马哥教育
IT人的高薪职业学院

SHELL脚本编程进阶

讲师：王晓春

本章内容

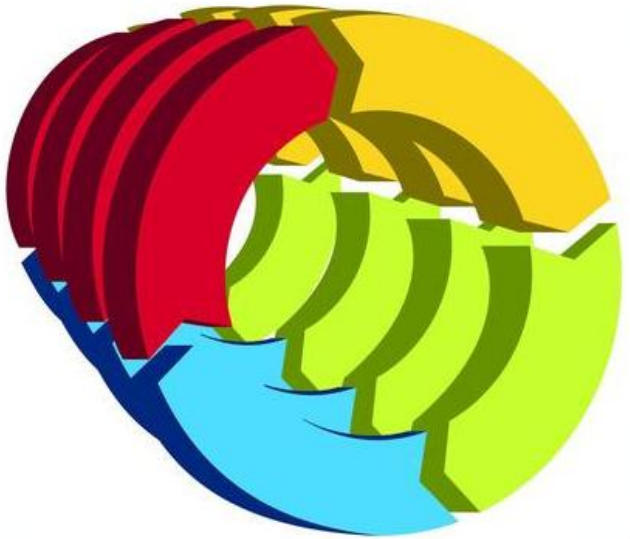
- 循环
- 信号捕捉
- 函数
- 数组
- 高级字符串操作
- 高级变量
- expect

流程控制

- 编程中的逻辑处理：
  - 顺序执行
  - 选择执行
  - 循环执行

循环

- 循环执行
  - 将某代码段重复运行多次
  - 重复运行多少次
    - 循环次数事先已知
    - 循环次数事先未知
  - 有进入条件和退出条件
- for, while, until



for循环

- for 变量名 in 列表;do
  循环体
  done
- 执行机制：
  依次将列表中的元素赋值给“变量名”；每次赋值后即执行一次循环体; 直到列表中的元素耗尽，循环结束

for循环

- 列表生成方式：
  - (1) 直接给出列表
  - (2) 整数列表：
    - (a) {start..end}
    - (b) $(seq [start [step]] end)
  - (3) 返回列表的命令
    - $(COMMAND)
  - (4) 使用glob，如：*.sh
  - (5) 变量引用
    - $@, $*

for特殊格式

- 双小括号方法，即((...))格式，也可以用于算术运算
- 双小括号方法也可以使bash Shell实现C语言风格的变量操作

```
I=10
((I++))
```

- for循环的特殊格式：

```
for ((控制变量初始化;条件判断表达式;控制变量的修正表达式))
do
    循环体
done
```

- 控制变量初始化：仅在运行到循环代码段时执行一次
- 控制变量的修正表达式：每轮循环结束会先进行控制变量修正运算，而后再做条件判断

练习:用for实现

- 1、判断/var/目录下所有文件的类型
- 2、添加10个用户user1-user10，密码为8位随机字符
- 3、/etc/rc.d/rc3.d目录下分别有多个以K开头和以S开头的文件；分别读取每个文件，以K开头的输出为文件加stop，以S开头的输出为文件名加start，如K34filename stop S66filename start
- 4、编写脚本，提示输入正整数n的值，计算1+2+...+n的总和
- 5、计算100以内所有能被3整除的整数之和
- 6、编写脚本，提示请输入网络地址，如192.168.0.0，判断输入的网段中主机在线状态
- 7、打印九九乘法表
- 8、在/testdir目录下创建10个html文件,文件名格式为数字N（从1到10）加随机8个字母，如：1AbCdeFgH.html
- 9、打印等腰三角形
- 10、猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第10天早上想再吃时，只剩下一个桃子了。求第一天共摘了多少？

while循环

- while CONDITION; do
  循环体
  done
- CONDITION：循环控制条件；进入循环之前，先做一次判断；每一次循环之后会再次做判断；条件为“true”，则执行一次循环；直到条件测试状态为“false”终止循环
- 因此：CONDTION一般应该有循环控制变量；而此变量的值会在循环体不断地被修正
- 进入条件：CONDITION为true
- 退出条件：CONDITION为false

练习：用while实现

- 1、编写脚本，求100以内所有正奇数之和
- 2、编写脚本，提示请输入网络地址，如192.168.0.0，判断输入的网段中主机在线状态，并统计在线和离线主机各多少
- 3、编写脚本，打印九九乘法表
- 4、编写脚本，利用变量RANDOM生成10个随机数字，输出这个10数字，并显示其中的最大值和最小值
- 5、编写脚本，实现打印国际象棋棋盘
- 6、后续六个字符串：efbaf275cd、4be9c40b8b、44b2395c46、f8c8873ce0、b902c16c8b、ad865d2f63是通过对随机数变量RANDOM随机执行命令： echo $RANDOM|md5sum|cut –c1-10 后的结果，请破解这些字符串对应的RANDOM值

until循环

- until CONDITION; do
  循环体
- done

- 进入条件： CONDITION 为false
- 退出条件： CONDITION 为true

循环控制语句continue

- 用于循环体中
- continue [N]：提前结束第N层的本轮循环，而直接进入下一轮判断；最内层为第1层

```
while CONDTIITON1; do
    CMD1
    ...
    if CONDITION2; then
        continue
    fi
    CMDn
    ...
done
```

循环控制语句break

- 用于循环体中
- break [N]：提前结束第N层循环，最内层为第1层

```
while CONDTIITON1; do
        CMD1
        ...
        if CONDITION2; then
                break
        fi
        CMDn
        ...
done
```

循环控制shift命令

- shift [n]
- 用于将参量列表 list 左移指定次数，缺省为左移一次。
- 参量列表 list 一旦被移动，最左端的那个参数就从列表中删除。while 循环遍历位置参量列表时，常用到 shift
- ./doit.sh a b c d e f g h
- ./shfit.sh a b c d e f g h

示例：doit.sh


```
#!/bin/bash
# Name: doit.sh
# Purpose: shift through command line arguments
# Usage: doit.sh [args]
while [ $# -gt 0 ]    # or (( $# > 0 ))
do
        echo  $*
        shift
done
```

示例：shift.sh


```
#!/bin/bash
#step through all the positional parameters
until [ -z "$1" ]
do
        echo "$1"
        shift
done
echo
```

创建无限循环

- while true; do
  循环体
- done

- until false; do
  循环体
- Done

练习

- 1、每隔3秒钟到系统上获取已经登录的用户的信息；如果发现用户hacker登录，则将登录时间和主机记录于日志/var/log/login.log中,并退出脚本
- 2、随机生成10以内的数字，实现猜字游戏，提示比较大或小，相等则退出
- 3、用文件名做为参数，统计所有参数文件的总行数
- 4、用二个以上的数字为参数，显示其中的最大值和最小值

特殊用法

- while循环的特殊用法（遍历文件的每一行）

```
while read line; do
        循环体
done < /PATH/FROM/SOMEFILE
```

- 依次读取/PATH/FROM/SOMEFILE文件中的每一行，且将行赋值给变量line

- 练习

扫描/etc/passwd文件每一行，如发现GECOS字段为空，则将用户名和单位电话为62985600填充至GECOS字段，并提示该用户的GECOS信息修改成功

select循环与菜单

- select variable in list
  do
  　　循环体命令
  done
- select 循环主要用于创建菜单，按数字顺序排列的菜单项将显示在标准错误上，并显示 PS3 提示符，等待用户输入
- 用户输入菜单列表中的某个数字，执行相应的命令
- 用户输入被保存在内置变量 REPLY 中
- select 是个无限循环，因此要记住用 break 命令退出循环，或用 exit 命令终止脚本。也可以按 ctrl+c 退出循环
- select 经常和 case 联合使用
- 与 for 循环类似，可以省略 in list，此时使用位置参量

函数介绍

- 函数function是由若干条shell命令组成的语句块，实现代码重用和模块化编程
- 它与shell程序形式上是相似的，不同的是它不是一个单独的进程，不能独立运行，而是shell程序的一部分
- 函数和shell程序比较相似，区别在于
  - Shell程序在子Shell中运行
  - 而Shell函数在当前Shell中运行。因此在当前Shell中，函数可以对shell中变量进行修改

定义函数

- 函数由两部分组成：函数名和函数体
- help function
- 语法一：

```
f_name （ ）{
        ...函数体...
}
```

- 语法二：

```
function f_name {
        ...函数体...
}
```

- 语法三：

```
function f_name （ ） {
        ...函数体...
}
```

函数使用

- 函数的定义和使用：
  - 可在交互式环境下定义函数
  - 可将函数放在脚本文件中作为它的一部分
  - 可放在只包含函数的单独文件中
- 调用：函数只有被调用才会执行

  调用：给定函数名

  函数名出现的地方，会被自动替换为函数代码
- 函数的生命周期：被调用时创建，返回时终止

函数返回值

- 函数有两种返回值：
- 函数的执行结果返回值：
  (1) 使用echo等命令进行输出
  (2) 函数体中调用命令的输出结果
- 函数的退出状态码：
  (1) 默认取决于函数中执行的最后一条命令的退出状态码
  (2) 自定义退出状态码，其格式为：
  return 从函数中返回，用最后状态命令决定返回值
  return 0 无错误返回
  return 1-255 有错误返回

交互式环境下定义和使用函数

- 示例：

```
dir() {
> ls -l
> }
```

- 定义该函数后，若在$后面键入dir，其显示结果同ls -l的作用相同

```
dir
```

- 该dir函数将一直保留到用户从系统退出，或执行了如下所示的unset命令

```
unset dir
```

在脚本中定义及使用函数

- 函数在使用前必须定义，因此应将函数定义放在脚本开始部分，直至shell首次发现它后才能使用
- 调用函数仅使用其函数名即可
- 示例：

```
cat func1
#!/bin/bash
# func1
hello()
{
  echo "Hello there today's date is `date +%F`"
}
echo "now going to the function hello"
hello
echo  “back from the function”
```

使用函数文件

- 可以将经常使用的函数存入函数文件，然后将函数文件载入shell
- 文件名可任意选取，但最好与相关任务有某种联系。例如：functions.main
- 一旦函数文件载入shell，就可以在命令行或脚本中调用函数。可以使用set命令查看所有定义的函数，其输出列表包括已经载入shell的所有函数
- 若要改动函数，首先用unset命令从shell中删除函数。改动完毕后，再重新载入此文件

创建函数文件

- 函数文件示例：

```
cat  functions.main
#!/bin/bash
#functions.main
findit()
{
   if [ $# -lt 1 ] ;  then
      echo "Usage:findit file"
      return 1
   fi
   find / -name $1 –print
}
```

载入函数

- 函数文件已创建好后，要将它载入shell
- 定位函数文件并载入shell的格式

  . filename 或 source filename
- 注意：此即<点> <空格> <文件名>

  这里的文件名要带正确路径
- 示例：

  上例中的函数，可使用如下命令

  . functions.main

检查载入函数

- 使用set命令检查函数是否已载入。set命令将在shell中显示所有的载入函数
- 示例：

```
set
 findit=( )
{
  if [ $# -lt 1 ]; then
  echo "usage :findit file";
  return 1
  fi
  find / -name $1 -print
}
…
```

执行shell函数

- 要执行函数，简单地键入函数名即可
- 示例：

```
findit groups
/usr/bin/groups
/usr/local/backups/groups.bak
```

删除shell函数

- 现在对函数做一些改动后，需要先删除函数，使其对shell不可用。使用unset命令完成删除函数
- 命令格式为：

  unset function_name
- 示例：

  unset findit

  再键入set命令，函数将不再显示
- 环境函数

  使子进程也可使用

  声明：export -f function_name

  查看：export -f 或 declare -xf

函数参数

- 函数可以接受参数：

传递参数给函数：调用函数时，在函数名后面以空白分隔给定参数列表即可；例如“testfunc arg1 arg2 ...”

在函数体中当中，可使用$1, $2, ...调用这些参数；还可以使用$@, $*, $#等特殊变量

函数变量

- 变量作用域：

  环境变量：当前shell和子shell有效

  本地变量：只在当前shell进程有效，为执行脚本会启动专用子shell进程；因此，本地变量的作用范围是当前shell脚本程序文件，包括脚本中的函数

  局部变量：函数的生命周期；函数结束时变量被自动销毁

- 注意：如果函数中有局部变量，如果其名称同本地变量，使用局部变量
- 在函数中定义局部变量的方法

  local NAME=VALUE

函数递归示例

- 函数递归：

  函数直接或间接调用自身

  注意递归层数

- 递归实例：

阶乘是基斯顿·卡曼于 1808 年发明的运算符号，是数学术语，一个正整数的阶乘（factorial）是所有小于及等于该数的正整数的积，并且有0的阶乘为1，自然数n的阶乘写作n!

n!=1×2×3×...×n

阶乘亦可以递归方式定义：0!=1，n!=(n-1)!×n

n!=n(n-1)(n-2)...1

n(n-1)! = n(n-1)(n-2)!

函数递归示例

- 示例：fact.sh

```
#!/bin/bash
#
fact() {
    if [ $1 -eq 0 -o $1 -eq 1 ]; then
                echo 1
    else
                echo $[$1*$(fact $[$1-1])]
    fi
}
fact $1
```

fork炸弹

- fork炸弹是一种恶意程序，它的内部是一个不断在fork进程的无限循环，实质是一个简单的递归程序。由于程序是递归的，如果没有任何限制，这会导致这个简单的程序迅速耗尽系统里面的所有资源
- 函数实现

  ```
  :(){ :|:& };:
  bomb() { bomb | bomb & }; bomb
  ```

- 脚本实现

  ```
  cat Bomb.sh
  #!/bin/bash
  ./$0|./$0&
  ```

练习

- 编写函数，实现OS的版本判断
- 编写函数，实现取出当前系统eth0的IP地址
- 编写函数，实现打印绿色OK和红色FAILED
- 编写函数，实现判断是否无位置参数，如无参数，提示错误

练习

- 编写服务脚本/root/bin/testsrv.sh，完成如下要求

(1) 脚本可接受参数：start, stop, restart, status

(2) 如果参数非此四者之一，提示使用格式后报错退出

(3) 如是start:则创建/var/lock/subsys/SCRIPT_NAME, 并显示“启动成功”

考虑：如果事先已经启动过一次，该如何处理？

(4) 如是stop:则删除/var/lock/subsys/SCRIPT_NAME, 并显示“停止完成”

考虑：如果事先已然停止过了，该如何处理？

(5) 如是restart，则先stop, 再start

考虑：如果本来没有start，如何处理？

(6) 如是status, 则如果/var/lock/subsys/SCRIPT_NAME文件存在，则显示“SCRIPT_NAME is running...”，如果/var/lock/subsys/SCRIPT_NAME文件不存在，则显示“SCRIPT_NAME is stopped...”

(7)在所有模式下禁止启动该服务，可用chkconfig 和 service命令管理

说明：SCRIPT_NAME为当前脚本名

练习

- 编写脚本/root/bin/copycmd.sh

(1) 提示用户输入一个可执行命令名称

(2) 获取此命令所依赖到的所有库文件列表

(3) 复制命令至某目标目录(例如/mnt/sysroot)下的对应路径下

如：/bin/bash ==> /mnt/sysroot/bin/bash

/usr/bin/passwd ==> /mnt/sysroot/usr/bin/passwd

(4) 复制此命令依赖到的所有库文件至目标目录下的对应路径下： 如：/lib64/ld-linux-x86-64.so.2 ==> /mnt/sysroot/lib64/ld-linux-x86-64.so.2

(5)每次复制完成一个命令后，不要退出，而是提示用户键入新的要复制的命令，并重复完成上述功能；直到用户输入quit退出

练习

- 编写函数实现两个数字做为参数，返回最大值
- 斐波那契数列又称黄金分割数列，因数学家列昂纳多·斐波那契以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……，斐波纳契数列以如下被以递归的方法定义：F（0）=0，F（1）=1，F（n）=F(n-1)+F(n-2)（n≥2）

  利用函数，求n阶斐波那契数列
- 汉诺塔（又称河内塔）问题是源于印度一个古老传说。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘，利用函数，实现N片盘的汉诺塔的移动步骤

信号捕捉trap

- trap '触发指令' 信号

  进程收到系统发出的指定信号后，将执行自定义指令，而不会执行原操作

- trap '' 信号

  忽略信号的操作

- trap '-' 信号

  恢复原信号的操作

- trap -p

  列出自定义信号操作

- trap finish EXIT

  当脚本退出时，执行finish函数

trap示例


```
#!/bin/bash
trap 'echo  “signal:SIGINT"'  int
trap -p
for((i=0;i<=10;i++))
do
        sleep 1
        echo $i
done
trap ''  int
trap -p
for((i=11;i<=20;i++))
do
        sleep 1
        echo $i
done
trap '-'  int
trap -p
for((i=21;i<=30;i++))
do
        sleep 1
        echo $i
done
```

数组

- 变量：存储单个元素的内存空间
- 数组：存储多个元素的连续的内存空间，相当于多个变量的集合
- 数组名和索引

  索引：编号从0开始，属于数值索引

  注意：索引可支持使用自定义的格式，而不仅是数值格式，即为关联索引，bash4.0版本之后开始支持

  bash的数组支持稀疏格式（索引不连续）
- 声明数组：

  declare -a ARRAY_NAME

  declare -A ARRAY_NAME　关联数组

  注意：两者不可相互转换

数组赋值

- 数组元素的赋值

  (1) 一次只赋值一个元素

  ARRAY_NAME[INDEX]=VALUE

  weekdays[0]="Sunday"

  weekdays[4]="Thursday"

  (2) 一次赋值全部元素

  ARRAY_NAME=("VAL1" "VAL2" "VAL3" ...)

  (3) 只赋值特定元素

  ARRAY_NAME=([0]="VAL1" [3]="VAL2" ...)

  (4) 交互式数组值对赋值

  read -a ARRAY

- 显示所有数组：declare -a

引用数组

- 引用数组元素
  ${ARRAY_NAME[INDEX]}
  注意：省略[INDEX]表示引用下标为0的元素
- 引用数组所有元素
  ${ARRAY_NAME[*]}
  ${ARRAY_NAME[@]}
- 数组的长度(数组中元素的个数)
  ${#ARRAY_NAME[*]}
  ${#ARRAY_NAME[@]}
- 删除数组中的某元素：导致稀疏格式
  unset ARRAY[INDEX]
- 删除整个数组
  unset ARRAY

数组数据处理

- 引用数组中的元素：

  数组切片：

  ${ARRAY[@]:offset:number}

  offset　　要跳过的元素个数

  number　　要取出的元素个数

  取偏移量之后的所有元素

  ${ARRAY[@]:offset}

- 向数组中追加元素：

  ARRAY[${#ARRAY[*]}]=value

- 关联数组：

  declare -A ARRAY_NAME

  ARRAY_NAME=([idx_name1]='val1' [idx_name2]='val2 '...)

  注意：关联数组必须先声明再调用

示例

- 生成10个随机数保存于数组中，并找出其最大值和最小值

```
#!/bin/bash
declare -i min max
declare -a nums
for ((i=0;i<10;i++));do
    nums[$i]=$RANDOM
    [ $i -eq 0  ] && min=${nums[$i]} &&  max=${nums[$i]}&& continue
    [ ${nums[$i]} -gt $max ] && max=${nums[$i]}
    [ ${nums[$i]} -lt $min ] && min=${nums[$i]}
done
echo “All numbers are ${nums[*]}”
echo Max is $max
echo Min is $min
```

示例

- 编写脚本，定义一个数组，数组中的元素对应的值是/var/log目录下所有以.log结尾的文件；统计出其下标为偶数的文件中的行数之和

```
#!/bin/bash
#
declare -a files
files=(/var/log/*.log)
declare -i lines=0
for i in $(seq 0 $[${#files[*]}-1]); do
    if [ $[$i%2] -eq 0 ];then
        let lines+=$(wc -l ${files[$i]} | cut -d' ' -f1)
    fi
done
echo "Lines: $lines."
```

练习

- 输入若干个数值存入数组中，采用冒泡算法进行升序或降序排序
- 将下图所示，实现转置矩阵matrix.sh

```
1 2 3                1 4 7
4 5 6   ===>         2 5 8
7 8 9                3 6 9
```

- 打印杨辉三角形

字符串切片

- ${#var}:返回字符串变量var的长度
- ${var:offset}:返回字符串变量var中从第offset个字符后（不包括第offset个字符）的字符开始，到最后的部分，offset的取值在0 到 ${#var}-1 之间(bash4.2后，允许为负值)
- ${var:offset:number}：返回字符串变量var中从第offset个字符后（不包括第offset个字符）的字符开始，长度为number的部分
- ${var: -length}：取字符串的最右侧几个字符

  注意：冒号后必须有一空白字符
- ${var:offset:-length}：从最左侧跳过offset字符，一直向右取到距离最右侧lengh个字符之前的内容
- ${var: -length:-offset}：先从最右侧向左取到length个字符开始，再向右取到距离最右侧offset个字符之间的内容

  注意：-length前空格

字符串处理

- 基于模式取子串

  ${var#*word}：其中word可以是指定的任意字符

  功能：自左而右，查找var变量所存储的字符串中，第一次出现的word, 删除字符串开头至第一次出现word字符串（含）之间的所有字符

  ${var##*word}：同上，贪婪模式，不同的是，删除的是字符串开头至最后一次由word指定的字符之间的所有内容

- 示例：

  file= “var/log/messages”

  ${file#*/}: log/messages

  ${file##*/}: messages

字符串处理

- ${var%word*}：其中word可以是指定的任意字符

  功能：自右而左，查找var变量所存储的字符串中，第一次出现的word, 删除字符串最后一个字符向左至第一次出现word字符串（含）之间的所有字符

  file="/var/log/messages"

  ${file%/*}: /var/log

- ${var%%word*}：同上，只不过删除字符串最右侧的字符向左至最后一次出现word字符之间的所有字符

- 示例：

  url=http://www.magedu.com:80

  ${url##*:}　　80

  ${url%%:*}　　http

字符串处理

- 查找替换

${var/pattern/substr}：查找var所表示的字符串中，第一次被pattern所匹配到的字符串，以substr替换之

${var//pattern/substr}: 查找var所表示的字符串中，所有能被pattern所匹配到的字符串，以substr替换之

${var/#pattern/substr}：查找var所表示的字符串中，行首被pattern所匹配到的字符串，以substr替换之

${var/%pattern/substr}：查找var所表示的字符串中，行尾被pattern所匹配到的字符串，以substr替换之

字符串处理

- 查找并删除

  ${var/pattern}：删除var表示的字符串中第一次被pattern匹配到的字符串

  ${var//pattern}：删除var表示的字符串中所有被pattern匹配到的字符串

  ${var/#pattern}：删除var表示的字符串中所有以pattern为行首匹配到的字符串

  ${var/%pattern}：删除var所表示的字符串中所有以pattern为行尾所匹配到的字符串

- 字符大小写转换

  ${var^^}：把var中的所有小写字母转换为大写

  ${var,,}：把var中的所有大写字母转换为小写

变量赋值

变量配置方式	str 没有配置	str 为空字符串	str 已配置非为空字符串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=	str 不变 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不变 var=$str
var=${str?expr}	expr 输出至 stderr	var=	var=$str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=$str

高级变量用法-有类型变量

- Shell变量一般是无类型的，但是bash Shell提供了declare和typeset两个命令用于指定变量的类型，两个命令是等价的
- declare [选项] 变量名
  - -r 声明或显示只读变量
  - -i 将变量定义为整型数
  - -a 将变量定义为数组
  - -A 将变量定义为关联数组
  - -f 显示已定义的所有函数名及其内容
  - -F 仅显示已定义的所有函数名
  - -x 声明或显示环境变量和函数
  - -l 声明变量为小写字母 declare –l var=UPPER
  - -u 声明变量为大写字母 declare –u var=lower

eval命令

- eval命令将会首先扫描命令行进行所有的置换，然后再执行该命令。该命令适用于那些一次扫描无法实现其功能的变量.该命令对变量进行两次扫描
- 示例：

```
[root@server ~]# CMD=whoami
[root@server ~]# echo  $CMD
   whoami
[root@server ~]# eval $CMD
   root
[root@server ~]# n=10
[root@server ~]# echo {0..$n}
      {0..10}
[root@server ~]# eval echo {0..$n}
      0 1 2 3 4 5 6 7 8 9 10
```

间接变量引用

- 如果第一个变量的值是第二个变量的名字，从第一个变量引用第二个变量的值就称为间接变量引用
- variable1的值是variable2，而variable2又是变量名，variable2的值为value，间接变量引用是指通过variable1获得变量值value的行为

  variable1=variable2

  variable2=value

间接变量引用

- bash Shell提供了两种格式实现间接变量引用

  eval tempvar=\$$variable1

  tempvar=${!variable1}

- 示例：

  [root@server ~]# N=NAME

  [root@server ~]# NAME=wangxiaochun

  [root@server ~]# N1=${!N}

  [root@server ~]# echo $N1

  wangxiaochun

  [root@server ~]# eval N2=\$$N

  [root@server ~]# echo $N2

  wangxiaochun

创建临时文件

- mktemp命令：创建并显示临时文件，可避免冲突
- mktemp [OPTION]... [TEMPLATE]
  - TEMPLATE: filenameXXX
    - X至少要出现三个
- OPTION：
  - -d: 创建临时目录
  - -p DIR或--tmpdir=DIR：指明临时文件所存放目录位置
- 示例：

```
mktemp /tmp/testXXX
tmpdir=`mktemp –d /tmp/testdirXXX`
mktemp  --tmpdir=/testdir  testXXXXXX
```

安装复制文件

- install命令：
  install [OPTION]... [-T] SOURCE DEST 单文件
  install [OPTION]... SOURCE... DIRECTORY
  install [OPTION]... -t DIRECTORY SOURCE...
  install [OPTION]... -d DIRECTORY...创建空目录
- 选项：
  -m MODE，默认755
  -o OWNER
  -g GROUP
- 示例：
  install -m 700 -o wang -g admins srcfile desfile
  install –m 770 –d /testdir/installdir

expect介绍

- expect 是由Don Libes基于Tcl（ Tool Command Language ）语言开发的，主要应用于自动化交互式操作的场景，借助 expect 处理交互的命令，可以将交互过程如：ssh登录，ftp登录等写在一个脚本上，使之自动化完成。尤其适用于需要对多台服务器执行相同操作的环境中，可以大大提高系统管理人员的工作效率

expect命令

- expect 语法：

  expect [选项] [-c cmds] [[-[f|b]] cmdfile] [args]

- 选项
  - -c：从命令行执行expect脚本，默认expect是交互地执行的

    示例：expect -c 'expect "\n" {send "pressed enter\n"}
  - -d：可以输出输出调试信息

    示例：expect -d ssh.exp
- expect中相关命令
  - spawn 启动新的进程
  - send 用于向进程发送字符串
  - expect 从进程接收字符串
  - interact 允许用户交互
  - exp_continue 匹配多个字符串在执行动作后加此命令

expect

- expect最常用的语法(tcl语言:模式-动作)
- 单一分支模式语法：
  - expect “hi” {send “You said hi\n"}
  - 匹配到hi后，会输出“you said hi” ，并换行
- 多分支模式语法：

```
expect "hi" { send "You said hi\n" } \
        "hehe" { send "Hehe yourself\n" } \
        "bye" { send "Good bye\n" }
```

  - 匹配hi,hello,bye任意字符串时，执行相应输出。等同如下：

```
expect {
        "hi" { send "You said hi\n"}
        "hehe" { send "Hehe yourself\n"}
        "bye" { send  " Good bye\n"}
}
```

示例


```
#!/usr/bin/expect
spawn scp /etc/fstab 192.168.8.100:/app
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send  "magedu\n" }
}
expect eof
```

示例


```
#!/usr/bin/expect
spawn ssh 192.168.8.100
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send  "magedu\n" }
}
interact
#expect eof
```

示例:变量


```
#!/usr/bin/expect
set ip 192.168.8.100
set user root
set password magedu
set timeout 10
spawn ssh $user@$ip
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send "$password\n" }
}
interact
```

示例:位置参数


```
#!/usr/bin/expect
set ip [lindex $argv 0]
set user [lindex $argv 1]
set password [lindex $argv 2]
spawn ssh $user@$ip
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send "$password\n" }
}
interact
#./ssh3.exp 192.168.8.100 root magedu
```

示例：执行多个命令


```
#!/usr/bin/expect
set ip [lindex $argv 0]
set user [lindex $argv 1]
set password [lindex $argv 2]
set timeout 10
spawn ssh $user@$ip
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send "$password\n" }
}
expect "]#" { send "useradd haha\n" }
expect "]#" { send "echo magedu |passwd --stdin haha\n" }
send "exit\n"
expect eof
#./ssh4.exp 192.168.8.100 root magedu
```

示例：shell脚本调用expect


```
#!/bin/bash
ip=$1
user=$2
password=$3
expect <<EOF
set timeout 20
spawn ssh $user@$ip
expect {
        "yes/no" { send "yes\n";exp_continue }
        "password" { send "$password\n" }
}
expect "]#" { send "useradd hehe\n" }
expect "]#" { send "echo magedu |passwd --stdin hehe\n" }
expect "]#" { send "exit\n" }
expect eof
EOF
#./ssh5.sh 192.168.8.100 root magedu
```

关于马哥教育

马哥教育
IT人的高薪职业学院

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：1661815153, 113228115
- QQ群：203585050, 279599283

祝大家学业有成

谢　谢