马哥教育
最专业的Linux培训机构

Shell脚本编程

http://www.magedu.com

本章内容

- 编程基础
- 脚本基本格式
- 变量
- 运算
- 条件测试
- 流程控制
- 函数
- 数组
- 高级字符串操作
- 高级变量
- 配置用户环境

编程基础

- 程序：指令+数据
- 程序编程风格：
  - 过程式：以指令为中心，数据服务于指令
  - 对象式：以数据为中心，指令服务于数据
- shell程序：提供了编程能力，解释执行

程序的执行方式

- 计算机：运行二进制指令
- 编程语言：
  - 低级：汇编
  - 高级：
    - 编译：高级语言-->编译器-->目标代码
      java,C#
    - 解释：高级语言-->解释器-->机器代码
      shell, perl, python

编程基本概念

- 编程逻辑处理方式：
  - 顺序执行
  - 循环执行
  - 选择执行
- shell编程：过程式、解释执行
  - 编程语言的基本结构：
    - 数据存储：变量、数组
    - 表达式：a + b
    - 语句：if

shell脚本基础

- shell脚本是包含一些命令或声明，并符合一定格式的文本文件
- 格式要求：首行shebang机制

  #!/bin/bash

  #!/usr/bin/python

  #!/usr/bin/perl
- shell脚本的用途有：
  - 自动化常用命令
  - 执行系统管理和故障排除
  - 创建简单的应用程序
  - 处理文本或文件

创建shell脚本

- 第一步：使用文本编辑器来创建文本文件
  - 第一行必须包括shell声明序列：#!
    #!/bin/bash
  - 添加注释
    注释以#开头
- 第二步：运行脚本
  - 给予执行权限，在命令行上指定脚本的绝对或相对路径
  - 直接运行解释器，将脚本作为解释器程序的参数运行

shell脚本示例

- #!/bin/bash
- #author: wang
- #Version: 1.0
- #Description:This script displays some information about your# environment

  echo "Greetings. The date and time are $(date)"
  echo "Your working directory is: $(pwd)"

脚本调试

- bash -n /path/to/some_script
  检测脚本中的语法错误

- bash -x /path/to/some_script
  调试执行

变量

- 变量：命名的内存空间
  - 数据存储方式：
    - 字符：
    - 数值：整型，浮点型
- 变量：变量类型
  - 作用：
    1. 数据存储格式
    2. 参与的运算
    3. 表示的数据范围
  - 类型：
    - 字符
    - 数值：整型、浮点型

变量

- 强类型：定义变量时必须指定类型、参与运算必须符合类型要求；调用未声明变量会产生错误

  如　java,python

- 弱类型：无须指定类型，默认均为字符型；参与运算会自动进行隐式类型转换；变量无须事先定义可直接调用

  如：bash 不支持浮点数

- 变量命名法则：

  1、不能使程序中的保留字：例如if，for；

  2、只能使用数字、字母及下划线，且不能以数字开头

  3、见名知义

  4、统一命名规则：驼峰命名法

bash中变量的种类

- 根据变量的生效范围等标准：

本地变量：生效范围为当前shell进程；对当前shell之外的其它shell进程，包括当前shell的子shell进程均无效

环境变量：生效范围为当前shell进程及其子进程

局部变量：生效范围为当前shell进程中某代码片断(通常指函数)

位置变量：$1，$2，...来表示，用于让脚本在脚本代码中调用通过命令行传递给它的参数

特殊变量：$?，$0，$*，$@，$#

本地变量

- 变量赋值：name='value'，
- 可以使用引用value：
  - (1) 可以是直接字串；name="root"
  - (2) 变量引用：name="$USER"
  - (3) 命令引用：name=`COMMAND`，name=$(COMMAND)
- 变量引用：${name}，$name
  - ""：弱引用，其中的变量引用会被替换为变量值
  - ''：强引用，其中的变量引用不会被替换为变量值，而保持原字符串
- 显示已定义的所有变量：set
- 删除变量：unset name

练习

- 1、编写脚本/root/bin/systeminfo.sh,显示当前主机系统信息，包括主机名，IPv4地址，操作系统版本，内核版本，CPU型号，内存大小，硬盘大小。
- 2、编写脚本/root/bin/backup.sh，可实现每日将/etc/目录备份到/root/etcYYYY-mm-dd中
- 3、编写脚本/root/bin/disk.sh,显示当前硬盘分区中空间利用率最大的值
- 4、编写脚本/root/bin/links.sh,显示正连接本主机的每个远程主机的IPv4地址和连接数，并按连接数从大到小排序

环境变量

- 变量声明、赋值：

  export name=VALUE

  declare -x name=VALUE

- 变量引用：$name，${name}
- 显示所有环境变量：

  export

  env

  printenv

- 删除：unset name
- bash有许多内建的环境变量：PATH，SHELL，USRE,UID，HISTSIZE，HOME，PWD，OLDPWD，HISTFILE，PS1

只读和位置变量

- 只读变量：只能声时，但不能修改和删除

  readonly name

  declare -r name

- 位置变量：在脚本代码中调用通过命令行传递给脚本的参数

  $1，$2，...：对应第1、第2等参数，shift [n]换位置

  $0：命令本身

  $*：传递给脚本的所有参数，全部参数合为一个字符串

  $@：传递给脚本的所有参数，每个参数为独立字符串

  $#：传递给脚本的参数的个数

  $@ $* 只在被双引号包起来的时候才会有差异

示例：判断给出的文件的行数

```
linecount="$(wc -l $1| cut -d' ' -f1)"
echo "$1 has $linecount lines."
```

算术运算

- bash中的算术运算:help let

  +，-，*，/，%取模（取余），**（乘方）

  实现算术运算：

  (1) let var=算术表达式

  (2) var=$[算术表达式]

  (3) var=$((算术表达式))

  (4) var=$(expr arg1 arg2 arg3 ...)

  (5) declare -i var = 数值

  (6) echo '算术表达式' | bc

- 乘法符号有些场景中需要转义，如*
- bash有内建的随机数生成器：$RANDOM（1-32767）

  echo $[$RANDOM%50] ：0-49之间随机数

赋值

- 增强型赋值：

  +=, -=, *=, /=, %=

- let varOPERvalue

  例如:let count+=3

  自加3后自赋值

- 自增，自减：

  let var+=1

  let var++

  let var-=1

  let var--

练习

- 1、编写脚本/root/bin/sumid.sh，计算/etc/passwd文件中的第10个用户和第20用户的ID之和
- 2、编写脚本/root/bin/sumspace.sh，传递两个文件路径作为参数给脚本，计算这两个文件中所有空白行之和
- 3、编写脚本/root/bin/sumfile.sh,统计/etc，/var，/usr目录中共有多少个一级子目录和文件

逻辑运算

- true, false
  1, 0
- 与:
  1 与 1 = 1
  1 与 0 = 0
  0 与 1 = 0
  0 与 0 = 0
- 或:
  1 或 1 = 1
  1 或 0 = 1
  0 或 1 = 1
  0 或 0 = 0

逻辑运算

- 非：！

  ！1 = 0

  ！0 = 1

- 短路运算：

  短路与：

  第一个为0，结果必定为0；

  第一个为1，第二个必须要参与运算；

  短路或：

  第一个为1，结果必定为1；

  第一个为0，第二个必须要参与运算；

- 异或：^

  异或的两个值,相同为假，不同为真

聚集命令

- 有两种聚集命令的方法：
  - 复合式：date； who | wc -l
    命令会一个接一个地运行
  - 子shell：(date； who | wc -l) >>/tmp/trace
    所有的输出都被发送给单个STDOUT和STDERR

退出状态

- 进程使用退出状态来报告成功或失败
  - 0 代表成功，1－255代表失败
  - $? 变量保存最近的命令退出状态
- 例如：

```
$ ping -c1 -W1 hostdown &> /dev/null
$ echo $?
```

退出状态码

- bash自定义退出状态码

  exit [n]：自定义退出状态码；

  注意：脚本中一旦遇到exit命令，脚本会立即终止；终止退出状态取决于exit命令后面的数字

  注意：如果未给脚本指定退出状态码，整个脚本的退出状态码取决于脚本中执行的最后一条命令的状态码

条件测试

- 判断某需求是否满足，需要由测试机制来实现；
  专用的测试表达式需要由测试命令辅助完成测试过程；
- 评估布尔声明，以便用在条件性执行中
  - 若真，则返回0
  - 若假，则返回1
- 测试命令：
  - test EXPRESSION
  - [EXPRESSION]
  - [[EXPRESSION]]

  注意：EXPRESSION前后必须有空白字符

条件性的执行操作符

- 根据退出状态而定，命令可以有条件地运行
  - && 代表条件性的AND THEN
  - || 代表条件性的OR ELSE
- 例如：

```
$ grep -q no_such_user /etc/passwd \
|| echo 'No such user'
No such user
$ ping -c1 -W2 station1 &> /dev/null  \
>    && echo "station1 is up"         \
>    || (echo 'station1 is unreachable'; exit 1)
station1 is up
```

test命令

- 长格式的例子：

```
$ test "$A" ==  "$B" && echo "Strings are equal"
$ test “$A” -eq “$B”  \
     && echo "Integers are equal"
```

- 简写格式的例子：

```
$ [ "$A" ==  "$B" ] && echo "Strings are equal"
$ [ "$A" -eq "$B" ] && echo "Integers are equal"
```

bash的测试类型

- 数值测试：
  - -gt：是否大于
  - -ge：是否大于等于
  - -eq：是否等于
  - -ne：是否不等于
  - -lt：是否小于
  - -le：是否小于等于

bash的测试类型

- 字符串测试：

  ==：是否等于；

  >：ascii码是否大于ascii码

  <：是否小于

  !=：是否不等于

  =~：左侧字符串是否能够被右侧的PATTERN所匹配

  注意：此表达式一般用于[[ ]]中；扩展的正则表达式

  -z "STRING"：字符串是否为空，空为真，不空为假

  -n "STRING"：字符串是否不空，不空为真，空为假

注意：用于字符串比较时的用到的操作数都应该使用引号

练习

- 1、编写脚本/root/bin/argsnum.sh，接受一个文件路径作为参数；如果参数个数小于1，则提示用户“至少应该给一个参数”，并立即退出；如果参数个数不小于1，则显示第一个参数所指向的文件中的空白行数
- 2、编写脚本/root/bin/hostping.sh，接受一个主机的IPv4地址做为参数，测试是否可连通。如果能ping通，则提示用户“该IP地址可访问”；如果不可ping通，则提示用户“该IP地址不可访问”
- 3、编写脚本/root/bin/checkdisk.sh，检查磁盘分区空间和inode使用率率，如果超过80%，就发广播警告空间将满

文件测试

- 存在性测试
  - -a FILE：同-e
  - -e FILE：文件存在性测试，存在为真，否则为假
- 存在性及类别测试
  - -b FILE：是否存在且为块设备文件
  - -c FILE：是否存在且为字符设备文件
  - -d FILE：是否存在且为目录文件
  - -f FILE：是否存在且为普通文件
  - -h FILE 或 -L FILE：存在且为符号链接文件
  - -p FILE：是否存在且为命名管道文件
  - -S FILE：是否存在且为套接字文件

文件测试

- 文件权限测试：
  - -r FILE：是否存在且可读
  - -w FILE：是否存在且可写
  - -x FILE：是否存在且可执行
- 文件特殊权限测试：
  - -u FILE：是否存在且拥有suid权限
  - -g FILE：是否存在且拥有sgid权限
  - -k FILE：是否存在且拥有sticky权限

文件测试

- 文件大小测试：
  - -s FILE：是否存在且非空
- 文件是否打开：
  - -t fd：fd表示文件描述符是否已经打开且与某终端相关
  - -N FILE：文件自动上一次被读取之后是否被修改过
  - -O FILE：当前有效用户是否为文件属主
  - -G FILE：当前有效用户是否为文件属组

文件测试

- 双目测试：

  FILE1 -ef FILE2：FILE1与FILE2是否指向同一个设备上的相同inode

  FILE1 -nt FILE2：FILE1是否新于FILE2

  FILE1 -ot FILE2：FILE1是否旧于FILE2

组合测试条件

- 第一种方式：

  COMMAND1 && COMMAND2 并且
  COMMAND1 || COMMAND2 或者
  ! COMMAND 非
  如：[[-r FILE]] && [[-w FILE]]

- 第二种方式：

  EXPRESSION1 -a EXPRESSION2 并且
  EXPRESSION1 -o EXPRESSION2 或者
  ! EXPRESSION
  必须使用测试命令进行；

- 示例：

```
# [ -z "$HOSTNAME" -o $HOSTNAME "==\
"localhost.localdomain" ] && hostname www.magedu.com
# [ -f /bin/cat -a -x /bin/cat ] && cat /etc/fstab
```

练习

- 1、编写脚本/bin/per.sh,判断当前用户对指定的参数文件，是否不可读并且不可写
- 2、编写脚本/root/bin/excute.sh ，判断参数文件是否为sh后缀的普通文件，如果是，添加所有人可执行权限，否则提示用户非脚本文件
- 3、编写脚本/root/bin/nologin.sh和login.sh,实现禁止和充许普通用户登录系统

使用read命令来接受输入

- 使用read来把输入值分配给一个或多个shell变量：

  -p　指定要显示的提示

  -t TIMEOUT

  read　从标准输入中读取值，给每个单词分配一个变量

  所有剩余单词都被分配给最后一个变量

  read -p "Enter a filename: " FILE

流程控制

- 过程式编程语言：
  - 顺序执行
  - 选择执行
  - 循环执行

条件选择if语句

- 选择执行：
- 注意：if语句可嵌套
- 单分支

```
if 判断条件;then
        条件为真的分支代码
fi
```

- 双分支

```
if 判断条件; then
        条件为真的分支代码
else
        条件为假的分支代码
fi
```

if 语句

- 多分支

```
if 判断条件1; then
        if-true
elif 判断条件2; then
        if-ture
elif 判断条件3; then
        if-ture
        ...
else
        all-false
fi
```


- 逐条件进行判断，第一次遇为“真”条件时，执行其分支，而后结束整个if语句

if实例

- 根据命令的退出状态来执行命令

```
if ping -c1 -W2 station1 &> /dev/null; then
    echo 'Station1 is UP'
elif grep "station1" ~/maintenance.txt &> /dev/null
  then
    echo 'Station1 is undergoing maintenance'
 else
    echo 'Station1 is unexpectedly DOWN!'
    exit 1
fi
```

条件判断：case语句

```
case 变量引用 in
PAT1)
        分支1
        ;;
PAT2)
        分支2
        ;;
...
*)
        默认分支
        ;;
esac
```

case支持glob风格的通配符：

*：任意长度任意字符

?：任意单个字符

[]：指定范围内的任意单个字符

a|b：a或b

练习

- 1、编写脚本/root/bin/createuser.sh，实现如下功能：使用一个用户名做为参数，如果指定参数的用户存在，就显示其存在，否则添加之；显示添加的用户的id号等信息
- 2、编写脚本/root/bin/yesorno.sh，提示用户输入yes或no，并判断用户输入的是yes还是no,或是其它信息
- 3、编写脚本/root/bin/filetype.sh,判断用户输入文件路径，显示其文件类型（普通，目录，链接，其它文件类型）
- 4、编写脚本/root/bin/checkint.sh,判断用户输入的参数是否为正整数

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；
    - $@, $*

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，密码同用户名
- 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开头的文件输出为文件加stop，以S开头的文件输出为文件名加start

  “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、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
- 6、打印九九乘法表

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，并统计在线主机和离线主机各多少
- 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

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
```

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。

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，则填充用户名和单位电话为62985600，并提示该用户的GECOS信息修改成功。

特殊用法

- 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

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与case

- 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。
- 语法一：

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

- 语法二：

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

- 语法三：

```
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。首先删除函数，使其对shell不可用。使用unset命令完成此功能.
- 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

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

函数递归实例

- 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
```

练习

- 1、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...”

其中：SCRIPT_NAME为当前脚本名

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

练习

- 2、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

- 3、编写函数实现两个数字做为参数，返回最大值
- 4、编写函数实现数字的加减乘除运算，例如输入 1 + 2，，将得出正确结果
- 5、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
- 6、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

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

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

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

- 数组的长度(数组中元素的个数)：

  ${#ARRAY_NAME[*]}

  ${#ARRAY_NAME[@]}

- 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 -a rand
declare -i max=0
declare -i min=32767
for i in {0..9}; do
        rand[$i]=$RANDOM
        echo ${rand[$i]}
        [ ${rand[$i]} -gt $max ] && max=${rand[$i]}
        [ ${rand[$i]} -lt $min ] && min=${rand[$i]}
done
echo "Max: $max  Min:$min"
```

示例

- 编写脚本，定义一个数组，数组中的元素是/var/log目录下所有以.log结尾的文件；要统计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."
```

数组数据处理

- 引用数组中的元素：
  - 所有元素：${ARRAY[@]}，${ARRAY[*]}
  - 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[*]}]
- 删除数组中的某元素：导致稀疏格式
  - unset ARRAY[INDEX]
- 关联数组：
  - declare -A ARRAY_NAME 注意：必须先声明，再调用
  - ARRAY_NAME=([idx_name1]='val1' [idx_name2]='val2'...)

练习

- 1、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

字符串处理

- bash的字符串处理工具：
- 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: -lengh}：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: -lengh}：从最左侧跳过offset字符，一直取到字符串的最右侧lengh个字符之前

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字符之间的所有字符

  ${var##*word}：同上，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/substi}：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substi替换之

  ${var//pattern/substi}：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substi替换之

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

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

字符串处理

- 查找并删除：

  ${var/pattern}：查找var所表示的字符串中，删除第一次被pattern所匹配到的字符串

  ${var//pattern}：所有

  ${var/#pattern}：行首

  ${var/%pattern}：行尾

- 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

- ${var:-value}：如果var为空或未设置，那么返回value；否则，则返回var的值
- ${var:+value}：如果var不空，则返回value，否则返回空值
- ${var:=value}：如果var为空或未设置，那么返回value，并将value赋值给var；否则，则返回var的值
- ${var:?error_info}：如果var为空或未设置，那么在当前终端打印error_info；否则，则返回var的值

- 为脚本程序使用配置文件，实现变量赋值
  (1) 定义文本文件，每行定义“name=value”
  (2) 在脚本中source此文件即可

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 将变量设置为只读属性

  -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 显示此脚本前定义过的所有函数名及其内容

  -F 仅显示此脚本前定义过的所有函数名

  -x 将变量声明为环境变量

  -l 将变量值转为小写字母 declare -l var=UPPER

  -u 将变量值转为大写字母 declare -u var=lower

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=variable2
- variable2=value
- 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

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 $N1
wangxiaochun
[root@server ~]# eval N2=\$$A
[root@server ~]# echo $2
wangxiaochun
```

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
```

创建临时文件

- mktemp命令：创建的临时文件可避免冲突
- mktemp [OPTION]... [TEMPLATE]
  - TEMPLATE：filename.XXX
    - 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/test.XXX
#tmpdir=`mktemp -d /tmp/testdir.XXX`
#mktemp  --tmpdir=/testdir  test.XXXXXX
```

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 file1 file2
  install -m -d /testdir/installdir

bash如何展开命令行

- 把命令行分成单个命令词
- 展开别名
- 展开大括号种的声明（{}）
- 展开波浪符声明（~）
- 命令替换$()　和　``）
- 再次把命令行分成命令词
- 展开文件通配（*、?、[abc]等等）
- 准备I/O重导向（<、>）
- 运行命令

防止扩展

- 反斜线（\）会使随后的字符按原意解释

```
$ echo Your cost: \$5.00
Your cost: $5.00
```

- 加引号来防止扩展
  - 单引号（'）防止所有扩展
  - 双引号（"）也防止所有扩展，但是以下情况例外：
    - $（美元符号） — 变量扩展
    - `（反引号） — 命令替换
    - \（反斜线） — 禁止单个字符扩展
    - !（叹号） — 历史命令替换

bash的配置文件

- 按生效范围划分，存在两类：
- 全局配置：
  - /etc/profile
  - /etc/profile.d/*.sh
  - /etc/bashrc
- 个人配置：
  - ~/.bash_profile
  - ~/.bashrc

shell登录两种方式

- 交互式登录：

  (1)直接通过终端输入账号密码登录；

  (2)使用“su - UserName”切换的用户

  执行顺序：

  /etc/profile --> /etc/profile.d/*.sh --> ~/.bash_profile --> ~/.bashrc --> /etc/bashrc

- 非交互式登录：

  (1)su UserName

  (2)图形界面下打开的终端

  (3)执行脚本

  执行顺序：

  ~/.bashrc --> /etc/bashrc --> /etc/profile.d/*.sh

Profile类

- 按功能划分，存在两类：

  profiile类和bashrc类

- profile类：为交互式登录的shell提供配置

  全局：/etc/profile，/etc/profile.d/*.sh

  个人：~/.bash_profile

  功用：

  (1) 用于定义环境变量

  (2) 运行命令或脚本

Bashrc类

- bashrc类：为非交互式和交互式登录的shell提供配置

  全局：/etc/bashrc

  个人：~/.bashrc

  功用：

  (1) 定义命令别名和函数

  (2) 定义本地变量

编辑配置文件生效

- 修改profile和bashrc文件后需生效

  两种方法:

  1重新启动shell进程

  2 . 或source

  例:

  . ~/.bashrc

Bash 退出任务

- 保存在~/.bash_logout文件中（用户）
- 在退出登录shell时运行
- 用于
  - 创建自动备份
  - 清除临时文件

练习

- 1、让用户（管理员或所有用户）的PATH环境变量的值多出一个路径，例如：/usr/local/apache2/bin
- 2、用户wang登录时自动启用别名rm='rm -i'
- 3、用户登录时，显示红色字体的警示提醒信息“hi,dangerous！”

关于马哥教育

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

马哥教育
最专业的Linux培训机构

Thank You!