

在linux中，对于每一个文件/目录来讲，都针对三个对象有三种权限

- (1) 都有三种权限，分别是r,w,x
- (2) 针对的对象是：user/group/others

#### 连接文件：

- (1) 硬链接 —— `ln` 源文件 目标文件：硬链接，不能连接目录，只能连接文件
- (2) 软连接 —— `ln -s` 源文件/目录 目标文件/目录

预备知识：对于每一个文件，都有一个i-node，和人一样，每个人都有自己的身份证（ID Card）

硬链接	软连接
同步更新	相当于Windows的快捷方式
i-node相同：-l的详细信息都一样	i-node不相同
删除一个文件，另一个文件仍然存在	删除源文件，目标文件失效
不能跨文件系统	能跨文件系统

管道 `command1 | command2 | command3 | .....`

解释：一个命令的输出，作为下一个命令（该命令必须具有接受input的能力才可以）的输入

具有接受input能力的命令：less, more, head, tail, wc

不具有接受input能力的命令：ls, cat, cp, mv

#### 命令连接符

； 逐条执行

`command1 && command2` `command1`执行成功，`command2`才会执行；否则，`command2`不执行

`command1 || command2` `command1`执行失败，`command2`才会执行；否则，`command2`不执行

#### 命令替换符

`command2 `command1`` `command1`的输出，作为`command2`的输入

```
gjw@ubuntu:~$ ls -l `which touch`  
lrwxrwxrwx 1 root root 10 Jun  7 05:34 /usr/bin/touch -> /bin/touch
```

`echo "/etc/profile"`的输出结果，参数化后，作为`cat`的输入

```
gjw@ubuntu:~$ cat `echo "/etc/profile"`  
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))  
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).  
  
if [ "$PS1" ]; then  
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then  
    # The file bash.bashrc already sets the default PS1.  
    # PS1='\h:\w\$ '  
    if [ -f /etc/bash.bashrc ]; then
```

## xargs -i关键字，必须与管道符号|一起使用

由于管道符号有上面的弊端：命令一旦不具有接受input能力，就会输出“意想不到”的错误结果，因此引出xargs -i关键字，它的作用：将管道前的输出进行处理，传递到管道后命令的参数{}上解释：

-i选项：表示将管道前命令的输出结果“分条传输”

{}：表示将管道前命令的输出结果，存放的位置，可以有零个或一个或多个，其中零个则默认为{}放在句尾。

```
find /etc -name init* -a -type -f | xargs -i cp {} {}.dev
```

\*通过下面例子，可以看到有无xargs的区别之处：假设当前所在目录为/home/gjw

```
gjw@ubuntu:~$ echo '/etc/profile' | cat
/etc/profile
gjw@ubuntu:~$
gjw@ubuntu:~$
gjw@ubuntu:~$
gjw@ubuntu:~$
gjw@ubuntu:~$ echo '/etc/profile' | xargs cat
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
  fi
fi
```

```
gjw@ubuntu:~$ echo '/etc/profile' | ls -l
total 44
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Desktop
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Documents
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Downloads
-rw-r--r-- 1 gjw gjw 8980 Jun 7 05:37 examples.desktop
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Music
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Pictures
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Public
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Templates
drwxr-xr-x 2 gjw gjw 4096 Jun 7 05:41 Videos
gjw@ubuntu:~$
gjw@ubuntu:~$ echo '/etc/profile' | xargs ls -l
-rw-r--r-- 1 root root 575 Oct 22 2015 /etc/profile
gjw@ubuntu:~$
```

## 输入/输出重定向（三种标准输入/输出方式）

标准输入（STDIN）：代码为0 使用< << 文件内容，输出到屏幕

标准输出（STDOUT）：代码为1 使用> >> 屏幕内容，写入到文件

标准错误输出（STDERR）：代码为2 使用2> 2>>

一个例子搞定：将正确和错误的数据，都写入到test.log文件中

find ~ -name .bashrc >test.log 2>test.log 错误写法

find ~ -name .bashrc >test.log 2&1 正确写法

find ~ -name .bashrc &>test.log 正确写法

### 双向重定向：tee

将数据流既可以在屏幕上显示（STDIN），又可以存入到文件中（STDOUT）

注解：使用tee需要与管道符|一起使用

例：caffe训练脚本：.....caffe train --solver=..... 2>&1 | tee log.txt

技术|为初学者介绍的 Linux tee 命令(6 个例子): <https://linux.cn/article-9435-1.html>

## 命令位置查询

1.which 命令名称：查看系统命令所在目录

2.whichis 命令名称：查看系统命令所在目录+该系统命令帮助信息文件存放位置

注意：只可以查找系统命令所在目录位置，不能查找其他文件所在位置

### 文件位置查询:

- 1.locate 文件名 # 在系统安装或定期更新的文件数据库中进行查找 (因此查找速度很快)
  - 2.find 搜索路径 [-选项] "匹配字符串" # 在某个路径下, 查找"关键字"匹配的文件和目录
- 选项介绍:

-name: 名字

-type: d目录, f文件, l软连接

-user: 所有者

(1) find的通配符 (注意: 与正则表达式的通配符不一样)

\* 任意0到无数个字符

? 任意一个字符

(2) 连接符号

-a: ---> 逻辑与and

-o: ---> 逻辑或or

例: find /etc -name init\* -a -type f # 查找: 名字为init\*, 并且, 类型是f

(3) find ... -exec 命令 {} \;

解释:

find ... -exec \; 是固定格式, 一个不能少, 必须全写

命令 {}: 二者位置可以根据命令的使用调节, {}表示前面find查询的结果

find /etc -name init\* -exec ls -l {} \;

find /etc -name init\* -exec rm -rf {} \;

find /etc -name init\*-a -type -f -exec cp {} {}.dev \; # 将查找到的文件, 复制一份, 名字为{}.dev

(4) 管道符号|, xargs -i

注解: -i表示分条传输, 将前面find查找到的结果, 一条一条传入{}中, 进行后续的处理

find /etc -name init\* | xargs -i ls -l {}

find /etc -name init\* | xargs -i rm -rf {}

find /etc -name init\*-a -type -f | xargs -i cp {} {}.dev

### 目录操作:

cd: 切换目录

pwd: 查看当前目录

touch: 创建/修改文件

mkdir: 创建目录 -p: 递归创建目录

cp [-R] 源文件/源目录 目标文件/目标目录 : 复制

注解: 对于目录, 必须使用-R选项

mv 源文件/源目录 目标文件/目标目录 : 剪切

rm [-R -f] 源文件/源目录 : 删除

注解:

对于目录, 必须使用-R选项

-f: 强制删除

## 读取文件内容操作:

cat

more: 可以向下翻页查看文件内容, 但是不能向上翻页

less: 上下翻页查看

head: head -20: 查看文件前20行

tail: tail -20: 查看文件后20行

## ls命令讲解: 非常重要 (尤其是-l选项的详解)

选项:

-a (all)隐藏文件也显示

-l (long)长属性, 即详细属性

-d (directory)查看目录的属性

例1: -d选项的使用方法

ls -l /etc 查看/etc目录下的所有文件的详细信息

ls -ld /etc 查看/etc目录的详细信息

例2:

ls -l text



文件类型:

d: 目录 -: 文件 l: 软连接文件

文件权限:

r: 读 w: 写 x: 可执行

硬链接数:

所有者: user

所属组: group

注意: 文件和目录的r,w,x权限, 有很大差别

文件	目录
r 可读取文件内容: cat,more,head,tail	列出目录下的文件: ls
w 可修改文件内容: vi	在目录下新建/删除文件: touch,mkdir,rm
x 可执行该文件	允许用户cd目录
对于文件来讲, 一般赋予r--权限	
对于目录来讲, 一般赋予r-x权限	

## 权限处理命令:

1.查看创建文件/目录的默认属性,掩码值: `umask [-S]`

`umask`

`>>>0222`

0: 特殊权限位

022: 用户权限位——用 $777-022=755$ 才是默认创建权限

`umask -S`

`>>>u=rwx,g=rx,o=rx`

Linux不成文的规定:

缺省创建的文件,不授予x权限,因此是644

缺省创建的目录,权限是655

2.`chown [用户] [文件/目录]` 改变所有者

3.`chgrp [用户组] [文件/目录]` 改变所属组

4.`chmod`: 两种语法结构

`chmod {ugo} {+|=} {rwx} 文件/目录`

`chmod 644 文件/目录`

其中: r-4 w-2 x-1, 644 表示 rw- r-- r--

## 特殊权限: SUID/SGID/SBIT

1.先介绍如何添加和删除这三种权限:

`chmod {ugo} {+|=} {s} 可执行程序`

`chmod 7755 可执行程序` : 7755的第一位表示 $4+2+1$ , 分别对应SUID+SGID+SBIT

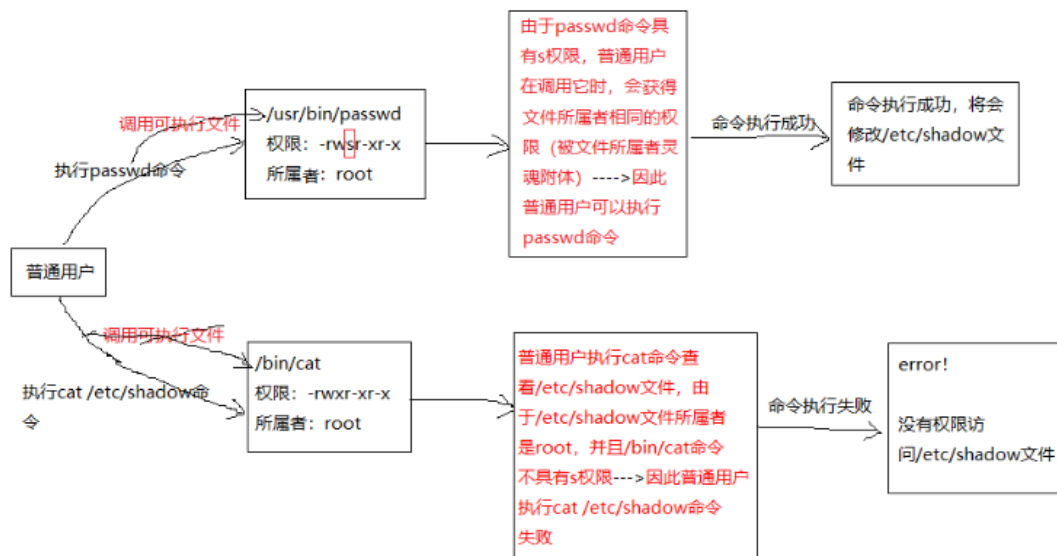
`chmod 755 可执行程序` : 删除SUID/SGID/SBIT权限

2.详细SUID权限

SUID权限的特色:

- (1) SUID权限只对可执行文件有效
- (2) 执行者对于该程序需要具有x权限
- (3) 本权限仅在执行该文件的过程中生效
- (4) 在执行过程中, 执行者暂时获得可执行文件所有者的权限

效果: 当普通用户执行一个具有s权限的程序时, 该用户将以它的所有者身份执行



## ACL权限：身份只有三种user,group,other，解决身份不够的情况

ACL权限主要针对以下三个方面进行权限控制：

- (1) 用户user
- (2) 用户组group
- (3) 默认属性mask：在某个目录下，创建文件/目录时，设定默认权限

场景：

教学老师开辅导班，创建了一个目录/project用于教学工作，创建了一个教学小组，将报班的学生都拉到小组中；  
对于报班的学生，给每个学生对/project目录都赋予rwx权限；  
对不报班的学生，都赋予---权限

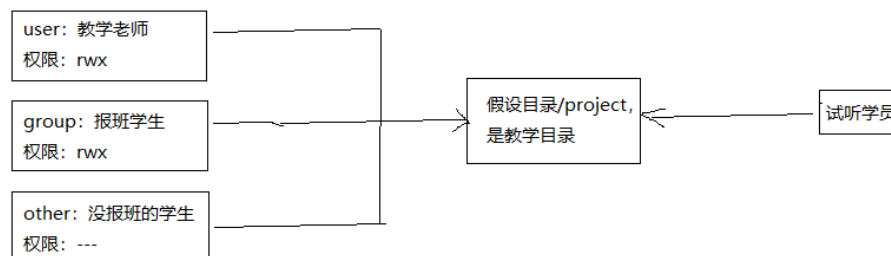
但是，此时此刻，有很多学生有意愿报班，但是报班前想试听下教学老师讲的好不好----->因此，此时又多出来“试听学员”身份，那么问题来了：

使用原有的user, group, others三个权限位，已经不够用了！

----->

因此，引出来ACL权限，

- (1) 可以赋予某个试听学员（张三），r-x权限，让张三能够听课，但是不让他修改/project目录内容
- (2) 可以添加试听组，并赋予r-x权限，让试听组内的成员能够听课，但是不让他们修改/project目录内容



注意：ACL权限必须要又文件系统的支持才行（大多数的文件系统都支持ACL权限），因此需要查看文件系统是否支持ACL权限

### 1.查看文件系统是否支持ACL权限

```
dumpe2fs -h /dev/hda2
```

```
>>> Default mount options: user_xattr acl
```

### 2.给文件系统添加支持ACL权限

(1) mount -o remount,acl / # 给/添加ACL权限，但是每次重启，都会失效

(2) vi /etc/fstab # 给/永久添加ACL权限（写入配置文件）

```
>>> LABEL=/1 / ext3 defaults,acl 1 1
```

### 3.设置ACL权限

setfacl [-bkRd] [{-m|-x} acl参数] 文件/目录名 : 设置文件/目录的ACL权限

-b: 删除所有的ACL权限

-k: 删除默认的ACL权限

-R: 递归设置ACL权限，即包括子目录都会被设置

-d: 设置默认的ACL权限！只针对目录而言，在该目录下创建的文件/目录会引用此默认值

-m: 设置指定的ACL权限

-x: 删除指定的ACL权限

例1: -d选项详解:

给/project目录，通过d选项设置默认ACL权限之后，在/project目录下新建的文件/目录都会继承该默认的ACL权限，使用方法：

```
set -m d:u:Mary:rx /project
```

效果：假设在目录/project下新建文件或目录，那么Mary用户对新建的文件/目录都具有rx权限

例2:

```
setfacl -m u:Mary:rx /project # 给Mary添加ACL的rx权限
setfacl -m g:STGroup:rx /project #给试听组STGroup添加rx权限
```

[说明]在添加完ACL权限后, 用ls -l查看详细属性时, 权限部分多了一个+, 与原本的770有很大差别, 如下: -rwxrwx---+, 那么应该怎么查看+代表的含义呢?

----> **getfacl** 文件/目录名

>>>输出内容

```
# file: /project
# owner:root
# group:root
user::rwx # 用户列表为空, 代表文件所有者root的权限
user:Mary:r-x # 针对Mary用户, 设置的ACL权限
group::rwx # 用户组列表为空, 代表文件所属组root的权限
group:STGroup:r-x # 针对试听组STGroup, 设置的ACL权限
mask::r-x # 为了防止权限过高, 用来控制最大的有效权限 ----> setfacl -m m:rx /project
other::--- # 其他人权限
```

### 压缩解压 (压缩文件的格式: **.gz** **.bz2** **.tar.gz** **.zip**)

(1) 先介绍两个只能压缩文件, 不能压缩目录的命令: gzip, bzip2

**gzip** #压缩后不保留源文件

-d 解压缩

例:

gzip test ----> 将test压缩生成test.gz, test不见了, 只剩下test.gz

gzip -d test.gz ----> 将test.gz解压缩为test

**bzip2**

-k 保留原文件, 不会删除原始的文件

-d 解压缩

例:

bzip2 -k test #保留原文件, 压缩test, 生成test.bz2

bzip2 -d test.bz2 #解压缩文件

(2) 打包命令, 只是将目录文件进行打包成.tar, 并没有进行压缩

**tar**

-v 在打包过程中, 显示正在处理的文件名

-c 新建打包文件

-C 目录: 加压缩到该目录下

-x 解包

-f filename: -f后面要接被处理的文件名, 建议-f单独写一个参数

-j 通过bzip进行压缩/解压缩, 此时文件名最好是\*.tar.bz2

-z 通过gzip进行压缩/解压缩, 此时文件名最好是\*.tar.gz

#### 事先说明规则:

对于打包, 打包并压缩来说, 格式都是**-f 目标文件名 要处理的文件名**

对于解包, 解包并解压缩来说, 格式都是**-f 要处理的文件名**

**打包:** tar -c -v -f filename.tar filename 将filename打包, 生成filename.tar

**解包:** tar -x -v -f filename.tar 将filename.tar解包, 生成filename

说了这么多, 其实最常用的压缩/解压缩命令格式见下:

**tar -jcv -f filename.tar.bz2 filename** # 打包, 压缩命令

**tar -zcv -f filename.tar.gz filename** # 打包, 压缩命令

解释:

-j表示使用bzip2压缩/解压缩格式,

-z表示使用gzip压缩/解压缩格式,

-c表示打包,

-v显示详细信息,

-f filename.tar.bz2 filename此处用法很特殊, 不太符合本意, 表示将filename打包压缩成filename.tar.bz2, 按道理应该-f filename filename.tar.bz2

**tar -jxv -f filename.tar.bz2 -C 解压缩后存放目录** #解包, 解压缩

**tar -zxv -f filename.tar.gz -C 解压缩后存放目录** #解包, 解压缩

解释:

-j表示使用bzip2压缩/解压缩格式,

-z表示使用gzip压缩/解压缩格式,

-x表示解包,

-v显示详细信息,

-f filename.tar.bz2, 此处表示filename.tar.bz2是要被解压缩的文件, 用法符合本意

-C 解压缩后存放目录



简述：登陆用户的过程-----> 当输入账号和密码后，系统做了哪些处理？

1. 先从/etc/passwd里查找是否有你输入的账号，如果没有，则退出；如果有，则将该账号对应的UID和GID（GID在/etc/group中）读取出来；另外，该账号对应的主文件夹和shell设置也一并都出来
2. 进入/etc/shadow找到对应的账号和密码，然后核对刚才输入的密码是否正确
3. 如果密码正确，则进入shell控管的阶段

#### 与账号相关的两个文件

/etc/passwd：管理用户的UID和GID

账号名称：密码：UID：GID：用户信息说明：主文件夹：shell解释器

```
gjlw : x :1000:1000: gjlw,,, : /home/gjlw : /bin/bash
```

/etc/shadow：专门管理密码

账号名称： 加密密码

```
gjlw:$1$AVZJAlV0$tDPuo9QsDTQQLERM5.Ni/:17689:0:99999:7:::
```

#### 与用户组相关的两个文件

用户组名：组密码：GID

用户组文件 /etc/group

```
root:x:0:
```

用户组名：组密码：

用户组密码文件 /etc/gshadow

```
root:*:::
```

在执行useradd时，系统会默认给我们创建很多相关用户信息，比如：默认的宿主目录，UID，账号密码失

效日，宿主目录下有哪些文件，邮箱，等等，可以通过useradd -D命令查看

```
root@ubuntu:/home/gjlw# useradd -D useradd创建用户时，查看默认值
```

```
GROUP=100 默认用户组
```

```
HOME=/home 默认主文件夹目录
```

```
INACTIVE=-1 默认失效日
```

```
EXPIRE=-1 账号失效日
```

```
SHELL=/bin/sh 默认的shell
```

```
SKEL=/etc/skel 用户主文件夹的内容数据参考目录
```

```
CREATE_MAIL_SPOOL=no 是否主动给用户创建邮箱
```

模板目录 /etc/skel 创建新用户时，该目录下有什么文件，再新创建的用户的宿主目录下也会生成相同的文件

登陆信息 /etc/issue 登陆之前显示linux内核版本

/etc/motd 登陆之后显示提示信息

#### 新增与删除用户

```
useradd gjlw
```

```
passwd gjlw
```

```
userdel -r gjlw #连同宿主目录也删除 find / -user gjlw -exec rm -rf {} \; #删除系统中所有属于gjlw的文件
```

usermod [-cdegGlsLU] username 修改用户的相关信息，写入/etc/passwd /etc/shadow /etc/group /etc/gshadow文件

-c 详细信息，如备注

-d 账号主文件夹 /etc/passwd的第六个字段

-e 日期 /etc/shadow的第八个字段

-f 天数 /etc/shadow的第七个字段

-g 初始用户组 /etc/passwd的第四个字段GID

-G 附加用户组 /etc/group

-l /etc/passwd的第一个字段，账号名称

-s shell解释器，如/bin/bash

-u UID，/etc/passwd的第三个字段

-L 暂时将用户的密码冻结，/etc/shadow的密码字段

-U 取消用户密码冻结，/etc/shadow的密码字段

#### 新增与删除用户组

groupadd [-g gid] [-r] 用户组名 # -r选项要加上

groupdel [groupname]

groupmod [-g gid] [-n group\_name] 用户组名

-g 修改已有的GID数字

-n 修改已有的组名

```
root@ubuntu:/home/gjlw# groupadd Group 新建用户组Group
root@ubuntu:/home/gjlw# grep Group /etc/group /etc/gshadow
/etc/group:Group:x:1002:
/etc/gshadow:Group:!::
```

修改Group用户组，gid=201，新组名=GroupMod

```
root@ubuntu:/home/gjlw# groupmod -g 201 -n GroupMod Group
root@ubuntu:/home/gjlw# grep GroupMod /etc/group /etc/gshadow
/etc/group:GroupMod:x:201:
/etc/gshadow:GroupMod:!::
```

```
root@ubuntu:/home/gjlw# groupdel GroupMod 删除用户组
root@ubuntu:/home/gjlw# grep GroupMod /etc/group /etc/gshadow
root@ubuntu:/home/gjlw#
```

删除之后，查找不到GroupMod组的相关信息

设置组密码  
设置组管理员  
将用户从组中添加/删除

```
root@ubuntu:/home/gjlw# groupadd G 新建用户组G
root@ubuntu:/home/gjlw# gpasswd G 给用户组G，创建密码
Changing the password for group G
New Password:
Re-enter new password:
```

```
root@ubuntu:/home/gjlw# gpasswd -A gjlw G 将gjlw用户，设为用户组G的管理员
root@ubuntu:/home/gjlw# grep G /etc/group /etc/gshadow
/etc/group:G:x:1002:
/etc/gshadow:G:$6$DvdR6d7bfb7LOU$50XXJeqyxWp0Yi.cpfUCDzwI8zpNohkcAgAvsTkGMPzik9nb0ZBgpu2RJTC5Nz3NPq//ZcqqsgAp7Kewsxs0:gjlw:
```

#### 切换到gjlw用户

gjlw@ubuntu:~\$ gpasswd -a wl G 将wl用户，添加到用户组G

Adding user wl to group G

gjlw@ubuntu:~\$ grep G /etc/group 查看G组信息

```
G:x:1002:wl
```

gjlw@ubuntu:~\$ gpasswd -d wl G 将wl用户，从用户组G移除

Removing user wl from group G

gjlw@ubuntu:~\$ grep G /etc/group 查看G组信息，wl用户已经不存在G组中

```
G:x:1002:
```

## 用户切换

su - 普通用户名，其中-表示，env也一同切换  
sudo su 切换到root用户

### sudo

可以使普通用户以root身份执行命令，并不是所有的用户都可以执行sudo，而是仅有/etc/sudoers内的用户才能够执行sudo

### 查询登陆用户信息

- 1.查询当前已经登陆系统的用户: w, who
- 2.查询每个账号的最近登陆时间: lastlog
- 3.列出当月登陆者的信息: last

```
[sudo] password for gjw:
root@ubuntu:/home/gjw# w
 23:55:35 up 5:30, 1 user, load average: 0.08, 0.03, 0.01
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU WHAT
gjw       tty7     :0            18:25    ?      24.28s  0.20s /sbin/upsta
root@ubuntu:/home/gjw# who
gjw       tty7     2018-06-07 18:25 (:0)
root@ubuntu:/home/gjw# last
gjw       tty7     :0            Thu Jun 7 18:25      gone - no logout
reboot    system boot  4.10.0-28-generi Fri Jun 8 02:24      still running
gjw       tty7     :0            Fri Jun 8 02:09 - crash (00:15)
reboot    system boot  4.10.0-28-generi Fri Jun 8 02:08      still running
gjw       tty7     :0            Fri Jun 8 02:03 - crash (00:05)
reboot    system boot  4.10.0-28-generi Fri Jun 8 02:02      still running
gjw       tty7     :0            Thu Jun 7 05:41 - crash (20:21)
reboot    system boot  4.10.0-28-generi Thu Jun 7 05:40      still running

wtmp begins Thu Jun 7 05:40:40 2018
root@ubuntu:/home/gjw# lastlog | grep gjw
gjw                                     **Never logged in**
```

### 用户对话

write 用户账号 [用户所在终端接口] # 只给一个用户发送信息  
wall "Hello,world." # 广播信息  
mail gjw -s "nice to meet you!" # 发送邮件

### 常用命令

```
gjw@ubuntu:~$ cat /etc/profile | wc
wc [-lwm]      27      97      575
-l: 统计多少行
-w: 统计英文单字个数
-m: 统计总字符个数
```

### 网络通信命令:

ifconfig [-a] 查看IP地址

ping 选项 IP地址

重点: 并不是能ping通, 就说明网络一定没有问题

网络故障排查, 经验:

- (1) ping对方主机: 如果能ping通, 则查看网络延迟和丢包率
- (2) 如果ping不通, 则ping本机地址: 如果能ping通, 说明自己本机的IP地址设置没有问题, 可能是对方主机/网络连接, 网络设备有问题
- (3) 如果不能ping通本机地址, 则ping回环地址127.0.0.1: 如果能ping通, 说明本机的TCP/IP协议没问题; 如果ping不通, 有很多种情况: 改变了ARP的地址; 防火墙; 屏蔽了应答; 等等

reboot 重启

shutdown -h now 关机

僵尸进程: 无法被父进程释放且一直在内存中 (执行完毕) 的进程——<defunct>

**进程查看：**找到最消耗资源的进程，杀死

(1) 查看某一时刻process

**ps -l** : 只查看自己的操作环境bash相关进程

**ps aux** : 查看系统所有进程

(2) 动态查看process

**top** 持续监测，有下面几种排序方式，快捷键

**P**: CPU

**M**: 内存占用

**Q**: 退出

**进程管理：**给予某个进程一个信号signal，去告知该进程，你让它做什么

**kill -signal PID**

**killall -signal 命令名称**

-signal:

1 启动被终止的进程

9 杀死

核心部分——正则表达式 (四剑客) : 以行为单位, 进行" 字符匹配 ", 找到匹配的行, 打印输出

剑客一: **grep** 过滤器

**grep** [-i -n -v] '查找字符串' 一条语句/文件名

-i 忽略大小写

-v 反选

剑客二: **sed** 主要用于替换功能 **sed** 's/要被替换字符串/新的字符串/g' 一条语句/文件名

选项:

-i 将结果写入文件

不加-i, 将结果打印在屏幕上

剑客三: **awk** 主要倾向于将一行分成数个" 字段 "来处理, 适应与小型数据处理

**awk** -F分隔符 '条件类型1 {动作1} 条件类型2 {动作2} ... ..' filename

```
gjjw@ubuntu:~$ last | head -5
gjjw  tty7      :0          Thu Jun  7 18:25   gone - no logout
reboot system boot  4.10.0-28-generi Fri Jun  8 02:24   still running
gjjw  tty7      :0          Fri Jun  8 02:09   - crash (00:15)
reboot system boot  4.10.0-28-generi Fri Jun  8 02:08   still running
gjjw  tty7      :0          Fri Jun  8 02:03   - crash (00:05)
gjjw@ubuntu:~$ last | head -5 | awk '{print $1 "\t" $2}'
gjjw  tty7
reboot system      $0 表示一整行
gjjw  tty7          $1 表示第一列
reboot system      $2 表示第二列
gjjw  tty7
```

```
gjjw@ubuntu:~$ last | head -5 | awk '{print $1 "\t" $2 "\t line = " NR "=" NF }'
gjjw  tty7      line = 1=11
reboot system   line = 2=10
gjjw  tty7      line = 3=10
reboot system   line = 4=10
gjjw  tty7      line = 5=10
```

行号 改行字段总数

以冒号: 为分隔符

```
gjjw@ubuntu:~$ cat /etc/passwd | head -5 | awk -F: '$1=="root" {print $0}'
root:x:0:0:root:/root:/bin/bash
```

如果该行的第一个字段\$1为root, 则打印这行\$0

**cut** 以行为单位, 指定 (分隔符) 进行切割, 取 (指定段)

**cut** -d '分割字符' -f 段范围

-d 分隔符

-f 取指定段

-f 1,4表示取第1, 4字段

-f 1-4表示取第1, 2, 3, 4字段

```
gjjw@ubuntu:~$ cat /etc/passwd | head -5
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
gjjw@ubuntu:~$ cat /etc/passwd | head -5 | cut -d: -f 1-4
root:x:0:0
daemon:x:1:1
bin:x:2:2
sys:x:3:3
sync:x:4:65534
```

对比cut与awk:

**awk**不但能实现**cut**的功能, 还能实现"条件 {执行语句}"的功能, 更加强大