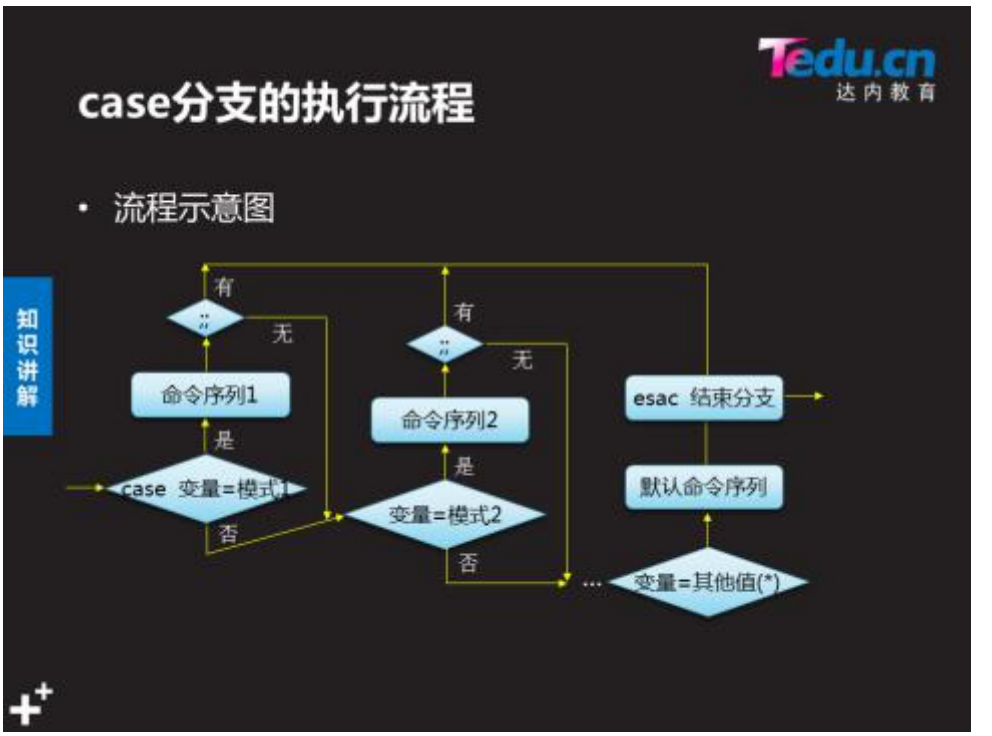
case 语句函数及中断字符串处理

一 case 语句　功能类似 if,代码比 if 要精简,但功能没有 if 强大,是简化版 if.

1.1 语法结构

```
case  变量  in
模式 1)
    命令序列 1;;
模式 2)
    命令序列 2;;
    .. ..
*)
    默认命令序列
esac
```



注意:除最后一行命令序列可不跟双分号外,其他命令序列必须有双分号.

基本用法示例:

```
#!/bin/bash
case $1 in
a|aa|A)      #模式内可用符号 | 书写该模式下多种情况.
        echo aaa;;
b|bb|B)
        echo bbb;;
```

```
*)
        echo "a|b"
esac
```

案例： 编写 test.sh 脚本，相关要求如下：

能使用 redhat、fedora 控制参数

控制参数通过位置变量$1 传入

当用户输入 redhat 参数，脚本返回 fedora

当用户输入 fedora 参数，脚本返回 redhat

当用户输入其他参数，则提示错误信息

解析：

脚本编写：

```
[root@svr5 ~]# vim test.sh
     #!/bin/bash
     case $1 in
     redhat)
          echo "fedora";;
     fedora)
          echo "redhat";;
      *)                                    #默认输出脚本用法
          echo "用法: $0 {redhat|fedora}"
```

```
        esac
```

赋予脚本权限:

```
    [root@svr5 ~]# chmod +x test.sh
```

案例:编写一键**部署**软件脚本,编写脚本实现一键部署 Nginx 软件（Web 服务器）:

一键源码安装 Nginx 软件

脚本自动安装相关软件的依赖包

解析:

源码安装 Nginx 需要提前安装依赖包软件 gcc,openssl-devel,pcre-devel

将软件包从真机 SCP 到虚拟机:

```
[student@room9pc01  02]$  scp  /linux-soft/02/lnmp_soft.tar.gz
root@172.25.0.11:/
[root@server0 /]# tar -xf lnmp_soft.tar.gz
[root@server0 /]# cd lnmp_soft/
[root@server0 lnmp_soft]# cp nginx-1.10.3.tar.gz /opt
```

#将源代码包复制到/opt 下

脚本编写:　　**#注意目录路径**

```
[root@server0 lnmp_soft]# vim nginx1.sh
    #!/bin/bash
    tar -xf nginx-1.10.3.tar.gz
    yum -y install gcc openssl-devel pcre-devel &> /dev/null
```

```
cd nginx-1.10.3
./cofigure &> /dev/null      #编译
make &> /dev/null
make install &> /dev/null    #安装
wait
echo “install finished!”
```

确认安装效果:

Nginx 默认安装路径为/usr/local/nginx,该目录下会提供 4 个子目录:

/usr/local/nginx/conf 配置文件目录

/usr/local/nginx/html 网站页面目录

/usr/local/nginx/logs Nginx 日志目录

/usr/local/nginx/sbin 主程序目录

主程序命令参数:

```
[root@svr5 ~]#/usr/local/nginx/sbin/nginx   #启动服务
[root@svr5 ~]#/usr/local/nginx/sbin/nginx -s stop   #关闭服务
[root@svr5 ~]#/usr/local/nginx/sbin/nginx -V   #查看软件信息
```

运行 nginx

```
[root@server0 sbin]# ./nginx
```

设置防火墙为 tursted

```
[root@server0 sbin]# firewall-cmd --set-default-zone=trusted
```

关闭 httpd,访问 172.25.0.11

[root@server0 sbin]# firefox 172.25.0.11

案例：编写 Nginx **启动**脚本，要求如下：

脚本支持 start、stop、restart、status

脚本支持报错提示

脚本具有判断是否已经开启或关闭的功能

解析:

脚本通过位置变量$1 读取用户的操作指令，判断是 start、stop、restart 还是 status。

netstat 命令可以查看系统中启动的端口信息，该命令常用选项如下：

-n 以数字格式显示端口号

-t 显示 TCP 连接的端口

-u 显示 UDP 连接的端口

-l 显示服务正在监听的端口信息，如 httpd 启动后，会一直监听 80 端口

-p 显示监听端口的服务名称是什么（也就是程序名称）

```
netstat -ntulp | grep nginx  #查询 nginx 服务状态
netstat -ntulp | grep :80  #查看 80 端口使用状态
```

经常用法:netstat -ntulp

nginx 非 yum 方式安装,systemctl 方式控制无效

脚本编写:

```
[root@server0 opt]# vim nginx2.sh
#!/bin/bash
case   $1 in
start|st)
       /usr/local/nginx/sbin/nginx;;  #开启
stop)
       /usr/local/nginx/sbin/nginx -s stop;;  #关闭
restart|re)
       /usr/local/nginx/sbin/nginx -s stop
       /usr/local/nginx/sbin/nginx;;  #重启先关再开
       #一个模式下有多行命令序列,只在最后的命令序列后加双分号
status)
       netstat -ntulp | grep -q nginx  #grep -q 不输出信息
       if     [ $? -eq 0 ];then      #查询成功
              echo "服务已开启"
       else                          #查询失败
              echo "服务未开启"
       fi;;    #注意双分号
*)
       echo "此脚本正确用法是 start|stop|restart|status";;
```

```
        #最后一个模式的命令序列后可不跟双分号
    esac
```

测试脚本：

```
[root@server0 opt]# bash nginx2.sh stop
[root@server0 opt]# bash nginx2.sh status
服务未开启
[root@server0 opt]# bash nginx2.sh start
[root@server0 opt]# bash nginx2.sh status
服务已开启
[root@server0 opt]# bash nginx2.sh restart
[root@server0 opt]# bash nginx2.sh status
服务已开启
[root@server0 opt]# bash nginx2.sh 123
此脚本正确用法是 start|stop|restart|status
[root@server0 opt]# bash nginx2.sh
此脚本正确用法是 start|stop|restart|status
```

二 函数

2.1 定义

在 shell 环境中,将一些需要重复使用的操作,定义为公共的语句块,即可称为函数

2.2 使用函数的好处

使脚本代码更简洁,增强易读性

提高 shell 脚本的执行效率

2.3 服务脚本中的函数应用

适用于比较复杂的启动\终止操作

方便在需要时多次调用

2.4 函数的定义

```
function 函数名 {
        命令序列
        ... ...
}
```

或

```
函数名() {
        命令序列
        ... ...
}
```

注意:{号前有一个空格

2.5 函数的调用

2.5.1 调用已定义的函数

格式:**函数名**

先定义了才能调用,就好比脚本的”内部命令”

2.5.2 函数传值

格式:**函数名　值 1　值 2　... ...**

传递的值作为函数的”位置参数”,即$1,$2,...$n

2.5.3 函数的查看

格式: **tpye 函数名**　#命令行内定义的函数

```
[root@server0 opt]# type xyz
xyz 是函数
xyz ()
{
    echo xyz;
    ls --color=auto /root/;
    echo $[2+8]
}
```

2.5.4 输出彩色字体

```
[root@server0 opt]# echo -e "\033[32mABC\033[0m"
\033[      32    m    ABC              \033[      0m
           色号        要输出的文本                  返回
```

案例: 编写脚本,输出彩色 ABCDEFG.

```
[root@server0 opt]# vim color.sh
     #!/bin/bash
     colorecho(){
     echo -e "\033[$1m$2\033[0m"
     }
     colorecho 31 ABCDEFG    //脚本内部调用函数,并传值
     colorecho 32 ABCDEFG
```

```
        colorecho 33 ABCDEFG
        colorecho 34 ABCDEFG
        colorecho 35 ABCDEFG
        colorecho 36 ABCDEFG
```

测试

```
[root@server0 opt]# bash color.sh
```

2.5.6 shell 版 fork 炸弹

```
[root@server0 opt]# vim fork.sh
        #!/bin/bash
        .(){        #定义函数.
        .|. &       #后台运行函数.,并将结果管道给下一个函数.
        }
                    #空行
        .           #函数定义完成后首次调用函数.
```

三 中断及退出

3.1 中断\退出及相关指令

类型	含义
break	跳出当前所在的循环体,执行循环体后的语句块
continue	跳过当前循环体内余下的语句,重新判断条件以决定是否继续要执行下一次循环

exit　　退出脚本,默认返回值是 0

案例:编写脚本,使用户输入的数字求和,用户输入 0 时,结束计算并输出之前所有数字之和.

```
[root@server0 opt]# vim qiuhe.sh
    #!/bin/bash
    a=0          #定义变量,用于输入的数字求和
    while :      #永久有效,一直循环
    do
        read -p "请输入数字:" n  #定义变量,单次存储用户输入的数字
        [ $n -eq 0 ] && break     #当用户输入 0 时,跳出循环体
        let a+=$n   #输入非 0 时,对输入的数字求和
    done
    echo "所有输入的数字之和为$a"    #break 跳出循环后执行的语句
    [root@server0 opt]# bash qiuhe.sh
    请输入数字:1
    请输入数字:2
    ......
    请输入数字:6
    请输入数字:0
    所有输入的数字之和为 21
```

案例:从数字 1-20 中查找 6 的倍数,找到之后输出到屏幕.

```
[root@server0 opt]# vim 6bs.sh
#!/bin/bash
for i in {1..20}
  do
        j=$[i%6]  #对 i 取余并赋值给 j
        [ $j -ne 0 ] && continue  #j 不为 0 时,跳过当前循环体内余
            下的语句,重新判断条件以决定是否执行下一次循环
       #[ $j -eq 0 ] || continue
        echo "$i 是 6 的倍数."    #循环体语句,当 j 为 0 时,输出 i
  done
[root@server0 opt]# bash 6bs.sh
6 是 6 的倍数.
12 是 6 的倍数.
18 是 6 的倍数.
```

四 字符串处理

4.1 子串截取

格式: ${var:起始位置:长度}   #var 为字符串变量的名称

起始位置从 0 开始,当为 0 时,可省略

```
[root@server0 opt]# nm="Tarena IT Group"
```

[root@server0 opt]# echo ${nm:0:6}

Tarena

案例:x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ12

34567890,共 62 个字符,随机截取 1 个字符,随机截取 8 个字符

```
#!/bin/bash
x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ12345678
90
echo ${x:$[RANDOM%62]:1}    #随机截取 1 个字符
pass=    #随机截取 8 个字符程序,初始 pass 为空
for i in {1..8}
      do
            n=$[RANDOM%61]
            pass1=${x:$n:1}
            pass=$pass$pass1
      done
echo $pass
```

4.2 子串替换

格式:

只替换第一个匹配结果 **${var/old/new}**

替换全部匹配结果 **${var//old/new}**

new 可为空,new 为空时删除字符串内的 old 字符[潜在的删除功能]

a=11223344

[root@server0 opt]# echo ${a/1/X} #替换第一个 1 为 X

X1223344

[root@server0 opt]# echo ${a//1/X}   #替换所有 1 为 X

XX223344

[root@server0 opt]# echo ${a/2/} #替换第 1 个 2 为空

1123344

[root@server0 opt]# echo ${a//2/} #替换所有 2 为空

113344

4.3 按条件掐头

格式:

从左到右,最短匹配删除　　**${变量名#*关键词}**

从左到右,最长匹配删除　　**${变量名##*关键词}**

*关键词:表示要删除的头部

#号用来最近删除头部,##用来最远删除头部

[root@server0 opt]# a=`head -n 1 /etc/passwd`

[root@server0 opt]# echo $a

root:x:0:0:root:/root:/bin/bash

[root@server0 opt]# echo ${a#*root}

#从左到右,删除第 1 个 root 及之前的字符

:x:0:0:root:/root:/bin/bash

[root@server0 opt]# echo ${a##*root}

#从左到右,删除最后 1 个 root 及之前的字符

:/bin/bash

[root@server0 opt]# echo ${a#*:root}

:/root:/bin/bash #删除第 2 个 root 及之前的字符

4.4 按条件去尾

格式:

从右到左,最短匹配删除　　**${变量名%关键词*}**

从右到左,最长匹配删除　　**${变量名%%关键词*}**

关键词*:表示要删除的尾部

%号用来最近删除尾部,%%用来最远删除尾部

[root@server0 opt]# echo $a

root:x:0:0:root:/root:/bin/bash

[root@server0 opt]# echo ${a%root*}

root:x:0:0:root:/ #从右到左,删除第 1 个 root 及之前的字符

[root@server0 opt]# echo ${a%%root*}

#从右到左,删除最后 1 个 root 及之前的字符

[root@server0 opt]# echo ${a%%root:/*}

root:x:0:0: #从右到左,删除到第 2 个 root:/及之前的字符

案例: 通过字符串删除功能,编写脚本,批量修改文件扩展名

```
[root@server0 opt]# touch abc{1..8}.txt
[root@server0 opt]# vim filerename.sh
    #!/bin/bash
    for FILE in /opt/*.txt
        do
            mv $FILE ${FILE%.*}.doc
            #删除文件后缀名,并添加.doc 后缀名
        done
#!/bin/bash
for FILE in /opt/*.$1
    do
        mv $FILE ${FILE%.*}.$2
    done
[root@server0 opt]# bash filerename.sh txt dc
```

4.5 变量初始值处理

格式: ${var:-word}

若变量 var 已存在且非空,则返回$var 的值

若变量 var 不存在或空,则返回字符串 Word,变量 var 值不变