# 常用命令

## Pwd

1. pwd is print working directory
2. 环境变量PWD储存当前目录

## Touch

1. touch 一般用于建立新文件和修改文件时间
2. 文件时间分为mtime, ctime, atime
   1. Mtime(motification time): 是指最后修改文件数据的时间。
   2. Ctime： 状态时间，指最后修改文件状态的时间，如修改权限或属性的时间
   3. Atime（access time）： 指在最后读取文件数据的时间，如“cat 文件名”
   4. 查看时间的时候，直接”ll 文件名”表示查询mtime
   5. 查看其它时间： ll –time=ctime/atime 文件名
3. 修改文件时间：
   1. Touch 已经存在的文件名： 更新文件所有时间为最新时间
   2. Cp -a 文件名 连属性一起复制文件，但是只能复制原文件的atime 和 mtime， 而ctime 为最新时间（可以理解为是文件的状态改变了）
   3. 单独修改mtime, atime, ctime 分别是：

touch -m/a/c 新时间 文件名

注意：修改mtime和atime的时候，因为改变了文件的时间属性，因此文件的ctime也会更新为最新时间

## 查看文件格式：File

# 文本显示

## Cat

1. 从第一行开始标准输出文本的全部内容
2. 直接输入cat，可以把输入变成标准输出。
   1. 如ls | cat ，就可以把ls 的内容标准输出到屏幕上
   2. Cat>txt 可以创建文件txt，并把输入的内容写入文件。如果再次使用cat>txt编辑文件，则会覆盖原来内容；
   3. 要不覆盖原文内容需要cat>>txt
3. Cat -n/-b 文件名
   1. -n代表把所有行都的行号都输出，而-b 把除空行以外的行号输出
4. Cat -s 文件名
   1. -s 表示suppress ，如果遇到多个空行，则压缩为一个空行输出
5. Cat -A 文件名
   1. 把换行号输出为$，Tab输出为^I, 输出所有空行和特殊符号
6. Cat -T/-E 文件名
   1. -T把tab输出为^I
   2. -E 把换行号用$表示

## Tac

1. 从最后一行开始标准输出文本的全部内容
2. Tac是以二进制形式读入数据
3. Unix和linux的回车是LF(LineFeed), 而DOS/window的回车是LF/CR(CarrageReturn)

## More

## Less

1. 与more相似，可以分页显示内容。但是可以向前翻也可以先后翻。
2. Less在开始显示之前并不会处理完所有的数据，因此比其它的编辑器（如vi）要快。
3. Less 下提供和用户的交互，输入h显示所有 交互下可以使用的命令。

## nl

1. Nl is numbers line of file
2. Nl 用于用标准格式输出文件内容，同时输出行号

## Head/tail

# 用户/用户组管理

1. 修改用户组： chgrp 新用户组 文件名
2. 修改用户/用户组： chown 新用户:新用户组 文件名
3. 添加新用户： useradd
4. 删除用户 ： userdel 用户名
   1. 删除用户的同时删除用户文件：userdel -r 组名
5. 添加新组： groupadd 组名
6. 删除组： groupdel 组名
7. 指定用户的用户组：
   1. 新建用户时： useradd -g 用户组 用户名
   2. 修改用户所属组： usermod -g 用户组 用户名
   3. -g代表的是指定用户的主组，-G则制定用户的附属组（可以有多个附属组，每个附属组名之间用逗号隔开）
   4. 查看用户属性： id 用户名。 可以获知UID,用户所属组，附属组和GID

# 权限问题：

## 普通权限

1. 文件的默认最高权限是666, 目录的默认权限是777.
2. 建立新文件/目录后，文件的默认权限不是666或777,而是要和umask做运算后的权限才是新建目录/文件的默认权限。

比如： umask=0022，那么文件的默认权限就是：

(Rw-rw-rw-) – (-------w--w-) =rw-r--r--

1. 所以最终默认权限是默认最高权限减去umask后三位规定的权限后所得的权限。
2. Umask的第一位表示特殊权限(SUID/SGID/SBIT)
3. 对目录和文件权限有不同的含义：
   1. 目录： r 权限代表能够查看文件中的内容

W权限能够添加和删除目录中的文件

X 权限代表能够cd切换到目录下

问题： 如果现在有目录student/brad, student权限为754, brad的权限为755。现在有用户atlas属于others用户，那么atlas 可以ls brad读出brad目录吗？

答：atlas不能访问brad目录。原因是虽然atlas对目录brad拥有读权限，可是atlas对brad的上级目录只有读权限，没有执行权限。

可以理解为atlas要读取brad目录，就要先切换进入到它的上级目录student中。但是这里因为atlas没有权限进入student目录中，因此即便对brad目录有读权限，可是也不能访问。

* 1. 文件：r 权限代表能够查看文件下的目录

W权限代表能够修改文件中的数据内容

X 权限代表执行文件

* 1. 目录下存放的仅仅是目录下文件的文件名，时间，inode等ll能查看的信息，而没有文件中的数据

1. 对于文件来说最高权限是执行权限，对于目录来说最高权限是写权限
2. chmod
   1. 1:x 2:w 4:r
   2. 比如chmod 755 文件名 ：表示赋予文件拥有者rwx权限，所有组和其他人r-x权限
   3. 也可以是chmod u+r 文件名 ： 表示给所有者添加r权限，chmod u-r 文件名 ： 相应地减去r权限

U: 所有者 g： 所有组 o:其他人

1. Chgrp： 更改所有组
2. Chown：更改所有组, 也能 “chown 所有者：所有组 文件名” 来同时改变文件的所有者和所属组
3. -R 参数代表递归地改变

## 特殊权限

1. 特殊权限有SUID ,SGID 和SBIT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 权限名称 | 适用文件类型 | 文件是否要有执行权限 | 权限表示 | Umask首位数字 | 作用 |
| SUID | 普通文件 | 是 | 用户权限x被s替代 | 4 | 临时转换为root身份执行文件 |
| SGID | 普通文件或者目录 | 是 | 用户组权限x被s替代 | 2 | 临时转化用户组身份执行 |
| SBIT | 目录 | 是 | Others权限x被t替代 | 1 | 可以在目录下新建文件。普通用户只能操作自己建立的文件 |

2. 如果没有给文件或者目录赋予执行权限，那么赋予s或t权限后会显示为S或T

## 隐藏属性

1. 添加/减少/设定文件隐藏属性：Chattr [+-=][参数] 文件名
   1. 比如： chattr +a txt ：表明给文件txt添加a属性，除了超级用户后，其它用户只能给文件添加新内容而不能删除文件或者修改文件中原本的内容
   2. Chattr -R [+-=][参数]目录名: 给目录递归地设定新属性
2. 查看文件隐藏属性：lsattr
3. 只有root才可以添加或删除隐藏属性
4. 文件默认拥有e属性（extent format）

# 查找

## 查找命令所在位置：Which

1. Which命令 在PATH中查找命令所对应的执行文件。
2. 因此，不同的用户（超级用户和普通用户）由于PATH不同，因此查找结果也不一样
3. 有些命令，例如cd，是bash的内置命令，因此通过which 命令无法找到。

## 文件查找： whereis, locate, find

1. Whereis :
   1. -b :只查找二进制文件
   2. -s : 只查找源代码文件
   3. -m：只查找man帮助文件
   4. –u : 查找非以上类型文件的其它特殊文件
2. Find :
   1. 按文件名查找： find 查找范围 -name 文件名
   2. 按文件所有者/组：
      1. find 查找范围 -user 所有者
      2. find 查找范围 -group 所属组
      3. 查找无组/所有者文件：find 范围 -nouser/-nogrroup
   3. 按文件大小： find 范围 -size [+-]大小
      1. 比如 find / -size +30k: 查找根目录下大于30k的文件（-30k则代表小于30k的文件）
   4. 按时间： find / -mtime 4/-4/+4
   5. 按权限： find / -perm [+- ]4755
      1. 在这里4755代表查找刚好权限为4755的文件
      2. +4755列出有4755全部或部分权限的文件
      3. -4755 列出有4755或更多权限的文件
      4. 例如： find /tmp/ -perm +7000 会显示出所有含有s和t属性的文件，哪怕只是--S------。**可以理解为s或s或t**

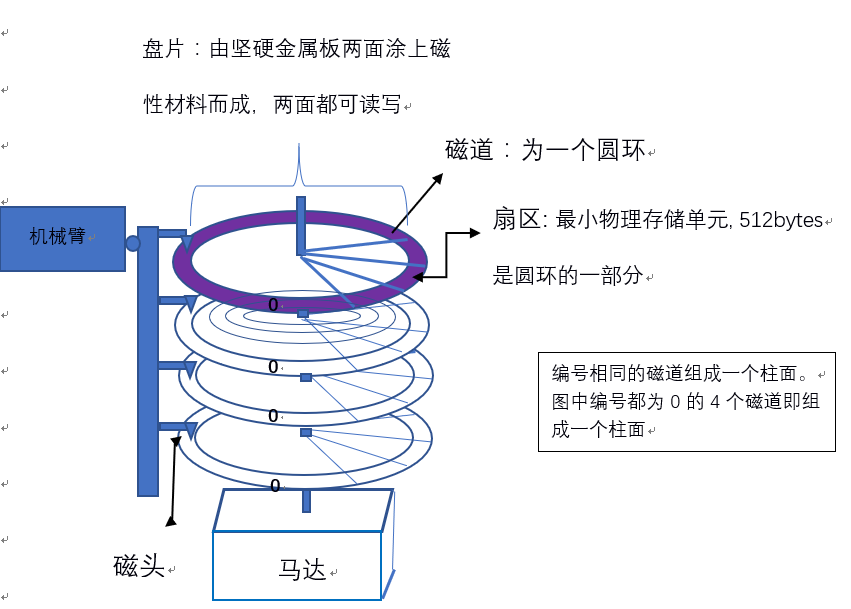
而find /tmp/ -perm -7000必须是同时都具有sst的文件才会被列出来。**可以理解为s与s与t**

* 1. 按文件类型：
     1. Find 范围 -type 类型参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 搜索位置 | 搜索速度 | 搜索对象 | 搜索方式 | 匹配方式 |
| which | PATH | 快 | 执行文件 | 命令 | 完全匹配 |
| Whereis | /var/lib/mlocate数据库 | 快 | 二进制文件，源代码文件，man帮助文件（不能是自己建立的普通文件） | 文件名 | 包含匹配 |
| Locate | /var/lib/mlocate数据库 | 快 | 所有文件 | 文件名 | 包含匹配 |
| Find | 硬盘 | 慢 | 所有文件 | 文件名，文件类型，权限，文件所有者/组，时间，文件大小等 | 完全匹配 |

# 磁盘与文件系统管理

## 硬盘



1. 硬盘的组成： 盘片，机械手臂，磁头
2. 硬盘不只有一层盘片，是由多层盘片组成
3. 柱面：
   1. 编号相同的磁道共同组成一个柱面。比如图中编号都为0的4个磁道组成一个柱面
   2. 柱面是分区的最小单位，也就是说一个磁盘分区不能小于一个柱面的容量
4. 硬盘存储容量=扇区字节数（512bytes）\* 磁道数（或柱面数） \* 盘面扇区数 \* 磁头数
5. 知道硬盘的CHS(cylinder, header, sector)就可以计算出硬盘的容量

## 分区

1. 上面讲到柱面是分区的最小单位。设立分区实际上就是指定分区的起始柱面和结束柱面。如果分配的容量不是整柱面，那么柱面剩下的部分不能被下一个分区所使用
2. 分区类型： 主分区，扩展分区，逻辑分区
3. 为什么最多只能有4个主分区：
   1. 因为硬盘的分区信息储存在第一个扇区的分区表中，分区表的容量为64bytes，每储存一个分区的信息要用掉16个字节，因此只能分4个主分区

4. fdisk命令

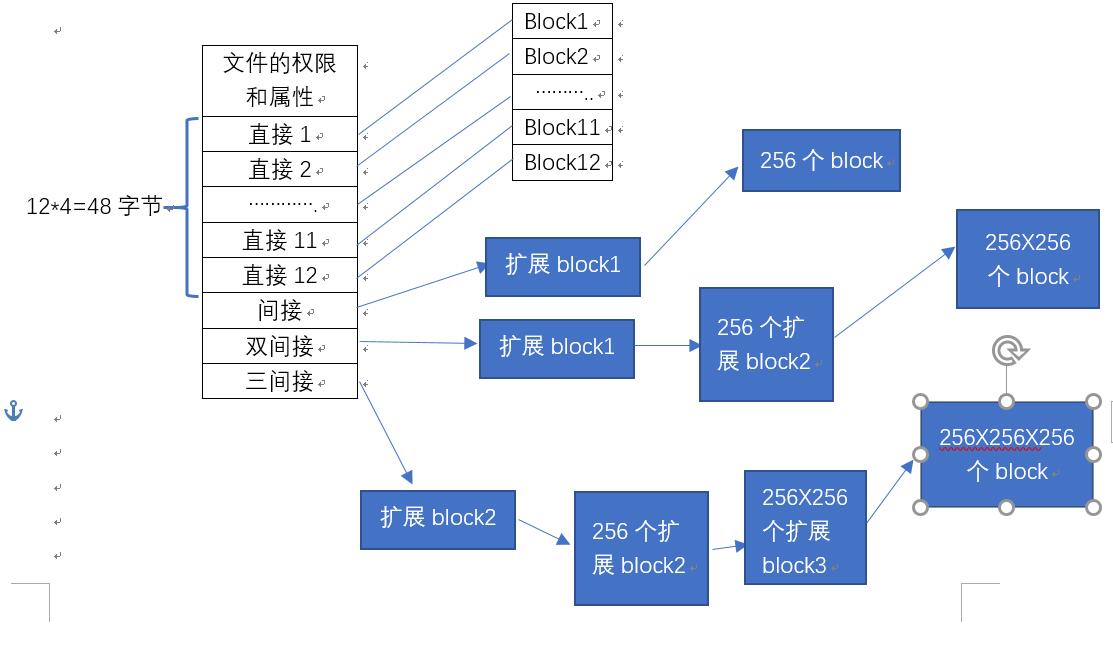
a. fdisk 最大只能处理16TB的分区，更大的就需要用到parted工具

b. 注意： fdisk后面跟的是设备名而不是分区名，所以是写/dev/sda 而不是/dev/sda1

## 文件系统（superblock/inode/block）

### 概述

1. 硬盘———>分区————>格式化————>分配挂载点。 经过这些步骤后，硬盘才能被用户正确访问
2. 指定文件系统，就是其中的格式化步骤
3. 指定文件系统，确定3样东西： super block , inode, data block。格式化后, inode/block的数量大小就固定了。
   1. Super block 记录文件系统的整体信息，包括inode/block的总量，使用量，剩余量等相关信息
      1. 一般superblock的大小为1024bytes
      2. 查看superblock信息： Dumpe2fs [参数] 设备文件名
      3. Block bitmap : 记录哪些block是空的，可以使用
      4. Inode bitmap:记录哪些inode是空的，可以使用
   2. Inode记录文件的权限和属性信息（这里注意区分文件和文件系统）每个文件都有且只有1个inode。
      1. 系统访问文件的时候先去访问inode，确定用户有访问权限之后再根据block号码去寻找相应的block
      2. 不是所有类型的文件系统都存在inode: 索引式文件系统的文件是通过inode记录block编号来访问数据(如ext2文件系统)；而非索引式文件系统则通过链表的方式来访问各个block（如FAT文件系统）
      3. Inode除了记录着文件的时间(mtime,ctime,atime)，权限，所有者/所属组等属性信息，还储存着文件data block的编号。
      4. Inode本身只有128bytes, 储存一个block编号需要4bytes, 这样的话如果文件太大，需要很多的block来储存，就需要储存很多的block号码，inode的容量就会不足。系统是怎么解决这个问题的呢？



上图：

1. 假设文件系统的block大小为1K。
2. Inode直接储存12个block号码，消耗48字节
3. Inode提供间接，双间接，三间接来扩展block号码的存储

**以间接为例子：**先指向一个扩展block1，这个block不是直接用来储存数据的data block，而是用于储存其它block的编号。

这样就有1K即1024bytes的空间来存储block号码，因此可以存储1024/4=256个block号码，再由这些号码去访问真的的data block。

同理，双间接可以储存256X256=65535个block号码。三间接可以储存256X256X256=16777216个block号码。

每个block为1k, 那么inode可以记录的文件大小为(不是储存，是记录):12K+256K+65535K+16777216K=16G，因此block=1k的文件系统下，单一文件最大可以是16G.

* 1. **Data block** 是真正储存数据的地方，文件系统会指定单个block的大小，可以是1k, 2k, 4k
     1. 根据上面inode的计算，不同的block大小会影响单一文件的最大容量

|  |  |  |  |
| --- | --- | --- | --- |
| Block大小 | 1K | 2K | 4K |
| 单一文件最大容量 | 16G | 256G | 2TB |

* + 1. 一个block不能储存超过一个文件。比如一个block=4K, 储存一个3K大的文件，剩下的1K不能再被其它文件利用
    2. 要根据自己储存文件的一般大小来设定block的大小。如果block太大，会造成磁盘空间的浪费；如果block太小，那么读/写一个文件要访问很多block，导致读写效率降低

1. 文件/目录新建过程：

确认文件是否有-wx权限———>inode Bitmap分配未使用的inode———>block bitmap分配未使用的block——>inode bitmap/block bitmap 自动更新——>superblock数据更新

* 1. 新建目录：系统会自动分配给目录一个inode和一个block.
     1. Inode记录目录的inode信息, 还记录分配到的block号码
     2. block用于储存目录下文件的inode信息，通过ll -a 命令可以查看目录下文件的inode号。
     3. 如果目录下的一个block不够，那么系统会再给目录一个block
  2. 新建文件：系统会给文件分配一个inode（当然需要的时候会有对应的扩展block），并根据文件的大小分配对应数量的inode。

1. VFS自动管理文件系统

### **格式化命令：** mkfs (make file system)

* 1. Mkfs[tab][tab]: 列出系统支持的所有文件系统（实际上是列出所有的文件系统创建工具）
  2. 如果有特殊要求，比如要指定block的大小，inode和block的数量等，使用命令**mke2fs**
  3. 注意扩展分区是不能指定文件系统的，因为拓展分区本身只是用来储存逻辑分区的，而不是用于储存数据。

### 文件系统的检测：fsck和badblocks

1.系统突然断电或者是硬盘损坏，都会导致硬盘和内存异步数据的错乱，进而导致文件系统的错乱。因此需要有工具来检测和修复文件系统，恢复数据，那么就要用到fsck

2. fsck命令不能在挂载的文件上使用，否则容易对文件系统造成严重的损害。因此使用fsck检测前需要先umount掉分区。

3. badblocks [options] 设备名： 用来检测硬盘或软盘的扇区是否有坏轨的现象。

## 挂载

1. 文件系统需要与目录树相结合才能被用户所使用，这个结合过程叫做挂载（类似于window中的分配盘符）

## 磁盘管理： df/du

1.df 是从super block 中直接读取磁盘信息，所以比较快；

而du 是要搜索文件，从Inode中读取文件信息，所以比较慢

2.df是查看磁盘的使用信息，du是查看文件的磁盘使用信息

**3.df**

a. 假设/root 下有目录brad, du brad显示的不是brad的磁盘使用情况，而是brad所在的文件系统的情况

b. df -a/-all 列出所有文件系统的信息，包括虚拟文件系统（实际是内存模拟的，不占用磁盘空间）

.

**4.Du:**

a. du 命令是递归地输出文件信息，也就是说如果有目录name/brad， 那么du name 会输出目录name和子目录brad的信息。如果想只输出该目录的信息，可以加参数-s/-S,或者使用—max-depth=0(相当于-s，--summarize)

-S，--separate-dirs: 目录大小不包含子目录的大小

-s, --summarize: 只列出总的目录大小，但大小包含了目录内所有文件/子目录的大小

b. 正常情况下只会显示目录的信息而不会列出普通文件的信息。如果要列出普通文件的信息需要加-a/--all

c. du -L/-D 软连接名： 会找出软连接的目标文件，列出目标文件的信息而不是软连接的信息。

# 文件读写/编辑

## **异步读写提高读写效率**



硬盘

返回

写入

传递

读取

CPU

内存

硬盘的速度比内存要慢很多，因此每次从硬盘读写都需要花费较多的时间。

为了节省时间，内存中划出了缓存区和缓冲区。

内存中的数据如果被CPU处理了，就标为dirty，如果没被处理就标为clean。

* + 1. 缓冲区： 提高数据处理效率。内存从硬盘中读取数据的时候，先把硬盘中的数据读取到缓冲区，然后再从缓冲区读取到内存中。
    2. 缓存区：提高数据的写入效率。处理好的数据先放在内存的缓存器，再一次性写入硬盘。这样可以避免内存频繁地访问硬盘，提高写的效率。
    3. 系统会将频繁访问的数据放入内存的缓冲区，以提高文件读写的效率
    4. 命令sycn命令系统立即把缓存中的内容写入硬盘

## 文件访问日志

如上文讲到，文件的建立需要经过以下步骤：

确认文件是否有-wx权限———>inode Bitmap分配未使用的inode———>block bitmap分配未使用的block——>inode bitmap/block bitmap 自动更新——>superblock数据更新

同样，在我们文件编辑的时候metadata（包括super block，block bitmap, inode bitmap）和inode, data block都会出现变化，理想状况下是这些数据全部都成功更新。

但是有时候文件编辑到一半系统突然非正常关机，上述的数据出现不一致的状况。这种情况系统是怎么处理 的呢？

1. 以前，系统会在开机后会检查super block中的数据（state和valid bit），如果发现系统是非正常关闭的，那么系统就会强制检查整个文件系统来解决问题。 但是这样做的缺点是实在太浪费时间了。
2. 因此文件访问日志应运而生！
   1. **预备**：文件修改的时候，访问日志会先记录是什么数据将要被修改
   2. **实际写入**：修改数据
   3. **结束**：成功修改保存后在日志中写入成功

这样子，每次系统开机只需要检查访问日志就知道有没有上述数据不一致的问题，大大节省了时间。

1. Ext3开始支持日志，ext2就不支持日志

## .文件名.swp文件

1. 文件非正常退出之后，系统会生成一个.文件名.swp（注：在文件名前面还有一个.）的文件，用于保存修改。 可以在恢复文件修改之后用rm命令删除掉该文件。

# 文件链接

## 软连接

1. 命令： ln -s 源文件 软连接名
2. 软连接是新建一个文件，这个文件存储源文件的inode，指向源文件。 因为是新建一个文件，所以需要增加inode和 block的使用

## 硬链接

1. 命令： ln 源文件 硬链接名
2. 硬链接是在源文件的block中储存链接信息，而不是新建一个文件，因此不需要消耗额外的inode和block
3. Ls -l 查看目录信息，权限后面的一列代表使用该inode的文件数量。每一个文件新建的时候都为2（文件本身和.）；而在目录下每建立一个子文件，这个数值都会增加1（因为子目录下的..指向父目录）
4. **对比：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 是否增加inode和block | 删除源文件后能否打开 | 指向目标 | 能否连接目录 | 修改后是否会改变源文件内容 |
| 软连接 | 是 | 否 | 源文件的inode | 能 | 是 |
| 硬链接 | 否 | 是 | 源文件的block | 否 | 是 |

# 下载

## Wget

1.可以离线下载：wget是非互动性下载工具，开始下载后，即便用户下线，wget仍然会继续下载

2.支持http,http2,ftp

3. 稳定性好：因为网络问题下载失败时，会不断尝试，直到下载完成

4. 递归下载？？？

5. 有配置文件/ect/wgetrc,这个文件初始化wget命令的默认值。所以可以通过改变这个文件改变wget的默认选项

# 软件安装

## RPM包

1. Rpm包安装的是二进制文件。

2. 安装rpm包存在依赖性。当依赖的的文件名是以 .so.2结尾的时候，代表这是一个文件，而不是一个软件包

此时是模块依赖，需要依赖一个库文件。

这个时候就要上www.rpmfind.net网上去查找这一库文件是在那一个软件包下，然后再把这个软件包安装上

包名和包全名：操作未安装过的软件包时要使用包全名；操作安装过的软件包使用包名。

因为安装过的软件包包名会被记录在/var/lib/rpm下的数据库里（\_\_db.001之类的），我们操作这些软件包时系统会自动到数据库下查找到这些软件的信息。

1. Rpm包安装： rpm -ivh 包全名
   1. -i(install):安装
   2. -v(verbose):显示详情
   3. -h(hash):显示安装进程
2. Rpm包升级： rpm -Uvh 包全名
   1. -U(upgrade):升级
   2. 如果软件安装过，那么就更新；如果软件没有安装过，那么就相当于安装命令。
3. Rpm包卸载： rpm -e 包名
   1. -e(erase):删除
   2. Rpm包需要卸载命令的原因是因为我们安装的时候并没有指定软件安装的位置，而是安装在软件包开发者指定的默认位置。
   3. 卸载rpm包和安装rpm包一样，都有依赖性

## Yum在线安装

1.由于rpm包安装有依赖性，安装起来很麻烦。因此我们需要yum在线安装，利用yum服务器自动帮我们安装。

2.Yum源文件：在/etc/yum.repos.d（repos是英文repository）目录下有是个文件，这些文件就是yum源文件。

a. 一般默认生效的是第一个Base源

b. mirrorlist 和 baseurl记录yum服务器，一般是其中一个生效。 Yum服务器一般默认是国外的服务器，速度比较慢，可以更换成网易的服务器（http://mirrors.163.com/centos/$releasever/os/$basearch/）。

c. gpgcheck=1表示检查rpm包，确保安全

d. gpgkey记录数字证书

e. enable=1代表容器生效

f. yum源文件识别的是后缀名.repo，改掉后缀名就会失效

g. yum源文件media.repo是代表把yum源改成我们挂载的光盘，而不使用在线yum源

3. 设置好新的yum源之后，yum list显示yum支持的所有软件包，最后一列为yum源文件容器名。

4. 问题：有一次yum update更新之后，执行yum list

出现提示：

Repository updates is listed more than once in the configuration

Repository extras is listed more than once in the configuration

Repository centosplus is listed more than once in the configuration

Repository contrib is listed more than once in the configuration

什么？这些容器都重复了？

Cd到目录/etc/yum.repos.d下，发现又自动生成了一个Centos-BASE.repo文件~~~所以出现了容器重复（因为我之前已经设置了163yum源，不需要这个了！）

解决：把多出来的这个BASE文件删掉就好了

源码包

1.编译，链接

# 定时执行

## Anacron

1.相关配置文件： /etc/anacrontab

2.anacron会自动检测执行的/etc/cron.daily | /etc/cron.weekly | /etc/cron.monthly三个文件上次的执行时间，如果在规定的n天内没有执行，那么anacron就会在延迟若干分钟后执行这三个文件中的执行文件

3. anacron脚本的执行只能精确到天，而不能想就的crontab工具那样精确到小时。

# bc

1. bc is an arbitrary precision calculator language
2. bc有两种属性，lenth和scale,lenth 代表数字的长度，scale代表小数位的长度
3. bc有四个变量：scale ,last, ibase,obase
   1. scale=number 定义小数位，默认为0
   2. last 保存最后输出的数，在bc下直接输入last便可以查看
   3. ibase和obase定义输入和输出的进制转换，范围可以是2~16进制，如果输入数字不在这个范围，则默认为2或16。 默认情况下ibase=obase=10。
   4. 可以输入的有效数为0-9，A-F
   5. 单个出现的数中保留其数值，比如即便在十进制下，A仍然等于10
4. Bc 的语法类似C语言
5. /\* \*/内包含注释，为了支持bash脚本，#后内容也表示注释

# 正则三剑客

## Sed

1. sed是一种流编辑器（stream editor），它一次处理一行内容。
2. 处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。
3. 文件内容并没有改变，除非你使用重定向存储输出。
4. 发现如果直接打印文件sed ‘p’passwd.log的内容，那么每行都会被打印两边.
   1. 这是因为sed编辑器会先把处理的行读取到pattern space 中缓存然后处理完再输出。
   2. 而在这个过程中sed会自动打印pattern space 中的内容，处理完又再输出打印一次，因此打印了两次。
   3. 写成sed -n ‘p’ passwd.log ，-n可以抑制pattern space 的自动打印

## Grep

## Awk

# 重定向输出/输入

## >和>>

1.＞　会覆盖原来的内容，　＞＞　不会覆盖原来的内容

2.重定向符号不用死记，可以看作是漏斗，尖的一端就是数据流动的方向

# Gcc

1. 常用参数：
   1. -o FILE :指定生成执行文件的名字
   2. -l:

# Git

1. 创建本地版本库：
   1. 切换到想要建立版本库的文件夹下，初始化版本库：git init
2. 连接远程库：
   1. 在github按照提示创建远程库
   2. 在本地控制台创建.ssh文件：ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)
   3. 在根目录下找到.ssh文件,打开id\_rsa.pub文件，复制内容，黏贴到github的ssh设置下（创建密钥，获得许可）
   4. 现在可以将内容push到github上了：git push origin master(当然可以是其他分支)
3. 工作区中所有内容加入到暂存区
4. git config –global同步不同的设备：
   1. 设备1的版本库push到远程库上
   2. 设备2从版本库clone文件下来,并且init一下再关联到同一个远程库
   3. 每次结束前，习惯性git add, git commit, git push
   4. 每次开始操作前习惯性：git pull（默认将最近一次push同步到本地master分支）
5. git add -A . : 把 –edit： 打开配置文件，修改名字和邮箱,修改完之后也要commit
6. 修改git的个人配置（用户名和邮箱）
   1. 打开配置文件：git config --global --edit，修改后保存

然后提交：git commit --amend –reset-author

* 1. 或者：git config --global user.name “name” git config --global user.email “name@example.com”
  2. 查看用户名/邮箱：git config -l

# 编辑工具

## Vim

1. vim提供在县帮助，在命令行模式下输入:help command就可以查看命令的帮助
   1. 在线帮助文档的使用，可以输入:help查看
   2. helpgrep [pat] 搜索自己想要查看的帮助.txt文档，然后打开查看
2. Vim的插件（plugin）安装在Vim的runtimepath目录下，你可以在Vim命令行下运行"set rtp“命令查看
3. vim存在多个配置文件vimrc，比如/etc/vimrc，此文件影响整个系统的Vim。还有~/.vimrc(没有的话可以自己创建)，此文件只影响本用户的Vim。而且~/.vimrc文件中的配置会覆盖/etc/vimrc中的配置。
4. vim filelist：支持同时打开多个文档，:bn切换到下一个file，:bp切换会上一个文件
5. 参数：
   1. vim -t: 关联tag files，详见:help tag-commands
   2. vim -D：进入debugging mode
   3. view：以RO模式打开文本，相当于vim -R,保护文件安全
   4. vim +[num]:打开文件同时定位光标到第num行
   5. vim +/[pat]: 打开文件同时定位到第一次出现[pat]的位置,并且高亮[pat]出现的地方
   6. +[command]或- c [command]：打开同时在命令行模式下执行指定命令，可以指定多个命令
      1. vim +split filelist：分不同窗格同时打开多个文件
   7. vim -d filelist: 打开多个文件，显示不同，相当与vimdiff

## ctags

1. ctags是一个索引工具
   1. ctags自动识别文件使用的语言（通过后缀名等方式），然后自动生成源文件的索引文件（tag files）：tags。
   2. ctags会忽略所有普通文件，只为代码源文件创建索引
   3. 有了这个所以文件，就可以轻松地在源文件中定位函数、变量、宏定义......代码阅读变得如此简单！
2. 在vimrc中配置：
   1. ctags一切的索引功能都依赖于索引文件tags，因此系统首先需要能够找到tags文件。如果源代码和tags文件在同一个目录下，那么无需配置，自动就能找到
   2. 如果要在其他子目录下使用cgets,那么就需要在配置文件vimrc下进行配置: set tags=[path]
3. 常用命令：
   1. ctags -R \* : 在目录下使用这个命令，为目录中的所有源文件创建索引。
   2. 光标放到函数/变量名上：
      1. ctrl+]: 跳转到函数/变量定义的位置
      2. ctrl+T: 从函数/变量定义的位置回转到调用的位置
   3. 打开文件的时候： vim -t [tag]： 打开源文件后光标定位在tag的位置（tag替换为要查找的内容）
   4. 命令行模式下：
      1. ：ts(表示taglist)。如果搜索的内容在同一文件/不同文件重复出现，那么:ts命令列出所有出现的位置供用户选择
      2. :tp(表示tag previse): 跳转到查找的tag上一次出现的位置
      3. ：tn(tag next)： 跳转到标记下一次出现的位置