链滴

1-SHELL 脚本编程基础

作者：Carey

原文链接：https://ld246.com/article/1609146351976

来源网站：链滴

shell脚本编程基础

- **内容概述**
- **编程基础**
- **脚本基本格式**
- **变量**
- **运算**
- **条件测试**
- **配置用户环境**
- **循环**

1 编程基础

Linus：Talk is cheap，show me the code

1.1 程序组成

- 程序：算法+数据结构
- 数据：是程序的核心
- 数据结构：数据在计算机中的类型和组织方式
- 算法：处理数据的方式

1.2 程序编程风格

面向过程语言

- 做一件事，排出个步骤，第一步干什么，第二步干什么，如果出现情况A，做什么处理，如果出现况B，做什么处理
- 问题规模小，可以步骤化，按部就班处理
- 以指令为中心，数据服务于指令
- C，shell

面向对象语言

- 一种认识世界、分析世界的方法论。将万事万物抽象为各种对象
- 类是抽象的概念，是万事万物的抽象，是一类事物的共同特征的集合
- 对象是类的具象，是一个实体
- 问题规模大，复杂系统
- 以数据为中心，指令服务于数据
- java，C#，python，golang等

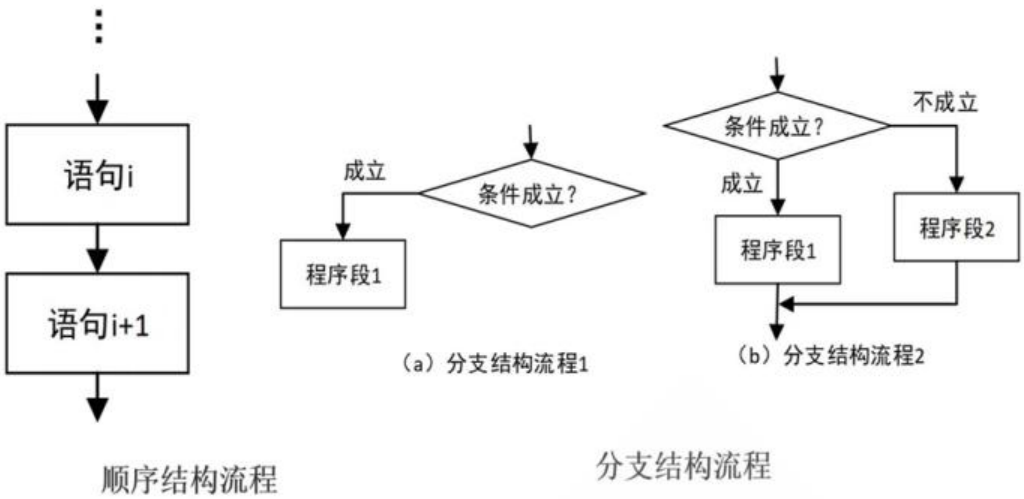
1.3 编程语言

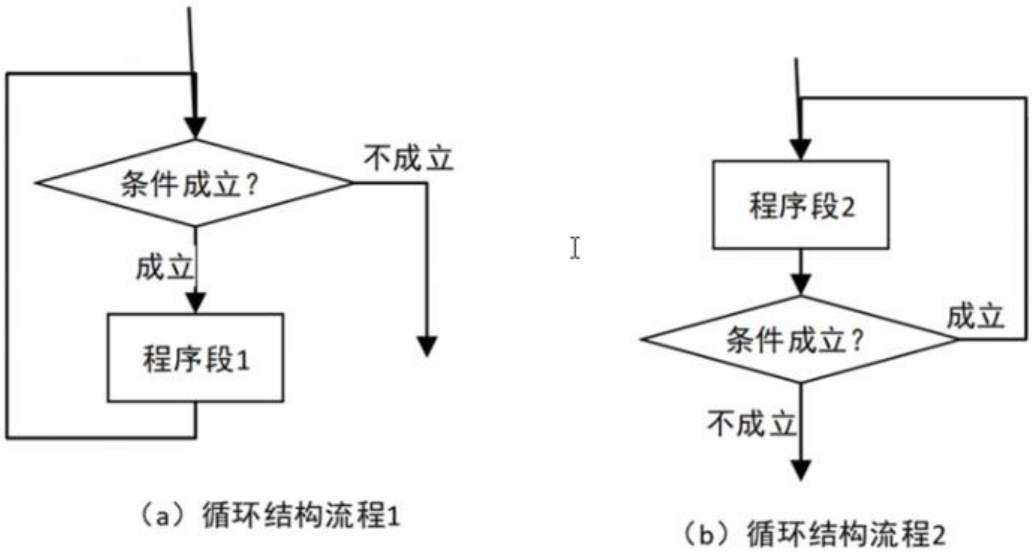
计算机：运行二进制指令

编程语言：人与计算机之间交互的语言。分为俩种：低级语言和高级语言

- 低级编程语言：
  - 机器：二进制的0和1的序列，称为机器指令。与自然语言差异太大，难懂、难写
  - 汇编：用一些助记符号替代机器指令，称为汇编语言
- 高级编程语言：
  - 编译：高级语言——编译器——及其代码文件——执行，如C，C++
  - 解释：高级语言——执行——解释器——机器代码，如：shell，python....

1.4 编程处理方式





（a）循环结构流程1　　（b）循环结构流程2

循环结构流程

三种处理逻辑

- **顺序执行：程序按从上到下顺序执行**
- **选择执行：程序执行过程中，根据条件的不同，进行选择不同分支继续执行**
- **循环执行：程序执行过程中需要重复执行多次某段语句**

2 shell 脚本语言的基本用法

2.1 shell 脚本的用途

- 将简单的命令组合完成复杂的工作，自动化执行命令，提高工作效率
- 减少手工命令的重复输入，一定程度上避免人为错误
- 将软件或应用的安装及配置实现标准化
- 用于实现日常性的，重复性的运维工作，如：文件打包压缩备份，监控系统运行状态并实现告警等

2.2 shell 脚本基本结构

shell 脚本编程：是基于过程式、解释执行的语言

编程语言的基本结构：

- 各种系统命令的组合
- 数据存储：变量、数组
- 表达式：a+b
- 控制语句：if

shell脚本：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/oython
#!/usr/bin/perl
```

2.3 shell脚本创建过程

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：#!

示例：

```
#!/bin/bash
```

添加注释，注释以#开头

第二步：加执行权限

给予执行权限，在命令行上指定脚本额绝对路径或相对路径

第三步：运行脚本

直接运行解释器，将脚本作为解释器程序的参数运行

2.4 shell脚本注释规范

1. 第一行一般为调用使用的语言
2. 程序名，避免更改文件名为无法找到正确的文件
3. 版本号
4. 更改后的时间
5. 作者相关信息
6. 该程序的作用，及注意事项
7. 最后是各版本的更新简要说明

2.5 第一个脚本

范例：远程主机执行脚本

```
[14:49:02 root@centos8 ~]#curl http://www.wangxiaochun.com/testdir/hello.sh | bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
Dload  Upload   Total   Spent    Left  Speed
100   388  100   388    0     0   3104      0 --:--:-- --:--:-- --:--:--  3104
hello, world
Hello, world!

[14:49:17 root@centos8 ~]#curl -s http://www.wangxiaochun.com/testdir/hello.sh | bash
hello, world
Hello, world!
```

范例：第一个Shell脚本hello world

```
[11:51:07 root@centos8 data]#cat hello.sh
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-05
#FileName：hello.sh
#URL: http://www.magedu.com
#Description：The test script
#Copyright (C): 2020 All rights reserved
#**********************************************************************
set -u
set -e
echo "hello,world"
[11:50:56 root@centos8 data]#chmod +x hello.sh
[11:51:03 root@centos8 data]#./hello.sh
hello,world
```

范例：备份脚本

```
[11:55:49 root@centos8 data]#cat backup-etc.sh
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-05
#FileName：backup-etc.sh
#URL: http://www.magedu.com
#Description：The test script
#Copyright (C): 2020 All rights reserved
#**********************************************************************
set -u
set -e
echo -e "\e[1;31mStarting backup...\e[1;0m"
cp -av /etc/ /data/etc`date +%F`/
echo -e "\e[1;31mStarting is finished\e[1;0m"
```

2.6 shell脚本调试

只检测脚本中的语法错误，但无法检查出命令错误，但不真正执行脚本

```
bash -n /path/to/spme_script
```

调试并执行

```
bash -x /path/to/some_script
```

范例：

```
[14:13:52 root@centos8 data]#bash -n backup-etc.sh
```

总结：脚本错误常见的有三种

- 语法错误，会导致后续的命令不继续执行，可以用bash -n 检查错误，提示的出错行数不一定是准的
- 命令错误，默认后续的命令还会继续执行，用bash -n无法检查出来，可以使用bash -x 进行观察
- 逻辑错误：只能使用bash -x进行观察

2.7 变量

2.7.1 变量

变量表示命名的内存空间，将数据放在内存空间中，通过变量名引用，获取数据

2.7.2 变量类型

变量类型：

- 内置变量，如：PS1，PATH,UID,HOSTNAME, $,BASGPID,PPID,?,HISTSIZE
- 用户自定义变量

不同的变量存在的数据不同，决定了以下

- 数据存储方式
- 参与的运算
- 表示的数据范围

变量数据类型：

- 字符
- 数值：整型、浮点型、bash不支持浮点数

2.7.3 编程语言分类

静态和动态语言

- 静态编译语言：使用变量前，先声明变量类型，之后类型不能改变，在编译时检查，如：java,c
- 动态编译语言：不用事先声明，可随时改变类型，如：bash，Python

强类型和弱类型语言

- 强类型语言：不同类型数据操作，必须经过强制转换才同一类型才能运算，如java，c#，python
- 弱类型语言：语言的运行时会隐式做数据类型转换。无需指定类型，默认均为字符型；参与运算会动进行隐式类型转换；变量无需事先定义可直接调用

2.7.4 Shell中变量命名法则

- 不能使程序中的保留字和内置变量：如if，for

- 只能使用数字、字母及下划线，且不能以数字开头，注意：不支持短横线“-”，和主机名相反
- 见名知义，用英文单词命名，并体现出实际作用，不要用简写，如：ATM
- 同一命名规则：驼峰命名法，studentname，大驼峰StudentName小驼峰studentName
- 变量名大写：STUDENT_NAME
- 局部变量小写
- 函数名小写

2.7.5 变量定义和引用

变量的生效范围等标准划分变量类型

- 普通变量：生效范围为当前shell进程；对当前shell之外的其他shell进程，包括当前shell的子shell程均无效
- 环境变量：生效范围为当前shell进程及其子进程
- 本地变量：生效范围为当前shell进程中某代码片段，通常指函数

变量赋值：

```
name='value'
```

其他赋值情况加双引号""

```
[16:14:58 root@centos8 script]#NAME="

> sss
> sss
> sss
> sss
> sss
> "
> [16:33:24 root@centos8 script]#echo $NAME
sss sss sss sss sss
[16:33:32 root@centos8 script]#echo "$NAME"

sss
sss
sss
sss
sss
```

value 可以是一下多种形式

```
直接字串：name='root'
变量引用：name='$USER'
命令引用：name=`COMMAND` 或者 name=$(COMMAND)
```

注意赋值是临时生效，当退出终端后，变量会自动删除，无法持久保持，脚本中的变量会随着脚本结，也会自动删除

范例：变量追加赋值

```
[14:47:53 root@centos8 data]#zhang=user
[14:48:04 root@centos8 data]#echo $zhang
user
[14:48:16 root@centos8 data]#zhang+=:zhangzhuo
[14:48:26 root@centos8 data]#echo $zhang
user:zhangzhuo
```

范例：利用变量实现动态命令

```
[14:48:28 root@centos8 data]#CMD=hostname
[14:50:21 root@centos8 data]#$CMD
centos8
```

变量引用：

```
$name
${name}
```

弱引用和强引用

- "$name" 弱引用，其中的变量会被替换为变量值
- '$name' 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>
```

范例：显示系统信息

```
[15:19:56 root@centos8 data]#cat systeminfo.sh
#!/bin/bash
#
#****************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-05
#FileName：systeminfo.sh
#URL: http://www.magedu.com
#Description：The test script
#Copyright (C): 2020 All rights reserved
#****************************************************************
set -u
set -e
RED="\e[1;31m"
GREEN="\e[;32m"
END="\e[0m"
echo -e "$GREEN----------------------------------------------------------------$END"
echo -e "HOSTNAME:       $RED`hostname`$END"
echo -e "IPADDR:         $RED`ifconfig ens33 | grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}' | head -n1`$E
D"
```

```
echo -e "OSVERSION:       $RED`cat /etc/redhat-release`$END"
echo -e "KERNEL:          $RED`uname -r`$END"
echo -e "CPU:             $RED`lscpu | grep 'Model name'| tr -s ' ' | cut -d: -f2`$END "
echo -e "MEMORY:          $RED`free -h | grep Mem|tr -s ' ' : | cut -d: -f2`$END"
echo -e "DISK:            $RED`lsblk|grep '^sd'|tr -s ' ' | cut -d ' ' -f4`$END"
echo -e "$GREEN----------------------------------------------------------------------$END"
```

范例：备份etc

```
[15:27:19 root@centos8 data]#cat backup-etc.sh
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-05
#FileName：backup-etc.sh
#URL: http://zhangzhuo.ltd
#Description：The test script
#Copyright (C): 2020 All rights reserved
#**********************************************************************
set -u
set -e
COLOR='echo -e \e[1;31m'
END='\e[0m'
BACKUP=/data
SRC=/etc
DATE=`date +%F`

${COLOR}mStarting backup...$END
cp -av $SRC ${BACKUP}${SRC}_$DATE
${COLOR}mStarting is finished$END"
```

2.7.6 环境变量

环境变量：

- 可以使子进程（包括孙子进程）继承父进程的变量，但是无法让父进程使用子进程的变量
- 一旦子进程修改从父进程继承的变量，将会新的值传递给孙子进程
- 一般只在系统配置文件中使用，在脚本中较少使用

变量声明和赋值：

```
声明并赋值
export name=VALUE
declare -x name=VALUE
或者俩者分俩步实现
name=value
export name
```

变量引用

```
$name
```

${name}

显示所有环境变量：

```
env
printenv
export
declare -x
```

删除变量：

```
unset name
```

bash内建的环境变量

```
PATH
SHELL
USER
UID
HOME
PWD
SHLVL    shell的嵌套层数，即深度
LANG
MAIL
HOSTNAME
HISTSIZE
_      下划线，表示前一个命令的最后一个参数
```

2.7.7 只读变量

只读变量：只能声明定义，但后续不能修改和删除，即常量

声明只读变量：

```
readonly name
declare -r name
```

查看只读变量：

```
readonly [-p]
declare -r
```

2.7.8 位置变量

位置变量：在bash shell中内置的变量，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个、第2个等参数，shift [n]换位置
$0       命令本身，也包扣路径
$*       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
```

注意:$@ $*只在被双引号包起来的时候才会有差异

清空所有位置变量

```
set --
```

范例：

```
[15:50:27 root@centos8 data]#cat arg.sh
#!/bin/bash
#
#****************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-05
#FileName：  arg.sh
#URL: http://www.magedu.com
#Description：  The test script
#Copyright (C): 2020 All rights reserved
#****************************************************************
set -u
set -e
echo "1st arg is $1"
echo "1st arg is $2"
echo "1st arg is $3"
echo "1st arg is ${10}"
echo "1st arg is ${11}"

echo "the number of arg is $#"
echo "all args are $*"
echo "all args are $@"
echo "the scriptname is `basename $0`"
[15:50:44 root@centos8 data]#./arg.sh {a..z}
1st arg is a
1st arg is b
1st arg is c
1st arg is j
1st arg is k
the number of arg is 26
all args are a b c d e f g h i j k l m n o p q r s t u v w x y z
all args are a b c d e f g h i j k l m n o p q r s t u v w x y z
the scriptname is arg.sh
```

范例：mv替换rm命令

```
#!/bin/bash
#
#****************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-08
#FileName：  rm.sh
#URL: https://www.zhangzhuo.ltd
#Description：  The test script
#Copyright (C): 2020 All rights reserved
#****************************************************************
set -u
```

```
set -e
COLOR="echo -e \033[1;31m"
END="\033[0m"
BACKUP=$1
SRC=/data/backup/
DATE=`date +%F_%T`

mkdir ${SRC}${DATE} >/dev/null
mv ${BACKUP} ${SRC}${DATE}
${COLOR}${BACKUP} to ${SRC}${DATE}
```

2.7.9 退出状态码变量

进程执行后，将使用变量?保存状态码的相关数字，不同的值反应成功或失败，?取值范围0-255

```
$?的值为0       代表成功
$?的值是1到255  代表失败
```

范例：

```
[15:56:52 root@centos8 data]#ping -c1 -w1 baidu.com &>/dev/null
[15:57:03 root@centos8 data]#echo $?
0
```

用户可以在脚本中使用以下命令自定义退出状态码

```
exit [n]
```

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
- 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

2.7.10 展开命令行

展开命令执行顺序

```
把命令行分成单个命令词
展开别名
展开大括号的声明{}
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
```

加引号来防止扩展

单引号('')防止所有扩展
双引号("")也可防止扩展，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

2.7.11 脚本安全和set

set命令：可以用来定制shell环境

$-变量

h:hashall,打开选项后，shell会将命令所在的路径hash下来，避免每次都要查询。通过set +h选项将选项关闭

i:interactive-comments,包含这个选项说明当前的shell是一个交互式的shell。所谓的交互式shell，脚本中，i选项是关闭的

m:monitor,打开监控模式，就可以通过job control来控制进程的停止、继续，后台或者前台执行等

B：braceexpand，大括号扩展

H：history，H选项打开，可以展开命令历史列表中的命令，可以通过！感叹号来完成，例如"!!"返回近的一个历史命令，“!n”返回第n个历史命令

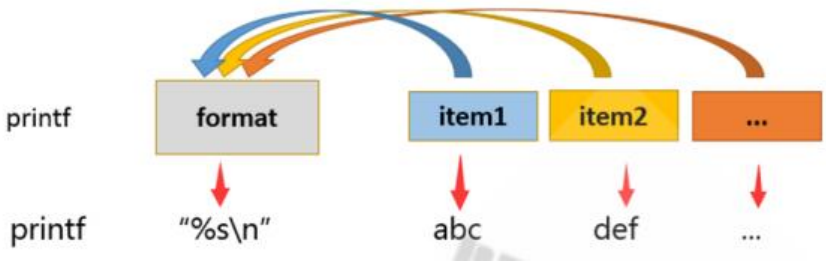
set命令实现脚本安全

- -u 在扩展一个没有设置的变量使，显示错误信息，等同于set -o nounset
- -e 如果一个命令返回一个非0退出状态值（失败）就退出，等同于set -o errexit
- -o option显示，打开或者关闭选项
  - 显示选项：set -o
  - 打开选项：set -o 选项
  - 关闭选项：set +o 选项
- -x 当执行命令时，打印命令及其参数，类似bash -x

2.8 格式化输出 printf

格式

printf "指定的格式" "文本1" "文本2" ...



常用格式替换符

```
替换符              功能
%s                 字符串
%f                浮点格式
%b                 相对应的参数中包含转义字符时，可以使用此替换符进行替换，对应转义字符会被转

%c                 ASCII字符，即显示对应参数的第一个字符
%d,%i               十进制整数
%o                 八进制值
%u                 不带正负号的十进制值
%x                 十六进制值 （a-f）
%X                十六进制值 （A-F）
%%                 表示%本身
```

说明：%s中的数字代表此替换输出字符串宽度，不足补空格，默认是右对齐，%-10s表示10个字符，-表示左对齐

常用转义字符

```
转义符              功能
\a                警告字符，通常为ASCII的BEL字符
\b                 后退
\f                换页
\n                 换行
\r                回车
\t                水平制表符
\v                 垂直制表符
\                 表示\本身
```

范例：

```
[19:09:13 root@centos8 ~]#printf "%s\n" 1 2 3 4
1
2
3
4
[19:10:11 root@centos8 ~]#printf "%f\n" 1 2 3 4
1.000000
2.000000
3.000000
4.000000
#.2f 表示保留俩位小数
[19:10:20 root@centos8 ~]#printf "%.2f\n" 1 2 3 4
1.00
2.00
3.00
4.00
[19:21:58 root@centos8 ~]#printf "(%s)" 1 2 3 4 ;echo
(1)(2)(3)(4)
[19:22:27 root@centos8 ~]#printf " (%s) " 1 2 3 4 ;echo " "
(1)  (2)  (3)  (4)
[19:23:15 root@centos8 ~]#printf "(%s)\n" 1 2 3 4
(1)
(2)
```

```
(3)
(4)
[19:23:20 root@centos8 ~]#printf "%s %s\n" 1 2 3 4
1 2
3 4
[19:23:42 root@centos8 ~]#printf "%s %s %s\n" 1 2 3 4
1 2 3
4

#%-10s表示宽度10个字符，左对齐
[19:25:31 root@centos8 ~]#printf "%-10s %-10s %-4s %s\n" 姓名 性别 年龄 体重 小明 男 20 70
小红 女 18 50
姓名      性别      年龄 体重
小明      男        20   70
小红      女        18   50

#将十进制转换16进制数
[19:28:06 root@centos8 ~]#printf "%x" 31; echo
1f

#将16进制C转换成十进制，16进制必须以0x开头
[19:31:39 root@centos8 ~]#printf "%d\n" 0x1f
31

[19:31:44 root@centos8 ~]#VAR="welcome to magedu" ; printf "\033[31m%s\033[0m\n" $V
R
welcome
to
magedu
[19:33:40 root@centos8 ~]#VAR="welcome to magedu" ; printf "\033[31m%s\033[0m\n" "$V
R"
welcome to magedu
```

2.9 算术运算

Shell允许在某些情况下对算术表达式进行求值，比如：let和declare内置命令，(())复合命令和算术扩。求值以固定宽度整数进行，不检查溢出，尽管除以0被困并标记为错误。运算符及其优先级，管联和值与C语言相同。以下运算符列表分组为等级优先运算级别。**级别按降序排列优先。**

```
id++ id--              vriable post-increment and post-decrement
++id --id              vriable post-increment and post-decrement
- + unary minus and plus
! ~                   logical and bitwise negation
**                    exponentiation 乘方
* / %                  multiplication,division,remainder，%表示取模，即取余数，示例：9%4=1
+ -addition，subtraction
<< >>                   left and right bitwise shifts
<= >= < >                 comparison
== !=                 equality and inequality
&                     bitwise AND
^                     bitwise exclusive OR
|                    bitwise OR
&&                      logical AND
||                    logical OR
```

```
expr?expr:expr          conditional operator
= *= /= %= += -= <<= >>= &= ^= |=     assignment
expr1,expr2             comma``
```

乘法符号有些场景中需要转义

实现算术运算：

(1) let var=算术表达式
(2) ((var=算术表达式))和上面等价
(3) var=$[算术表达式]
(4) var=$((算术表达式))
(5) var=$(expr arg1 arg2 arg3 ...)
(6) declare -i var = 数值
(7) echo '算术表达式' | bc

内建随机数生成器变量：

$RANDOM　取值范围：0-32767

范例：

```
#生成0-49之间随机数
[19:51:09 root@centos8 script]#echo $[$RANDOM%50]
22
#随机字体颜色
[19:52:23 root@centos8 script]#echo -e "\033[1;$[RANDOM%7+31]mhello\033[0m"
hello
```

增强型赋值：

```
+=     i+=10  相当于 i=i+10
-=     i-=j   相当于i=i-j
*=
/=
%=
++ i++,++i    相当于i=i+1
-- i--,--i    相当于i=i-1
```

格式：

```
let varOPERvalue
```

范例：

```
[19:52:24 root@centos8 script]#let i=10*2
[19:58:18 root@centos8 script]#echo $i
20
[19:58:23 root@centos8 script]#((j=i+10))
[19:58:39 root@centos8 script]#echo $j
30
```

自增，自减

范例：鸡兔同笼

```
[20:05:26 root@centos8 script]#chook_rabbit.sh 60 188
RABBIT:34
CHOOK:26
[20:05:30 root@centos8 script]#cat chook_rabbit.sh
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-08
#FileName：chook_rabbit.sh
#URL: https://www.zhangzhuo.ltd
#Description：The test script
#Copyright (C): 2020 All rights reserved
#**********************************************************************
HEAD=$1
FOOT=$2
RABBIT=$(((FOOT-HEAD-HEAD)/2))
CHOOK=$[HEAD-RABBIT]
echo RABBIT:$RABBIT
echo CHOOK:$CHOOK
```

2.10 逻辑运算

true，false

1,真
0,假

与：&：和0相与，结果为0，和1相与，结果保留原值

1 与 1 = 1
1 与 0 = 0
0 与 1 = 0
0 与 0 = 0

或：|：和1相或结果为1，和0相或，结果保留原值

1 或 1 = 1
1 或 0 = 1
0 或 1 = 1
0 或 0 = 0

非：！

! 1 = 0　! true
! 0 = 1　! false

异或：^

异或的俩个值，相同为假，不同为真。俩个数字X，Y异或得到结果Z，Z在和任意俩者之一X异或，将出另一个值Y

1 ^ 1 = 0

1 ^ 0 = 1
0 ^ 1 = 1
0 ^ 0 = 0

范例：

[17:03:12 root@centos8 data]#x=10;y=20;temp=$x;x=$y;y=$temp;echo x=$x,y=$y
x=20,y=10
[17:05:16 root@centos8 data]#x=10;y=20;x=$[x^y];y=$[x^y];x=$[x^y];echo x=$x,y=$y
x=20,y=10

短路运算

- 短路与

CMD1 短路与 CMD2
第一个CMD1结果为真（1），第二个CMD2必须要参与运算，才能得到最终的结果
第一个CMD1结果为假（0），总的结果必定为0，因此不需要执行CMD2

- 短路或

CMD1 短路或 CMD2
第一个结果为真（1），总的结果必定为1，因此不需要执行CMD2
第一个结果为假（0），第二个CMD2必须要参与运算，才能得到最终的结果

2.11 条件测试命令

条件测试：判断某需求是否满足，需要测试机制来实现，专用的测试表达式需要有测试命令辅助完成

测试过程

，实现评估布尔声明，以便用在条件环境下进行执行

若真，则状态码变量$?返回0

若假，则状态码变量$?返回1

条件测试命令

test EXPRESSION

[EXPRESSION] #和test等价，建议使用[ ]

[[EXPRESSION]]

注意：EXPRESSION前后必须有空白字符

帮助：

[17:06:38 root@centos8 data]#type [
[18:43:48 root@centos8 data]#help [
[18:43:53 root@centos8 data]#help test

2.11.1 变量测试

判断 NAME 变量是否定义
[-v NAME]
判断 NAME 变量是否定义并且是名称引用，bash 4.4新特性
[-R NAME]

2.11.2 数值测试

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

算术表达式比较

== 相等
!= 不相等
<=

> =
> <
>

2.11.3 字符串测试

test和 [ ] 用法

test和 [ ] 用法
-z STRING 字符串是否为空，没定义或空为真，不空为假
-n STRING 字符串是否不空，不空为真，空为假
STRING 同上
STRING1 = STRING2 是否等于
STRING1 != STRING2 是否不等于

> ascii码是否大于ascii码
> < 是否小于

[[ ]]用法

[[expression]] 用法
== 左侧字符串是否和右侧的PATTERN相同
注意：此表达式用于[[ ]]中，PATTERN为通配符
=~ 左侧字符串是否能够被右侧的正则表达式的PATTERN所匹配
注意：此表达式用于[[ ]]中：扩展正则表达式

建议：当使用正则表达式或通配符使用[[ ]]，其他情况一般使用 [ ]

范例：使用 [ ]

在比较字符串时，建议变量放在""中
[18:44:09 root@centos8 data]#[-z "$str"]
[18:58:32 root@centos8 data]#echo $?

0

范例:使用 [[ ]]

通配符
[18:58:36 root@centos8 data]#FILE=test.log
[19:00:02 root@centos8 data]#[["$FILE" == *.log]]
[19:00:30 root@centos8 data]#echo $?
0

正则表达式
[19:01:27 root@centos8 data]#FILE=test.txt
[19:01:59 root@centos8 data]#[["$FILE" =~ \.log$]]
[19:02:01 root@centos8 data]#echo $?
1

[[==]] ==右侧的*作为通配符不要加"",只想做*，需要加""或转义
[19:02:03 root@centos8 data]#NAME="linux*"
[19:05:16 root@centos8 data]#[["$NAME" == "linux*"]]
[19:05:18 root@centos8 data]#echo $?
0

2.11.4 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
-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
-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

注意：最终结果有用户对文件的实际权限决定，而非文件属性决定

范例：

[19:12:12 root@centos8 data]#[-d /etc/]
[19:12:41 root@centos8 data]#echo $?
0
[19:12:47 root@centos8 data]#[-d /etc/fstab]
[19:12:52 root@centos8 data]#echo $?

```
1
[19:12:54 root@centos8 data]#[ -w /etc/fstab ]
[19:13:00 root@centos8 data]#echo $?
0
```

文件属性测试

```
-s FILE:          是否存在且非空
-t df           fd文件描述符是否在某终端已经打开
-N FILE:          文件自从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
FILE1 -ef FILE2    FILE1是否是FILE2的硬链接
FILE1 -nt FILE2    FILE1是否新于FILE2（mtime）
FILE1 -ot FILE2    FILE1是否旧于FILE2
```

2.12 关于（）和 { }

(CMD;CMD2;...)和{CMD1;CMD2;...;}都可以将多个命令组合在一起，批量执行

(list)会开启子shell，并且list中变量赋值及内部命令执行后，将不在影响后续的环境

帮助查看：man bash 搜索（list）

{ list; }不会启子shell，在当前shell中运行，会影响当前shell环境

帮助查看：man bash 搜索{ list; }

2.13 组合测试条件

2.13.1 第一种方式 [ ]

```
[ EXPRESSION1 -a EXPRESSION2 ]  并且，EXPRESSION1和EXPRESSION2都是真，结果才为真
[ EXPRESSION1 -o EXPRESSION2 ]  或者，EXPRESSION1和EXPRESSION2只要有一个为真，结
就为真
[ ! EXPRESSION ]              取反
```

说明：-a和-o需要使用测试命令进行，[[ ]]不支持

2.13.2 第二种方式

```
COMMAND1 && COMMAND2  并且，短路与，代表条件性的AND
如果1成功将执行2，否则将不执行2
COMMAND1 || COMMAND2  或者，短路或，代表条件性的OR
如果1成功将不执行2，否则将执行2
！ COMMAND           非，取反
```

2.14 使用read命令来接受输入

使用read来吧输入值分配给一个或多个shell变量，read从标准输入中读取值，给每个单词分配一个量，所有剩余单词都被分配给最后一个变量，如果变量名没有指定，默认标准输入的赋值给系统内置变量REPLY

格式：

```
read [options] [name ...]
```

常见选项：

```
-p        指定要显示的提示
-s        静默输入，一般用于密码
-n N      指定输入的字符长度N
-d '字符'  输入结束符
-t N      TIMEOUT为N秒
```

范例：面试题read和输入重定向

```
[19:51:56 root@centos8 ~]#cat test.txt
1 2
[19:51:59 root@centos8 ~]#read i j < test.txt ; echo i=$i j=$j
i=1 j=2
[19:52:31 root@centos8 ~]#echo 1 2 | read x y ; echo $x $y

[19:52:57 root@centos8 ~]#echo 1 2 | (read x y ; echo $x $y)
1 2
[19:53:42 root@centos8 ~]#echo 1 2 | { read x y ; echo $x $y; }
1 2
```

3 bash shell 的配置文件

bash shell的配置文件很多，可以分为下面类别

全局配置：

```
/etc/profile
/etc/profile.d/*.sh
/etc/bashrc
```

个人配置：

```
~/.bash_profile
~/.bashrc
```

3.2 shell登录的俩种方式分类

3.2.1 交互式登录

- 直接通过终端输入账号密码登录
- 使用su - UserName 切换用户

配置文件生效的执行顺序：

```
放在每个文件最前面
/etc/profile
/etc/profile.d/*.sh
```

/etc/bashrc
~/.bash_profile
~/.bashrc
, etc/bashrc
放在每个文件最后
/etc/profile.d/*.sh
/etc/bashrc
/etc/profile
~/.bashrc
~/.bash_profile

注意：文件之间的调用关系，写在同一个文件的不同位置，将影响文件的执行顺序

3.2.2 非交互式登录

- su UserName
- 图形界面下打开的终端
- 执行脚本
- 任何其他的bash实例

3.3 按功能划分分类

profile类和bashrc类

3.3.1 Profile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

功能：

- 用于定义环境变量
- 运行命令或脚本

3.3.2 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

功能：

- 定义别名和函数
- 定义本地变量

3.4 编辑配置文件生效

修改profile和bashrc文件后需生效俩种方法：

- 重新启动shell
- source|. 配置文件

注意：source会在当前shell中执行脚本，所以一般只用于执行配置文件，火灾脚本中调用另一个脚本场景

3.5 bash 退出任务

保存在~/.bash_logout文件中（用户），在退出登录shell时运行

功能：

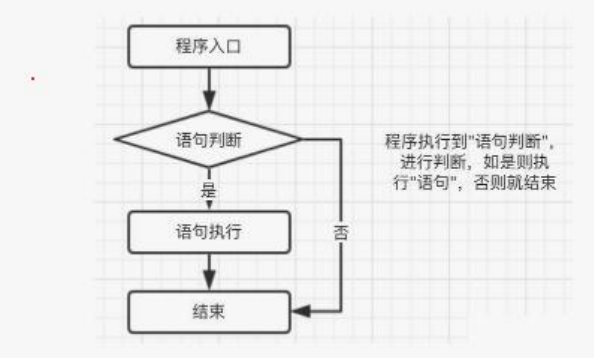
- 创建自动备份
- 清除临时文件

4 流程控制

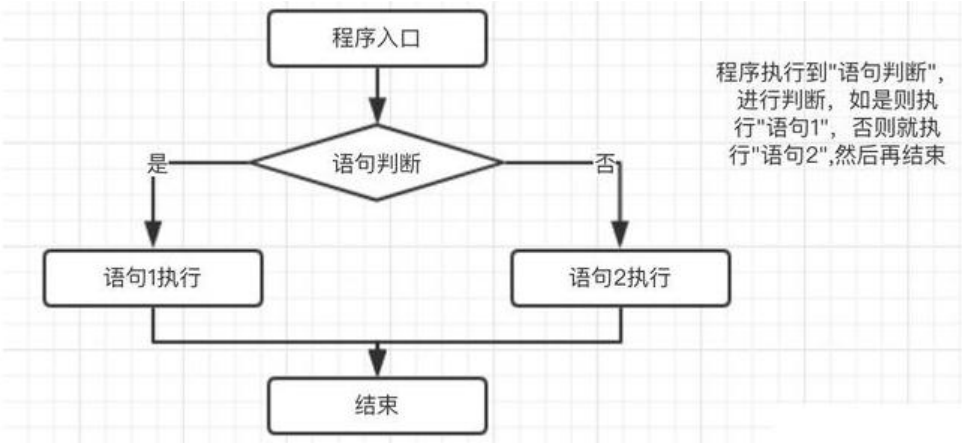
4.1 条件选择

4.1.1 条件判断

4.1.1.1 单分支条件



4.1.1.2 多分支条件



4.1.2 选择执行 if 语句

格式：

if COMMANDS;then COMMANDS; [elif COMMANDS; then COMMANDS;].. [else COMMANS;] fi

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

多分支

if 判断条件1;then
条件1为真的分支代码
elif 判断条件2;then
条件2为真的分支代码
elif 判断条件3;then
条件3为真的分支代码
...
else
以上条件都为假的分支代码
fi

说明：

- 多个条件时，逐个条件进行判断，第一次遇“真”条件时，执行其分支，而后结束整个if语句
- if语句可嵌套

范例：

[12:57:28 root@centos6 ~]#declare -f

4.1.3 条件判断case 语句

格式：

case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;]... esac

case 变量引用 in
PAT1)
分支1
;;

```
PAT2)
分支2
;;
...
*)
默认分支
;;
asac
```

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
   | 或者，如：a|b
```

范例：

范例1

```
read -p "Do you agree(yes/no)?" INPUT
INPUT=`echo $INPUT | tr 'A-Z' 'a-z'`
case $INPUT in
y|yes)
echo "You input is YES"
;;
n|no)
echo "You input is NO"
;;
*)
echo "Input fales,please input yes or no!"
esac
```

范例2

```
read -p "Do you agree(yes/no)?" INPUT
case $INPUT in
[Yy]|[Yy][Ee][sS])
echo "You input is YES"
;;
[nN]|[Nn][Oo])
echo "You input is NO"
;;
*)
echo "Input fales,please input yes or no!"
esac
```

4.2 循环

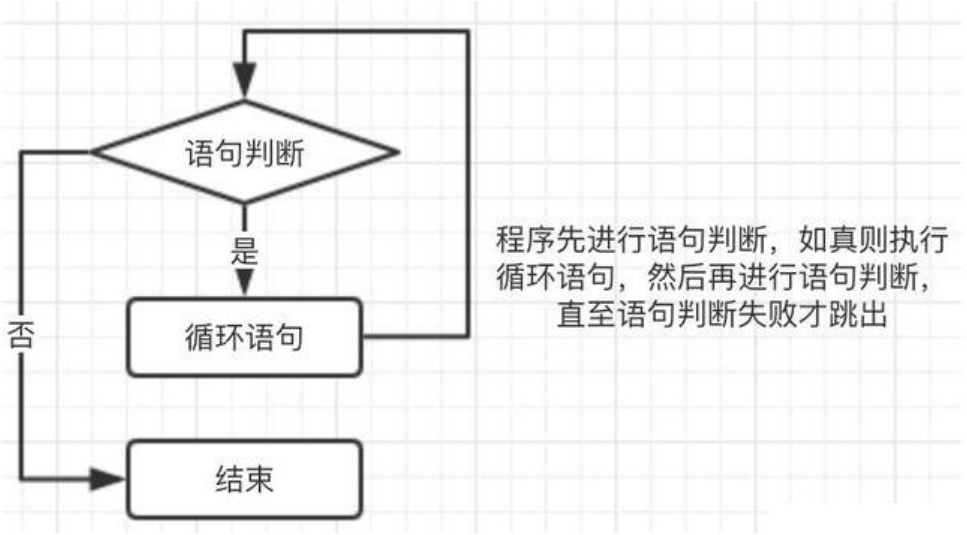
4.2.1 循环执行介绍

将某行代码段重复运行多次，通常有进入循环的条件和退出循环的条件

重复运行次数

- 循环次数事先已知
- 循环次数事先未知

常见的循环命令：for，while，until



4.2.2 循环for

Centos7的for帮助比Centos8全面
[13:15:32 root@centos8 data]#help for

格式1：

for NAME [in WORDS ...] ; do COMMANDS;done
方式1
for 变量名 in 列表;do
循环体
done
方式2
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，循环束
- 如果省略[in WORDS ...],此时使用位置参量

for循环列表生成方式：

- 直接给出列表
- 整数列表

{start..end}
$(seq [start [step]] end)

- 返回列表的命令

$(COMMAND)

- 使用glob，如*.sh
- 变量引用，如 @,*,$#

范例：面试题，计算1+2+3...+100的结果

```
[13:15:53 root@centos8 data]#sum=0;for i in {1..100};do let sum+=i;done ;echo sum=$sum
sum=5050
[13:25:43 root@centos8 data]#seq -s+ 100|bc
5050
```

范例：

```
sum=0
for i in $* ; do
let sum+=i
done
echo sum=$sum
[13:31:26 root@centos8 data]#./for_sum.sh {1..100}
sum=5050
```

生产案例：将指定目录下的文件所有文件的后缀改名为bak后缀

```
for FILE in /data/zhang/* ; do
PRE=`echo $FILE | sed -nr 's/(.*)\.([^.]+)$/\1/p'`
mv -v $FILE   $PRE.bak
done
```

范例：九九乘法表

```
for i in {1..9};do
for j in `seq $i`;do
echo -e "${j}*${i}=$[i*j]\t\c"
done
echo
done
```

范例：面试题，要求将目录YYY-MM-DD/中所有文件，移动到YYYY-MM/DD/下

```
1.创建YYYY-MM-DD格式的目录，当前日期一年前365天到目前共365个目录，里面有10个文件.log
缀的文件
for i in {1..365};do
mkdir /data/mkdir/`date -d "-${i} day" +%F`
cd /data/mkdir/`date -d "-${i} day" +%F`
for j in {1..10};do
touch zz${j}.log
done
done
2.将上面的目录移动到YYYY-MM/DD/下
DIR=/data/mkdir
cd $DIR
```

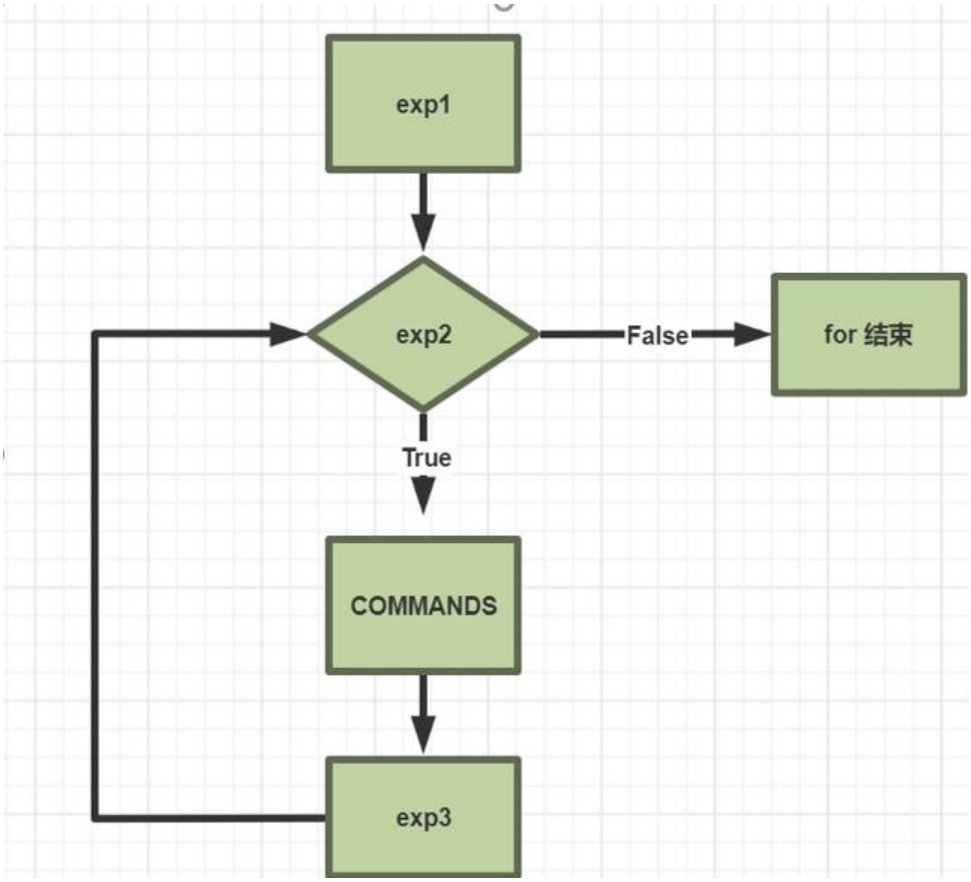
```
for i in *;do
YYYY_MM=`echo $i | cut -d"-" -f1,2`
DD=`echo $i | cut -d"-" -f3`
[ -d $YYYY_MM/$DD ] || mkdir -p $YYYY_MM/$DD &>/dev/null
mv $i/* $YYYY_MM/$DD
done
rm -rf $DIR/*-*-*
```

面试题：扫描一个网段：10.0.0.0/24,判断此网段中主机在线状态，将在线的主机IP打印出来

```
IP="192.168.10."
for i in {1..254};do
{ ping -c1 -W1 ${IP}${i} &>/dev/null && echo "${IP}${i} is up" || echo "${IP}${i} is down"; }
done
```

格式2：

双括号方法，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语言风格的量操作




```
for ((: for ((exp1;exp2;exp3));do COMMANDS;done

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
```

说明：

- 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，而后在做条件判断

范例：九九乘法表

```
for ((i=1;i<=9;i++));do
    for ((j=1;j<=i;j++));do
        echo -e "${j}*${i}=$[j*i]\t\c"
    done
    echo
done
```

范例：等腰三角形

```
read -p "请输入三角形的行数:"  b
for ((i=1;i<=b;i++));do
    for ((k=0;k<=b-i;k++));do
        echo -e " \c"
    done
   for ((j=1;j<=2*i-1;j++));do
        echo -e "*\c"
   done
   echo
done
```

范例：生成进度

```
[09:56:39 root@centos8 ~]#for ((i=1;i<=100;i++));do printf "\e[4D%3d%%" $i;sleep 0.01s; do
e
100%
```

范例：

```
[09:58:09 root@centos8 ~]#for ((;;));do echo for;sleep 1;done
for
for
for
for
for
for
```