# #CPU架构

CPU架构：精简指令集和复杂指令集

## 1.精简指令集

设计理念:每个指令集较为精简，每个指令的运行时间都很短，完成的动作很单纯，执行效能较佳；

但是要做复杂的事情，需要多个指令来完成。

应用领域:常用于学术领域的大型工作站。

典型案例：ARM架构

## 2.复杂指令集

设计理念:每个小指令可以执行一些较为低阶的硬件操作，指令数目多而且复杂。因为指令执行较为复杂

所以每条指令花费的时间较长，但每条个别指令可以处理的工作较为丰富。

应用领域:个人计算机

典型案例:x86架构

# #简单命令

date 显示日期和时间

cal [month] [year] 显示日历

bc 简易计算器

ls -a -l -h 显示目录

mkdir -p 递归创建目录或文件

ll、cp、rm、mv、rmdir、touch

文件查阅

cat -b(列出行号) 由第一行开始显示文件内容

nl 显示的文件加上行号

less 一页一页显示文件，支持前后翻页

head [-n number] 只看前几行数据

tail [-n number] 只看后几行数据

文件查找

which -a(所有) 查找可执行的文件

whereis 在特定的目录中寻找文件或目录名

# #Vim编辑器

vim 命令常用快捷键

## 1.一般模式

***ctrl f***  向下移动一页

***ctrl b***  向上移动一页

***n<space>***  光标向后面移动n个字符

***0***  移动到当前行的第一列

***$*** 移动到当前行的最后一列

***G***  移动到这文件的最后一行

***gg***  移动到这个文件第一行

***/word***  向光标之下寻找名称为word的字符串

***?word***  向光标之上寻找名称为word的字符串

***X、x***  删除一个字符

***dd*** 删除光标所在的行

***yy***  复制光标所在的行

***p,P***  粘贴已复制的数据

***u*** 撤销前一个动作

***Ctrl r***  重做上一个动作

***.***  重复前一个动作

## 2.指令模式

***:set nu*** 显示行号

***:n1,n2s/word1/word2/gc*** n1和n2为数字，在第和之间寻找word1，并询问是否将该字符串取代为word2.

## 3.区块选择

v 字符选择，会将光标经过的地方反白选择

V 行选择，会将光标经过的地方反白选择

y 将反白的地方复制起来

d 将方反白的地方删除掉

p 将刚刚复制的区块，在游标所在处贴上

## 4.存档与恢复

(1)多人协同同时编辑一个文件，此时会提示错误。找到另外一个同时编辑的人。

(2)网络中断，重新编辑就问题时，会提示警告信息。并且会生产.swp结尾的文件

解决办法:vim -r xxx,然后删除.swp的文件.

# #磁盘管理与分区

硬盘说明：

(1)Linux硬盘分为IDE硬盘和SCSI硬盘，目前基本都是SCSI硬盘。

(2)对于IDE硬盘，驱动器标识符符为”hdx~”，hd表示分区所在设备的类型，这里指IDE硬盘。”x”表示盘号(a，b，c);”~”代表分区，前四个分区用1到4表示，是主分区或者拓展分区，从5开始就是逻辑分区。

(3)对于SCSI硬盘标识符为”sdx~”,SCSI硬盘是用”sd”表示分区所在设备的类型。

(4)正常的实体机器大概使用的都是/dev/sd[a-]的磁盘文件名，至于虚拟机环境底下，为了加速，可能就会使用/dev/vd[a-p] 这种装置文件名。

(5) SCSI/SATA/USB硬盘机这些装置，在Linux内的文件名为/dev/sd[a-p]。

## 1.MBR分区

早期的Linux 系统为了兼容于 Windows 的磁盘，因此使用的是支持 Windows 的 MBR(Master Boot Record, 主要开机纪录区) 的方式来处理开机管理程序与分区表。

MBR主分区、拓展分区与逻辑分区的概述：

①主分区和拓展分区总共最多只能分四个(硬盘的限制)。

②拓展分区最多只能有一个(操作系统的限制)。

③逻辑分区是由拓展分区持续切割出来的分区槽。

④能够被格式化后，作为数据存取的分区槽为主分区和逻辑分区。拓展分区无法格式化。

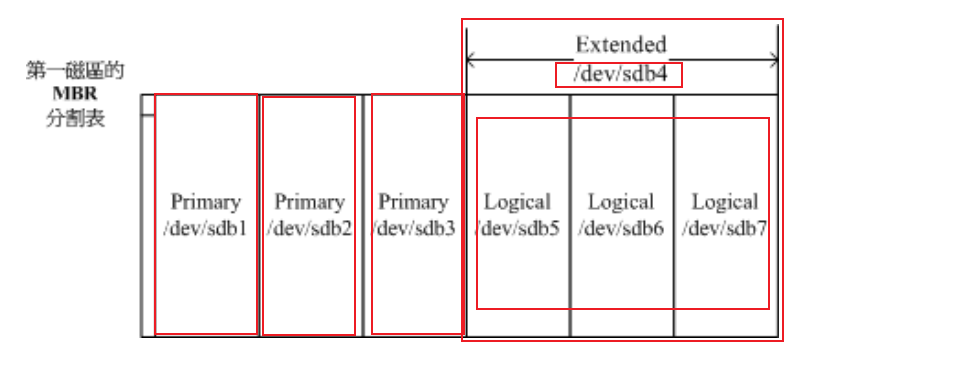
⑤拓展分区的目的是使用额外的扇区来记录分区信息，拓展分区本身不能被拿来格式化。拓展分区必须再划分成逻辑分区才能使用。

分区分配实例：

假如一台PC有两块SATA硬盘，想在第二块硬盘分出6个可用的分区槽(可以被格式化来存储数据之用)，则每个分区槽在Linux系统下的装置名为何，且分区类型各为何，至少写出两种不同的分区方式。

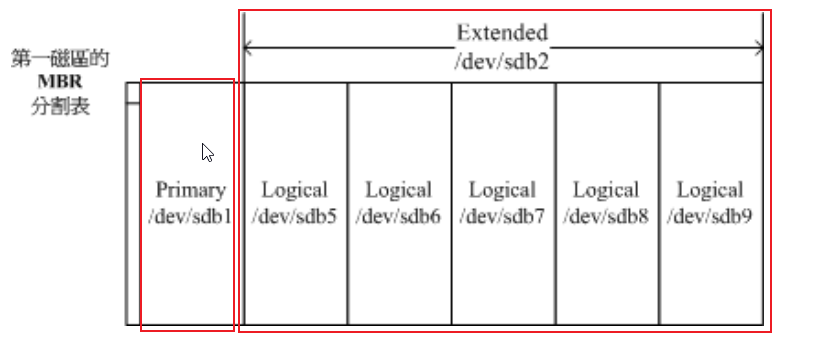
解答：由于P(primary)+E(extended)最多只能有四个，其中E最多只能有一个。现在要求6个可用的分区槽，因此不可能分出四个P。因此只存在两种情况：一种是将前四号全部用完，一种是仅花费一个P及一个E的情况。

①P+P+P+E模式



实际可用的是/dev/sdb1, /dev/sdb2, /dev/sdb3, /dev/sdb5, /dev/sdb6, /dev/sdb7这六个，至于/dev/sdb4这个延伸分区本身仅是提供来给逻辑分区槽建立之用。

②P+E模式



因为1~4号是保留给主要/延伸分区槽的，因此第一个逻辑分区槽一定是由5号开始的，故/dev/sdb3, /dev/sdb4就会被保留下来没有用到了。

## 2.GPT分区

GPT的意思是GUID Partition Table，即“全局唯一标识磁盘分区表”。是另外一种更加先进新颖的磁盘组织方式，一种使用UEFI启动的磁盘组织方式。最开始是为了更好的兼容性，后来因为其更大的支持内存（mbr分区最多支持2T的磁盘），更多的兼容而被广泛使用，特别是苹果的MAC系统全部使用gpt分区。gtp不在有分区的概念，所有CDEF盘都在一段信息中存储。可以简单的理解为更先进但是使用不够广泛的技术。

## 3.MBR与GPT的区别

①分区：mbr最多支持四个主分区，gpt没有限制。如果想跑多系统，mbr最多4个而gpt没有限制。

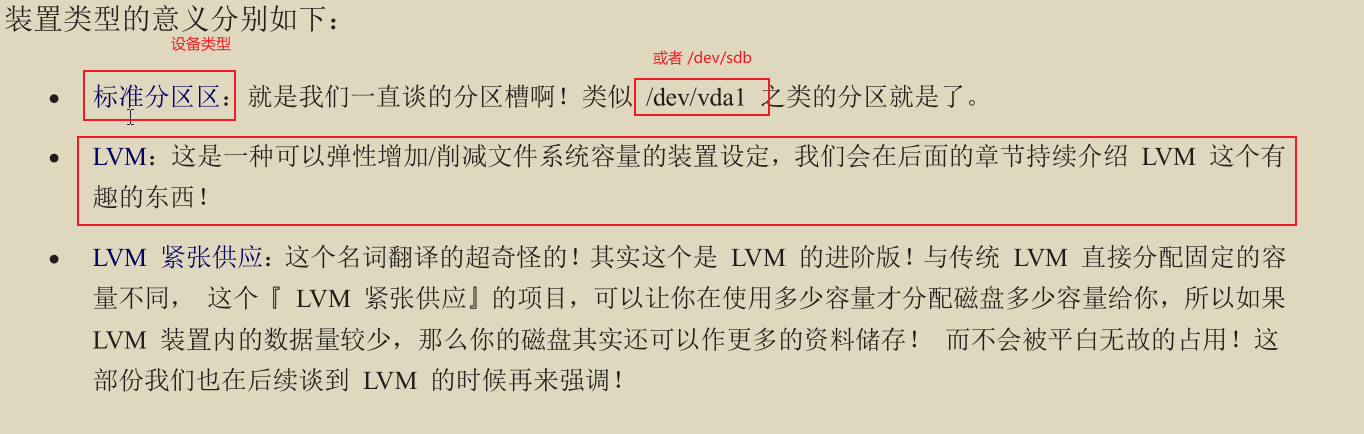
②系统：win7只能用mbr分区（也可以但是很麻烦，不建议），从Win8开始微软建议你使用gpt。

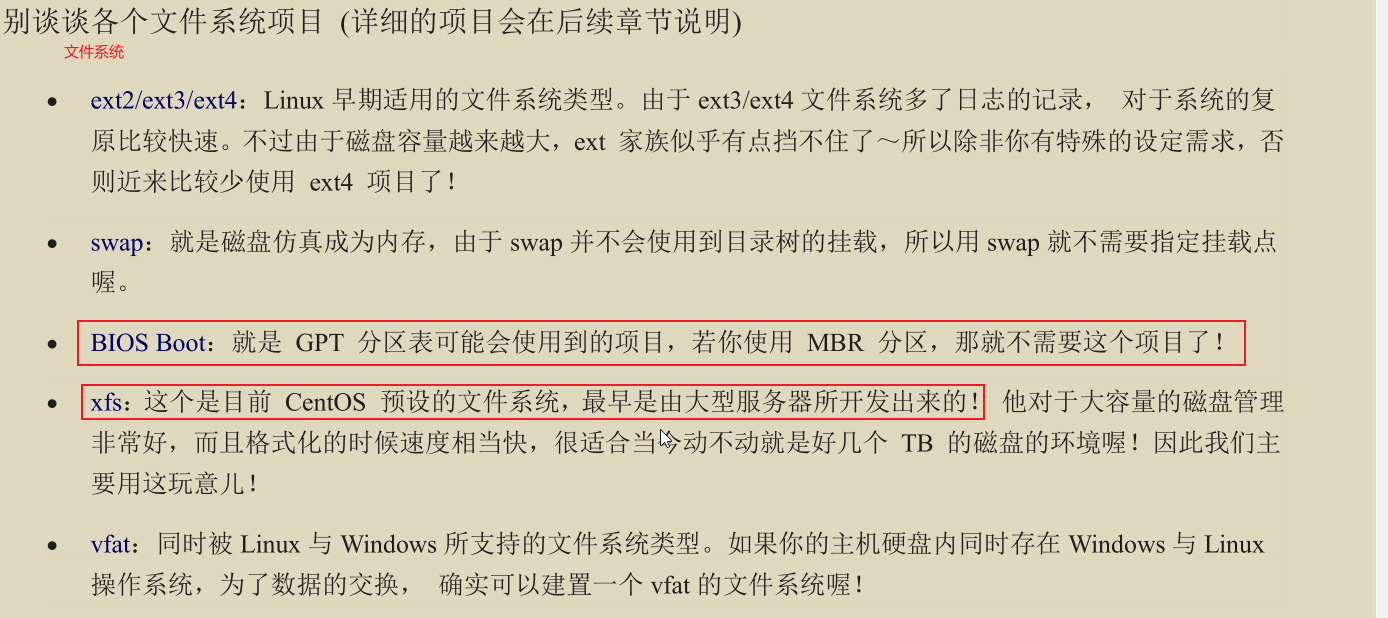
③其他：gpt是由uefi启动的，而uefi是后来才提出的概念，兼容性和稳定性不如bios+mbr。

常用命令

## 4.文件系统

文件系统通常会将两部分数据放在不同的区块，权限和属性放置到inode中，实际数据放置在datablock区块中。另外 还有一个超级区块(superblock)会记录整个文件系统的整体信息，包括inode与block的总量、使用量、剩余量等信息。





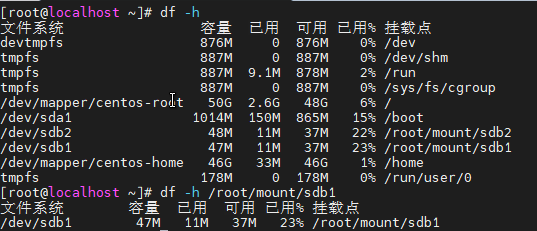
## 5.文件系统的操作

### df

列出文件系统的整体磁盘使用量，可用于监控主机磁盘告警信息

df -h使用习惯单位显示磁盘使用率

df -h /root/mount/sdb1 使用习惯单位显示指定挂载点磁盘使用率

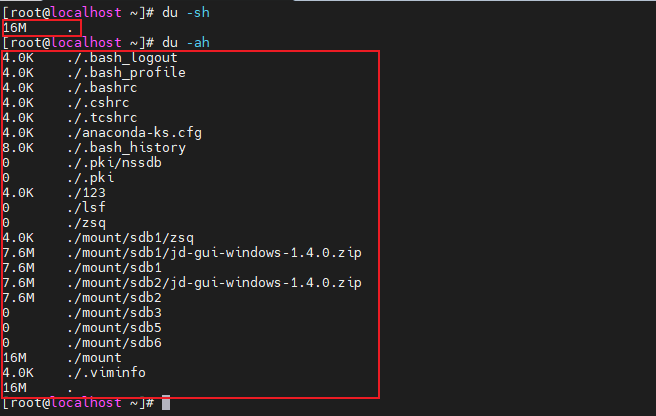


### du

评估文件系统的磁盘使用量(常用在估计目录所占容量), 统计目录或文件大小

du -sh列出当前目录下的文件和子目录的磁盘占用总量之和

du -ah列出当前目录下全部文件和子目录的磁盘占用量



df与du的区别

①df命令是从文件系统考虑的，不光要考虑文件占用的空间，还要统计被命令或程序占用的空间(最常见的就是文件已经删除，但是程序并没有释放空间)

②du命令是面向文件的，只会计算文件或目录占用的空间

### ln

链接命令

ln [-sf] source target

-s 建立软链接，默认参数是硬链接

-f 如果目标文件存在，就主动的将目标文件直接删除后再建立

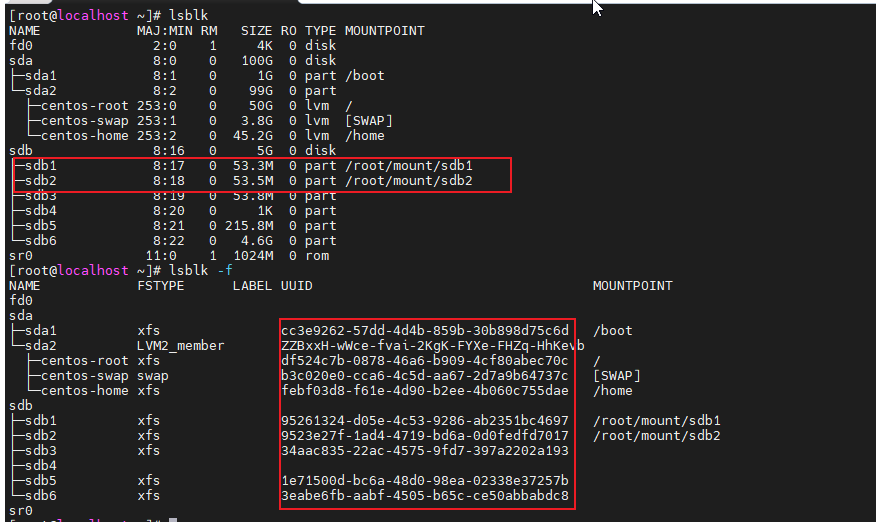
### lsblk

lsblk

列出系统上的所有磁盘列表

lsblk -f

列出该磁盘内的文件系统名称

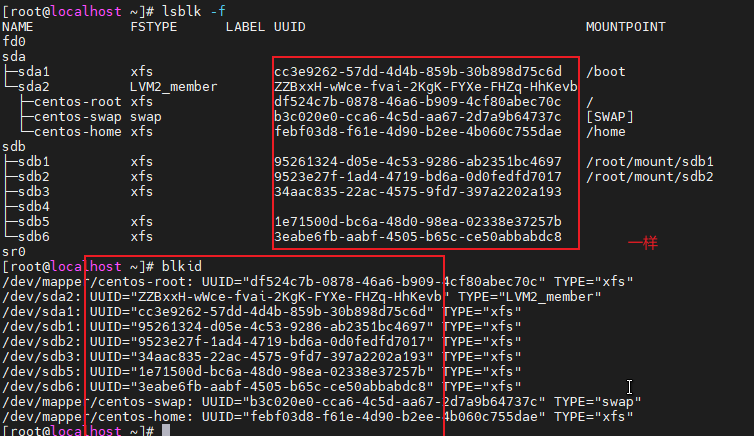


### blkid

列出设备的UUID等参数

lsblk -f

同样可以列出设备的UUID等参数



### parted

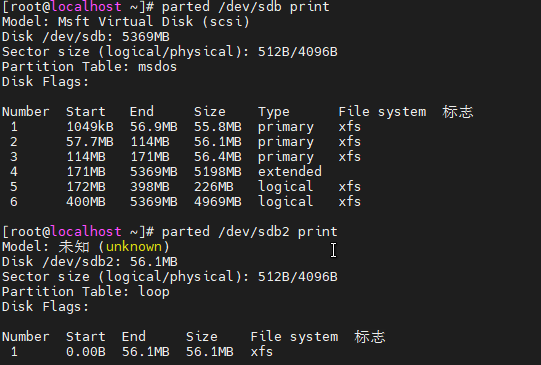
列出磁盘分区表类型与分区信息

parted /dev/sdb print

列出整个设备的分区信息

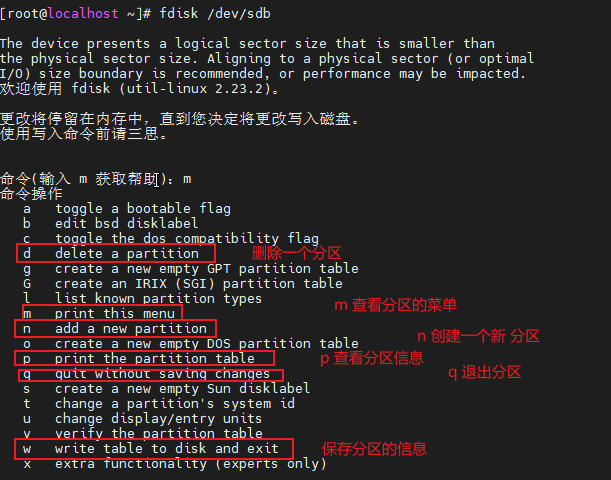
parted /dev/sdb2 print

列出设备某个分区的信息



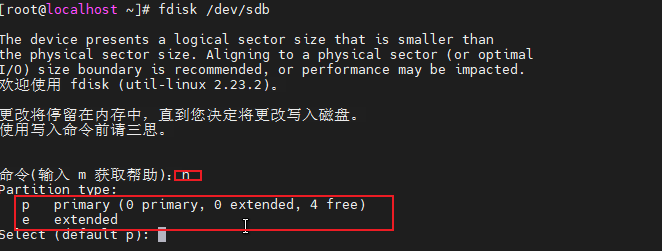
### fdisk

fdisk /dev/sdb



分区步骤：

①输入命令:n,开始新分区



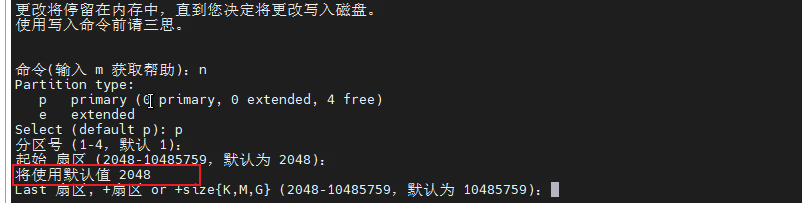
②选择主分区还拓展分区



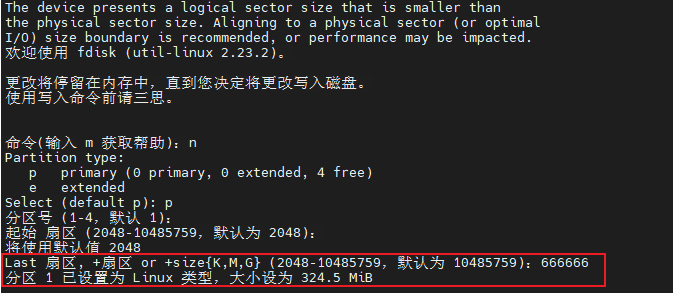
③选择分区号(默认都是主分区：总共四个：包含主分区和拓展分区)



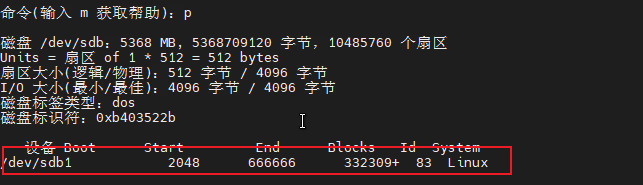
④选择起始扇区



⑤选择结束扇区

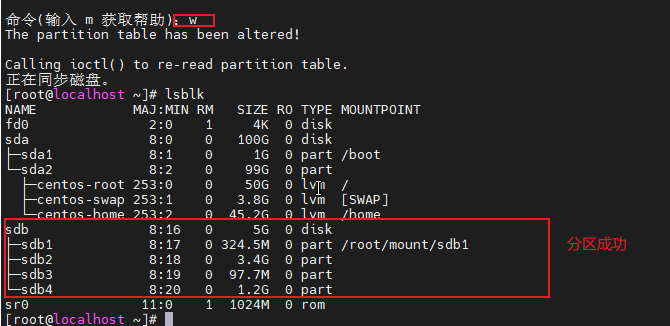


⑥分区成功



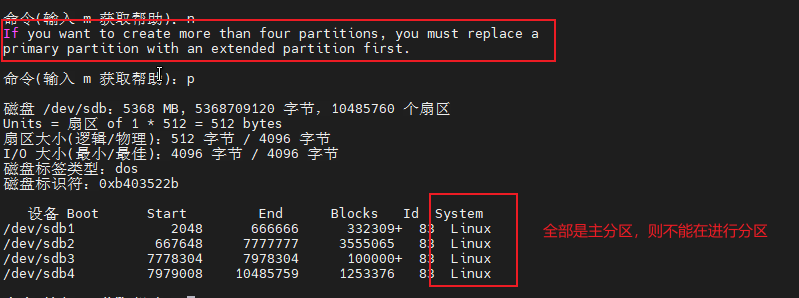
⑦继续进行分区，重复步骤①时

⑧分区结束，保存退出



注意：

如果分区数量大于4，则一定要保留一个拓展分区，否则，最多只能分四个主分区。



### mount

mount -a

依照配置文件/etc/fstab的数据将所有未挂载的磁盘都挂载上来

mount -l

显示全部的挂载信息

mount 装置名 挂载点

用法: mount /dev/sdb1 /root/mount/sdb1

mount UUID=**''** 挂载点

用法：mount UUID=**'9523e27f-1ad4-4719-bd6a-0d0fedfd7017'** /root/mount/sdb2

umount 将设备文件卸载

umount ***[***-fn***]*** 卸载装置文件名或挂载点

-f 强制卸载

-l 立即卸载文件系统，比-f还强

-n 不更新/etc/mtab情况卸载

umount /root/mount/sdb

建议用挂载点卸载，因为装置可能被其他方式挂载

umount /dev/sdb2

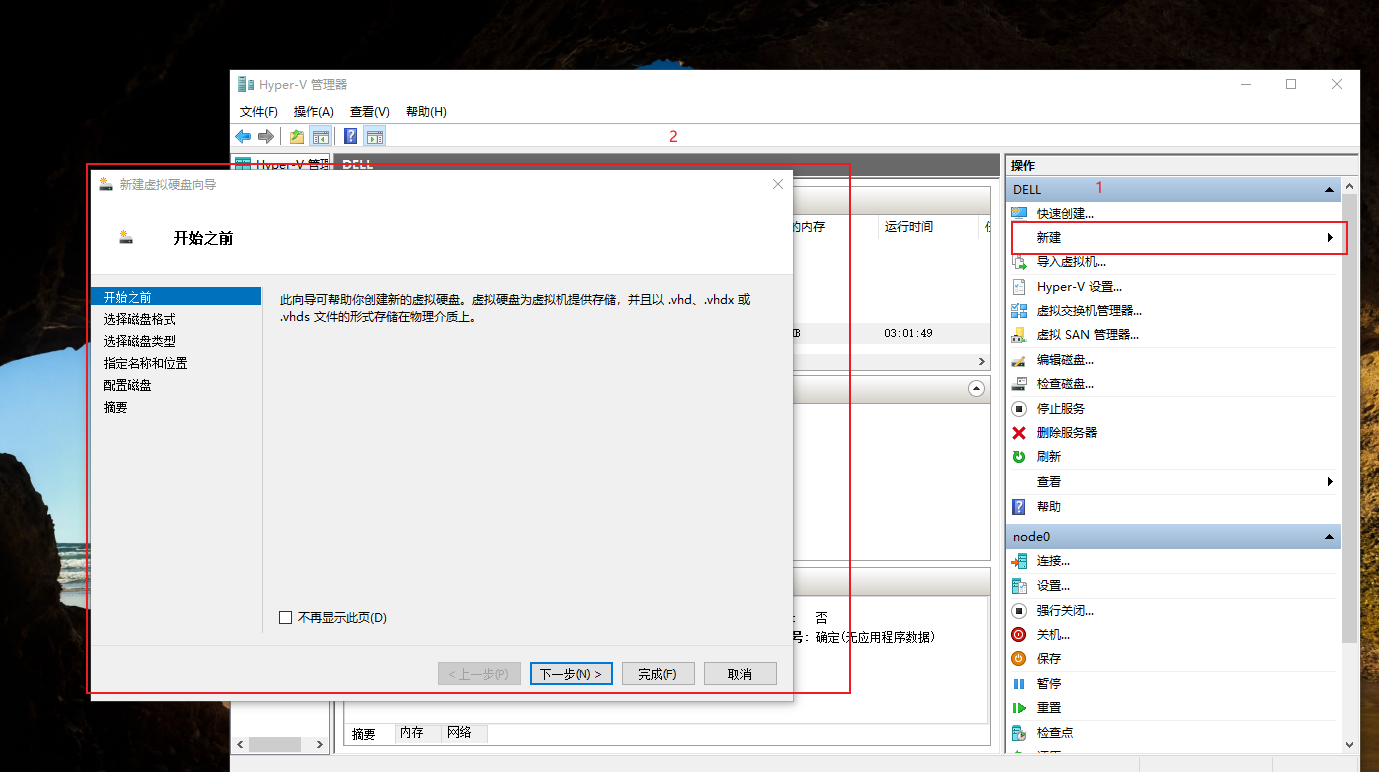
### mkfs.xfs

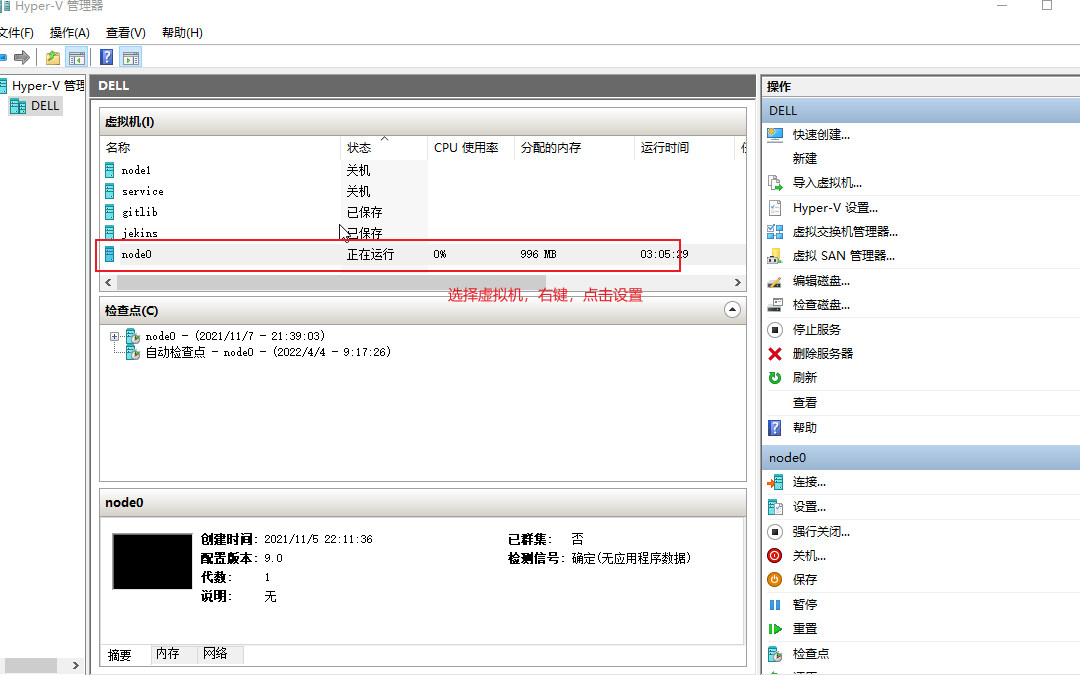
磁盘分区完成以后就是对文件系统进行格式化。

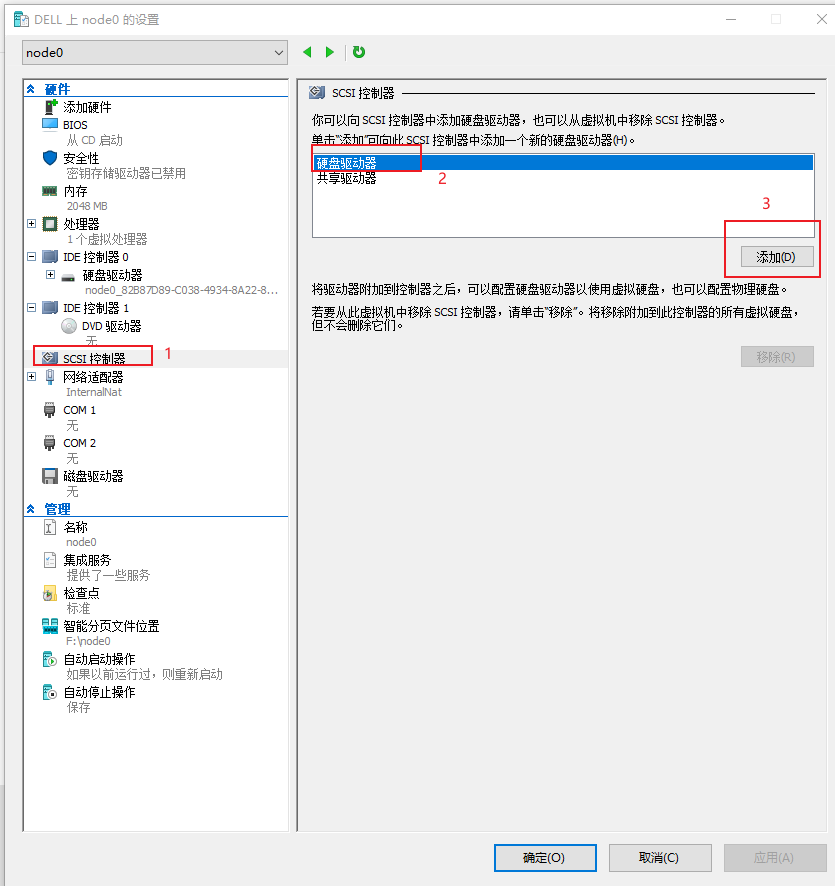
mkfs.xfs /dev/sdb4

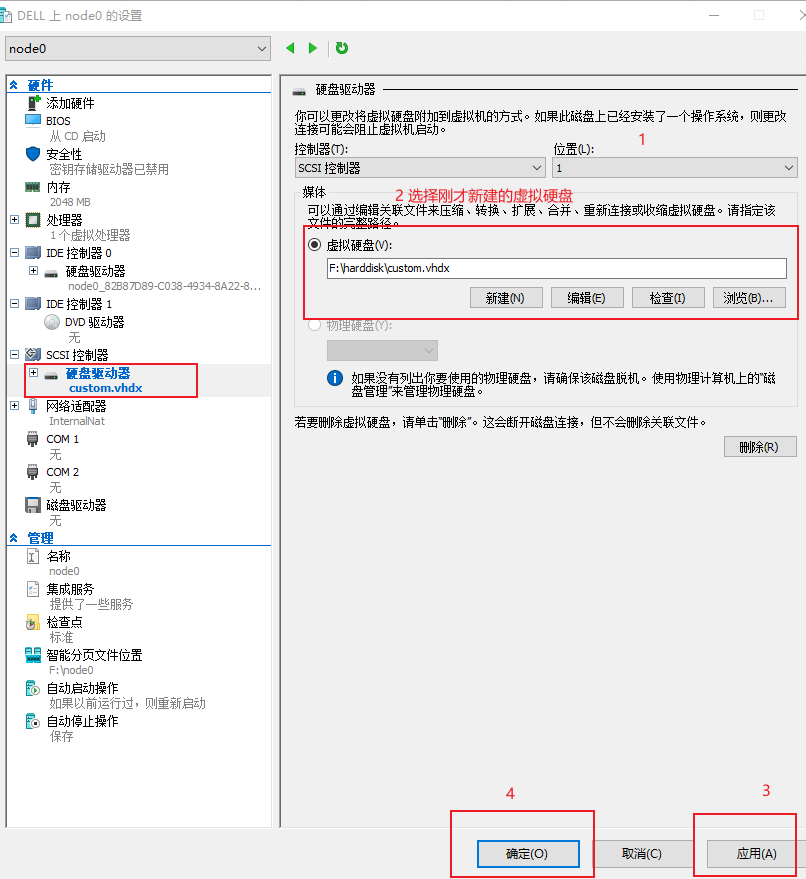
## 6.磁盘分区与挂载

(1)在虚拟机上新建硬盘，并添加到对应的虚拟机上，选择默认选项，名称不要选择中文。

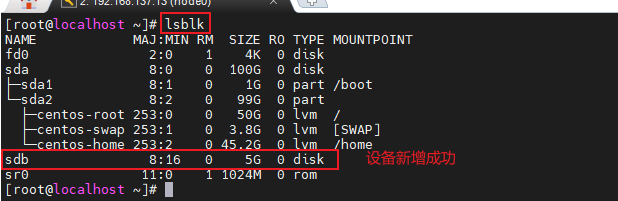






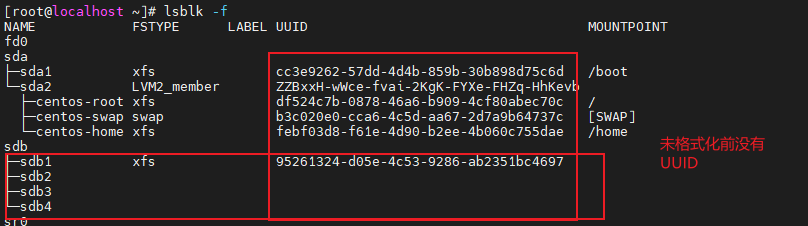


(2)查看设备是否添加成功

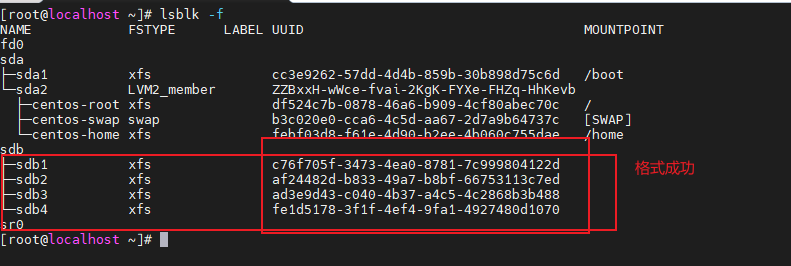


(3)执行分区命令-见fdisk命令

(4)磁盘格式化命令



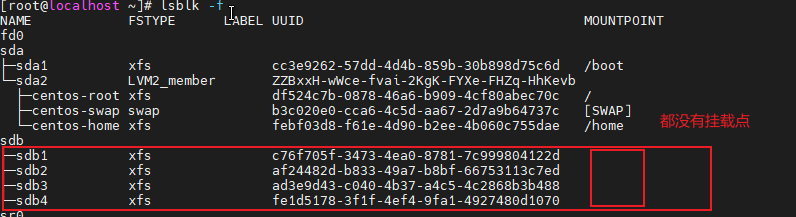
mkfs.xfs /dev/sdb1  
mkfs.xfs /dev/sdb2  
mkfs.xfs /dev/sdb3  
mkfs.xfs /dev/sdb4

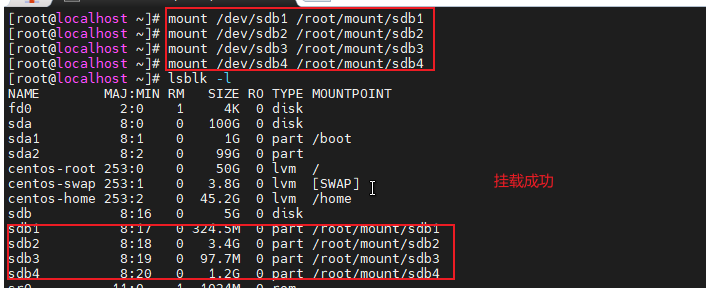


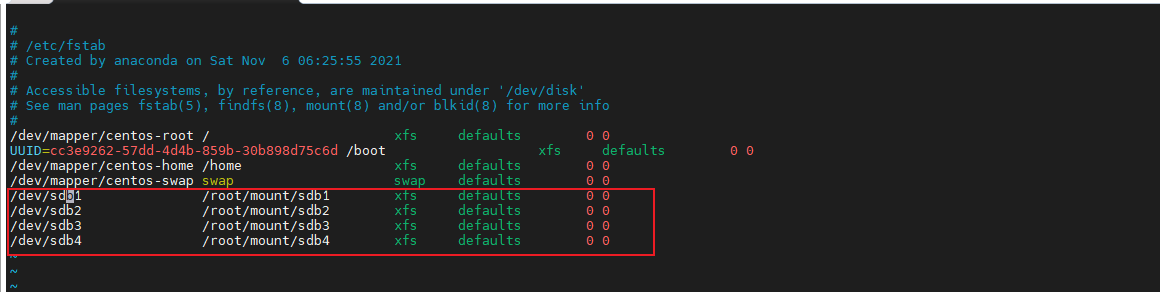
注意：

拓展分区不能执行磁盘格式化命令，拓展分区相当于虚拟区域，没有占用。经常用于建立超过四个分区的场景。

(5)磁盘格式化成功以后，进行挂载命令mount



 (6)手动挂载，重启以后会失效，必须配置开机或者重启自动挂载



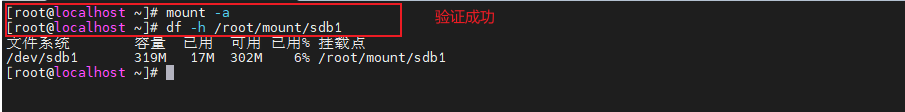
/etc/fstab是开机时的配置文件，如果在修改该文件发生错误，则将导致无法顺利开机成功。

注意：

修改/etc/fstab文件时，一定要验证一下语法是否正确，通过mount –a 挂载所有，然后验证自定义的挂载目录是否成功。

例如：

mount -a  
df /root/mount/sdb1

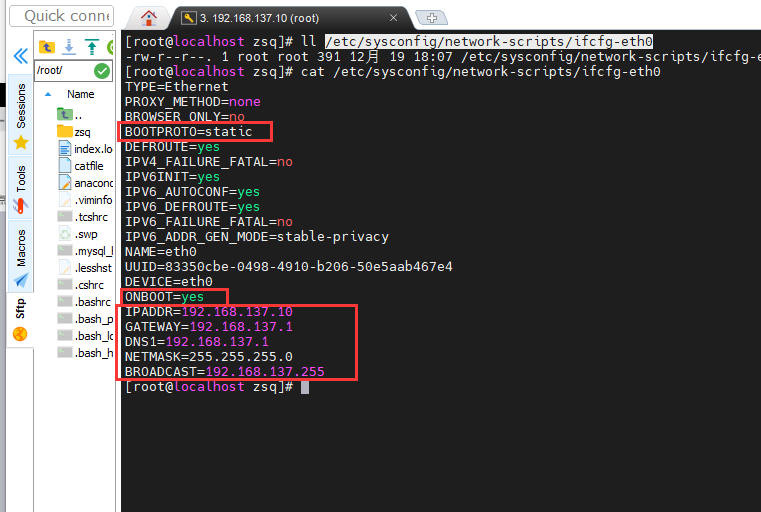


# #网络配置

查看IP的指令：ifconfig

配置固定IP：

/etc/sysconfig/network-scripts/ifcfg-eth0



Linux网配置原理图(含虚拟机)



# #文本处理

## 1.grep

常用参数：-n 列出行号

-v 对过滤的条件取反

-i 忽略大小写

-w 精确匹配，仅完全匹配字词

-c 统计匹配成功以后数据的行数

-x 精确匹配，仅完全匹配一行

(1)利用[ ]来搜寻集合字符

Grep –n ‘t[ae]st’ index.log -->匹配出tast和test；

(2)[^]反选

Grep –n ‘[^g]oo’ index.log -->匹配非g开头且后缀为oo的字符串；

(3)行首和行尾字符^和$

Grep –n ‘^the’ index.log --> 过滤以‘the’开头的行；

Grep –n ‘the$’ index.log --> 过滤以‘the’结尾的行；

(4)grep 使用or、and、not操作

①or操作

ll | grep **"1.txt\|2.txt"**ll | grep -P **"1.txt|2.txt"**

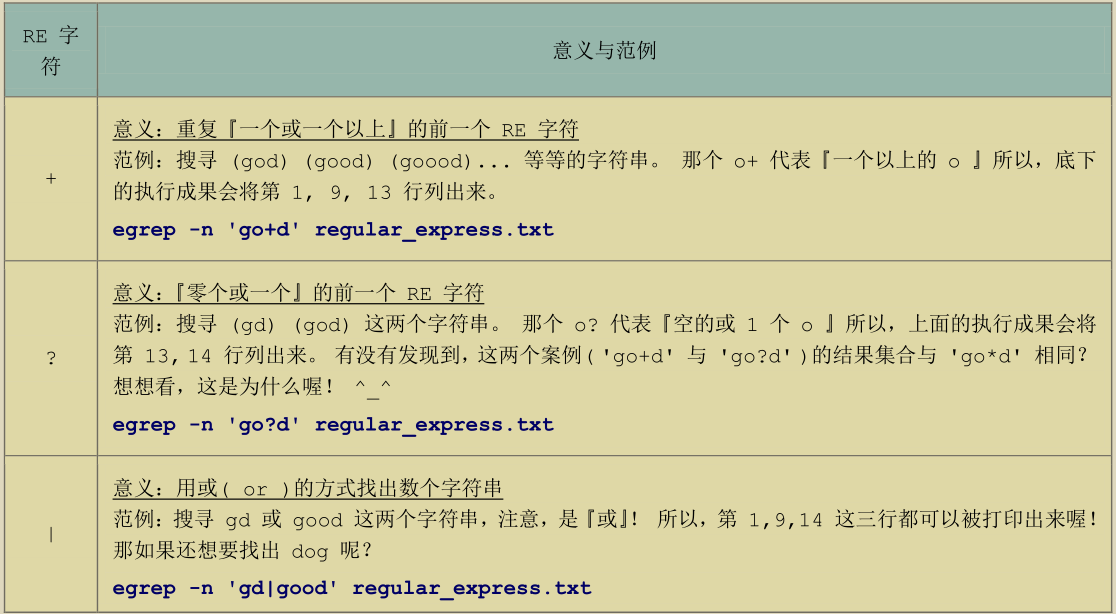
② and操作

ll | grep -E **"pattern1.\*pattern2"**ll | grep pattern1 | grep pattern2

③ not操作

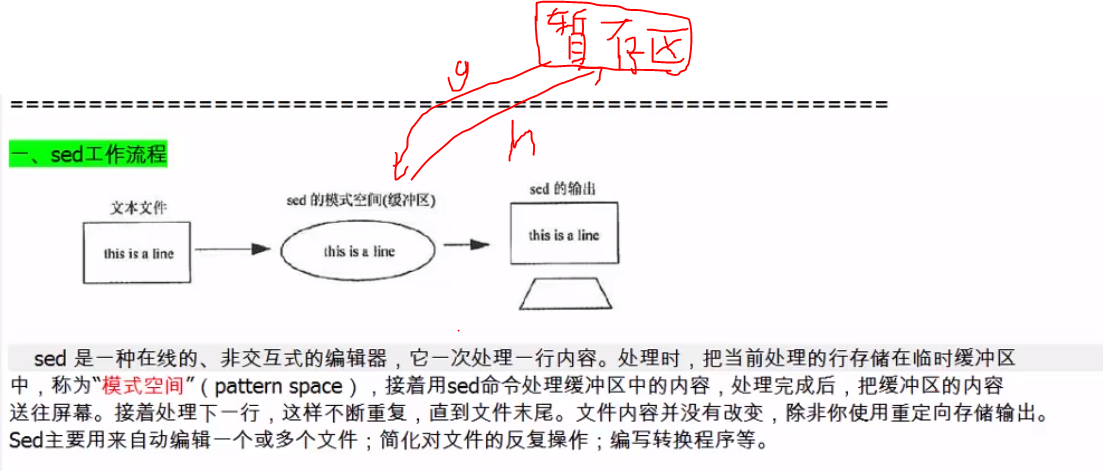
ll | grep -v pattern1





## 2.sed

原理图：



sed 可以用行为单位进行部分数据的搜寻并取代功能。

sed **'s/要被取代的字符串(可以是正则匹配)/新的字符串/g'**

sed **"s/#join/content/g"** sh.tpl **>** actal.sh

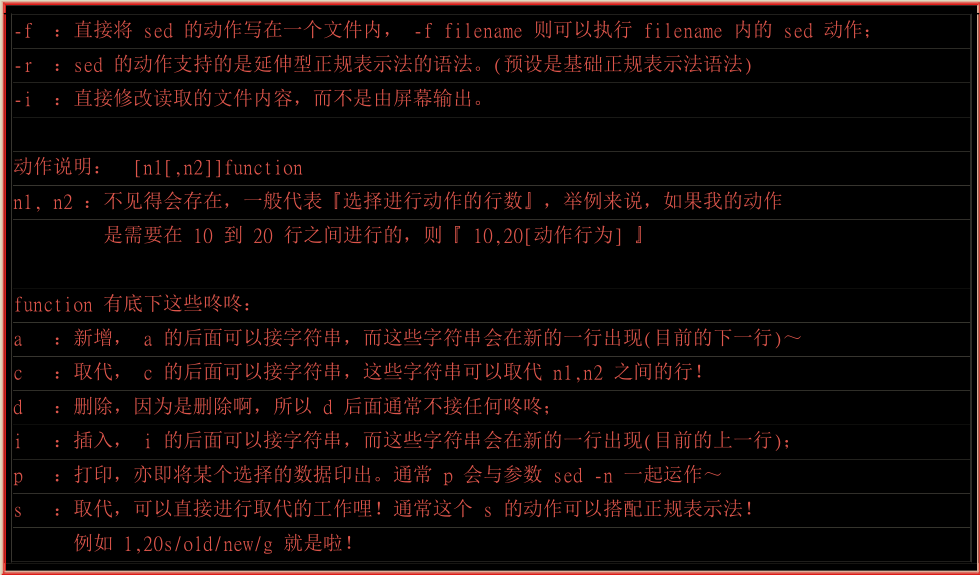
将sh.tpl文件中的“#join”替换为content，但是不会修改原文件，而是将替换的内容重新输入到actal.sh文件。

sed s#**要被取代的字符串(可以是正则匹配)**#**新的字符串**#g

echo **"/a/b/c"** | sed s#/a/b/##g

将/a/b/c 替换为 c

注意：使用s///g替换/符号则会报错，可以采用#分割符，则不会报错



r-支持拓展正则表达式

sed –r ‘1~2d’ passwd 删除奇数行

sed –r ‘0~2’ passwd 删除偶数行

常用参数：

-r、s、d

Sed –r ‘s/root/bin/g’ passwd 查找替换

h 把模式空间里的内容复制到暂存区(覆盖)

H 把模式空间里的内容复制到暂存区(追加)

g 取出暂存区的内容，将其复制到模式空间，覆盖该处原有的内容

G 取出暂存区的内容，将其复制到模式空间，追加在原有内容的后面

x 交换缓冲区和模式空间的内容

暂存和取用命令：

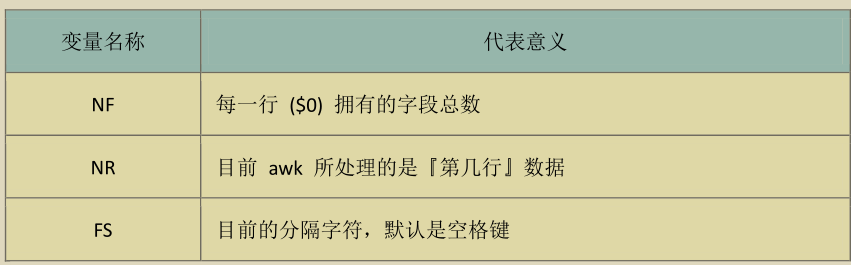
Sed –r ‘1h;$G’ passwd

Sed –r ‘1{h;d};$G’ passwd

Sed –r ‘1h;2,$G’ passwd

Sed –r ‘1h;2,3H;$G’ passwd

## 3.awk



Awk [option] ‘commands’ filenames

选项：

-F 定义输入字段分割符，默认是以空格或者tab键分割

命令：

BEGIN{} {} END{}

行处理前 行处理 行处理后

BEGIN{ }通常用于定义一些变量，如BEGIN{FS=”:”}

命令格式：

awk ‘pattern’ filename

awk ‘{action}’ filename

awk ‘pattern {action}’ filename

awk ‘条件类型1{动作1} 条件类型2{动作2} …’ filename

内部变量：

$0：awk变量$0保存当前行记录的值

NR：目前awk所处理的第几行数据

FNR：当前文件的总条数

NF：每一行包含的字段总数，保存记录的字段数($1,$2,…,$100)

FS：输入字段分隔符，默认为空格

OFS：输出字段分隔符

RS：输入记录分隔符

ORS：输出记录分隔符

字段分隔符：FS OFS

记录分隔符：RS ORS

注意：

(1)awk的指令间隔：所有awk的动作，即在{}内的动作，如果需要多个指令辅助时，可利用英文分号(;)间隔，或者直接以【Enter】键来隔开每个指令。

cat pay.txt | awk **'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" } NR>=2{total = $2 + $3 + $4;printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'**

(2)逻辑运算当中，如果是等于的情况，则务必使用两个等号。

(3)格式化输出时，在printf的格式设定当中，务必加上**\n**,才能进行分行。

(4)与bash shell的变量不同，在awk当中，变量可以直接使用，不需要加上$符号。

(5)awk 输出的变量，直接用echo打印，换行失效；需要将变量用双引号包裹，换行才生效

***[***root@localhost ~***]****# ll | grep "^[^总]" | awk '{print $9}'*123  
anaconda-ks.cfg  
lsf  
zsq  
***[***root@localhost ~***]****# file=$(ll | grep "^[^总]" | awk '{print $9}')****[***root@localhost ~***]****# echo $file*123 anaconda-ks.cfg lsf zsq  
***[***root@localhost ~***]****# echo "$file"*123  
anaconda-ks.cfg  
lsf  
zsq

综上：对于awk输出的变量用于for循环不要加双引号，用于显示查询则需要加上双引号。

(6)调用外部变量

常用：*-v ip=$IP 将外部变量传入到awk表达式中*

awk -v ip=$IP **'ip==$1{print $2}'** $HOST\_INFO

正则匹配：

(1)匹配整行

awk **'/^alice/'** /etc/passwds  
awk **'$0~/^alice/'** /etc/passwds

(2)匹配字段：匹配操作符(~、!~)

awk **'$1~/^alice/s /etc/passwds**

关系运算法:

awk **'$1==0'** /etc/passwds  
awk **'$2<7'** /etc/passwds

条件表达式：

awk **'{if($3>"123"){print $3}else{print $4}}'** /etc/passwds

算数运算：

awk –F: **'$3\*10 > 500'** /etc/passwd

逻辑运算：

&& == a&&b

|| == a || b

! == !a

awk –F: **'$1~/root/ && $3 <= 15'** /etc/passwd

循环

awk -F: **'{i=1;while(i<NF){print $i;i++}}'** /etc/hosts

awk **'BEGIN{for(i=1;i<10;i++){print i;}}'**

数组

awk –F: **'{username[++i]=$1}END{print username[1]}'** /etc/passwd

索引遍历：

awk -F: **'{user[++j]=$1} END{for(i in user){print i,user[i]}}'** /etc/passwd

其中i是索引

统计/etc/passwd中各类型shell的数量

awk -F: **'{shell[$NF]++} END{for(i in shell){print i,shell[i]}}'** /etc/passwd

统计tcp状态访问的数量，并且排序取最小的两个。

netstat -ant | awk **'{status[$NF]++}END{for(i in status) print i,status[i]}'** | sort -k2 | head -2

awk打印单双引号

awk **'BEGIN {ORS=","} {print "\"\"" }'** white\_list.txt

输出结果:

"","","","",

awk **'BEGIN {ORS=","} {print "'**\'\'**'" }'** white\_list.txt

输出结果：

'','','','',

awk输出记录去掉最后一个分隔符

white\_list.txt文件的内容：

10  
12  
13  
14

awk **'{q=p;p=$0}NR>1{ORS=","; print q}END{ORS=""; print p}'** white\_list.txt

结果：10,12,13,14

awk查看某个时间段的日志

# 导出02:14到02:16分的日志

awk **'/2022-06-24T02:14/,/2022-06-24T02:1[6-9]/'** app.log **>** app0215.log

# 使用sed也是可以的sed -n **'/2022-06-24T02:14/,/2022-06-24T02:1[6-9]/p'** app.log **>** app0215.log

注：awk与sed实际并不解析时间，它们只是按正则匹配，匹配到第一个正则时，开始输出行，直到遇到第二个正则关闭，如果你的日志中没有能匹配第二个正则的行，将导致一直输出到尾行！  
故一般需要将第二个正则变宽松点，如上面的/2022-06-24T02:1[6-9]/，以避免出现这种情况。

## 4.wc

wc用于计算文件的字节数、单词数、字符数以及行数。

wc [选项] 文件

-c 统计字节数，计算机存储的最小单位

-l 统计行数

-m 统计字符数，计算机使用的文字和符号，不同的编码里面，一个字符和字节的对应关系是不同的

-w 统计单词数。一个单词被定义为由空白、跳格或换行字符分隔的字符串。

*# 输出结果：行数 单词数 字节数  
# wc concurrent.sh 等价于 wc -lwc concurrent.sh****[***root@localhost ~***]****# wc concurrent.sh* 26 86 595 concurrent.sh

***[***root@localhost ~***]****# wc -l concurrent.sh*26 concurrent.sh

***[***root@localhost ~***]****# wc -w concurrent.sh*86 concurrent.sh

***[***root@localhost ~***]****# wc -c concurrent.sh*595 concurrent.sh

## 5.sort

sort 用于将文本文件内容加以排序，以行为单位来排序。

-u 在输出的行中去除重复行

-r 降序，默认升序

-n 指定排序规则：要以数值来排序

-t<分隔字符> 设定行中的间隔符

-k 根据-t分隔的列来指定排序的列

-b  忽略每行前面开始出的空格字符

-o<输出文件> 将排序的结果输出到指定的文件

注意：

①-t 和 –k 一般配对使用

原始文件：fruit.txt

pear:90:2.3  
banans:30:5.5  
orange:20:3.4  
apple:10:2.5  
apple:10:2.5  
apple:10:3.5

----------------------去重------------------------------

sort -u fruit.txt

apple:10:2.5

apple:10:3.5

banans:30:5.5

orange:20:3.4

pear:90:2.3

----------------------指定排序的列-----------------------

sort -t: -k3 -u fruit.txt

pear:90:2.3

apple:10:2.5

orange:20:3.4

apple:10:3.5

banans:30:5.5

------------------排序结果输出到文件-----------------------

sort -t: -k3 -u -r -n -o res.txt fruit.txt

res.txt:

banans:30:5.5

apple:10:3.5

orange:20:3.4

apple:10:2.5

pear:90:2.3

## 6.find

Linux find 命令用来在指定目录下查找文件。

匹配参数：

-name 匹配的文件名，支持模糊匹配

-type 查看文件的类型：d:目录；f:一般文件

-mtime 查找在过去多少天曾被更改过的文件或目录

-depth 查找的深度

操作参数：

-delete 删除指定的文件，使用前建议测试，避免删除其他文件，或者指定目录删除

-print 打印出匹配到的文件

-exec 对匹配的文件执行该参数所给出的shell命令。格式如下：

-exec **<**command**> *{}*** \;

注意：{}与\;之间有空格

(1)查看指定目录下的文件：

find /root/zsq/ -print  
***[***root@localhost zsq***]****# find /root/zsq/* -print/root/zsq/  
/root/zsq/anaconda-ks.cfg  
/root/zsq/concurrent.sh  
/root/zsq/connect.log  
/root/zsq/fruit.txt  
/root/zsq/res.txt

①查找的结果还包含自己，故进行文件删除操作时一定要小心

(2)查看指定目录下指定名称的文件：

find /root/zsq/ -name **"\*.txt"  
*[***root@localhost zsq***]****# find /root/zsq/ -name "\*.txt"*/root/zsq/fruit.txt  
/root/zsq/res.txt

注意：①模糊匹配需要加双引号

(3)查看指定目录下指定名称指定类型的文件：

find /root/zsq/ -name zsq -type d -print  
***[***root@localhost zsq***]****# find /root/zsq/ -name zsq -type d -print*/root/zsq/

(4)查看过去180天的内指定目录下指定名称指定类型的文件：

find /root/zsq/ -name **"\*.txt"** -type f -mtime +180 -print

(5)删除过去180天的内指定目录下指定名称指定类型的文件：

三种方法：

①-delete

find /root/zsq/ -name **"\*.txt"** -type f -mtime +180 -delete

②-exec

find /root/zsq/ -name **"\*.txt"** -type f -mtime +180 -exec rm -rf ***{}*** \;

③xargs

find /root/zsq/ -name **"\*.txt"** -type f -mtime +180 | xargs rm -rf

# #Bash

## 1.shell变量

Echo: 取出变量中的内容; echo $PATH。

变量的设定规则：

(1)变量与变量内容以一个等号来连接；

(2)等号两边不能直接接空格符；

(3)变量内容若有空格可使用双引号或单引号将变量内容包裹起来，但是：

a.双引号内的特殊字符如$等，可以保持原有的特性；

b.单引号内的特殊字符则仅为一般字符。

(4)用\可以将特殊符号($、空格)变成一般字符。

(5)在一系列指令的执行中，还需要由其他额外的指令提供信息时，可以用反单引号。

`命令`或$(命令)：将命令执行的结果复制给变量。

如：version=$(uname -r) 或 version=`uname -r`

(6)若为变量扩增内容时，则可用”$变量名称”或”${变量}累加类容”；如

PATH=”$PATH”:/home/bin或PATH=${PATH}:/home/bin

(7) export 将自定义变量转换为环境变量，该变量可以在其他字程序中执行。

(8) unset取消对变量的设置，如：unset myname。

(9)env和export命令可以查看系统有哪些环境变量，set可以观察所有的变量包括环境变量和自定义变量。

(10)获取字符串的长度，在${}中使用”#”获取长度。

name=”test”

echo ${#name}

(11)提取子串

a.1:4从第二个开始，往后截取4个字符

b.::从第一个字符开始，往后截取4个字符

name=**"12345"**echo **$*{***name:1:4***}***echo **$*{***name::4***}***

2345

1234

## 2.预定义变量

$$ 目前线程的代号

$? 关于上个执行指令的回传值,如果成功的执行该指令，则会回传一个0值。

$! 后台运行的最后一个进程的进程号

子程序仅会继承父程序的环境变量(使用export导出的变量)，而不会继承父程序的自定义变量(普通变量)。

## 3.键盘变量的读取与其他类型的定义

(1)read 读取来自键盘的收入的变量

-p 后面接提示字符

-t 后面接等待的秒数

read -p 'please enter you name:' -t 30 named

ps:提示使用者30秒内输入自己的名字，将该输入的字符串作为名为named变量的内容。

(2)declare/typeset 定义变量的类型

declare [选项] variable

选项：

-a:定义为数组类型

-i:定义为整数类型

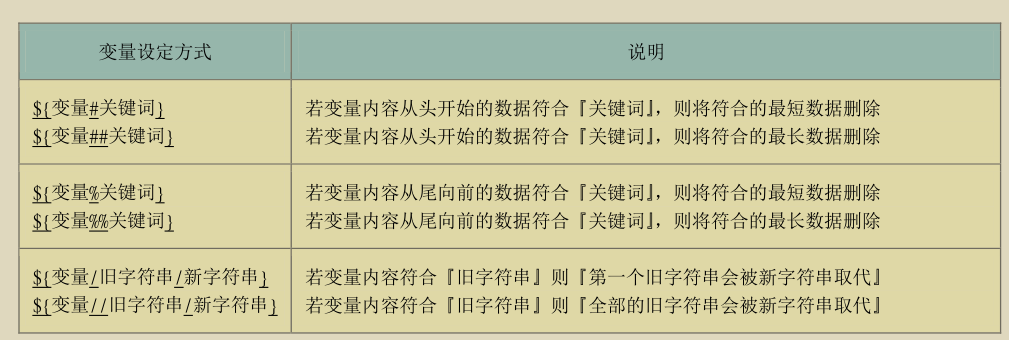
-x:将后面的变量变成环境变量

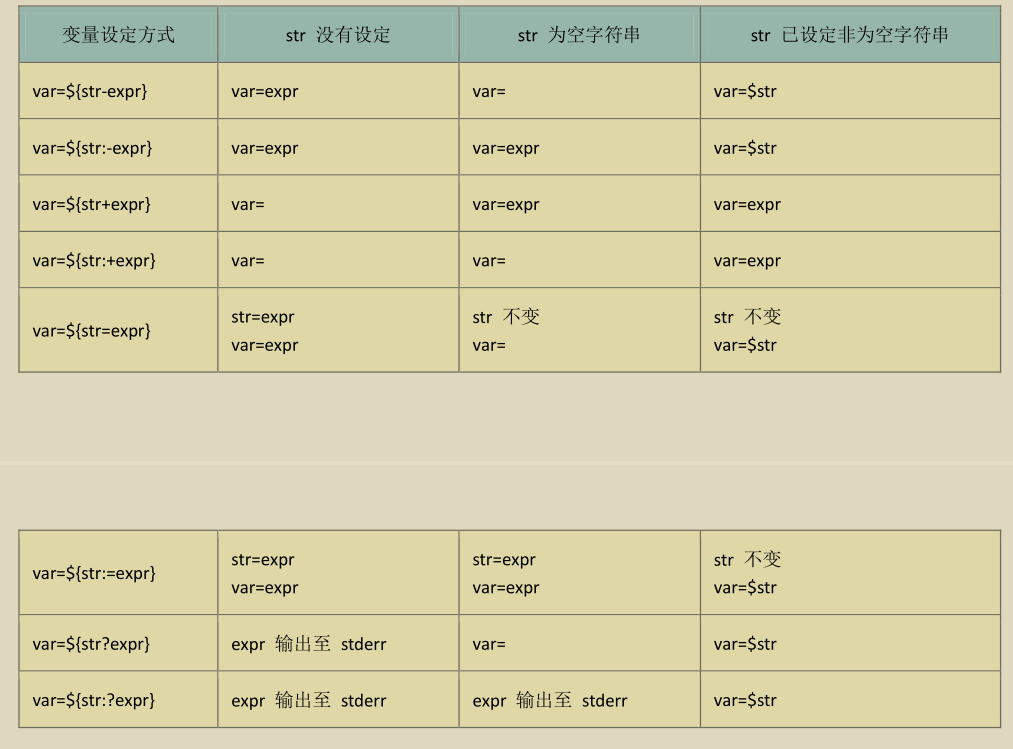
-r:将变量设置为只读，且内容也不能被修改

注意：

变量默认类型为字符串。

## 4.变量内容的替换





5.Bash Shell的操作环境

指令运行的顺序：

(1)以相对/绝对路径执行命令

(2)由alias找到该指令来执行

(3)有bash内建指令来执行

(4)透过$PATH这个变量的顺序搜寻找的第一个指令来执行

ps:type -a ls 查询指令的搜寻顺序

bash登录的欢迎界面：

/etc/issue,/etc/motd

## 5.bash的环境配置文件

login与non-login shell

login shell：需要输入用户名和密码，才能获取bash。

login shell只会读取两个配置文件：

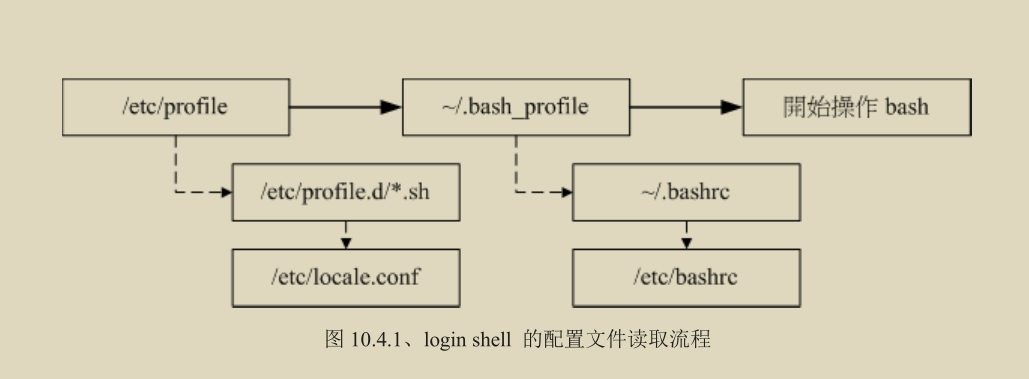
(1)/etc/profile：系统整体的设定，最好不要修改。

(2)~/.bash\_profile或~/.bash\_login或~/.profile:输入个人设置。

non-login shell：不需要密码重新输入登录。

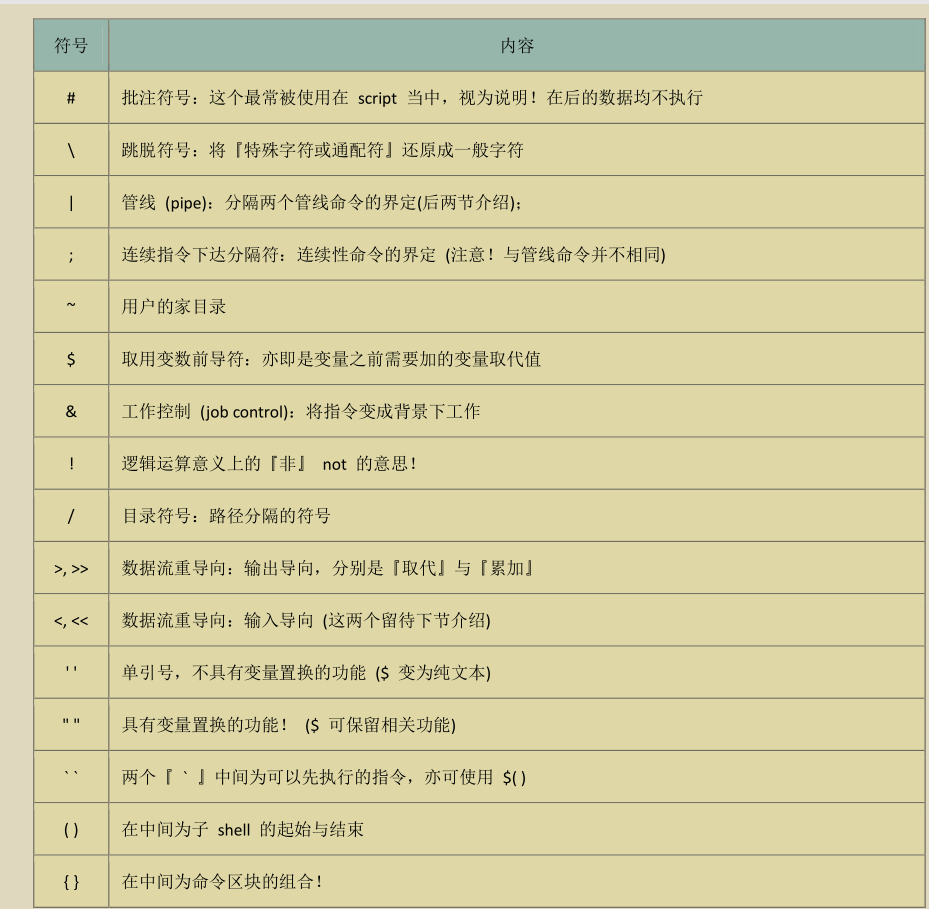
non-login shell只会读取~/.bashrc配置文件

login shell的配置文件的读取流程



Source 和 . 读取配置文件，让其在父进程中生效。

## 6.通配符与特殊符号



## 7.数据流重导向

标准输入流与输出流

(1)标准输入：代码为0， 使用<或者<<;

(2)标准输出：代码为1， 使用>或者>>;

(3)标准错误输出：代码为2，使用2>或者2>>;

(4)/dev/null垃圾桶黑洞装置：可以吃掉任何导向这个装置的信息。

Ps：ls /etc 2> /dev/null

ls /etc > /index.txt

1> 表示以覆盖的方法将“正确的数据”输出到指定的文件或者装置上。

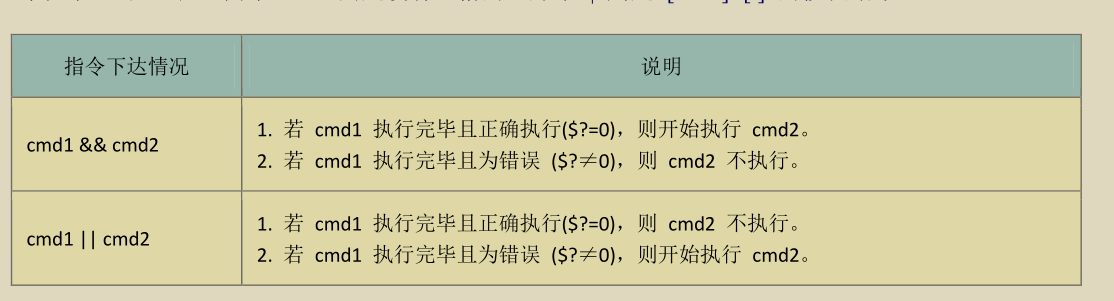
1>> 表示以追加的方法将“正确的数据”输出到指定的文件或者装置上。

2> 表示以覆盖的方法将“错误的数据”输出到指定的文件或者装置上。

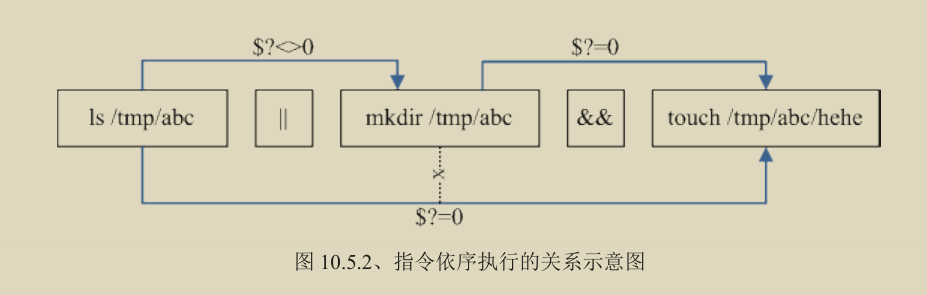
2>> 表示以追加的方法将“错误的数据”输出到指定的文件或者装置上。

2、命令执行的判断依据

$?与&& 或 ||

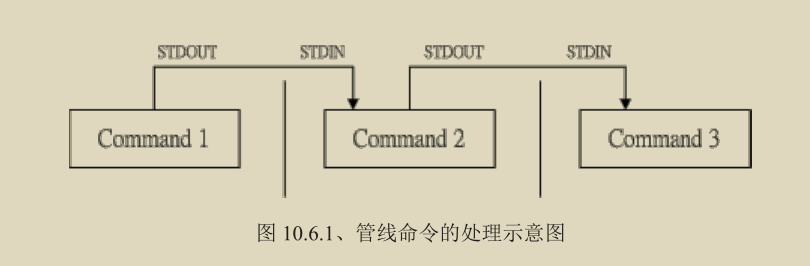


ls /tmp/abc || mkdir /tmp/abc && touch /tmp/abc/hehe 的执行流程图如下：



3、管道(|)

管道处理的示意图：



(1)管道命令仅会处理标准输出，对于标准错误输出会予以忽略。

(2)管道命令必须要能够接受来自前一个指令的数据成为标准输入继续处理才行。

注意：less、more、head、tail、grep都是支持管道命令，ls、cp、mv不支持。

# #shell编程

目的：

(1)自动化管理的重要依据

(2)追踪和管理系统的重要工作

(3)简单的入侵检测功能

(4)连续指令单一化

(5)简易的数据处理

(6)跨平台支持与学习历程较短

## 1.shell script执行的方式

script执行方式的差异

(1)直接方式执行(sh script，./script)：当子程序完成后，在子程序内的各项变量或者动作将会结束而不会传回到父程序中。

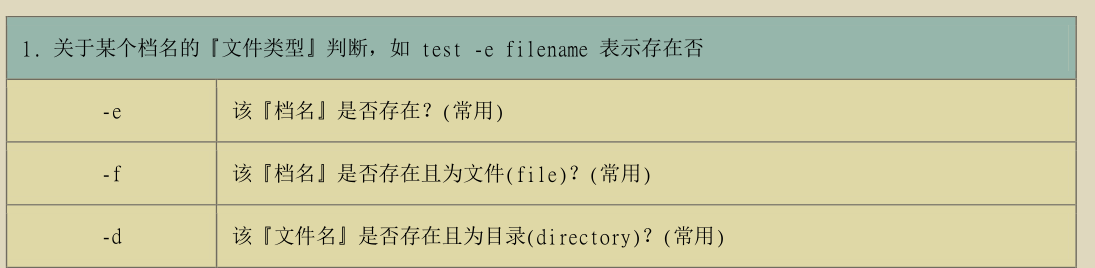
(2)source或小数点(.)方式执行：在父程序中执行，变量和动作直接在父程序中。

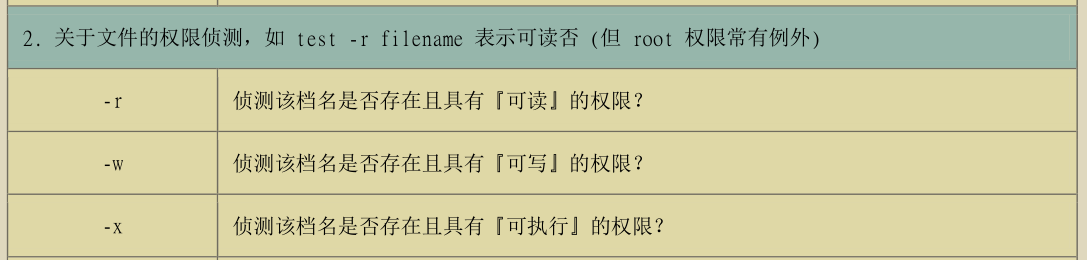
## 2.条件判断

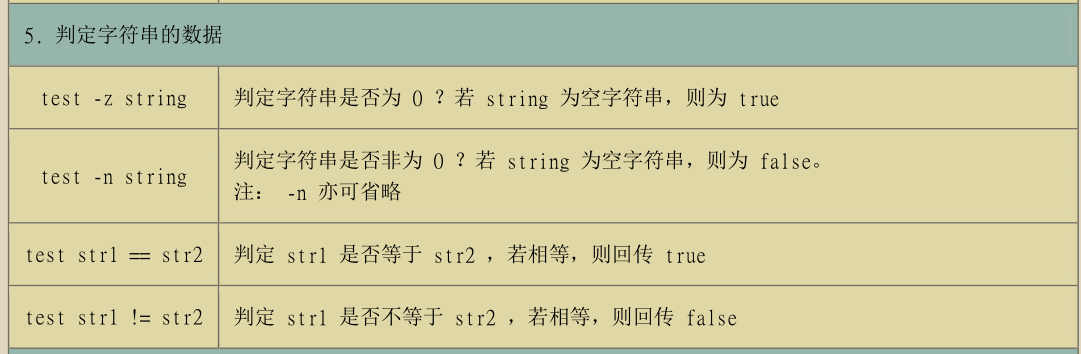
### (1)test指令

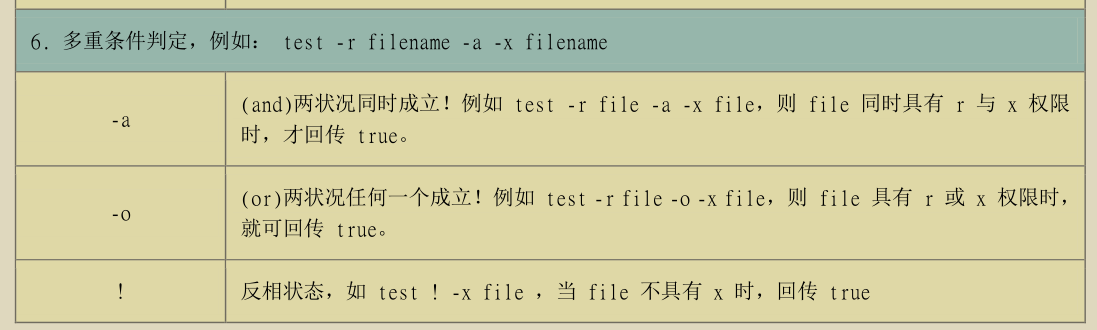
如：test –e /root/zsq/index.txt && echo “Exit” || echo “Not exit” 判断目录下是否存在该文件。

常见的test判断的参数和指令：









### (2)判断符[ ]

例如：[ -z “${HOME}” ];echo $? 判断${HOME}这个变量是否为空。

注意：

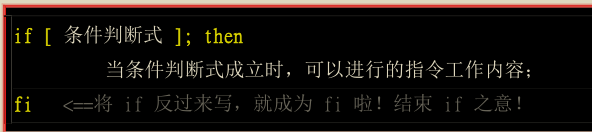
a.在中括号[]内的每个组件都需要有空格键来分割；

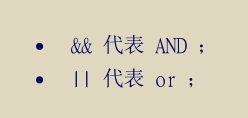
b.在括号中的变量，最好都用双引号包裹；

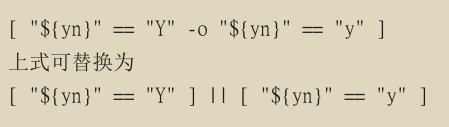
c.在括号中的常量，最好都用单引号或者双引号包裹。

### (3)if 判断

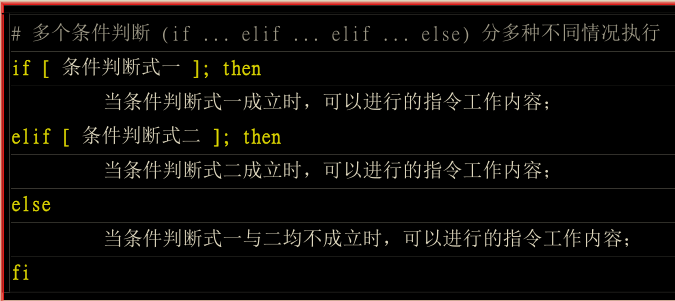
a.单层判断句







b.多重复杂判断式



c.常用的判断条件：

=字符串的比较

-gt大于

-lt小于

-ge大于等于

-le小于等于

-eq等于

-ne 不等于

$((计算式))用来进行数值运算

d.表达式计算的两种方式：

①$(())：加减乘除，不添加空格

$(((2+3)\*4))

②$[]: 加减乘除，不添加空格

$[(2+3)\*4]

e.[[]] 与 []的区别

①&&、|| 、<、>操作符能够正常存在于[[]]条件判断结构中。

②&&、|| 、<、>操作符出现在[ ]中则会报错。

③可以直接使用if [[ $a != 1 && $a != 2 ]]。

④若用单括号, 则为if [ $a -ne 1] && [ $a != 2 ]或者if [ $a -ne 1 -a $a != 2 ]。

⑤[[ ]]中增加模式匹配特效

### (4)三元表达式

原理:

&&的优先级比||高，如果前面的&&执行成功，后面的就不会执行；相反，后面的||就会执行。

例如：

a=**$*([* "**$b**" ==** 5 ***]* &&** echo **"**$c**" ||** echo **"**$d**"*)***

count=**$*([*** -f result ***]* &&** echo **$*(***awk **'{print $1}'** result | grep -cx **"**$id**"*)* ||** echo 0***)***

status=**$*([* "**$a**" ==** 1 ***]* &&** echo false **||** echo true***)***

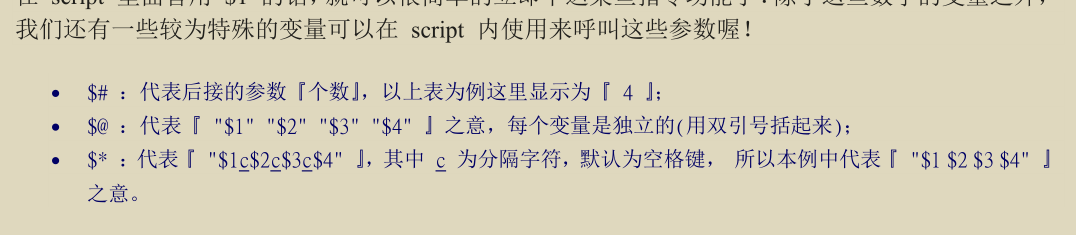
## 3.shell script默认参数

shell script可以执行一个带参数的脚本，该参数就是shell script的默认参数。shell script的默认参数变量($0,$1,$2,…,$n)。

示意图：

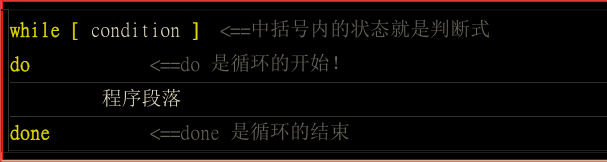


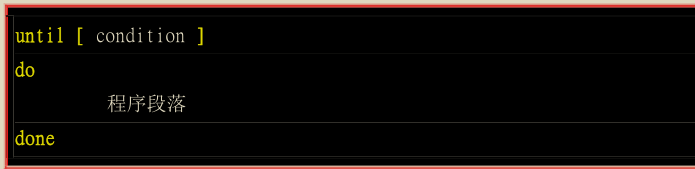
特殊的参数变量：



## 4.循环

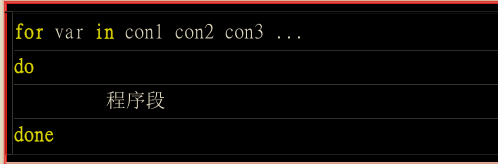
### (1)不定循环





**while** true; ***do*** read confirm  
 ***if [* "**$confirm**" == 'y' *]***; **then** echo **"准备升级数据库"** break  
 **elif *[* "**$confirm**" == 'n' *]***; **then** echo **"取消升级数据库，程序退出"** exit  
 **else** echo **"请输入y 或 n 确认是否升级数据库"  
 *fi  
done***

### (2)固定循环



a. bash只支持一维数组，不支持多维数组。

b.定义数据：使用小括号做边界、使用空格分离，array\_name=(li wang xiang zhang)。

c.获取数组元素：

echo ${arrayname[@]}

# 输出"li zhang xiang zhang" 输出数组所有元素

d.获取元素的个数：${#arrayname[@] } 或者${#arrayname[\*]}。

e.获取单个元素的长度:${#arrayname[0]}。

f.遍历方式：

①标准for循环

**for *((*** i = 0; i **< $*{***#array\_name[@]***}***; i++ ***))***; ***do*** echo **$*{***array\_name[i]***}  
done***

②for in循环

**for** i in **$*{***array\_name[@]***}*** ; ***do*** echo $i  
***done*for** i in **$*{***array\_name[\*]***}*** ; ***do*** echo $i  
***done***

## 5. debug

shell script追踪与debug

sh [-nvx] script.sh

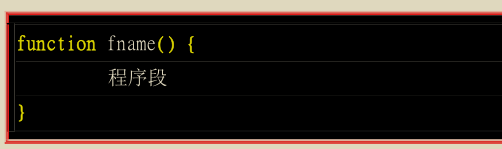
选项与参数：

-n 不执行script，仅查询语法的问题。

-v 在执行script前，先将script的内容打印输出到屏幕上。

-x 将使用到的script内容显示到屏幕上，常用。

## 6.函数(function)



注意：

(1)由于shell script的执行方式是由上而下，由左至右的，因此function的设定一定要在程序的最前面。

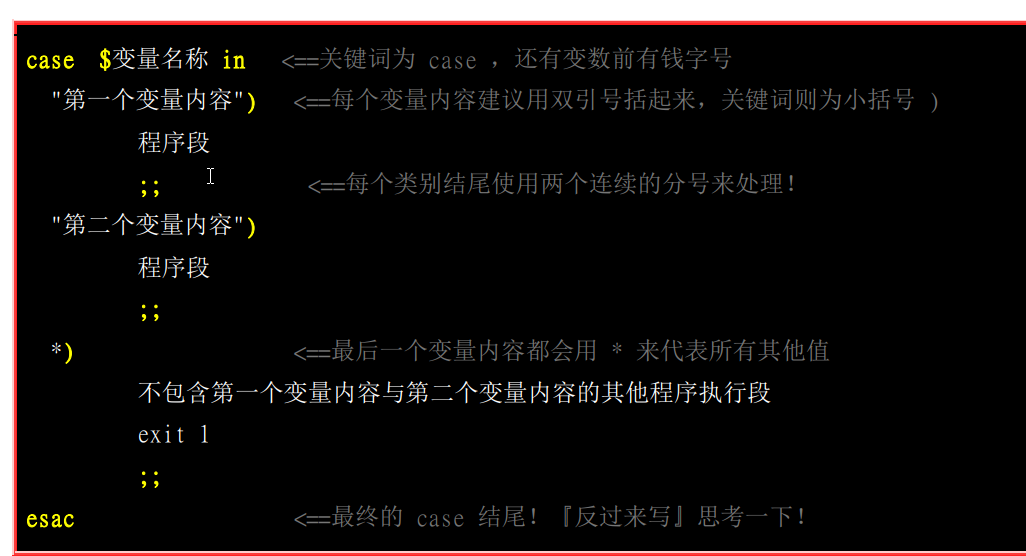
(2)function也有内置的变量，与shell script很相似。函数名称代表$0,后续的变量也是以$1，$2等来取代的。

(3)函数的返回值：

a.函数默认是将标准输出传递出来，不是返回值

b.使用$?接收上一程序的返回值状态，即return的返回值，而且$?的值必须为整数。

## 7.Case语句



read -p **"输入："** x  
***case*** $x **in  
 "1"**)  
 echo 1  
 **;;  
 "2"**)  
 echo 2  
 **;;** \*)  
 echo **"其他"** exit  
***esac***

一般来说，使用***case*** $x **in** 语法，$x 变量大致有两种获取的方式：

(1) 直接下达：如script.sh variable 的方式直接给予$x这个变量。

(2) 交互式：透过read这个指令让用户输入变量的内容。

## 8.set –e or set +e

set-e或者set+e；set -e表示从当前位置开始，如果出现任何错误都将触发exit。相反，set +e表示不管出现任何错误继续执行脚本。

如果脚本是有状态的（每个后续步骤都依赖前一个步骤），那么请使用set -e，在脚本出现错误时立即退出脚本。如果要求所有命令都要执行完（很少会这样），那么就使用set +e。

## 9.set –x or set +x

常用于脚本调试，set –x 是开启调试，set +x 是关闭调试。

①执行set -x后，对整个脚本有效

②针对部分代码片段调试，则选择set –x 和 set +x 配套使用。如:

FILENAMES=**$*(***ls -f***)***echo $FILENAMES  
set -x  
tmp=123  
echo $tmp  
set +x  
basename $0

此段代码中，只有set –x 与 set +x 之间的代码才会输出调试信息。

## 10.长句换行

在shell中为避免一个语句过长，可以使用”\”进行换行。使用”\”换行，在脚本执行过程中还是当做一行一个语句执行，不同于enter直接换行。

**注意：\ 前添加一个空格 。\ 后无空格直接换行。**

案例：

*# -e 代表执行sql语句*/mysql/mysql/bin/mysql \  
-h test\_host -P 3306 \  
-u test\_user -ptest\_password \  
-e **"use test\_database; source data\_faile; "**

# #进程管理

程序：通常为二进制程序，放置在储存媒体中，为实体文件的型态存在。

进程：程序被触发后，执行者的权限与属性、程序的代码与所需数据等都会被加载到内存中，操作系统并给予这个内存单元一个标识符(PID)，可以说，进程就是一个正在运作中的程序。

服务：常驻在内存中的进程通常是负责一些系统所提供的功能以服务用户各项任务，这些常驻程序被称为服务(daemon)。

## 1.工作管理

(1)当我们登入bash后，在单一终端机接口下同时进行多个工作的行为管理。

注意：

前景：可以控制与下达指令的这个环境称为前景的工作。

背景：可以自行运行的工作，无法使用[Ctrl] + C 终止它，可以使用bg/fg呼叫该工作。

背景中执行的进程不能等待terminal/shell 的输入。

(2)&：将指令丢到背景中去执行

(3)nohup：在脱机或者注销的情况下，让工作继续执行。

用法：



& 常与 nohup 搭配。

## 2.进程管理

ps –aux | grep xxx

ps –aux | more

ps –ef 可以查看父进程

Kill 进程号

top、uptime

# #任务调度

1、使用者的设定

为了安全性，crontab可以限制使用者的账号，使用的限制数据有：

(1)/etc/cron.allow:将可以使用crontab的账号写入其中，不在这个文件内的使用者则不可以使用crontab；

(2)/etc/cron.deny：将不可以使用crontab的账号写入其中，若不在这个文件内的使用者则可以使用crontab；

(3)当用户使用crontab建立定时任务时，该项工作会被记录到/var/spool/cron里，而且是以账号来做为区分。如：root使用crontab后，它的定时任务会被记录到/var/spool/cron/root。

(4)cron执行的每一项定时命令，都会被记录到/var/log/cron这个文档中，便于查找问题。

crontab进行任务调度的设置

crontab [选项]

-r 终止所有任务调度，若要移除一项，用-e去编辑。

-e 编辑任务调度

-l 列出当前所有任务调度

Systemctl crond restart 重启任务调度

执行步骤：

(1)cron –e

(2)\*/1 \* \* \* \* ls –l /ect >> /temp/to.txt

(3)当保存退出后程序开始执行

(4)在每一分钟执行ls –l /ect >> /temp/to.txt

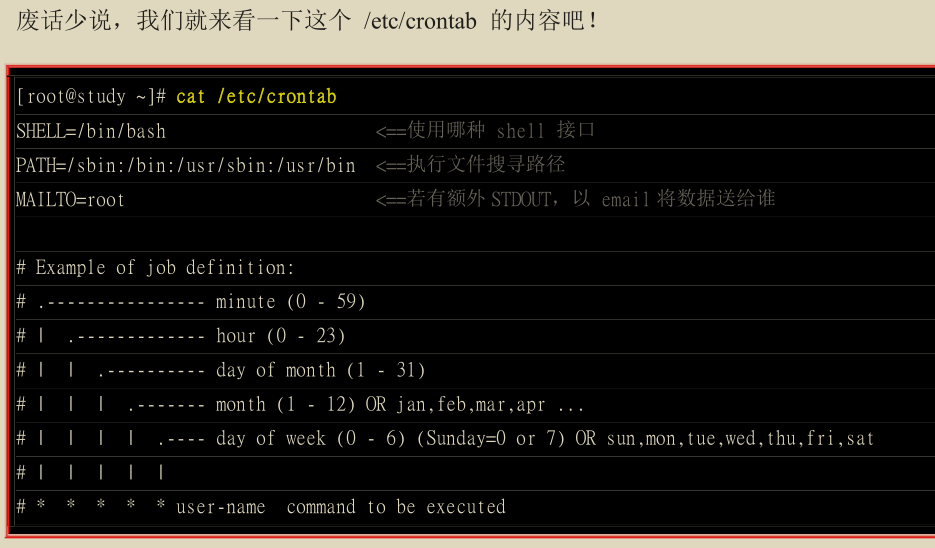
2、系统配置文件：/etc/cron.d/\*、/etc/crontab

Crontab –e 是针对使用者的cron来设计的；如果是系统的定时任务则只要编辑/etc/crontab文件。

注意：

(1)crontab –e 中的crontab其实是/usr/bin/crontab这个文件，但是/etc/crontab是一个纯文本。

(2)系统默认的定时任务都是以root的身份来执行的。



Crond服务读取配置文件的位置

(1)/etc/crontab

(2)/etc/cron.d/\*

(3)/var/spool/cron/\*

/etc/crontab和/etc/cron.d/是放系统定时任务的配置文件，/var/spool/cron/\*则是跟用户自己的定时任务有关。

如果想开发新的软件，并且该软件拥有自己的crontab定时指令，则可以将配置文件放到/etc/cron.d/\*目录下，在此目录下的文件是crontab的配置文件脚本。

总结：

(1)个人使用定时任务使用crontab –e指令

(2)系统维护管理使用vim /etc/crontab,便于管理

(3)自己开发软件使用vim /etc/cron.d/newfile

注意：

(1)定时任务调用docker命令时，执行失败，需要去掉-it参数。

# #系统服务(daemon)

当服务的名称建立后，被挂上Linux使用时，通常会在服务的名称之后加上一个d,如firewall服务，它的程序文件会被取为firewalld，这个d就代表daemon。

## 1.daemon与systemd

1、deamon服务的启动

所有的服务自动脚本都放在/etc/init.d下，启动、关闭、重新启动、观察状态的命令如下：

(1)启动: /etc/init.d/daemon start

(2)关闭: /etc/init.d/daemon stop

(3)重新启动: /etc/init.d/daemon restart

(4)状态观察: /etc/init.d/daemon status

如：systemctl firewalld status、systemctl sshd statud.

2、systemd的配置文件

(1) /usr/lib/systemd//system:每个服务最主要的启动脚本设置，有点像/etc/init.d下的文件。

(2) /run/systemd/system/:系统执行过程中所产生的服务脚本。

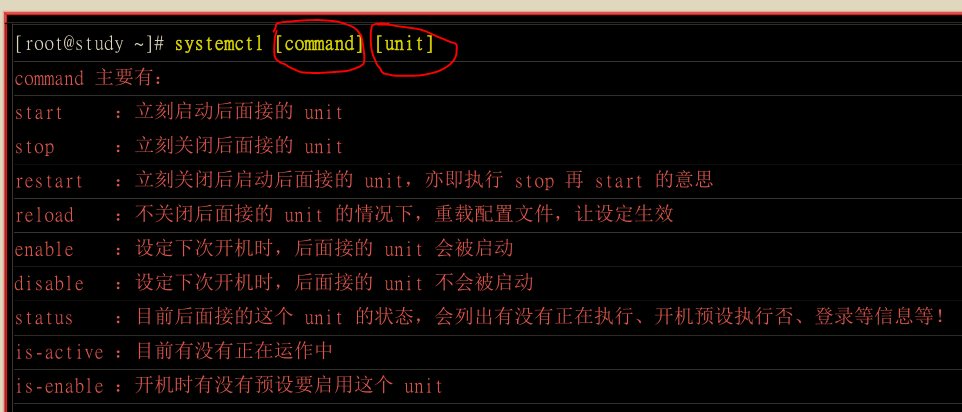
(3/etc/systemd/system/:管理员依据系统的需求所建立的执行脚本。

注意：

系统到底开机会不会执行某些服务其实是看/etc/systemd/system/下的设置，该目录下是一系列的软链接。而实际执行的systemd的启动脚本配置文件都放在/usr/lib/systemd//system下。

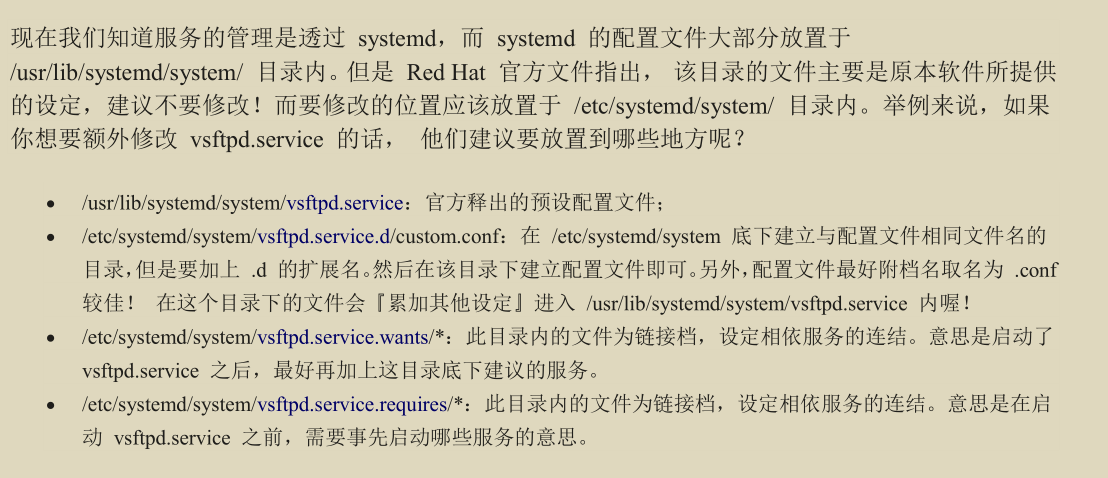
3、systemctl管理服务

Systemd启动服务的机制，主要通过systemctl命令来执行的。



systemctl – 查看系统上所有的服务

systemctl list-dependencies – 查看服务之间的依赖



# #日志管理

针对日志处理所需的服务和程序包括：

System-journald.service：最主要的日志者，由systemd提供。

Rsyslog.service：主要记录系统与网络等服务的日志。

Logrotate：主要在进行日志文件的轮替功能。

# #包管理

## 1.yum

常用命令：

yum list | grep java

yum install java

yum upate

yum clean

## 2.rpm

①RPM包验证

主要用来判断软件的相关文件是否被修改，用于定位问题。

rpm -Va  
rpm -V 已安装的软件名称  
rpm -Vp 某个RPM文件的档名  
rpm -Vf 系统上面的某个文件

-v 后面加的是软件名称，若该软件所包含的文件被改动过，才会列出来

-Va 列出目前系统上所有可能被改动过的文件

-Vp 后面加的是文件名，列出该软件可能被改动过的文件

-Vf 列出某个文件是否被改动过

# #shell脚本校验与格式化

## 1.脚本校验-shellcheck

shellcheck shell脚本规范校验命令使用：

参数：  
-f 规范校验结果显示的格式

checkstyle: xml格式  
diff: 差异对比的格式  
gcc: 精简格式  
json: json格式  
json1: json格式  
quiet: 静默格式  
tty:默认  
-P <dir> 指定源文件的目录  
eg:

shellcheck –f diff ping.sh

shellcheck –P /root/local/ ping.sh

shellcheck –f json –P /root/local/ ping.sh

## 2.脚本格式化-shfmt

shfmt shell 脚本格式命令使用：  
参数：  
-w 将格式化后的脚本写回文件  
-i unit 缩进的长度  
-ci case语句将会被缩进

-sr 重定向运算符后跟空格  
eg:  
 shfmt sh.sh  
 shfmt -w sh.sh  
 shfmt -w -i 4 sh.sh  
 shfmt -w -i 4 -ci sh.sh

# #高亮提示信息与打印日志

## 1.echo

shell脚本中echo显示内容带颜色显示,echo显示带颜色，需要使用参数-e。常用于打印日志等级：严重、错误

格式如下：

echo -e **"\033[字背景颜色;文字颜色m 字符串 \033[0m"**

例如：

echo -e **"\033[41;36m 高亮 \033[0m"**

其中41的位置代表背景色，36的位置代表文字的颜色

注：

1.背景颜色和文字颜色之间是英文的分号

2.文字颜色后面有个m

3.字符串前后可以没有空格

字颜色：

echo -e **"\033[30m 黑色字 \033[0m"**echo -e **"\033[31m 红色字 \033[0m"**echo -e **"\033[32m 绿色字 \033[0m"**echo -e **"\033[33m 黄色字 \033[0m"**echo -e **"\033[34m 蓝色字 \033[0m"**echo -e **"\033[35m 紫色字 \033[0m"**echo -e **"\033[36m 天蓝字 \033[0m"**

最后面控制选项说明:

\33[0m 关闭所有属性

　　\33[1m 设置高亮度

　　\33[4m 下划线

　　\33[5m 闪烁

　　\33[7m 反显

　　\33[8m 消隐

　　\33[30m — \33[37m 设置前景色

　　\33[40m — \33[47m 设置背景色

　　\33[nA 光标上移n行

　　\33[nB 光标下移n行

　　\33[nC 光标右移n行

　　\33[nD 光标左移n行

　　\33[y;xH设置光标位置

　　\33[2J 清屏

　　\33[K 清除从光标到行尾的内容

　　\33[s 保存光标位置

　　\33[u 恢复光标位置

　　\33[?25l 隐藏光标

\33[?25h 显示光标

## 2.printf

Printf使用引用文本或空格分隔的参数，此外还可以在printf中使用格式化字符串，控制字符串的宽度、左右对齐方式等。默认的printf不会像echo自动添加换行符，需要手动添加\n。

命令语法：printf format-string ***[***arguments...***]***

①format-string: 为格式控制字符串

②arguments: 为参数列表。

实例：

①printf **"Hello, Shell\n"**②printf **"%-10s %-8s %-4s\n"** 姓名 性别 体重kg  
③printf **"%-10s %-8s %-4.2f\n"** 郭靖 男 66.1234  
④printf **"\033[32m%-10s %-8s %-4.2f\n\033[0m"** 杨过 男 48.6543

输出结果：



①%s、%c、%d、%f 都是格式替代符，％s输出一个字符串，％d整型输出，％c输出一个字符，％f输出实数，以小数形式输出。

②%-10s指一个宽度为10个字符（-表示左对齐，没有则表示右对齐），任何字符都会被显示在10个字符宽的字符内，如果不足则自动以空格填充，超过也会将内容全部显示出来。

③%-4.2f 指格式化为小数，其中.2 指保留2位小数。

④**\033[32m 033[0m** 表示输出的文本的颜色，用于高亮显示或者提示。

## 3.tput

tput命令将通过terminfo数据库对终端会话进行初始化和操作。通过使用 tput，可以更改几项终端功能，如移动或更改光标、更改文本属性（字体大小、颜色、加粗），以及清除终端屏幕的特定区域。与 UNIX 中的大多数命令一样，tput 命令既可以用在 shell 命令行中也可以用在 shell 脚本中。

(1)参数

①字符串输出参数设置

　　bel 警铃

　　blink 闪烁模式

　　bold 粗体

　　civis 隐藏光标

　　clear 清屏

　　cnorm 不隐藏光标

　　cup 移动光标到屏幕位置(x,y)

　　el 清除到行尾

　　ell 清除到行首

　　smso 启动突出模式

　　rmso 停止突出模式

　　smul 开始下划线模式

　　rmul 结束下划线模式

　　sc 保存当前光标位置

　　rc 恢复光标到最后保存位置

sgr0 正常屏幕

rev 逆转视图

②数字输出参数设置

　　cols 列数目

　　ittab 设置宽度

　　lines 屏幕行数

③布尔输出参数设置

　　chts 光标不可见

　　hs 具有状态行

④景色

setaf ColorNumbe 设置前景色

setab ColorNumber 设置背景色

tput clear # 清屏  
tput sc # 保存当前光标位置  
tput cup 10 13 # 将光标移动到 row col  
tput civis # 光标不可见  
tput cnorm # 光标可见  
tput rc # 显示输出  
tput sgr0 # 正常屏幕

# #Shell高级

## 1.多进程

(1)wait 和 sleep

在 shell 中使用 wait 是在等待上一批或上一个脚本执行完（即上一个的进程终止），再执行wait之后的命令。

如果wait后面不带任何的进程号或作业号，那么wait会直至当前shell中进程的所有子进程都执行结束后，才继续执行下一步。

sleep是使系统休眠一定的时间之后再去执行下面的任务。有时候写Shell的脚本，用于顺序执行一系列的程序，有些程序在停止之后并没能立即退出，如果shell还没等其退出就接着执行下一行，这么就出乱子了。因此可以使用sleep命令，保证上一条命令执行完以后继续执行下面的命令。

(2) shell 多线程脚本***{}&***

**#!/bin/bash**start\_time=**$*(***date +%s***)*for *((***i=1;i<=1000;i++***))  
do  
{*** *# 测试的命令* sleep 5s  
 echo **"**$i**"  
*}*** &  
***done****#等待1000个后台任务全都执行完成之后，再继续执行执行脚本*wait  
end\_time=**$*(***date +%s***)***echo **"TIME:$*(***expr $end\_time- $start\_time***)*"**

用{}把循环体括起来，后加一个&符号，代表每次循环都把命令放入后台运行，一旦放入后台，就意味着{}里面的命令交给操作系统的一个线程处理了，循环1000次，就有1000个&把任务放入后台，操作系统会并发1000个线程来处理。

(3) 如果是shell中等待使用wait，则不会等待调用函数中子任务。在函数中使用wait，则只等待函数中启动的后台子任务。

(4) 在shell中使用wait命令，相当于高级语言里的多线程同步。

## 2.远程登录的方式

(1).用户名+密码：交互式

(2).密钥验证：配置公钥以后，直接远程登录或执行代码

步骤：

A.机器1生成密钥对并将公钥发给机器2，机器2将公钥保存。

①在机器1上执行：ssh-keygen ，生成公钥和私钥。

②将机器1上的~/.ssh/id\_rsa.pub文件的内置复制，然后将内容追加到机器2上~/.ssh/authorized\_keys文件即可。

③或者使用ssh-copy-id 命令从机器1复制到机器2上，和②的效果一样

B.机器1要登录机器2时，机器2生成随机字符串并用机器1的公钥加密后，发给机器1。

C.机器1用私钥将其解密后发回给机器2，验证成功后登录

配置成功以后则可以通过以下命令免密登录或执行脚本。

*#SSH免密码登陆避免首次需要输入yes*ssh -o StrictHostKeyChecking=no  
ssh -o StrictHostKeyChecking=no root@192.168.137.19 **"~/update.sh"**

（3）常用的命令

ssh-keygen 生成公钥和私钥

非交互式生成：ssh-keygen -t rsa -q -P **""** -f ~/.ssh/id\_rsa  
ssh-copy-id 复制公钥到其他主机

交互式生成：ssh-copy-id -p 22 root@192.168.137.10

## 3.Expect用法

在开发中经常会写一些shell脚本去自动化完成一些重复的工作，但是对于一些具有交互式的命令(ssh,scp)，就会很痛苦。expect的出现就能很好解决这个问题。

通过Shell可以实现简单的控制流功能，如：循环、判断等。但是对于需要交互的场合则必须通过人工来干预，有时候我们可能会需要实现和交互程序如telnet服务器等进行交互的功能。而expect就使用来实现这种功能的工具。

expect的核心是spawn、expect、send、set、interact、expect eof、exp\_continue

spawn：调用要执行的命令

expect：等待命令提示信息的出现，也就是捕捉用户输入的提示：

send：发送需要交互的值，替代了用户手动输入内容

set：设置变量值

interact：执行完成后保持交互状态，把控制权交给控制台，这个时候就可以手工操作了。如果没有这一句登录完成后会退出，而不是留在远程终端上。

expect eof：这个一定要加，与spawn对应表示捕获终端输出信息终止，类似于if....endif

exp\_continue：附加于某个 expect 判断项之后，可以使该项被匹配后，还能继续匹配该 expect 判断语句内的其他项。

注意：

(1)脚本的执行方法与bash shell不一样，比如：expect example.sh。

(2) 向一个脚本传递参数时，expect则将脚本的执行参数保存在数组$argv中，在脚本中一般将其赋值给变量：set 变量名 [lindex $argv 参数]。

(3) 若登陆后便退出远程终端，则写expect eof即可。

(4) expect脚本必须以interact或expect eof结束，执行自动化任务通常expect eof就够了。

(5) set timeout 10：设置超时10s。

**案例1：传参执行：expect expect.sh p1 p2 p3 p4**

**#!/usr/bin/expect***# 传参数*set IP ***[***lindex $argv 0***]***set USER\_NAME ***[***lindex $argv 1***]***set PORT ***[***lindex $argv 2***]***set PASSWORD ***[***lindex $argv 3***]***set timeout 10  
spawn ssh-copy-id -p $PORT $USER\_NAME@$IP  
expect {  
 **"(yes/no)" *{***send **"yes\r"**; exp\_continue***}* "password:" *{***send **"**$PASSWORD**\r"**;***}***}  
expect eof

**案例1：保持交互状态**

**#!/usr/bin/expect**set timeout 10  
spawn ssh root@192.168.137.10  
expect **"password:"**send **"123456\r"**interact

执行完成后保持交互状态(远程登录到了192.168.137.10)，把控制权交给控制台，这个时候就可以手工操作了。

## 4.文件描述符

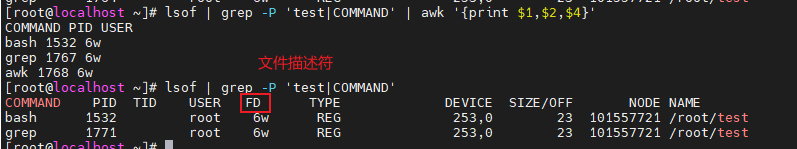
Linux系统将所有设备都当作文件来处理，而Linux用文件描述符来标识每个文件对象。

vincent@geek:~/test**$** exec 6**>**test  
vincent@geek:~/test**$** echo **'i love linux shell!!!'** 1>&6  
vincent@geek:~/test**$** cat test  
i love linux shell!!!

①将文件描述符6指向test文件。

②用&符和文件描述符6就可引用它。可以把文件描述符想象成一个文件的地址或引用，指向的过程就是修改默认位置的过程。而用&符来找到它指向的真正目标文件，从而向其写入数据。

查看文件描述符的命令：lsof



ls /proc/$$/fd 查看当前进程的文件描述符

exec 6<> test 打开文件描述符  
exec 6<&- 关闭文件描述符

注意：exec digit<> filename 用法

①4<>这几个字符不要加空格，必然连着写。filename前可以加空格。

②exec 打开一个文件

③exec 释放一个文件(释放文件的句柄)

④当一个文件FD未被释放，删除原文件也不会影响FD，文件也还在。

## 5.管道文件

①管道分为匿名管道和有名管道两种管道，管道文件是建立在内存之上可以同时被两个进程访问的文件。

②mkfifo创建有名管道，属于系统调用。

③匿名管道( | 就是匿名管道)只能用在父子进程间，有名管道可以在任意两个进程间通信。

④管道空，读阻塞；管道满，写阻塞。

⑤无论是匿名还是有名管道，写入的数据都在内存中。

⑥管道是把一个程序的输出作为另一个程序的输入。

⑦重定向是把输出定向到文件或者标准流。

## 6.多进程并发

**#!/usr/bin/env bash****>**connect.log  
CONCURRENT\_NUM=10  
FIFO=/tmp/fifo.$$  
mkfifo **"**$FIFO**" &&** exec 8<> **"**$FIFO**" &&** rm -f **"**$FIFO**"  
for** i in **$*(***seq $CONCURRENT\_NUM***)*** ; ***do*** echo **"connect"** 1>&8  
***done*for** i in ***{***1..200***}*** ; ***do*** read -u 8  
 ***{*** echo -e **"-- current loop:[cmd id:**$i **; fifo id:** $REPLY**]"** IP=192.168.137.$i  
 ping -c1 -W1 $IP &>/dev/null  
 ***if [*** $? -eq 0 ***]***; **then** echo $IP >>connect.log  
 echo **"**$IP **is up"  
 else** echo **"**$IP **is down"  
 *fi*** sleep 1  
 echo **"connect"** 1>&8  
 ***}*** &  
***done***wait  
exec 8<&-  
echo **"success"**

注意：

①mkfifo先创建管道文件，再通过exec将该文件绑定到文件描述符8。后面的rm操作的疑惑：其实当该文件绑定到文件描述符后，内核已经通过open系统调用打开了该文件，这个时候执行rm操作，删除的是文件的Inode，系统已经连接到文件的block块区。

②如果遇到过这样的情况，就会明白：如果线上的nginx日志没有切分，access.log会越来越大，直接rm access.log文件后，文件不见了，但df查看系统并没有释放磁盘空间。这就是因为rm只是删除了inode，但这之前nginx早已经通过open打开了这个文件，nginx进程的进程控制块中的文件描述符表中对应的fd，已经有相应的文件指针指向了该文件在内存中的文件表，以及其在内存中的v节点表，并最终指向文件的实际存储块。因此nginx依然可以继续写日志，磁盘还在被写入。只有重启或者reload，让进程重新读一次配置，重新打开一遍相应的文件时，才会发现该文件不存在的，并新建该文件。而这时因为Inode节点已经释放，再用df查看时就能看到可用空间增大了。

③REPLY变量。这是上述循环中，用来存放read命令从FIFO中读到的内容。