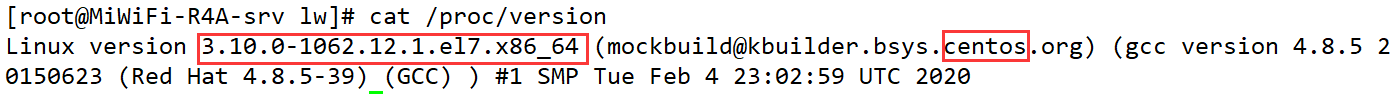
Shell编程

**Linux 3.10.0-1062.12.1.el7.x86\_64 centos**



|  |  |  |
| --- | --- | --- |
| user info | dubulingbo | dubulingbo@163.com |
| create date | 2020/03/17 | |
| modify date | 2020/03/22 | |
| version | testing | |

目录

[Shell基础 3](#_Toc35370148)

[1. Shell概述 3](#_Toc35370149)

[2. 脚本执行方法 3](#_Toc35370150)

[3. bash的基本功能 5](#_Toc35370151)

[3.1 别名与历史命令 5](#_Toc35370152)

[3.2 输出/输入重定向 6](#_Toc35370153)

[3.3 管道符 8](#_Toc35370154)

[3.4 通配符 9](#_Toc35370155)

[3.5 Bash中的其他特殊符号 10](#_Toc35370156)

[Bash变量 11](#_Toc35370157)

[1. Bash变量与变量分类 11](#_Toc35370158)

[1.1 什么是变量 11](#_Toc35370159)

[1.2 变量命名规则 11](#_Toc35370160)

[1.3 变量分类 11](#_Toc35370161)

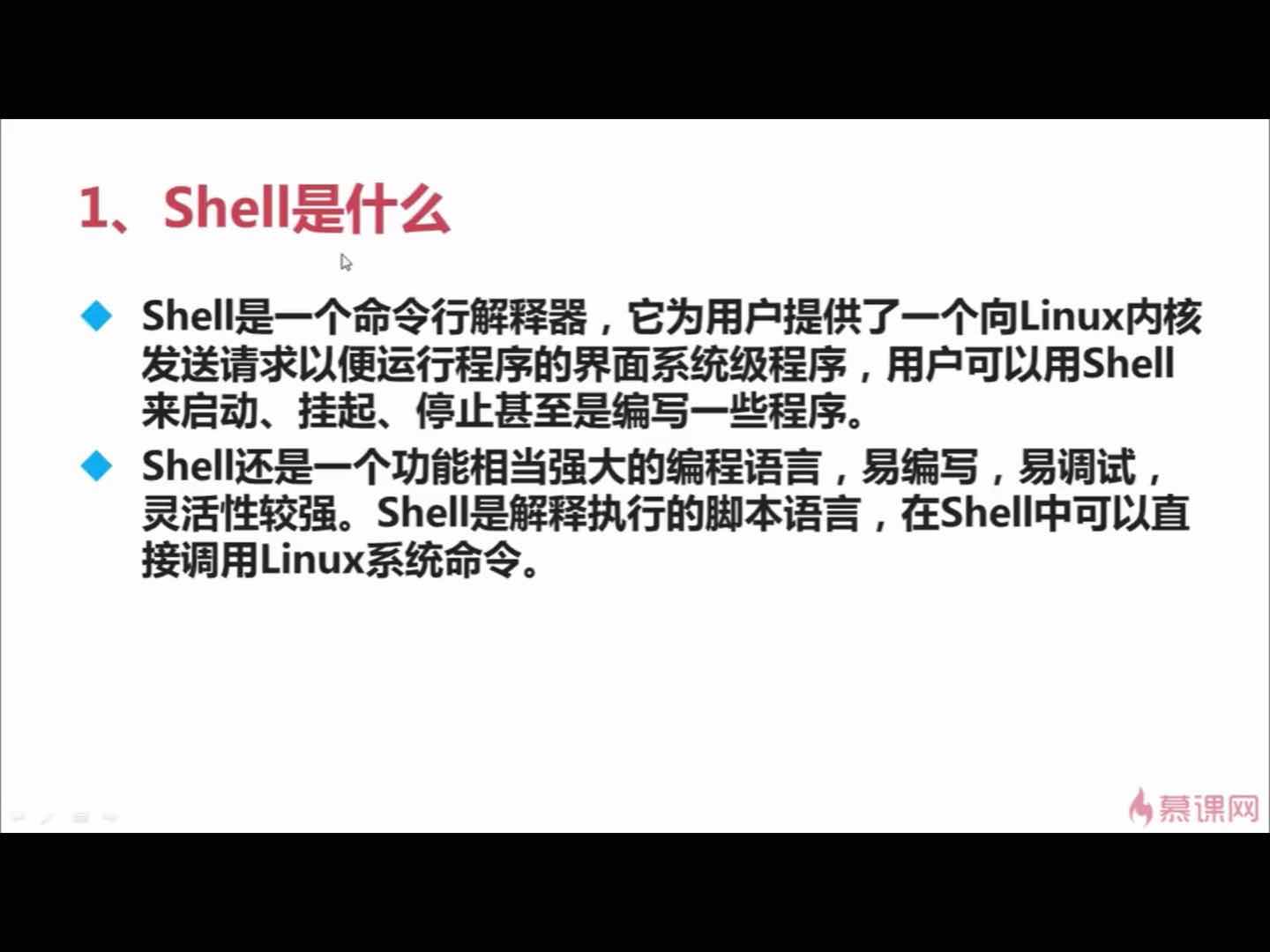
# Shell基础

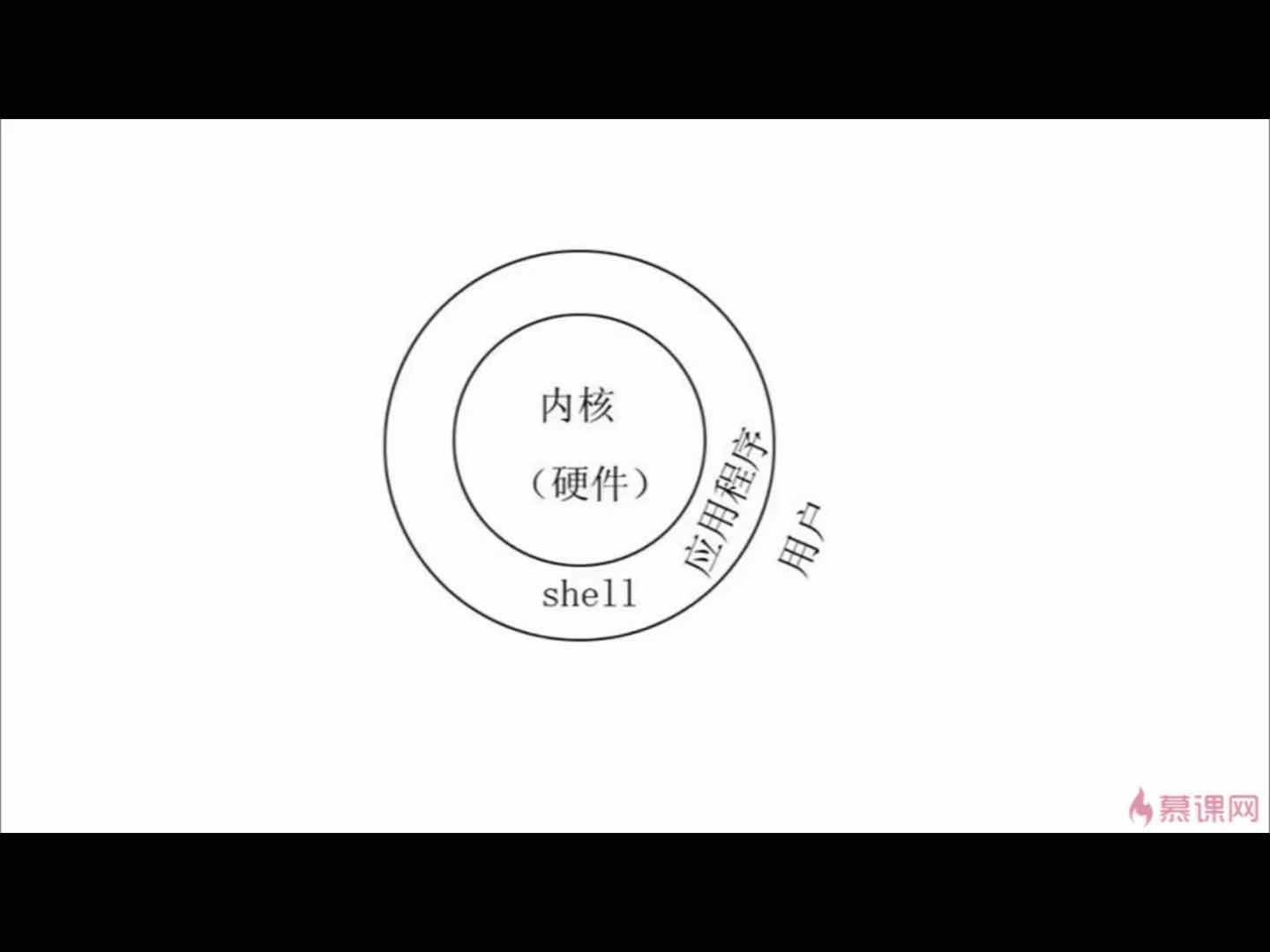
不管是PHP还是JAVA，它是用来编写应用程序或网站。而shell，它是一个脚本语言（所见即所得）。它不需要执行编译过程之后再执行；它是编译过程放在执行过程中，所以，执行起来要慢得多。

Shell主要是帮助管理员，简化管理操作。比如批量添加用户，系统定时备份等。

## 1. Shell概述

用户->Shell/应用程序->Linux内核





## 2. 脚本执行方法

**echo [选项] [输出内容]**

选项：

-e : 支持反斜杠控制的字符转换



**控制字符：**

\a : 输出警告音

\b : 推格键，也就是向左删除键

\n : 换行符

\r : 回车符

\t : 制表符，也就是Tab键

\v : 垂直制表符

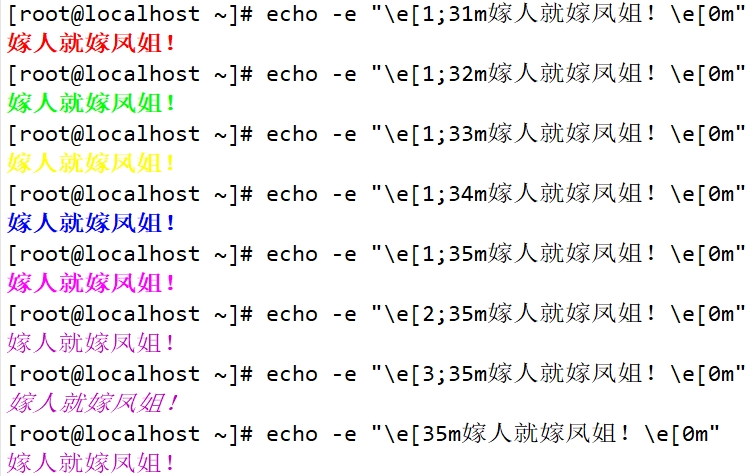
\0nnn : 按照八进制ASCII码表输出字符。其中0为数字零，nnn为三位八进制数

\xhh : 按照十六进制ASCII码表输出字符。其中hh是两位十六进制数



输出颜色字体

**echo -e “\e[1;31m嫁人就嫁凤姐！\e[0m”**



颜色选项：

30m=黑色，31m=红色，32m=绿色，33m=黄色

34m=蓝色，35m=洋红，36m=青色，37m=白色

编写脚本案例：

1. 编写脚本文件，保存为 \*.sh

2. 脚本执行

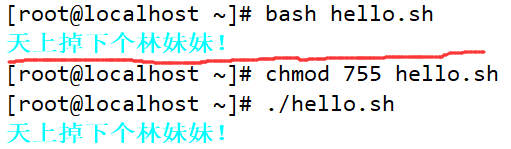
法一：赋予权限，直接运行

chmod 755 \*.sh

./\*.sh

法二：通过bash调用执行脚本

bash \*.sh





## 3. bash的基本功能

### 3.1 别名与历史命令

**alias**

查看系统中的别名

**alias ls=’ls –color=auto’**

设置别名

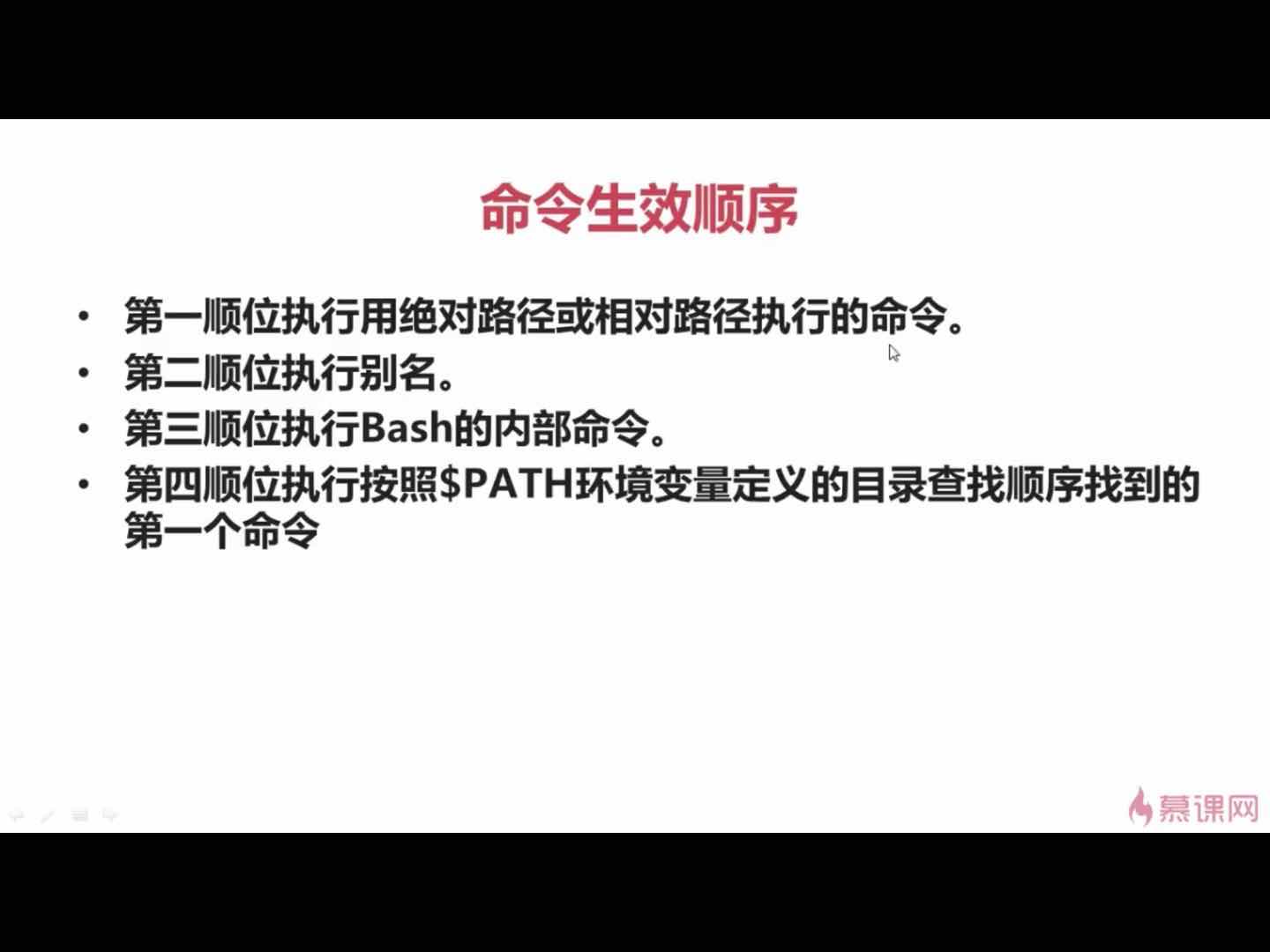
**vi ~/.bashrc**

写入环境变量配置文件可使别名永久生效

**unalias 别名**

删除别名

命令生效顺序：



**history [选项] [历史命令保存文件]**

查看历史命令

选项：

-c : 清空历史命令

-w : 把缓存中的历史命令写入历史命令保存文件~/.bash\_history

历史命令默认会保存最近的1000条，可以在环境变量配置文件/etc/profile中进行修改

历史命令的调用：

- 使用**上下箭头键**

- 使用“**!n**”重复执行第n条历史命令

- 使用“**!!**”重复执行上一条命令

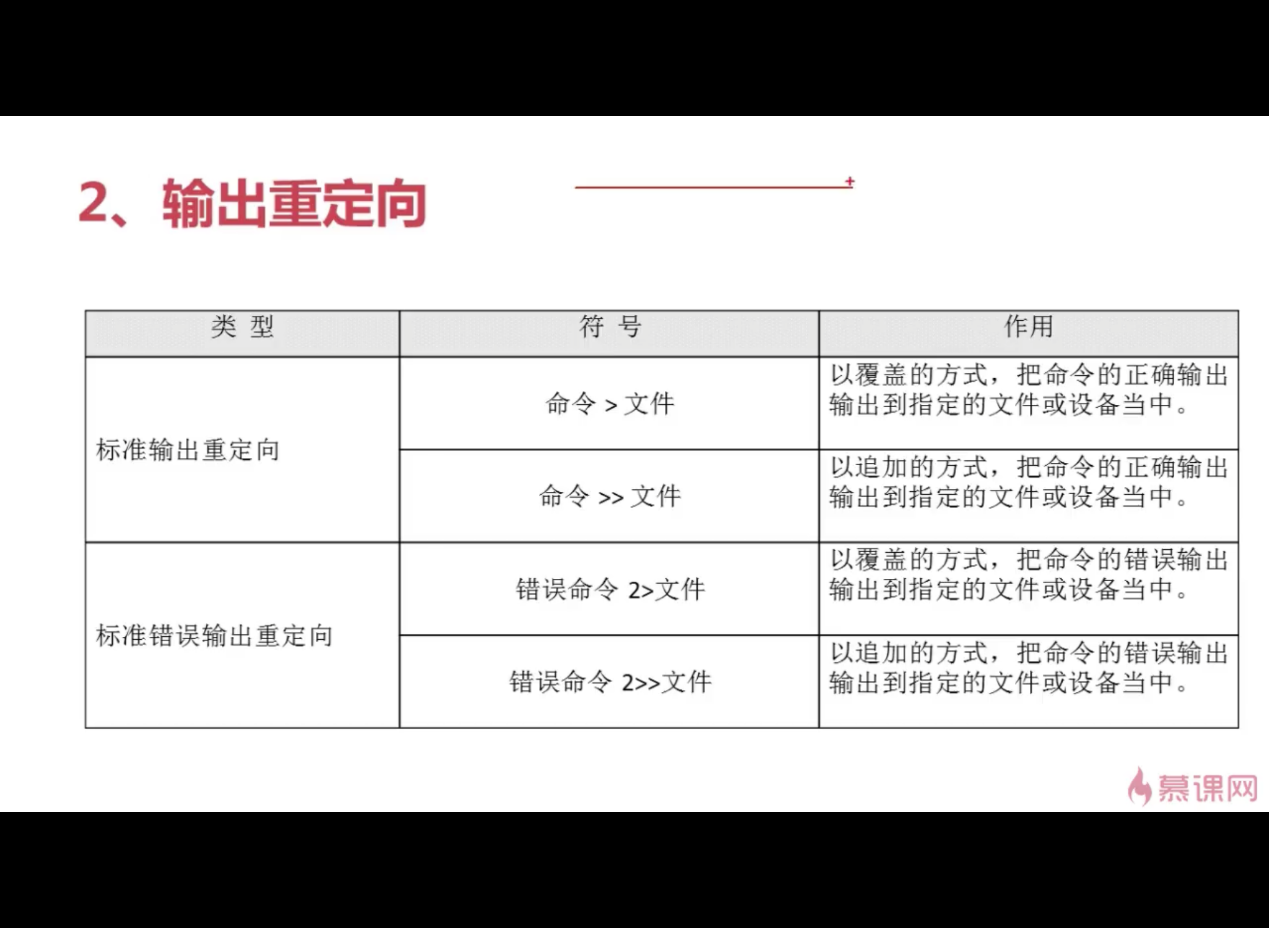
- 使用“**!字串**”重复执行最后一条以该字串开头的命令

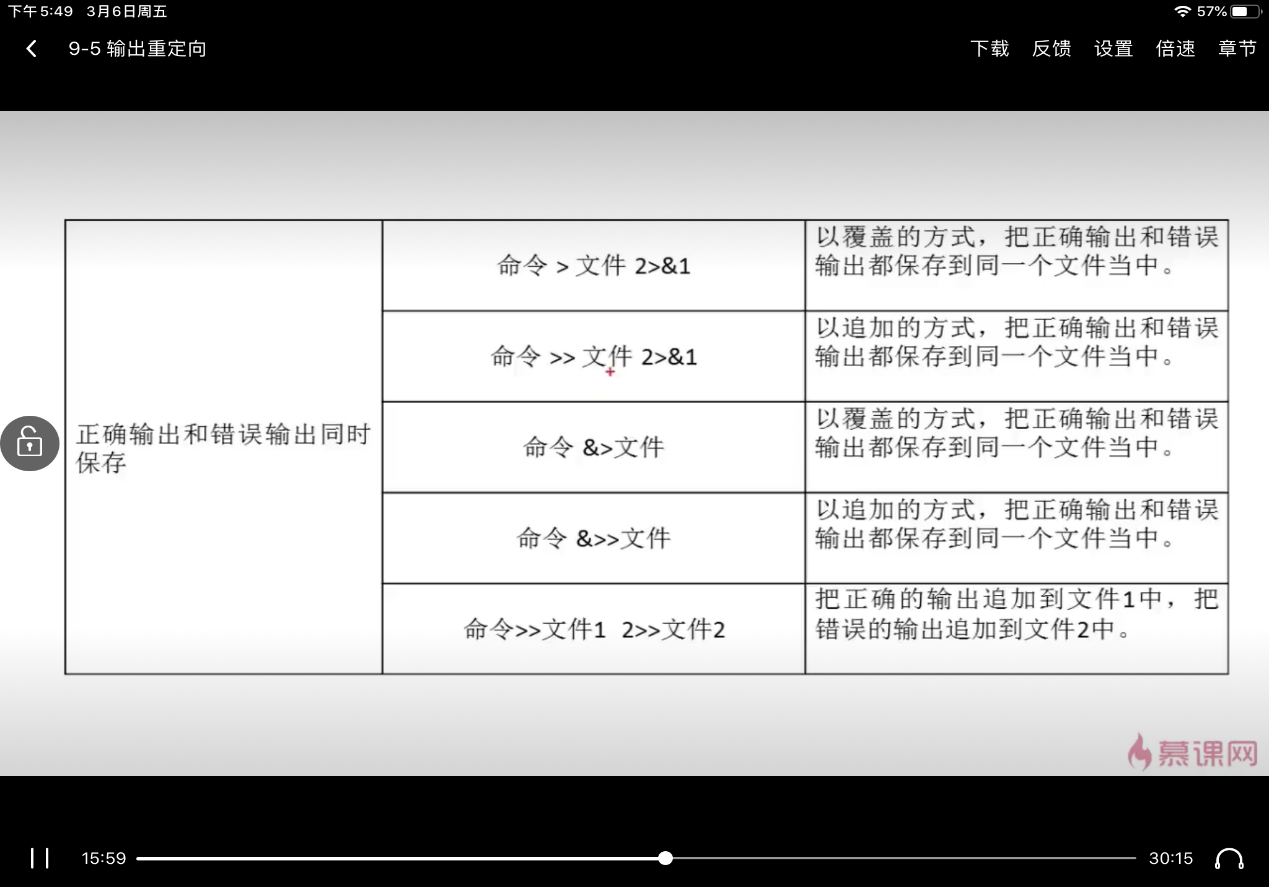
### 3.2 输出/输入重定向

1. 标准的输入/输出

|  |  |  |  |
| --- | --- | --- | --- |
| **设备** | **设备文件名** | **文件描述符** | **类型** |
| **键盘** | /dev/stdin | 0 | 标准输入 |
| **显示器** | /dev/stdout | 1 | 标准输出 |
| **显示器** | /dev/stderr | 2 | 标准错误输出 |

2. 输出重定向





3. 输出重定向

**wc [选项] [文件名]**

选项：

-c : 统计字节数

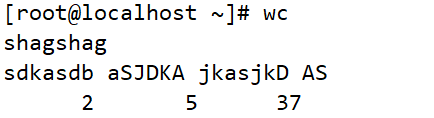
-w : 统计单词数

-l : 统计行数

**wc**

键盘输入

Ctrl+D 出结果



**命令<文件**

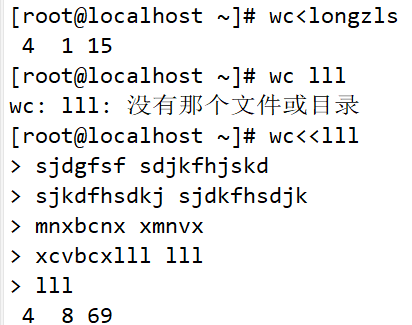
把文件作为命令的输入

**命令<<标识符**

**…**

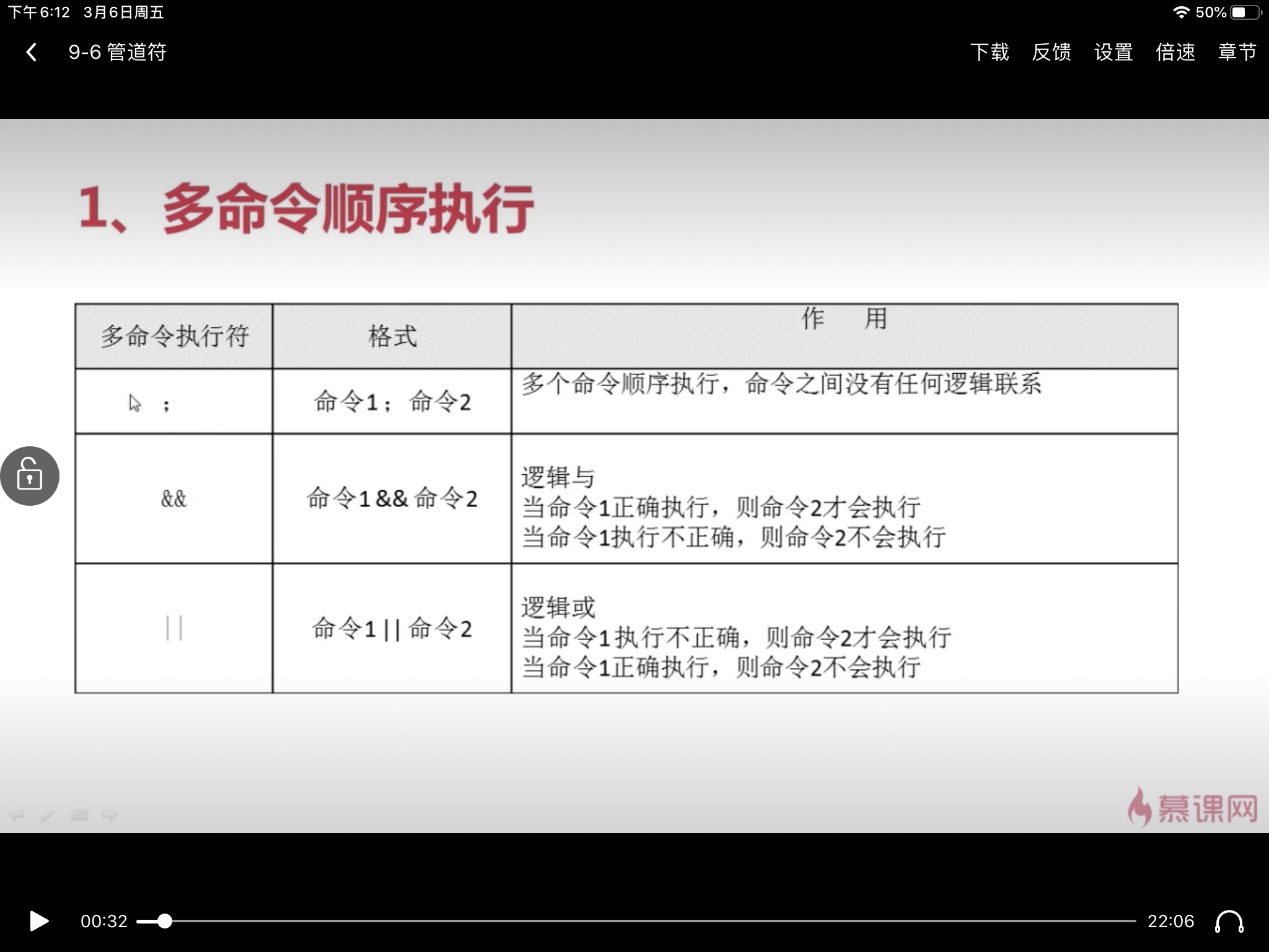
**标识符**

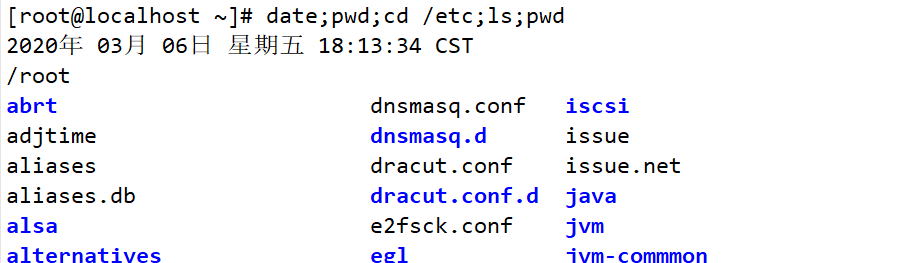
把标识符之间的内容作为命令的输入



### 3.3 管道符

#### 3.3.1多命令顺序执行







#### 3.3.2管道符 |

**命令1 | 命令2**

将命令1的正确输出作为命令2的操作对象

**ls -l /etc | more**

**netstat -an**

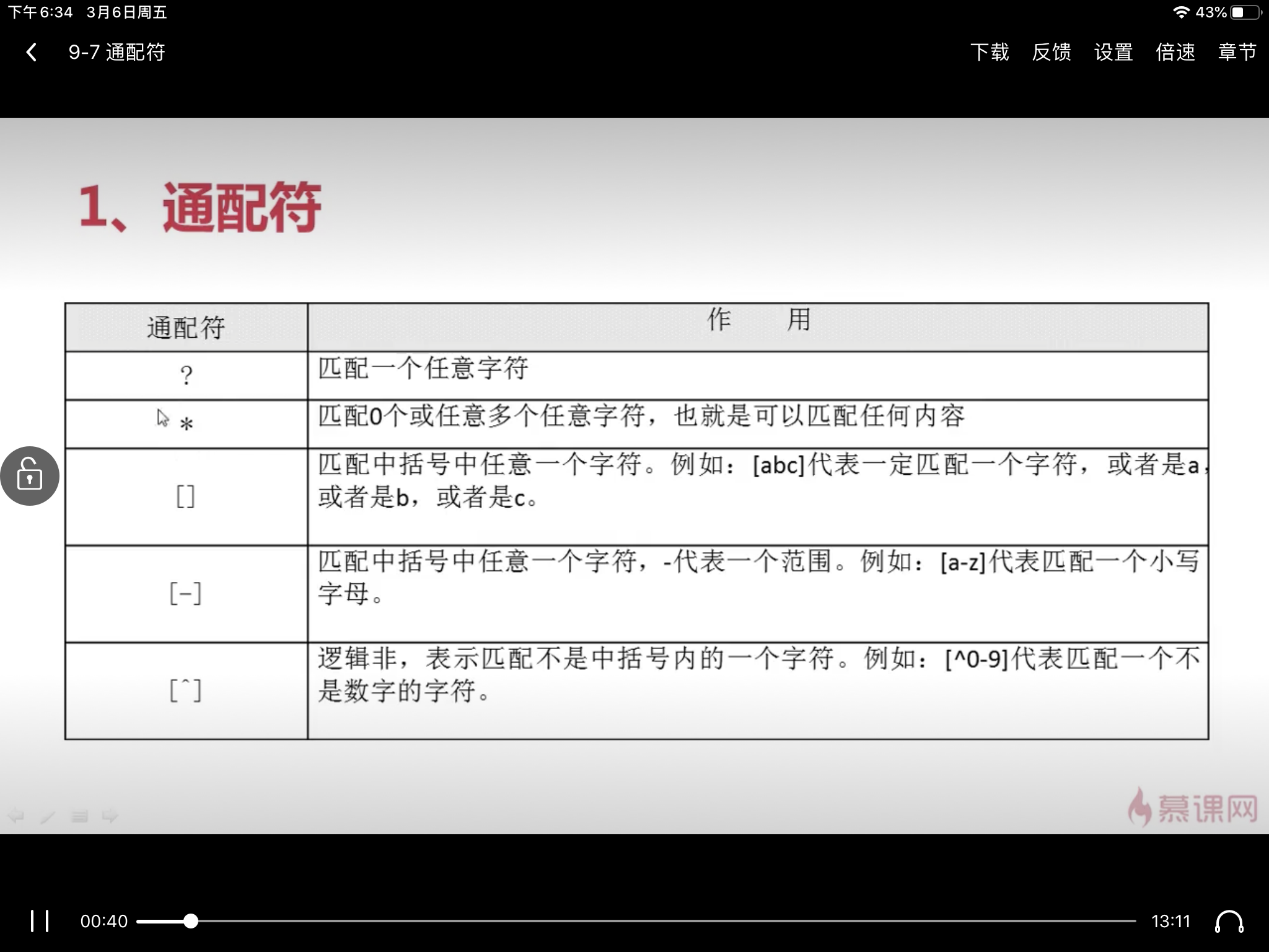
查看系统中的所有网络连接

**netstat -an | grep ESTABLISHED**

查看正在连接的网络



### 3.4 通配符



### 3.5 Bash中的其他特殊符号



# Bash变量

## 1. Bash变量与变量分类

### 1.1 什么是变量

变量是计算机内存的单元，其中存放的值可以改变。

变量让你能够把程序中准备使用的每一段数据都赋给一个简短、易于记忆的名字，因此它们十分有用。

### 1.2 变量命名规则

- 变量名只能由数字、字母和下划线组成，且首字符只能是字母或下划线

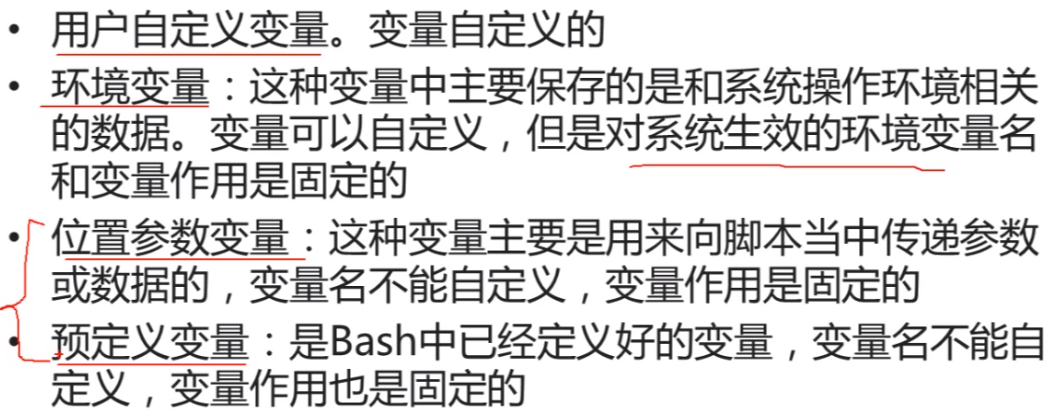
- 变量名的长度不能超过255个字符

- 变量名在有效范围内必须唯一

- 变量名最好能代表一定的含义 \*

- 在Bash中，变量默认类型都是字符串型

### 1.3 变量分类



#### 1.3.1 用户自定义变量

变量名称，内容和作用都可以自己定义。

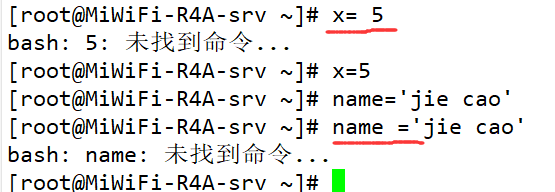
##### 定义变量

变量名=变量值

例如：

x=5，name=”jie cao”……

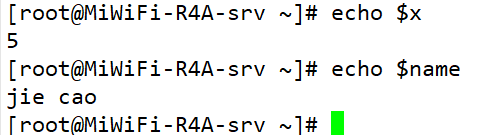
注意：等号左右不能加任何空格，否则会报错



##### 变量调用

echo $变量名 输出变量的值

只要用到变量，就要在变量名前面加上$

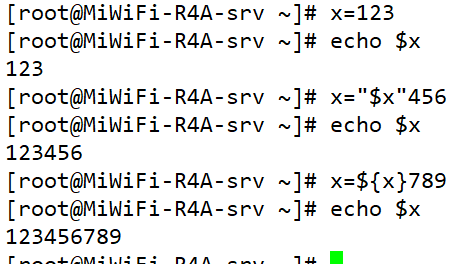


##### 变量值叠加

x=123

**法一：**x=”$x”456

**法二：**x=${x}789



##### 变量查看

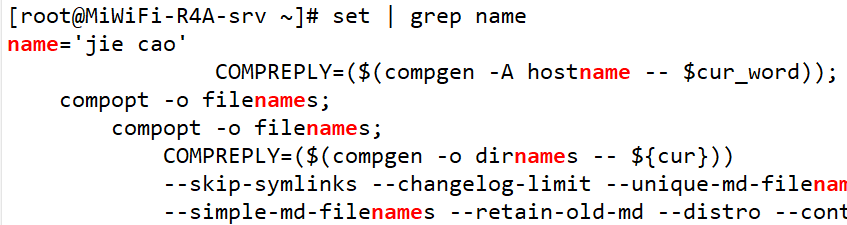
**set**

显示系统中已经存在的shell变量，以及设置shell变量的新变量值

参考链接：https://man.linuxde.net/set

选项：

-u：如果设定此选项，调用未声明变量时会报错（默认无任何提示）



**env** 查看所有环境变量

##### 变量删除

**unset 变量名**（变量名前面不能加$）

#### 1.3.2 环境变量

定义每个用户的操作环境。它是全局变量，而自定义变量是局部变量，所谓局部变量就是只能在当前的shell环境中有效，而全局变量没有这个限制。

对系统生效的环境变量名称和变量作用是固定的。

##### 用户自定义的环境变量

*export 变量名=变量值*

或

*变量名=变量值*

*export 变量名*

环境变量的删除，查看等操作和用户自定义的变量操作一样

##### 常用的系统环境变量

HOSTNAME：主机名

SHELL：当前的shell

TERM：终端环境

HISTSIZE：历史命令的条数

SSH\_CLIENT：当前操作环境是用ssh连接的，这里记录客户端IP

SSH\_TTY：ssh连接的终端编号（pts/1…）

USER：当前登录的用户

**PATH系统环境变量：**系统查找命令的路径

**echo $PATH** 查看PATH环境变量

**PATH=”$PATH”:/root/sh** 增加PATH变量的值

**PS1系统环境变量：**提示符

- \d ：可显示出『星期 月 日』的日期格式，如："Mon Feb 2"

- \H ：完整的主机名。

- \h ：仅取主机名在第一个小数点之前的名字

- \t ：显示时间，为 24 小时格式的『HH:MM:SS』

- \T ：显示时间，为 12 小时格式的『HH:MM:SS』

- \A ：显示时间，为 24 小时格式的『HH:MM』

- \@ ：显示时间，为 12 小时格式的『am/pm』样式

- \u ：目前使用者的账号名称，如『root』；

- \v ：BASH 的版本信息

- \w ：完整的工作目录

- \W ：利用 basename 函数取得工作目录名称，所以仅会列出最后一个目录名。

- \# ：下达的第几个命令。

- \$ ：提示字符，root -> #，普通用户 -> $

例如：PS1='[\u@\h \w \A #\#]\$ '

在终端中修改环境变量的值，只是临时修改，需要永久生效只能修改相应的配置文件

##### 环境变量配置文件（主要是针对系统登录时生效的文件）

**source 配置文件** 参考链接：https://man.linuxde.net/source

在当前bash环境下读取并执行**配置文件**中的命令

或

**. 配置文件（有空格）**

环境变量配置文件主要是定义对系统操作环境生效的系统默认环境变量，如PATH，HISTSIZE，PS1，HOSTNAME等。

###### 常用的环境变量配置文件

系统登录时加载的配置文件

**/etc/profile**

**/etc/profile.d/\*.sh**

**~/.bash\_profile**

**~/.bashrc**

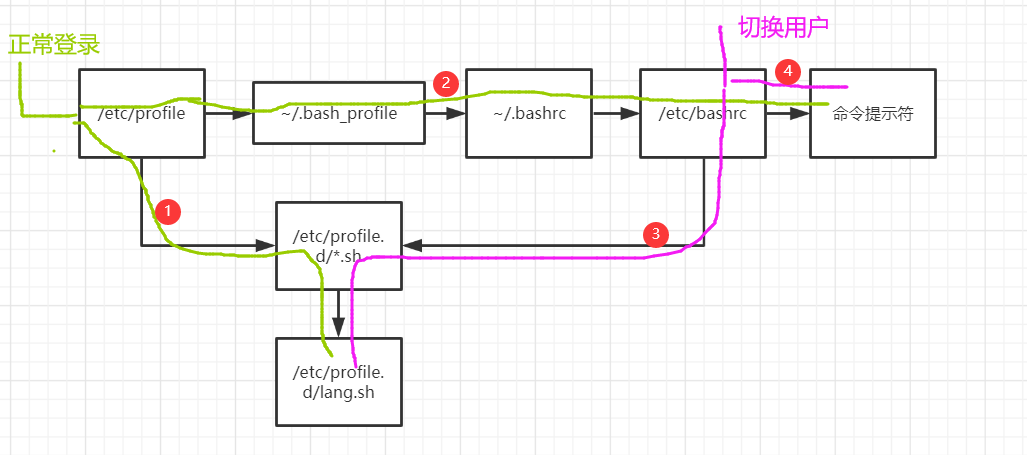
**/etc/bashrc**

参考链接： blog.csdn.net/weixin\_34337381/article/details/92419551

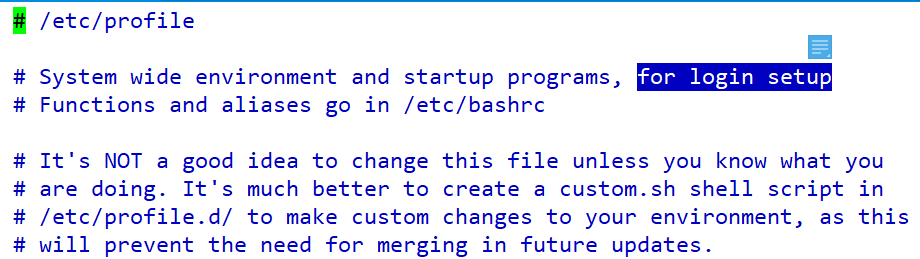
###### 文件加载顺序

正常登录：路线一

通过su命令切换用户：路线二



**vim /etc/profile**



/etc/profile的作用：

**USER变量**

**LOGNAME变量**

**MAIL变量**

**PATH变量**

**HOSTNAME变量**

**HISTSIZE变量**

**umask：设置用户的默认权限**

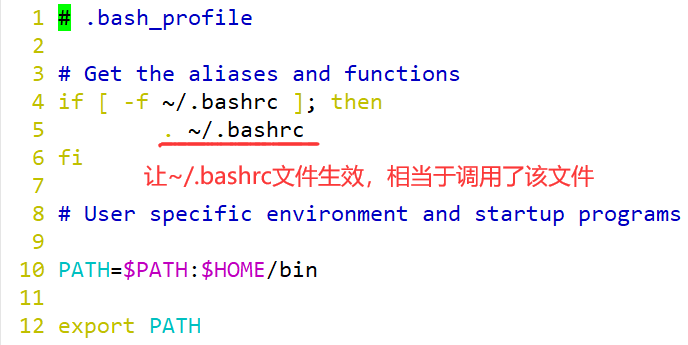
**调用/ect/profile.d/\*.sh文件**

解释umask，请参考[umask命令](#_umask命令)

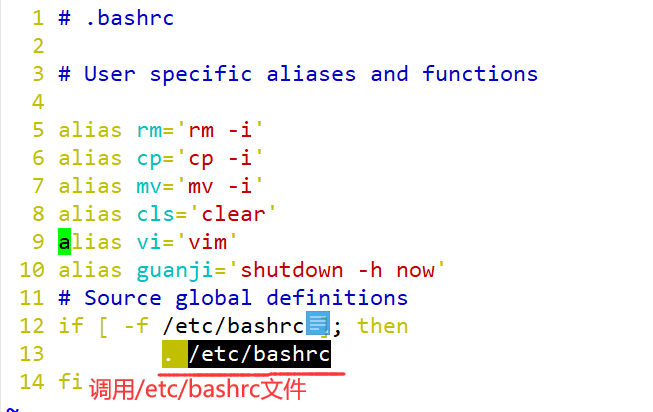
**vim /etc/profile.d/lang.sh** 设置系统的默认语系



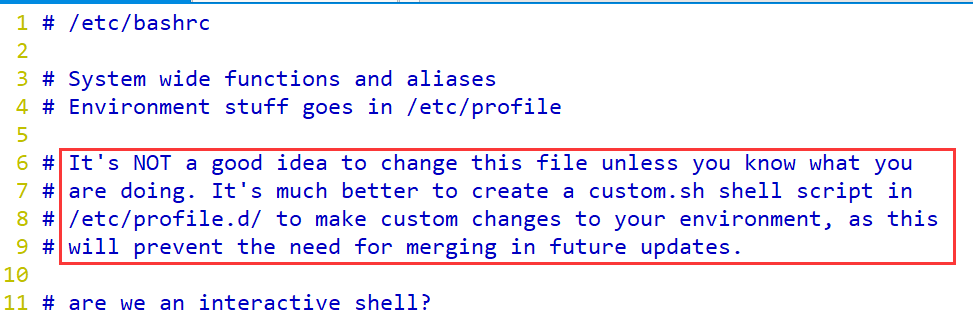
**vim ~/.bash\_profile** 调用~/.bashrc



**vim ~/.bashrc** 定义别名 + 调用/etc/bashrc



**vim /etc/bashrc**



/etc/bashrc文件的作用：

**- PS1变量**（最终生效）

**- umask**

**- PATH变量**

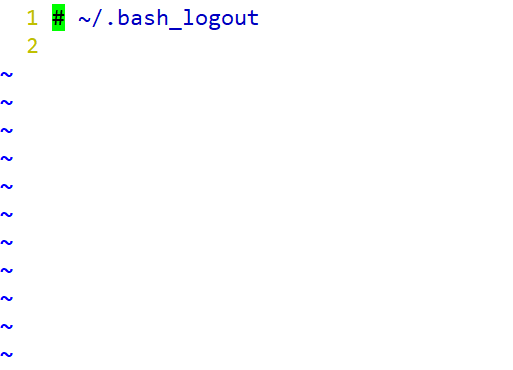
**- 调用/etc/profile.d/\*.sh文件**

注意：PS1环境变量是在加载配置文件的最后一个设置的

##### 其他配置文件

1. 注销时生效的环境变量配置文件

vim ~/.bash\_logout 默认为空，也就是在注销时不做任何配置操作



可以在里面设置自己注销时的操作，比如清除历史命令，清楚痕迹。

2.历史命令的配置文件

vim ~/.bash\_history

里面记录的是已经**存档（存在硬盘上）**的之前该用户敲过的历史命令



3. Shell登录信息

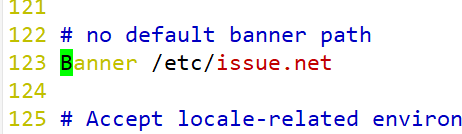
**本地终端欢迎信息**：/etc/issue

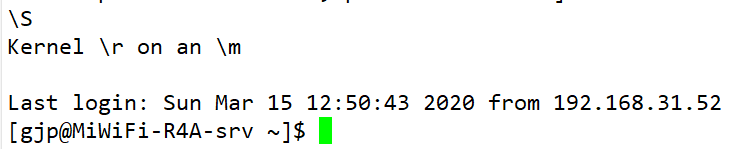
|  |  |
| --- | --- |
| 转义符 | 作 用 |
| \d | 显示当前系统的日期 |
| \s | 显示操作系统名称 |
| \l | 显示登录终端号，这个比较好用 |
| \m | 显示硬件体系结构，如i386、i686等 |
| \n | 显示主机名 |
| \o | 显示域名 |
| \r | 显示内核版本 |
| \t | 显示当前系统时间 |
| \u | 显示当前登录用户的序列号 |

**远程登录欢迎信息：/etc/issue.net**

- 转义符在/etc/issue.net文件中不能使用

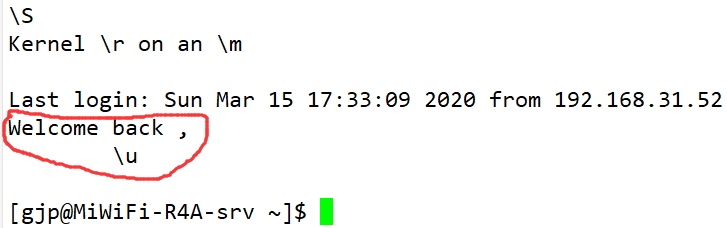
- 是否显示此欢迎信息，由SSH的配置文件/etc/ssh/sshd\_config决定，加入“Banner /etc/issue.net”行才能显示（记得要重启服务）





**登录后欢迎信息：/etc/motd**

不管是远程登录还是本地登录，都可以显示此欢迎信息



##### 语系变量

**locale** 显示当前系统语系



- LANG：定义系统主语系的变量

- LC\_ALL：定义整体语系的变量

**echo $LANG** 查看系统当前语系

**locale -a | more** 查看Linux支持的所有语系

**cat /etc/locale.conf**查询系统默认语系,该文件是语系的配置文件

参考链接：https://www.cnblogs.com/li5206610/p/7828618.html



##### Linux中文支持

前提条件：正确安装了中文字体和中文语系

- 如果有图形界面，可以正确支持中文显示

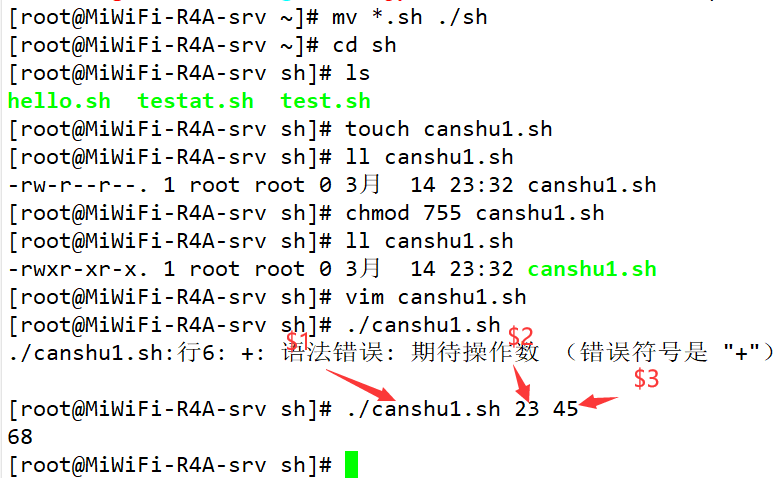
- 如果使用第三方远程工具，只要语系设定正确，也可以致辞中文显示

- 如果是纯字符界面，逆序安装第三方插件（如zhcon等）

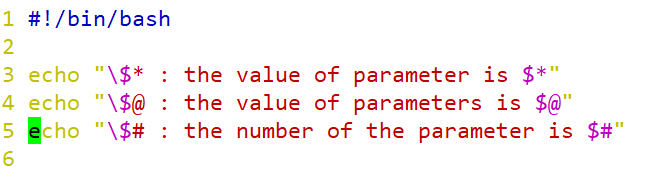
#### 1.3.3 位置参数变量

|  |  |
| --- | --- |
| **位置参数变量** | **作 用** |
| **$n** | n为数字，$0代表命令本身，$1-$9代表第一个到第九个参数，十以上的参数需要用大括号包含，如${10}表示第十个参数 |
| **$\*** | 表示命令中所有的参数，$\*把所有的参数看成一个整体 |
| **$@** | 表示命令中所有的参数，不过$@把每个参数区分对待 |
| **$#** | 代表命令中所有参数的个数 |

**演示$n：**



**演示$\*，$@，$#：**



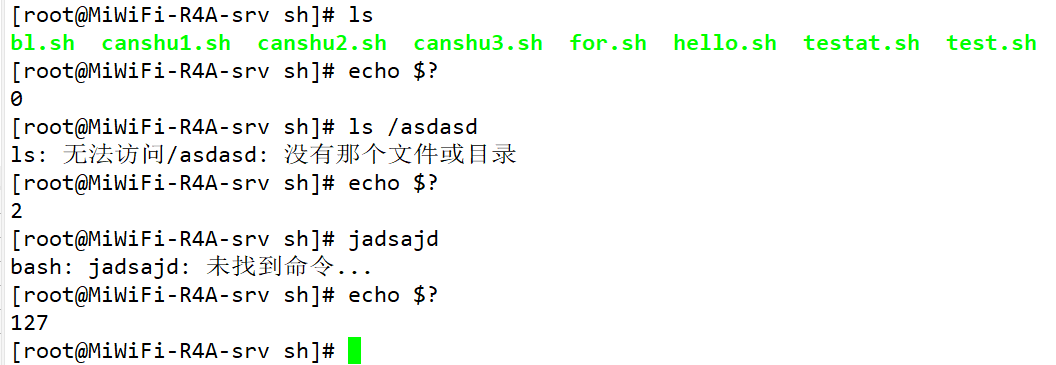


注意：取位置参数变量里的值要用**双引号！！！**

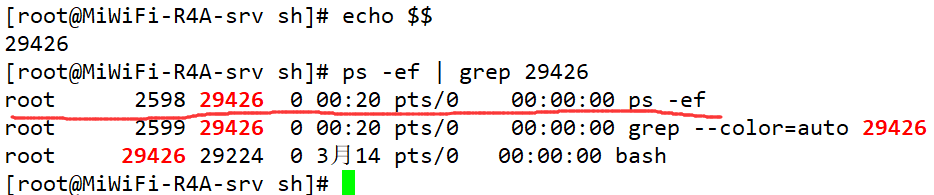
#### 1.3.4 预定义变量

|  |  |
| --- | --- |
| **预定义变量** | **作用** |
| **$?** | 最后一次执行的命令的返回状态。如果这个变量的值为0，证明上一个命令正确执行；非零就是不正确执行（具体是哪个数有命令本身决定） |
| **$$** | 当前进程的进程号（PID） |
| **$!** | 后台运行的最后一个进程号（PID） |

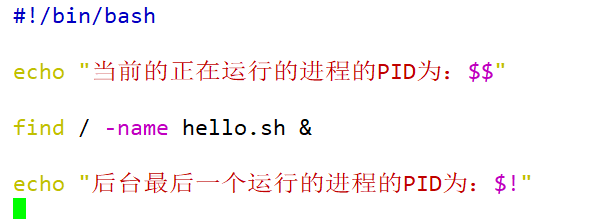
**演示$?：**



**演示$$：**



**演示$!：**





##### 接收键盘输入

向shell程序传递参数，前面讲过可以使用$n，但是这个传递方式很不灵活，一般只有程序的开发者才知道应该传递几个参数。而read就可以使程序的使用者也能轻易的知道需要传递的参数个数。

**read [选项] [变量名]**

从键盘读取变量的值，通常用在shell脚本中与用户进行交互的场合。该命令可以一次读取多个变量的值，变量和输入的值都需要使用空格隔开。在read命令后面，如果没有指定变量名，读取的数据将被自动赋值给特定的变量REPLY

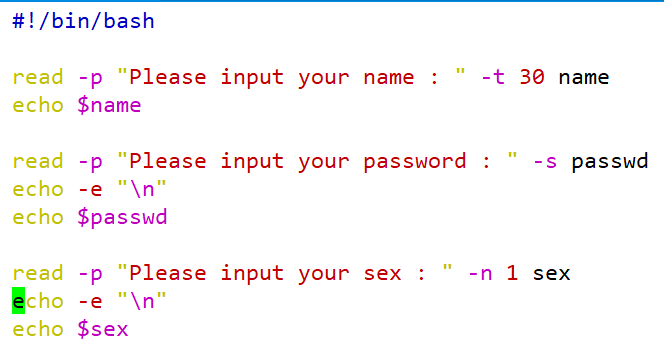
选项：

**-p “提示信息”**：在等待read输入时，输出提示信息

**-t 秒数**：read命令会一致带带用户输入，使用此选项可以指定等待时间

**-n 字符数**：read命令只接受指定的字符数，就会执行

**-s**：隐藏输入的数据，适用于机密信息的输入



# Shell运算符

Shell变量的特点：弱类型，默认是字符串类型（不能进行数值运算）。

## 1. declare命令

### 1.1 declare声明变量类型

**declare [+/-][选项] 变量名**

选项：

-：给变量设定类型属性

+：取消变量的类型属性

-a：将变量声明为数组型

-i：将变量声明为整数型

-x：将变量声明为环境变量

-r：将变量声明为只读变量

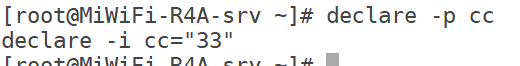
-p：显示指定变量的被声明的类型

默认为字符串型

### 1.2 声明整型变量：

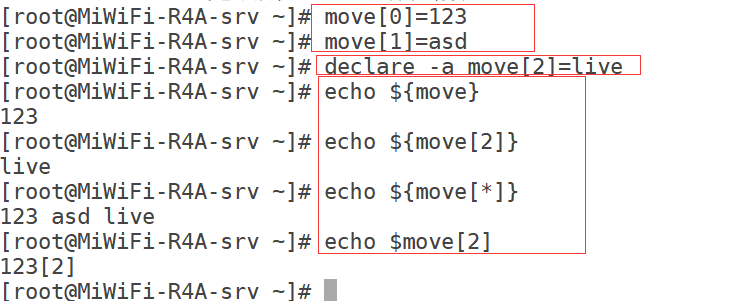
**declare -i 变量名=x**声明cc为整型变量，其值为$aa与$bb的和





### 1.3 声明数组变量：

declare -a 变量名=x

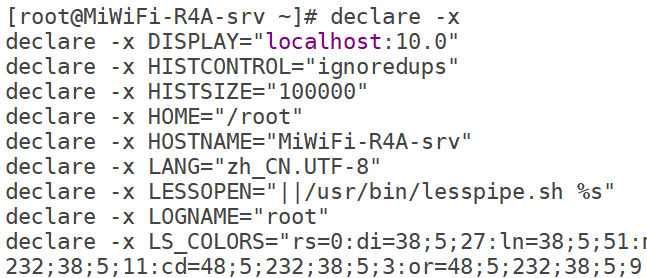


### 1.4 声明环境变量：

**declare -x 变量名=x**

和export命令作用相似，但其实执行的**都是declare命令**

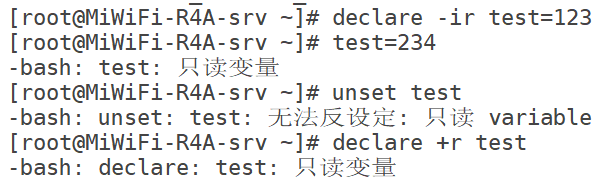
**declare -x** 显示所有已声明的环境变量



### 1.5 声明变量只读属性：

declare -r 变量名

给变量赋予只读属性，但是要注意只读属性会让变量不能修改不能删除，甚至不能取消只读属性



**查询变量属性**：

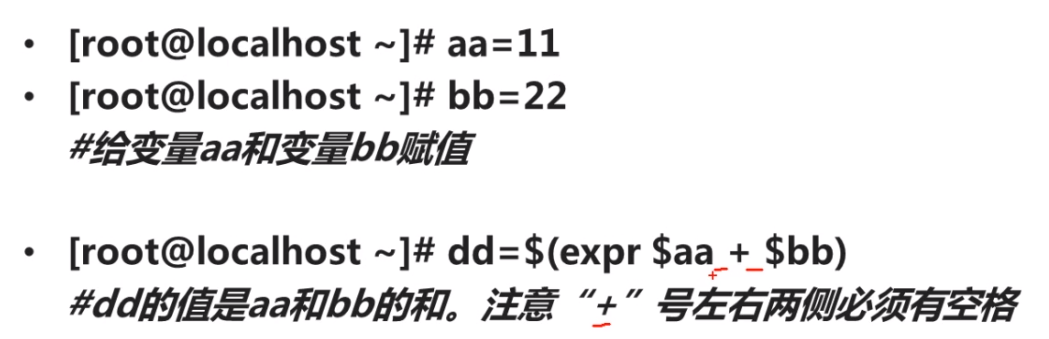
**declare -p** 查询所有变量的属性

**declare -p 变量名** 查询指定变量的属性

## 2. 数值运算的方法

### 2.1 使用declare命令在声明时指定其为整型变量

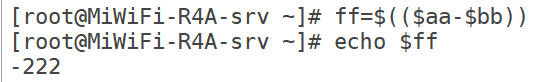
### 2.2 使用expr和let数值运算工具

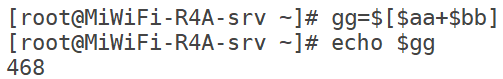


**变量名=$(expr 整型值 + 整型值)**

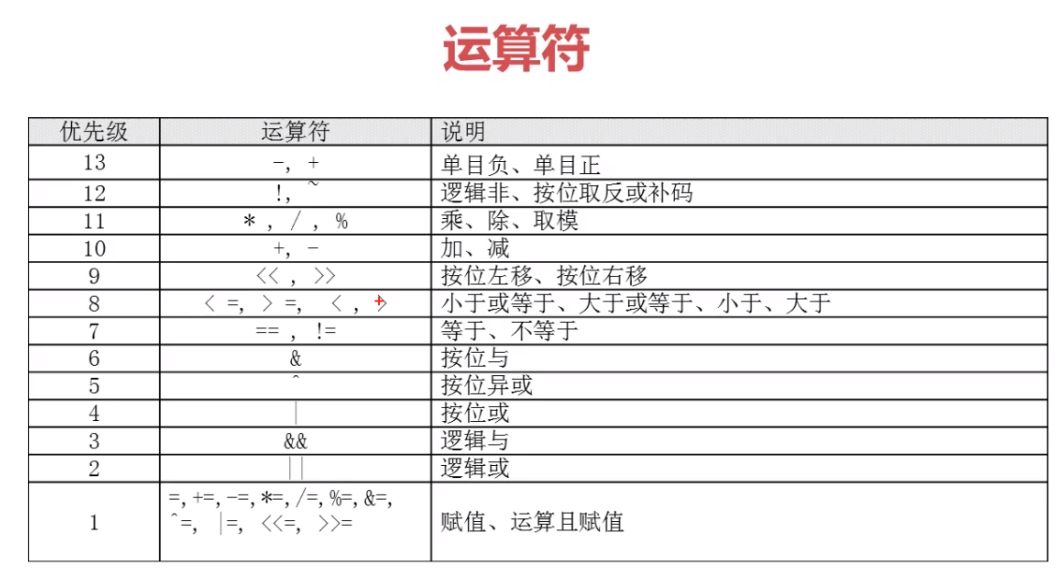
注意：“+”号的左右两侧必须有空格

### 2.3 “$((运算式))”或“$[运算式]”



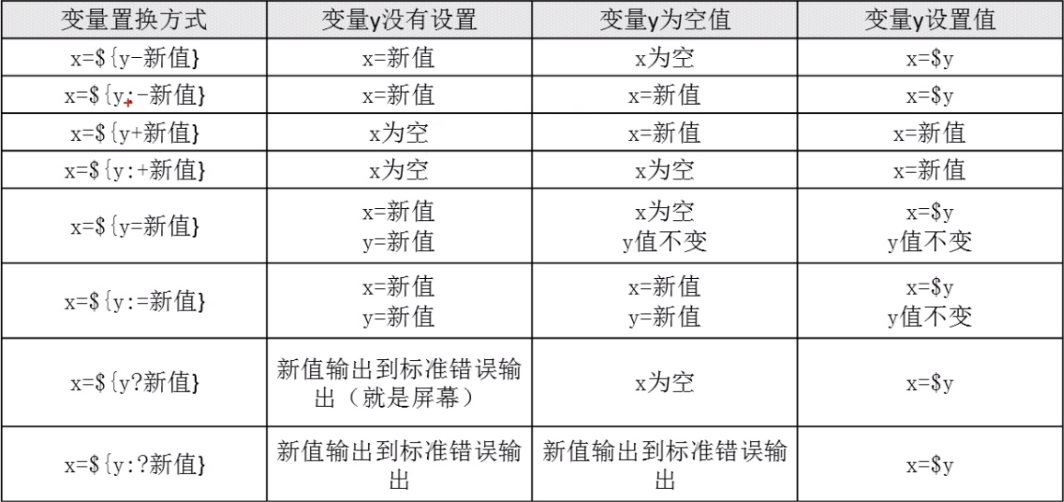


### 2.4 Shell中支持的运算符



## 3. 变量测试

只针对Shell有效；格式复杂多样，语法简单；一般用于脚本优化



# Shell正则表达式

## 1. 正则表达式

### 1.1 什么是正则表达式

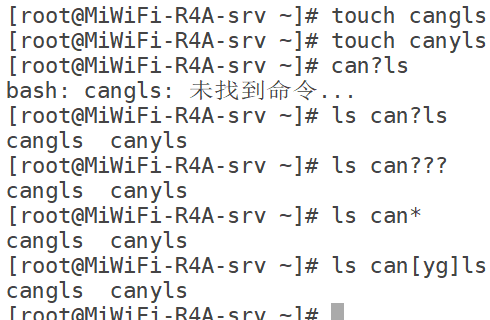
正则表达式是用于描述字符排列和匹配模式的一种语法规则。它主要用于字符串的模式分割、匹配、查找及替换操作。

### 1.2 正则表达式与通配符

Linux中既有表达式又有通配符，这是与其它语言不同的地方！其中**正则表达式**用来在文件中匹配符合条件的字符串，正则是包含匹配。grep、awk、sed等命令可以支持正则表达式。而**通配符**用来匹配符合条件的文件名，通配符是完全匹配。ls、find、cp这些命令不支持正则表达式，所以只能使用shell自己的通配符来进行匹配。

#### 1.2.1 通配符

请参考[Shell基础.3.4节](#_3.4_通配符)，这里不再赘述！

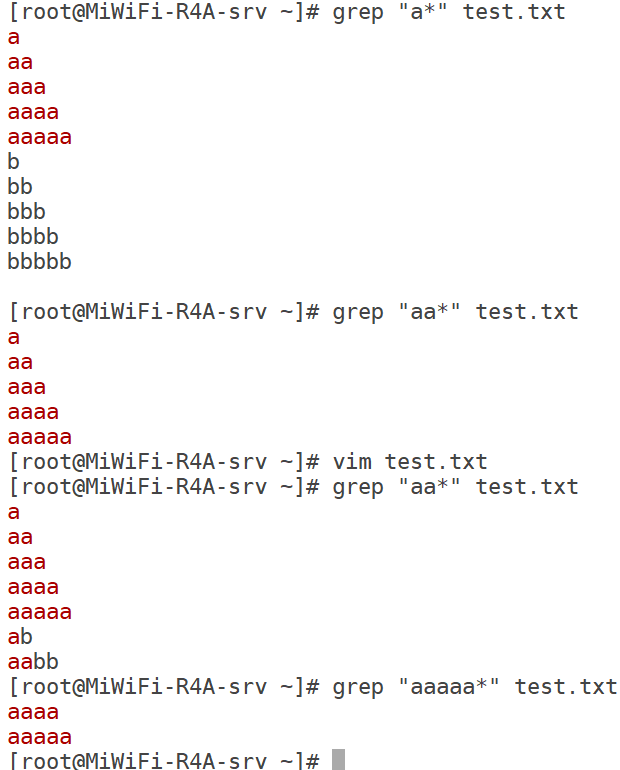


#### 1.2.2 基础正则表达式

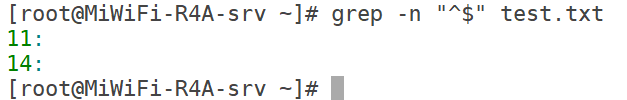
|  |  |
| --- | --- |
| **元字符** | **作 用** |
| **\*** | 前一个字符匹配0次或任意多次 |
| **.** | 匹配除了换行符外任意一个字符 |
| **^** | 匹配行首。例如：^hello会匹配以hello开头的任意行 |
| **$** | 匹配行尾。例如hello$会匹配以hello结尾的行 |
| **[]** | 匹配中括号中指定的任意一个字符，只匹配一个字符。  例如：[aeiou]匹配任意一个元音字母，[0-9]匹配任意一位数字，  [a-z][0-9]匹配小写字母和一位数字构成的两位字符 |
| **[^]** | 匹配除中括号的字符以外的任意一个字符。例如：[^0-9]匹配任意一位非数字字符，[^a-z]表示匹配任意一位非小写字母 |
| **\** | 转义符。用于取消特殊符号的含义 |
| **\{n\}** | 表示其前面的字符恰好出现n次。例如：[0-9]\{4\}匹配4位数字，  [1][3-8][0-9]\{9\}匹配手机号码 |
| **\{n,\}** | 表示其前面的字符出现不小于n次。例如：[0-9]\{2,\}表示两位及以上的数字 |
| **\{n,m\}** | 表示其前面的字符至少出现n次，最对出现m次。例如：[a-z]\{6,8\}匹配6到8位的小写字母。 |

扩展正则表达式这里不提

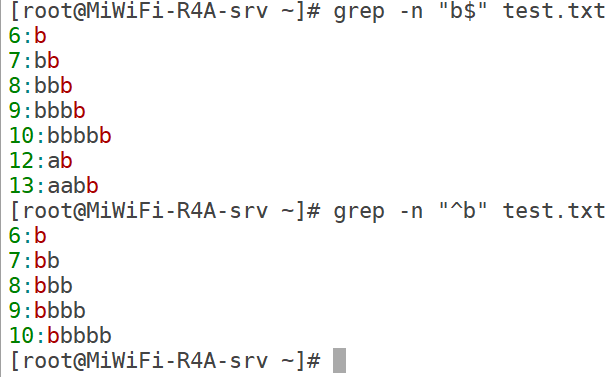
测试“\*”：



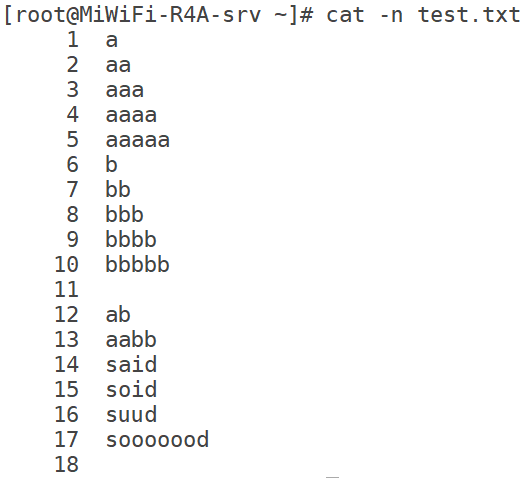
“^$”：匹配空白行



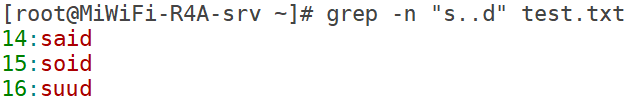
“^M”,“n$”：以M开头的行，以n结尾的行



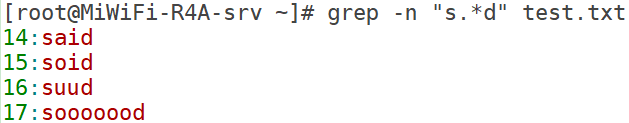
**文档内容：**



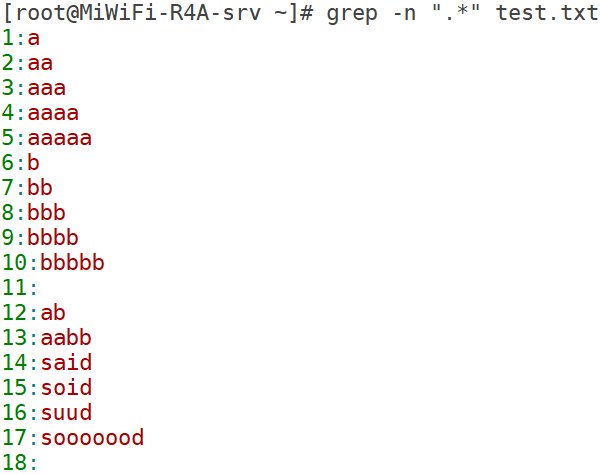
**“s..d”：**匹配在s和d之间一定有两个字符的行



**“s.\*d”：**匹配在s和d之间有任意字符

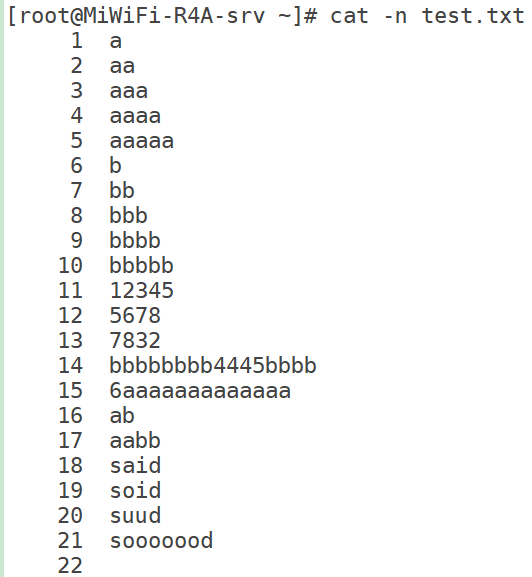


**“.\*”：**匹配所有内容

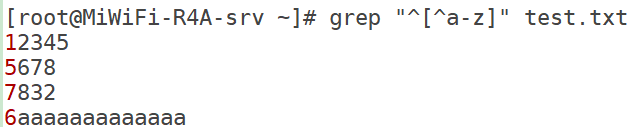


注意“a\*”与“.\*”的区别！

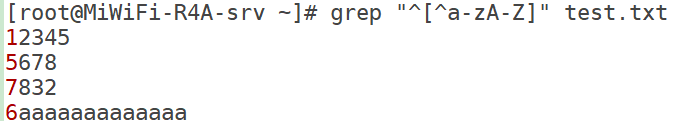
**文档内容：**



**“^[^a-z]”：**匹配不用小写字母开头的行

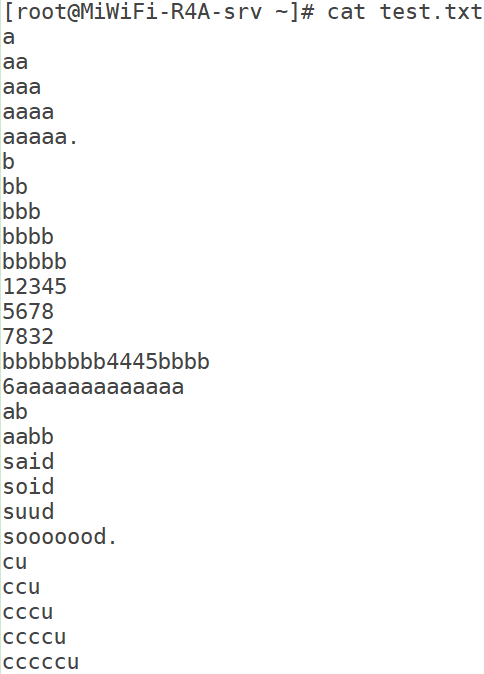


**“^[^a-zA-z]”：**匹配不用字母开头的行

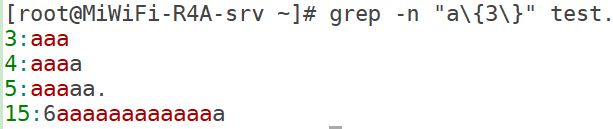


注：Linux中对大小写敏感，不支持A-z

**文档内容**：

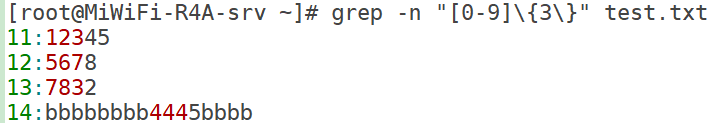


**“a\{3\}”**：匹配a字母连续出现三次的字符串



注意：该正则要实现桥号出现三次必须在前后加定界符（前后补习要有匹配的内容）

**“[0-9]\{3\}”**：匹配包含连续的三个数字的字符串

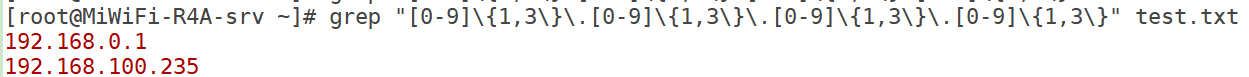


案例：

**[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}**：匹配日期格式



[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}：匹配IP地址



## 2. 字符截取命令

用来进行列截取，比如：需要统计系统所有普通用户的用户名，这就不能只用grep命令（如下图）。以下将这样的提取命令。

grep “/bin/bash” /etc/passwd | grep -v “root”



### 2.1 cut字段截取

**cut [选项] 文件名**

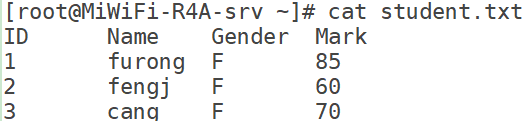
选项：

-f 列号：提取第几列

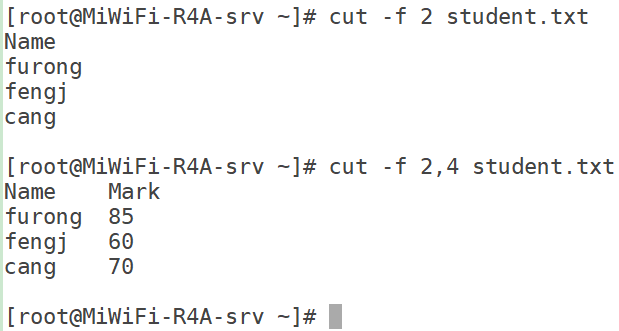
-d 分隔符：按照指定分隔符分割列（默认是一个TAB（制表符）分割）

**案例1**：

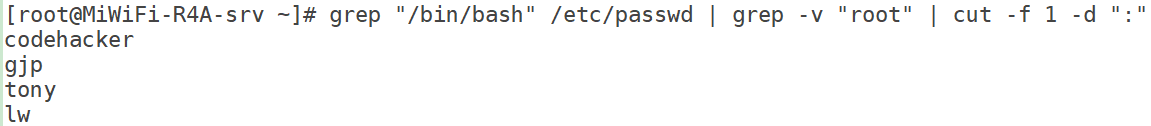
新建student文件，内容如下：



cut -f 2 student.txt



这是我们就可以回答刚刚提出的问题了：



注：cut命令会严格按照-d后面的分割符进行分割，比如你指定一个空格就是按一个空格进行分割。

### 2.2 printf命令

**printf ‘输出类型格式’ 输出内容**

**输出类型：**

%ns：输出字符串，n是数字，指代输出n个字符

%ni：输出整数，n指代输出n为数字

%m.nf：输出浮点数，m和n都是数字，指代输出的整数位数和小数位数，如%8.2f代表共输出8位数，其中2位小数，6位整数

**输出格式：**

\a：输出警告声音

\b：输出退格键

\f：清除屏幕

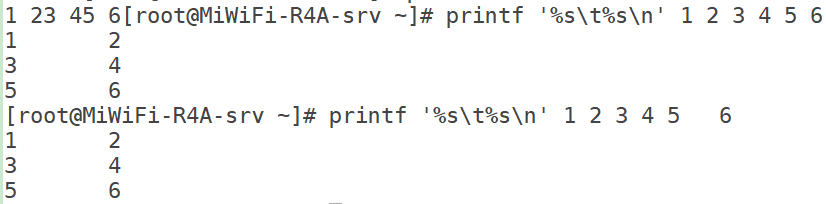
\n：换行

\r：回车

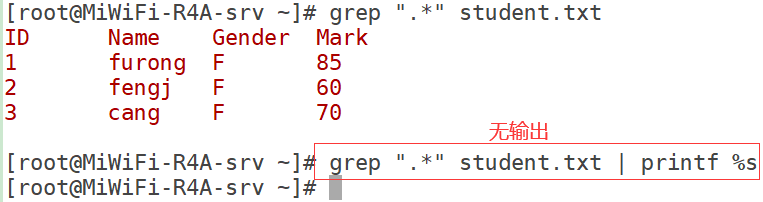
\t：水平输出TAB键

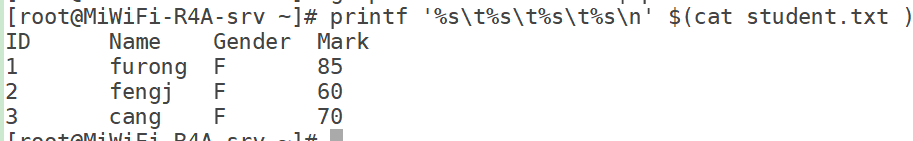
\v：垂直输出TAB键

printf是最底层的输出命令，echo是对printf进行了格式处理！



printf不支持数据流操作，也就是说不能在|后面直接接printf命令！





### 2.3 awk命令（过于复杂）

**awk ‘条件1{动作1} 条件2{动作2} …’ 文件名**

参考链接：https://man.linuxde.net/awk

条件：

一般用关系表达式作为条件

x > 10判断变量x是否大于10

x >= 10大于等于

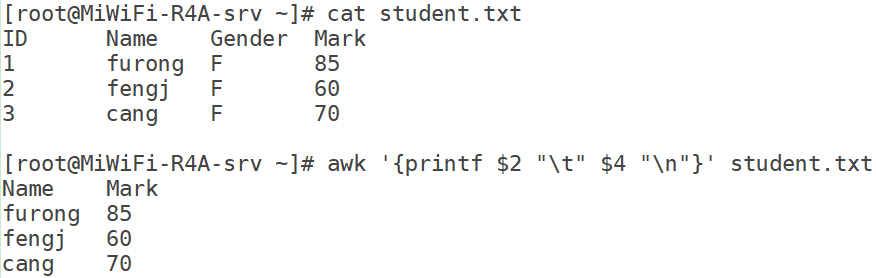
x <= 1=小于等于

动作：

格式化输出

流程控制语句

**案例1**：





$n取第n列

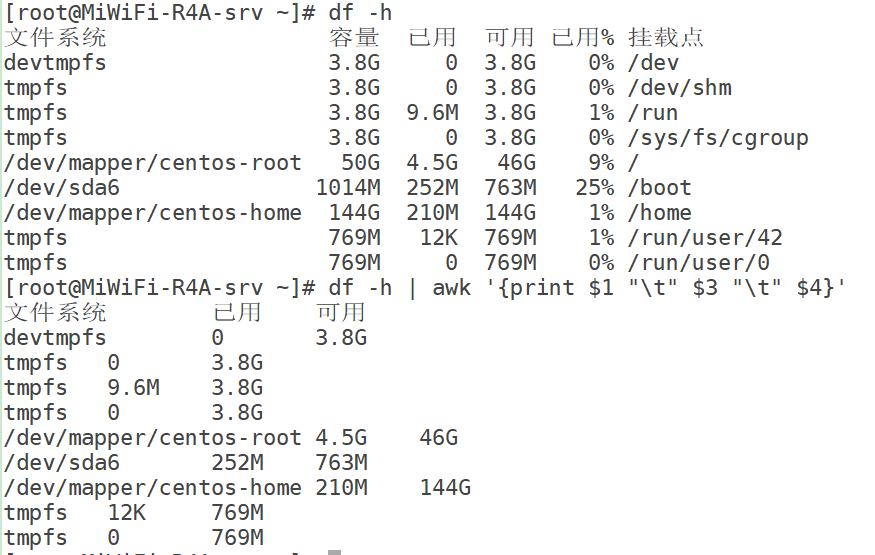
print会自动输出换行符，而printf则需要手动加入

**工作原理**：

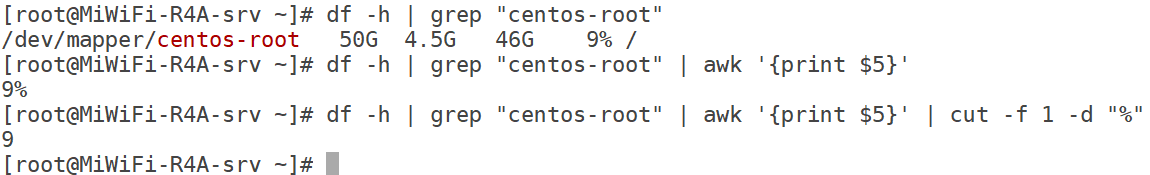
先将文件的以行为单位读取

判断是否符合添加，如果符合条件就执行后面的动作

**案例2**：截取df中的有效信息



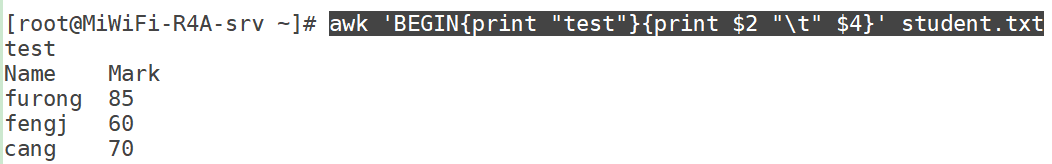
再截取根分区的信息：



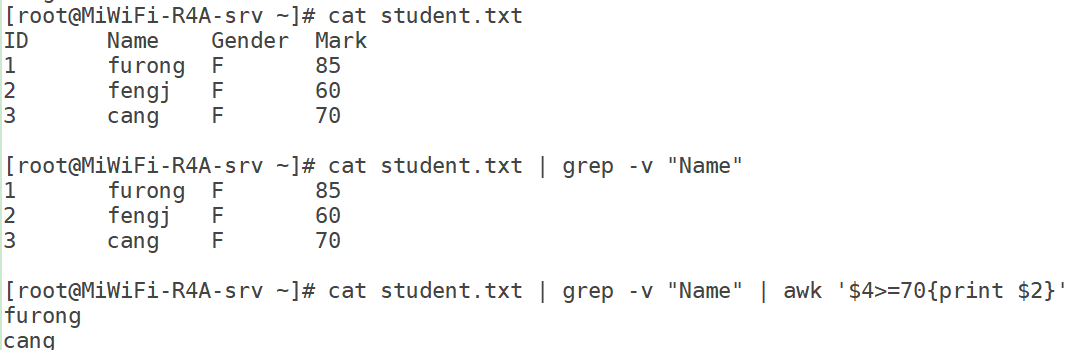
**关系表达式**：

**BEGIN**：在执行后面的命令之前先执行BEGIN后面的内容

awk '**BEGIN{print "test"}**{print $2 "\t" $4}' student.txt



cat student.txt | grep -v "Name" | **awk '$4>=70{print $2}'**



### 2.4 sed命令

sed是一种几乎包括在所有UNIX平台（包括Linux）的轻量级流编辑器。sed主要是用来将数据进行选取、替换、删除、新增的命令。

**sed [选项] ‘[动作]’ 文件名**

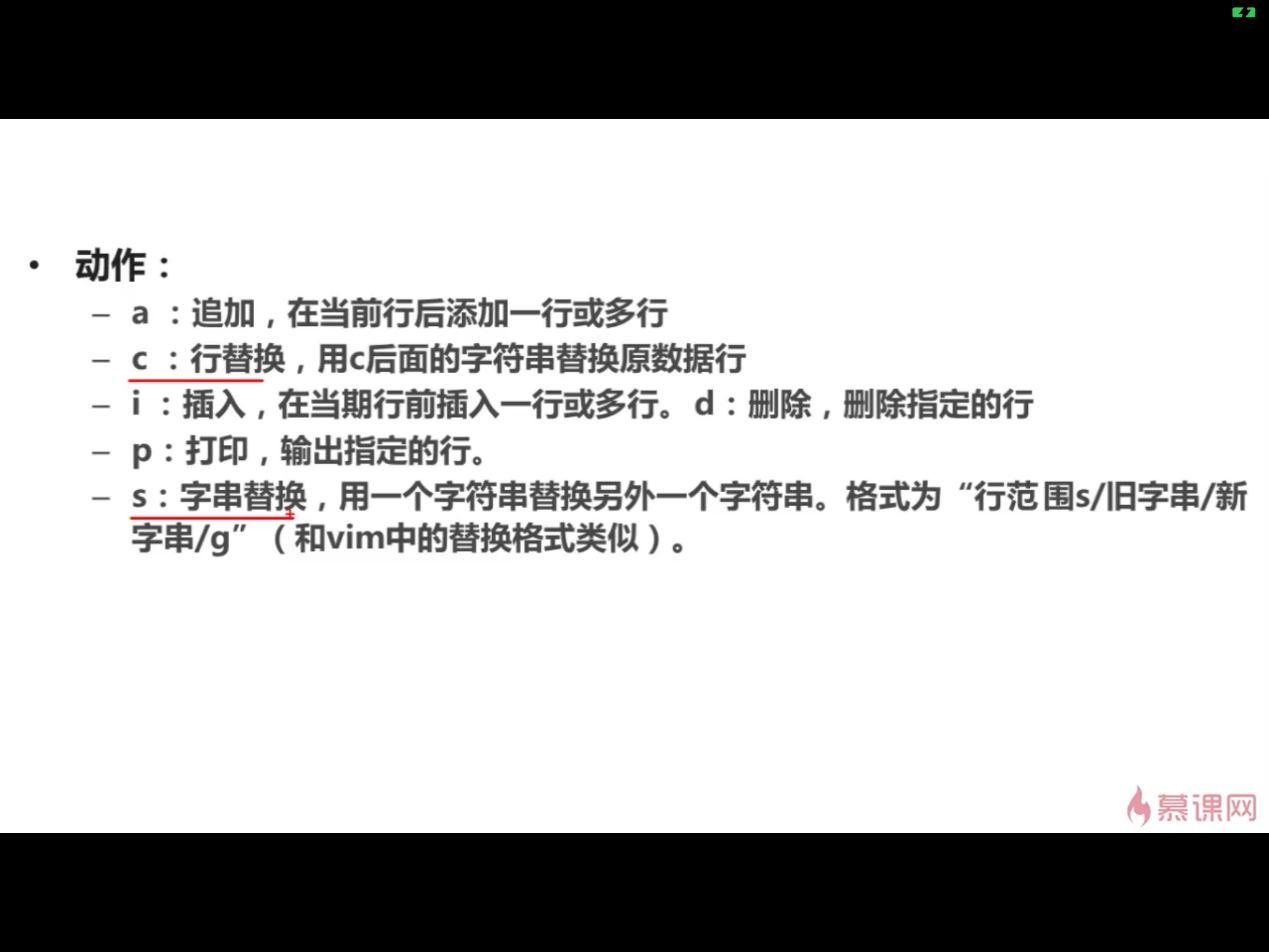
选项：

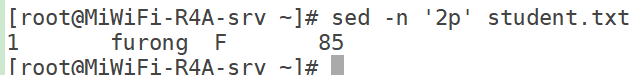
-n：一般sed命令会把所有数据都输出到屏幕，如果加入此选项，则只会把经过sed命令处理的行输出到屏幕

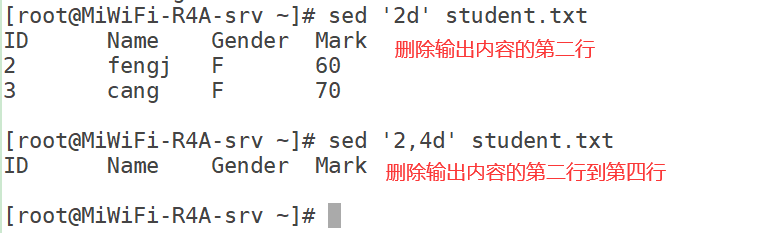
-e：允许对输入数据应用多条sed命令编辑

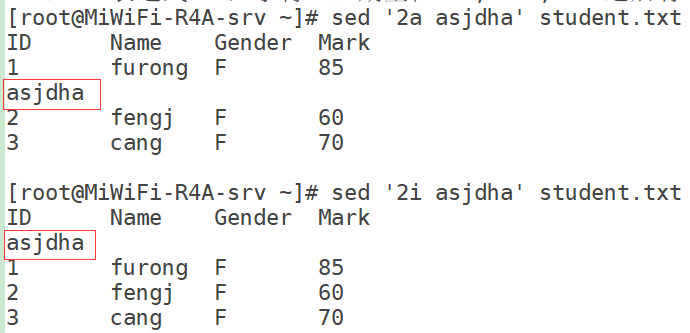
-i：用sed的修改结果直接修改读取数据文件，而不是由屏幕输出

动作：









## 3. 字符处理命令

### 3.1 sort命令

**sort [选项] 文件名** 排序命令

选项：

-f：忽略大小写

-n：以数值型进行排序，默认使用字符串型排序

-r：反向排序

-t：指定分割符，默认是TAB

-k n[,m]：按照指定的字段范围排序。从第n字段开始，m字段介绍（默认到行尾）

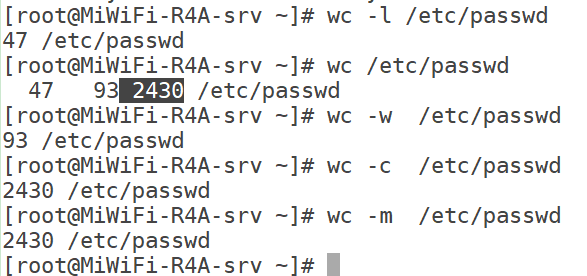
**sort -n -t ":" -k "3,3" /etc/passwd**



### 3.2 wc命令

**wc [选项] 文件名**  统计命令

选项：**-l**：只统计行数 **-w**：只统计单词数 **-m**：只统计字符数

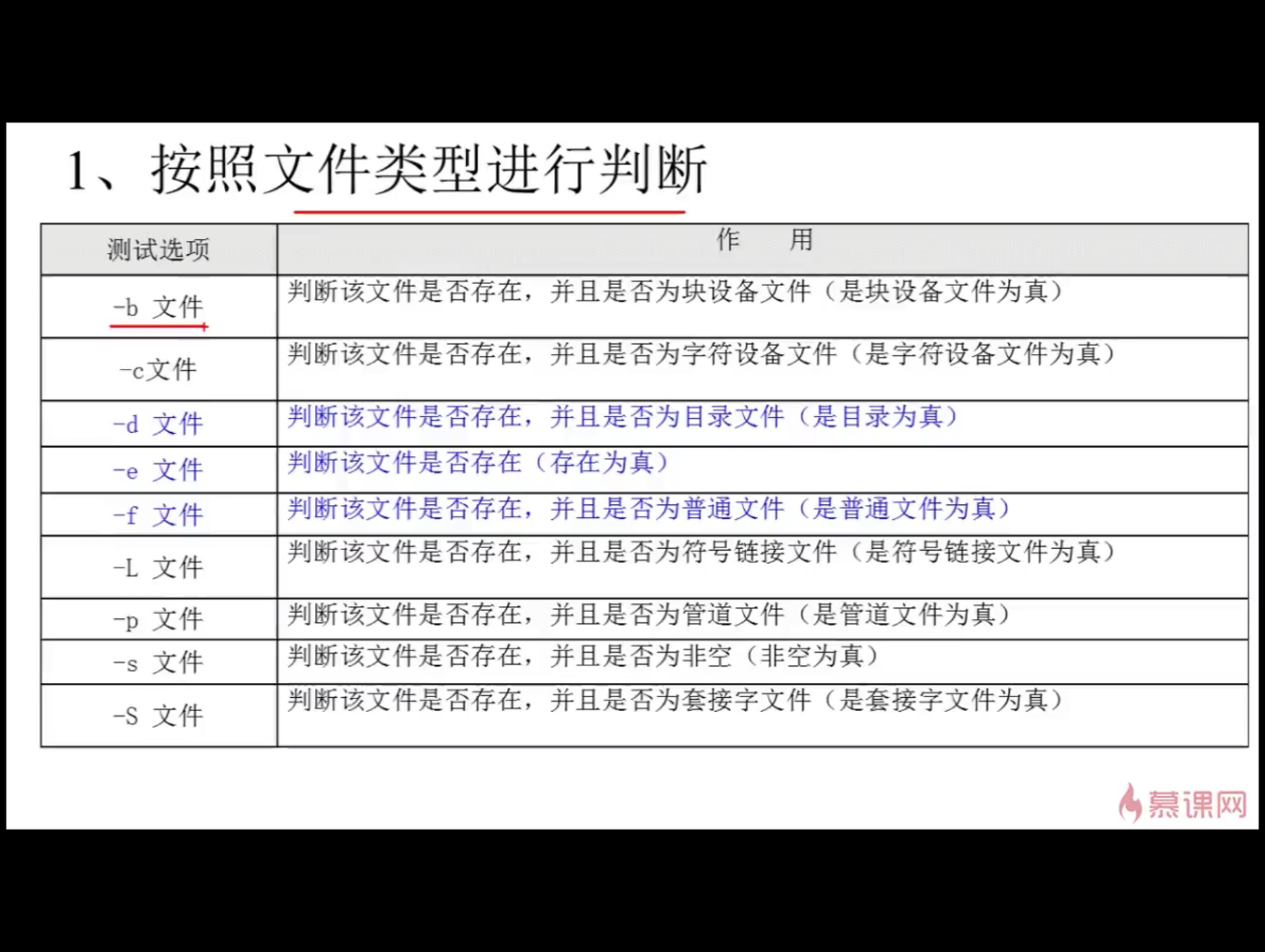


# 条件判断与流程控制

## 1. 条件判断语句

### 1.1 概述

#### 1.1.1 按照文件类型进行判断

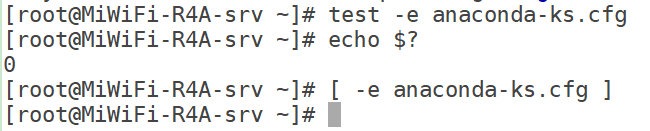


**两种判断格式**：

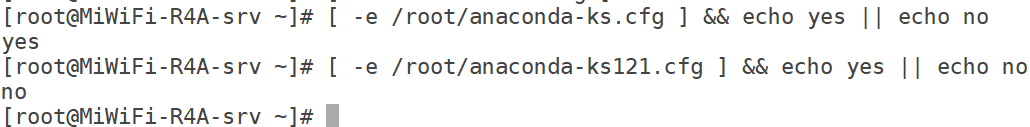
**test -e /root/anaconda-ks.cfg**

或

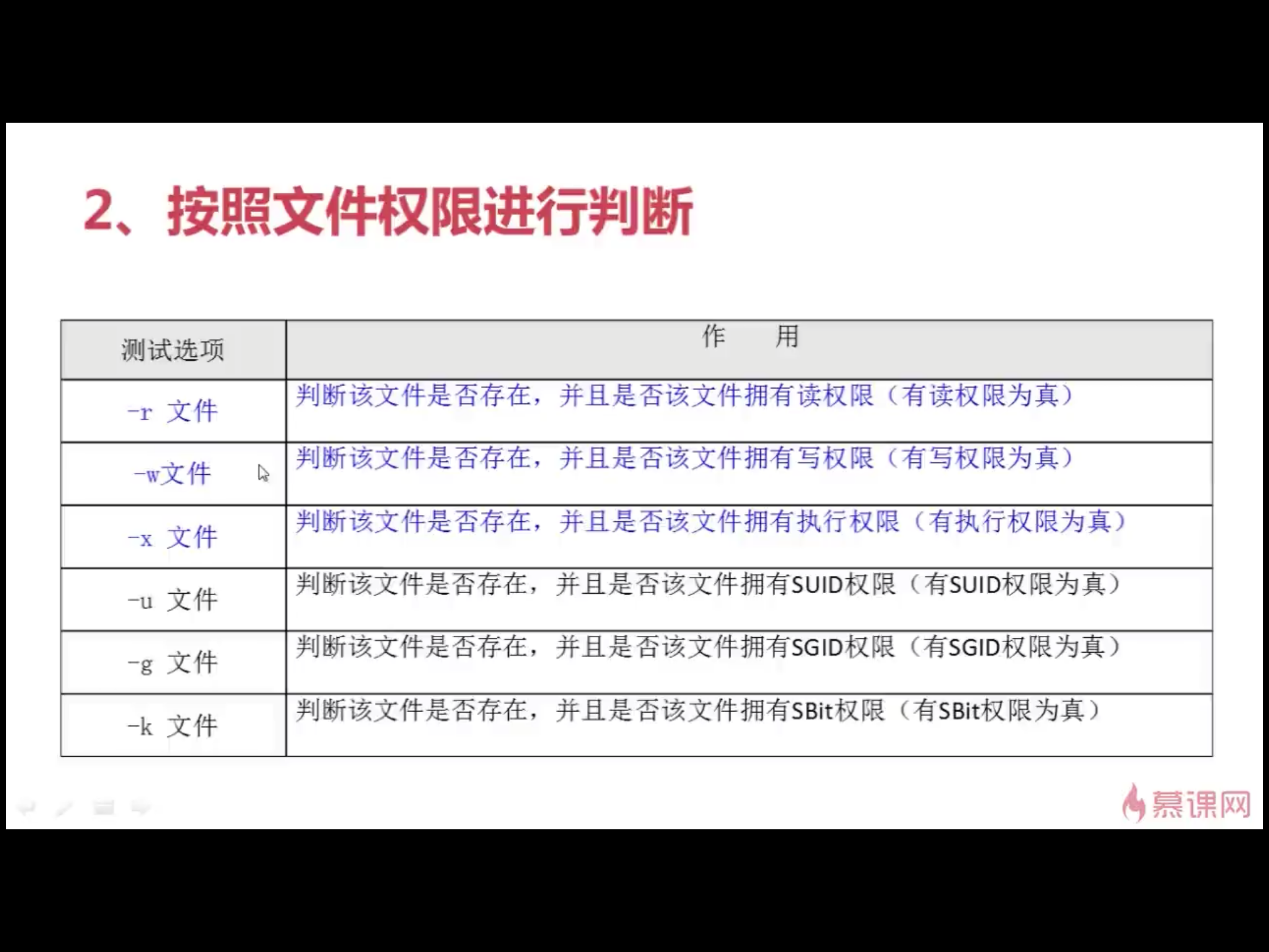
**[ -e /root/anaconda-ks.cfg ]** （推荐使用）



**[ -e /root/anaconda-ks.cfg ] && echo yes || echo no**

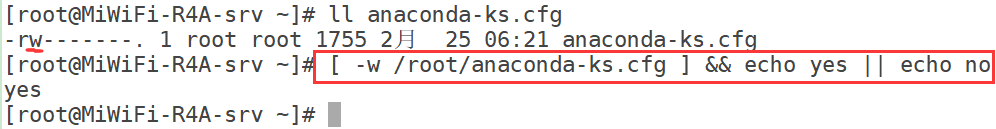


#### 1.1.2 按照文件权限判断



**[ -w /root/anaconda-ks.cfg ] && echo yes || echo no**

只要有写权限就会正确执行，不管是哪种身份



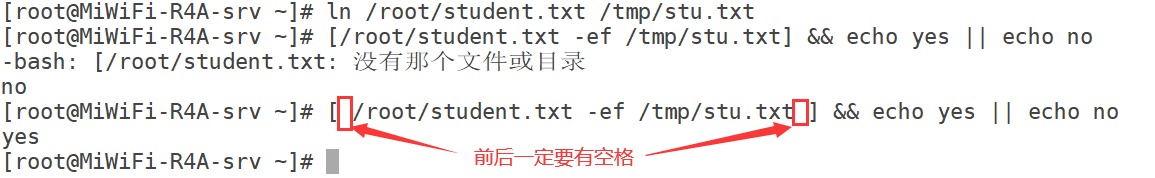
#### 1.1.3 两个文件进行比较



**案例**：

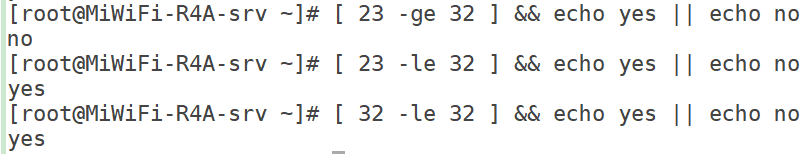
**ln /root/student.txt /tmp/stu.txt**

**[/root/student.txt -ef /tmp/stu.txt] && echo yes || echo no**

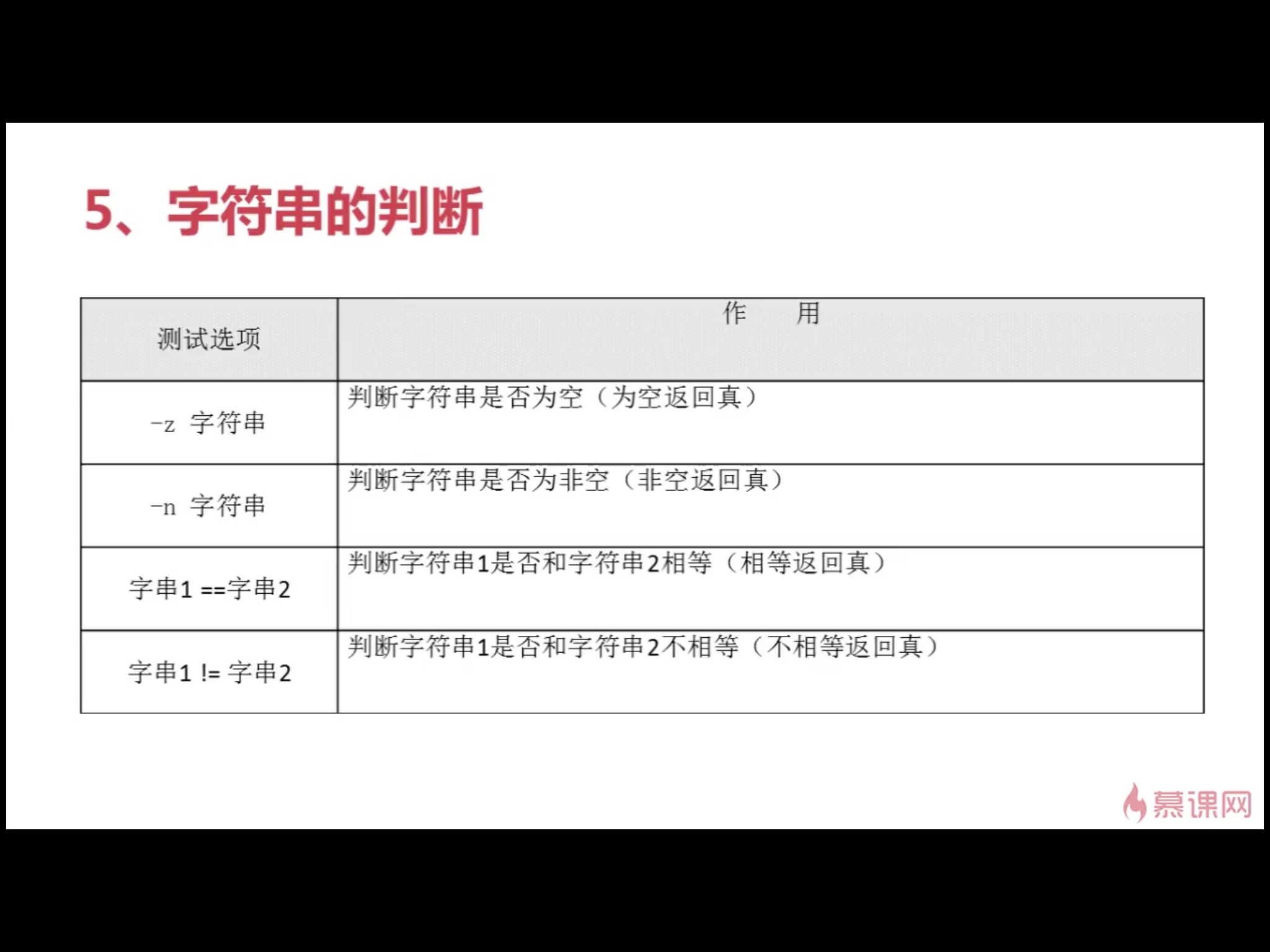


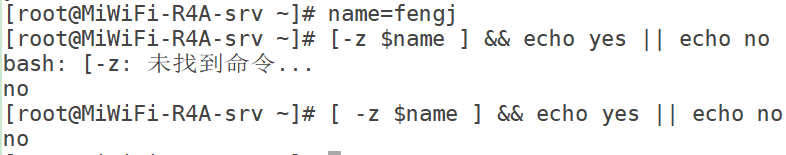
#### 1.1.4 两个整数之间的比较





#### 1.1.5 字符串的判断





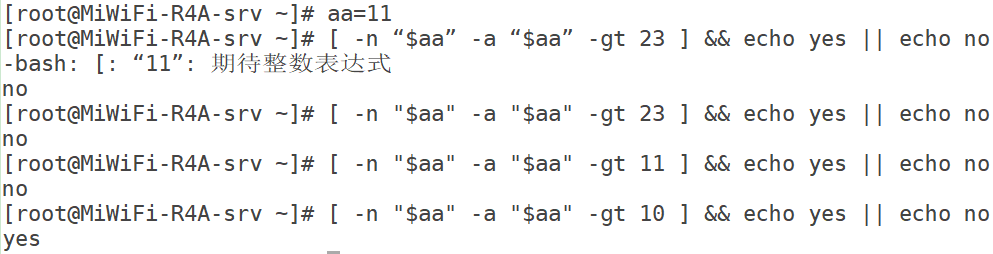
#### 1.1.6 多重条件判断



**案例：**

**aa=11**

**[ -n “$aa” -a “$aa” -gt 23 ] && echo yes || echo no**



学习脚本实例的好处：

1.掌握Linux语法结构

2.了解Shell的作用

3.分析编程思想（多用多总结）

### 1.2 单分支if语句

**if [ 条件判断式 ];then**

**程序**

**fi**

或者

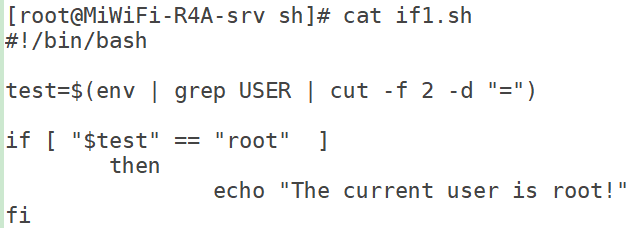
**if [ 条件判断式 ]**

**then**

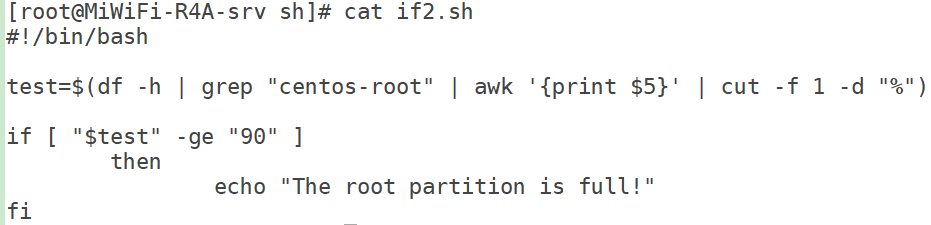
**程序**

**fi**

判断当前用户是不是root



判断根分区的使用量是否大于90%



### 1.3 双分支if语句

**if [ 条件判断式 ]**

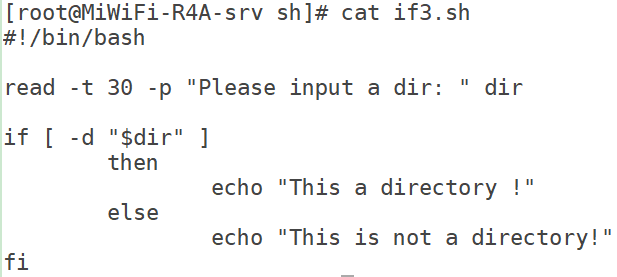
**then**

**条件成立时，执行程序**

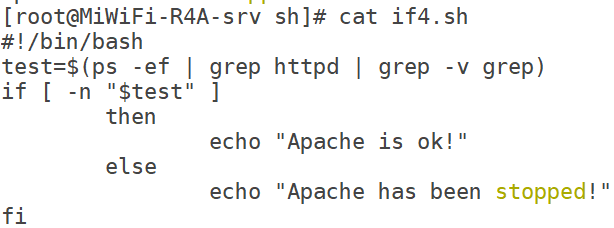
**else**

**条件不成立时，执行的另一个程序**

**fi**



实例：判断apache服务是否启动



### 1.4 多分支if语句

**if [ 条件判断式1 ]**

**then**

**条件判断式1成立时，执行程序1**

**elif [ 条件判断式2 ]**

**then**

**当条件判断式2成立时，执行程序2**

…省略更多的条件…

**else**

**当所有条件不成立时，最后执行此程序**

**fi**

**案例1**：实现简易的纯字符界面的计算器

#!/bin/bash

# 纯字符界面的简易计算器

read -t 30 -p "Please input number 1: " num1

read -t 30 -p "Please input number 2: " num2

read -t 30 -p "Input a operator(+|-|\*|/): " ope

if [ -n "$num1" -a -n "$num2" -a -n "$ope" ]

then

test1=$(echo $num1 | sed 's/[0-9]//g')

test2=$(echo $num2 | sed 's/[0-9]//g')

testope=$(echo "+-\*/" | grep "$ope")

if [ -z "$test1" -a -z "$test2" -a -n "$testope" ]

then

if [ "$ope" == "+" ]

then

result233=$(( $num1 + $num2 ))

elif [ "$ope" == "-" ]

then

result233=$(( $num1 - $num2 ))

elif [ "$ope" == "\*" ]

then

result233=$(( $num1 \* $num2 ))

elif [ "$num2" -eq "0" ]

then

echo "Error : / divided zero!"

exit 11

else

result233=$(( $num1 / $num2 ))

fi

echo "$num1 $ope $num2 = $result233"

else

echo "Input erorr!"

exit 10

fi

else

echo "Input is null!"

exit 9

fi

### 1.5 多分支case语句

case语句和if…elif…else语句一样都是多分支条件语句，不过和if多分支语句不同的是，case语句只能判断一种条件关系，而if语句可以判断多种条件关系。

语法：

**case $变量名 in**

**“值1”)**

**如果变量的值等于值1，则执行程序1**

**;;**

**“值2”)**

**如果变量的值等于值2，则执行程序2**

**;;**

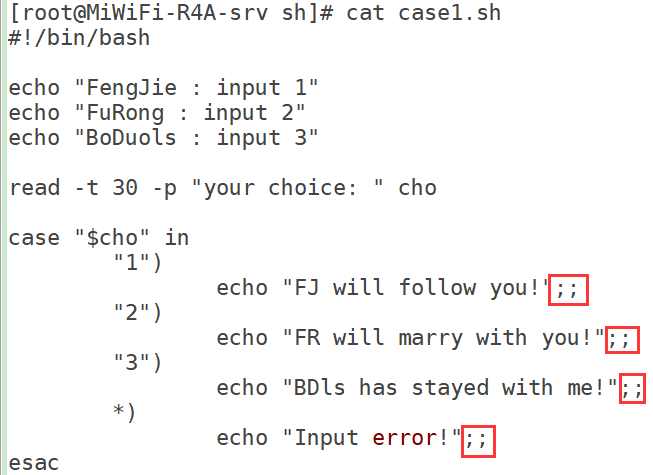
…省略其他分支…

**\*)**

**如果变量的值都不是以上的值，则执行此程序**

**;;**

**esac**



## 2. 循环语句

### 2.1 for循环

#### 2.1.1 语法1

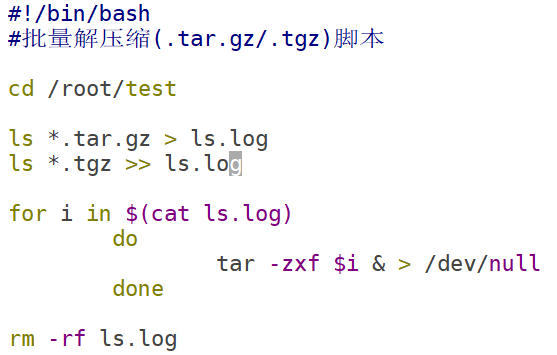
**for 变量 in 值1 值2 值3**

**do**

**程序**

**done**

案例1：批量解压特定目录下的.tar.gz或.tgz压缩包



#### 2.1.2 语法2

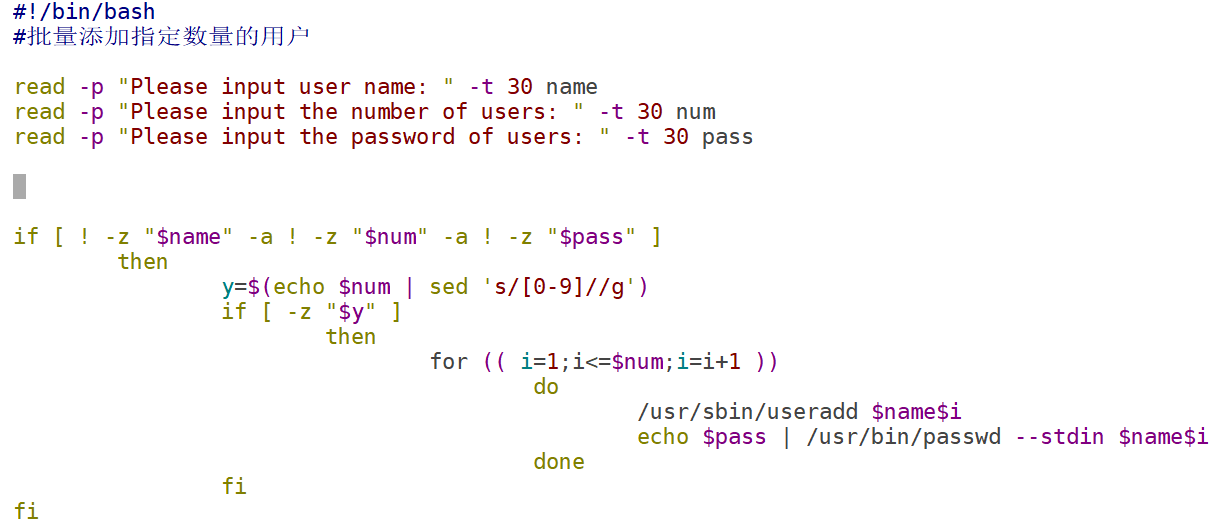
**for (( 初始值;循环控制条件;变量变化 ))**

**do**

**程序**

**done**

案例2：批量添加普通用户



案例3：删除所有普通用户



### 2.2 while和until循环

#### 2.2.1 while循环

while循环是不定循环，也称条件循环。只要条件判断式成立，循环就会一直继续，知道条件判断式不成立，循环才会停止。这就和for的固定循环不太一样了。

**语法**：

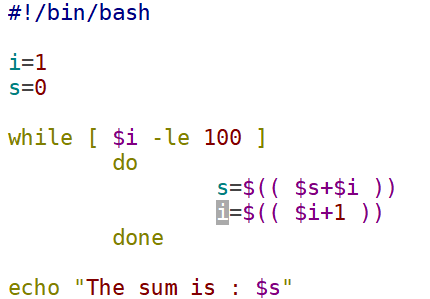
**while [ 条件判断式 ]**

**do**

**程序**

**done**

案例1：从一加到一百



#### 2.2.2 until循环

until循环和while循环相反，until循环时只要条件判断式不成立则进行循环，并执行循环程序，一旦循环条件成立则终止循环。

**语法**：

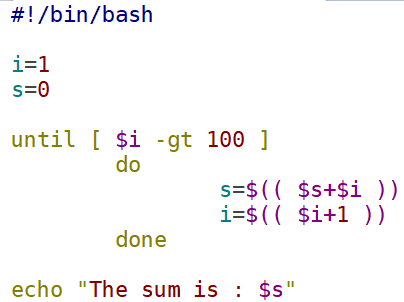
**until [ 条件判断式 ]**

**do**

**程序**（条件不成立才执行）

**done**

案例2：从1加到100



## 3. 总结

- 编写shell程序最主要用来简化管理员操作

- shell编程更多的考虑程序的功能实现，其次才是效率