**目录**

**目录操作** ls cd mkdir tree

**文件操作** touch stat cat tac more head\tail ln

**目录文件操作** rm cp mv

**基本权限管理** chmod

**所有者和所属组**  chown umask

**帮助** man wharis apropos info help --help

**搜索** whereis which locate

**find**

**压缩与解压缩、打包** tar

**网络命令** ip ping netstat write wall mail

**vim**

**软件安装管理** pacman yay

**用户和用户组**  passwd usermod chage id su gpasswd

**sudo权限** visudo

**特殊shell**

**文件系统** df du free dumpefs

**mount**

**分区 开机自动挂载** cfdisk mkfs fdisk lsblk blkid

**shell基础** grep

**变量**

**标准读取与运算符** read source

**环境变量配置文件**

**shell登录信息**

**自定义bash快捷键**

**语系与桌面系统配置文件**

**shell编程**

**正则表达式**

**awk、sed、字符处理** cut sort wc

**条件判断** test

**流程控制**

**daemon服务与systemd** systemctl

**system V的init**

**systemd**

**常用服务**

**目录操作**

**Ls exa替代**

-a 所有文件 -A同a不列出. ..目录，隐藏文件示重要系统文件

-h 人性化可读

-i innode节点

-l 长格式

-F /目录 \*可执行文件 @链接 |管道

-d 显示目录信息，而不是目录下的文件信息

-R 递归显示，类似tree

ls -l

- rw-r--r--. 1 root root 1690 5月14 18:20 install.log

文件类型 权限 引用数 所有者 所属组 大小(byte) 修改时间 文件名

文件类型：-普通文件：纯文本文件(ASCII)、二进制文件(scripts)、数据格式文件(如/var/log/wtmp)

d目录-l链接

-b块(存储)设备如硬盘 -c字符(输入)设备,如键盘、鼠标

-s数据接口文件(网络数据承接，启动程序监听用户需求，用户端通过socket进行数据沟通,run/system/journal/stout=)

-p管道符(FIFO,pipe)主要为了解决多个程序同时存取一个文件所造成错误的问题，first-in-first-out

socket、FIFO主要与程序有关，了解进程后再理解，可man fifo、socket了解

权限：最后的. 是SE Llinux安全规则管理，参见ACL

引用数：文件的表示多少文件连接到此节点i-node；目录的表示该目录有多少个一级目录

修改时间：ctime、mtime都会显示

一个文件只一个所有者一个组

一个人可属于多个组一个组多个成员共享文件

**cd**

~ 家目录；不接参数也直接进入家目录

- 上次所在目录

. 当前目录

.. 上级目录 ../..上两级目录

0 =!$

!$ 上一次命令的参数

**mkdir** 创建多个目录用空格隔开

-p 递归创建目录

-m 权限

**tree**  参数与ls类似

注：

linux无扩展名，自己添加扩展名主要为了方便认识文件用途，与执行与否无关

**文件操作 增删改查**

**touch** 文件不存在，创建文件，存在则修改时间戳

-a 指修改文件访问时间access

-m 只修改文件数据修改时间modify

-d 用指定的日期时间而不是现在的时间

atime访问时间 ctime状态修改时间 mtime数据修改时间（两者不一定同步修改）

注意：linux文件无创建时间

**stat** 查看文件的详细信息

-f查看文件系统信息

**Cat bat替代**

-n 行号

-A -vET（v特殊字数、E每行结尾的回车符$、Tab键^I）

**more**分屏显示 less分行显示

空格 向下翻页

b 向上翻页

g 第一页

G 最后一页

回车 向下滚动一行

/字符串 向下搜索

?字符串 向上搜索

n 搜索到字符串时，查找下一个/上一个

q 退出

**vimdiff**

**head tail**

-n 行数（可直接写数字head -10 、tail -n 10）

-f tail用监听文件新增内容

**ln**  （选项 源文件 目标文件）建立硬链接，不给于名字则名字相同

-s 软连接，必须是绝对路径；类似快捷方式

软连接：同步修改 lrwx ->源文件

删除源文件链接文件则显示文件不存在；删除链接文件不受影响

会建立新inode和block，block中不存储实际数据，而是存储源文件文件名及inode号

可连接目录、可跨文件系统

硬链接（不常用）：同步修改

删除哪一个文件，只要还有一个文件存在，这个文件就能访问

不会建立新inode，不会改变inode总数

不能跨文件系统（分区），不同文件系统inode号重新计算

不能连接目录

**目录文件都能操作**

**rm**

-f 强制force

-r 递归删除recursive，用于目录

-i 交互删除，询问用户

-v 详细过程可视

rm -rf

**cp**

-f 强制

-i 交互

-r 递归，用于目录

-v 详细过程可视

-p 目标保留源文件属性，备份时

-u 源文件较目标文件新且同名文件不存在时才复制update

-d 源文件为软连接，则复制软连接而不是源文件

-l 复制为硬链接 -s 复制为软连接

**mv** 剪切移动改名，移动目录不需要-r

-f 强制

-i 交互

-v 详细过程可视

-n 目标文件存在不覆盖移动且不询问

注：

HFS: Filesystem Hierarchy Standard，让使用者了解到已安装软件通常放置于那个目录下

usr是Unix Software Resource的缩写，Unix操作系统软件资源所放置的目录，而不是使用者的数据，所有系统默认的软件（distribution发布者提供的软件）都会放置到/usr

类似Windows 系统的“C:\Windows\（当中的一部份）+ C:\Program files\”这两个目录的综合体，系统刚安装完毕时，这个目录会占用最多的硬盘容量

/var就是在系统运行后才会渐渐占用硬盘容量的目录。

/var目录主要针对常态性变动的文件，包括高速缓存（cache）、登录文件（log file）以及某些软件运行所产生的文件，包括程序文件（lockfile, run file）或者例如MySQL数据库的文件

将许多原本应该要在根目录 （/） 里面的目录，将他内部数据全部挪到 /usr 里面去，然后进行链接设置

/bin --> /usr/bin  
/sbin --> /usr/sbin  
/lib --> /usr/lib  
/lib64 --> /usr/lib64  
/var/lock --> /run/lock  
/var/run --> /run

**权限管理**

**chmod**

-R 递归

ugoa +-= rwx 基本权限，多个身份同时修改权限用“,”隔开

r4 w2 x1 644文件基本权限 755文件执行权限和文件夹基本权限

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 文件 |  | 文件夹 |  |
| r | 读取文件 | cat、more、head | 查看目录下内容 | ls |
| w | 修改文件 | vim、echo | 在目录中新建、删除、复制、剪切目录及文件  文件夹的最高权限 | touch mkdir  rm  cp mv |
| x | 执行文件  文件的最高权限 | 可执行文件  py sh等 | 进入目录,rx需同时有 | cd |

对文件有写权限不能删除文件

ext文件系统格式化时,格式化为三部分：超级块；一部分占空间较小保留inode信息(默认128byte)；大部分格式化为block(数据块)保存真实数据（默认4k），一个放不开占用多个，可能连续也可能分散分布;

文件的inode保存了：文件权限(rwx)、所有者、所属组、大小、atime、ctime、mtime、文件数据真正存在的block编号，每个文件都需要占用一个inode，但不包含文件名

目录的inode记录的这目录下所有一级子文件和子目录的文件名及其inode的对应

保留在上级目录的block中,要删除文件需要拥有上级目录的写权限

**所有者和所属组**

inode号、**权限**、所有者时间、所在的block 编号1

inode号、**权限**、所有者时间、所在的block编号2

/home的block 1

test的block 2

目录**/home**的inode

文件**test**的inode

根目录inode固定是2

判断权限是否可访问文件夹**/home**的block 1

如有，访问block查找文件**test**的inode

通过**test**文件的inode号找到**test**文件的inode信息

判断权限是否可访问**test**文件

如有，通过inode信息中block位置找到test文件实际block

读取block读取test文件内容

test文件> **inode号**

test文件的**数据**

**chown** 选项 所有者：所属组 文件或目录

-R 递归

chown aria:aria test

chown :aria test也可只修改组

**chgrp**

**umask** 让新建文件和目录拥有默认权限

-S 字母表示

umask 不接参数显示默认权限

0022 第1个0时特殊权限SetUID、SetGID、Sticky BIT

022 - --- -w- -w- ,

文件最大权限是666，- rw- rw- rw-，相减-rw-r--r--,644,文件基本权限

目录最大权限是777，d rwx rwx rwx，相减drwxr-xr-x,755，目录基本权限

注意不能直接用数字相减，需换算成权限后相减

umask 033 接参数修改默认权限，但只是临时生效，如需永久生效，修改/etc/profile

umask -s

u=rwx，g=rw，o=rx

**帮助**

**Man tldr替代**

-f 查看命令拥有哪个级别及其作用的帮助 =whatis

-k 查看命令中包含指定字符的所有相关命令的帮助，模糊查找命令 =apropos

man 级别 命令

执行以上命令报错，更新whatis数据库

makewhatis

级别

1. 普通用户可执行的系统命令和可执行文件的帮助
2. 内核可调用的函数和工具的帮助
3. c语言函数的帮助
4. 设备和特殊文件的帮助
5. 配置文件的帮助
6. 游戏的帮助
7. 杂项的帮助
8. 超级用户可执行的系统命令的帮助
9. 内核的帮助

查找命令在哪 whereis

**info**

完整资料类似电子书，分章节

**help**

内置命令的帮助，胆大多数未外部命令，用得少

man help 显示所有内置命令

**--help选项**

**搜索**

**搜索命令**

**whereis** 查找二进制命令、源文件、帮助文档在哪，不能搜索普通文件

-b 只查找二进制命令

-m 只查找帮助文档

**which** 查找二进制命令，有别名还找别名

**搜索普通文件**

**locate** 只能按文件名，但搜索速度快，建立locate数据库在库中搜索文件名

-i 忽略大小写

系统命令也是文件，也可以按照文件名搜索系统命令

locate mkdir

locate数据库在退出登录或重新启动才更新，手动更新

updatedb

配置文件 /etc/updatedb.conf

认为某些文件系统、文件类型和目录没必要搜索，如光盘、.git文件、临时目录/tmp等

**find** 路径 选项 搜索内容 完全匹配的才能搜索到，模糊搜索用通配符 **fd替代**

**按文件名搜索**

-name 区分大小写

-iname 不区分大小写

-inum 按inode搜索

**按文件大小搜索**

-size[+-]大小

25k 正好25k

-25k 小于25k

+25k 大于25k

b默认单位512byte(比特) c字节 w双字节(中文)

k(KB)小写的 M(MB) G(GB)大写的

**按修改时间搜索**

-[atime ctime mtime][+-]时间 按天搜索

-[amin cmin mmin][+-]时间 按分钟搜索

5

当前

6

-5：5天内修改的

5：第5天（5-6）这一天

+5：第6天之前的

**按权限搜索**

-perm 权限 搜索正好等于“此权限”的文件

-perm -权限 搜索全部包含“此权限”的文件

-perm +权限 搜索包含“此权限”的任一权限的文件

1. --w------- 文件1
2. -rw------- 文件2

755 -rwxr-xr-x 文件3

444 -r--r--r-- 文件4

Find . -perm 200 可搜到文件1

find . -perm -200 可搜到文件2、3，文件4不包含--w-------（所有者不含写权限）

Find . -perm +444 可搜到文件2、3、4，文件1任一身份都不含-r--r--r--（不含读权限）

**按所有者、所属组搜索**

-uid 用户ID

-gid 组ID

-user 用户名

-group 组名

-nouser 没有所有者的文件，常用，查找垃圾文件，外来文件可无所有者

find / -nouser

**按文件类型搜索**

-type d 目录

-type f 文件

-type l 连接

**逻辑运算符**

-a and逻辑与

-o or逻辑或

-not not逻辑非

-regex 识别正则表达式 -iregex忽略大小写

find -regex “.\*\(\.txt|\.pdf\)$” 找.txt或.pdf结尾的文件

! 非，否定参数

find /home ! -name “\*.txt”

**同时执行 {}内为命令1的结果作为命令2执行的参数**

-exec 命令2 {} \; 不询问

-ok 命令2 {} \； 询问

find / -nouser -exec rm -rf {} \;

find / \*.bac -ok rm -rf {} \;

find / \*.bac -delete

find / -name “\*.txt” -exec cp {} ./book \;

find / -name “\*.txt” -ok cat {} \; >>./book.txt

**压缩解压缩**

**tar**  压缩包 源文件或目录，多个文件空格分开

-c 打包

-x 解包

-f 指定文件名

-v 显示过程可视

-z gz格式

-j bz2格式

-t 测试不解包，查看包内文件

-C目录 指定解包位置

tar -zxvf .tar.gz

tar -jxvf .tar.bz2

tar -zcvf

tar -jcvf

tar -zcvf test.tar.gz /home/aria/

tar -zxcf test.tar.gz -C /tmp

tar -cvf test.tar test1.py test2.sh test/

将tset1.py、test2.sh、test目录打包为test.tar

tar -xvf test.tar -C /home/aria/test

**关机**

**shutdown**

-h 关机 **=halt** poweroff init 0

-r 重启 **=reboot** init 6

-c 取消已执行的命令

shutdown -[r h] [now 05:30 6:00& +10]

现在 指定5:30,前台终端 指定6:00，后台 10分钟后

**网络命令**

**ip**

-s 如ip -s link 更详细信息

Ip link 显示网络设备运行状态 =ifconfig

Ip addr 显示网卡IP信息

Ip route 显示路由系统信息

Ip link set eth0 up 开启网卡 =ifup eth0

Ip link set eth0 down 关闭网卡 =ifdown eth0

Ip addr add 192.168.31.1 设置Eth0网卡IP地址

Ip addr del 192.168.31.1

**ping** 网络探测，测试通信情况

ping -c 3 [www.baidu.com](http://www.baidu.com)

ping 192.168.103.101

-c 次数

**netstat** 需安装net-tools工具 **ss替代**

-a 列出所有网络，包括socket程序

-n 使用ip地址和端口号显示，不使用域名和服务名

-t 使用TCP协议端口的连接情况

-u 使用UDP协议端口的连接情况

-l 仅显示监听状态的连接

-p 显示PID和程序名

-r 显示路由表

netstat -tlun

状态常用的右listen监听establish建立连接

netstat -tlunp 显示哪个程序占用了端口，并获得程序的PID

netstat -an 查看所有连接

**ifconfig** 现在被ip命令取代，安装net-tools可以继续用ifconfig命令

无参数 主要查看IP地址、子网掩码、MAC地址，其他信息了解即可

Ifconfig eth0 192.168.31.1 临时配置ip地址

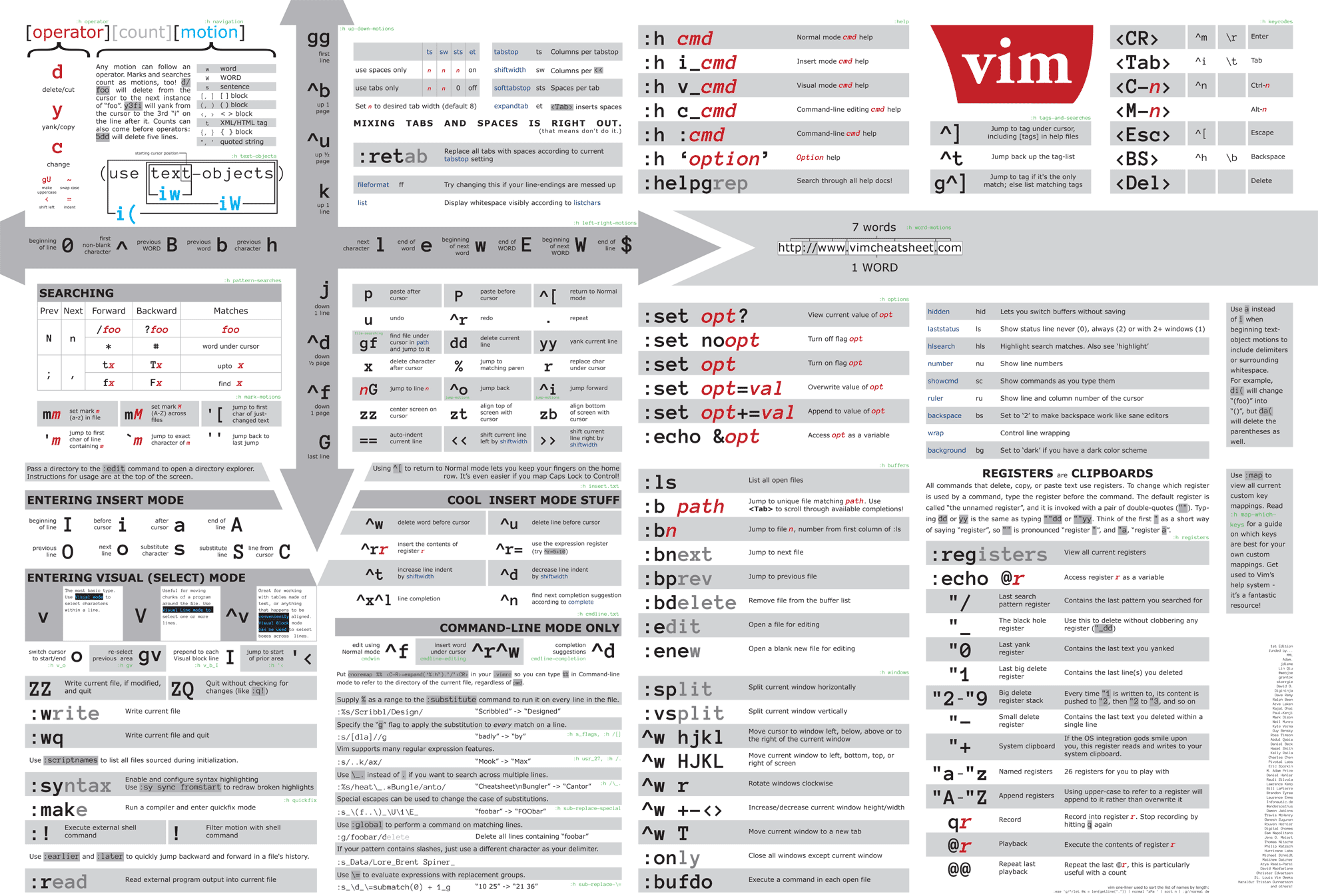
write 给其他用户发送信息

write user2 [pts/1]

I’will shutdownin 5 min

wall 给所有用户发送信息 mail

**Vim**



**命令模式**

h左j下k上l右

w 下一单词词首

b 上一单词词首

e 下一单词词尾

$ 行尾 n$ 当前行n行之后的行尾

0或^ 行首 n^当前行n行之前的行首

f 移动到第一个符合条件的字符前

You can you up,no can no bb，如fp移动到光标后第一个p字符前

% 匹配的括号间切换(){}[]

nG或:n 移动到指定行(nG命令模式、:n编辑模式)

u 撤销操作

x 删除光标所在字符

nx 删除光标所在位置后n个字符

dd 删除整行，之后粘贴则为剪切

ndd 删除多行

dG 删除光标所在行至文件末尾的内容

D 删除光标处到这行行尾的内容 :起始行,终止行d 删除指定范围的行

yy Y 复制单行

nyy nY 复制光标所在行及之后的多行

p 粘贴当前光标行下

P 粘贴当前光标行上

**输入模式** i当前光标所在位置插入随后输入文本，光标后的文本右移

I当前光标所在行的行首(第一个非空白字符)插入随后输入文本

a当前光标位置之后一个字符插入随后输入命令

A当前光标所在行的行尾插入随后输入文本

o当前光标所在行的下面插入，光标停在行首

O当前光标所在行的上面插入，光标停在行首

**编辑模式** / 向下查找字符串

? 向上查找字符串

:set ic 查找时忽略大小写 :set noic

特殊字符需用转义字符\，如\、&、?、^、$，”/10\$”查找10$

查找指定行:/^root查找以root为行首的行/root$查找以root行尾的行

r 替换

R 连续替换，Esc结束

:替换起始处，替换结束处 s/源字符串/替换的字符串/g 替换范围内

:10,20 s/roor/aria/g

:% s/源字符串/替换的字符串/g 替换全篇

不加g只替换每行第一个找到的字符串

:w 保存不退出 :w ~/python/test.py

:q 退出不保存

:! 强制性操作

:wq 保存并退出 ZZ(保存或不保存退出)

:q! 强制退出不保存

**配置** ~/.vimrc中永久定义

:set nu 显示行号

:syn on 语法检测

:set hlsearch 搜索字符串高亮

:set ruler 右下角状态栏

:set showmode 左下角状态栏

以上为默认，:set all查看全部参数

**多窗口编辑**

:sp file2 水平打开(上下排列)

:vs file2 垂直打开(左右排列)

vim -o file1 file2 直接水平打开

Ctrl+WW 切换窗口

:r file2 读取另一文件内容

**区域复制**

ctrl+V 任意选择 可视块

v 选择一行 可视

然后y复制，p粘贴

**定义快捷键** ~/.vimrc中永久定义

:map 快捷键 命令

:unmap

:map ^A ggO#!/bin/bash<Esc>0#<Esc>0

^A ctrl+A,需同时按ctrl+V+A，或先按ctrl+V再ctrl+A，直接输入无效

ggO#!/bin/bash<Esc>o#<Esc>o

在首行插入#!/bin/bash 次行插入# 第三行写程序 <Esc>要诸个字符输入

:map ^E [aria@linux.com](mailto:aria@linux.com)

**vim中与shell交互**

:!命令 执行命令

:r !date 将命令的结果导入文档光标处

文本格式转换

windows文件列的结束符号有两个控制字符，归位字符^M换行字符^J

linux中只有一个换行字符\n功能同^J，linux文本放在win中成为首尾相连的一行

unix2dos 把linux的\n转换成win中^M、^J

dos2unix 把win中^M、^J转换成linux的\n

**vim宏记录**

q宏名字，左下角显示记录中， 然后操作 q表示宏记录

再按q退出宏记录

在需要操作的地方运行@宏名字 n@宏名字执行n次 @@执行上次的宏

**ab命令**

:ab 替代符 原始信息

:ab mymail [aria@limux.com](mailto:aria@limux.com)**软件安装管理**

1. **源码安装**

**位置**

/opt 用户自定义安装软件

/usr/src 内核源代码

/usr/local 用户下载得源代码，如/usr/local/pycharm

**解包**

tar -zxvf pycharm.tar.gz -C /usr/local

tar -jxvf pycharm.tar.bz2 -C /usr/local

**安装源码软件**

**./configure --prefix=/usr/local/pycharm**

#软件配置，可以看帮助文件 ./configure --help |more

**make #编译**

**make install #安装**

**升级**

diff -Naur /usr/local/pycharm/old.file /下载/pycharm/new.file >diff.patch

patch -pn < diff.patch #-pn用来同步两个目录，补丁文件记录得目录中取消几个/，n就是几，此例为2,取消了“/”及"/下载“两个文件夹，依赖补丁文件，其中记录了文件所在；简单点直接将补丁文件复制到需要打补丁的文件夹中，n取0.

然后 make、mke install

**取消升级**

patch -R < diff.patch

**源码包卸载**

rm -rf /usr/local/pycharm

**2、包管理器 pacman、yay**

**常用**

-Syyu 强制升级系统，需要root权限

-S 安装、升级软件包、包组(包含大量软件，可选择序号安装，^表示非，如^2，不安装第2个软件)，需要root权限

-Rsn 卸载软件并清除不需要的依赖，需要root权限

-Sc 清除缓存、保留有效版本，需要root权限

--needed 避免已有的包重复安装

-Qi 查询本地已安装软件详细信息、依赖关系，需要准确软件名，可通过Qe查询，或不接软件名

-Ql 查询本地包文件列表、安装位置

-Fl 查询远程包文件列表、安装位置

-F 查询包含某软件包的软件库及安装位置，记忆模糊时可用

如pacman -F pycharm

-Si 查询包详细信息，不一定是已安装的包，需要准确软件名

-Ss 查询云端软件包简略信息，不需要精确软件信息，包含是否已安装

-Qs 查询本地已安装软件名字作用、依赖关系，需要准确软件名，可通过Qe查询，或不接软件名

-Q 列出、查询本地已安装的软件名字

-Qe 所有明确安装的软件，不需要接软件名

-Qkk 查询软件文件有无变化，”| grep 有变化"

-Qk 查询软件文件有无缺失

-Qo 某文件系统由某软件包所拥有

-Qdt 查询所有不再作为依赖的软件包（孤儿），不需要接软件名

-Qet 查询所有明确安装且不被需要的软件包，不需要接软件名

pactree 依赖树

-r 一个安装的软件包被那些包依赖

参数

-S sync同步

-R remove删除

-Q query查询

-U upgrade升级,可安装 本地包

-F file查询文件数据库

-Sw 下载包但不安装

-Fy 同步文件数据库

-c clean

-Q的参数

-y refresh

-k check

-o owner

-i info

-l list

-s search

-e explicit明确的

-d dependent

-t unrequired

-c changlog

**用户和用户组**

**配置文件**

/etc/passwd 用户信息

用户名称(记忆):密码标志(x,删除后则无密码):UID(通过此识别用户分配权限)：GID(初始组)：用户说明:家目录：登录后的shell

忘记密码进入单用户模式删除此文件的密码标志清空密码

/etc/shadow 影子文件 实际加密密码、密码有效期

用户名称:真正加密密码(加!!､\*使密码临时失效):最后一次修改时间:两次修改间隔时间:有效期:到期前警告天数：过期后宽限天数:账号失效时间:保留

/etc/group 组名:组密码标志:GID:组中的用户（附加用户）

先在group中查询用户组的GID和组名

再在passwd中查找该GID是哪个用户的初始组，同时提取该用户的用户名和UID

最后shadow中提取和这个用户匹配的密码

/etc/gshadow 如果给用户组设定了组管理员并设定了组密码，那么组密码在此

组名:组密码:组管理员用户名:组中的附加用户

**用户管理相关文件**

/home/user

/var/spool/mail/user

/etc/skel 让每个用户家目录都有某个文件或文件夹

修改信息推荐用以上文件，也可以用以下命令，其实也是通过更改以上文件改变信息的

**用户**

**useradd** 一般使用默认设置即可，默认添加用户目录及用户组

/etc/default/useradd(或命令useradd -D) /etc/login.defs 默认值

-G 已存在的附加组

**passwd**

-S 查看用户密码状态/etc/shadow的内容

-d 删除指定账户密码

-l 暂时锁定用户

-u 解锁用户

**usermod** 修改已存在用户的信息

-G 添加到附加组

**chage**  修改用户密码状态

-l 列出用户详细密码状态

**userdel**

-r 删除用户同时删除家目录，不删除就是无属主属组的垃圾文件

**id** 查看UID、GID

**su** 切换身份

- 连带的用户环境变量一起切换，必带 如su - root

**组**

groupadd

groupmod

groupdel 不含初始用户，useradd删除用户时默认删除用户组初始组

gpasswd 把用户添加进组或从组中删除

-a 添加进组

-d 从组中删除

usermod -G group user

gpasswd -a user group

newgroup 改变有效组

**sudo权限** /etc/sudoers

一般用户能够具有 sudo 的使用权，是管理员事先审核通过后才开放 sudo 的使用权的

-b ：将后续的指令放到背景中让系统自行执行，而不与目前的 shell 产生影响  
 -u ：后面可以接欲切换的使用者，若无此项则代表切换身份为 root

**visudo**

root ALL=(ALL) ALL

aria ALL= /sbin/shutdown –r now  
 # %wheel ALL=(ALL) ALL ← 此行是注释的，未生效，要生效拿掉注释  
  
root ALL = (ALL) ALL  
#使用者用户名 登陆者来源主机名称=（可切换的身份） 授权命令（绝对路径）

# %wheel ALL = (ALL) ALL  
#%使用者组名 登陆者来源主机名称=（可切换的身份） 授权命令（绝对路径）

●用户名/组名：哪个用户或用户组可以使用sudo命令，注意**组名前加“%”**。  
●登陆者来源主机名称： 当这个账户由哪部网络主机连线本linux主机，信任来源的意思

如果写ALL， 则代表用户可以由任何主机登录；

如果写固定IP， 则代表用户可以由指定的主机登录。

如果我们在这里写本机的IP地址，不代表只允许本机的用户使用指定命令，而代表 指定的用户可以从任何IP地址来登录当前服务器。  
●可切换的身份： 就是把来源用户切换成什么身份使用。

（ALL）代表可以切换成任意身份。 这个字段可以省略。  
●授权命令： 代表root把什么命令授权给普通用户。

默认是ALL， 代表任何命令。

如果需要给哪个命令授权，则只需写入命令名即可，**一定要写成绝对路径**

**实例**

usermod -a -G wheel pro1 将用户加入wheel组即可使用sudo权限

%wheel ALL=（ALL） NOPASSWD: ALL 不需要密码即可使用 sudo

aria ALL=（root） !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]\*, !/usr/bin/passwd root

! 代表不可执行，可以执行passwd 任意字符，但是passwd与passwd root这两个指令例外！如此一来 aria就无法改变 root 的密码了

通过别名创建 visudo

User\_Alias ADMPW = pro1, pro2, pro3, myuser1, myuser2  
 Cmnd\_Alias ADMPWCOM = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]\*, !/usr/bin/passwd root  
 ADMPW ALL=（root） ADMPWCOM

通过User\_Alias创建出一个新帐号，这个帐号名称及Cmnd\_Alias（命令别名）、Host\_Alias（来源主机名称别名） 都需要使用大写字符的

User\_Alias ADMINS = pro1, pro2, pro3, myuser1  
 ADMINS ALL=（root） /bin/su -

输入“ sudo su - ”并且输入“自己的密码”后， 立刻变成 root的身份！不但 root 密码不会外流，多人共管一部主机时常常使用的技巧

两次 sudo 操作的间隔超过 5 分钟，那就得要重新输入一次你的密码了

**特殊shell**

系统账号的shell是 /sbin/nologin，是个无法登陆的合法 shel ，系统帐号不需要登陆

无法登陆”指的仅是：“这个使用者无法使用bash交互或其他shell来登陆系统，但可使用其他的系统资源

如邮件服务器上的帐号大部分用来收受信件并不需要登陆主机，这些单纯使用mail的帐号以/sbin/nologin做他们的shell，我们可以登录管理主机，其他人无法登录。

想让nologin者知道无法登陆主机时，可创建/etc/nologin.txt，说明不能登陆原因，下次登录显示此文件的内容，而不是默认的内容了

**PAM模块**

是一套应用程序接口API，提供验证机制，只要使用者将验证阶段的需求告知PAM后， PAM就能够回报使用者验证的结果（成功或失败）

仅是一套验证的机制，又可以提供给其他程序所调用引用，不论使用什么程序，都可以使用PAM来进行验证，能让帐号密码或者是其他方式的验证具有一致的结果

PAM 用来进行验证的数据称为模块Modules，每个 PAM 模块的功能都不太相同。

PAM 借由一个与程序相同文件名的配置文件来进行一连串的认证分析需求。

1. 使用者开始执行/usr/bin/passwd并输入密码；  
   2. passwd这支程序调用PAM模块进行验证；  
   3. PAM模块会到**/etc/pam.d/**找寻与程序passwd同名的配置文件；  
   4. 依据/etc/pam.d/passwd内的设置，引用相关的PAM模块逐步进行验证分析；  
   5. 将验证结果（成功、失败以及其他讯息）回传给passwd；  
   6. passwd根据PAM回传的结果决定下一个动作（重新输入新密码或者通过验证）

cat /etc/pam.d/passwd  
 #%PAM-1.0 <==PAM版本的说明而已！  
 auth include system-auth <==每一行都是一个验证的过程  
 account include system-auth  
 password substack system-auth  
 -password optional pam\_gnome\_keyring.so use\_authtok  
 password substack postlogin  
 验证类别 控制标准flag PAM 模块与该模块的参数

include这个关键字代表“请调用后面的文件来作为这个类别的验证”上述的每一行都要重复调用/etc/pam.d/system-auth那个文件来进行验证

验证类别type

**auth authentication认证**：通常需要密码来检验，后接模块用来检验使用者的身份。  
**account帐号：**authorization授权时检验使用者是否具有正确的权限，如过期密码无法登陆  
**session会议期间：**在这次登陆或使用指令期间，PAM所给予的环境设置。

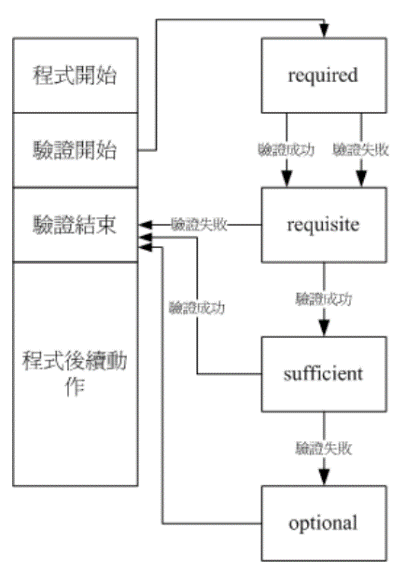
如使用su或sudo可在/var/log/secure里发现关于pam的说明，而且记载 的数据是“session open, session close”的信息！  
**password密码：**提供验证的修订工作时，就是修改/变更密码

顺序：要先验证身份auth后，给予适当的授权与权限设置account，才需要记录登陆与登出的信息session，如运行期间需要密码修订才用到password

控制标准control flag 验证通过的标准，放行方式

required  
此验证若成功则带success标志，若失败则带有 failure 的标志，但不论成功或失败都会继续后续的验证流程。  
由于后续的验证流程可以继续进行，因此相当有利于数据的登录 （log） ，这也是 PAM 最常使用 required 的原因。  
requisite  
若验证失败则立刻回报原程序 failure 的标志，并终止后续的验证流程。若验证成功则带有 success 的标志并继续后续的验证流程。

这个项目与 required 最大的差异，就在于失败的时候还要不要继续验证下去？由于 requisite 是失败就终止， 因此失败时所产生的 PAM 信息就无法通过后续的模块来记录了。  
sufficient  
若验证成功则立刻回传 success 给原程序，并终止后续的验证流程；若验证失败则带有 failure 标志并继续后续的验证流程。与 requisits 刚好相反！  
optional  
这个模块控制项目大多是在显示讯息而已，并不是用在验证方面



PAM 模块与该模块的参数

/etc/pam.d/\*：每个程序个别的 PAM 配置文件；  
 /lib64/security/\*：PAM 模块文件的实际放置目录；  
 /etc/security/\*：其他 PAM 环境的配置文件；  
 /usr/share/doc/pam-\*/：详细的 PAM 说明文件

pam\_securetty.so：  
限制系统管理员 (root) 只能够从安全的 (secure) 终端机登入；安全的终端机设定写在 /etc/securetty，知道为什么root无法透过ssh登入Linux主机  
 pam\_nologin.so：  
这个模块可以限制一般用户是否能够登入主机之用。

当/etc/nologin存在时，则所有一般使用者均无法再登入系统，在他们的终端机上会将该文件的内容显示出来！所以，**正常的情况下这个文件应该是不能存在系统中的**。 但这个模块对 root 以及已经登入系统中的一般账号并没有影响。 (注意喔！这与 /etc/nologin.txt 并不相同！)  
 pam\_selinux.so：  
SELinux 是个针对程序来进行细部管理权限的功能。由于 SELinux 会影响到用户执行程序的权限，因此利用PAM模块，将SELinux暂时关闭，等到验证通过后再予以启动！  
 pam\_console.so：  
当系统出现某些问题，或者是某些时刻你需要使用特殊的终端接口登入主机时， 这个模块可以帮助处理一些文件权限的问题，让使用者可以透过特殊终端接口顺利登入系统。  
 pam\_loginuid.so：  
系统账号与一般账号的 UID 是不同的，一般账号UID均大于 1000 才合理。 为了验证使用者的UID真的是我们所需要的数值，可以使用这个模块来进行规范！  
 pam\_env.so：  
用来设定环境变量的一个模块，如果你有需要额外的环境变量设定，可以参考 /etc/security/pam\_env.conf 这个文件的详细说明。  
 pam\_unix.so：  
这是个很复杂且重要的模块，可以用在验证阶段的认证功能，可以用在授权阶段的账号许可证管理，可以用在会议阶段的登录文件记录等，甚至也可以用在密码更新阶段的检验！非常丰富的功能！ 这个模块在早期使用得相当频繁！  
 pam\_pwquality.so：  
可以用来检验密码的强度！包括密码是否在字典中，密码输入几次都失败就断掉此次联机等功能，都是这模块提供的！ 最早之前其实使用的是 pam\_cracklib.so 这个模块，后来改成 pam\_pwquality.so 这个模块，但此模块完全兼容于pam\_cracklib.so同时提供了 /etc/security/pwquality.conf 这个文件可以额外指定默认值！比较容易处理修改！  
 pam\_limits.so：  
ulimit限制用户的某些系统资，就是这个模块提供的能力！还有更多细部的设定可以参考：/etc/security/limits.conf 内的说明

**文件系统 duf替代（df、du的融合）**

**df** [挂载点或分区设备文件名] 查看已挂载的文件系统命令

-a 所有文件系统信息，包含特殊文件系统，几乎都保存在内存中，占用量为0

-T 显示文件系统类型如ext4、vfat 类似lsblk -f

**du** [目录或文件名] 统计目录或文件占磁盘空间大小

无参数 默认只统计当前目录下所有子目录的总磁盘占用量，不统计子文件

-a 显示每个子文件磁盘占用量

**-s** 统计总磁盘占用量，不列出子目录和子文件磁盘占用量

ls -l可看到文件大小，但是目录多数为4k，这是因为目录下子目录名和子文件名保存到父目录(即当前查看的目录)的block中(默认4k)，父目录下子目录及子文件不多，一个block就能放下，父目录就占一个block(4k)

类似图书馆书籍检索目录和实际书籍的关。ls -l查看到的是一页纸的目录，但实际书籍有多少看不到。如果不想看父目录下子目录名和子文件名的占用空间，而是想看父目录下子目录和子文件实际磁盘占用量用du。

df du统计的数据有时不一致。

df从文件系统出发，通过文件系统未分配的空间确定已分配的空间大小，不仅考虑文件占用空间，还考虑被命令、程序占用的空间(文件已删除，但程序未释放空间)，统计分区更准确，是真正的空闲空间。

du面向文件，只计算文件或目录占空磁盘

**free** 查看内存和swap分区使用情况

buffers缓冲:写入数据时加速的，把分散的数据集中写入硬盘（冲入）

cached缓存:读取数据时加速的，把读取的数据保存内存中，再次读取在内存中读取

$ free -m

total used free shared buffers cached

Mem: 1009 741 267 0  **104 359**

-/+ buffers/cache: 278 **731**

Swap: 1537 0 1537

注意-/+ buffers/cache:这一行——第一个数据是真正的“正在使用”的内存，第二个数据是“可利用”的内存（但不是“未使用”）。

上述例子是从一台总内存1G的笔记本电脑上取得的，已使用的内存有741M，剩下的内存貌似连浏览器都开不了呢！然而，根据我们上面的解释，这台机器只有278M内存是正在使用的，而剩下的731M内存都是可以使用的。根据后面的数字，104M的内存作为缓冲区，359MB内存用来缓存数据，两者都可以在需要时被释放。剩下的268M内存是完全没有使用的。 完全未用的：731-104-359

**dumpe2fs** /dev/sdb2 显示磁盘最详尽状态

-h 超级块 -b显示块

**mount**

无参数 系统中已挂载的设备 -l会显示卷标名称

-a 依/etc/fstab自动挂载

mount [-t 文件系统][-L 卷标名][-o 特殊挂载] 设备文件名 挂载点

**特殊挂载选项 -o**

defaults rw suid dev exec auto nouser async

atime/noatime 更新访问时间，默认更新

async/sync 异步/同步

auto/noauto mount -a时，自动按照fstab挂载

exec/noexec 是否允许再文件系统执行可执行文件

remount 重新挂载，指定修改特殊权限

rw/ro 读写/只读

suid/nosuid SetUID权限，默认有

user/nouser 是否允许普通用户挂载，默认不

usrquota 用户磁盘配额，默认不

grpquota 组磁盘配额，默认不

mount -o remount，noexec /home

mount /dev/sdb1 /mnt/disk**分区**

**cfdisk 图形界面，好用**

fdisk 支持MBR GPT gdisk 支持GPT

sfdisk 逼格高fdisk

parted 即时生效，无保存生效概念；可调整分区大小(resizepart)

调整已挂载使用的分区，不会影响分区知中的数据

但一定要先卸载分区，且要调整的分区已经建立文件系统

**mkfs**  格式化分区，不能调整默认参数（一般不用调整），但简单易用

mke2fs 格式化分区

-t 文件系统如ext2/3/4

-b 指定block大小

**fdsik**

**-l** 所有挂载的磁盘的详细分区情况

**lsblk** 树形简单显示挂载磁盘信息及挂载点

-f 文件系统信息 类似df -T

**blkid** 每个文件系统的UUID

lsusb

-v 详细信息

lspci

-k 内核驱动、模块

**SWAP分区**

cfdisk /dev/sdb 等命令建立swap分区 编号82

mkswap /dev/sdb1 格式化swap分区

swapon /dec/sdb1 加入swap分区

swapoff /dec/sdb1 取消swap分区

开机自动挂载

/dev/sdb1 swap swap defaults 0 0

**开机自动挂载** /etc/fstab

分区设备文件名或UUID 最好用UUID(硬盘通用唯一识别码),新加磁盘重启后顺序会改变

挂载点

挂载参数mount

指定分区是否被dump备份 0不1每天2不定期

每个分区的lost+found文件夹就是自动备份数据所在

指定分区是否被fsck检测 0不 其他数字为优先级，越小优先级越高，根分区为1 fsck，

此时为开机自检，一般无需人工执行

**工作日程 自动化维护**

软件及系统更新pacman、自动updatedb更新文件名数据库(locate)、mandb、登录文件logratate轮替、logwatch分析、移除暂存文件tmpwatch在背景下定时完成

命令/运行 记录要运行的任务 记录执行的任务 服务

单一突发性 at /var/spool/at atd(不一定enable)

循环例行性 crontab个人 /var/spool/cron(不用vim) /var/log/cron cronie

编辑/etc/crontab系统

**整分钟时间：最小单位示分钟，每分钟检查一次**/var/spool/[cron|at]、/etc/crontab**;可脱机继续连续工作**

**at**

先搜寻/etc/at.allow 在者能用，不在者不能用at，即使不在at.deny中；一个账号一行 管理严格  
 若/etc/at.allow不存在 就寻找/etc/at.deny这个文件，在者不能用at不在者都可以用at 管理松散

若都不存在 只有root能用at

只使用一个文件进行限制，系统默认保留at.deny

at [-mldv] TIME 命令最好使用绝对路径(运作时在下达指令的工作目录)

-m at工作完成后，即使无输出讯息，亦以emai通知

-l 相当于**atq**，查询所有该用户的at工作；

-d 相当于**atrm** ，取消一个在at排程中的工作 at -d 工作号码；

-c 列出实际指令内容 at -c 工作号码

-v 可以使用较明显的时间格式列出at排程中的任务栏表；

TIME：时间格式

HH:MM 04:00 今日HH:MM进行，若已过，则明天的HH:MM进行

HH:MM YYYY-MM-DD 04:00 2015-07-30 强制规定在某年某月的某一天的特殊时刻进行

HH:MM[am|pm] [Month] [Date] 04pm July 30 也是一样，强制在某年某月某日的某时刻进行

HH:MM[am|pm] + number [minutes|hours|days|weeks]

now + 5 minutes 在某个时间点『再加几个时间后』才进行

at的标准输出/标准错误输出都会传送到执行者的mailbox而终端机看不到信息，若想看到使用echo "Hello" > /dev/tty1

**batch**  CPU的工作负载load小于 0.8 的时候，才进行工作，也是使用at指令，可用atq、atrm管理

load：CPU 在单一时间点所负责的工作数量，不是CPU使用率。  
 若一只程序需一直使用CPU的运算功能，使用率达100%，CPU的load趋近于1，仅负责一个工作

若同时执行这样的两个程序，CPU的使用率还是100%，工作负载则变成2

uptime查询1、5、15分钟平均负载量

/etc/

│-----cron.d/ #默认使用此文件下配置，若有crontab则使用它

│-------**0hourly** **#参照此文件执行其他文件，若不正确此目录下及anacron启动器都不运行**

│-----crontab #默认不存在，存在则用此文件，systemctl status cronie可能输出此文件不存在

│-----cron.deny #默认保留deny文件，若无则存在allow文件

│-----cron.minutely/ #每分钟运行一次

│-----cron.hourly/ #每小时的第1分钟运行一次目录下的可执行文件或链接

**│------0anacron #确保anacron程序每天执行一次**

**│-----anacrontab**

anacron

**│-----cron.daiy/**

**│-----cron.weekly/**

**│-----cron.monthly/**

anacron其实每小时都会被抓出来执行一次，担心anacron误判时间参数，因此/etc/cron.hourly/里面的 anacron才会在档名之前加个0 (0anacron)，让anacron最先进行，就是为了让时间戳先更新！以避免anacron 误判crontab尚未进行任何工作

crontab、at定时去执行，过了就过了，不重新来一遍；anacron定期执行，某一周期的执行

如果每周日的需要执行的动作是放置于/etc/crontab，该动作只要过期就过期了，并不会被抓回来重新执行。

但如果是放置在/etc/cron.weekly/目录下，该工作就会定期，一定会在一周内执行一次

如果你关机超一周，那么开机后的数个小时内，该工作就会主动的被执行，/etc/anacrontab内有定义 **cronie** /etc/cron.allow、/etc/cron.deny同上 基础系统默认使用systemd/Timers，cronie是cron的实现，也提供了0anacron程序

**crontab** [-u username] [-l|-e|-r] 个人的例行性循环工作，/usr/bin/crontab

-u ：只有 root 才能进行这个任务，可帮其他使用者建立/移除 crontab 工作排程；

**-e ：编辑 crontab 的工作内容**

-l ：查阅 crontab 的工作内容

-r ：移除所有的crontab的工作内容，若仅要移除一项，请用-e去编辑

范例一：用aria的身份在每天的 12:00 发信给自己

crontab -e # 此时会进入 vi 的编辑画面让您编辑工作！注意到，每项工作都是一行。

0 12 \* \* \* mail -s "at 12:00"aria < /home/dmtsai/.bashrc

分 时 日 月 周 |<==============指令串==============>|

0-59 0-23 1-31 1-12 0-7(0、7都表示周天) 周与日月不共存

|  |  |
| --- | --- |
| 特殊字符 | 代表意义 |
| \*(星号) | 代表任何时刻都接受的意思！如上例的日月周都是\*，代表着不论何月、何日的礼拜几的12:00都执行后续指令 |
| ,(逗号) | 代表分隔时段的意思。若要下达的工作是3:00与6:00时，就是0 3,6 \* \* \* command，代表3与6都适用 |
| -(减号) | 代表一段时间范围内，若腰8点到12点之间的每小时的20分都进行一项工作：20 8-12 \* \* \* command |
| /n(斜线) | 那个n代表数字，亦即是『每隔n单位间隔』的意思，例如每五分钟进行一次，则：\*/5 \* \* \* \* command 用\*与/5搭配，也可以写成 0-59/5 |

**/etc/crontab**  MAILTO可发送邮箱到893904875@qq.com等（如未找到/usr/bin/sendmail,就需要SMTP服务）

\* \* \* \* \* **user-name** command to be executed 需要先指定执行者

执行脚本

系统 个人

/etc/crontab /var/spool/cron/\*

 **/etc/cron.d/\* #可在此目录下自定义自己脚本的例行性工作配置文件** -rw-r--r--. 1 root root 128 Jul 30 2014 0hourly  
 ……

cat **/etc/cron.d/0hourly**  
 # Run the hourly jobs  
 SHELL=/bin/bash  
 PATH=/sbin:/bin:/usr/sbin:/usr/bin  
 MAILTO=root  
 01 \* \* \* \* root run-parts **/etc/cron.hourly** #同/etc/crontab

run-parts脚本会在大约5分钟内随机选一个时间来执行/etc/cron.hourly目录内的所有执行文件

在/etc/cron.hourly/的文件，必须是能被直接执行的指令脚本，而不是分、时、日、月、周的设定值

/etc/cron.hourly/，/etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly/后三者由anacron执行，

anacron的执行方式则放在/etc/cron.hourly/0anacron

1. crontab -e命令 个人化的行为，别人不知道
2. /etc/crontab配置文件（大家都能读取） 系统维护管理，易追踪
3. /etc/cron.d/diy.file 自己的脚本、软件，全新的配置文件  
   4、将指令放置到(或链接到)/etc/cron.hourly/ 目录下，则该指令就会被cronie在每小时的1分开始后的5分钟内，随机取一个时间点来执行

**anacron**

cat **/etc/cron.hourly/0anacron**

#!/bin/sh

# Check whether 0anacron was run today already

if test -r /var/spool/anacron/cron.daily; then

day=`cat /var/spool/anacron/cron.daily`

fi

if [ `date +%Y%m%d` = "$day" ]; then

exit 0;

fi

# 检验前一次执行 anacron 时的时间戳！

# Do not run jobs when on battery power

if test -x /usr/bin/on\_ac\_power; then

/usr/bin/on\_ac\_power >/dev/null 2>&1

if test $? -eq 1; then

exit 0

fi

fi

/usr/sbin/anacron -s

# 仅是执行 anacron -s 的指令

anacron [-sfn] [job]..

anacron -u [job]..

选项与参数：

-s ：开始一个连续的执行各项工作 (job)，会依据时间记录文件的数据判断是否进行；

-f ：强制进行，而不去判断时间记录文件的时间戳；

-n ：立刻进行未进行的任务，而不延迟 (delay) 等待时间；

-u ：仅更新时间记录文件的时间戳，不进行任何工作。

job ：由 /etc/anacrontab 定义的各项工作名称

cat **/etc/anacrontab**  
SHELL=/bin/sh  
PATH=/sbin:/bin:/usr/sbin:/usr/bin  
MAILTO=root  
RANDOM\_DELAY=45 # 随机给予最大延迟时间，单位是分钟  
START\_HOURS\_RANGE=3-22 # 延迟多少个小时内应该要执行的任务时间  
1 5 cron.daily nice run-parts /etc/cron.daily  
7 25 cron.weekly nice run-parts /etc/cron.weekly

@monthly 45 cron.monthly nice run-parts /etc/cron.monthly

天数 延迟时间 工作名称定义 实际要进行的指令串

# 天数单位为天；延迟时间单位为分钟；工作名称定义可自定义；指令串则通常与 crontab 的设定相同

more **/var/spool/anacron/\***

/var/spool/anacron/cron.daily

20150731

/var/spool/anacron/cron.monthly

20150703

/var/spool/anacron/cron.weekly

20150727

# 上面则是三个工作名称的时间记录文件以及记录的时间戳

**天数**：anacron 执行当下与时间戳 (/var/spool/anacron/内的时间纪录文件) 相差的天数，若超过此天数，就准备开始执行，若没有超过此天数，则不予执行后续的指令。  
**延迟时间**：若确定超过天数导致要执行排程工作了，那么请延迟执行的时间，因为担心立即启动会有其他资源冲突的问题吧！  
**工作名称定义**：只是会在 /var/log/cron 里头记载该项任务的名称！通常与后续的目录资源名称相同即可。  
**实际要进行的指令串**：与0hourly相同，通过run-parts来处理

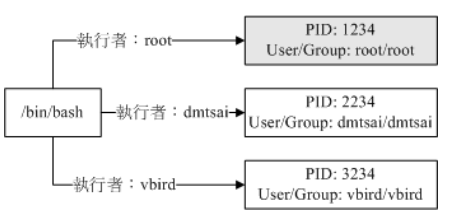
执行流程应该是这样的 (以 cron.daily 为例)：  
1. 由 /etc/anacrontab 分析到 cron.daily 这项工作名称的**天数**为 1 天；  
2. 由 /var/spool/anacron/cron.daily 取出最近一次执行 anacron 的**时间戳**；  
3. 由上个步骤与目前的时间**比较**，若差异天数为 1 天以上 (含 1 天)，就准备进行指令；  
4. 若准备进行指令，根据/etc/anacrontab的设定，将延迟5分钟+ 3小时 (看 START\_HOURS\_RANGE的设定)；  
5. 延迟时间过后，开始执行后续指令，亦即『 run-parts /etc/cron.daily 』这串指令；  
6. 执行完毕后， anacron 程序结束

**进程管理**

程序 硬盘中的实体程序文件，用户执行来触发

进程 运行时加载至内存中的程序

为了管理进程，进程给与执行者权限、属性等参数，并包括程序所需的脚本与数据，依据登入者的  
UID/GID (/etc/passwd) 最后给一个**PID**，通过PID判断权限，由某进程（父进程）衍生出来的其他进程（子进程）在一般状态下，也会沿用这个进程的相关权限



**程序**

**进程**

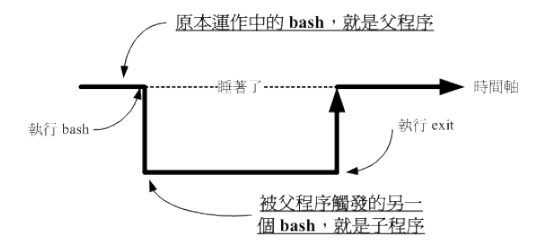
**父进程PPID**

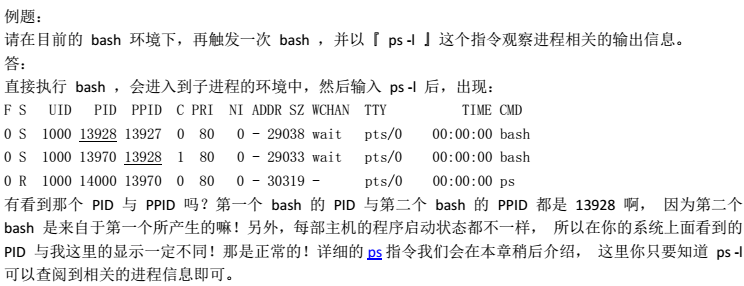
用此bash执行其他命令如ls、cat等

**子进程**

**子进程**

**PID**





若将有问题的进程关闭，又自动的产生，且PID与原先的不一样，若不是 crontab的影响，肯定有一支父进程存在，杀掉子进程后，父进程就会主动再生一支。

**shell基础**

用户 应用 shell 内核层 硬件

shell种类 sh bash zsh(manjaro自带)，也是命令，可临时切换shell

**chsh**

-s shell名称 更改系统预设的shell环境eg.chsh -s /bin/ zsh，重启后即更改了shell

-l 列出系统可用的shell，等同打开 /etc/shells文件

echo $SHELL 显示当前shell

**echo**

-e 支持反斜线控制的字符转换

-n 取消输出后行末的换行符（内容输出后不换行）

**控制字符**

\\ \本身

\a 警告音

\b 退格键

\c 同-n

\e ESC键

\f 换页符

**\n 换行符**

**\r 回车键**

**\t 制表符，TAB键**

\v 垂直制表符

**\e[1; 标准格式，颜色输出开始**

**\e[0m 颜色输出结束**

**\e[1m 内部的都有颜色**

echo -e “\e[1;32m 颜色输出 \e[0m” 输出绿色\e[1;颜色输出开始\e[0m颜色输出结束

字体： 30m黑色31m红色32m绿色33m黄色34m蓝色35m洋红36m青色37m白色

字体背景：40m黑色41m红色42m绿色43m黄色44m蓝色45m洋红46m青色47m白色

**脚本格式**

#!/bin/bash

#comment说明

#authors作者版权

-----------------------------

命令

-----------------------------

**运行方式**

加执行权限 1、chmod 755 bash.sh 2、chmod o+x bash.sh

运行 1、/home/aria/bash.sh 2、./bash.sh 3、bash bash.sh

把脚本目录添加到PATH路径内，编辑./.bashrc或./.zshrc，然后执行source ~/.bashrc

如果这样还不行，表示没有~/.bash\_profile文件或文件中没有执行~/.bashrc文件

可以在~/.bash\_profile中加入命令 source ~/.bashrc 后保存

换了oh-my-zsh后，需要更新的是vim ~/.zshrc，然后source ~/.bashrc

**history**

-c 清空历史记录

-w 把缓存中的历史记录存入文件中~/.bash\_history，一般登出时才把缓存保存入文件

调用历史记录

上下箭头

!n 第n条记录

!! 重复执行上一次命令

!字符串 重复执行最后一条以弓字符串空头的命令

!$ 重复上一条命令的最后一个参数

**alias**  别名=’原命令’ upgrade=’sudo pacman -Syyu’

unalias 别名 unalias upgrade

命令执行级别

1、绝对路径或相对路径命令

2、别名

3、bash内部命令

4、按照$PATH环境变量定义的目录查找的第一条命令

如需永久有效配置~/.bashrc 如更换bash后执行其他相应命令，如~/.zshrc

**快捷键CTRL+**

**A 光标移动到开头**

**E 结尾**

**C 终止**

**L 清屏，相当于clear**

**U 删除或剪切光标前的命令**

K 删除或剪切光标后的命令

**Y 粘贴**

**R 搜索**

**D 退出当前终端**

Z 暂停，放入后台，少用

S 暂停屏幕输出

Q 回复屏幕输出

**重定向** >覆盖 >>追加

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

输出重定向 **命令 & >> file ls test & >> out.log**

命令 >> file 2> &1 ls >> out.log 2>&1

以追加模式把正确、错误输出都保存在一个文件中

可覆盖可追加

**命令 >>file1 2 >> file2**

把正确的输出追加到file1中，把错误的输出追加到file2

输入重定向 < <<

**wc**

-c 统计字节数

-w 统计单词数

-l 统计行数

wc -l <countnum.file

**多命令顺序执行**

; 无任何逻辑，不论报错与否都执行

&& 逻辑与，前面的命令正确才执行，$?返回0

|| 逻辑否，前面的命令错位才执行，$?返回非0

**grep** 行提取命令，只要一行数据包含搜索内容就会列出整行数据

-i 忽略大小

-v 反向查找、不包含 grep -v root /etc/passwd 在文件中显示不含root的行

-n 输出行号

grep “aria” /etc/passwd

find 用于搜索符合条件的文件名，完全匹配 通配符

grep 用于搜索符合条件的字符串，非完全匹配 正则表达式

**管道符 |** 连接多条命令，命令1**正确输出**作为命令2的输入

**通配符**

? 匹配一个任意字符

\* 匹配任何内容

[] 匹配中括号内任意一个字符，[abc]表示一定匹配其中一个，或a或b或c

[-] 匹配中括号内任意一个字符代表范围，[a-z]表示匹配一个小写英文字母

[^] 逻辑非，匹配不是中括号内的一个字符，[^0-9]匹配一个不是数字的字符，[^a-z]非字母

**Bash中的特殊符号**

‘’ 所有的特殊字符都无意义，如”$”、”`”、”\”。’$LANG’不调用变量，而是输出$LANG

“” 除”$”、”`”、”\”外的特殊字符无意义,”$UID”还是调用变量输出uid号

`` 系统命令，bash中先执行，易看错， echo `ls`

**$() 系统命令，bash中先执行，推荐使用，echo $(date)**

() 用于执行一串命令时，在子shell中执行(在当前终端开启新shell，新PID的shell)

{} 用于执行一串命令时，在当前shell中执行；也用于变量的调用

**[] 变量的测试**

# shell脚本中用于注释

**$ 调用变量的值，$DESKTOP\_SESSION**

\ 转义符，\之后的字符失去特殊作用，如\\输出\，\$输出$

**()与{}**

命令都在括号内，用;隔开

其内的某命令重定向只影响该命令，但括号外的重定向则影响括号内的所有命令

()最后一条命令可以不用; 各命令不必与括号有空格

{}最后一条命令必须要用; 第一条命令与左括号必须有空格

name=yuze

(name=aria;echo $name;date)

aria

2020年5月9日星期六 22:35

echo $name

yuze #在子shell执行只在子shell生效，命令结束回到父shell修改就会丢失

name=yuze

{ name=aria;echo $name;date }

aria

2020年5月9日星期六 22:36

echo $name

aria #在当前shell执行，变量的值已经被修改了

**变量**

用户自定义变量：最常见，本地变量、局部变量

环境变量：全局变量，保存与操作环境有关的数据

如登录用户、家目录、命令提示符、桌面样式、分辨率等

可自由定义(自己添加，但不能修改系统环境)

但对系统起作用的变量名由系统预先设定(但值可以自己改)

位置参数变量：向脚本传递参数或数据，变量名不能自定义，作用是固定的

预定义变量的一种

预定义变量：bash已经定义好的变量，变量名不能自定义，作用是固定的

**用户自定义变量** 只在当前中生效，子shell即不会生效

**变量赋值** 变量=值 变量两边无空格 name=aria

变量内有空格用”” age=”18 years old”

不能以数字开头 1name是错位的

**变量调用** $变量 $name

**变量叠加**  ${变量1}变量2 “$变量1”变量2 ${name}18y ”$name”” 18 year old”

**命令调用** $(命令) `命令` $(date) `ls -lh`

**变量查看**  set 查看所有变量 删除 unset

**环境变量**  全局变量，在当前和其所有子shell中生效;

如写入配置文件，所有shell中生效，若不，shell退出则无效

**声明环境变量** export name=aria

**查询环境变量** env

**PATH变量** 系统查找命令的路径，用”:”分开export $PATH=${PATH}:/dir

在~/.bashrc中添加PATH=${PATH}:/pycharm

**PS1变量：命令提示符设置**echo $PS1 [ \u@\h \w]\s [用户名@简写主机名]提示符

PS1=’[\u@\h \#条 \w]\$’ 变量必须用单引号’’包含

**LANG语系变量** locale 查看使用语系 /etc/locale.conf /etc/default/locale

linux中语系主要用变量设置，知道LANG和LC\_ALL即可，其他变量会依赖这两者的值而变化。LANG定义系统主语系，一般用此变量定义系统语系，LC\_ALL定义整体语系

**位置参数变量**  用于向命令或程序传递信息

$n n为数字，$0命令本身，$1-$9代表1-9个参数，10以上参数用大括号包含${10}

$\* 代表命令行中所有参数，作为一个**整体** ls a b c a b c作为一个整体

$@ 代表命令行中所有参数，每个参数**区别对待** ls a b c a b c作为三个参数

$# 命令行中所有参数的个数

**预定义参数**

$? 最厚执行命令的返回状态，返回0为真，命令正确执行，非0，命令执行错位

$$ 当前进程的PID ($PID) 可使用变量给临时文件命名

$! 后台运行的最后一个进程的PID

$- 使用set命令设定的flag一览

**read 选项 变量名 接收间盘输入**

-p “提示信息” 等待read输入时，输出的提示信息

-t 秒数 指定等待输入的时间

-n 字符数 只接收指定的字符数就会执行

-s 隐藏输入的数据，用于机密信息的输入

不提供变量存储到默认变量REPLY中

提供一个变量，将整个输入赋予该变量

提供多个变量，输入行分为若干字，一个一个的赋予各个变量，命令行上的最后一个变 量获得剩余的所有值

read -t 30 -p “please input your name:” name

**运算符**

**$((运算式))**  $[运算式]

c=$(( $a+$b )) c=$[ $a+$b ]

**运算式两侧必须有空格**

数值运算

declare [+-] 选项 变量名 使用declare声明变量类型

- 设定类型属性 -a 数组型

+ 取消类型属性 -i 整数型

declare -i c=$a+$b

数组型 编号成为下标 下标从0开始 ${变量[下标]}调用 用数组格式赋值系统即认为数组型

name[0]=”aria”

name[1]=”yiming”

name[3]=”yuze”

echo ${name} echo ${name[\*]} echo ${name[1]}

aria#只写数组名输出第1个 aria yiming yuze yiming

expr c=$(expr $a + $b) +左右必须有空格

let let c=$a+$b

**source** 配置文件 source ~/.bashrc

**.** 配置文件 . ~/.bashrc

强制执行脚本中的全部命令，忽略脚本权限

让重新配置的环境变量配置文件强制生效

**环境变量配置文件分类**

1. **登录时生效的**

**/etc/profile**

**/etc/profile.d/\*.sh**  一系列配置文件

~/.bashrc

~/bash\_profile 不一定有

**/etc/bash.bashrc**

粗体的三个文件对所有用户有效，home文件夹的文件对当前用户有效

**/etc/profile** ~/bash\_profile ~/.bashrc **/etc/bash.bashrc 命令提示符**

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

**/etc/locale.conf /etc/default/locale**

**/etc/profile**

定义umask默认权限

设置PATH变量

调用/etc/profile.d/\*.sh

重新配置全局bash设置

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

freetype2.sh TrueType字体解释版本

gawk.sh/.csh awk

locale.sh 1、在XDG路径中加载/etc/locale.conf

XDG(x desktop group) 基本目录规范，指定文件格式及所在的文件夹

$LANG

依次判断

上面的配置文件优先级高

$XDG\_CONFIG\_HOME/locale.conf

$HOME/.config/locale.conf

/etc/locale.conf

2、如未定义，则则定义默认LANG语系

3、当locale(7)变量存在时声明所有locale(7)变量

LANG LC\_TIME LC\_PAPER LC\_MEASUREMENT

LANGUAGE LC\_COLLATE LC\_NAME LC\_IDENTIFICATION

LC\_CTYPE LC\_MONETARY LC\_ADDRESS LC\_ALL

LC\_NUMERIC LC\_MESSAGE LC-TELEPHONE

snapd.sh snapcraft全新的软件环境，软件类似被封装到Docker容器中，由后台服务snapd提供支持

home-locale-bin.sh export PATH=”$HOME/.locale/bin:$PATH”

vte.sh/.csh Vitual Terminal Emulator，终端设置

$HOME $USER $HOSTNAME 提示符

perlbin.sh/.csh perl路径

gpm.sh genaral purpose mouse 通用鼠标

jre.sh/.csh java 指向/usr/lib/jvm/default/bin由archlinux-java助手脚本设置的连接位置

~/bash\_profile [[-f ~/.bashrc ]] && . ~/.bashrc 调用~/.bashrc

~/.bashrc 定义用户及root终端PS1及PS1、ls、grep等命令颜色

改变终端标题、字符：user@hostname:路径

定义别名alias

complete 命令补全

shopt显示和设置shell的行为选项

-s 激活指定的shell行为

-u 关闭指定的shell行为

eg.histappend

expand\_aliasescdspell

cdspell

QT\_SELECT

**ex解包（支持常用包如tar.gztar.bz gz bz2 tar zip z 7z rar tgz tzb2等**

**ex test.tar.gz 相当于 tar -zxvf test.tar.gz**

**可自己再定义一个打包的函数命令**

**/etc/bash.bashrc** PS1变量，用户提示符

1. 注销时生效

/etc/bash.bash\_logout 无内容，退出时执行的一些操作，比如清除垃圾文件，备份数据可写里面

~/.bash\_logout 同上

1. 配置文件小总结

**/etc目录下**

profile

profile.d/\*.sh

bash.bashrc

bash.bash\_logout

bash\_completion.d/\*

**用户目录下**

.bashrc

.bash\_profile

.bash\_logout

.bash\_history 历史命令保存文件

**shell登录信息**

1. **/etc/issue** tty终端登录前的欢迎界面

aria’s linux

happy up happy day

\d日期 \t时间

\r内核版本 (\n)主机名 @ \u登录用户序列号 (\l)登陆的终端号

Wed May 12 2020 21:43:52

5.6.8-1-MANJARO

aria 1 @ (tty2)

1. /etc/motd message of the day 终端登陆后的信息，是否可执行命令fortune
2. /etc/issue.net 远程登录如ssh的欢迎界面，需要配置/eyc/ssh/sshd\_config，加入Banner /etc/issue.net

**自定义bash快捷键**

**stty** stty 关键字 快捷键

-a 显示所有快捷键

stty intr ^P CTRL+P 强制终止 ^人工输入即可

**语系与桌面系统配置文件**

**配置文件**

1、 /etc/locale.conf 全局有效，不推荐，tty下乱码，配置简单，其实tty下中文本身就乱码

2、 ~/.bashrc 终端登录读取并应用

~/.xinitrc statx界面时

~/.xprofile gdm(gnome)、kdm(kde)等wm图形管理器

LANG=”zh\_CN.UTF-8”

LANGUAGE=zh\_CN.en\_US”

LC-CTYPE=”en\_US.UTF-8”

3、 ~/.xprofile 图形界面全局expert LC\_ALL=”zh\_CN.UTF-8”

**中文字体**

下载ttf格式字体或在win下复制字体文件

cd /usr/share/fonts/

sudo cp ~/Downloads/\*.ttf /usr/share/fonts/

chmod +x \*.ttf

sudo mkfontscale #安装字体

sudo mkfontdir

sudo fc-cache   -fv

sudo pacman -S --noconfirm powerline-fonts awesome-terminal-fonts wqy-microhei wqy-zenhei ttf-fira-code wqy-microhei-lite wqy-bitmapfont ttf-arphic-ukai ttf-arphic-uming adobe-source-han-sans-cn-fonts adobe-source-han-serif-cn-fonts noto-fonts-cjk && fc-cache -fv

/etc/fonts/conf.d 目录下是全局配置，软连接到/etc/fonts/conf.avail目录下文件

/usr/share/fonts 字体目录

~/font.conf 用户字体配置文件

**中文输入法**

**Rime中州中州韵挂载小鹤双拼**

sudo pacman -S fcitx-im #安装fcitx输入法框架

sudo pacman -S fcitx-configtool #输入法配置文件

sudo pacman -S fcitx-rime #安装rime

fcitx-qt4与fcitx-im部兼容，只能安装一个，安装im，qt4删除了键盘-布局

**添加输入法配置文件 (sudo vim ~/.xprofile)**

export GTK\_IM\_MODULE=fcitx

export QT\_IM\_MODULE=fcitx

export XMODIFIERS="@im=fcitx"

重启后输入法生效

图形界面中或使用远程连接工具Xshell等工具，只要正确设置了语系即可正确显示中文

纯字符界面(tty1-6)中文乱码，需安装fbterm zhcon(两者已停止开发了)

**shell编程**

**基础正则表达式**

\* **前**一个字符匹配任意多次,a\*无意义，匹配所有内容aa\*匹配至少含一个a的行

. 匹配一个字符,一个.匹配所有内容，s.\*d匹配sd之间任意多字符

^ 匹配行首^aria匹配aria开头的行^A N$  **^$匹配空行**

$ 匹配行首^aria匹配aria开头的行

[] 匹配任意一个

[^] 匹配[]中字符以外的任意一个字符

\ 转义符取消特殊字符的含义 \.$匹配以.结尾的行

\{n\} 前面的字符恰好出现n次，[0-9]\{4\}匹配四位数字

\{n,\} 前面的字符出现n次以上，[0-9]\{4,\}匹配四位及以上数字

\{n,m\} 前面的字符出现n-m次，[0-9]\{4,\}匹配4-8位数字

**扩展正则表达式**

+ 前一个字符匹配一次或任意多次 go+gle匹配gogle google go……gle

? 前一个字符匹配0或1次 colou?r 匹配colour color

| 匹配两个或多分支选择 xterm|uvrxt|terminator 匹配包含以上三者的行

() 匹配其整体为一个字符 hello(world|aria) 匹配hello world和hello aria

**字符截取与命令替换 cut awk sed grep 后三者称三剑客**

**cut** 列提取命令 **默认制表符,对空格识别不好，但是简单易用**

-f 提取第几行

-d 按指定分隔符分隔列

-c 不依赖分隔符，而通过字符范围(首行为0)进行提取

n- n个字符到行尾 -m 行首到m个字符 n-m n到m个字符

cut -f 5 `df` cut -d “:” /etc/passwd

cut -f 1,5 `df` cut -c 2-8 file

cut -f 1-4 `df`

**awk编程**

**printf** 格式化输出printf ‘输出类型格式’ 输出内容 不自动换行而print自动换行

输出类型 %ns 输出字符串 %5s

%ni 输出整数 %4i

%m.nf 输出浮点数 %4.2f 12.34 四位数，两位为小数

控制字符 见p22

\n 换行符

\t 制表符，TAB键

\r 回车键

printf ‘%s\t %i\t %4.2f \t \n’ $(cat student|grep -v name)

**awk** awk ’条件1 {动作1} 条件2 {动作2} ...’ 文件

**条件pattern**

语句块 BEGIN 程序开始尚未读取数据之前执行一次

PATTERN 主要内容，就是条件

END 程序执行完所有数据，结束时执行一次

以上三者语句块都可无

运算 <、>、<=、>=、==、!=

A~B判断字符串A是否包含匹配B的子字符串、A!~B

正则 /正则/

内置变量 $0 $n，分别代表整体数据、第n字段

NF 字段数

NR 行数

FS 指定分隔符

**动作** 格式化输出 流程控制语句 变量或数值赋值 内置函数

awkv只要不检测到完整的单引号就不会执行，所以换行加上”\n”

只要分隔开不管是空格还是制表符都识别 **默认任意空格**

awk只识别、查找//内的字符

有BEGIN执行定义动作

无BEGIN，读入第1行，赋予$0、$n等变量

依据条件判断动作是否执行；符合则执行，否则读入下一行，无条件则每行都执行动作

读入下一行重复以上步骤

**df |grep sda2|awk ‘{printf “$6”}’|cut -d “&” -f 1 获取磁盘利用率不要%**

awk ‘BEGIN{printf “This is BEGIN \n”}{printf”END \n”}{printf $2 “\t” $6 “\n”} student 打印2、6字段

This is BEGIN

aria 95.65

END

cat student | grep -v name | awk ’$6 >= 90 {printf $2 “\n”}’

awk ‘$2 ~ /aria/ {printf $6 “\n”}’ student 95.65

awk ‘/aria/ {printf $6 “\n”}’ student 1 aria 82 95 89 95.65

cat /etc/passwd | grep /bin/bash | awk ‘BEGIN{FS=:} {printf $1 “\t” $3 “t\” 行\号:NR“\t”字段数:NF”\n”}’ 命令太长 用\回车，取消特殊输出来换行，命令继续输出

root 0 行号:1 字段数:7 root:x:0:0::/root:/bin/bash

**awk流程控制**

awk'NR==2{php1=$3}  
NR==3{php2=$3}  
NR==4{php3=$3;totle=php1+php2+php3;print "totle php is " totle}' student.txt  
#统计PHP成绩的总分

“NR==4{php3=$3;totle=php1+php2+php3;print"totle phpis"totle}”

（“NR==4”是条件，后面{}中的都是动作） 是指如果输入数据是第四行， 就把第四行的第三个字段的值赋予变量“php3”； 然后定义变量totle的值是“php1+php2+php3”； 最后输出“totle php is”关键字， 后面加变量totle的值。

多个条件{动作}可以用空格分隔， 也可以用回车分隔。  
在一个动作中， 如果需要执行多条命令，则需要用“;”分隔， 或用回车分隔。  
在awk中， 变量的赋值与调用都不需要加入“$”符号。  
在条件中判断两个值是否相同， 请使用“==”， 以便和变量赋值进行区分。

awk自带条件可判断

awk'{if (NR>=2)  
{if ($4>90) printf $2 " is a good man!\n"}}' student.txt

awk' NR>=2 {test=$4}  
test>90 {printf $2 " is a good man!\n"}' student.txt

**awk函数**

function 函数名（参数列表） {  
函数体  
}

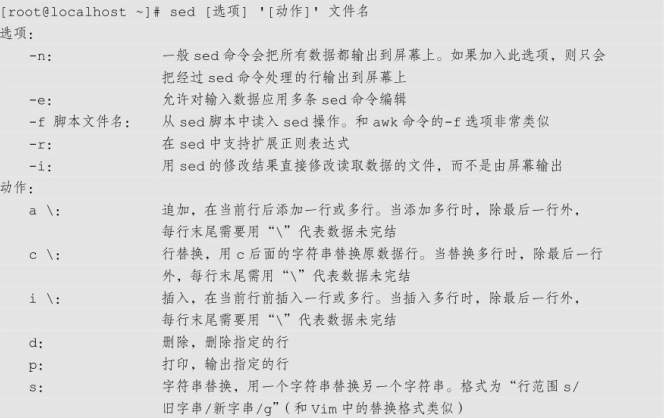
awk 'function test(a,b) { printf a "\t" b "\n" }  
#定义函数test， 包含两个参数， 函数体的内容是输出这两个参数的值  
{ test($2,$6) } ' student.txt

**awk中的外部脚本调用**

vim pass.awk  
BEGIN {FS=":"}  
{ print $1 "\t" $3}

然后使用“-f”选项来调用这个脚本  
awk -f pass.awk /etc/passwd

**sed 流编辑器数据 增删改查 主要变成用**



sed所做的修改并不会直接改变文件的内容（如果是用管道符接收的命令的输出， 则连文件都没有） ,而是把修改结果只显示到屏幕上， 除非使用“-i”选项才会直接修改文件。

所有的动作必须使用“单引号”包含； 其次，在动作中可以使用数字代表行号， 逗号代表连续的行范围。还可以使用“$”代表最后一行，如果动作是“2,$d”，则代表从第二行删除到最后一行。

s前不加数字表示全文

sed -n '2p' student.txt 显示

sed '2,4d' student.txt 删除，实际文本未修改

sed '2a hello' student.txt 追加

sed -n'2i hello \ 插入多行，\前必须有空格

world' student.txt  
hello  
world

sed '3s/74/99/g' student.txt 字符替换

sed -e 's/Liming//g ; s/Gao//g' student.txt 多个动作

**字符处理命令**

**sort 排序 选项少，默认即可**

-f 忽略大小写

-b 忽略每行前面的空白部分

**-n 以数值型进行排序， 默认使用字符串型排序**

**-r 反向排序**

-u 删除重复行。 就是uniq命令

**-t 指定分隔符， 默认分隔符是制表符**

**-k n[,m] 按照指定的字段范围排序。 从第n个字段开始， 到第m个字段结束（默认到行尾）**

**sort命令默认是用每行开头的第一个字符来进行排序的**

“-k”选项可以直接使用“-k3”，代表从第三个字段到行尾都排序（第一个字段先排序，如果一致，则第二个字段再排序，直到行尾）

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

root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin  
uucp:x:**10**:14:uucp:/var/spool/uucp:/sbin/nologin  
operator:x:11:0:operator:/root:/sbin/nologin  
abrt:x:173:173::/etc/abrt:/sbin/nologin  
daemon:x:**2**:2:daemon:/sbin:/sbin/nologin

因为sort默认是按照**字符**排序的， 前面用户的UID的第一个字符都是1， 所以这么排序，而不是**数字**

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

root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin  
daemon:x:2:2:daemon:/sbin:/sbin/nologin  
adm:x:3:4:adm:/var/adm:/sbin/nologin  
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

**uniq** 取消重复行的命令， 其实和“sort-u”选项是一样的

-i: 忽略大小写

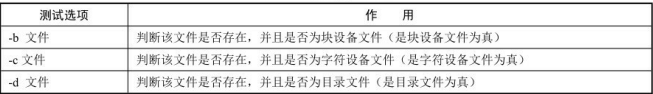
**wc** 统计命令

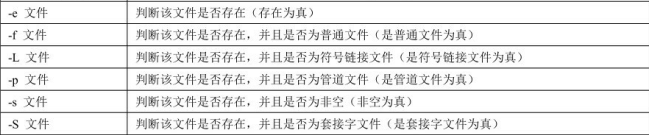
无参数 都统计

-l 只统计行数  
-w 只统计单词数  
-m 只统计字符数

**条件判断**

test 推荐使用 **[]** 方式 “[]”的内部和数据之间必须使用空格

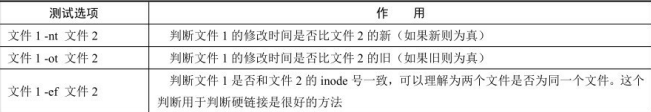
**文件类型判断**



**文件权限判断**



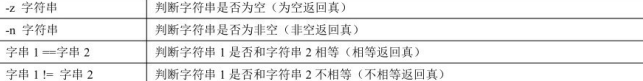
**两个文件对比**



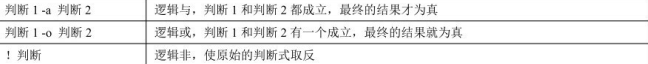
**两个整数对比 用数学符号也可以，如ge等同于>=**



**字符串判断**



**多重判断**



**流程控制**

条件判断控制（if、 case）   
循环控制（for、 while、 until）   
特殊流程控制语句（exit、 bread、 continue）

**if条件判断**

**单分支if条件语句** 只有一个判断条件，符合条件则执行某个程序，否则不做

If [ 条件判断式 ];then  
程序  
fi

if [ 条件判断式 ]  
then

程序

fi

[ 条件判断式 ]就是使用test命令判断，中括号和条件判断式之间必须有空格  
then后面跟符合条件之后执行的程序，可以放在[]之后，用“;”分隔；也可以换行写入，就不需要“;”了

#!/bin/bash  
#统计根分区的使用率  
rate=$(df -h | grep "/dev/sda3" | awk '{print $5}' | cut -d "%" -f1)  
#把根分区使用率作为变量值赋予变量rate  
if [ $rate -ge 80 ]  
then  
echo "Warning! /dev/sda3 is full!!"

#打印警告信息。 在实际工作中，也可以向管理员发送邮件  
fi

**双分支if条件语句** 当条件判断式成立时， 则执行某个程序；

当条件判断式不成立时， 则执行另一个程序

if [ 条件判断式 ]  
then  
 当条件判断式成立时， 执行的程序  
else  
 当条件判断式不成立时， 执行的另一个程序  
fi

#!/bin/bash  
ntpdate asia.pool.ntp.org &>/dev/null #同步系统时间  
date=$(date +%y%m%d)  
size=$(du -sh /home/aria/linux)  
if [ -d /tmp/dbbak ]  
then  
 echo "Date : $date!" > /tmp/dbbak/dbinfo.txt  
 echo "Data size : $size" >> /tmp/dbbak/dbinfo.txt  
 cd /tmp/dbbak  
 tar -zcf mysql-lib-$date.tar.gz / home/aria/linux dbinfo.txt &>/dev/null  
 #打包压缩数据库与临时文件,把所有输出丢入垃圾箱不看任何输出  
 rm -rf /tmp/dbbak/dbinfo.txt  
else  
 mkdir /tmp/dbbak  
 echo "Date : $date!" > /tmp/dbbak/dbinfo.txt  
 echo "Data size : $size" >> /tmp/dbbak/dbinfo.txt  
 cd /tmp/dbbak  
 tar -zcf mysql-lib-$date.tar.gz dbinfo.txt /var/lib/mysql &>/dev/null  
 rm -rf /tmp/dbbak/dbinfo.txt  
fi

**多分支if条件语句** 当条件判断式1成立时，执行程序1；

当条件判断式2成立时，执行程序2,次类推;

当所有条件都不成立时， 执行最后的程序

if [ 条件判断式1 ]  
 then  
 断式1成立时， 执行程序1  
elif [ 条件判断式2 ]  
 then  
 断式2成立时， 执行程序2  
…省略更多条件…  
else  
 当所有条件都不成立时， 最后执行此程序  
fi

#!/bin/bash  
read -p "Please input a filename: " file  
if [ -z "$file" ]  
 then  
 echo "Error,please input a filename"  
 exit 1# 退出程序，并定义返回值为1（把返回值赋予变量$?）  
elif [ ! -e "$file" ]  
 then  
 echo "Your input is not a file!"  
 exit 2  
elif [ -f "$file" ]  
 then  
 echo "$file is a regulare file!"  
elif [ -d "$file" ]  
 then  
 echo "$file is a directory!"  
else  
 echo "$file is an other file!"  
fi

**多分支case条件语句**

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

多分支case条件语句只能判断变量中的值到底是什么，更加适合**单条件多分支**的情况

如请选择“yes/no”；或在命令的输出中选择是执行第一个选项，还是执行第二个选项

而多分支if条件语句那样，可以判断多个条件

case语句会取出变量中的值，然后与语句体中的值逐一比较。

如果数值符合，则执行对应程序；

如果数值不符，则依次比较下一个值；

如果所有的值都不符合，则执行“\*)” （“\*”代表所有其他值）中的程序。

在每个分支程序之后要以“;;”（双分号） 结尾，代表该程序段结束

#!/bin/bash

read -p “please input choose yes\no:” -t 30 choose

case $choose in

“yes”)

echo “yes”

;;

“no”)

echo “no”

;;

“\*”)

echo “error”

;;

esac

**for循环**

**语法一：**for 变量 in 值1 值2 值3…  
 do  
 程序  
 done

在这种语法中，for**循环的次数取决于in后面值的个数**（以空格分隔），有几个值就循环几次，并且每次循环都把值赋予变量。

也就是说，假设in后面有三个值，for会循环三次，第一次循环会把值1赋予变量， 第二次循环会把值2赋予变量，以此类推。

for i in `seq 1 15` #显示1-15的值

do

echo $i

done

#!/bin/bash  
cd /aria  
ls \*.tar.gz > ls.log  
for i in $(cat ls.log)

do  
tar -zxf $i &>/dev/null

done  
rm -rf /aria/ls.log

**语法二：**for (( 初始值；循环控制条件；变量变化 ))  
 do  
 程序  
 done

在语法二中需要注意以下几点。

**初始值**： 在循环开始时，需要给某个变量赋予初始值， 如i=1。

**循环控制条件**：指定变量循环的次数，如i<=100，则i的值小于等于100，循环就会继续

**变量变化**：每次循环后，变量该如何变化，如i=i+1，代表每次循环后，变量i的值都加1

#!/bin/bash  
user=$(cat /etc/passwd | grep "/bin/bash"|grep -v "root"|cut -d ":" -f 1)

#提取可以登录的用户，取消root用户，截取第一列用户名

for i in $user  
do  
 userdel -r $i  
done

**while循环** 不定循环， 也称作条件循环

主要是指循环可以一直进行，直到用户设定的条件不成立

while [ 条件判断式 ]  
do  
 程序  
done

从1加到100两种循环的写法，两者类似

#!/bin/bash  
i=1  
s=0  
while [ $i **-le** 100 ] #如果变量i的值小于等于100，则执行循环“le等于<=”  
 do  
 s=$(( $s+$i ))  
 i=$(( $i+1 ))  
 done  
echo "The sum is: $s"

#!/bin/bash  
s=0  
for (( i=1;i**<=**100;i=i+1 ))  
 do  
 s=$(( $s+$i ))  
 done  
echo "The sum of 1+2+...+100 is : $s"

**until循环** 不定循环， 也称作条件循环，和while循环相反

只要条件判断式不成立，则进行循环，并执行循环程序；

一旦条件判断式成立，则中止循环

until [ 条件判断式 ]  
do  
 程序  
done

#!/bin/bash

i=1  
s=0  
until [ $i **-gt** 100 ] #循环，直到变量i的值大于100， 就停止循环  
 do  
 s=$(( $s+$i ))  
 i=$(( $i+1 ))  
 done  
echo "The sum is: $s"

**函数**

自己定义的程序集合，当我们在某个脚本中需要重复使用同一项功能时，就可以先把这项功能定义为一个函数，在每次使用时只要调用此函数即可

function 函数名 () {  
程序

}

#!/bin/bash

function sum {

s=0  
for (( i=1;i**<=**100;i=i+1 )) #循环直到i大于**$1**为止，$1是函数sum的第一个参数(控制条件)  
 do  
 s=$(( $s+$i ))  
 done  
echo "The sum of 1+2+...+100 is : $s"

}

read -p “please input a num:” -t 30 num

y=${echo $num |sed ‘s/[0-9]//g’} #把变量num的值替换为空，并赋予变量y，确保为数字

if [ -z “$y”]

then

sum $num #调用sum函数，并把变量num作为sum函数的第一个参数

else

echo “error,please input a num”

fi

函数也有自己的位置参数变量，$0代表函数名，$1代表函数的第一个参数，$2代表函数的第二个参数，以此类推。

**特殊流程控制语句**

**exit [返回值] exit 8**

系统中是有exit命令的，用于退出当前用户的登录状态。

但是在Shell脚本中，exit语句是用来退出当前脚本的

即碰到exit语句，·

如果exit未定义返回值，则脚本执行后的返回值是执行exit语句之前最后执行命令的返回值。

**break语句 continue语句**

当程序执行到break语句时，会结束整个当前循环

continue语句也是结束循环的语句，但只会结束单次当前循环，而下次循环会继续

for (( 条件判断语句 ))

do

程序语句

break

跳出循环

程序语句

done

for (( 条件判断语句 ))

跳出当前循环

do

程序语句

continue

程序语句

done

s=0  
for(( i=1;i**<6**;i=i+1 ))

do

if [ “$i” -eq 4 ]  
 then

break

fi

echo " $s"

done

s=0  
for(( i=1;i**<=**6;i=i+1 ))

do

if [ “$i” -eq 4 ]  
 then

continue

fi

echo " $s"

done

运行

1

2

3

运行

1

2

3

5

6

**daemon与服务service**

系统为了某些功能必须要提供一些服务（不论是系统本身还是网络方面），称为 service

但service 的提供需要程序的运行，达成这个 service 的程序称为daemon

达成循环型例行性工作调度服务（service）cron的程序为 crond 这个daemon

daemon一般名称为服务名加d

**早期 System V 的 init 管理行为中 daemon 的主要分类**

系统核心第一支调用的程序是 init

然后init根据自订的执行等级（runlevel）来唤醒不同的服务，以进入不同的操作界面

**服务的启动、关闭与观察等方式：** 所有的服务启动脚本通通放置于 /etc/init.d/ 下面，基本上用shell script 写的脚本

启动：/etc/init.d/daemon start  
 关闭：/etc/init.d/daemon stop  
 重新启动：/etc/init.d/daemon restart  
 状态观察：/etc/init.d/daemon status

**服务启动的分类：** 独立启动模式 （stand alone）：服务**独立启动**，常驻于内存，提供本机或用户的 服务行为，反应速度快。  
 总管程序 （super daemon）：由特殊的 xinetd 或 inetd 这两个**总管程序**提供 socket 对应或 port 对应的管理。

当没有用户要求某 socket 或 port 时，所需要的服务是不会被启动的。

若有用户要求时， xinetd 总管才会去唤醒相对应的服务程序。

当该要求结束时，这个服务也会被结束掉。

因为通过 xinetd 所总管，因此这个家伙就被称为 super daemon。

好处是可以通过 super daemon 来进行服务的时程、连线需求等的控制。

缺点是唤醒服务需要一点时间的延迟。

**服务的相依性问题：** 你要启动网络服务，需要接入网络，才能唤醒网络服务

init 在管理员自己手动处理这些服务时，是没有办法协助相依服务的唤醒的

**执行等级的分类：** 1）单人维护模式、3）纯文本模式、5）文字加图形界面。

各个执行等级的启动脚本是通过/etc/rc.d/rc[0-6]/SXXdaemon链接到/etc/init.d/daemon，链接文件名（SXXdaemon）的功能为：S为启动该服务，XX是数字，为启动的顺序。

由于有SXX的设置，因此在开机时可以“依序执行”所有需要的服务，同时也能解决相依服务的问题。这点与管理员自己手动处理不太一样就是了。

**制定执行等级默认要启动的服务：** 若要创建如上提到的 SXXdaemon 的话，不需要管理员手动创建链接文件，通过如下的指令可以来处理默认启动、默认不启动、观察默认启动否的行为：  
 默认要启动： chkconfig daemon on  
 默认不启动： chkconfig daemon off  
 观察默认为启动否： chkconfig --list daemon

**执行等级的切换行为：** init 5 init会主动去分析 /etc/rc.d/rc[35].d/这两个目录内的脚本，然后启动转换 runlevel 中需要的服务。就完成整体从纯命令行（runlevel 3）切换到图形界面（runlevel 5）

**systemd**

**平行处理所有服务，加速开机流程：** init 启动脚本是“一项一项依序启动”的模式，不相依的服务也要一个一个的等待

目前我们的硬件主机系统与操作系统几乎都支持多核心架构了，支持多线程

systemd 可以让所有的服务同时启动，系统启动的速度变快

**一经要求就回应的 on-demand 启动方式：** systemd仅有一只systemd服务搭配systemctl指令来处理，无须额外指令

systemV 还要 init,chkconfig, service... 等等指令

systemd常驻内存，任何要求on-demand都可以立即处理后续daemon启动的任务

**服务相依性的自我检查：** systemd 会自动分析处理相依性并自动启动依赖的服务

**依 daemon 功能分类：** systemd 先定义所有的服务为一个服务单位unit并归类到不同的服务类型type

如service, socket, target, path, snapshot, timer

旧的 init 仅分为 stand alone 与 super daemon

**将多个 daemons 集合成为一个群组：** 如同 systemV 的 init 里头有个 runlevel 的特色

systemd 将许多daemon功能集合成一个target，主要在设计操作环境的创建

**向下相容旧有的 init 服务脚本：** systemd 是可以相容于 init 的启动脚本的

不过systemd也是有些地方无法完全取代 init 的

仅有 runlevel 1, 3, 5 有对应到 systemd 的某些 target 类型而已

全部的systemd都用systemctl这个管理程序管理，而systemctl支持的语法有限制，不像 /etc/init.d/daemon 就是纯脚本可以自订参数，systemctl 不可自订参数

如果服务是管理员自己手动执行启动，

例如你自己手动输入 crond启动crond服务， systemd 将无法侦测并管理该服务

systemd 启动过程中，无法与管理员通过standard input传入讯息

自行撰写 systemd 的启动设置时，务必要取消互动机制，连通过启动时传进的标 准输入讯息也要避免

**与system运行相关的目录**

**/usr/lib/systemd/system/：** 官方带每个服务**默认的、最主要的**启动脚本配置**实际文件**

类似/etc/init.d 下面的文件，优先序最低

**/run/systemd/system/：** 系统**执行过程中**产生的服务脚本，优先序稍高  
 **/etc/systemd/system/：** 依据主机系统的需求**管理员所创建**的执行脚本,优先序最高

类似 /etc/rc.d/rc5.d/Sxx 之类的功能，都是**链接文件**

**尽量在/etc/systemd/system下面修改配置**

/etc/sysconfig/\*： 几乎所有服务都会将初始化的一些**选项设置**写入到此目录

mandb要更新的manpage索引要加的参数写入到此目录下的man-db

网络设置则写在network-scripts/ 这个目录内

/var/lib/： 一些会产生数据的服务会将数据写入到/var/lib/目录中

Mariadb的数据库默认写入/var/lib/mysql/

/run/： 放置许多daemon的暂存盘，如lock file以及PID file

系统开机会不会执行某些服务看/etc/systemd/system/ 下面的设置，但仅是链接到/usr/lib/systemd/system/的执行脚本配置文件。

实际执行的systemd启动脚本配置文件放置在 /usr/lib/systemd/system/ 下面，在此修改查阅配置文件。

systemctl list-sockets 查看socket 服务，知道正在监听本机服务需求的socket file位置

**systemd的unit类型说明**

|  |  |
| --- | --- |
| 扩展名 | 主要服务功能 |
| .service | **一般服务类型service unit**：  主要是系统服务，包括服务器本身所需要的本机服务以及网络服务  **最常见** |
| .socket | **内部程序数据交换的插槽服务socket unit**：**随用随启动**  主要是IPC（Inter-process communication）传输讯息插槽档（socket file）功能  通常在监控讯息传递的插槽档，当有通过此插槽档传递讯息来说要链接服务时，就依据当时的状态将该用户的要求传送到对应的daemon，若 daemon 尚未启动，则启动该daemon 后再传送用户的要求。  一般是不常用到的服务，因此在开机时通常会稍微延迟启动的时间。  **一般用于本机服务比较多**，例如我们的图形界面很多的软件都是通过 socket 来进行本机程序数据交换的行为。  与早期的xinetd这个super daemon相似 |
| .target | **执行环境类型target unit：是一群 unit 的集合**  如multi-user.target 其实就是一堆服务的集合  选择执行multi-user.target就是执行一堆其他.service或/及.socket 之类服务 |
| .mount .automount | **文件系统挂载相关的服务automount unit / mount unit：**  来自网络的自动挂载、NFS文件系统挂载等与文件系统相关性较高的程序管理 |
| .path | **侦测特定文件或目录类型path unit**：需要侦测特定的目录来提供伫列服务  最常见的打印服务通过侦测打印伫列目录来启动打印功能，需要.path 的服支持 |
| .timer | **循环执行的服务timer unit：类似anacrontab**  由systemd主动提供的，比anacrontab 更加有弹性 |

**通过systemctl管理服务**

systemd这个启动服务的机制，通过仅有的systemctl指令来处理

**管理单一服务的启停、开机运行、状态观察**

**start ：立刻启动后面接的 unit**

stop ：立刻关闭后面接的 unit

restart ：立刻关闭后启动后面接的 unit，亦即执行 stop 再 start 的意思

reload ：不关闭**后面接的unit**的情况下，重新载入配置文件，让设置生效

daemon-reload：后面不接参数，重新产生、加载所有系统服务配置

**enable ：设置下次开机时，后面接的 unit 会被启动**

disable ：设置下次开机时，后面接的 unit 不会被启动

**status ：后面接的这个unit的详细状态，是否正在执行、开机默认执行、登录**

**show : 显示默认设置**

is-active ：目前有没有正在运行中

is-enable ：开机时有没有默认要启用这个 unit

systemctl status atd.service  
atd.service - Job spooling tools  
**Loaded: loaded （/usr/lib/systemd/system/atd.service; enabled）  
Active: active （running） since Mon 2015-08-10 19:17:09 CST; 5h 42min ago**Main PID: 1350 （atd）  
CGroup: /system.slice/atd.service  
└─  
1350 /usr/sbin/atd -f  
Aug 10 19:17:09 study.centos.vbird systemd[1]: Started Job spooling tools.  
  
**Loaded：这行在说明开机是否自启动**

enabled 这个 daemon 将在开机时被执行  
 disabled 这个 daemon 在开机时不会被执行  
 static 这个daemon不可自启动(enable不可)

不过可能会被其他enabled的服务来唤醒（相依属性的服务)

mask 这个daemon无论如何都无法被启动,已被强制注销(非删除)，

**强力disable**可通过 systemctl unmask 方式改回原本状态

**Active：现在daemon的状态**

active(running) 正有一只或多只程序正在系统中执行的意思

active(exited) 仅执行一次就正常结束的服务,目前并没有任何程序在系统中执行。 如开机或者是挂载时才会进行一次的 quotaon 功能，只须执行一次之后，就交给文件系 统去自行处理，通畅shell写的小型服务，大多是属于这种类型（无须常驻内存）。  
 active(waiting) 正在执行当中，等待其他的事件才能继续处理,如打印伫列相关服务  
 inactive(dead) 这个服务目前没有运行

后面几行说明这个程序的PID 状态以及最后一行显示这个服务的**登录文件信息**  
登录文件信息格式为：时间、讯息发送主机、哪一个服务的讯息、实际讯息内容

**不应该用kill的方式来关掉一个正常的服务，systemctl无法继续监控该服务**

**关闭服务实践**

1.先看看 cups 的服务是开还是关？  
systemctl status cups.service  
cups.service - CUPS Printing Service  
Loaded: loaded （/usr/lib/systemd/system/cups.service; enabled）  
Active: inactive （dead） since Tue 2015-08-11 19:19:20 CST; 3h 29min ago  
#有趣得很！竟然是 enable 但是却是 inactive

2. 那就直接关闭，同时确认没有启动  
systemctl stop cups.service  
systemctl disable cups.service  
rm '/etc/systemd/system/multi-user.target.wants/cups.path'  
rm '/etc/systemd/system/sockets.target.wants/cups.socket'  
rm '/etc/systemd/system/printer.target.wants/cups.service'  
# 消掉三个链接文件！也就是说，这三个文件可能是有相依性的问题

netstat -tlunp | grep cups  
# 现在应该不会出现任何数据！没有cups的任务在执行当,所以不会有port产生

3. 尝试启动 cups.socket 监听用户端的需求  
systemctl start cups.socket  
systemctl status cups.service cups.socket cups.path  
cups.service - CUPS Printing Service  
Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）  
Active: inactive （dead） since Tue 2015-08-11 22:57:50 CST; 3min 41s ago  
cups.socket - CUPS Printing Service Sockets  
Loaded: loaded （/usr/lib/systemd/system/cups.socket; disabled）  
Active: active （listening） since Tue 2015-08-11 22:56:14 CST; 5min ago  
cups.path - CUPS Printer Service Spool  
Loaded: loaded （/usr/lib/systemd/system/cups.path; disabled）  
Active: inactive （dead） # 确定仅有 cups.socket 在启动，其他的并没有启动的状态！

4. 尝试使用 lp 这个指令来打印看看？  
echo "testing" | lp  
lp: Error - no default destination available. #实际上就是没有打印机！所以有错误也没关系

systemctl status cups.service  
cups.service - CUPS Printing Service  
Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）  
Active: active （running） since Tue 2015-08-11 23:03:18 CST; 34s ago

netstat -tlunp | grep cups  
tcp 0 0 127.0.0.1:631 0.0.0.0:\* LISTEN 25881/cupsd  
tcp6 0 0 ::1:631 :::\* LISTEN 25881/cupsd  
# cups自动被启动了

打印服务会启用port 631来提供网络打印机的打印功能,但无须一直启动631端口,因此多了一个名为 cups.socket 的服务，这个服务可以在用户有需要打印时，才会主动唤醒cups.service

如果仅disable/stop cups.service 而忘记了其他两个服务的话，那么当有用户向其他两个 cups.path, cups.socket 提出要求时，cups.service就会被唤醒

正规是要关闭cups.service时连同两个会唤醒service的cups.socket与cups.path都关闭

不正规的作法mask cups.service 注销服务

保持刚刚的状态，关闭cups.service，启动cups.socket，然后注销cups.servcie

systemctl stop cups.service  
systemctl mask cups.service  
ln -s '/dev/null' '/etc/systemd/system/cups.service'  
#其实这个mask注销的动作，只是让启动的脚本变成空的设备而已

systemctl status cups.service  
cups.service  
Loaded: masked （/dev/null）  
Active: inactive （dead） since Tue 2015-08-11 23:14:16 CST; 52s ago

systemctl start cups.service  
Failed to issue method call: Unit cups.service is masked. # 再也无法唤醒

通过mask，可以不必管其他相依服务可能会启动到这个想要关闭的服务了，非正规但有效

systemctl unmask cups.service #取消注销   
rm '/etc/systemd/system/cups.service'

systemctl status cups.service  
cups.service - CUPS Printing Service  
Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）  
Active: inactive （dead） since Tue 2015-08-11 23:14:16 CST; 4min 35s ago

**通过system查看系统上所有服务**

**systemctl [command] [--type=TYPE] [--all]  
 list-units ：依据unit列出目前有启动的unit。--all会列出没启动的。  
 list-unit-files ：依据 /usr/lib/systemd/system/内的文件，将所有文件列表说明。  
 --type=TYPE：unit type主要有service, socket, target等**

**systemctl 不加参数，其实默认就是 list-units**UNIT LOAD ACTIVE SUB DESCRIPTION  
proc-sys-fs-binfmt\_mis... loaded active waiting Arbitrary Executable File Formats File System  
sys-devices-pc...:0:1:... loaded active plugged QEMU\_HARDDISK  
sys-devices-pc...0:1-0... loaded active plugged QEMU\_HARDDISK  
sys-devices-pc...0:0-1... loaded active plugged QEMU\_DVD-ROM  
.....（中间省略）.....  
vsftpd.**service** loaded active running Vsftpd ftp daemon  
.....（中间省略）.....  
cups.socket loaded failed failed CUPS Printing Service Sockets  
.....（中间省略）.....  
LOAD = Reflects whether the unit definition was properly loaded.  
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.  
SUB = The low-level unit activation state, values depend on unit type.  
**141 loaded units listed.** Pass --all to see loaded but inactive units, too.  
To show all installed unit files use 'systemctl list-unit-files'.

# 列出的项目中，主要的意义是：  
# UNIT ：项目的名称、各个unit的类别（看扩展名）  
# LOAD ：开机时是否会被载入，默认 systemctl 显示的是**有载入**的项目而已喔！  
# ACTIVE ：目前的状态，须与后续的SUB搭配！用 systemctl status观察时active的项目  
# DESCRIPTION ：详细描述

systemctl list-unit-files

UNIT FILE STATE  
proc-sys-fs-binfmt\_misc.automount static  
dev-hugepages.mount static  
dev-mqueue.mount static  
proc-fs-nfsd.mount static  
.....（中间省略）.....

systemd-tmpfiles-clean.timer static

**336 unit files listed.**

systemctl list-units --type=service --all  
#只显示\*.service

systemctl list-units --type=service --all | grep cpu  
cpupower.service loaded inactive dead Configure CPU power related settings  
#确实有喔！可以改变CPU电源管理机制的服务

**管理system管理不同操作环境**

**systemctl list-units --type=target --all**

UNIT LOAD ACTIVE SUB DESCRIPTION  
 basic.target loaded active active Basic System  
 cryptsetup.target loaded active active Encrypted Volumes  
 emergency.target loaded inactive dead Emergency Mode  
 final.target loaded inactive dead Final Step  
 getty.target loaded active active Login Prompts  
 graphical.target loaded active active Graphical Interface  
 local-fs-pre.target loaded active active Local File Systems （Pre）  
 local-fs.target loaded active active Local File Systems  
 multi-user.target loaded active active Multi-User System  
 network-online.target loaded inactive dead Network is Online  
 network.target loaded active active Network  
 nss-user-lookup.target loaded inactive dead User and Group Name Lookups  
 paths.target loaded active active Paths  
 remote-fs-pre.target loaded active active Remote File Systems （Pre）  
 remote-fs.target loaded active active Remote File Systems  
 rescue.target loaded inactive dead Rescue Mode  
 shutdown.target loaded inactive dead Shutdown  
 slices.target loaded active active Slices  
 sockets.target loaded active active Sockets  
 sound.target loaded active active Sound Card  
 swap.target loaded active active Swap  
 sysinit.target loaded active active System Initialization  
 syslog.target not-found inactive dead syslog.target  
 time-sync.target loaded inactive dead System Time Synchronized  
 timers.target loaded active active Timers  
 umount.target loaded inactive dead Unmount All Filesystems  
 LOAD = Reflects whether the unit definition was properly loaded.  
 ACTIVE = The high-level unit activation state, i.e. generalization of SUB.  
 SUB = The low-level unit activation state, values depend on unit type.  
 **26 loaded units listed.** To show all installed unit files use 'systemctl list-unit-files'.

**跟操作界面相关性比较高的 target**

graphical.target：文字加上图形界面，这个项目已经包含了下面的multi-user.target项目！  
**multi-user.target**：纯文本模式  
rescue.target：救援模式。在无法使用root登陆的情况下，systemd 在开机时会多加一个额 外的暂时系统，与原本的系统无关。这时你可以取得root的权限来维护系统。 这是额外系统，因此可能需要动到 chroot 的方式来取得原有的系统  
emergency.target：紧急处理系统错误,需要使用root登陆的情况,无法用rescue.target时用  
shutdown.target：就是关机的流程。  
getty.target：设置需要的tty，如果想要降 tty的数目,可以修改这个东西的配置文件

**systemctl [command] [unit.target]**

**get-default ：取得目前的target**

**set-default ：设置后面接的target成为默认的操作模式**

**isolate ： 不重新开机的情况下,切换到后面接的模式**

systemctl poweroff 系统关机  
 systemctl reboot 重新开机  
 systemctl suspend 进入暂停模式  
 systemctl hibernate 进入休眠模式  
 systemctl rescue 强制进入救援模式  
 systemctl emergency 强制进入紧急救援模式

suspend：暂停模式会将系统状态数据保存到**内存中,**关闭掉大部分系统硬件,**并未实际关机**

当唤醒机器时数据在内存中恢复,重新驱动被关闭的硬件,正常运行,唤醒的速度快

hibernate：休眠模式将系统状态保存到**硬盘中**，保存完毕后，**将计算机关机**

唤醒系统时将保存在硬盘中的系统状态恢复回来,唤醒的性能与硬盘速度有关

**通过system管理各服务之间的依赖性**

**systemctl list-dependencies [unit] [--reverse]  
 --reverse:反向追踪谁使用这个unit**

systemctl list-dependencies  
**default.target 中间有省略的内容**├─abrt-ccpp.service  
├─abrt-oops.service  
├─vsftpd.service  
**├─basic.target**│ ├─alsa-restore.service  
│ ├─alsa-state.service  
**│ ├─sockets.target**│ │ ├─avahi-daemon.socket  
│ │ ├─dbus.socket  
**│ ├─sysinit.target**│ │ ├─dev-hugepages.mount  
│ │ ├─dev-mqueue.mount  
**│ └─timers.target**│ └─systemd-tmpfiles-clean.timer  
**├─getty.target**│ └─getty@tty1.service  
**└─remote-fs.target**

systemctl list-dependencies --reverse #反查谁会用到 multi-user.target   
default.target  
└─graphical.target

systemctl list-dependencies graphical.target  
graphical.target  
├─accounts-daemon.service  
├─gdm.service  
├─network.service  
├─rtkit-daemon.service  
├─systemd-update-utmp-runlevel.service  
**└─multi-user.target** ├─abrt-ccpp.service  
 ├─abrt-oops.service  
.....（下面省略）.....

graphical.target 就是在 multi-user.target 下面再加上 accounts-daemon, gdm, network, rtkit-deamon,systemd-update-utmp-runlevel 等服务而已，了解daemon之间的相关性，出问题时，可以找到正确的服务依赖流程！

**与system运行相关的目录**

**/usr/lib/systemd/system/：** 官方带每个服务**默认原始的**启动脚本配置**实际文件**

类似/etc/init.d 下面的文件，优先序最低

**/run/systemd/system/：** 系统**执行过程中**产生的服务脚本，优先序稍高  
 **/etc/systemd/system/：** 依据主机系统的需求**管理员所创建**的执行脚本，优先序最高

类似 /etc/rc.d/rc5.d/Sxx 之类的功能，都是**链接文件**

**尽量在/etc/systemd/system下面修改配置**

/etc/sysconfig/\*： 几乎所有服务都会将初始化的一些**选项设置**写入到此目录

mandb要更新的man page索引要加的参数写入到此目录下的man-db

网络设置则写在network-scripts/ 这个目录内

/var/lib/： 一些会产生数据的服务会将数据写入到/var/lib/目录中

Mariadb的数据库默认写入/var/lib/mysql/

/run/： 放置许多daemon的暂存盘，如lock file以及PID file

系统开机会不会执行某些服务看/etc/systemd/system/ 下面的设置，但仅是链接到/usr/lib/systemd/system/的执行脚本配置文件。

实际执行的systemd启动脚本配置文件放置在 /usr/lib/systemd/system/ 下面，在此修改查阅配置文件。

systemctl list-sockets 查看socket 服务，知道正在监听本机服务需求的socket file位置

**网络服务与端口对应简介**

<http://ftp.ksu.edu.tw/> ftp://ftp.ksu.edu.tw/

两个网址都是指向 ftp.ksu.edu.tw的FTP网站，但是浏览器上面显示的结果却是不一样

指向不同的服务，一个是 http 这个WWW的服务，一个则是ftp这个文件传输服务

**通讯协定**：统一整个网际网络的端口号对应服务的功能，好让所有的主机都能够使用相同的机制来提供服务与要求服务，有些约定俗成的服务都放置在同一个端口号上面

cat /etc/services  
 ftp 21/tcp  
 ftp 21/udp fsp fspd  
 ssh 22/tcp # The Secure Shell （SSH） Protocol  
 ssh 22/udp # The Secure Shell （SSH） Protocol  
 http 80/tcp www www-http # WorldWideWeb HTTP  
 http 80/udp www www-http HyperText Transfer Protocol

# 这个文件的内容是以下面的方式来编排的：  
 # <daemon name> <port/封包协定> <该服务的说明>

封包协议：可靠连线的 TCP 封包以及较快速但为非连线导向的 UDP

可以修改/etc/services更改端口号，并不建议如此，因为很有可能会造成一些协定的错误

除非你要架设一个地下网站，否则的话，使用/etc/services原先的设置就好

**管壁网络服务**

使用systemctl去观察启动的服务时，发现很多个daemon，其实systemd将许多原本不被列为daemon 的程序都纳入到systemd的管辖监测范围内

那些大部分都属于Linux 统基础运行所需要的环境，没有什么特别需求的话，最好都不要更动，除非你自己知道自己需要什么

除本机服务之外，要注意的是网络服务喔，建议关闭非必要的网络服务

**网络服务：会产生一个网络监听端口port的程序**

netstat -tlunp  
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name  
tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1340/sshd  
tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2387/master  
tcp6 0 0 :::555 :::\* LISTEN 29113/vsftpd  
tcp6 0 0 :::22 :::\* LISTEN 1340/sshd  
tcp6 0 0 ::1:25 :::\* LISTEN 2387/master  
**udp 0 0 0.0.0.0:5353 0.0.0.0:\* 750/avahi-daemon: r  
udp 0 0 0.0.0.0:36540 0.0.0.0:\* 750/avahi-daemon: r**

**zero configuration规范由apple提出，它的实现：开源avahi、Apple的bonjour**

**#傻瓜式连接局域网，像计算机、打印机等，类似局域网广播、网上邻居等**

**关闭网络服务**

systemctl stop avahi-daemon.service  
systemctl stop avahi-daemon.socket  
systemctl disable avahi-daemon.service avahi-daemon.socket

**system针对service类型的配置文件**

**/usr/lib/systemd/system/**vsftpd.service：

官方释出的默认原始配置文件；  
**/etc/systemd/system/**vsftpd.service.d/custom.conf：

在/etc/systemd/system下面创建与配置文件相同文件名的目录，但是要加上.d 的扩展名，然后在该目录下创建配置文件即可，配置文件最好附文件名取名为.conf，在这个目录下的文件会“累加其他设置”进入 /usr/lib/systemd/system/vsftpd.service内。  
**/etc/systemd/system/vsftpd.service.wants**/\*：

此目录内的文件为链接文件，设置相依赖服务的链接。意思是启动了vsftpd.service之**后**，最好再加上这目录下面建议的服务。  
**/etc/systemd/system/vsftpd.service.requires**/\*：

此目录内的文件为链接文件，设置相依服务的链接。意思是在启动vsftpd.service 之**前**，需要事先启动哪些服务的意思。

建议不要修改原设置，而是到另几个目录去进行额外的自定义设置  
硬要修改原始的/usr/lib/systemd/system下面的配置文件，也可以，且能减少配置文件

cat /usr/lib/systemd/system/sshd.service

**[Unit]**  # 这个项目与此 unit 的解释、执行服务相依性有关  
Description=OpenSSH server daemon  
After=network.target sshd-keygen.service  
Wants=sshd-keygen.service

**[Service]**  # 这个项目与实际执行的指令参数有关  
EnvironmentFile=/etc/sysconfig/sshd  
ExecStart=/usr/sbin/sshd -D $OPTIONS  
ExecReload=/bin/kill -HUP $MAINPID  
KillMode=process  
Restart=on-failure  
RestartSec=42s

**[Install]**  # 这个项目说明此 unit 要挂载哪个 target 下面  
WantedBy=multi-user.target

**配置文件结构分解**

**[Unit]：**unit本身的说明以及与其他依赖daemon的设置

**[Service], [Socket], [Timer], [Mount], [Path]**：

不同的unit type就得要使用相对应的设置项目。主要规范服务启动的脚本、环境配置文件文件名、重新启动的方式等  
**[Install]**：将此 unit 安装到哪个 target 里面去

**设置规则：**设置项目通常是**可以重复**的，可以重复设置两个After，但后面的设置会取代前面的；

若要将设置值归零，可以使用类似“After=“的设置  
设置参数需要有“是/否”的项目（布林值, boolean）

**可以使用 1, yes, true, on 代表启动，用 0, no, false, off 代表关闭**空白行、开头为 # 或 ; 的代表注解

**system配置详解**

|  |  |
| --- | --- |
| **[Unit] 部份** | |
| 设置参数 | 参数意义说明 |
| Description | 使用systemctl list-units、systemctl status时，输出给管理员看的简易说明 |
| Documentation | 提供管理员能够进行进一步的文件查询的功能！提供的文件可以是如下的数据： Documentation=http://www.... Documentation=man:sshd（8） Documentation=file:/etc/ssh/sshd\_config |
| After  在后接服务  之后启动最好 | 说明**此unit是在哪个daemon 启动之后才启动**的意思！  仅说明服务启动顺序，**并非强制**要求后面接的服务一定要启动此unit才能启动    以sshd.service为例，提到After后有network.target 以及sshd-keygen.service，若这两个unit 未启动而强制启动sshd.service 的话， sshd.service还是能启动。  这与 Requires 的设置是有差异的！ |
| Before  在后接服务  之前启动最好 | 在**什么服务启动前最好启动此unit服务**的意思  不过这仅是规范服务启动的顺序，**并非强制要求**的意思。 |
| Requires  **必须**需后接服务开启才开启 | **明确的定义此unit需要在哪个 daemon 启动后才能够启动**  相依赖服务，如果在此项设置的前导服务没有启动，那么此unit就不会被启动！ |
| Wants | 规范**此unit之后最好还要启动什么服务比较好**  **非明确规范**，主要希望创建让使用者比较好操作的环境  Wants后接的服务如果未启动，不会影响到这个unit本身 |
| conflicts | 代表冲突的服务  即这个项目后面接的服务如果启动，那么我们此unit 本身就不能启动！  此unit有启动，则此项目后的服务就不能启动！ |

|  |  |
| --- | --- |
| **[Service] 部份** | |
| 设置参数 | 参数意义说明 |
| Type | 说明这个daemon启动的方式，会影响到 ExecStart ！有下面几种类型： **simple**：默认值，这个 daemon 主要由 ExecStart 接的指令串来启动，**启动后常驻于内存中**。 **forking复刻：**由ExecStart启动的程序通过spawns延伸出其他子程序来作为此daemon的主要服务，原生的父程序在启动结束后就会终止运行。  传统的unit服务大多属于这种项目，如httpd 这个WWW服务，当httpd的程序因为运行过久因此即将终结了，则systemd会再重新生出另一个子程序持续运行后，再将父程序删除。据说这样的性能比较好！ **oneshot：**与simple 类似，不过这个程序在工作完毕后就结束了，**不会常驻在内存中**。  dbus：与simple类似，但这个daemon必须要在取得一个D-Bus的名称后，才会继续运行！因此设置这个项目时，通常也要设置 BusName= 才行！ idle：与simple类似，要执行这个daemon**必须要所有的工作都顺利执行完毕后才会执行**。  这类的 daemon 通常是开机到最后才执行即可的服务！  比较重要的项目是simple, forking 与oneshot 了！很多服务需要**子程序（forking）**，而有更多的动作只需要在开机的时候执行一次（oneshot），如文件系统的检查与挂载等 |
| EnvironmentFile | 指定启动脚本的环境配置文件！  例如sshd.service的配置文件写入到 /etc/sysconfig/sshd 当中  可以使用 Environment= 后面接多个不同的Shell变量来给予设置 |
| ExecStart | 就是实际执行此daemon的指令或脚本程序。  可用 ExecStartPre及ExecStartPost两个设置项目来在实际启动服务前进行额外的指令行为  但是你得要特别注意的是，指令串仅接受“指令 参数 参数...”的格式，不能接受<, >, >>, |, &等特殊字符，很多的bash 语法也不支持，最好直接写入到指令脚本里面去！  使用Type=oneshot也支持完整的bash语法，其他的Type不支持这些字符。 |
| ExecStop | 与systemctl stop 的执行有关，关闭此服务时所进行的指令。 |
| ExecReload | 与systemctl reload 有关的指令行为 |
| Restart | 当设置 Restart=1 时，则当此 daemon 服务终止后，会再次的启动此服务。  举例来说，如果你在 tty2 使用文字界面登陆，操作完毕后登出，基本上，这个时候 tty2 就已经结束服务了。但是你会看到屏幕又立刻产生一个新的 tty2 的登陆画面等待你的登陆！那就是 Restart 的功能！  除非使用 systemctl 强制将此服务关闭，否则这个服务会源源不绝的一直重复产生！ |
| Remain  AfterExit | RemainAfterExit=1时，则当此daemon所属的所有程序都终止之后，此服务会再尝试启动。 这对于Type=oneshot的服务很有帮助 |
| TimeoutSec | 若这个服务在启动或者是关闭时，因为某些缘故导致无法顺利“正常启动或正常结束”的情况下，则我们要等多久才进入“强制结束”的状态！ |
| KillMode | 可以是process, control-group, none 的其中一种  如果是proces则daemon 终止时，只会终止主要的程序（ExecStart 接的后面那串指令）；如果是control-group时，由此daemon所产生的其他control group的程序，也都会被关闭；  如果是none的话，则没有程序会被关闭 |
| RestartSec | 与Restart有点相关性，若此服务被关闭需要重新启动大概要sleep多少时间再重新启动 默认是 100ms（毫秒） |

|  |  |
| --- | --- |
| **[Install] 部份** | |
| 设置参数 | 参数意义说明 |
| WantedBy | 这个设置后面接的大部分是 \*.target unit  **这个unit本身是附挂在哪一个target unit下面的**  大多的服务性质的unit都是附挂在multi-user.target下面  graphical.target需要multi-user.target |
| Also | 此unit本身被enable时，Also后面接的unit也请enable  有依赖性的服务可以写在这里 |
| Alias | 进行一个链接的别名，systemctl enable相关的服务时，则此服务会进行链接文件的创建；  multiuser.target是用来作为默认操作环境default.target的，当设置用成default.target时，/etc/systemd/system/default.target就会链接到 /usr/lib/systemd/system/multi-user.target |

**system配置文件修改实践**

将vsftpd 的port改成555号，可能需要使用到两个端口，分别是正常的21以及特殊的555，这两个port都启用的情况下，你可能就得要使用到两个配置文件以及两个启动脚本设置了！

默认的port 21:使用/etc/vsftpd/vsftpd.conf 配置文件,以及/usr/lib/systemd/system/vsftpd.service 设置脚本；  
特殊的port 555:使用/etc/vsftpd/vsftpd2.conf 配置文件,及/etc/systemd/system/vsftpd2.service 设置脚本。

**# 1. 先创建好所需要的配置文件**cd /etc/vsftpd  
cp vsftpd.conf **vsftpd2.conf**vim vsftpd.conf  
 listen\_port=555  
diff vsftpd.conf vsftpd2.conf  
 128c128  
 < #listen\_port=555  
 > listen\_port=555  
**# 2. 开始处理启动脚本设置**cd /etc/systemd/system  
**cp /usr/lib/systemd/system/vsftpd.service vsftpd2.service**vim vsftpd2.service  
 [Unit]  
 Description=Vsftpd second ftp daemon  
 After=network.target  
 [Service]  
 Type=forking  
 ExecStart=/usr/sbin/vsftpd /etc/vsftpd/**vsftpd2.conf** [Install]  
 WantedBy=multi-user.target  
**# 重点在改了vsftpd2.conf 这个配置文件！  
# 3. 重新载入 systemd 的脚本配置文件内容**systemctl daemon-reload  
systemctl list-unit-files --all | grep vsftpd  
 vsftpd.service enabled  
 vsftpd2.service disabled  
 vsftpd@.service disabled  
 vsftpd.target disabled  
systemctl status vsftpd2.service  
 vsftpd2.service - Vsftpd second ftp daemon  
 Loaded: loaded （/etc/systemd/system/vsftpd2.service; disabled）  
 Active: inactive （dead）  
systemctl restart vsftpd.service vsftpd2.service  
systemctl enable vsftpd.service vsftpd2.service  
systemctl status vsftpd.service vsftpd2.service  
netstat -tlunp 搞定

**多重的重复设置方式：以getty为例**

tty1~tty6 [终端是由agetty指令达成的，主要管理getty.target，实际产生tty1~tty6的是getty@.service提供](mailto:终端是由agetty指令达成的，主要管理getty.target，实际产生tty1~tty6的是getty@.service提供)

cat //usr/lib/systemd/system/getty@.service

[Unit]  
Description=Getty on %I  
Documentation=man:agetty(8)man:systemd-getty-generator(8)  
Documentation=http://0pointer.de/blog/projects/serial-console.html  
After=systemd-user-sessions.service plymouth-quit-wait.service  
After=rc-local.service  
Before=getty.target  
ConditionPathExists=/dev/tty0  
[Service]  
**ExecStart=-/sbin/agetty --noclear %I $TERM**Type=idle  
Restart=always  
RestartSec=0  
UtmpIdentifier=%I  
TTYPath=/dev/%I  
TTYReset=yes

TTYVHangup=yes  
TTYVTDisallocate=yes  
KillMode=process  
IgnoreSIGPIPE=no  
SendSIGHUP=yes  
[Install]  
**WantedBy=getty.target #挂附在此unit上面**

systemctl show getty.target # show指令可将getty.target的默认设置值也取出来  
Names=getty.target  
Wants=getty@tty1.service  
WantedBy=multi-user.target  
Conflicts=shutdown.target  
Before=multi-user.target  
**After=getty@tty1.service getty@tty2.service getty@tty3.service getty@tty4.service  
getty@tty5.service getty@tty6.service**  
.....后面省略

当我们执行完 getty.target之后，他会持续要求getty@tty1.service 等六个服务继续启动。那我们的 systemd 就会这么作：

先看/usr/lib/systemd/system/, /etc/systemd/system/有没有getty@tty1.service的设置，若有就执行，若没有则执行下一步；  
找getty@.service的设置，若有则将@后面的数据带入成%I的变量，进入getty@.service 执行！

也就是说其实getty@tty1.service 实际上是不存在的！主要是通过 getty@.service 来执行

getty@.service的目的是为了要简化多个执行的启动设置， 他的命名方式是这样的：

原始文件：执行服务名称@.service  
 可执行文件案：执行服务名称@范例名称.service

ExecStart=-/sbin/agetty --noclear %I $TERM %I 指的就是“范例名称”

[getty@tty1.service的%I就是tty1](mailto:getty@tty1.service的%25I就是tty1)，执行脚本就会变成/sbin/agetty --noclear tty1

**以一个配置文件来启动多个tty1服务给用户登陆**

**改变tty数量**

1. 修改默认的 logind.conf 内容，将原本 6 个虚拟终端机改成 4 个  
   vim /etc/systemd/logind.conf  
   [Login]  
   NAutoVTs=4 #原本是6个而且还注解，取消注解，然后改成 4 吧！  
   ReserveVT=0  
   2. 重新启动 getty.target  
   systemctl restart systemd-logind.service

按下 [ctrl]+[alt]+[F1]~[F6]就会发现，只剩下四个可用的tty！后面的 tty5, tty6已经被放弃了！  
那么我暂时需要启动 tty8 ，systemctl start getty@tty8.service， 不需额外创建其他的启动服务配置文件

**暂时新增 vsftpd 到 2121 端口**

根据 vsftpd@.service的建议，于 /etc/vsftpd/ 下面先创建新的配置文件  
cd /etc/vsftpd  
cp vsftpd.conf **vsftpd3.conf**  
vim vsftpd3.conf

listen\_port=2121

**systemctl start vsftpd@vsftpd3.service**systemctl status vsftpd@vsftpd3.service  
 vsftpd@vsftpd3.service - Vsftpd ftp daemon  
 Loaded: loaded （/usr/lib/systemd/system/vsftpd@.service; disabled）  
 Active: active （running） since Thu 2015-08-13 01:34:05 CST; 5s ago

netstat -tlnp  
Active Internet connections （only servers）  
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name  
tcp6 0 0 :::2121 :::\* LISTEN 16404/vsftpd  
tcp6 0 0 :::555 :::\* LISTEN 12672/vsftpd  
tcp6 0 0 :::21 :::\* LISTEN 12670/vsftpd

启用了vsftpd@vsftpd3.service，代表要使用的配置文件在 /etc/vsftpd/vsftpd3.conf的意思！

所以可以通过修改config文件、直接启动vsftpd@.service 而无须重新设置启动脚本

为啥这次 FTP 增加了 2121 端口却不用修改 SELinux 呢？这是因为默认启动小于 1024 号码以下的端  
口时， 需要使用到 root 的权限，因此小于 1024 以下端口的启动较可怕

**自己创建服务**

vim /backups/backup.sh #编写脚本  
#!/bin/bash  
source="/etc /home /root /var/lib /var/spool/{cron,at,mail}"  
target="/backups/backup-system-$（date +%Y-%m-%d）.tar.gz"  
[ ! -d /backups ] && mkdir /backups  
tar -zcvf ${target} ${source} &> /backups/backup.log

vim /etc/systemd/system/backup.service #编写启动脚本的设置  
[Unit]  
Description=backup my server  
**Requires=atd.service** #ExecStart里用到at指令，因此，atd.service 是一定要的服务！[Service]  
Type=simple  
ExecStart=/bin/bash -c " echo /backups/backup.sh | **at** now"  
[Install]  
WantedBy=multi-user.target

systemctl daemon-reload  
systemctl start backup.service  
systemctl status backup.service

backup.service - backup my server  
Loaded: loaded （/etc/systemd/system/backup.service; **disabled**）  
Active: **inactive （dead）**  
Aug 13 07:50:31 study.centos.vbird systemd[1]: Starting backup my server...  
Aug 13 07:50:31 study.centos.vbird bash[20490]: job 8 at Thu Aug 13 07:50:00 2015  
Aug 13 07:50:31 study.centos.vbird systemd[1]: Started backup my server.  
# 为什么 Active 是 inactive 呢？这是因为我们的服务仅是一个简单的 script 啊！  
# 因此执行完毕就完毕了，不会继续存在内存中喔！

**timer的配置文件**

|  |  |
| --- | --- |
| [Timer] 部份 | |
| 设置参数 | 参数意义说明 |
| OnActiveSec | 当 timers.target 启动多久之后才执行这只unit |
| OnBootSec | 当开机完成后多久之后才执行 |
| OnStartupSec | 当 systemd 第一次启动之后过多久才执行 |
| OnUnitActiveSec | 这个timer配置文件所管理的unit服务在最后一次启动后，隔多久再执行一次 |
| OnUnitInactiveSec | 这个timer配置文件所管理的unit服务在最后一次停止后，隔多久再执行一次 |
| OnCalendar | 使用**实际时间**(非循环时间)的方式来启动服务的意思 |
| Persistent | 当使用OnCalendar的设置时，指定该功能要不要持续进行的意思  通常是设置为 yes，能够满足类似 anacron 的功能 |
| Unit | 一般不需设置，基本都是 sname.server +sname.timer  如果sname并不相同时，那在 .timer 的文件中，就要指定是哪一个service unit |

us 或 usec：微秒 （10-6 秒）  
ms 或 msec：毫秒 （10-3 秒）  
s, sec, second, seconds  
m, min, minute, minutes  
h, hr, hour, hours  
d, day, days  
w, week, weeks  
month, months  
y, year, years

隔 3 小时： 3h 或 3hr 或 3hours  
隔 300 分钟过 10 秒： 10s 300m  
隔 5 天又 100 分钟： 100m 5day  
# 通常英文的写法，小单位写前面，大单位写后面，所以先秒、再分、再小时、再天数等

|  |  |
| --- | --- |
| 英文口语 | 实际的时间格式代表 |
| now | Thu 2015-08-13 13:50:00 |
| today | Thu 2015-08-13 00:00:00 |
| tomorrow | Thu 2015-08-14 00:00:00 |
| hourly | \*-\*-\* \*:00:00 |
| daily | \*-\*-\* 00:00:00 |
| weekly | Mon \*-\*-\* 00:00:00 |
| monthly | \*-\*-01 00:00:00 |
| +3h10m | Thu 2015-08-13 17:00:00 |
| 2015-08-16 | Sun 2015-08-16 00:00:00 |

开机后 2 小时开始执行一次这个 backup.service，自从第一次执行后，未来我每两天要执行一次 backup.service

vim /etc/systemd/system/backup.timer

[Unit]

Description=backup my server timer  
[Timer]  
OnBootSec=2hrs  
OnUnitActiveSec=2days  
[Install]  
WantedBy=multi-user.target

systemctl daemon-reload

systemctl enable backup.timer  
systemctl restart backup.timer

systemctl list-unit-files | grep backup

backup.service disabled # 这个不需要启动！只要 enable backup.timer 即可！  
backup.timer enabled

**systemctl show timers.target**

**ConditionTimestamp=Thu 2015-08-13 14:31:11 CST** # timer 这个 unit 启动的时间！

**systemctl show backup.service**

**ExecMainExitTimestamp=Thu 2015-08-13 14:50:19 CST** # backup.service 上次执行的时间

systemctl show backup.timer

NextElapseUSecMonotonic=2d 19min 11.540653s # 下一次执行距离 timers.target 的时间

上次执行**backup.service**的时间是在2015-08-13 14:50 ，由于设置两个小时执行一次，因此下次应该是  
2015-08-15 14:50 执行才对

**由于timer是由timers.target这个unit所管理的，而这个unit的启动时间是在 2015-08-13 14:31**

**最终backup.timer所纪录的下次执行时间，其实是与 timers.target 所纪录的时间差！**

因此是“ (2015-08-15 14:50) –(2015-08-13 14:31) ”才对！ 所以时间差就是 2d 19min

星期天凌晨 2 点运行

[Unit]  
Description=backup my server timer2  
[Timer]  
OnCalendar=Sun \*-\*-\* 02:00:00  
Persistent=true  
Unit=backup.service  
[Install]  
WantedBy=multi-user.target

systemctl show backup2.timer  
NextElapseUSecRealtime=45y 7month 1w 6d 10h 30min

OnCalendar 的方法对照的时间并不是 times.target 的启动时间，而是 Unix  
标准时间！ 亦即是 1970-01-01 00:00:00 去比较的

**常用启动服务**

|  |  |
| --- | --- |
| **服务名称** | **功能简介** |
| **acpid** | 电源管理接口，笔记本建议开启 |
| **accounts-daemon （可关闭）** | （系统）使用accounts.service所提供的D-Bus界面来进行使用者帐号信息的查询  基本上是与 useradd, usermod, userdel 等软件有关。 |
| **alsa-X （可关闭）** | （系统）开头为 alsa 的服务大部分都与音效有关  服务器且不开图形界面的话，这些服务可以关闭 |
| **atd** | （系统）单一的例行性工作调度，配置文件在/etc/at.{allow,deny} ，crond |
| **auditd** | （系统）SELinux所需服务的一项  开启SELinux 审核信息写入 /var/log/audit/audit.log，不开启记录到syslog中 |
| **avahi-daemon （可关闭）** | （系统）用户端服务，可通过 Zeroconf自动分析与管理网络，类似网上邻居  Zeroconf 较常用在笔记本电脑与行动设备上，发现打印机、共享文件夹等，可关闭 |
| **Apparmor** | Selinux类似的高效易用的访问控制系统，对操作系统和应用从内到外保护 |
| **bluetooth（可关闭）** |  |
| **cpupower** | （系统）提供 CPU 的运行规范，可参考 /etc/sysconfig/cpupower |
| **cronie** | （系统）系统配置文件为 /etc/crontab，循环定时任务 |
| **org.cups.cupsd**  **（可关闭）** | （系统/网络）用来管理打印机的服务，可以提供网络连线的功能，有点类似打印服务器的功能 可以在 Linux 本机上以浏览器的 http://localhost:631 来管理打印机 |
| **dbus** | （系统）使用 D-Bus 的方式在不同的应用程序之间传送讯息，充当路由器的角色  如应用程序间的讯息传递、每个使用者登陆时提供的讯息数据等。 |
| **haveged** | 随机数生成器，解决系统熵值过低的问题，建议开启 |
| **gpm** | 通用鼠标管理，允许在tty内用鼠标复制粘贴 |
| **ftpd** |  |
| **gdm** | （系统）GNOME 的登陆管理员，就是图形界面上一个很重要的登陆管理服务！ |
| **getty@** | （系统）就是要在本机系统产生几个文字界面 （tty） 登陆的服务啰！ | |
| **vmtoolsd** | （系统）让实体机器来创建虚拟机的，如果不玩虚拟机，这些服务可以先关闭。  如果你的Linux本来就在虚拟机的环境下，这些服务没有用 | |
| **lightdm** | Light display manager显示管理器 | |
| **lvm2-\*** | （系统）跟 LVM 相关性较高的许多服务，最好不关！ | |
| **ModemManager NetworkManager** | （系统/网络）主要就是调制解调器、网络设置等服务 | |
| **polkit** | 在系统层级进行权限控制，通过集中策略精细授权，与sudo不同，未赋予root权限，如运行gparted将用户按群组或用户划分，如wheel群组，定义魅操作由某些用户执行，操作前需要额外确认，如输入密码确认用户是否属于某群组群组 | |
| **quotaon** | （系统）启动 Quota 要用到的服务 | |
| **rsyncd.cocket** | .ser vice 远程数据备份 | |
| **rpc\*** | RPC远程过程调用，包括 NFS, NIS 等等都需要这东西的协助 | |
| **sshd** | 远端连线服务器的软件功能，这个通讯协定比 telnet 好的地方在于 sshd 在传送数据时可以进行加密 | |
| **snapd.arrarmor** | snapcraft全新的软件环境，软件类似被封装到Docker容器中，由后台服务snapd提供支持 | |
| **syslog** | 日志守护进程 | |
| **systemd-\*** | （系统）大概都是属于系统运行过程所需要的服务，没必要都不要更动它的默认状态！ | |
| **tlp** | 高级电源管理系统开关 | |
| **upower** | 电源管理 | |
| **udev-trigger**  **udevvd** | （系统）监控设备对应表 （device mapper） 的主要服务，不能关，使用我们的周边设备与储存设备 | |
| **ufw** | （系统/网络）就是防火墙  以前有 iptables 与 ip6tables 等防火墙机制，新的 firewalld | |
| **wpa** | Wifi protected access 保护无线电脑网络安全 | |