

Linux 总复习参考文档

作者 Beihai

2020 年 12 月 8 日 完成

2020 年 12 月 21 日 更改

2020 年 12 月 23 日 更改

一：Linux 系统的组成

- **Kernel** 内核
- **Shell** 命令行如 bash
- **Applications** 应用程序

● **Kernel**

- 1、管理程序的运行，为程序分派资源，处理程序之间的通信
- 2、管理对文件系统的读写
- 3、把对文件系统的操作映射成对磁盘或其他设备的操作
- 4、管理输入，输出，将设备映射成设备文件
- 5、管理网络

● **Shell**

- 1、命令解释器
- 2、编程语言
- 3、系统的启动过程: init-> login -> passwd -> shell

二：命令集

- **pwd**
(print working directory)
- **ln**
目录中每一对文件名称和索引节点号称为一个链接
 - ◆ 硬链接 (Hard Link)
 - 原文件名和链接文件名都指向相同的物理位址。 (i-node number)
 - 如果删除硬链接文件的源文件，硬链接文件仍然存在，而且保留了原有的内容
 - 不允许普通用户对目录建立硬链接；硬链接不能跨越文件系统
 - (不能跨越不同的分区)
 - 对于一个物理文件可以有多个文件名。
 - ln 用于对一个已经存在的文件再创建一个新的链接而不复制文件的内容。
 - ln 后的链接文件名与物理文件具有相同的权限。
 - 格式：

```
ln [-s] target [link name]
```

■ -s: 符号链接。不加此选项代表硬连接。

■ Target: 链接所对应的源文件

■ Link name: 链接文件名。**如果是符号链接，也可以是目录名。**

□ ls -l 命令可列出一个文件拥有的硬链接数。

□ 在文件的类型中 l 表示链接文件。

◆ 符号链接 (Symbolic Link)

◆ 它的内容是它所链接的文件的**路径名**

◆ 每个链接文件都有各自的索引节点号。

◆ 由于存放的是源文件的路径名，
因而如果删除符号链接文件的源文件，
符号链接文件将无法找到原有资料。

◆ **可用于链接目录**，且符号链接能够跨越文件系统

● date

//显示日期和时间

```
Wed Sep 29 09:58:29 CST 2004
```

//设置日期

```
# date -s 20041229
```

//设置时间

```
# date -s 12: 23: 23
```

```
# date -s "2009-12-12 12:12:12"
```

CST: China Standard Time

● cal

```
cal 1 2000
```

月份 + 年份

```
cal
```

仅显示当前月份

```
cal 21
```

只有一个数字。默认为年。

● who

显示登陆到系统的所有用户的详细信息

```
denise pts/1 Jun 8 07:07
```

```
joe lft/0 Jun 8 08:34
```

```
the user joe, on terminal lft/0, logged in at 8:34 a.m. on  
June 8.
```

● alias

```
alias ls='ls --color=auto'
```

```
sway:~$ alias today='date +"%A, %B %-d, %Y"'
```

```
sway:~$ today
```

星期二, 十月 9, 2018

```
unalias cp 取消别名
```

● whoami

显示当前用户有效帐号

● ls

常用选项说明：

- -a 用于列出目录中的所有文件，包括文件名以“.”开头的隐藏文件
- -l 以长格式列出文件的详细信息：文件的类型、操作权限、链接数、属主名、属组名、字节数以及最近修改时间
- -d 显示目录名而不显示其中的文件。
- -R 递归列出子目录
- -r 以逆序显示文件名列表
- -x 显示时以字母顺序
- **--color** 用不同颜色区分文件类型

--color 选项：用颜色区分文件类型

在用户的主目录下（root 为 /root）有一隐含文件 .bashrc（用 ls -a 可查到），在其中加入 alias ls="--color" 后，运行 #source .bashrc 就可以了。

颜色含义为：

绿色--->可执行文件

蓝色--->目录

红色--->压缩文件

浅蓝色-->连结文件

灰色--->一般文件(没有定义的文件)

颜色自定义可修改/etc/DIR_COLORS

- -F 选项会在显示目录条目时，在目录后加一个/
- 注：正常的显示顺序为按 ASCII 排序方式。
 - 每行的第一个字符表明文件的类型：
 - ◆ d 目录
 - ◆ - 普通文件
 - ◆ l 符号链接文件
 - ◆ b 块设备文件
 - ◆ c 字符设备文件

● **touch**

```
$touch -t 199801151110 file1
```

将 file1 的时间记录改为 1998 年 1 月 15 日 11 点 10 分

● **cat**

cat 命令将文件的文本内容一次全部显示在屏幕上：

- cat [-n] filename
- **-n** : 打印行号
- **-b** : 打印行号,空行不编号
- 此外，cat 命令还经常被用来进行文件的**合并、建立、覆盖和添加**内容等操作：
 - cat filename1 filename2 > filename3 (合并)
 - cat filename1 >> filename2
- \$cat section1.1
- \$cat section1.1 section1.2 section1.3 >section1
- \$cat section1.1>> section1.2

- **more**

分屏显示文件内容。你可以随心所欲地在文本上上下移动

- 常用按键：

- **f** 键或**<Space>**: 进入下一页。
- **b** 键: 前一页
- **<Enter>**: 向下移动一行
- **q** 键或**<Ctrl+C>**: 退出。

- **less**

less 工具也是对文件或其它输出进行分页显示的工具，应该说是 linux 正统查看文件内容的工具，功能极其强大。**less** 的用法比起 **more** 更加的有弹性。在 **more** 的时候，我们并没有办法向前面翻，只能往后面看，但若使用了 **less** 时，就可以使用 [**pageup**] [**pagedown**] 等按键的功能来往前往后翻看文件，更容易用来查看一个文件的内容！除此之外，在 **less** 里头可以拥有更多的搜索功能，不止可以向下搜，也可以向上搜。

- **wc**

该命令能够统计文件中的字符数、单词数和行数。

□ **\$wc [-options] [Filename]**

□ 选项：

- c : 显示字符数。
- l : 显示行数。
- w : 显示单词数。

□ 单词是指由空格或 tabs 分开的最大字串。

□ 若不加任何选项，则会将文件中字符数、单词数和行数全部列出。

□ (**wc** 等价于 **wc -lwc**)

□ 若不给出文件名，则从标准输入中读取。

□ **\$ wc - lwc file1 file2**

```
4 4 33      file1
7 7 52      file2
11 11 85    total
```

\$who | wc -l

\$cat aaa | wc

- **id** 显示用户 id 以及群组 id

- **chmod** 改变权限

语法：

chmod [-cfvR] [--help] [--version] mode file...

mode : 权限设定字串，格式如下：

[ugoa...][[+=][rwxX]...][,...]

+增加权限

-取消权限

=设定唯一权限

- c : 若该文件权限确实已经更改，才显示其更改动作
- f : 若该文件权限无法被更改也不要显示错误讯息
- v : 显示权限变更的详细资料
- R : 对目前目录下的所有文件与子目录进行相同的权限变更
(即以递归的方式逐个变更)
- help : 显示辅助说明
- version : 显示版本

实例：

将文件 file1.txt 设为所有人皆可读取：

```
chmod ugo+r file1.txt
```

将文件 file1.txt 设为所有人皆可读取：

```
chmod a+r file1.txt
```

将文件 file1.txt 与 file2.txt 设为该文件拥有者，
与其所属同一个群体者可写入，但其他以外的人则不可写入：

```
chmod ug+w,o-w file1.txt file2.txt      注意逗号分隔
```

将 ex1.py 设定为只有该文件拥有者可以执行：

```
chmod u+x ex1.py
```

将目前目录下的所有文件与子目录皆设为任何人可读取：

```
chmod -R a+r *
```

此外 chmod 也可以用数字来表示权限如：

```
chmod 777 file
```

语法为：

```
chmod abc file
```

只有文件所有者和超级用户可以修改文件或目录的权限。

● cp

复制文件

□ 复制文件命令 (copy——cp)：

```
cp [-options] src_file dst_file
```

□ 参数：

src_file：源文件或目录

dst_file：目的文件或目录

如果目的文件是一个目录，那么将源文件拷贝到目的目录下
拷贝时，既可以使用相对路径，也可以使用绝对路径。

□ 选项：

-i：在覆盖文件之前提示用户，由用户确认。

-R 或 -r：递归复制目录，即复制相应的目录及其所有子目录。

- 文件-->文件


```
$cp file1 file2
$cp -i file1 file2
Overwrite file2? N
```
- 文件-->目录


```
$cp file1 directory1
$cp file1 /home/test/directory1
```
- 目录-->目录


```
$cp -R props oldprop (目录 props 下的文件拷贝到 oldprop)
$cp props/*.* oldprop (props 目录下带.*的文件复制到 oldprop 目录)
$cp -r letters/thankyou oldletters
```

● **whereis**

文件搜索

- 显示指令的二进制码、原始码与在线手册说明文件的存放目录。
- 格式:


```
whereis [-options] [file]
```
- 选项:
 - b : 只查找二进制文件;
 - m : 只查找在线手册 manual 路径下的文件;
 - s : 只查找原始码文件;
- whereis 找不到某个文件的部分原因可能是这个文件没有存在于任何 whereis 命令搜索的子目录中。 whereis 命令检索的子目录是固定编写在它的程序中的。
- 例子:


```
$ whereis passwd
passwd: /usr/bin/passwd  /etc/passwd
/usr/share/man/man1/passwd.1.bz2
$ whereis -b passwd
passwd: /usr/bin/passwd  /etc/passwd
$ whereis -m passwd
passwd: /usr/share/man/man1/passwd.1.bz2
```

● **grep**

- 允许对文本文件的内容进行模式查找。如果找到匹配模式， grep 显示包该模式的所有行。
- 格式:


```
grep [-option] pattern files
```

pattern 代表一个匹配模式， **files** 指查找针对的文件。
若指定了多个文件名，结果中会在每一行的前面显示文件名。
- **pattern** 匹配模式:

若匹配模式中包含字符串参数，则最好用引号括起，以免引起歧义
- 例子

从 phone1 文件中查找所有含 800 的行

```
$grep 800 phone1  
从以 phone 开头的文件中查找所有含 800 的行  
$grep 800 phone*  
phone1:8008101818 hardware  
phone2:8008105882 software  
在 star 文件中查找包含星号的行  
$grep * star (这样是错的!!!)
```

```
$grep '*' star  
在 grep 命令中输入字符串参数时，最好将其用双引号括起来。在调用模  
式匹配时，应使用单引号。
```

● mv

mv 命令用于对文件进行名称更改或路径迁移。

对文件的迁移，可以是单纯的迁移；也可以是迁移兼更改文件名。

格式：

```
mv [-options] src_file dst_file  
-f 强制执行。直接覆盖已存在的目的文件，不显示覆盖前的询问讯息。  
-i 交互执行。当已存在同名的目标文件名时，在覆盖之前给出提示，  
由用户确认后才予以覆盖。  
$mv aaa bbb          把文件 aaa 改名为 bbb  
$mv aaa /home/paul  把文件 aaa 移动到/home/paul 下  
$ls  
proposal version1  
$mv -i version1 proposal  
overwrite proposal? N  
将 version1 改名为 proposal，并提示用户确认。  
$mv file1 ..
```

把当前目录下的 file1 文件移动到当前目录的父目录下，并改名为 new-filename

```
$mv file1 ../new-filename
```

● rm

删除

□ 删除文件或目录

□ 格式：

```
rm [-options] files
```

□ 选项：

-f：强制执行。

-i：交互执行，在执行删除前提示确认。

-R 或 r：递归的删除目录

□ files 可以是一个文件，也可以多个文件。

当要删除多个文件时，

多个文件名之间要用空格隔开。

- **mkdir**

创建目录

```
$mkdir test  
$mkdir -p test1/test2/test3  
(递归创建，即使路径上的文件夹本来不存在)  
$mkdir -m 711 testqq    (指定权限)  
$ ls -l test*  
drwxrwxr-x  2 test  test  4096 Feb  6 20:47 test/  
drwxrwxr-x  3 test  test  4096 Feb  6 20:48 test1/  
drwx--x--x  2 test  test  4096 Feb  6 20:48 testqq/
```

- **ping**

检测主机。使用 ICMP 传输协议，发出要求回应的信息
(信息比较复杂。不会呜呜呜)

- **find**

在指定目录下查找文件

- **-user name:** 按照文件属主来查找文件。
➢ 在/etc 目录下查找文件属主为 uucp 的文件：
 `$find /etc -user uucp`
- **-group name:** 按照文件所属的组来查找文件。
➢ 在/apps 目录下查找属于 accts 用户组的文件：
 `$find /apps -group accts`
- **-type type:** 查找某一类型的文件(b、c、d、f、l)。
 d: 目录文件 f: 普通文件
➢ 为了在当前目录下查找目录类型的文件：
 `$find . -type d`
- **-mtime -n +n n :** 按照文件的更改时间来查找文件：
 • -n 最后的更改时间在 n 天以内。
 • n , +n 更改时间距现在 n 天以上。
 • 最后更改时间在 5 日以内：
 `$find / -mtime -5`
- **-size n[c]:** 查找文件长度为 n 块(block)的文件
带有 c 时表示文件长度以字节计。
➢ 在当前目录下查找文件长度大于 1 M 字节的文件：
 `$find . -size +1000000c`
➢ 为了在当前目录下查找长度超过 10block 的文件 (1 个 block 等于 512 字节)：
 `$find . -size +10`
- 当匹配到一些文件以后，可能希望对其进行某些操作，
这时就可以使用**-exec** 选项。
 `$find . -name "m*" -exec ls -l {} \;`
 `-rw-r--r-- 1/shape/misc`
 `-rw-r--r-- 1/size/medium`

- rw-r--r-- 1/misc
- -exec 后的命令形式:


```
command {} \;
注意{}和\;之间的空格。
```
- find 命令在当前目录中查找所有文件名以 .LOG 结尾、更改时间在 5 日以上的文件，并删除它们。


```
$find . -name "*.LOG" -mtime +5 -ok rm {} \;
```

- find . -name “*.c”**
将.c 结尾的文件全部找出
- find /var/log -type f -perm 644 -exec ls -l {} \;**
查找权限为 644 的文件
- find / -type f -size 0 -exec ls -l {} \;**
查找系统中所有 (/) 长度为 0 的普通文件，并列出其完整路径
- 当匹配到一些文件以后，可能希望对其进行某些操作，这时就可以使用 -exec 选项。
exec 是将查找的范围一次性发送到后面的命令当中，如果查找到的结果较多会导致数据溢出。使用 ls 数据超过 16w，该命令会报错。


```
$find . -name "m*" -exec ls -l {} \;
-rw-r--r-- 1 ... ... ./shape/misc
-rw-r--r-- 1 ... ... ./size/medium
-rw-r--r-- 1 ... ... ./misc
```
- -exec 后的命令形式:


```
command {} \;
注意{}和\;之间的空格。
```
- find 命令在当前目录中查找所有文件名以 .LOG 结尾、更改时间在 5 日以上的文件，并删除它们。


```
$find . -name "*.LOG" -mtime +5 -ok rm {} \;
```
- -name file: 寻找名为 file 的文件，要找的文件名包括在引号中，可以使用通配符。
 - 想要在当前目录及子目录中查找所有的 .txt 文件:


```
$find . -name "*.txt"
```
 - 在当前目录及子目录中查找文件名以任一个大写字母开头的文件:


```
$find . -name "[A-Z]*"
```
 - 想要在/etc 目录中查找文件名以 host 开头的文件:


```
$find /etc -name "host*"
```
 - 想要查找根目录下的文件:


```
$find / -name "*"
```

● sleep

延迟一段时间
sleep 5m (五分钟)

● echo

echo please insert \ (换行符)

```
> diskette  
please insert diskette  
//当你按下‘\’，再按回车，就能换行。  
-e 解释输出转义字符  
-n 不换行输出
```

- **chroot**

更换根目录

- **vi**

□ vi 是一个较大的 Linux 命令，在启动的时候也有它自己的选项和参数

□ 基本语法：

 vi [-options] [+n] [file]

□ 常用选项：

 -r 恢复系统突然崩溃时正在编辑的文件。

 -R 以只读方式打开文件

 +[n] 指明进入 vi 后直接位于文件的第 n 行，如果只有“+”而不指定 n，则光标位于文本的最后一行

 如果该文件不存在，会自动建立新文件。

 □ i (当前光标位置插入)

 □ a (光标后一个位置插入)

 □ o (光标下一行插入新的行)

 □ 命令模式下：最简单的方式是按键盘的上、下、左、右方向键

 nG 跳到第 n 行

 G 跳到最后一行

 0 跳到行首

 \$ 跳到行尾

 ctrl+b 前移一页

 ctrl+f 后移一页

➤ 底行模式下：

 - :n 将光标移到第 n 行

 - 可以规定命令操作的行号范围

 ▪ 数值 -- 行号；

 ▪ . -- 光标所在行的行号；

 ▪ \$ -- 正文最后一行的行号；

➤ 例：

 ▪ :..+5 将光标移动到当前行往下的第 5 行。

 ▪ :345 将光标移到第 345 行

➤ 正向搜索 (/)：从光标所在位置起向文件末尾方向搜索。

➤ 反向搜索 (?)：从光标所在位置起向文件开头方向搜索。

➤ 搜索得到结果后，可以使用重复命令 n 或 N 沿着相同或相反的方向重复上一次的搜索

/str1 正向搜索字符串 str1

?str2 反向搜索字符串 str2

n 继续搜索，找出 str1 字符串下次出现的位置

N 反方向搜索

字符串替换 (substitute)

➤ 语法:

[n1, n2]s/str1/str2/[g][c]

在第 n1 行到 n2 行的范围内将字符串 str1 用 str2 替换:

c 每次替换都由用户确认

g 对行中搜索字符串的每次出现进行替换。

不加 g 只对行中搜索字符串的首次出现进行替换;

➤ 例子:

:s/str1/str2/

用字符串 str2 替换该行中首次出现的字符串 str1

:s/str1/str2/g

用字符串 str2 替换该行中所有出现的字符串 str1

:.,\$s/str1/str2/g

用字符串 str2 替换正文当前行到末尾所有出现的字符串 str1

:1,\$s/str1/str2/g

用字符串 str2 替换正文中所有出现的字符串 str1

Vi 编辑器:撤销和显示当前状态(命令模式)

• 撤销上一命令结果: u

• Ctrl-g 命令显示当前编辑文本的状态，
包括文本共有多少行、文件名以及目前光标停在多少行。

➤ 例子: Ctrl-g

:w file 将编辑的内容写入 file 文件

➤ 用户不用退出 vi 就可以运行任何 Linux 命令.

:! command

执行完后,再按一下<Enter>键回到命令方式

Vim 和 gvim 的高级特色

• Vim 代表 Vi IMproved, 如同其名称所暗示的那样, Vim 作为标准 UNIX 系统 vi 编辑器的提高版而存在。 Vim 除提供和 vi 编辑器一样强大的功能外, 还提供有多级恢复、命令行历史以及命令及文件名补全等功能。

• gvim 是 vi 的 X Window 版本, 该版本支持鼠标选中, 一些高级的光标移动功能, 并且带有菜单和工具按钮。

● top

实时显示 process 的动态

用法: top [-] [d delay] [q] ...

top -n 2

//表示更新两次之后终止更新显示

top -d 3

```
//更新周期为 3 秒  
top -p 139  
//显示进程号为 139 的进程信息，CPU、内存占用率等  
□ 交互命令  
[Space] 立即刷新显示。  
d 设置刷新进程的时间间隔，你会被提示输入一个数（秒）  
q 退出 top 命令。  
k 杀死某进程。你会被提示输入进程 ID 以及要发  
送给它的信号。  
h 显示帮助屏幕  
n 显示的进程数量。你会被提示输入数量。  
u 选择用户。  
M 按内存用量排序。  
P 按 CPU 用量排序。
```

- **wget**

下载文件的命令。（比较复杂。不会呜呜呜）

- **dmesg**

显示开机信息

- **rmdir**

删除目录

对目录必须有写权限

- 被删除的目录是个空目录，否则只能尝试-p 选项
- 格式： rmdir [-p] directory

-p：多层次的空目录删除。由指定目录的最底层开始，逐层尝试删除空目录，当碰到非空目录时便停止删除的动作。

- **locate**

查找符合条件的文档，他会去保存文档和目录名称的数据库内，查找合乎范
样式条件的文档或目录。

//查找 passwd 文件

locate passwd

//搜索 etc 目录下所有以 sh 开头的文件

locate /etc/sh

//忽略大小写搜索当前用户目录下所有以 r 开头的文件：

locate -i ~r

- 格式：

locate [file]

- locate 命令使用的是一个文件名数据库，只要我们在这个数据库
文件中进行检索就能得到所需信息，而不必去搜索整个硬盘驱动器。
- 用 locate 命令查找文件位置又比 whereis 命令速度更快
- 限制：数据库的更新是每星期执行一次，当我们新建了一个文件，
但还在数据库没有更新之前就搜索该文件，locate 会显示“找不到”
- 可执行 updatedb 手工更新数据库

- **mail** 发送 E-mail

```
mail <username>
```

```
$ mail joe
Subject: meeting
Don't forget about the meeting today!
<ctrl-d>
Cc: <enter>
```



- **# wall**

```
# wall the system will be shutdown from 10pm today
```

此时所有在线用户会收到以下信息：

```
Broadcast message from root (pts/0) (Wed Sep 14 11:16:16 2005):
```

```
the system will be shutdown from 10pm today
```

- **write**

若一用户**在线**，可用 write 命令发送信息。

```
$write <username>
```

```
$write joe
```

```
$write sam  
按<ctrl-d> 来结束会话.
```

EOF 说明另一个人结束会话
这一方也需按<ctrl-d> 来结束会话

- **msg**

决定是否接受 write、wall 命令发送的消息，
但对 root 发送的消息不起作用。

```
$ mesg n    拒绝消息  
$ mesg y    允许接受
```

- **man**

是一个帮助命令，可以通过这个命令显示需要命令信息。

man man

man who

man -a who 显示所有手册页

man -k keyword 在手册页标题数据库中搜索 keyword 关键字并显示包含匹配标题的列表。

<Space bar> :每次向后翻一页

<Enter> :每次向后移动一行

 : 每次向前翻一页

<f> : 每次向后翻一页

<ctrl-c> or q: 退出 man 命令.

手册信息主要包含：

NAME : 标题名称

SYNOPSIS : 命令的语法描述.

DESCRIPTION : 命令可用选项描

[] : [] 内的内容为可选项

a | b : 要么为 a ,要么为 b.

{ } : 强制选项

man 后面可以接：

- 1 用户命令
- 2 系统调用
- 3 例程， 即库函数。
- 4 设备， 即 /dev 目录下的特殊文件。
- 5 文件格式描述， 例如 /etc/passwd.
- 6 游戏
- 7 杂项， 例如宏命令包、惯例等。
- 8 系统管理员工具， 只能由 root 启动。
- 9 其他 (Linux 特定的)， 用来存放内核例行程序的文档。

- **help**

```
#help  
#help if  
#help set  
#help help
```

- **w** 显示目前登入系统的用户信息
- **su** 变更使用者 (switch user)
- **last** 显示用户最近的登录信息
使用权限：所有使用者
- **du** 显示目录或文件的大小

Linux **du** (英文全拼: *disk usage*) 命令用于显示目录或文件的大小。

du 会显示指定的目录或文件所占用的磁盘空间。

语法

du

```
[-abcDhHklmsSx][-L <符号连接>][-X <文件>]  
[--block-size][--exclude=<目录或文件>]  
[--max-depth=<目录层数>][--help][--version][目录或文件]
```

参数说明：

-a 或 **-all** 显示目录中个别文件的大小。

-b 或 **-bytes** 显示目录或文件大小时，以 byte 为单位。

-c 或 **--total** 除了显示个别目录或文件的大小外，同时也显示所有目录或文件的总和。

- **df** 显示磁盘使用情况

Linux **df** (英文全拼: *disk free*) 命令用于显示目前在 Linux 系统上的文件系统磁盘使用情况统计。

语法

df [选项]... [FILE]...

文件**-a**, **--all** 包含所有的具有 0 Blocks 的文件系统

文件**--block-size={SIZE}** 使用 {SIZE} 大小的 Blocks

文件**-h**, **--human-readable** 使用人类可读的格式(预设值是不加这个选项的...)

- **tail** 显示文件末尾

- **head**

查看文件开头部分的内容。

```
head -n (行数) qwe.txt  
-c 20 (显示前 20 字节)  
-v 显示文件名
```

-q 隐藏文件名

三：Linux 常用操作

● 常用按键

<backspace> 删除
<ctrl-c> 中断当前的命令并返回 Shell.
<ctrl-d> 中断当前的通信或从文件中退出。
<ctrl-u> 删除整行.

- 命令是大小写敏感的，不同于 Windows.
- 命令、选项和参数之间必须用空格格开
- 若命令在一行内写不完，可在行尾加\，再按下回车，再接着写。
- > 的意思是覆盖，>>的意思是追加。
- 输入输出重定向
 - 若要指定命令的标准输入、输出或错误输出，而不是使用缺省值，就需要使用文件重定向机制。
 - 重定向格式：
Command operator file
 - 输入重定向 <
\$mail team01 < letter
 - \$wc < /etc/passwd
 - 输出重定向 >
 - I. \$ls > directory.out
 - \$cat file1 file2 > file3
 - I. \$ls *.doc >> directory.out

● 命名：

- Linux 文件名的最大长度为 256 个字符
- 通常由字母、数字、“.”（点号）、“_”（下划线）或“-”（减号）组成，文件名中不能含有“/”符号。
- 避免使用具有特别意义的字符：
? * @ # \$ & () \ | ; ' " ` < > [] 等
(大括号{}不是?)
- 隐藏文件一般是以“.”符号开头；
- 在文件名中的空格或制表符，在引用文件时必须用引号将其括起来；
- 避免使用 ‘+’ 和 ‘-’ 符号作为文件名的第一个字符
- 大小写敏感

● 三种基本的文件类型

- 普通文件 (regular file)
 - 二进制文件
 - 文本文件

- 目录文件 (directory)
 - 是一个包含文件的容器，用于存放目录中文件列表信息。
- 设备文件
 - 块设备：
 - ◆ 以块为单位进行随机存取。
 - ◆ 常见块设备：软盘、光盘、硬盘。
 - 字符设备：
 - ◆ 以单个字符为单位进行顺序存取。
 - ◆ 常见的字符设备：打印机、终端、键盘、鼠标
- linux 中
 - >表示覆盖原文件内容（文件的日期也会自动更新）
 - >>表示追加内容（会另起一行，文件的日期也会自动更新）。
- 使用 echo \$HOME 可以查看自己的主目录

四、Linux 常见术语

tty 终端机代号

Linux 系统中的 /etc/passwd 文件，是系统**用户配置文件**，存储了系统中所有

用户的基本信息，并且所有用户都可以对此文件执行**读**操作。

真正的密码保存在 /etc/shadow 文件中（下一节做详细介绍）。

五、用户与权限

□ 在设定文件权限时，用字符表示用户类型：

u 文件的所有者

g 同组用户

o 其它用户

a all=ugo, 即所有用户

□ 对文件和目录的三种权限

· 读 (r)

· 写 (w)

· 执行 (x)

□ 十进制表示

· 用三个十进制数字表示文件权限：xyz

· x、y、z 分别代表文件所有者、同组用户和其他用户的文件权限值（一个数字对应一种人）。

· 文件权限值为该类型用户对文件的读、写、执行权限值的累加：

读(r): 4 写(w): 2 执行(x): 1 禁止(-): 0

□ !IMPORTANT 命令执行对文件权限的限制

➢ 在移动文件时**不需要被移动文件的权限**，但需其所在**目录具有写权限**。

➢ 在目录下增删文件、子目录时要有**目录的写权限** (**touch, rm, mkdir, rmdir**)

➢ 用 **ls** 列目录要有**目录的读权限**

- 进入目录或将该目录作路径分量时(cd)要有目录的执行权限，故要使用任一个文件，必须有该文件及找到该文件的路径上所有目录分量的执行权限。
- 仅当要打开一个文件时，即执行涉及到文件内容的操作时，才需要文件的许可。

一个目录同时具有读权限和执行权限才可以打开并查看内部文件，而一个目录要有写权限才允许在其中创建其它文件，这是因为目录文件实际保存着该目录里面的文件的列表等信息

六、 Shell

● 环境变量

- 为使 Shell 编程更有效，系统提供了 Shell 环境变量
- Shell 环境变量：
 - 保存路径名、文件名、数字等信息
 - 用户定制用户自己的工作环境
 - 保存有用信息，使系统获知用户相关设置
 - 保存暂时信息
- 环境变量定义
 变量名=值 或 {变量名=值} (无空格!)
 aaa=1234 或 {aaa=1234}
- 显示环境变量：显示单个环境变量取值。

```
$echo $aaa
$echo ${aaa}
```
- 清除环境变量
 使用 unset 命令清除环境变量。

```
$unset 变量名
◆ 清除之后变量的值为空
```
- 显示所有 Shell 环境变量
 使用 set 命令显示所有 Shell 环境变量。

```
$set
set 输出可能很长。查看输出时可以看出 Shell 已经设置了一些用户变量以使工作环境更加容易使用。
```
- 本地环境变量：
 - 只能在当前 Shell 中使用的变量，
 - 不传递给该 Shell 创建的任何子进程
- 全局环境变量：
 - 可以用于所有进程的变量，一般为大写。
 - variable-name=value
`export variable-name`
 全局环境变量设置完成。
 - 显示环境变量

echo 命令：显示单个变量

env 命令：

显示所有环境变量

格式：`$env`

- **Shell** 中有一些预留的环境变量名，他们各有其用途，且不能用作他用，称为嵌入 **Shell** 变量。

- **HOME** 变量：

- 保存用户主目录的完全路径名

```
$echo $HOME
```

- **LOGNAME** 变量：

- 保存登陆用户名

```
$echo $LOGNAME
```

- **PS1** 变量：

- 保存主提示符

- 基本提示符包含 **Shell** 提示符，缺省（系统默认状态的意思）对超级用户为#，其他为\$。可以使用任何符号作提示符。

```
$ PS1="HH->"
```

- **PATH** 变量：

- 保存进行命令或脚本查找的目录顺序，不同的目录路径名之间用冒号分隔开。

```
PATH=/bin:usr/bin:/etc:/home/team01/bin:.
```

- 正确排列这个次序很重要，可以在执行命令时节省时间。
 - 加入新的查找路径：

```
$PATH=$PATH:filelist
```

```
$export PATH
```

- 本地环境变量-变量赋值

- ◆ 常用格式：

```
Variable-name=value
```

```
{Variable-name=value}
```

- ◆ 没有给变量设置值：变量含有空值(不是 null，而是什么都没有)

- ◆ 显式地给环境变量赋空值

```
Variablename=
```

```
Variablename=""
```

```
Variablename=''
```

- 规则：

- 环境变量名只能是字母、数字、下划线的组合，但是数字不能是开头字符。
 - 等号两边不能直接接单独的空格符。
 - 如果变量所取的值中包含空格，必须使用引号将变量取值括起来。

```
$ name=my name
```

```
name: command not found
```

```
$ name='my name'  
$ name="my name"  
$ name=my\ name
```

- **read 指令**

- 格式:

```
read variables
```

- **variables** 可以是多个变量，它们之间用空格隔开
 - 例子：

```
$read var1 var2 var3  
Hello my friends  
(将 var1、var2、var3 的值分别设置为 Hello、my、friend)
```

```
$read var1 var2 var3  
Hello my dear friends  
$echo $var3  
dear friends  
(若读入的内容个数比变量多，则将多出的都赋给最后一个变量)
```

```
$read var1 var2 var3  
Hello friends  
$echo $var3  
$  
(若变量比读入的内容个数多，则多出的变量将设为空值)
```

- **变量的应用**

- 变量值的结合：将变量并排

- echo \$variable-name1\$variable-name2.....
 - echo \${variable-name1}\${variable-name2}.....

- **反引号**

- 反引号

- 反引号括起来的内容被 Shell 解释为命令行
 - 执行时，Shell 首先执行该命令行，并用它的标准输出结果取代整个反引号（包括反引号括起来的部分）。
 - 反引号可以与其它引号结合使用。

- 反引号的嵌套使用

- 内层的反引号要使用反斜线将其转义

```
$whoami  
test  
$ abc=`echo The current user is \`whoami``  
//这里的\被解释了（注意，使用\而不是“ ”）  
$ echo $abc  
The current user is test
```

- **通配符**

- Shell 提供了一套完整的字符串模式匹配规则，或者称为元字符
 - 用户可以在作为命令参数的文件名中包含这些元字符（通配符）构成一个模式串
 - 这样就可以按照所要求的模式来匹配文件名、路径名、字符串等。
 - 常用的三种通配符：
 - * 匹配方括号[]中指定范围内的字符
 - ? 可以使用横杠-来连接两个字母或数字，以表示范围
例：[a-z]
 - [...] 匹配不在指定范围内的任何字符。
例：[!a-z]
 - 三种通配符可以结合使用。

- 列出以 i 或 o 开头的文件名：
`$ls [io]*`
- 列出以 a、b、c、d、m 中任一个字符开头的文件：
`$ls [abcdm]*` 或者 `$ls [a-dm]*`
- 列出所有以 log. 开头、后面跟随任意一位数字、然后可以是任意字符串的文件名：
`$ls log.[0-9]*`
- 列出所有以大写字母开头的文件名：
`$ls [A-Z]*`
- 以 . 开头，最后一个字符可以是除 a 和 z 之外的任意字符的文件名：
`$ls .*[!za]`
- 列出所有以 LPS 开头、其后可以是任何两个字符，后面跟随一个非数字字符、然后是任意字符串的文件名：
`$ls LPS? ? [! 0-9]*`

- # 注释

- 在 Shell 编程或 Linux 的配置文档中，经常要对某些正文行进行注释，以增加程序的可读性。

- 在 Shell 程序中以字符 # 开头的正文行表示注释行。

```
$cat /etc/hosts.allow
#
# hosts.allow      This file describes the names of the
#                  hosts which are allowed to use the
#                  local INET services, as decided by
#                  the '/usr/sbin/tcpd' server.
```

- 位置变量参数

- Shell 在解释用户命令时，将把命令行的第一个字作为命令，而其它字作为参数通过位置变量传递给程序。
- 位置变量参数是在调用 Shell 程序的命令行中按照各自位置决定的变量。
- \$1,...,\$9 分别代表第一,...,九个参数.其中 1-9

是真正的参数名（变量名），“\$”符只是用来标识变量的替换。

- 第九个以后的参数都不会被访问。
- 位置变量\$0 指命令对应的可执行名。
- 示例：(执行一个 Shell 程序 Param)

```
$Param I like very much this film  
$0      1   2   3   4   5   6
```

● 特定变量参数

□ 用于传递 Shell 程序运行时的相关控制信息

□ 含义：

\$# 传递到脚本的参数个数
\$* 所有参数
\$@ 所有参数，每个参数都用双引号括起

\$? 最后一个程序或命令的执行状态，用数字表示：0 表示执行没有错误，其他任何值表明有错误。

\$\$ 脚本运行的当前 shell 的 PID

特定变量参数	
\$cat Param	\$Param I like it
echo " script name: "	script name:
echo \$0	----- 程序(脚本)名称----- /home/test/scripts/ Param
echo "show arguments:"	show arguments:
echo \$*	----- 显示所有的参数----- I like it
echo "my process id:"	my process id:
echo \$\$	----- 执行Param的当前子Shell的PID----- 26306
echo "Did my script go with any errors?"	Did my script go with any errors?
echo \$?----- 0	最后命令执行无误，返回状态为----- 0

● Shell 程序的运行方式

□ Shell 脚本：可以把一组 Shell 命令或 Shell 程序语句组合在一起，放在一个文件或程序(script)，称为 Shell 脚本

□ Shell 脚本的三种运行方式：

- sh 方式
- . 方式
- 赋予执行权限方式

● tips

- 用户必须对文件具有读 & 执行权限

● 标准文件 & 文件描述符

- 在 Shell 中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。

- 文件 文件描述符
- 输入文件 - 标准输入 - **stdin** 0
- 输出文件 - 标准输出 - **stdout** 1
- 错误输出文件 - 标准错误输出 - **stderr** 2
- 标准输入的文件描述符为 0，它是命令的输入，缺省是键盘
也可以是文件或其他命令的输出。
- \$mail team01
- Subject:
- 标准输出的文件描述符为 1，它是命令的输出，缺省是屏幕
也可以是文件。
- \$ls
- aa bb test.txt
- 标准错误输出的文件描述符为 2。这是命令产生错误的输出，
缺省是屏幕，同样也可以是文件。
- 容易混淆
- 例：假设已经建立了文件 filea，但没有建立文件 fileb
- \$cat filea fileb
- This is output from filea.
- cat: cannot open fileb
- ◆ 错误重定向
 - 将命令或程序执行的错误信息放到文件中
 - 两种方式：
 - 一：
 \$cat filea fileb > errfile
 This is output from filea.
 \$cat errfile
 cat: cannot open fileb
 \$cat filea fileb 2> /dev/null
 # dev/null
 是一个特殊的文件，它总是一个空文件。
 这个操作等同于把错误输出删除掉，而不输出。
 - 二：
 \$cat filea fileb 2>> errfile

● 重定向的结合使用

- Command > outfile 2> errfile < infile
 以 infile 为标准输入，
 将标准输出送到 outfile，标准错误输出送到 errfile.
- Command >> outfile 2>> errfile < infile
 以 infile 为标准输入，
 将标准输出追加到 outfile，标准错误输出追加到 errfile.

把 stdout 和 stderr 同时重定向到 outfile

\$Command > outfile 2>&1

- 管道

- 把一个命令在屏幕上的输出传递给另一个命令作为输入。用竖杠 | 表示。

- 一般形式：

命令 1 | 命令 2

\$ls

aa Param set_dir spec_var spec_var2 test

\$ls | wc -w

6

在当前目录中执行文件列表操作，如果没有管道的话，所有文件就会显示出来。当 Shell 看到管道符号以后，就会把所有列出的文件交给管道右边的命令，因此管道的含义正如它的名字所暗示的那样：把信息从一端传送到另外一端。

\$set | more

- 过滤 & 正则表达式

- 当从一个文件或命令输出中抽取或过滤文本时可以使用正则表达式 (R E)。
 - 正则表达式是一些字符串模式的集合，由一些特殊字符或进行模式匹配操作时使用的元字符组成。也可以使用规则字符作为模式中的一部分。
 - 以行为单位，来进行字符串的处理行为
 - 只是一个字符串处理的标准依据，必须借助支持它的工具程序来使用它
 - 很多不同的 Unix 命令都采用正则表达式，如：grep,vi 等

符号	含义
1. ^	只匹配行首，用在字符串前面
2. \$	只匹配行尾，用在字符串后面
3. X*	0 个或多个字符 X
4. .	只匹配任意单字符
5. [字符表]	字符表中的任意字符
6. [^字符表]	任意不在字符表中的字符
7. \	屏蔽其后的特殊字符的特殊含义
8. \{n\}	前导的正则表达式重复 n 次
9. \{min,max\}	前导的正则表达式重复 min~max 次
10. 注意：有些元字符虽然与 Shell 的通配符一样，但意义有所不同	

- 匹配模式举例：

- 1. ^d 行首是 d 的行
 - 2. z\$ 行尾字符为 z
 - 3. ^a\$ 仅有一个字符 a 的行

- 4. `^$` 空行
- 5. `\.` 包含字符`.`的行
- 6. `^w.*s$` 以 w 开始, s 结尾的行
- 7. `[aA]` a or A
- 8. `[a-f]` a 到 f
- 9. `[^0-9]` 非数字
- 10. `[0-9]\{3\}` 包含 3 个数字的行 (重复三次就是包含三个数字的行)

11. `grep pattern1 | pattern2 files :`

显示匹配 pattern1 或 pattern2 的行,

12. `grep pattern1 files | grep pattern2 :`

显示既匹配 pattern1 又匹配 pattern2 的行。

这里还有些用于搜索的特殊符号:

`<` 和 `>` 分别标注单词的开始与结尾。

例如:

`grep man *` 会匹配 ‘Batman’、‘manic’、‘man’等,

`grep \ '<man\ '` * 匹配‘manic’和‘man’，但不是‘Batman’，

`grep \ '<man>\ '` 只匹配‘man’，而不是‘Batman’或‘manic’等其他的字符串。

`\'^'`: 指匹配的字符串在行首,

`\'$\'`: 指匹配的字符串在行尾,

如果您不习惯命令行参数，可以试试图形界面的‘grep’，如 `reXgrep`。这个软件提供 AND、OR、NOT 等语法，还有漂亮的按钮。如果您只是需要更清楚的输出，不妨试试 `fungrep`。

□ 常用的 grep 选项:

- ✓ `-c` 只输出匹配行的计数。
- ✓ `-n` 显示匹配行及行号。
- ✓ `-v` 显示不包含匹配文本的所有行。
- ✓ `-w` 选项默认匹配一个单词

● grep 和正则表达式

```
$grep '^B' phone1
$grep '^[^48]' phone1      (行首符号)
$grep '5$' phone1
$grep '^dh' phone1
$grep '^a.*0$' phone1
$grep '^P...C$' phone1
$grep '4\{2\}' phone1
$grep '\.' phone1
```

● grep 和管道 |

```
$ps -ef | grep 'team01'
$more phone|grep '^B' 等价于 $grep '^B' phone
```

```

$cat sample.txt | grep 'High' | wc -l
→ $grep 'High' sample.txt | wc -l
$ls -l
-rw-rw-r--    1 admin      admin           51 4月 7 14:31 aa
查看所有类型为“目录”的文件:
$ls -l | grep '^d'
查看所有类型为“目录”,且组成员及其他用户对其具有执行权限的文件:
$ls -l | grep '^d....x..x'

```

- 命令组

- 多个命令在一行输入时,用“;”作为命令间分隔符。

```

$ls -R > outfile ; exit
$ls -R > outfile
$exit
$cat /home/mydir/mysubdir/mydata \
> /home/yourdir/yourdata

```

Shell 关于做题的其他知识点

- && 和 ||

- 命令 1 && 命令 2

只有命令 1 为真(即成功被执行,返回 0),命令 2 才能够被执行。

```
$ ls s* && rm s*
```

- 命令 1 || 命令 2

如果命令 1 执行失败,那么就执行命令 2。

```
$cd /dir1 || echo cannot change to /dir1
```

- 算数表达式

- 若要计算表达式 $2*i+1$ 的值?

```
$i=2
```

```
$echo 2*$i+1
```

- 如何进行变量的整数运算?

```
$echo $((2*i+1))
```

格式为两层括号,外加\$,里面变量不用加\$。

- \$ ((exp))

- 算数扩展提供变量的整数运算机制

- Shell 的内建命令,老版本的 Shell 可能不支持

- 形式:

```
$((expression))
```

- expression: 算术表达式,由变量和运算符组成,
- 运算符源于 c 语言。

□ 用法:

- 显示输出:

```
echo $((expression))
```

- 变量赋值:

```
variable=$((expression))
```

□ \$((expression)) 举例:

- 计算表达式 $2*i+1$ 的值

```
$i=2
```

```
$echo $((2*i+1))
```

算术扩展中包含的只有变量、运算符和常数。

- \$echo \$((2*(i+1)))

计算 $2*(i+1)$ 的值: expression 内可以使用括号来强制分组

- \$echo \$((2 *(i +1)))

在\$((expression))的双小括号内, 空格可以任意添加

表达式内的变量若未定义, 则当作其值为 0

算术扩展可以用来判断真假 (真为 1 ; 假为 0)

● expr

□ expr 用于进行整数运算

□ 格式:

expr 表达式

- 表达式由操作数和运算符组成。操作数一般是整数, 也可以是字符串。
- 表达式的各部分(操作数和运算符之间)必须以空格分隔。
- 表达式中某些对 shell 有特殊意义的字符必须转义。

□ 运算符可为:

+ 加法

- 减法

* 乘法

/ 整除

% 求模(余数)

= 相等

□ \$expr 6 + 3 //9

□ \$var1=6

□ \$var2=3e

□ \$expr var1 / var2 //expr: non-numeric argument

□ \$expr \$var1 / \$var2 //2

□ \$expr \$var1 * \$var2 //expr: syntax error

□ \$expr \$var1 * \$var2 注意, \$ 千万别忘了加

- 多个算术表达式可以组合在一起

```
$expr 5 + 7 / 3
```

- 还可以使用反引号改变计算次序:

```
$expr `expr 5 + 7` / 3
```

- expr 命令一般用于整数值, 此外, 还可用于字符串测试。

```
$s1="hello"
```

```
$expr $s1 = "hello" //1 (显示真假值)
```

!注意，expr 中括号是不可用的。请用`。

```
expr 3 * (2 + 1)
bash: syntax error near unexpected token `('
```

- **test**

- 用于测试一种或几种条件
- 格式：(下面两种形式等效，都是 test)

```
test expression
[ expression ]
```
- 使用方括号时，要注意在条件**两边加上空格**。
涉及到变量值（\$variable）的比较时最好把变量取值用**双引号括起来**，以免 Shell 误解变量值。
- 测试结果反映在**退出状态**中，而不是直接显示输出：
 条件为真 => 退出状态为 0
 条件为假 => 退出状态非 0 //**注意！条件为 true 是 0**
- 可测试的条件分为 4 类：
 - 测试两个字符串之间的关系
 - 测试两个整数之间关系
 - 测试文件是否存在或是否具有某种状态或属性。
 - 测试多个条件的与(and)或(or)组合。
 - 分别对应 4 种不同的操作符：
 - 字符串操作符
 - 整数操作符
 - 文件操作符
 - 逻辑操作符
- ◆ 一、字符串比较
- ◆ 涉及到变量值比较时最好把变量值（\$variable）用双引号括起来
- 例子：

```
$str1=abcd
$str2="abcd "
$test "$str1"
[$ "$str1" ]
$echo $?     //测试$str1是否非空 (非空则 0)
//0
$test "$str1" = "$str2"
[$ "$str1" = "$str2" ]
$echo $?     //判断是否相等
//1
```

- ◆ 二、整数操作符

- int1 -eq int2 数值相等
- int1 **-ne** int2 数值不等
- int1 -lt int2 int1 < int2
- int1 -gt int2 int1 > int2
- int1 -le int2 int1 <= int2
- int1 -ge int2 int1 >= int2

- ◆ 当 `string` 和 `int` 比较时，看操作符。

例子：整数操作符与字符串操作符

```
$str1=1234  
$str2=01234  
[$ "$str1" = "$str2" ]  
$echo $?          //1 (字符串比较)  
[$ "$str1" -eq "$str2" ]  
$echo $?          //0 (数值比较)
```

- 文件操作符：测试文件状态

- `-e file` 文件 `file` 存在
- `-d file` 文件 `file` 是一个目录
- `-f file` 文件 `file` 是一个普通文件
- `-s file` 文件 `file` 大小不为 0
- `-r file` 可读
- `-w file` 可写
- `-x file` 可执行

- ◆ 逻辑操作符：测试多个条件的与(`and`)或(`or`)组合

- `-a` 逻辑与
- `-o` 逻辑或
- `!` 逻辑非

- `until` , `while` 等

语法比较简单，不再赘述。

Linux 进程

- 概念：一个进程是一个程序的一次执行的过程。
- 操作系统通过进程来控制对 CPU 和其他系统资源的访问，并且使用进程来决定在 CPU 上运行哪个程序、运行多久。
- Linux 系统的一个重要特点：可以同时启动多个进程。
- 进程和程序不同：
 - 程序是静态的，是保存在磁盘上的可执行代码和数据的集合；
 - 进程是动态的，是 Linux 系统的基本调度单位。
- 父进程和子进程
 - 一个进程创建新进程称为创建了子进程(`child process`)。
 - 创建子进程的进程称为父进程。
- 进程号
 - `PID: Process Identity number`。一个 PID 唯一地标识一个进程。
 - `PPID: Parent Process ID`。进程的父进程号。

ps 命令

- 用于列出当前进程清单
- 可以用它来确定有哪些进程正在运行以及运行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多的资源等等。

- 它显示的进程列表是一个静态列表，也就是说，这个列表是在我们启动这个命令时正在运行的进程的快照。

- 格式：

`ps [-options]`

不跟任何选项表示查看系统中属于自己的进程。

- 选项：

`-e`: 显示所有进程。

`-f`: 全格式(full list)。

`-l`: 长格式。

`-w`: 宽输出。

`-a`: 显示所有用户进程。

`-u`: 打印用户格式，显示用户名和进程起止时间。

`-x`: 显示与终端无关的所有进程。

- 常用组合选项：

`-ef`: 察看当前系统中运行的所有进程

`aux`: 显示系统中所有用户进程及其所有者，并显示详细的进程信息

- 例子：

显示简单的进程信息可直接用 `ps` 命令，不加任何参数。

```
$ps
```

PID	TTY	TIME	CMD
-----	-----	------	-----

17069	pts/2	00:00:00	bash
-------	-------	----------	------

18503	pts/2	00:00:00	ps
-------	-------	----------	----

TIME: 进程占用的 cpu 时间

CMD: 被执行的命令行

linux 上进程有 5 种状态：

1. 运行 R(正在运行或在运行队列中等待)
2. 中断 S(休眠中，受阻，在等待某个条件的形成或接受到信号)
3. 不可中断 D(收到信号不唤醒和不可运行，进程必须等待直到有中断发生)
4. 僵死 Z(进程已终止，但进程描述符存在，直到父进程调用 `wait4()` 系统调用后释放)
5. 停止 T(进程收到 `SIGSTOP`, `SIGSTP`, `SIGTIN`, `SIGTOU` 信号后停止运行)

`ps` 工具标识进程的 5 种状态码：

`D` 不可中断 uninterruptible sleep (usually IO)

`R` 运行 runnable (on run queue)

`S` 中断 sleeping

`T` 停止 traced or stopped

`Z` 僵死 a defunct ("zombie") process

jobs 显示后台任务的执行情况

- 用于显示和控制后台正在执行的和被挂起的任务序列。

- 格式 `jobs [-options]`

- 选项：

`-l`: 显示后台任务的进程号与信息。

`-p`: 只显示后台任务的 PID。

- n: 显示上次通知用户后，执行状态有更动的后台任务状态。
- r: 显示执行中的后台任务。
- s: 显示暂停执行的后台任务。

□ 例子：

```
$du -a /user > user.data &
//将/user 目录下的所有文件和目录所占用的磁盘空间情况
//输出到 user.data 文件
```

```
[1] 237
$find / -name core -type f &
[2] 238
$jobs -l
[1] + 237 Running du -a /user > user.data
[2] - 238 Running find / -name core
```

kill

- 向正在执行的进程发送信号
- 进程接受到信号后进行信号对应的相应操作
- 常用信号：
 - 信号 15 (SIGTERM)
 - 终止进程运行
 - 为 kill 命令的缺省信号
 - 经常用于后台进程的终止。
 - 当某个进程占用的 CPU 时间过多，或是某进程已经挂起，可以用这种方法终止其执行。
 - 除 root 用户外，一般用户只能终止属于自己的进程。
 - 信号 09(SIGKILL)
 - 强制终止进程

signal	meaning
01	挂起(hangup) <Ctrl + z>
02	中断(interrupt)
09	强制终止一个进程。
15	(缺省值)正常终止一个进程。

□ 格式：

- ```
kill [-option] [signal] {PID | %job ID}
• signal - 信号。与-s 配合使用。不指定则送出信号 15(TERM)。
• pid - 要终止的进程号。
• job ID - 要终止的进程对应的后台任务编号。
```

□ 选项

- s: 指定需要送出的信号，既可以是信号名也可以是对应数字。

## 进程的挂起和回复

- 进程挂起
  - 前台进程的挂起:  
`<Ctrl+z>`
  - 后台进程的挂起:  
`kill -s 01 {PID|%job ID}`
- 进程恢复
  - 恢复到前台:  
`fg %jobID`
  - 恢复到后台:  
`bg %jobID`

### nohup

- 用户退出系统时，一般来说会结束该用户的所有正在运行的程序
- 如果某些后台程序没有执行完，怎么办？
- **nohup** 命令使**进程在用户退出后仍能继续执行**
- 格式:
 

```
nohup {command|script} &
command: 命令
script: 程序或脚本
{a|b}: a 和 b 中必选一项
```
- 例:
 

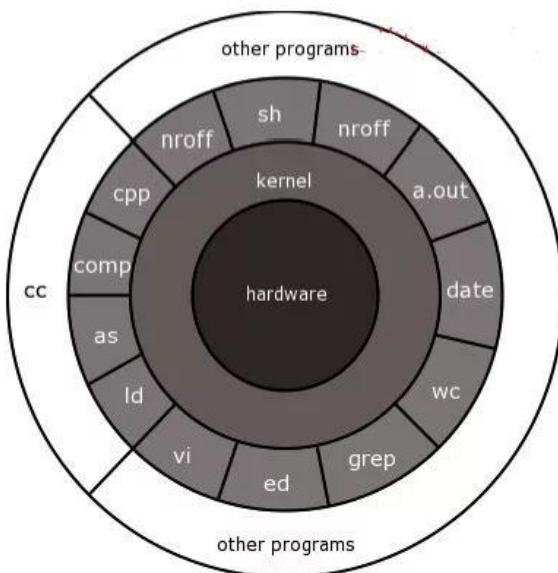
```
$ nohup du -a /user > user.data &
```
- **nohup.out** 文件:
 

存放运行后的所有错误和输出信息。

### 其他进程管理命令

- **pgrep** 找某进程
- **fuser** 显示使用某文件的进程
- **setsid** 另起一个 session 运行
- **disown**
- **screen**

### Linux 结构示意图：



# Linux 一些概念（由梁思睿整理）

## Kernel 内核：

1. 管理程序运行，为系统分配资源，处理程序间的通信
2. 管理对文件系统的读写，把对文件系统的操作映射成对磁盘或其他设备的操作
3. 管理存储器，为程序分配内存，并且管理虚拟内存
4. 管理输入输出，将设备映射成设备文件
5. 管理网络

## Shell

1. 是一个命令解释器，是内核和用户之间的接口
2. 是一个编程语言

## Shell 和内核的关系

Shell 是内核的保护层，对用户屏蔽内核的复杂性。

保护内核以免用户误操作造成危害。

## Linux

标准的 linux 系统都具有一套称为应用程序的程序集，包括文本编辑器、Xwindow、办公套件、Internet 工具和数据库等。

## 文件

文件是 Linux 用来存储信息的基本结构，它是被命名的，存储在某种媒介（如磁盘、光盘和磁带等）上的一种信息集合。

## 文件系统

Linux 文件系统实现数据存储（物理）介质独立性

## 目录文件

目录文件存储一组相关文件的位置、大小等与文件有关的信息，但是并不包含文件的具体内容。

目录文件中的每一项主要表示的是一个文件名（或子目录名）以及文件的索引节点号。

## 索引节点

一个文件的索引节点能够指向该文件内容所在的数据块的位置，除此以外它还记录了该文件的属性。目录文件就是通过 i-node 表与文件之间建立对应关系的。

## 块设备：

以块为单位进行随机存取。如：软盘、光盘、硬盘。

## 字符设备：

以单个字符为单位进行顺序存取。如：打印机、终端、键盘、鼠标。

## 文件系统：

指文件存在的物理空间。

### 树状层次结构

Linux 将分属不同分区的、单独的文件系统整理成一个系统的，总的目录层次结构，即树状结构。

### 挂接

Linux 通过挂接一个文件系统将该新文件系统加入它的文件系统树中，所有的文件系统，不管是什类型，都挂接在文件系统树的一个目录上并且该文件系统之上的文件将会掩盖掉这个挂接目录中原有的内容。这个目录称为挂接目录或挂接点。

### 链接

目录中每一对文件名称名和索引节点号称为一个链接。

### 进程：

一个进程是一个程序的一次执行的过程。

操作系统通过进程来控制对 CPU 和其他系统资源的访问，并且使用进程来决定 CPU 上运行哪个程序、运行多久。

### 程序与进程的不同：

程序：程序是静态的，是保存在磁盘上的可执行代码和数据的集合。

进程：进程是动态的，是 Linux 系统的基本调度单位

## Linux 常见目录说明

**/bin** 存放二进制可执行文件(**ls, cat, mkdir** 等)，常用命令一般都在这里。

**/etc** 存放系统管理和配置文件

**/etc/shadow** 存放用户口令的加密信息

**/home** 存放所有用户文件的根目录，是用户主目录的基点。

比如用户 **user** 的主目录就是**/home/user**，可以用~**user** 表示。

**/usr** 用于存放系统应用程序，比较重要的目录**/usr/local** 本地系统管理员软件安装目录（安装系统级的应用）。这是最庞大的目录，要用到的应用程序和文件几乎都在这个目录。

**/usr/x11r6** 存放 x window 的目录

**/usr/bin** 众多的应用程序

**/usr/sbin** 超级用户的一些管理程序

**/usr/doc** linux 文档

`/usr/include` linux 下开发和编译应用程序所需要的头文件  
`/usr/lib` 常用的动态链接库和软件包的配置文件  
`/usr/man` 帮助文档  
`/usr/src` 源代码,linux 内核的源代码就放在`/usr/src/linux`  
`/usr/local/bin` 本地增加的命令  
`/usr/local/lib` 本地增加的库

**/dev** 存放设备文件

**/mnt** 系统管理员安装临时文件的安装点。  
系统提供这个目录是让用户临时挂载其他的文件系统。

**/tmp** 存放各种临时文件，是公用的临时文件存储点。

**/var** 存放运行时需要改变数据的文件，也是某些大文件的溢出区，  
比方说各种服务的日志文件（系统启动日志等。）等。

**/opt** 额外安装的可选应用程序包所放置的位置。  
一般情况下，我们可以把 `tomcat` 等都安装到这里。

**/boot** 存放用于系统引导时使用的各种文件

**/proc** 虚拟文件系统目录，是系统内存的映射。  
可直接访问这个目录来获取系统信息。

**/root** 超级用户的目录