12 done
13
14 exit 0
~
"10-29SelectVegetable.sh" 14L, 190C          14,6          All
```

示例2：使用函数中的select结构来创建菜单

root@tomlab1:~

```
 1 #!/bin/bash
 2
 3 PS3='Choose your favorite vegetable: '
 4 echo
 5
 6 choice_of ()
 7 {
 8   select vegetable
 9   do
10     echo
11     echo "Your favorite veggie is $vegetable."
12     echo
13     break
14   done
15 }
16
17 choice_of "beans" "carrots" "potatoes"
18
19 exit 0
~
"10-30SelectVegetable.sh" 19L, 229C          19,6          All
```

总结

- 循环 Loops
- 嵌套循环 Nested Loops
- 循环控制 Loop Control
- 测试与分支 Testing and Branching

其它课程


KVM
虚拟化
进阶与提高


Linux
开源虚拟化 KVM
极速入门


群集原理与实践精讲
Linux
24/7 service
LINUX Cluster
Theory AND Practice


做实验 ①
学 ②
存储


从零开始
构、建
你自己的 Linux
Linux


OpenStack
极速入门