expect实现无交互登录

expect([ɪkˈspekt]期待)是从它发展出来的。如果你想要写一个能够自动处理输入输出的脚本（如向用户提问并且验证密码）又不想面对C或者Perl，那么expect是你的最好的选择。它可以用来做一些linux下无法做到交互的一些命令操作

安装和使用expect

```
[root@exercise1 opt]# yum  -y  install expect
使用expect创建脚本的方法

1）定义脚本执行的shell
#!/usr/bin/expect
这里定义的是expect可执行文件的链接路径（或真实路径），功能类似于bash等shell功能

2）set timeout 30
设置超时时间，单位是秒，如果设为timeout -1  意为永不超时

3）spawn
spawn是进入expect环境后才能执行的内部命令，如果没有装expect或者直接在默认的SHELL下执行是找不到spawn命令的。不能直接在默认的shell环境中进行执行主要功能，它主要的功能是给ssh运行进程加个壳，用来传递交互指令。

4）expect
这里的expect同样是expect的内部命令
主要功能：判断输出结果是否包含某项字符串，没有则立即返回，否则就等待一段时间后返回，等待时间通过timeout进行设置

5）send
执行交互动作，将交互要执行的动作进行输入给交互指令
命令字符串结尾要加上"\r"，如果出现异常等待的状态可以进行核查

6）exp_continue
继续执行接下来的交互操作

```

```
7）interact
执行完后保持交互状态，把控制权交给控制台；如果不加这一项，交互完成会自动退出

8）$argv
expect脚本可以接受从bash传递过来的参数，可以使用[ lindex $argv n]获得，n从0开始，分别表示第一个，第二个，第三个……参数
```

```
例子：免密码通过SSH登录服务器　　这里不是用密钥
注：运行脚本时，要把#号后面的注释删除，不然无法运行
```

[root@exercise1 opt]# vim /opt/ssh.exp

#!/usr/bin/expect

set ipaddr "192.168.119.140" #相当于shell脚本的赋值

set name "root"

set passwd "123456"

set timeout 30　#设置超时时间，单位是秒；expect 超时等待的时间。默认 timeout 为 10s。

spawn ssh $name@$ipaddr　# spawn 是进入 expect 环境后才可以执行的 expect 内部命令，如果没有 装 expect 或者直接在 shell 下执行是找不到 spawn 命令的。这个就好比 cd 是 shell 的内建命令，离开 shell，就无法执行 cd 一样。 它主要的功能是给 ssh 运行进程加个壳，用来传递交互指令。

expect {

"yes/no" { send "yes\r";exp_continue }

"password" { send "$passwd\r" } #执行交互动作，与手工输入密码的动作等效。

}

expect "#" #判断上次输出结果里是否包含“password:”的字符串，如果有则立即返回，向下执行；否则就一直等待，直到超时时间 break

send "touch /root/ufo1011.txt\r"

send "ls /etc > /root/ufo1011.txt\r"

send "mkdir /tmp/ufo1011\r"

send "exit\r"

expect eof #执行完成上述命令后，退出 expect，把控制权交给控制台(终端)，变回手工操作

[root@exercise1 opt]# expect ssh.exp　　#开始执行

```
对服务器批量管理
[root@base  ~]# cat ip_pass.txt #这里写上要执行的 IP 地址和 root 用户密码
192.168.245.131 123456
192.168.245.132 123456
192.168.245.133 123456

[root@base  ~]# cat ssh2.exp #编写要执行的操作
#!/usr/bin/expect
set ipaddr [lindex $argv 0]   #相当于ipaddr=$1
set passwd [lindex $argv 1]  #相当于passwd=$2
set timeout 30
spawn ssh root@$ipaddr
expect {
"yes/no" { send "yes\r";exp_continue }
"password" { send "$passwd\r" }
}
expect "#"
send "touch /root/a.txt\r"
send "ls /etc > /root/a.txt\r"
send "mkdir /tmp/a\r"
send "exit\r"
expect eof
[root@base  ~]# cat login.sh #开始执行
#!/bin/bash

for ip in `awk '{print $1}' /root/ip_pass.txt`  #打印ip_pass.txt里面的每一行的第一个参数
do
    pass=`grep $ip /root/ip_pass.txt|awk '{print $2}'` #打印ip_pass.txt里面的每一行的第二个参数
    expect /root/ssh2.exp $ip $pass   相当于sh xxx.sh $1 $2
done
```

awk基本应用

grep 和 egrep：文本过滤的
sed：流编辑器，实现编辑的
awk：文本报告生成器，实现格式化文本输出

概念

AWK是一种优良的文本处理工具，Linux及Unix环境中现有的功能最强大的数据处理引擎之一。
awk命名:Alfred Aho Peter、Weinberger和brian kernighan三个人的姓的缩写。
awk---->gawk即：gun awk\
在linux上常用的是gawk,awk是gawk的链接文件

任何awk语句都是由模式和动作组成，一个awk脚本可以有多个语句。模式决定动作语句的触发条件和触发时间。

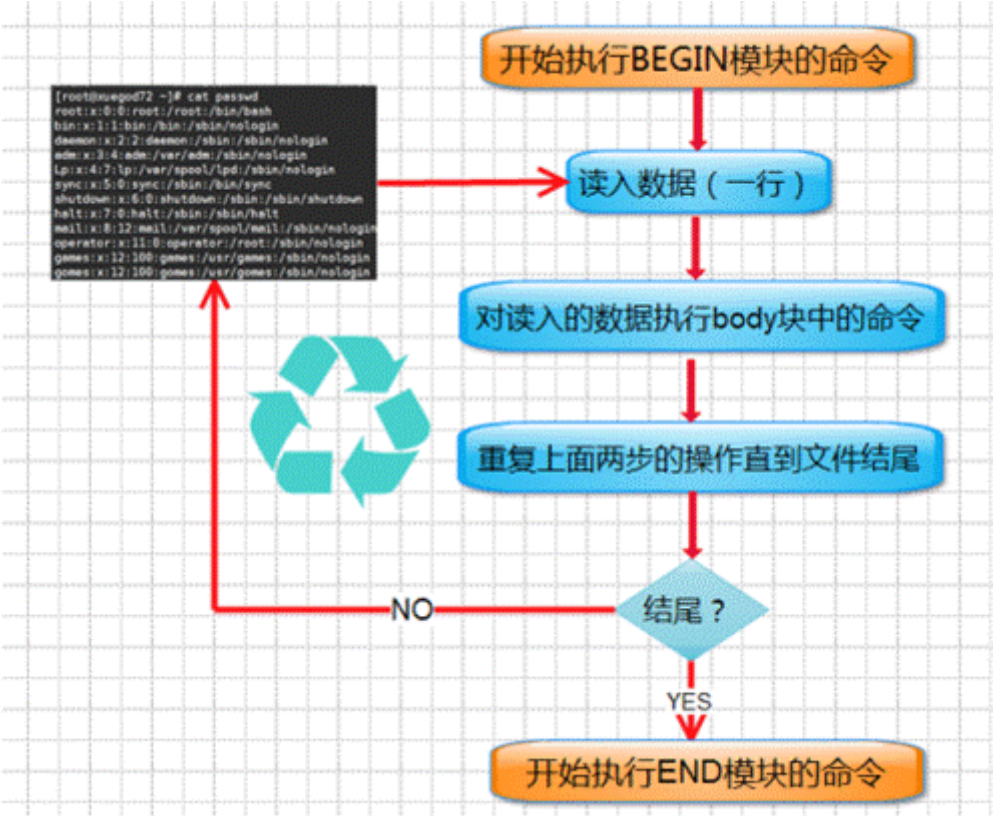
它的语法结构如下：

```
awk -F " " ' BEGIN { cmd1 }  $3>1000 {  print  $1 $2  } END{ cmd2 } ' file1.txt

    选项      |  开始模块    | 范围条件     | 操作命令主体    | 结束模块    |     文件
```

例

```
[root@exercise1 opt]# awk 'BEGIN{FS=":"}$3==0{print $0}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
[root@exercise1 opt]#
```



特殊模块：

BEGIN语句设置计数和打印头部信息，在任何动作之前进行

END语句输出统计结果，在完成动作之后执行

AWK它工作通过三个步骤

1、读：从文件、管道或标准输入中读入一行然后把它存放到内存中

2、执行：对每一行数据，根据AWK命令按顺序执行。默认情况是处理每一行数据，也可以指定模式

3、重复：一直重复上述两个过程直到文件结束

AWK支持两种不同类型的变量：内置变量，自定义变量

awk内置变量（预定义变量）(默认规则)

$n当前记录的第n个字段，比如:$1表示第一个字段，$2表示第二个字段

$0这个变量包含执行过程中当前行的文本内容

NF表示字段数，在执行过程中对应于当前的字段数，NF：列的个数

NR表示记录数，在执行过程中对应于当前的行号

FNR各文件分别计数的行号

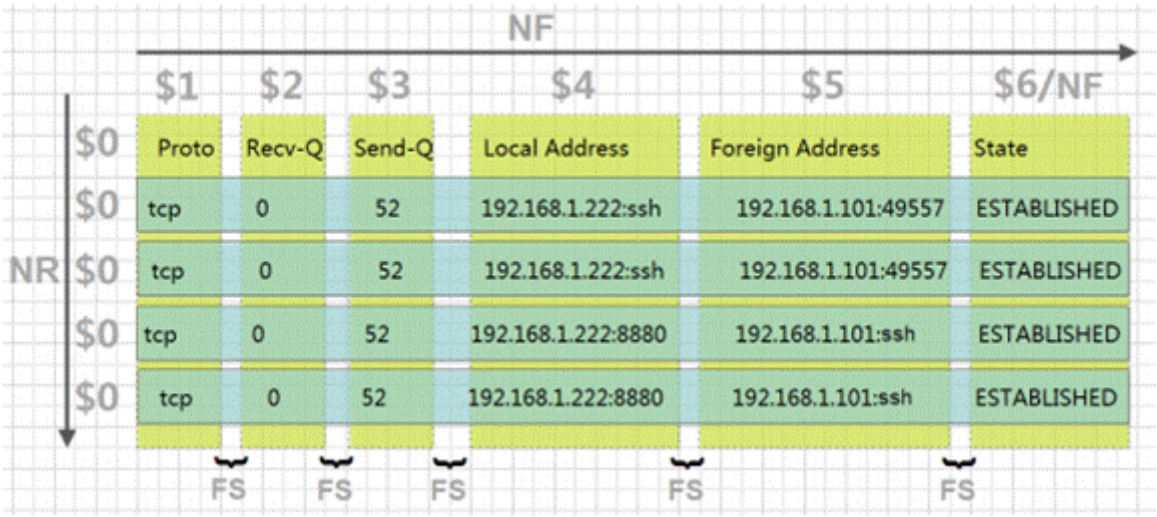
FIELDWIDTHS：定义每个数据字段宽度

FS：输入列字段分隔符（默认是空格）

RS：输入行分隔符（默认是换行符\n）

OFS：输出列字段分隔符（默认值是空格）

ORS：输出行分隔符（默认值是换行符\n）



```
[root@home opt]# awk  '{print $0}' <<< `ifconfig`
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 inet 192.168.245.129 netmask 255.255.255.0 broadcast 192.168.245.255 inet6 fe80::5bd8:5895:a4f8:73fe prefixlen 64 scopeid 0x20<link> ether 00:0c:29:7a:33:79 txqueuelen 1000 (Ethernet) RX packets 7657 bytes 720261 (703.3 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 4901 bytes 639150 (624.1 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 inet 127.0.0.1 netmask 255.0.0.0 inet6 ::1 prefixlen 128 scopeid 0x10<host> loop txqueuelen 1 (Local Loopback) RX packets 121 bytes 14110 (13.7 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 121 bytes 14110 (13.7 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@home opt]# awk  '{print $6}' <<< `ifconfig`
192.168.245.129
```

```
例子1：使用 NR 行号来定位，然后提取 IP 地址
#注：这个思路很好，之前都是通过 过滤关键字来定位，这次是通过行号，多了一种思路

[root@exercise1 opt]# ifconfig ens33 | awk 'NR==2{print $2}'
192.168.119.142
[root@exercise1 opt]#

注：NR==2 表示行号
```

例子2：NR 与 FNR 的区别

[root@exercise1 opt]# awk '{print NR "\t" $0}' /etc/hosts /etc/hostname

1 127.0.0.1 localhost localhost.localdomain localhost4

localhost4.localdomain4
2 ::1 localhost localhost.localdomain localhost6
localhost6.localdomain6
3 exercise1

[root@exercise1 opt]# awk '{print FNR "\t" $0}' /etc/hosts
/etc/hostname
1 127.0.0.1 localhost localhost.localdomain localhost4
localhost4.localdomain4
2 ::1 localhost localhost.localdomain localhost6
localhost6.localdomain6
1 exercise1

注：对于 NR 来说，在读取不同的文件时，NR 是一直加的；
对于 FNR 来说，在读取不同的文件时，它读取下一个文件时，FNR 会从 1 开始重新计算的
对于一个文件来说，NR=FNR

例子3：
[root@exercise1 opt]# awk -F ":" '{print NR,$0,"被切割成了",NF,"列"}'
/etc/passwd
1 root❌0:0:root:/root:/bin/bash 被切割成了 7 列
2 bin❌1:1:bin:/bin:/sbin/nologin 被切割成了 7 列
3 daemon❌2:2:daemon:/sbin:/sbin/nologin 被切割成了 7 列
4 adm❌3:4:adm:/var/adm:/sbin/nologin 被切割成了 7 列
5 lp❌4:7:lp:/var/spool/lpd:/sbin/nologin 被切割成了 7 列
.....

例子4：
[root@exercise1 opt]# vim b2.txt
12132154562154.468
544867897545465486
545215468789489798

[root@exercise1 opt]# awk 'BEGIN{FIELDWIDTHS="2 3 4 6"}{print 1,2,3,4}' /opt/b2.txt

12 132 1545 62154.

54 486 7897 545465

54 521 5468 789489

[root@exercise1 opt]#

例子5：

[root@exercise1 opt]# vim b3.txt

a1,b2,c3,d4

a5,b6,c7,d8

a9,b10,c11,d12

[root@exercise1 opt]# awk 'BEGIN{FS=",";OFS=" "}{print $1,$2,$3}' /opt/b3.txt

a1 b2 c3

a5 b6 c7

a9 b10 c11

[root@exercise1 opt]# awk 'BEGIN{FS=",";OFS=">"}{print 1,2,$3}' /opt/b3.txt

a1>b2>c3

a5>b6>c7

a9>b10>c11

#FS识别文本原来的字段分隔符为“，”；OFS定义输出到终端时字段分隔符用“>”显示

例子6：

[root@exercise1 opt]# awk 'BEGIN{RS="/"}{print $0}' /opt/c.txt

root❌0:0:root:

root:

bin

bash

bin❌1:1:bin:

bin:

sbin

nologin

例子7：

[root@home opt]# cat b.txt|awk 'BEGIN{RS="\n"; ORS=","}{print $0}'
a1,b2,c3,d4,a5,b6,c7,d8,a9,b10,c11,d12,

#RS识别文本原来的换行符为“\n”，ORS输出到终端时换行符用“,”代替，但“，”不能换行，所以形成一行

内置数组变量

ARGV 数组，保存命令行本身这个字符串，如awk '{print $0}' a.txt b.txt 这个命令中，ARGV[0]保存awk，ARGV[1]保存a.txt；

ARGC awk 命令数组中元素的个数；

FILENAME awk命令所处理的文件名称（以文件行数处理统计）；

ENVIRON 调用当前 shell 中 环境变量；

```
例子：
[root@exercise1 opt]# awk   '{print ARGV[0],ARGV[1],ARGV[2],ARGC,FILENAME,ENVIRON["HOSTNAME"]}'  a2.txt  b2.txt
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 a2.txt exercise1
awk a2.txt b2.txt 3 b2.txt exercise1
awk a2.txt b2.txt 3 b2.txt exercise1
awk a2.txt b2.txt 3 b2.txt exercise1
[root@exercise1 opt]#
```

用户自定义变量

定义变量和引用变量，均不需要加$符号(推荐使用小写)

```
例子：
[root@exercise1 opt]# awk 'BEGIN{var="1"}{print var}' a2.txt
1
1
```

使用选项定义变量
[root@exercise1 opt]# awk -v var=1 '{print var}' a2.txt
1
1

常用的命令选项：

-F fs指定分隔符

-v 赋值一个用户自定义变量

-f 指定脚本文件,从脚本中读取awk命令

（1）分隔符的使用

用法：-F fs 其中fs是指定输入分隔符，fs可以是字符串或正则表达式，分隔符默认是空格

常见写法：-F: -F, -F[Aa]

```
例子1：
[root@exercise1 opt]# echo "AA BB CC DD" | awk '{print $2}'
BB
[root@exercise1 opt]# echo "AA|BB|CC|DD" | awk -F "|" '{print $2}'    #因为|符号具有特殊含义，所以需要加引号
BB
```

```
[root@exercise1 opt]# echo "AA,BB,CC,DD" | awk -F "," '{print $2}'
BB
[root@exercise1 opt]# echo "AA,BB,CC,DD" | awk -F, '{print $2}'
BB
[root@exercise1 opt]# awk -F: '{print $1}' /etc/passwd   #以：分隔，打印第1列用户名
root
bin
daemon
adm
lp
sync
......
[root@exercise1 opt]#
```

例子2：指定多个分隔符

[root@exercise1 opt]# echo "12AxAbADXaAD52" | awk -F"[Aa]" '{print $6}'

D52

例子3：使用FS指定分隔符

[root@exercise1 opt]# echo "12AxAbADXaAD52" | awk 'BEGIN{FS="aA"}{print $2}'

D52

[root@exercise1 opt]#

例4：使用正则指定分隔符

[root@home opt]# echo "abaadkjiaabb" |awk -F"[a]+" '{print 1,2,$3}'

b dkji bb

[root@home opt]# echo "abaadkjiaabb" |awk -F"(aa)" '{print 1,2,$3}'
ab dkji bb

[root@home opt]# echo "abaadkjiaabb" |awk -F"[a]" '{print 1,2,3,4,5,6}'
b dkji bb
[root@home opt]# echo "abaadkjiaabb" |awk -F"[a]" '{print $1}'

[root@home opt]# echo "abaadkjiaabb" |awk -F"[a]" '{print $2}'

b

注：用“[]”作为分隔符时，字符会占一个空格位显示

例子5：过滤出本系统的IP地址
[root@exercise1 opt]# ifconfig ens33 | grep netmask
inet 192.168.119.142 netmask 255.255.255.0 broadcast 192.168.119.255
[root@exercise1 opt]# ifconfig ens33 | grep netmask | awk '{print $2}'
192.168.119.142

（2）条件的对比

数值对比支持的符号: >, <, >=, <=, ==, !=

```
例子
[root@exercise1 opt]# awk -F ":" '$3>500{print $1,$3}' /etc/passwd
polkitd 999
chrony 998
[root@exercise1 opt]#
```

字符串比对

可以通过 / 扩展RE / 进行字符串比对

```
$1  ~   / 正则 /     # 字段值匹配
$1 !~   / 正则 /     # 字段值不匹配
$1  ~    /a1|a2/      # 字段值匹配   a1或a2
以上三种属于模糊匹配
```

例子

```
例1：打印$1匹配root显示的行
[root@exercise1 opt]# awk -F: '$1 ~ /root/{print $0}'
/etc/passwd
root:x:0:0:root:/root:/bin/bash
[root@exercise1 opt]# awk -F: '($1 ~ /root/){print $0}'
/etc/passwd
root:x:0:0:root:/root:/bin/bash
[root@exercise1 opt]#
```

例2：打印$1不匹配root的行

[root@exercise1 opt]# awk -F: '$1 !~ /root/{print $0}' /etc/passwd | head -1

bin❌1:1:bin:/bin:/sbin/nologin

[root@exercise1 opt]#

例3：打印$1只要有 bin 或 root 的行

[root@exercise1 opt]# awk -F":" '($1 ~ /bin|root/){print $0}' /etc/passwd

root❌0:0:root:/root:/bin/bash

bin❌1:1:bin:/bin:/sbin/nologin

[root@exercise1 opt]#

例4：

[root@exercise1 opt]# awk -F ":" '$1 ~ /(bin|root)/{print $0}' /etc/passwd

root❌0:0:root:/root:/bin/bash

bin❌1:1:bin:/bin:/sbin/nologin
[root@exercise1 opt]#

支持逻辑运算符： &&， ||

```
例子1：取出当前系统中普通数据分区 ， 使用量超过 10% 的
[root@home opt]# df -h |  awk '$1 !~ /(tmpfs|cdrom|sr0)/ && $5 > 10 {print $1,$5}'
文件系统 已用%
/dev/sda3 20%
/dev/sda1 58%
```

例子2：打印id号是大于等于1000或者最后一列是/bin/bash的用户
[root@exercise1 opt]# awk -F: '$3>=1000 || $NF == "/bin/bash" {print $1 "\t" $3 "\t" $Nf}' /etc/passwd
root 0 root❌0:0:root:/root:/bin/bash
[root@exercise1 opt]#

例子3：匹配以root开头的行到abc结尾的行，并打印第一个字段
[root@exercise1 opt]# cat /etc/passwd | awk -F":" '/^root/,/abc$/{print $1}'
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
systemd-network
dbus

polkitd
postfix
sshd
chrony
abc
[root@exercise1 opt]#

（3）关系运算符的使用

```
例子1：
[root@exercise1 opt]# echo "3 2 3 4 5" | awk '{print $1+10}'
13
```

例子2：
[root@exercise1 opt]# echo "one two three four" | awk '{print NF}'
4
[root@exercise1 opt]# echo "one two three four" | awk '{print $NF}'
four
[root@exercise1 opt]# echo "one two three four" | awk '{print $(NF-2)}' #打印倒数第 3 列
two
[root@exercise1 opt]# echo "one two three four" | awk '{print $(NF/2-1)}' #打印第1列
one

实战：统计当前内存的使用率
[root@exercise1 opt]# vim user_cache.sh
#!/bin/bash
USEFREE=`free | awk 'NR==2{print $3/$2*100"%"}'`
echo -e "内存使用百分比： \e[32m${USEFREE}\e[0m"
[root@exercise1 opt]# sh user_cache.sh
内存使用百分比：13.627%

```
例：打印出 passwd 文件中用户 UID 小于 10 的用户名和它登录使用的shell

[root@exercise1 opt]# awk -F: '$3<10{print $1 $NF}' /etc/passwd    #直接输出格式太乱
root/bin/bash
bin/sbin/nologin
daemon/sbin/nologin
adm/sbin/nologin
lp/sbin/nologin
sync/bin/sync
shutdown/sbin/shutdown
halt/sbin/halt
mail/sbin/nologin
[root@exercise1 opt]#
```

[root@exercise1 opt]# awk -F: '$3<10{print $1 "<==========>" $NF}' /etc/passwd #awk 格式化输出

root<==>/bin/bash
bin<==>/sbin/nologin
daemon<==>/sbin/nologin
adm<==>/sbin/nologin
lp<==>/sbin/nologin
sync<==>/bin/sync
shutdown<==>/sbin/shutdown
halt<==>/sbin/halt
mail<==>/sbin/nologin
[root@exercise1 opt]#

```
例：在$1和$NF之间加一下\t    相当于多个空格
[root@exercise1 opt]# awk -F: '$3<10{print $1 "\t" $NF}' /etc/passwd
root    /bin/bash
bin /sbin/nologin
daemon  /sbin/nologin
```

```
adm /sbin/nologin
lp  /sbin/nologin
sync    /bin/sync
shutdown    /sbin/shutdown
halt    /sbin/halt
mail    /sbin/nologin
[root@exercise1 opt]#

注：awk 最外面使用了单引号'' ,里面都使用双引号",特殊符号都要加双引号
输出多个列时，可以加，分隔一下

[root@exercise1 opt]# awk -F: '$3<10{print $1,$NF}' /etc/passwd   #加，一样也是呈现空格
root /bin/bash
bin /sbin/nologin
daemon /sbin/nologin
adm /sbin/nologin
lp /sbin/nologin
sync /bin/sync
shutdown /sbin/shutdown
halt /sbin/halt
mail /sbin/nologin
[root@exercise1 opt]#
```

```
例：awk直接修改文件          #相当于sed的另存为w
[root@exercise1 opt]# awk -F: '{print $1 > "a2.txt"}' /etc/passwd
[root@exercise1 opt]# cat /opt/a2.txt
root
bin
daemon
adm
lp
sync
```

练习

```
cat >>/server/files/reg.txt<<EOF
```

```
Zhang Dandan    41117397    :250:100:175
Zhang Xiaoyu    390320151   :155:90:201
Meng  Feixue    80042789    :250:60:50
Wu  Waiwai  70271111    :250:80:75
Liu Bingbing    41117483    :250:100:175
wang    Xiaoai  3515064655  :50:95:135
zi  Gege    1986787350  :250:168:200
Li  Youjiu  918391635   :175:75:300
Lao Nanhai  918391635   :250:100:175
EOF

姓氏是Zhang的人，显示他的第二次捐款金额及她的名字

显示所有以41开头的ID号码的人的全名和工D号码

显示所有ID号码最后一位数字是1或5的人的全名

显示X1aoyu的捐款.每个值时都有以＄开头.如＄520＄200＄135
```

BEGIN

作用	
用来统计和计算	
定义变量	

例:

```
[root@home opt]# cat a.txt
12132154562154.468
544867897545465486
545215468789489798
[root@home opt]# awk '{print 1+10}' a.txt
11
11
11
[root@home opt]# awk 'BEGIN{print 1+10}'
11

注：可以看出，awk只要涉及处理文本内容，就需要在后面加文件
```

END

1.统计/etc/services中一共有多少行

```
[root@home opt]# awk '{i++;print i}' /etc/services seq
10
。。。
11170
11171
11172
11173
11174
11175
11176
11176
[root@home opt]# awk '{i++;}END{print i}' /etc/services
11176
```

2.统计/etc/services中空行的数量　(至少4种方法)

```
[root@home opt]# awk '/^$/{i++}END{print i}' /etc/services
17
[root@home opt]# grep  "^$" /etc/services|wc -l
17

```

113.70.183.203 - - [23/Dec/2021:17:15:35 +0800] "GET / HTTP/1.1" 200 486 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
113.70.183.203 - - [23/Dec/2021:17:15:35 +0800] "GET / HTTP/1.1" 200 486 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
113.70.183.203 - - [23/Dec/2021:17:15:35 +0800] "GET / HTTP/1.1" 200 486 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
113.70.183.203 - - [23/Dec/2021:17:15:35 +0800] "GET / HTTP/1.1" 200 486 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
113.70.183.203 - - [23/Dec/2021:17:15:35 +0800] "GET / HTTP/1.1" 200 486 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"

3.统计access.log中总共使用多少流量

4.统计access.log中状态码是200的次数，及状态码为200的流量总和

5.统计access.log中不同状态码的次数，及状态码为200的流量总和

扩展

条件表达式

表达式　　{条件? if-true:if-false } 问号前面是条件，如果条件为真执行if-true,为假执行if-false

```
例1：如果passwd中UID小于10，则给变量USER赋值成aaa，否则赋值成bbb
[root@exercise1 opt]# awk -F: '{$3<10? USER="aaa":USER="bbb";print $1,USER}' /etc/passwd
root aaa
bin aaa
daemon aaa
adm aaa
lp aaa
……
```

用if(条件){命令1；命令2}elif(条件){命令；}else{命令}中，在比较条件中用()扩起来，在AWK中，如果条件为1为真，0为假
例：如果UID大于10，则输出user=>用户名，否则输出pass=>用户名

```
[root@exercise1 opt]# awk -F: '{if($3<10){print "user=>"$1}else{print "pass=>"$1}}' /etc/passwd
user=>root
user=>bin
user=>daemon
user=>adm
user=>lp
user=>sync
……
```

格式化输出(了解一下)

printf命令：格式化输出 printf “FORMAT”， item1，item2.....

format使用注意事项：

1、其与print命令的最大不同是，printf需要指定format样式

2、format用于指定后面的每个item的输出格式

3、printf语句不会自动打印换行符；\n

4、format格式的指示符都以%开头，后跟一个字符；如下：
%c:显示字符的ASCII码
%d,%i：十进制整数
%e,%E：科学计数法显示数值
%f:显示浮点数
%g,%G:以科学计数法的格式或浮点数的格式显示数值；
%s:显示字符串
%u:无符号整数%%:显示%自身

```
例子1：输入passwd文件中的第1列内容，输出时不会换行
[root@exercise1 opt]# awk -F: '{printf "%s",$1}'
/etc/passwd    #不会自动换行
rootbindaemonadmlpsyncshutdownhaltmailoperatorgamesftpno
bodysystemd-networkdbuspolkitdpostfixsshdchrony
```

例子2：换行输出
[root@exercise1 opt]# awk -F: '{printf "%s\n",$1}' /etc/passwd

例子3：在输出的字母前面添加自定义字符串USERNAME:
[root@exercise1 opt]# awk -F: '{printf "USERNAME:%s\n",$1}'
/etc/passwd
USERNAME:root
USERNAME:bin
USERNAME:daemon
USERNAME:adm
USERNAME:lp
USERNAME:sync

例子4：对$1和$NF都做格式化输出
[root@exercise1 opt]# awk -F: '{printf "USERNAME: %s
%s\n",$1,$NF}' /etc/passwd
USERNAME: root /bin/bash

USERNAME: bin /sbin/nologin
USERNAME: daemon /sbin/nologin

例子5：对1和NF都做格式化输出，在1和NF两者之间添加一串==字符进行输入

[root@exercise1 opt]# awk -F: '{printf "USERNAME:%S===%s\n",1,NF}' /etc/passwd
USERNAME:%S===root
USERNAME:%S=====bin

---

awk 修饰符：

```
N: 显示宽度;
-: 左对齐;
一个字母占一个宽度。默认是右对齐

例子1：显示时用10个字符串右对齐显示。如果要显示的字符串不够10个宽度，以字符串的左边自动添加。一个字母占一个宽度。默认是右对齐
[root@exercise1 opt]# awk -F":" '{printf "%10s\n",$1}' /etc/passwd
  root
   bin
daemon
   adm
    lp
  sync
  ……
[root@exercise1 opt]#
```

例子2：使用10个宽度，左对齐显示
[root@exercise1 opt]# awk -F":" '{printf "%-10s\n",$1}' /etc/passwd
root
bin
……

例子3：第1列使用15个字符宽度左对齐输出，最后一列使用15个字符宽度右对齐输出

[root@exercise1 opt]# awk -F: '{printf "USERNAME:%-15s %15s\n",1,NF}' /etc/passwd

USERNAME:root /bin/bash

USERNAME:bin /sbin/nologin

.....

例子4：使用开始和结束模块来格式化输出

```
[root@exercise1 opt]# vim /opt/test2.awk
BEGIN{
print "UserId\t\t\tShell"
print "-------------------------------"
FS=":"
}
$3>=500 && $NF=="/sbin/nologin"{
printf "%-20s %-20s\n", $1,$NF
}
END{
print "-------------------------------"
}
[root@exercise1 opt]# awk -f test2.awk /etc/passwd
UserId            Shell
-------------------------------
polkitd             /sbin/nologin
chrony              /sbin/nologin
-------------------------------
```

使用shellcheck工具检测脚本　　#亲测有BUG，一些有错误的地方不显示报错

```
安装
[root@exercise1 opt]# yum install -y epel-release
[root@exercise1 opt]# yum install ShellCheck
```

创建测试脚本

[root@exercise1 opt]# cat a.sh

#!/bin/bash

if[$# -eq 0]

then

    echo"no para"

else

    echo"$# para"

fi

exit 0

检测

[root@exercise1 opt]# shellcheck a.sh