一　find(查找命令)

由于 Linux 系统大部分都是应用于服务器上，所以查询数据的时候并没有图形化界面查找数据的便捷性。

1　语法(find [path] [<选项> <参数>])

2　path

表示在哪个路径查找，如果省略，则默认在当前目录查找

3　选项

按条件查找，在使用时可通过感叹号！取反

-name	按文件名查询
-perm	按文件权限查询
-size	按文件的大小查找
-user	按用户(属主)查询
-group	按用户组查询
-type	按文件的类型查询 b - 块设备文件。 d - 目录。 c - 字符设备文件。 p - 管道文件。 l - 符号链接文件。　（这是个小写的 L） f - 普通文件。

4　参数

指定选项时，必需给选项传递参数，比如按 name 查找，则需要传递你要查找哪个 name

5　案例

查找 jdk 安装的目录。

```
# bash
```

```
find / -name java
```

查找当前系统中所有的.log 后缀名的文件

```
# bash
find / -name "*.log"
```

查找系统中/home 目录下的非普通文件

```
# bash
find /home ! -type f
```

查找当前用户/home 目录下权限为 700 的文件

```
#bash
find /home -perm 700
```

查找/dev 目录下的块设备文件

```
#bash
find /dev -type b
```

二　sed(逐行处理工具)

sed 本身是一个逐行处理工具，会逐行处理到文件的结束。默认情况下不修改源文件，因为 sed 是将源文件内容逐行 copy 到一个临时缓冲区(模式空间)，对其进行编辑，行处理结束后，将其输出到屏幕上，也可以通过数据重定向将结果导入到新的文件中去。

sed 本身提供修改源文件的选项。但是如果修改源文件时，结果内容并不会发送到屏幕上。

shell 中 sed 可以自动执行，而 vim(半自动)需要与用户交互

1　语法(sed [option] "<action]>" <filename>)

```
sed [option] "<action]>" <filename>
```

2 option(选项)

-e	允许对输入数据应用多条 sed 命令进行编辑。 一条语句执行多个命令?
-i	表示直接操作源文件

3 action(动作)

s:	字符串匹配/查找
i:	插入
a:	追加
d:	删除
c:	替换

■ 注意：选项和动作的字母 i 不是同样的功能。

4 filename(文件)

要修改的文件

5 案例

action 的的 / 也可用 |替代

5.A 将全文的 h 替换为 H。

S 表示查找字符串 g 表示全局

```
# bash
    sed "s/h/H/g" demo
```

5.B 修改全文的 h/H，第一个 l/L

```
# bash
    sed -e "s/h/H/g" -e "s/l/L/1" demo
```

Or # 多条指令的另外一种写法：

```
sed "s/h/H/g;s/l/L/1" demo
```

5.C 修改全文的第一个和第二个 h/H

注：第一个命令执行后，原有的第二个变为第一个

```
# bash
    sed "s/h/H/1;s/h/H/1" demo
```

5.D 将文件中的 ONBOOT=no 修改为 ONBOOT=yes

```
# bash
    sed -i "s/ONBOOT=no/ONBOOT=yes/g" eth0
```

5.E 在文件中进行插入新的内容：

源文件内容：

```
hello  teduhadoop
hello  hadoop
hello  hdfs ，  hi  sed
```

5.1、在第一行插入内容"hello bigdata"

```
# bash
    sed "1 i hello bigdata" demo
```

5.2、在第一行追加内容"hello 小强"

```
# bash
    sed "1 a hello 小强" demo
```

5.F 删除匹配的行

```
# bash
    sed "/hdfs/d" demo
```

表示将匹配的到 hdfs 的整行删除，如果匹配的内容是 h 的话，当前文件的内容会被全部删除。

三　grep(强大文本搜索工具选项)

这是一款强大文本搜索工具选项：

-number	同时显示匹配行上下的 n 行
-b，--byte-offset	印匹配行前面打印该行所在的块号码。
-c,--count	只打印匹配的行数，不显示匹配的内容。
-i，--ignore-case	忽略大小写差别。
-q，--quiet	取消显示，只返回退出状态。0 则表示找到了匹配的行。
--color	将匹配内容上色区分
-n，--line-number	在匹配的行前面打印行号。
-v，--revert-match	反检索，只显示不匹配的行。

```
#bash
    cd /root
    ls |grep -2 log  #匹配上下两行
     查找目录 包含 log 的，然后把匹配到的行的上下二行也显示
    ls |grep -n log  #匹配行号
    ls | grep  log --color
```

四　tail (文本监控)

文本监控，通常情况下用于监视文件的增长。

1　场景：

大数据环境中，很多软件在启动时，不会将真正的启动/过程日志打印在屏幕上，因为内容繁多，会影响程序员观察启动过程中，哪个进程没有启动。像此种场景，我们就可以利用 tail 工具用来监视该软件启动日志文件的实际内容。

2　语法（tail [选项] fileName）

tail [选项] fileName

3　选项：

-f	用于监控文件的增长！！！
-n	从指定的行中进行监控文件的内容

五　cut(文件中剪切数据)

不会改变源文件，只是显示出来
cut 命令在文件中负责剪切数据用的

1　选项：

-b	字节
-c	字符
-f	提取第几列
-d	按指定分隔符分割列

2　案例：

源文件内容：

```
192.168.1.1
192.168.1.3
192.168.1.5
192.168.1.4
```

截取第 11 个字节：

```
# bash
```

```
cut -b 11 demo
```

输出: 1

3

5

4

截取第 7-9 的字节

```
# bash
    cut -b 7-9 demo
```

截取最后一个字节进行排序:

```
# bash
    cut -b 11 demo | sort
```

输出:

1

3

4

5

以点为分隔符 获取第二个字段

```
# bash
    cut -d . -f 2 demo
```

六　history(查看 Linux 中曾经执行过的命令)

history

该命令可以用来查看 Linux 系统中曾经执行过的命令(默认 1000 条)。

用法:

!!	运行上一条命令
!88	运行第 88 条命令

!88 /test	运行第 88 条命令并在命令后面加上/test （可以用 ls 举例）
!ls	运行上一个 ls 命令
!ls:s/CF/l	运行上一个 ls 命令，其中把 CF 替换成 l
history -c	表示清除历史命令 # 学习阶段不要使用此命令，会清空历史命令，不利于学习

七 shell 脚本

文件以#!/bin/bash 开头

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

1 变量

1.A 定义变量

```
a=$1
b=$2
```

注意：=二边没有空格,$1 表示从传递的参数中取第一个

1.B 变量使用($变量名)

```
if [ $a -eq $b ];then

echo "两个数相等"

fi
```

2 if(条件判断语句)

2.A 语法 1(if)

```
if [ $a -eq $b ];then

echo "两个数相等"

fi
```

2.B 语法 2(if else if)

```
if [ $a == $b ] then
echo "a 等 于 b"
elif [ $a -gt $b ]
then
echo "a 大 于 b"
elif [ $a -lt $b ]
then
echo "a 小 于 b"
else
echo " 没 有 符 合 的 条 件 "
fi
```

2.C 语法 3 ([]可用 test 替代)

```
if test $a =$b ;then

echo "两个数相等"

fi
```

2.D 完整实例

```
#!/bin/bash
a=$1
b=$2
if [ $a == $b ] then
echo "a 等 于 b"
elif [ $a -gt $b ]
then
echo "a 大 于 b"
elif [ $a -lt $b ]
then
echo "a 小 于 b"
else
echo " 没 有 符 合 的 条 件 "
fi
```

2.E 注意

中括号与变量需要有空格

如果 then 换行写，可以省略 “ ;”

3 test(判断)

Shell 中的 test 命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试。

3.A 数值测试

参数	说明
-eq	等于则为真
-ne	不等于则为真
-gt	大于则为真
-ge	大于等于则为真
-lt	小于则为真
-le	小于等于则为真

3.A.a 案例

```
#!/bin/bash
num1=100
num2=100
if test $[num1] -eq $[num2]
then
        echo '两个数相等！'
else
        echo '两个数不相等！'
fi
```

3.B 字符串

参数	说明
=	等于则为真
!=	不相等则为真
-z 字符串	字符串的长度为零则为真
-n 字符串	字符串的长度不为零则为真

3.B.a 实例

```
#!/bin/bash
num1="piaolaoshi"
num2="PIAOLAOSHI"
if test $num1 = $num2
then
    echo '两个字符串相等!'
else
    echo '两个字符串不相等!'
```

```
fi
```

3.C 文件

参数	说明
-e 文件名	如果文件存在则为真
-r 文件名	如果文件存在且可读则为真
-w 文件名	如果文件存在且可写则为真
-x 文件名	如果文件存在且可执行则为真
-s 文件名	如果文件存在且至少有一个字符则为真
-d 文件名	如果文件存在且为目录则为真
-f 文件名	如果文件存在且为普通文件则为真
-c 文件名	如果文件存在且为字符型特殊文件则为真
-b 文件名	如果文件存在且为块特殊文件则为真

3.C.a 案例

```
#!/bin/bash
cd /bin
if test -e ./bash
then
        echo '文件已存在!'
else
        echo '文件不存在!'
fi
```

3.C.b 注意

root 用户进行判断，任何文件都可读
普通用户判断可读时，必须用户、用户组、其它都可读才返回可读

3.C.c 其它逻辑操作

另外，Shell 还提供了与(-a)、或(-o)、非(!)三个逻辑操作符用于将测试条件连接起来，其优先级为："!"最高，"-a"次之，"-o"最低。例如：

```
#!/bin/bash
cd /bin
if test -e ./notFile -o -e ./bash
then
    echo '至少有一个文件存在!'
else
    echo '两个文件都不存在'
fi
```

4 运算符

4.A 算数运算符

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
+	加法	`expr $a + $b` 结果为 30。
-	减法	`expr $a - $b` 结果为 -10。
*	乘法	`expr $a \* $b` 结果为 200。
/	除法	`expr $b / $a` 结果为 2。
%	取余	`expr $b % $a` 结果为 0。
=	赋值	a=$b 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[$a == $b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[$a != $b] 返回 true。

4.A.a 案例

```
#!/bin/bash
a=10
b=20
val=`expr $a + $b`
echo "a + b : $val"
val=`expr $a \* $b`
echo "a * b : $val"
if [ $a == $b ]
then
    echo "a 等于 b"
fi
if [ $a != $b ]
then
    echo "a 不等于 b"
fi
```

注意：条件表达式要放在方括号之间，并且要有空格，例如: **[$a==$b]** 是错误的，必须写成 **[$a == $b]**。

乘号(*)前边必须加反斜杠(\)才能实现乘法运算；**`expr $a + $b`等于$(($a+$b))**

4.B 关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 true。	[$a -eq $b] 返回 false。
-ne	检测两个数是否不相等，不相等返回 true。	[$a -ne $b] 返回 true。
-gt	检测左边的数是否大于右边的，如果是，则返回 true。	[$a -gt $b] 返回 false。
-lt	检测左边的数是否小于右边的，如果是，则返回 true。	[$a -lt $b] 返回 true。
-ge	检测左边的数是否大于等于右边的，如果是，则返回 true。	[$a -ge $b] 返回 false。
-le	检测左边的数是否小于等于右边的，如果是，则返回 true。	[$a -le $b] 返回 true。

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

```
a=10
b=20

if [ $a -eq $b ]
then
    echo " a 等于  b"
else
    echo " a 不等于  b"
fi
if [ $a -gt $b ]
then
    echo "a 大于  b"
else
    echo "a 不大于  b"
fi
if [ $a -lt $b ]
then
    echo "a 小于  b"
else
    echo "a 不小于  b"
fi
```

4.C 逻辑运算符

以下介绍 Shell 的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

运算符	说明	举例
&&	逻辑的 AND	[[$a -lt 100 && $b -gt 100]] 返回 false
||	逻辑的 OR	[[$a -lt 100 || $b -gt 100]] 返回 true

```
#!/bin/bash
a=10
b=20

if [[ $a -lt 100 && $b -gt 100 ]]
then
    echo "返回  true"
else
    echo "返回  false"
fi

if [[ $a -lt 100 || $b -gt 100 ]]
then
    echo "返回  true"
else
    echo "返回  false"
fi
```

4.D 字符串运算符

下表列出了常用的字符串运算符，假定变量 a 为 "abc"，变量 b 为 "efg":

运算符	说明	举例

=	检测两个字符串是否相等，相等返回 true。	[$a = $b] 返回 false。
!=	检测两个字符串是否相等，不相等返回 true。	[$a != $b] 返回 true。
-z	检测字符串长度是否为 0，为 0 返回 true。	[-z $a] 返回 false。
-n	检测字符串长度是否为 0，不为 0 返回 true。	[-n "$a"] 返回 true。
$	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

```
#!/bin/bash
a="abc"
b="efg"

if [ $a = $b ]
then
    echo "a 等于 b"
else
    echo " a 不等于 b"
fi
if [ -z $a ]
then
    echo "字符串长度为 0"
else
    echo " 字符串长度不为 0"
fi
if [ $a ]
then
    echo "$a : 字符串不为空"
else
    echo "$a : 字符串为空"
fi
```

4.E 文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。

操作符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-c $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。
-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于 0），不为空返回 true。	[-s $file] 返回 true。
-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。

```
#!/bin/bash
file="/root/install.log"
if [ -r $file ]
then
    echo "文件可读"
```

```
else
    echo "文件不可读"
fi
if [ -w $file ]
then
    echo "文件可写"
else
    echo "文件不可写"
fi
if [ -f $file ]
then
    echo "文件为普通文件"
else
    echo "文件为特殊文件"
fi
if [ -d $file ]
then
    echo "文件是个目录"
else
    echo "文件不是个目录"
fi
if [ -e $file ]
then
    echo "文件存在"
else
    echo "文件不存在"
fi
```

5 循环结构

5.A for 循环

5.A.a 格式

```
for 变量名 in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

也可以用 in {1..9},表示循环 1-9

5.A.b 实例

```
#!/bin/bash
for num in 1 2 3 4 5
```

```
do
    echo "The value is: $num"
done

#只循环一次 输出：This is a string
for str in 'This is a string'
do
    echo $str
done
```

5.B while 循环

5.B.a 格式：

```
while condition
do
   command
done
```

5.B.b 实例 1

以下是一个基本的 while 循环，测试条件是：如果 int 小于等于 5，那么条件返回真。int 从 1 开始，每次循环处理时，int 加 1。运行上述脚本，返回数字 1 到 5，然后终止。

```
#!/bin/bash
int=1
while [ $int -le 5 ]
do
    echo $int
    let "int++"
done
```

5.B.c 实例 2

读取键盘信息 保存到 person
echo –n 不换行

```
#!/bin/bash
echo -n '输入你最喜欢的明星：'
while read person
do
    echo "是的！$person 是一个好人"
done
```

5.C until 循环

until 循环执行一系列命令直至条件为 true 时停止。

until 循环与 while 循环在处理方式上刚好相反。

以下实例我们使用 until 命令来输出 0 ~ 9 的数字:

```
#!/bin/bash
a=0
until [ ! $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

5.D 无限循环

```
#无限循环语法格式：
while :
do
    command
done
#或者
while true
do
    command
done
```

5.E 练习

5.E.a 1

1.创建目录/tmp/scripts

2.切换工作目录至此目录中

3.复制/etc/pam.d 目录至当前目录，并重命名为 test

4.将当前目录 test 所有者改为 bin，属组改为 adm

5.将 test 文件的权限改为用户可读可写可执行，用户组可读可写，其他人没有任何权限

```
#!/bin/bash
    mkdir -p /tmp/scripts
    cd /tmp/scripts
    cp -r /etc/pam.d ./test
```

```
chown  bin:adm ./test
chmod  760 ./test
```

5.E.b 99 乘法表

```
#!/bin/bash
for i in {1..9};do
 for y in {1..9};do
      if [  $y -le $i ];then
            echo -n "$y*$i=$(($y*$i))  "

     fi
 done
 echo ""
done
```

5.E.c 猜数

- 脚本生成一个 100 以内的随机数,提示用户猜数字,根据用户的输入,提示用户猜对了,
- # 猜小了或猜大了,直至用户猜对脚本结束。
- # RANDOM 为系统自带的系统变量,值为 0-32767 的随机
- # 使用取余算法将随机数变为 1-100 的随机数

```
#!/bin/bash
num=$[RANDOM%100+1]
echo "$num"
while :
do
        read -p "计算机生成了一个  1-100 的随机数,你猜: "  cai
   if [ $cai -eq $num ]
  then
        echo "恭喜,猜对了"
        exit
        elif [ $cai -gt $num ]
        then
                echo "不好意思,猜大了"
        else
                echo "不好意思,猜小了"
        fi
done
```

6 case(多选择语句)

Shell 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

下面的脚本提示输入 1 到 4，与每一种模式进行匹配：

```
echo '输入 1 到 4 之间的数字:'
echo '你输入的数字为:'
read num
case $num in
    1) echo '你选择了 1'
    ;;
    2) echo '你选择了 2'
    ;;
    3) echo '你选择了 3'
    ;;
    4) echo '你选择了 4'
;;
  5|6|7) echo ‘你选择了 5 或 6 或 7’
  ;;
    *) echo '你没有输入 1 到 4 之间的数字'
    ;;
esac
```

每个语句以;;分隔

以 case 开始 esac 结束

read num 读入值保存到 num

*匹配任何内容

7 break(跳出循环)

break 命令允许跳出所有循环（终止执行后面的所有循环）。

下面的例子中，脚本进入死循环直至用户输入数字大于 5。要跳出这个循环，返回到 shell 提示符下，需要使用 break 命令。

```
#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字:"
    read num
    case $num in
        1|2|3|4|5) echo "你输入的数字为 $num!"
```

```
    ;;
    *) echo "你输入的数字不是 1 到 5 之间的! 游戏结束"
        break
    ;;
  esac
done
```

也可用 exit 之间退出程序，同时可以返回一个值

例:

```
exit 1
```

8 函数

linux shell 可以用户定义函数，然后在 shell 脚本中可以随便调用。

```
#!/bin/bash
sayHi(){
    echo "hello 1906"
}
sayHi
```

8.A 带 return 的函数

```
#!/bin/bash

funWithReturn(){
    echo "这个函数会对输入的两个数字进行相加运算..."
    echo "输入第一个数字: "
    read num1
    echo "输入第二个数字: "
    read num2
    echo "两个数字分别为 $num1 和 $num2 "
    return $(($num1+$num2))
}
funWithReturn
echo "输入的两个数字之和为 $? "
```

8.B 函数参数

在 Shell 中，调用函数时可以向其传递参数。在函数体内部，通过 $n 的形式来获取参数的值，例如，$1 表示第一个参数，$2 表示第二个参数...

```
#!/bin/bash
funWithParam(){
    echo "第一个参数为 $1 "
    echo "第二个参数为 $2 "
    echo "第十个参数为 $10 "
    echo "第十个参数为 ${10} "
    echo "第十一个参数为 ${11} "
    echo "参数总数有 $# 个"
    echo "作为一个字符串输出所有参数 $* "
}
funWithParam 1 2 3 4 5 6 7 8 9 34 73
```

注意：$10 不能获取第十个参数，获取第十个参数需要${10}。当 n>=10 时，需要使用${n}来获取参数。