2-SHELL 脚本编程进阶

作者：Carey

原文链接：https://ld246.com/article/1609234331454

来源网站：链滴

4 流程控制

4.1 循环

4.1.1 循环 while

格式：

while COMMANDS; do COMMANDS; done

while CONDITION; do
循环体
done

说明：

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“true”，则执行一次循环；直到条件测试状态为“false”终止循环，因此：CONDITION一般应该循环控制变量；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

无限循环

while true; do
循环体
done

```
while : ; do
循环体
done
```

范例：

```
[11:03:16 root@centos8 ~]#sum=0;i=1;while ((i<=100));do let sum+=i;let i++;done;echo $su

5050
```

范例：根据硬盘使用百分比来发送警告邮件

```
WARNING=80
while : ;do
USE=`df | sed -rn '/^\/dev/s#.* ([0-9]{1,3})%.*#\1#p' | sort -nr | head -1`
if [ $USE -gt $WARNING ];then
echo Disk will be full ftom `hostname -I` | mail -s "disk warning" 1191400158@qq.com
fi
sleep 10
done
```

4.2.4 循环nutil

格式：

```
until COMMANDS; do COMMANDS;done
until CONDITION;do
循环体
done
```

说明：

进入条件：CONDITION为false

退出条件：CONDITION为true

范例：

```
[11:42:41 root@centos8 ~]#sum=0;i=1;until ((i>100));do let sum+=i;let i++;done;echo $sum
5050
```

无限循环

```
until false;do
循环体
done
```

4.2.4 循环控制语句continue

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

格式：

```
while CONDITION1;do
```

```
CMD
....
if CONDITIPN2;then
continue
fi
CMDn
...
done
```

范例：

```
for ((i=0;i<10;i++));do
for ((j=0;j<10;j++));do
[ $j -eq 5 ] && continue 2
echo $j
done
echo ----------------------
done
[11:55:05 root@centos8 ~]#./continue_for.sh
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
```

```
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4
```

4.2.5 循环控制语句 break

break [N]：提前结束第N层整个循环，最内层为1层

格式：

```
while CONDITION1;do
CMD1
...
if CONDITION2;then
break
fi
CMDn
...
done
```

范例：

```
for ((i=0;i<10;i++));do
for ((j=0;j<10;j++));do
[ $j -eq 5 ] && break
echo $j
done
echo ----------------------
done
[11:59:08 root@centos8 ~]#./break_for.sh
0
1
2
3
4
-

0
1
2
3
4
```

-

0
1
2
3
4
-

范例：

```
NUM=$[RANDOM%10]

while read -p "请输入0-9之间的数字：" INPUT;do
if [ $INPUT -eq $NUM ];then
echo "恭喜猜对了！"
break
elif [ $INPUT -gt $NUM ];then
echo "数字太大了，重新猜！"
else
echo "数字太小了，重新猜！"
fi
done
```

4.2.6 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hift

范例：doit.sh

```
until [ -z "$1" ]
do
echo "$1"
shift
done
[14:18:08 root@centos8 ~]#./doit.sh a b c d
a
b
c
d
```

范例：创建用户

```
if [ $# -eq 0 ];then
echo "Usage: `basename $0` user1 user2 ..."
exit
fi

while [ "$1" ];do
if id $1 &> /dev/null;then
echo $1 is exist
```

```
else
useradd $1
echo "$1 is created"
fi
shift
done
echo "All user is created"
[14:23:44 root@centos8 ~]#./shift_batch_usr.sh zz cy
zz is exist
cy is created
All user is created
```

4.2.7 while 特殊用法 while read

while 循环的特殊用法，遍历文件或文本的每一行

格式：

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

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

范例：

```
[14:24:00 root@centos8 ~]#echo zhangzhuo | read x ; echo $x

[14:27:23 root@centos8 ~]#echo zhangzhuo | while read x ;do echo $x;done
zhangzhuo
[14:27:58 root@centos8 ~]#echo zhang | { read x; echo $x ;}
zhang
[14:28:20 root@centos8 ~]#echo zhang | ( read x; echo $x )
zhang
[14:28:33 root@centos8 ~]#echo zhang cheng wang | (read X Y Z; echo $X $Y $Z)
zhang cheng wang
[14:29:07 root@centos8 ~]#echo zhang cheng wang | while read X Y Z;do echo $X $Y $Z;done
zhang cheng wang
```

范例：

```
WARNING=80
df | sed -rn '/^\/dev/s#.* ([0-9]{1,3})%.*#\1#p' | while read DEVICEUSE;do
echo $DEVICEUSE
if [ $DEVICEUSE -gt $WARNING ];then
echo Disk will be full ftom `hostname -I` | mail -s "disk warning" 1191400158@qq.com
fi
done
```

范例：找到ssh连接本主机失败超过三次的ip加入到防火墙中

```
MAX=3
lastb | sed -rn '/ssh:/s#.* ([0-9.]{1,4}{3}[0-9]{1,3}) .*#\1#p' | sort | uniq -c | while read count ip ;
o
```

```
if [ $count -gt $MAX ];then
echo $ip
iptables -A INPUT -s $ip -j REJECT
fi
done
```

范例：查看/sbin/nologin的shell类型的用户名和UID

```
while read passwd ;do
if [[ $passwd =~ /sbin/nologin ]];then
echo $passwd | cut -d: -f1,3
fi
done < /etc/passwd
```

4.2.8 循环与菜单 select

格式：

```
select NAME [in WORDS ... ;]do COMMANDS;done
```

select variable in list ;do

循环体命令

done

说明：

- select 循环主要用于创建菜单，按数字顺序排列的菜单项显示在标志错误上，并显示PS3提示符，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是个无限循环体，因此要用break命令退出循环，或用exit命令终止脚本。也可以按ctrl+c退循环
- 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

范例：

```
sum=0
PS3="请点菜（1-4）："
select MENU in 北京烤鸭 佛跳墙 小龙虾 点菜结束;do
case $REPLY in
1)
echo $MENU 价格是 100
let sum+=100
;;
2)
echo $MENU 价格是 88
let sum+=88
;;
3)
echo $MENU 价格是 66
```

```
let sum+=66
;;
4)
echo "点菜结束，退出"
break
;;
*)
echo "点菜错误，重新选择"
;;
esac
done
echo "总价格是：$sum"
```

5 函数 function

5.1 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程序的一分

函数和shell程序区别

- Shell程序在子Shell中运行
- 函数在当前Shell中运行。因此在当前Shell中，函数可对shell中变量进行修改

5.2 管理函数

函数由俩部分组成：函数名和函数体

帮助参看：help function

5.2.1 定义函数

```
#语法一
func_name(){
函数体
}
#语法二
function func_name{
函数体
}
#语法三
function func_name(){
函数体
}
```

5.2.2 查看函数

```
#查看当前已定义的函数名
```

```
declare -F
#查看当前已定义的函数定义
declare -f
#查看指定当前已定义的函数名
declare -f func_name
#查看当前已定义的函数名定义
declare -F func_name
```

5.2.3 删除函数

格式：

```
unset func_name
```

5.3 函数调用

函数的调用方式

- 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，通过给定函数名调用函数，函数名出现的地方，会被自动替换为函代码

函数的生命周期：被调用时创建，返回时终止

5.3.1 交互式环境调用函数

交互式环境下定义和使用函数

范例：

```
[15:18:30 root@centos8 ~]#dir(){

> ls -l
> }
[15:23:24 root@centos8 ~]#dir
 total 36
-rwxr-xr-x 1 root root 481 Dec 26 11:58 break_for.sh
-rwxr-xr-x 1 root root 486 Dec 26 11:53 continue_for.sh
```

范例：实现判断Centos的主版本

```
[15:23:27 root@centos8 ~]#centos_version(){
> sed -rn 's#^.* +([0-9]+)\..*#\1#p' /etc/redhat-release
> }
 [15:25:12 root@centos8 ~]#centos_version
 8
```

5.3.2 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，调函数仅使用其函数名即可

```
hello(){
echo "Hello date is `date +%F`"
}
echo "now going to the function hello"
hello
echo "back from the function"
[15:29:59 root@centos8 ~]#./func1.sh
now going to the function hello
Hello date is 2020-12-26
back from the function
```

5.3.3 使用函数文件

可以将经常使用的函数存入一个单独的函数文件，然后将函数文件载入shell，在进行调用函数

文件名可以任意选取，但最好与相关任务有某种联系，例如：functions

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delcare -f或set命令查看所有义的函数，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，在重新载入此文件

实现函数文件的过程：

- 创建函数文件，只存放函数的定义
- 在shell脚本或交互式shell中调用函数文件，格式如下：

```
. filename 或 source filename
```

范例：

```
[15:49:02 root@centos8 ~]#cat functions.sh
#!/bin/bash
hello(){
echo Run hello Function
}
hello2(){
echo Run hello2 Function
}
[15:48:33 root@centos8 ~]#. functions.sh
[15:48:42 root@centos8 ~]#hello
Run hello Function
[15:48:46 root@centos8 ~]#hello2
Run hello2 Function
[15:48:49 root@centos8 ~]#declare -f hello hello2
hello ()
{
echo Run hello Function
}
hello2 ()
{
```

```
echo Run hello2 Function
}
```

5.4 函数返回值

函数的执行结果返回值：

- 使用echo等命令进行输出
- 函数体中调用命令的输出结果

函数的退出状态码：

- 默认取决于函数中执行的最后一条命令的退出状态码
- 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

5.5 环境函数

类似于环境变量，也可以定义环境函数，使子进程也可以使用父进程定义的函数

定义环境函数：

```
export -f function_name
declare -xf function_name
```

查看环境函数：

```
export -f
declare -xf
```

5.6 函数参数

函数可以接受参数：

- 传递参数给函数：在函数后面以空白分割给定参数列表，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

范例：实现进度条功能

```
print_chars(){
#传入的第一个参数指定要打印的字符串
local char="$1"
#传入的第二个参数指定要打印多少次指定的字符串
local number="$2"
local c
for ((c=0;c<number;++c));do
printf "$char"
```

```
done
}
COLOR=32
declare -i end=50
for ((i=1;i<=end;i++));do
printf "\e[1;${COLOR}m\e[80D["
print_chars "#" $i
print_chars " " $((end - i))
printf "] %3d%%\e[0m" $((i * 2))
sleep 0.1s
done
echo
```

```
[17:32:51 root@centos8 ~]#./progress_chart.sh
[####################                          ]  42%
```

5.7 函数变量

变量作用域：

- 普通变量：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，本地变量的作用范是当前shell脚本程序文件，包括脚本中的函数
- 环境变量：当前shell和子shell有效
- 本地变量：函数的生命周期；函数结束时变量自动销毁

注意：

- 如果函数中定义了普通变量，且名称和局部变量相同，则使用本地变量
- 由于普通变量和局部变量会冲突，建议在函数中只使用本地变量

在函数中定义本地变量的方法

```
local NAME=VALUE
```

5.8 函数递归

函数递归：函数直接或间接调用自身，注意递归层数，可能会陷入死循环

递归示例：

阶乘是基斯顿·卡曼于1808年发明的运算符号，是数学术语，一个正整数的阶乘是所有小于及等于该的正整数的积，并且0和1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：n!=1,n!=(n-1)!*n

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

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

范例：fact.sh

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

范例：测试递归的嵌套深度

```
test(){
let i++
echo i=$i
test
}
test
```

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

参考：https://en.wikipedia.org/wiki/Fork_bomb

函数实现：

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

脚本实现

```
./$0|./$0&
```

后面的&表示后台执行

6 其他脚本相关

6.1 信号捕捉 trap

```
#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 '触发指令' 信号

#忽略信号的操作
trap '' 信号

#恢复信号的操作
trap '-' 信号

#列出自定义信号操作
trap -p

#当脚本退出时，执行finish函数
trap finish EXIT
```

范例：

```
trap "echo 'Press ctrl+c or ctrl+\'" int quit
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
```

范例：当脚本正常或异常退出时，也会执行finish函数

```
finish(){
echo finish | tee -a /root/finish.log
}

trap finish exit
while true ;do
echo running
sleep 1
done
```

6.2 创建临时文件mktemp

mktemp命令用于创建并显示临时文件，可避免冲突

格式：

```
mktemp [OPTION]... [TEMPLATE]
说明:TEMPLATE:filenameXXX,X至少要出现三个

常见选项：

-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
```

范例：

```
[10:40:10 root@centos8 ~]#mktemp
/tmp/tmp.5pZirsD5E2
```

```
[10:42:58 root@centos8 ~]#mktemp --tmpdir=/testdir testXXXXXX
[10:43:11 root@centos8 ~]#tmpdir=`mktemp -d /tmptestdirXXX`
```

范例：

```
DIR=`mktemp -d /tmp/trash-$(date +%F_%H-%M-%S)XXXXXX`
mv $* $DIR
echo $* is move to $DIR
```

6.3 安装复制文件

install 功能相当于cp,chmod,chown,chgrp等相关工具的集合

install命令格式：

```
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
-d DIRNAME 目录
```

范例：

```
[10:54:33 root@centos8 ~]#install -m 700 -o zhang -g zhang finish.log zzzz
[10:55:13 root@centos8 ~]#install -m 770 -d /testdir/install
```

6.4 交互式转化批处理工具expect

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的场景，借助expect处理交互式的命令，可以将交互过程如：ssh登录，ftp登录等写在一个脚本上，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作效率

范例：安装expect及mkpasswd工具

```
[10:56:02 root@centos8 ~]#yum -y install expect
[11:01:16 root@centos8 ~]#rpm -ql expect

[11:02:41 root@centos8 ~]#mkpasswd -l 15 -d 3 -C 5
LB7g3L[9hkiMmaV
```

expect语法：

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

常见选项：

- -c：从命令行执行expect脚本，默认expect是交互地执行的

- -d：可以输入输出调试信息

示例：

```
expect -c 'expect "\n" {send "pressed enter\n"}'
expect -d ssh.exp
```

expect中相关命令

- spawn　　启动新的进程
- expect　　从新进程接收字符串
- send　　用于向进程发送字符串
- 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最常用的语法(tcl语言：模式-动作)

单一分支模式语法：

```
[11:19:03 root@centos8 ~]#expect
expect1.1> expect "hi" {send "You said hi\n"}
sssssssssssshi
You said hi
匹配到hi后，会输出you said hi，并换行
```

多分支模式语法：

```
[11:22:16 root@centos8 ~]#expect
expect1.1> expect "hi" {send "hi"} "hehe" {send "hehe\n"} "bye" {send ""sssss" }
```

范例1：

```
#!/usr/bin/expect
spawn scp /etc/redhat-release 192.168.10.102:/root
expect {
"yes/no" { send "yes\n";exp_continue }
"password" { send "123456\n" }
}
expect eof
[11:29:02 root@centos8 ~]#expect expect.sh
```

范例2：

```
#!/usr/bin/expect
spawn ssh 192.168.10.102
expect {
"yes/no" { send "yes\n";exp_continue }
"password" { send "123456\n"}
}
interact
[11:35:01 root@centos8 ~]#expect expect_ssh_ubuntu.sh
```

范例3：expect变量

```
#!/usr/bin/expect
set ip 192.168.10.102
set user root
set password 123456
set timeout 10
spawn ssh $user@$ip
expect {
"yes/no" { send "yes\n";exp_continue }
"password" { send "${password}\n"}
}
interact
[11:35:01 root@centos8 ~]#expect expect_ssh_ubuntu.sh
```

范例4：expect 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
[11:42:30 root@centos8 ~]#./expect4.sh 192.168.10.102 zhang 123456
```

范例5：expect执行多个命令

```
#!/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"}
}
expect "]#" { send "useradd cy\n" }
expect "]#" { send "echo -e '123456' | passwd  --stdin cy\n" }
expect "]#" { send "exit\n" }
expect eof
[14:02:07 root@centos8 ~]#./expect5.sh 192.168.10.71 root 123456
俩个主机都是centos不然设置密码不能用这个方法
```

范例6：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 123456 | passwd --stdin hehe\n" }
expect "]#" { send "exit\n" }
expect eof
EOF
```

范例7：shell脚本利用循环调用expect在Centos和Ubuntu上批量创建用户

```
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
#Date: 2020-12-29
#FileName：  expect7.sh
#URL: https://www.zhangzhuo.ltd
#Description：  The test script
#Copyright (C): 2020 All rights reserved
#**********************************************************************
NET=192.168.10
user=root
password=123456
IPLIST="
71
102
"

for ID in $IPLIST;do
ip=$NET.$ID

expect <<EOF
set timeout 20
spawn ssh $user@$ip
expect {
"yes/no" { send "yes\n";exp_continue }
"password" { send "${password}\n" }
}
expect "#" { send "useradd test\n" }
expect "#" { send "useradd txt\n" }
expect "#" { send "exit\n" }
expect eof
EOF
done
```

范例8：

```
#!/bin/bash
#
#**********************************************************************
#Author:zhangzhuo
#QQ: 1191400158
```

```
#Date: 2020-12-29
#FileName：expect8.sh
#URL: https://www.zhangzhuo.ltd
#Description：The test script
#Copyright (C): 2020 All rights reserved
#********************************************************************
NET=192.168.10
user=root
password=123456
IPLIST="
71
"

for ID in $IPLIST;do
ip=$NET.$ID

expect <<EOF
set timeout 20
spawn ssh $user@$ip
expect {
    "yes/no" { send "yes\n";exp_continue }
    "password" { send "${password}\n" }
}
expect "#" { send "sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config\n" }
expect "#" { send "setenforce 0\n" }
expect "#" { send "exit\n" }
expect eof
EOF
done
```

7 数组 array

7.1 数组介绍

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

- 索引的编号从0开始，属于数值索引
- 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)

7.2 声明数组

```
#普通数组可以不事先声明，直接使用
declare -a ERRAY_NAME
#关联数组必须事先声明，在使用
declare -A ARRAY_NAME
```

注意：俩者不可相互转换

7.3 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
```

范例：

```
[14:47:05 root@centos8 ~]#weekdays[0]="zhang"
[14:56:14 root@centos8 ~]#weekdays[4]="zhuo"
```

(2)一次赋值全部元素

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

范例：

```
[14:56:22 root@centos8 ~]#title=("ceo" "coo" "cto")
[14:58:10 root@centos8 ~]#num=({1..10})
[14:58:22 root@centos8 ~]#alpha=({a..g})
[14:58:36 root@centos8 ~]#file=(*.sh)
```

(3)只赋值特点元素

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

(4)交互式数组值对赋值

```
read -a ARRAY
```

范例：俩种格式不能相互转换

```
[14:58:46 root@centos8 ~]#declare -A course
[15:01:07 root@centos8 ~]#declare -a course
-bash: declare: course: cannot convert associative to indexed array
[15:01:14 root@centos8 ~]#file=(*.sh)
[15:01:30 root@centos8 ~]#declare -A file
-bash: declare: file: cannot convert indexed to associative array
```

范例：关联数组示例

```
[15:01:38 root@centos8 ~]#i=a
[15:02:26 root@centos8 ~]#j=1
[15:02:29 root@centos8 ~]#declare -A arr
[15:02:37 root@centos8 ~]#arr[$i$j]=zhang
[15:02:57 root@centos8 ~]#j=2
[15:03:02 root@centos8 ~]#arr[$i$j]=cy
[15:03:21 root@centos8 ~]#echo ${arr[*]}
cy zhang
[15:03:34 root@centos8 ~]#echo ${arr[a1]}
zhang
```

```
[15:04:16 root@centos8 ~]#echo ${arr[a2]}
cy
```

7.4 显示所有数组

显示所有数组：

```
[15:04:19 root@centos8 ~]#declare -a
```

范例：

```
[15:04:19 root@centos8 ~]#declare -a
declare -a BASH_ARGC=()
declare -a BASH_ARGV=()
declare -a BASH_COMPLETION_VERSINFO=([0]="2" [1]="7")
```

7.5 引用数组

引用数组元素

```
${ARRAY_NAME[INDEX]}
#如果省略[INDEX]表示引用下标为0的元素
```

范例：

```
[15:09:09 root@centos8 ~]#declare -a title=([0]="ceo" [1]="coo" [2]="cto")
[15:09:13 root@centos8 ~]#echo ${title[1]}
coo
[15:09:15 root@centos8 ~]#echo ${title[2]}
cto
[15:09:21 root@centos8 ~]#echo ${title[3]}
```

引用数组所有元素

```
${ARRAY_NAME[*]}
${ARRAY_NAME[@]}
```

范例：

```
[15:09:23 root@centos8 ~]#echo ${title[@]}
ceo coo cto
[15:10:50 root@centos8 ~]#echo ${title[*]}
ceo coo cto
```

数组的长度，即数组中元素的个数

```
${#ARRAY_NAME[*]}
${#ARRAY_NAME[@]}
```

范例：

```
[15:10:54 root@centos8 ~]#echo ${#title[*]}
3
[15:12:18 root@centos8 ~]#echo ${#title[@]}
```

3

7.6 删除数组

删除数组中的某个元素，会导致稀疏格式

unset ARRAY[INDEX]

[15:12:21 root@centos8 ~]#echo ${title[*]}
ceo coo cto
[15:14:47 root@centos8 ~]#unset title[1]
[15:14:59 root@centos8 ~]#echo ${title[*]}
ceo cto

删除整个数组

nuset ARRAY

范例：

[15:15:03 root@centos8 ~]#unset title
[15:15:59 root@centos8 ~]#echo ${title[*]}

7.7 数组数据处理

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}

范例：

[15:19:31 root@centos8 ~]#echo ${num[@]:2:3}
2 3 4
[15:19:53 root@centos8 ~]#echo ${num[@]:6}
6 7 8 9 10

向数组中追加元素：

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

范例：

[15:20:02 root@centos8 ~]#num[${#num[@]}]=11
[15:22:00 root@centos8 ~]#echo ${#num[*]}
12
[15:22:18 root@centos8 ~]#echo ${num[*]}
0 1 2 3 4 5 6 7 8 9 10 11

7.8 关联数组

```
declare -A ARRAY_NAME
ARRAY_NAME=([idx_name]='val1' [idx_name]='val2' ...)
```

注意：关联数组必须先声明在调用

范例：

```
[15:24:06 root@centos8 ~]#name[ceo]=zhang
[15:26:37 root@centos8 ~]#name[cto]=wang
[15:26:48 root@centos8 ~]#name[coo]=cy
[15:26:59 root@centos8 ~]#echo ${name[ceo]}
cy
[15:27:11 root@centos8 ~]#echo ${name[cto]}
cy
[15:27:32 root@centos8 ~]#echo ${name[coo]}
cy
[15:27:35 root@centos8 ~]#declare -A name
-bash: declare: name: cannot convert indexed to associative array
[15:27:56 root@centos8 ~]#unset name
[15:28:03 root@centos8 ~]#declare -A name
[15:28:11 root@centos8 ~]#name[ceo]=zhang
[15:28:19 root@centos8 ~]#name[cto]=wang
[15:28:31 root@centos8 ~]#name[coo]=cy
[15:28:40 root@centos8 ~]#echo ${name[*]}
zhang wang cy
```

范例：关联数组

```
[15:28:59 root@centos8 ~]#declare -A student
[15:30:20 root@centos8 ~]#student[name1]=lijun
[15:30:36 root@centos8 ~]#student[name2]=ziqiang
[15:30:44 root@centos8 ~]#student[age1]=18
[15:31:04 root@centos8 ~]#student[age2]=16
[15:31:10 root@centos8 ~]#student[gender1]=m
[15:31:30 root@centos8 ~]#student[city1]=najing
[15:33:05 root@centos8 ~]#student[city2]=anhui
[15:33:13 root@centos8 ~]#student[gender2]=m
[15:33:36 root@centos8 ~]#student[name50]=alice
[15:33:50 root@centos8 ~]#student[name3]=tom
[15:35:09 root@centos8 ~]#for i in {1..50};do echo student[name$i]=${student[name$i]};done
```

7.9 范例

范例：生产10个随机数保存于数组中，并找出其最大值和最小值

```
declare -i min max
declare -a nums
for ((i=0;i<10;i++));do
nums[$i]=$RANDOM
[ $i -eq 0 ] && min=${nums[0]} && max=${nums[0]} && continue
[ ${nums[$i]} -gt $max ] && max=${nums[$i]} && continue
[ ${nums[$i]} -lt $min ] && min=${nums[$i]}
done
echo "ALL numbers are ${nums[*]}"
```

```
echo Max is $max
echo Min is $min
```

范例：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.log结尾的文件；统出其下标偶数的的文件中的行数之和

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

8 字符串处理

8.1 字符串切片

8.1.1 基于偏移量取字符串

```
#返回字符串变量var的长度
${#var}

#返回字符串变量var中第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}

#返回字符串变量var中第offset个字符后(不包括offset个字符)的字符开始，长度为number的部分
${var:offset:bunber}

#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: -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，即：掐头去尾
${var:offset:-length}

#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
${var: -length:-offset}
```

范例：

```
[16:02:06 root@centos8 ~]#str=abcdef我你他
[16:14:10 root@centos8 ~]#echo ${#str}
9
[16:14:21 root@centos8 ~]#echo ${str:2}
cdef我你他
[16:14:30 root@centos8 ~]#echo ${str:2:3}
cde
[16:14:36 root@centos8 ~]#echo ${str: -3}
我你他
```

```
[16:14:48 root@centos8 ~]#echo ${str:-3}
abcdef我你他
[16:15:00 root@centos8 ~]#echo ${str:2:-3}
cdef
[16:15:16 root@centos8 ~]#echo ${str: -2:-3}
-bash: -3: substring expression < 0
[16:15:35 root@centos8 ~]#echo ${str: -3:-2}
我
[16:15:43 root@centos8 ~]#echo ${str:-3:-2}
abcdef我你他
[16:15:52 root@centos8 ~]#echo ${str: -5:-2}
ef我
```

8.1.2 基于模式取字串

```
#其中word可以是指定的任意字符，自左而右，查找var变量所存储的字符串，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右word指定的字符之间的所有内容
${var##*word}:
```

范例：

```
[16:16:00 root@centos8 ~]#file="var/log/messages"
[16:21:46 root@centos8 ~]#echo ${file#*/}
log/messages
[16:22:00 root@centos8 ~]#echo ${file##*/}
messages

#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
${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
${var%%word*}
```

范例：

```
[16:28:01 root@centos8 ~]#file="var/log/messages"
[16:28:03 root@centos8 ~]#echo ${file%/*}
var/log
[16:28:20 root@centos8 ~]#echo ${file%%/*}
var
```

范例：

```
[16:28:24 root@centos8 ~]#url=http://www.zhangzhuo.ltd:8080
[16:29:10 root@centos8 ~]#echo ${url##*:}
8080
[16:29:23 root@centos8 ~]#echo ${url%%:*}
http
```

8.2 查找替换

#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
${var/%pattern/substr}

8.3 查找并删除

#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
${var/%pattern}

8.4 字符大小写转换

#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
${var,,}

9 高级变量

9.1 高级变量赋值

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

范例：

```
[16:29:36 root@centos8 ~]#title=ceo
[16:33:22 root@centos8 ~]#name=${title-zhang}
[16:33:40 root@centos8 ~]#echo $name
ceo
[16:33:51 root@centos8 ~]#title=
[16:34:23 root@centos8 ~]#name=${title-zhang}
[16:34:28 root@centos8 ~]#echo $name

[16:34:30 root@centos8 ~]#unset title
[16:34:39 root@centos8 ~]#name=${title-zhang}
[16:34:42 root@centos8 ~]#echo $name
zhang
```

范例：

```
[16:34:46 root@centos8 ~]#title=ceo
[16:35:48 root@centos8 ~]#name=${title:-zhang}
[16:36:03 root@centos8 ~]#echo $name
ceo
[16:36:09 root@centos8 ~]#title=
[16:36:14 root@centos8 ~]#name=${title:-zhang}
[16:36:17 root@centos8 ~]#echo $name
zhang
[16:36:35 root@centos8 ~]#unset title
[16:36:43 root@centos8 ~]#name=${title:-zhang}
[16:36:47 root@centos8 ~]#echo $name
zhang
```

9.2 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俩个命令用于指定变量的类型，个命令是等价的

```
declare [选项] 变量名

选项：
-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,相当于export
-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
```

9.3 变量间接引用

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现其功能的变量,该命令对变量进行两次扫描

范例：

```
[16:36:50 root@centos8 ~]#CMD=whoami
[16:40:29 root@centos8 ~]#echo $CMD
whoami
[16:40:33 root@centos8 ~]#eval $CMD
root
[16:40:45 root@centos8 ~]#n=10
[16:40:50 root@centos8 ~]#echo {0..$n}
{0..10}
[16:41:02 root@centos8 ~]#eval echo {0..$n}
0 1 2 3 4 5 6 7 8 9 10
[16:41:18 root@centos8 ~]#for i in `eval echo {1..$n}`;do echo i=$i;done
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
[16:42:14 root@centos8 ~]#i=a
[16:42:19 root@centos8 ~]#j=1
[16:42:22 root@centos8 ~]#$i$j=hello
-bash: a1=hello: command not found
[16:42:33 root@centos8 ~]#eval $i$j=hell
[16:42:46 root@centos8 ~]#echo $i$j
a1
```

9.3.2 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

bash shell提供了俩种格式实现间接变量引用

```
#方法1
eval tempvar=\$$variable1
#方法2
tempvar=${!variable1}
```

范例：

```
[16:42:54 root@centos8 ~]#ceo=name
[16:45:16 root@centos8 ~]#name=zhang
[16:45:22 root@centos8 ~]#echo $ceo
name
[16:45:27 root@centos8 ~]#echo $$ceo
67706ceo
```

```
[16:45:38 root@centos8 ~]#echo $$
67706
[16:45:44 root@centos8 ~]#echo \$$ceo
$name
[16:45:53 root@centos8 ~]#eval echo \$$ceo
zhang
[16:46:05 root@centos8 ~]#eval tmp=\$$ceo
[16:46:20 root@centos8 ~]#echo $tmp
zhang
[16:46:28 root@centos8 ~]#echo ${!ceo}
zhang
```

范例：

```
[16:46:38 root@centos8 ~]#N1=N2
[16:47:13 root@centos8 ~]#N2=zhangzhuo
[16:47:20 root@centos8 ~]#eval NAME=\$$N1
[16:47:32 root@centos8 ~]#echo $NAME
zhangzhuo
[16:47:48 root@centos8 ~]#NAME=${!N1}
[16:48:02 root@centos8 ~]#echo $NAME
zhangzhuo
```

9.3.3 变量引用reference

```
ceo=zhang
title=ceo
declare -n ref=$title
[ -R ref ] && echo reference
echo $ref
ceo=cy
echo $ref
[16:50:38 root@centos8 ~]#./test.sh
reference
zhang
cy
```