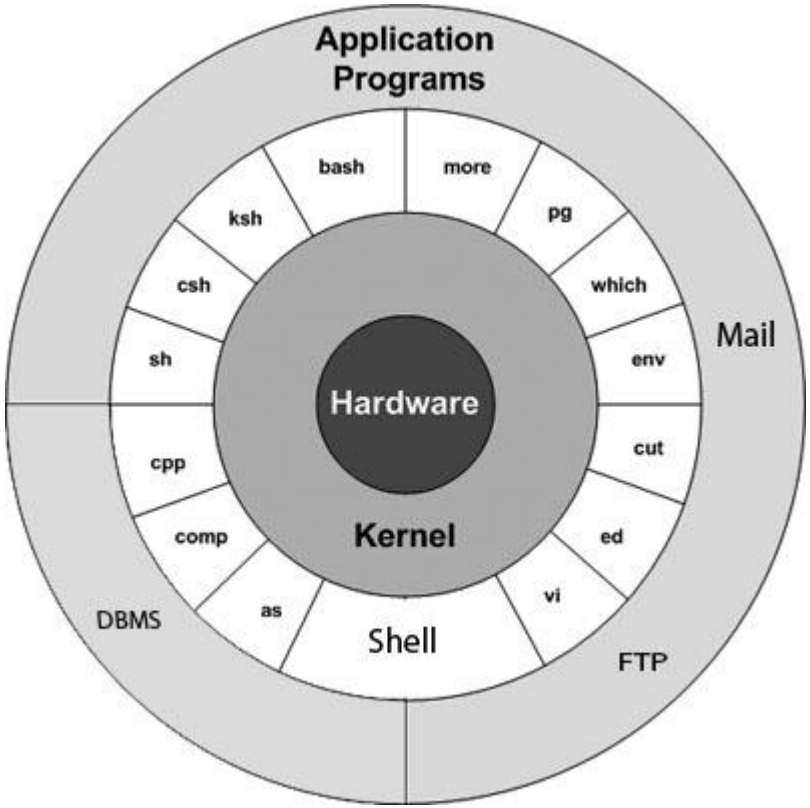
Shell脚本编程基础



生命健康信息学院 解增言

内容

①

- Linux命令帮手
  - 命令行编辑
  - screen工具
- Shell特殊字符（1）
  - 通配符
  - 正则表达式

②

- Shell特殊字符（2）
  - 引号
  - 转义符与路径符
  - 输入输出重定向
  - 注释和后台命令
  - 命令组合符
  - 成组命令

③

- Shell简介
  - Shell的概念
  - Shell的特点
  - Shell版本
  - Shell程序示例
  - Shell程序执行方法
- Shell变量
  - 用户自定义变量
  - 位置变量
  - Shell预定义变量
  - 环境变量

④

- 算术运算
- 控制结构
- 函数
- 作业控制
- Shell内置命令
- Shell脚本调试
- Shell脚本实例

内容

①

- Linux命令帮手
  - 命令行编辑
  - screen工具
- Shell特殊字符（1）
  - 通配符
  - 正则表达式

②

- Shell特殊字符（2）
  - 引号
  - 转义符与路径符
  - 输入输出重定向
  - 注释和后台命令
  - 命令组合符
  - 成组命令

③

- Shell简介
  - Shell的概念
  - Shell的特点
  - Shell版本
  - Shell程序示例
  - Shell程序执行方法
- Shell变量
  - 用户自定义变量
  - 位置变量
  - Shell预定义变量
  - 环境变量

④

- 算术运算
- 控制结构
- 函数
- 作业控制
- Shell内置命令
- Shell脚本调试
- Shell脚本实例

复习

- **用户自定义变量**：变量赋值、变量引用
- **位置变量**：$1、$2、…；命令行参数与位置变量的关系，位置变量赋值（set），位置变量移动（shift）
- **Shell预定义变量**（特殊变量）：$#、$*、…
- **环境变量**：PATH、LOGNAME、…

本节重点及难点

- **重点**：判断、循环
- **难点**：算术运算

- 练习：算术运算、if判断结构、for循环结构

算术运算：整数运算

- **let**
  - i=2; let i=i+1
- **expr**
  - expr 1 + 1　　#运算符两侧需有空格
  - expr 3 * 5　　#*需转义
- **(( ))**
  - ((i=i+1))　　#变量赋值
  - j=$((i*3))　　#返回表达式的值时用$
- **$[ ]**
  - j=$[i*2]　　#返回表达式的值

4种算术表达式比较

算术表达式类型	let	expr	(( ))	$[ ]
计算等式	√		√	
返回表达式的值		√	√	√
运算符两侧有空格		√	√	√
运算符两侧无空格	√		√	√

let、expr、$[ ]：功能有缺陷
let、expr：格式要求严格

算术运算：浮点数运算

```
$ echo “scale=4;10/3” | bc   #管道用法
$ bc                         #交互式用法
```

控制结构

- **顺序执行**：换行或分号
- **判断**：if、case
- **循环**：for、while、until、select

条件测试

- **test（shell内部命令）**

  $ test -d ~/genome

- **[ ]（shell内部命令）**

  $ [$num -gt 2 -a $num -le 5]

  其中使用-a/-o表示逻辑与和逻辑或

- **[[ ]]（Shell关键字）**

  $ [[$color == “Green” || color = “Red”]]

  其中逻辑与/或用&&/||表示，通配符不需加引号，变量前可以不加$

- **命令（执行成功则条件为真）**

  $ echo $1 | grep “-h” >/dev/null　　#$1中包含-h则条件为真

条件测试

- 数值测试运算符
- 字符串测试运算符
- 文件测试运算符
- 逻辑运算符
- 特殊条件测试

数值测试运算符

n1 –eq n2　　n1等于n2时为真（equal）

n1 –ne n2　　n1不等于n2时为真（not equal）

n1 –lt n2 n1小于n2时为真（less than)

n1 –le n2n1小于或等于n2时为真（less than or equal）

n1 –gt n2　　n1大于n2时为真（greater than）

n1 –ge n2　　n1大于或等于n2时为真（greater than or equal）

字符串测试运算符

s1 = s2	字符串s1和s2一样时为真
s1 == s2	同上
s1 != s2	字符串s1和s2不一样时为真
s1 < s2	按字典顺序s1在s2之前时为真
s1 > s2	按字典顺序s1在s2之前时为真
-z s1	字符串s1长度为0时为真
-n s1	字符串s1长度大于0时为真
s1	字符串s1不为空时为真

文件测试运算符：文件类型

-f filename 文件存在且是普通文件则为真

-d filename 文件存在且是目录文件则为真

-L filename 文件存在且是符号链接则为真

-p filename 文件存在且是管道文件则为真

-b filename 文件存在且是块设备文件则为真

-c filename 文件存在且是字符设备文件则为真

-S filename 文件存在且是socket文件则为真

文件测试运算符：文件权限

-r filename	文件存在且用户可读则为真
-w filename	文件存在且用户可写则为真
-x filename	文件存在且用户可执行则为真

文件测试运算符：文件大小

-e filename	文件（包括目录）存在则为真
-s filename	文件存在且大小不为0则为真

文件测试运算符：文件比较

file1 **-nt** file2　　判断文件file1是否比file2更新（根据mtime），或者判断file1存在但file2不存在

file1 **-ot** file2　　判断文件file1是否比file2更旧，或者判断file2存在但file1不存在

file1 **-ef** file2　　判断文件file1和file2是否互为硬链接（？）

特殊条件测试

- **：**，不作任何事情，返回值为0
- **true**，总为真，返回值为0
- **false**，总为假，返回值为1

逻辑运算符

- **!**：逻辑非，如：[! –f hello.pl]
- **-a**：逻辑与，用在[ ]中，如：[-f hello.pl -a -f hello.c]
- **-o**：逻辑或，用在[ ]中，如：[-f hello.pl -o -f hello.sh]
- **&&**：逻辑与，用在[[ ]]中，如：[[-f hello.pl && -f hello.c]]
- **||**：逻辑或，用在[[ ]]中，如：[[-f hello.pl || -f hello.c]]

- **()**：优先运算：如：
  –[$1 -gt 10 -a $2 -gt 20 -o $3 -lt 10]
  –[$1 -gt 10 -a \($2 -gt 20 -o $3 -lt 10 \)]

[ ]与[[ ]]总结

- 表示逻辑与/或：在[ ]内用-a/-o，在[[ ]]内用&&/||
- 整数比较用-gt、-lt…，字符串比较用>、<…
- [ ]内用>和<时需转义（否则是重定向符）
- [ ]是Shell内置命令，[[ ]]是Shell关键字

- **思考：下面的条件表达式的值。**

  [[5 -gt 10]]

  [[5 > 10]]

5 < $a <9 可写成：

[$a -gt 5 -a $a -lt 9] 或 test $a -gt 5 -a $a -lt 9

[[$a -gt 5 && $a -lt 9]]

[[a -gt 5 && a -lt 9]]　　#[[ ]]内的变量可以没有$

下面的语法不会出错，但结果是按照字符串比较

[[$a > 5]]

[$a \> 5]

if语句

```
if 条件
then
        命令
elif 条件
then
        命令
else
        命令
fi
```

if语句示例

```
if test -f /home/pub/seq/at.fa
then
        echo “File exists!”
else
        echo “File does not exist!”
fi
```

case语句

```
case 变量名 in
  模式字符串1) 命令;;
  模式字符串2) 命令;;
  ……
  *) 命令;;
esac
```

case示例

```
#!/bin/bash
week=$1
case $week in
    Saturday|Sunday) echo “Weekend”;;
    Monday|Tuesday|Wednesday|Thursday|Friday)
        echo “Work day”;;
    *) echo “Unknown date”;;
esac
```

for语句：值表形式

- 一般格式：for 变量 in 值表;do 命令表;done
- 如：

```
for i in "${person[@]}"                    #数组
do
        echo $i
done
```

或

```
for i in rose lily "Chinese rose"                    #列表
for i in {a..z}                                      #大括号扩展
for i in `seq 1 10`                        #命令扩展
```

for语句：算术表达式形式

- 一般格式：for ((e1;e2;e3));do 命令表;done
- 如：

```
for ((i=1;i<5;i++))          #注意是两对小括号
do
    echo $i
done
```

while语句

- 一般形式：

  while 测试条件

  do

  　　命令表

  done

until语句

- 一般形式：

  until 测试条件

  do

  　　命令表

  done

select语句

- select语句通常用于菜单设计，其一般形式：

  select identifier [in word…]

  do

  　　命令表

  done

select语句示例

```
$ cat select.sh
#!/bin/bash
echo "What is your favourite fruit?"
select fruit in Apple Banana Orange Pineapple
do
        break
done
echo "Your favourite fruit is $fruit!"
```

```
$ ./select.sh
What is your favourite fruit?
1) Apple
2) Banana
3) Orange
4) Pineapple
#? 2
Your favourite fruit is Banana!
```

跳出循环

- **continue**：跳过本次循环，执行下一次循环
- **break**：退出循环体
- **exit**：退出脚本

shell函数

- **格式：**

```
[function] 函数名 ()
{
        命令表
}
```

- **注意事项：**
  - 函数需先定义，后引用
  - 函数返回值可用return n定义，否则为函数最后一个命令的返回值
  - Shell函数定义时不需要指定参数，可直接通过位置变量调用参数

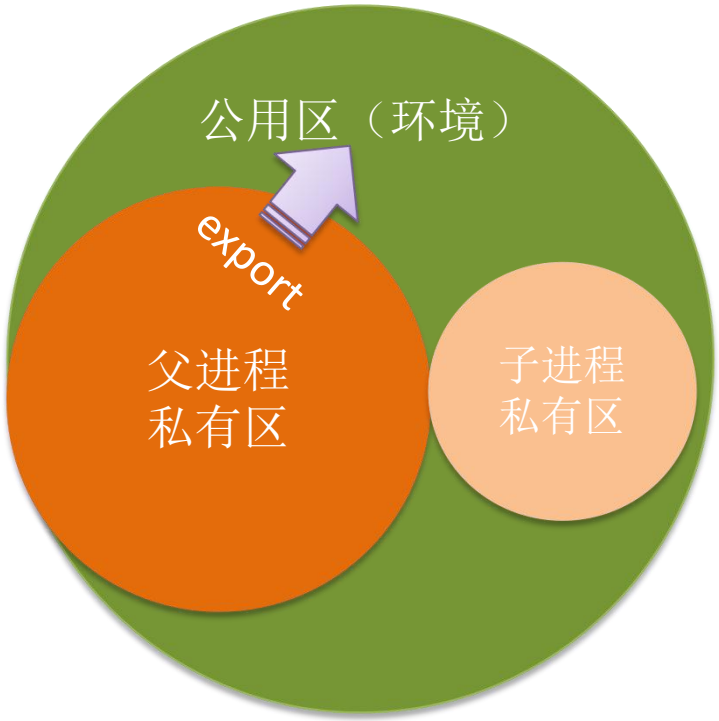
shell函数示例

```
function sum ()
{
    local total=0          #Shell变量默认为全局变量，local定义为局部变量
    while [ $1 ]
    do
        total=$((total+$1))
        shift
    done
    echo $total
}
```

进程、父进程与子进程

- **进程**，就是程序的一次执行过程
- 由一个进程创建的另一个进程，前者叫**父进程**，后者是前者的**子进程**
- 会**产生子进程的几种情况**：
  - &，提交后台作业
  - 管道，管道里的每一个命令在一个单独的子进程里执行
  - 圆括号命令列表( )
  - 执行外部命令或程序

私有变量和公用变量



作业控制

- **Ctrl + z** 暂停正在运行的命令
- **fg** 将后台程序放到前台
- **bg** 将暂停的程序放到后台继续执行
- **&** 放在命令尾部，后台执行
- **jobs** 查看当前在后台运行的命令
- **kill** 终止后台进程
- **Ctrl + c** 终止前台进程

shell内置命令

- **内置命令**：内置在shell代码中，执行时无需到磁盘上定位，速度快，如cd

- **外部命令**：存放在磁盘上，如/bin、/usr/bin目录下，执行时需从PATH定义的路径中查找，如date

shell内置命令

- : .
- alias
- break
- bg
- bind
- builtin
- cd
- continue
- declare
- dirs
- disown
- echo
- anable
- eval
- exec
- exit
- export
- fc
- fg
- getopts
- hash
- help
- history
- jobs
- kill
- let
- local
- logout
- popd
- pushd
- pwd
- read
- readonly
- return
- set
- shift
- stop
- suspend
- test
- times
- trap
- type
- typeset
- ulimit
- umask
- unalias
- unset
- wait

eval命令

- 强制shell对命令行进行两次扫描，执行变量或命令替换，如：

```
a=protein
b="$"a
echo $b                 #$a
eval echo $b            #protein
eval echo "$"b          #$a
```

exec、source和.命令

- 三个命令都不会启动新的进程
- exec用要被执行命令替换当前的shell进程，并且将老进程的环境清理掉，如：

  exec ls

  执行ls后，shell将退出，因此exec通常放在脚本里面使用，而不是直接在命令中用

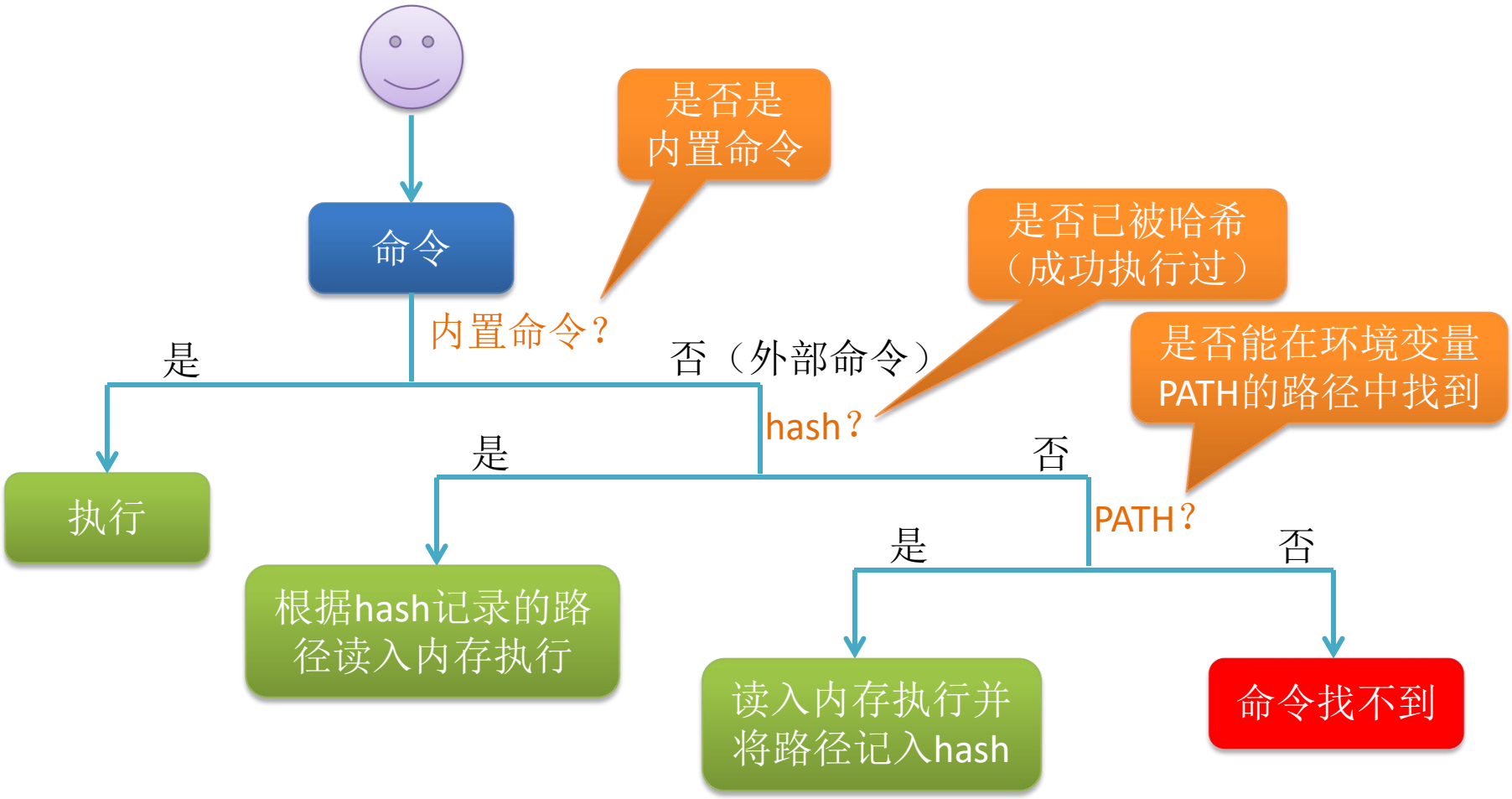
- source命令或者.命令，不会为脚本新建shell，而只是将脚本包含的命令在当前shell执行，执行完返回当前shell如：

  source test.sh

hash命令

- Linux为了提高在PATH路径中查询命令的速度，采用了命令哈希表来记录成功执行过的外部命令。当shell在PATH中找到一个命令时，将这个命令及其路径存入哈希表。用户输入一个外部命令时，shell将先在哈希表中进行查询，哈希表中不存在时再去PATH指定的路径中寻找。
- hash命令列出当前登录shell记录的使用过的外部命令的路径及相关调用次数等信息。加-r选项为清空命令哈希表。

总结：Shell命令执行过程



unset命令

- 删除已定义的变量，如：

```
$a=protein
$echo $a
  protein
$unset a        #变量名前不能有$
$echo $a
```

readonly命令

•设置变量为只读，后面不能再用赋值语句改变该变量的值，防止一个变量被修改。如：

```
$a=protein
$readonly a        #变量名前不能有$，参考export与unset
$a=CDS
  -bash: a: 只读变量
$unset a
  -bash: unset: a: 无法反设定: 只读 variable
```

type命令

- 显示一个命令的类型或如何被解释的，如：

```
$type cd
  -cd 是 shell 内嵌
$type ls
  -ls 是 `ls --color=auto' 的别名
$type man
  -man 已被哈希 (/usr/bin/man)
```

wait命令

- 等待一个进程结束，如：

```
$wait 3702      #等待PID为3702的进程结束
$wait           #等待当前shell所有活动的进程结束
```

shell脚本调试

- 注释掉部分代码
- 利用echo、tee等输出关键信息，跟踪变量的值
- 使用trap命令捕捉信号
- 利用shell脚本调试选项：-c，-n，-x，-v
- 使用bash专用调试器，如bashdb

shell脚本调试

- 注释掉部分代码
- 利用echo、tee等输出关键信息，跟踪变量的值
- 使用trap命令捕捉信号
- 利用shell脚本调试选项：-c，-n，-x，-v
- 使用bash专用调试器，如bashdb

trap命令

- trap命令用于指定在接收到信号后要执行的命令。trap命令的一种常见用途是在脚本程序被中断时完成清理工作。如：

  trap “echo Terminated; rm tmp_files” SIGINT

- 常用信号（系统产生的信号）：

SIGHUP(1)	挂起，通常因终端掉线或用户退出而引发
SIGINT(2)	中断，通常因按下Ctrl+C组合键而引发
SIGQUIT(3)	退出，通常因按下Ctrl+\组合键而引发
SIGABRT(6)	中止，通常因某些严重的执行错误而引发
SIGALRM(14)	报警，通常用来处理超时
SIGTERM(15)	终止，通常在系统关机时发送

- Kill –l可以列出所有的信号

trap命令

- 伪信号（shell产生的信号）
  - –EXIT　从一个函数中退出或整个脚本执行完毕
  - –ERR　当一条命令返回非零值时(代表命令执行不成功)
  - –DEBUG脚本中每一条命令执行之前

shell脚本调试选项

- **-c**：使Shell解析器从字符串而非文件中读取并执行命令

```
$sh -c 'a=1;b=2;let c=$a+$b;echo "c=$c"'
```

- **-n**：读一遍脚本中的命令但不执行，用于检查脚本中的语法错误

- **-v**：执行脚本的同时将执行过的脚本命令打印到标准错误输出（变量替换前的命令）

- **-x**：提供跟踪执行信息，将执行的每一条命令和结果依次打印出来（变量替换后的命令）

使用shell脚本调试选项的方法

- 脚本运行时在命令行中设定调试选项

```
$sh -n script.sh
```

- 脚本开头提供调试选项

```
#!/bin/bash -v
```

- 脚本中用set命令启用或禁用调试选项

```
#! /bin/sh
if [ -z "$1" ]
then
        set -x
        echo "ERROR: Insufficient Args. "
        exit 1
        set +x
fi
```

总结：Shell中的括号—小括号

- 单小括号()
  - –命令组（新建一个子Shell）：(ls; pwd) |sort
  - –命令替换：current_path=$(pwd)
  - –数组赋值（列表）：week=(Mon Tue Wed)
- 双小括号(())
  - –整数运算：((j=i+1))
  - –循环：for ((i=1;i<5;i++))

总结：Shell中的括号—中括号

- 单中括号[ ]
  - –条件测试（Bash的内部命令=test）： if [$# -eq 0]
  - –通配符，表示字符组或范围：[aCe]、[0-5]
  - –数组下标：echo ${week[0]}
  - –整数运算：$[2*3]
- 双中括号[[ ]]
  - –Bash程序语言的关键字，用于条件表达式：如

  if [[$1 == “-h” || $1 == “--help”]]

总结：Shell中的括号—大括号

- 变量名

–普通变量或数组：${num}_chr, ${week[*]}

–置换变量：

${var:-string}, ${var:+string}, ${var:=string}, ${var:?string}

–模式匹配替换：

${var%pattern}, ${var%%pattern}, ${var#pattern},

${var##pattern}

–字符串提取和替换： ${var:num}, ${var:num1:num2},

${var/pattern/pattern}, ${var//pattern/pattern}

总结：Shell中的括号—大括号（续）

- 命令组（也叫内部组，里面的命令在当前Shell运行，不新建子Shell）

  $ { cat rna_stat; paste rna_type rna_num; } |sort

- 大括号扩展

  $ touch {a,b}_{1,2,3}.txt

- 函数：

```
function sum()
{
        ...
}
```

Shell关键字总结（1）

- if：用作条件判断，与fi一起使用
- elif：用作条件判断
- then：用作条件判断，与if、elif一起使用
- else：用作条件判断
- fi：用作条件判断，与if一起使用
- case：用作条件判断，与esac一起使用
- esac：用作条件判断，与case一起使用
- [[：用在条件表达式中，与]]一起使用
-]]：用在条件表达式中，与[[一起使用

Shell关键字总结（2）

- for：用在循环语句中
- while：用在循环语句中
- until：用在循环语句中
- select：循环的一种，可用来实现选择菜单
- do：用在shell循环中，与done一起使用
- done：用在shell循环中，与do一起使用
- !：表示否定、反义
- function：用于定义函数
- time：用于计算命令的运行时间

课后作业9

（1）在当前目录编写shell脚本max.sh，求所有命令行参数中的最大值，如./max.sh 3 9 5的结果为9。（注意：应该允许任意个命令行参数，即能求任意个数中的最大值）

（2）在当前目录下编辑shell脚本factorial.sh，计算10的阶乘。

（3）在当前目录下编写shell脚本day.sh，输出当天是星期几。（如：星期三，或Wednesday）

（4）在当前目录下编写shell脚本rev.sh，实现逆序输出字符串，字符串利用命令行参数读入。例如：sh rev.sh hello的结果是olleh。


CHONGQING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
重慶郵電大學
1950

The End