

课程目标

- 掌握case语句的基本语法结构
- 掌握函数的定义及调用
- 掌握常用的正则表达式元字符含义

一、case语句

case语句为多选择语句。可以用case语句匹配一个值与一个模式，如果匹配成功，执行相 匹配的命令。

```
case var in
    pattern 1)
        command1
        ;;
    pattern 2)
        command2
        ;;
    pattern 3)
        command3
        ;;
    *)
        command4
        ;;
esac
```

定义变量; var代表是变量名
模式1; 用 | 分割多个模式, 相当于or
需要执行的语句
两个分号代表命令结束
default, 不满足以上模式, 默认执行*) 下面的语句
esac表示case语句结束

案例1

当给程序传入start、stop、reload三个不同参数时分别执行相应命令。

```
#!/bin/bash
case $1 in
    start|S)
        echo "service is running...."
        ;;
    stop|T)
        echo "service is stoped..."
        ;;
    reload|R)
        echo "service is restart..."
        ;;
    *)
        echo "请输入你要的动作"
        ;;
esac

#!/bin/bash
case $1 in
    start|S)
```

```

read -p "Input your service name:" name
case $name in
    apache)
        service apache start
        echo "$name service is running...."
        ;;
    mysql)
        service mysql25 start
        echo "$name service is running..."
        ;;
esac
;;
stop|P)
echo "service is stoped...."
;;
reload|restart)
echo "service is restart..."
;;
*)
echo "the $0 usage:$0 start|stop|reload..."
;;
esac

#!/bin/bash
#up service
case $1 in
    start)
        read -p "Input your start service[vsftpd]:" var1
        case $var1 in
            vsftpd)
                service vsftpd start
                ;;
            apache)
                service apache start
                ;;
            '')
                true
                ;;
            *)
                echo "请输入你要启动的服务，谢谢"
                ;;
esac
;;
stop|S)
service iptables stop
;;
reload|restart)
service iptables restart
;;
*)
echo "Input your action..."
;;

```

```

esac

#!/bin/bash
case $1 in
    start|S)
        read -p "Input your service name:" names
        case $names in
            apache|httpd)
                service apache start
                ;;
            vsftpd|ftp)
                service vsftpd start &>/dev/null
                pgrep $names &>/dev/null
                if [ $? -eq 0 ];then
                    echo "$names服务启动成功"
                else
                    echo "$names服务启动失败"
                fi
                ;;
            *)
                echo "请输入你需要启动的服务名."
                ;;
        esac
        ;;
    stop|T)
        echo "service is stoped..."
        ;;
    reload|R)
        echo "service is restart..."
        ;;
    *)
        echo "请输入你要的动作"
esac

```

案例2

模拟一个多任务维护界面。当执行程序时先显示总菜单，然后进行选择后做相应维护监控操作。

```

h   显示命令帮助
f   显示磁盘分区
d   显示磁盘挂载
m   查看内存使用
u   查看系统负载
q   退出程序

```

分析：

1. 打印菜单
2. 等待用户输入需要的操作编号 case语句 read定义变量

```
echo "h 显示命令帮助"
```

```
...
```

```
cat <<EOF
```

```
h    显示命令帮助
```

```
f    显示磁盘分区
```

```
d    显示磁盘挂载
```

```
m    查看内存使用
```

```
u    查看系统负载
```

```
q    退出程序
```

```
EOF
```

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

```
#打印菜单
```

```
cat <<-EOF
```

```
h    显示命令帮助
```

```
f    显示磁盘分区
```

```
d    显示磁盘挂载
```

```
m    查看内存使用
```

```
u    查看系统负载
```

```
q    退出程序
```

```
EOF
```

```
#让用户输入需要的操作
```

```
while true
```

```
do
```

```
read -p "请输入需要操作的选项[f|d]:" var1
```

```
case $var1 in
```

```
h)
```

```
cat <<-EOF
```

```
h    显示命令帮助
```

```
f    显示磁盘分区
```

```
d    显示磁盘挂载
```

```
m    查看内存使用
```

```
u    查看系统负载
```

```
q    退出程序
```

```
EOF
```

```
;;
```

```
f)
```

```
fdisk -l
```

```
;;
```

```
d)
```

```
df -h
```

```
;;
```

```
m)
```

```
free -m
```

```
;;
```

```
u)
```

```
uptime
```

```
;;
```

```

        q)
        exit
    ;;
esac
done

#!/bin/bash
#打印菜单
menu(){
cat <<-END
    h    显示命令帮助
    f    显示磁盘分区
    d    显示磁盘挂载
    m    查看内存使用
    u    查看系统负载
    q    退出程序
    END
}
menu
while true
do
read -p "请输入你的操作[h for help]:" var1
case $var1 in
    h)
        menu
        ;;
    f)
        read -p "请输入你要查看的设备名字[/dev/sdb]:" var2
        case $var2 in
            /dev/sda)
                fdisk -l /dev/sda
                ;;
            /dev/sdb)
                fdisk -l /dev/sdb
                ;;
        esac
        ;;
    d)
        lsblk
        ;;
    m)
        free -m
        ;;
    u)
        uptime
        ;;
    q)
        exit
        ;;
esac
done

```

课堂练习1

1. 输入一个等级（A-E），查看每个等级的成绩；如：输入A，则显示“90分~100分”，依次类推
2. 模拟2人第一次相亲的场景，使用read让用户输入它的名字，性别，年龄（年龄放在性别判断后）；在case里面再嵌套case菜单，使之选项更丰富。

要求： 1)

- 对性别进行判断，如果不输入男或者女，则显示“你是泰国来的吗？”
- 如果是男的，对其年龄进行判断。

2)

- 如果男的年龄大于等于18岁则显示“某某先生，你结婚了吗？”；
- 如果对方回答结了或者yes，则显示“结了你来这凑什么热闹”；
- 如果对方回答没有或者no，再次询问“那你有房有车吗？”；
- 如果既不说结了也不说没结则显示：“你到底结没结婚啊？”
- 如果回答有房有车，则显示“咱去民政局领证吧”；
- 如果回答没有，则显示“不好意思，我去下洗手间。”；
- 如果既不说有又不说没有，则显示“别浪费时间，请正面回答”。
- 如果男的年龄小于18岁，则显示“某某某你个小毛孩也来这凑热闹啦”

3) 如果是女的，并且年龄大于等于18岁，则显示“某某女士你好”；否则显示“某某小姐你好”

参考：

```
#!/bin/bash
read -p "输入你的姓名:" name
read -p "输入你的性别:" gender

case "$gender" in
    男|man|male|boy )
        read -p "输入你的年龄:" age
        if [ $age -ge 18 ];then
            read -p "$name先生,你结婚了吗?" answer
            case "$answer" in
                结了|有|yes )
                    echo "结了你来干嘛?"
                    ;;
                没结|没有|没|no )
                    read -p "有房有车吗?" answer2
                    case "$answer2" in
                        有)
                            echo "咱就直接去民政局领证吧"
                            ;;
                        没有 )
                            echo "不好意思，我去下洗手间"
                            ;;
                        * )
                            echo "别浪费时间，请正面回答"
                        ;;
                    esac
                ;;
            esac
        fi
    esac
```

```

                esac
                ;;
            * )
                echo "你到底结没结?"
            esac

        else
            echo "$name小子"
        fi
        ;;
    女|woman|female|girl|lady )
        read -p "输入你的年龄:" age
        if [ $age -ge 18 ];then
            echo "$name女士"
        else
            echo "$name小姐"
        fi
        ;;
    * )
        echo "你是泰国来的吗?"
    esac

```

该程序有个bug：如果输入年龄为负数或者0也是可以的，如何修复bug？增加一个条件：如果输入的年龄小于等于10则显示：“不跟你玩了。。。“

在最后加入如下语句即可：

```

if [ $age -ge 18 ];then
    echo "$name女士"
elif
    [ $age -le 10 ];then
        echo "不跟你玩了"
        exit 1
    else
        echo "$name小姐"
    fi
    ;;
* )
    echo "你是泰国来的吗?"
esac

```

二、函数

变量名=变量值

函数——>命令集合 实现某个功能 eg:批量创建5个用户

shell中允许将**一组命令集合或语句**形成一段**可用代码**，这些代码块称为shell函数。给这段代码起个名字称为函数名，后续可以直接调用该段代码的功能。

语法结构：

```
函数名()  
{  
    函数体（一堆命令的集合，来实现某个功能）  
}
```

```
function 函数名()  
{  
    函数体（一堆命令的集合，来实现某个功能）  
}
```

```
function_name() {  
    command  
    command  
}
```

```
function function_name() {  
    command  
    command  
}
```

函数中return说明:

1. **return**可以结束一个函数，类似于前面讲的循环控制语句**break**(结束当前循环，执行循环体后面的代码)
2. **return**默认返回函数中最后一个命令的退出状态，也可以给定参数值，该参数值的范围是**0-256**之间。
3. 如果没有**return**命令，函数将返回最后一个Shell的退出值。

1. 函数的调用

1.1 当前命令行调用

```
[root@MissHou shell04]# cat fun1.sh  
#!/bin/bash  
hello(){  
    echo "hello lilei $1"  
    hostname  
}  
menu(){  
    cat <<-EOF  
    1. mysql  
    2. web  
    3. app  
    4. exit  
    EOF  
}  
  
[root@MissHou shell04]# source fun1.sh  
[root@MissHou shell04]# . fun1.sh  
  
[root@MissHou shell04]# hello 888  
hello lilei 888
```



```
MissHou.itcast.cc
[root@MissHou shell04]# menu
1. mysql
2. web
3. app
4. exit
```

1.2 定义到用户的环境变量中

```
/etc/profile    /etc/bashrc    ~/.bash_profile ~/.bashrc
```

```
[root@MissHou shell04]# cat ~/.bashrc
# .bashrc
```

```
# User specific aliases and functions
```

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

```
hello(){
echo "hello lilei $1"
hostname
}
menu(){
cat <<-EOF
1. mysql
2. web
3. app
4. exit
EOF
}
```

注意：
当用户打开bash的时候会读取该文件

1.3 脚本中调用

```
#!/bin/bash
#打印菜单
source ./fun1.sh
menu(){
cat <<-END
    h    显示命令帮助
    f    显示磁盘分区
```

```

d    显示磁盘挂载
m    查看内存使用
u    查看系统负载
q    退出程序
END
}
menu    //调用函数

```

2. 应用案例

需求：写一个脚本让用户输入基本信息(姓名，性别，年龄)，如不输入一直提示输入，最后根据用户的信息输出对应的内容

思路：

- 循环直到输入字符串不为空 （该功能可以定义为一个函数，方便下面脚本调用）
- 根据用户输入信息做出匹配判断 case 语句

```

#!/bin/bash
#该函数实现用户如果不输入内容则一直循环直到用户输入为止，并且将用户输入的内容打印出来
input_fun()
{
    input_var=""
    output_var=$1
    while [ -z $input_var ]
    do
        read -p "$output_var" input_var
    done
    echo $input_var
}

或者
#!/bin/bash
fun(){
    read -p "$1" name
    if [ -z $name ];then
        fun $1
    else
        echo $name
    fi
}

#调用函数并且获取用户的姓名、性别、年龄分别赋值给name、sex、age变量
name=$(input_fun 请输入你的姓名:)
sex=$(input_fun 请输入你的性别:)
age=$(input_fun 请输入你的年龄:)

#根据用户输入的性别进行匹配判断
#根据用户所输入的内容进行判断输入
case $sex in

```

```

man|男)
if [ $age -ge 18 -a $age -le 25 ];then
    echo "青少年，活力无限...，你可以茫然你也可以迷茫，但是要好好学习，不要挥霍浪费你的资本"
elif [ $age -gt 25 -a $age -le 35 ];then
    echo "中年人，压力好大，但是家庭幸福的人儿"
elif [ $age -lt 18 ];then
    echo "小伙子不错"
else
    echo "$name先生，你油腻了吗?"

fi
;;
woman|女)
echo "$name 小姐姐你好漂亮"
;;
*)
echo "你是泰国来的吗?"
;;

esac

```

描述以下代码含义：

```

:()
{
    :|:&
}
:

:(){:|:&}:

```

三、综合案例

任务/背景：

现有的跳板机虽然实现了统一入口来访问生产服务器，yunwei用户权限太大可以操作跳板机上的所有目录文件，存在数据被误删的安全隐患，所以希望你做一些安全策略来保证跳板机的正常使用。

具体要求：

1. 只允许yunwei用户通过跳板机远程连接后台的应用服务器做一些维护操作
2. 公司运维人员远程通过yunwei用户连接跳板机时，跳出以下菜单供选择：

欢迎使用Jumper-server，请选择你要操作的主机：

1. DB1-Master
2. DB2-Slave
3. Web1
4. Web2
- h. help
- q. exit

3. 当用户选择相应主机后，直接**免密码登录**成功
4. 如果用户不输入一直提示用户输入，直到用户选择退出

思路：

1. 需要当yunwei用户登录时执行一个脚本（该脚本放到哪里？）yunwei用户的家目录里的.bashrc
2. 脚本中需要打印一个菜单供用户选择（case..esac 和函数）
3. 免密码登录（昨天推公钥的脚本）

```
#!/bin/bash
# jumper-server
# 定义函数
# 定义菜单打印功能的函数
menu()
{
    cat <<-EOF
    欢迎使用Jumper-server，请选择你要操作的主机：
    1. DB1-Master
    2. DB2-Slave
    3. Web1
    4. Web2
    h. help
    q. exit
    EOF
}

while true
do
    # 调用函数来打印菜单
    clear
    menu
    # 菜单选择，case...esac语句
    read -p "请选择你要访问的主机：" host
    case $host in
        1)
            ssh root@10.1.1.2
            ;;
        2)
            ssh root@10.1.1.3
            ;;
        3)
            ssh root@10.1.1.2
            ;;
        h)
    
```

```
menu
;;
q)
exit
;;
esac
done
```

将脚本放到yunwei用户家目录里的.bashrc里执行:

```
bash ~/jumper-server.sh
exit
```

```
#!/bin/bash
#公钥推送成功
trap '' 1 2 3 19
#打印菜单用户选择
```

```
menu(){
cat <<-EOF
欢迎使用Jumper-server, 请选择你要操作的主机:
1. DB1-Master
2. DB2-Slave
3. Web1
4. Web2
h. help
q. exit
EOF
}
```

#调用函数来打印菜单

```
menu
while true
do
read -p "请输入你要选择的主机[h for help]: " host
```

#通过case语句来匹配用户所输入的主机

```
case $host in
1|DB1)
ssh root@10.1.1.1
;;
2|DB2)
ssh root@10.1.1.2
;;
3|web1)
ssh root@10.1.1.250
;;
h|help)
clear;menu
;;
q|quit)
exit
;;
```

```

esac
done

#!/bin/bash
#jumper-server
#菜单打印
trap '' 1 2 3
while true
do
cat <<-END
欢迎使用Jumper-server, 请选择你要操作的主机:
1. DB1-Master
2. DB2-Slave
3. Web1
4. Web2
5. exit
END
#让用户选择相应的操作
read -p "请输入你要操作的主机: " host
case $host in
    1)
        ssh root@10.1.1.2
        ;;
    2)
        ssh root@10.1.1.3
        ;;
    3)
        ssh root@10.1.1.4
        ;;
    5)
        exit
        ;;
    *)
        clear
        echo "输入错误, 请重新输入..."
        ;;
esac
done

自己完善功能:
1. 用户选择主机后, 需要事先推送公钥; 如何判断公钥是否已推
2. 比如选择web1时, 再次提示需要做的操作, 比如:
clean log
重启服务
kill某个进程

```

补充:

1) SIGHUP	重新加载配置
2) SIGINT	键盘中断^C
3) SIGQUIT	键盘退出
9) SIGKILL	强制终止
15) SIGTERM	终止（正常结束），缺省信号
18) SIGCONT	继续
19) SIGSTOP	停止
20) SIGTSTP	暂停^Z

四、正则表达式

1. 什么是正则表达式

正则表达式（Regular Expression、regex或regexp，缩写为RE），也译为正规表示法、常规表示法，是一种字符模式，用于在查找过程中匹配指定的字符。

许多程序设计语言都支持利用正则表达式进行**字符串操作**。例如，在Perl中就内建了一个功能强大的正则表达式引擎。

正则表达式这个概念最初是由Unix中的工具软件（例如sed和grep）普及开的。

支持正则表达式的程序如：locate | find | vim | grep | sed | awk

2. 第一类正则

- 名词解释：

元字符:指那些在正则表达式中具有**特殊意义的专用字符**,如:点(.) 星(*) 问号(?)等

前导字符：即位于元字符前面的字符 ab==**c**==* aoo==**o**==.

- 正则中常用的元字符

示例文本：

```
[root@server ~]# cat 1.txt
ggle
gogle
google
gooogle
gooooooogle
gooooooooooogle
taobao.com
taotaobaobao.com

jingdong.com
dingdingdongdong.com
10.1.1.1
Adfjd8789JHfdsdf/
a87fdjfkdkLJK
7kdjfd989KJK;
bSKJjkkdsdjf878.
cidufKJHJ6576,
```

```
hello world
helloworld yourself
```

(1) . 任意单个字符，除了换行符
(2) * 前导字符出现0次或连续多次 ab*能匹配“a”，“ab”以及“abb”，但是不匹配“cb”
grep abcd* 1.txt abc abcd abcd abcd abcd abcd abcd

```
grep abc*
abc
ab
abcc
acb
```

(3) .* 任意长度的字符 ab.* ab888 ab abc abcc bb
(4) ^ 行的开头
(5) \$ 行的结尾
(6) ^\$ 空行

(7) [] 匹配指定字符组内的任一单个字符 [abc]
(8) [^] 匹配不在指定字符组内的任一字符 [^abc]
(9) ^[] 匹配以指定字符组内的任一字符开头 ^[abc]
(10) ^[^] 匹配不以指定字符组内的任一字符开头 ^[^abc]

(11) \< 取单词的头
(12) \> 取单词的尾
(13) \<\> 精确匹配符号

(14) \{n\} 匹配前导字符连续出现n次 go\{2\} google gooogle
(15) \{n,\} 匹配前导字符至少出现n次
(16) \{n,m\} 匹配前导字符出现n次与m次之间

(17) \(\) 保存被匹配的字符

将192.168.0.254 换成 192.168.1.254

```
vim 1.txt
:%s#\ (192\ .168\ )\ .0\ .\ (254\ )#\ 1\ .100\ .\ 2   //底行模式下匹配
将10.1.1.1替换成10.1.1.254
:%s#\ (10\ .1\ .1\ )\ .1#\ 1\ .254#g
:%s/\ (10\ .1\ .1\ )\ .1/\ 1\ .254/g

# sed -n 's#\ (192\ .168\ )\ .0\ .254#\ 1\ .1\ .254#p'
```

找出含有192.168的行，同时保留192.168并标记为标签1，之后可以使用\1来引用它。最多可以定义9个标签，从左边开

始编号，最左边的是第一个。

```
[root@server shell05]# sed -n 's#10.1.1.1#10.1.1.254#p' 1.txt
10.1.1.254
[root@server shell05]# sed -n 's#\ (10.1.1\) .1# \1.254#p' 1.txt
10.1.1.254
```

将helloworld yourself 换成hellolilei myself

```
vim 1.txt
:%s#\ (hello\) world your\ (self\)# \1lilei my\2#g
```

```
# sed -n 's/\ (hello\) world your\ (self\) / \1lilei my\2/p' 1.txt
hellolilei myself
```

```
[root@server shell05]# sed -n 's/helloworld yourself/hellolilei myself/p' 1.txt
hellolilei myself
[root@server shell05]# sed -n 's/\ (hello\) world your\ (self\) / \1lilei my\2/p' 1.txt
hellolilei myself
```

[0-9] [a-z] [A-Z] [a-zA-Z] [a-Z]

Perl内置正则：

```
\d      匹配数字 [0-9]
\w      匹配字母数字下划线[a-zA-Z0-9_]
\s      匹配空格、制表符、换页符[\t\r\n]
```

```
#grep -P '\d' test.txt
#grep -P '\w' test.txt
#grep -P '\s' test.txt
```

- 扩展类的正则表达式 grep -E 或则 egrep

扩展正则表达式元字符

+	匹配一个或多个前导字符	bo+ boo bo
?	匹配零个或一个前导字符	bo? b bo
a b	匹配a或b	
()	组字符	hello myself yourself (my your)self

{n}	前导字符重复n次	\{n\}
{n,}	前导字符重复至少n次	\{n,\}
{n,m}	前导字符重复n到m次	\{n,m\}

```
# grep "root|ftp|adm" /etc/passwd
# egrep "root|ftp|adm" /etc/passwd
# grep -E "root|ftp|adm" /etc/passwd
```

```
# grep -E 'o+gle' test.txt
```

```
# grep -E 'o?gle' test.txt

# egrep 'go{2,}' 1.txt
# egrep '(my|your)self' 1.txt

[root@jumper shell06]# grep '[0-9]\{2\}\.[0-9]\{1\}\.[0-9]\{1\}\.[0-9]\{1\}' 1.txt
10.1.1.1
[root@jumper shell06]# grep '[0-9]{2}\.[0-9]{1}\.[0-9]{1}\.[0-9]{1}' 1.txt
[root@jumper shell06]# grep -E '[0-9]{2}\.[0-9]{1}\.[0-9]{1}\.[0-9]{1}' 1.txt
10.1.1.1
[root@jumper shell06]# grep -E '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' 1.txt
10.1.1.1
[root@jumper shell06]# grep -E '([0-9]{1,3}\.){3}[0-9]{1,3}' 1.txt
10.1.1.1
```

3. 第二类正则

表达式	功能	示例
[[:alnum:]]	字母与数字字符	[[:alnum:]]+
[[:alpha:]]	字母字符(包括大小写字母)	[[:alpha:]]{4}
[[:blank:]]	空格与制表符	[[:blank:]]*
[[:digit:]]	数字	[[:digit:]]?
[[:lower:]]	小写字母	[[:lower:]]{4,}
[[:upper:]]	大写字母	[[:upper:]]+
[[:punct:]]	标点符号	[[:punct:]]
[[:space:]]	包括换行符，回车等在内的所有空白	[[:space:]]+

```
[root@server shell05]# grep -E '^[[[:digit:]]]+' 1.txt
[root@server shell05]# grep -E '^^[[:digit:]]+' 1.txt
[root@server shell05]# grep -E '[[[:lower:]]]{4,}' 1.txt
```

4. 课堂作业

在自己虚拟机里创建如下内容的文件：

```
# cat test.txt
Aieur45869Root0000
9h847RkjfkIIIhello
```

```
rootHllow88000dfjj
8ikuioerhfhuplioking
hello world
192.168.0.254
welcome to uplooking.
abcderfkdfkdtest
rlllA899kdfkdfj
iiiA848890ldkfjdkfj
abc
12345678908374
123456@qq.com
123456@163.com
abcdefg@itcast.com23ed
```

要求如下：

1、查找不以大写字母开头的行（三种写法）。

```
grep ^[^A-Z] 1.txt
```

```
grep ^[^:upper:] 1.txt
```

```
grep -v ^[A-Z] 1.txt
```

2、查找有数字的行（两种写法）

```
grep [0-9] 1.txt
```

```
grep [[:digit:]] 1.txt
```

```
grep -P '\d' 1.txt
```

3、查找一个数字和一个字母连起来的

```
grep -E '[0-9][a-zA-Z]|[a-zA-Z][0-9]' 1.txt
```

4、查找不以r开头的行

```
grep ^[^r] 1.txt
```

```
grep -v ^r 1.txt
```

5、查找以数字开头的

```
grep ^[0-9] 1.txt
```

6、查找以大写字母开头的

```
grep ^[A-Z] 1.txt
```

7、查找以小写字母开头的

```
grep ^[a-z] 1.txt
```

8、查找以点结束的

```
grep '\.$' 1.txt
```

9、去掉空行

```
grep -nv ^$ 1.txt
```

10、查找完全匹配abc的行

```
grep -w 'abc' 1.txt
```

```
grep '^abc$' 1.txt
```

```
grep '\<abc\>' 1.txt
```

11、查找A后有三个数字的行

```
grep 'A[0-9]\{3\}' 1.txt
```

```
grep -E 'A[0-9]{3}' 1.txt
```

```
grep -P 'A\d{3}' 1.txt
```

12、统计root在/etc/passwd里出现了几次

```
grep -o root /etc/passwd|wc -l
```

13、用正则表达式找出自己的IP地址、广播地址、子网掩码

```
ifconfig eth0|grep Bcast|grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}'
ifconfig eth0|grep Bcast| grep -E -o '([0-9]{1,3}.){3}[0-9]{1,3}'
ifconfig eth0|grep Bcast| grep -P -o '\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}'
ifconfig eth0|grep Bcast| grep -P -o '(\d{1,3}.){3}\d{1,3}'
ifconfig eth0|grep Bcast| grep -P -o '(\d+.){3}\d+'
```

```
# egrep --color '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' /etc/sysconfig/network-
scripts/ifcfg-eth0
IPADDR=10.1.1.1
NETMASK=255.255.255.0
GATEWAY=10.1.1.254
```

```
# egrep --color '[:digit:]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}'
/etc/sysconfig/network-scripts/ifcfg-eth0
IPADDR=10.1.1.1
NETMASK=255.255.255.0
GATEWAY=10.1.1.254
```

14、找出文件中的ip地址并且打印替换成172.16.2.254

```
grep -o -E '([0-9]{1,3}\.){3}[0-9]{1,3}' 1.txt |sed -n 's/192.168.0.(254)/172.16.2.1/p'
```

15、找出文件中的ip地址

```
grep -o -E '([0-9]{1,3}\.){3}[0-9]{1,3}' 1.txt
```

16、找出全部是数字的行

```
grep -E '^[0-9]+$' test
```

17、找出邮箱地址

```
grep -E '^([0-9]+@[a-z0-9]+\.[a-z]+)$'
```

grep --help:

匹配模式选择:

Regex selection and interpretation:

-E, --extended-regexp	扩展正则
-G, --basic-regexp	基本正则
-P, --perl-regexp	调用perl的正则
-e, --regexp=PATTERN	use PATTERN for matching
-f, --file=FILE	obtain PATTERN from FILE
-i, --ignore-case	忽略大小写
-w, --word-regexp	匹配整个单词

课后作业

写一个自动搭建apache服务的脚本，要求如下： 1、用户输入web服务器的IP、域名以及数据根目录 2、如果用户不输入则一直提示输入，直到输入为止 3、当访问www.test.cc时可以访问到数据根目录里的首页文件“this is test page”

参考:

```
#!/bin/bash
conf=/etc/httpd/conf/httpd.conf
input_fun()
```

```

{
    input_var=""
    output_var=$1
    while [ -z $input_var ]
    do
        read -p "$output_var" input_var
    done
    echo $input_var
}

ipaddr=$(input_fun "Input Host ip[192.168.0.1]:")
web_host_name=$(input_fun "Input VirtualHostName [www.test.cc]:")
root_dir=$(input_fun "Input host Documentroot dir:[/var/www/html]:")

[ ! -d $root_dir ] && mkdir -p $root_dir
chown apache.apache $root_dir && chmod 755 $root_dir
echo this is $web_host_name > $root_dir/index.html
echo "$ipaddr $web_host_name" >> /etc/hosts

[ -f $conf ] && cat >> $conf <<end
NameVirtualHost $ipaddr:80
<VirtualHost $ipaddr:80>
    ServerAdmin webmaster@$web_host_name
    DocumentRoot $root_dir
    ServerName $web_host_name
    ErrorLog logs/$web_host_name-error_log
    CustomLog logs/$web_host_name-access_log common
</VirtualHost>
end

```