执行命令

通常来说，执行命令的格式如下所示：

```
command [options] [argument1 [argument2] [……]]
```

上述命令详细说明：

- 一行命令中第一个输入的部分绝对是命令 (command) 或可执行文件 (例如 shell 脚本)；
- 中括号 [] 并不会存在于实际的命令中，表示可选的意思；
- 加入选项 (option) 时，通常选项前面会带 `-` 号，如果使用的是选项的完整名，则选项前带有 `--` 号。如果使用多个短选项，可以合并使用，如 `-abc`；也可以分开使用，中间用空格隔开，如 `-a -b -c`；
- 加入参数 (argument) 时，参数可以时选项的参数，或者命令的参数；
- 命令、选项、参数等这几个东西之间用空格隔开，无论空几个空格，shell 都视为一格；
- 按下回车后，命令立即执行；如果命令太长需要换行，可以用 `\` 来转义回车键；
- 命令区分大小写。

- pwd

作用
打印当前工作目录的绝对路径。

语法

```
pwd [option]
```

选项（option）

- **-L, --logical**　默认选项，显示当前目录的逻辑地址；（**e.g.2**）
- **-P, --physical**　显示当前目录的实际物理地址。（**e.g.3**）

样例

- **e.g.1** 显示当前目录。

```
[root@rudder home]# pwd
/home
```

- **e.g.2** 当前目录是软连接，显示逻辑地址 (软连接的绝对路径)。

```
# 方法一
vagrant@vagrant:~/test/my-site_slink$ pwd
/home/vagrant/test/my-site_slink

# 方法二
vagrant@vagrant:~/test/my-site_slink$ pwd -L
/home/vagrant/test/my-site_slink
```

> `my-site_slink` 是一个软连接，实际指向的目录为 `./project/my-site`

- **e.g.3** 当前目录是软连接，显示实际物理地址 (软连接实际指向目录的却对路径)。

```
vagrant@vagrant:~/test/my-site_slink$ pwd -P
/home/vagrant/test/project/my-site
```

- ls

作用
列出指定目录下的文件。

语法

```
ls [options] [argument1 [argument2] [……]]
```

选项（option）

- **-a, --all** 列出所有文件，甚至包括以圆点开头的隐藏文件；（**e.g.2**）
- **-d, --directory** 列出当前目录下的目录；（**e.g.3**、**e.g.4**）
- **-F, --classify** 在每个所列出的名字后面加上一个指示符。例如，如果名字是目录名，则会加上一个'/'字符；（**e.g.5**）
- **-h, --human-readable** 当以长格式列出时，文件的大小由原来以字节数显示替换为以人更可读的格式显示；（**e.g.6**）
- **-i** 显示文件的 inode 号；
- **-l** 以长格式显示结果；（**e.g.6**）
- **-r, --reverse** 以相反的顺序来显示结果。通常，ls 命令的输出结果按照字母升序排列；
- **-S** 按照文件大小来排序；
- **-t** 按照修改时间来排序。

> 命令 `ll` 等价于 `ls -al`

参数（argument）

- 可以用文件名，如果文件是普通文件则只显示该文件，如果文件是目录文件则显示该目录下的所有文件；
- 可以使用通配符来表示文件名；
- 可以混合使用；
- 参数数量可以是多个。

样例

- **e.g.1** 基本用法

```
# 列出当前目录下的所有非隐藏文件
vagrant@vagrant:~$ ls
foo.txt  test

# 列出指定文件
vagrant@vagrant:~$ ls foo2.txt
foo2.txt

# 列出指定目录中的内容
vagrant@vagrant:~$ ls test
folder1  folder2

# 列出以 .txt 结尾的文件
vagrant@vagrant:~$ ls *.txt
foo2.txt

列出多个目录下的内容
vagrant@vagrant:~$ ls test folder1
folder1:

test:
folder1  folder2
```

- **e.g.2** 列出当前目录下的所有内容。

```
vagrant@vagrant:~$ ls -a
.              .bashrc  .gnupg          .pycharm_helpers
.vbox_version
..             .cache   .local          .ssh                        .viminfo
.bash_history  .config  .mysql_history  .sudo_as_admin_successful  .wget-hsts
.bash_logout   foo.txt  .profile        test
```

- **e.g.3** 列出当前目录下的所有非隐藏目录。

```
vagrant@vagrant:~$ ls -d */
test/
```

- **e.g.4** 列出当前目录下的所有隐藏目录。

```
vagrant@vagrant:~$ ls -d .*/
./  ../  .cache/  .config/  .gnupg/  .local/  .pycharm_helpers/  .ssh/
```

- **e.g.5** 给列出的名字后面增加一个提示符 。

```
vagrant@vagrant:~$ ls -F
foo.txt  test/
```

- **e.g.6** 以长格式打印内容。

```
vagrant@vagrant:~$ ls -l
total 8
-rw-rw-r-- 1 vagrant vagrant  453 May 6 02:44 foo.txt
drwxrwxr-x 2 vagrant vagrant 4096 May 6 02:26 test
```

长格式输出各个字段的含义：

- -rw-rw-r--　　文件权限
- 1　　硬链接数量
- vagrant　　所属用户
- vagrant　　所属用户组
- 453　　文件大小，单位默认 byte
- May 6 02:44　　上次修改文件的日期和时间
- foo.txt　　文件名名
- **e.g.7** 以长格式打印内容时，以人更可读的形式显示文件大小。

```
vagrant@vagrant:~$ ls -lh
total 8.0K
-rw-rw-r-- 1 vagrant vagrant  453 May  6 02:44 foo.txt
drwxrwxr-x 2 vagrant vagrant 4.0K May  6 02:26 test
```

- **e.g.8** 以字母降序打印内容。

```
vagrant@vagrant:~$ ls -r
test  foo.txt
```

- cd

作用
跳转到指定目录。

语法

```
cd [options] [parameter]
```

选项

- **-L** (默认值) 如果要切换到的目标目录是一个符号连接，那么切换到符号连接所在的目录。
- **-P** 如果要切换到的目标目录是一个符号连接，那么切换到它指向的物理位置目录。
- **-** 当前工作目录将被切换到环境变量OLDPWD所表示的目录，也就是前一个工作目录。

参数
指定要切换到的目录。

样例

- 跳转到指定目录。

```
vagrant@vagrant:~$ cd /bin
vagrant@vagrant:/bin$
```

- 跳转到上一层目录。

```
vagrant@vagrant:/bin$ cd ..
vagrant@vagrant:/$
```

- 跳转到上一次所在目录。

```
# 会打印上一次所在目录的路径
vagrant@vagrant:/$ cd -
/bin

# 不会打印上一次所在目录的路径
vagrant@vagrant:/$ cd ${OLDPWD}
```

- 跳转到家目录。

```
# 方法一
vagrant@vagrant:/bin$ cd ~
vagrant@vagrant:~$

# 方法二
vagrant@vagrant:/bin$ cd
vagrant@vagrant:~$
```

- 跳转到指定用户家目录。

```
cd ~vagrant
```

- 把上个命令的参数作为 cd 的参数使用。

```
cd !$
```

- file

作用
查看文件类型。

语法

```
file [options] [file_names]
```

选项

- **-z** 查看压缩文件内部信息。

样例

- 查看指定文件类型。

```
vagrant@vagrant:/vagrant/django-twitter$ file requirements.txt
requirements.txt: ASCII text, with CRLF line terminators
```

- 当前目录下所有文件类型。

```
vagrant@vagrant:/vagrant/django-twitter$ file *
accounts:                          directory
friendships:                       directory
manage.py:                         Python script, ASCII text executable,
with CRLF line terminators
mysql-apt-config_0.8.15-1_all.deb: Debian binary package (format 2.0)
newsfeeds:                         directory
provision.sh:                      Bourne-Again shell script, UTF-8 Unicode
text executable, with CRLF line terminators
README.md:                         ASCII text, with no line terminators
requirements.txt:                  ASCII text, with CRLF line terminators
testing:                           directory
tweets:                            directory
twitter:                           directory
utils:                             directory
Vagrantfile:                       UTF-8 Unicode text
```

- 查看压缩文件内部信息。

```
root@tnak-VirtualBox:/home/tnak# file -z test.bz2
test.bz2: empty (bzip2 compressed data, block size = 900k)
```

- less

作用

浏览文本文件。

语法

```
less [options] [file_name]
```

选项

- **-N** 显示行号。

样例

```
vagrant@vagrant:/vagrant/django-twitter$ less requirements.txt

vagrant@vagrant:/vagrant/django-twitter$ less -N requirements.txt
```

命令内部操作

命令	行为
Page UP or b	向上翻滚一页
Page Down or space	向下翻滚一页
UP Arrow	向上翻滚一行
Down Arrow	向下翻滚一行
G	移动到最后一行
1G or g	移动到开头一行
/charaters	向前查找指定的字符串
n	向前查找下一个出现的字符串，这个字符串是之前所指定查找的
h	显示帮助屏幕
q	退出 less 程序

- cp

作用
将源文件复制给一个具体的文件，或复制到一个已经存在的目录中，也可以同时复制多个源文件到一个指定的目录中。

语法

```
cp [options] [source_file] [target_file]
cp [options] [source_files] [path]
```

> 这两种用法都可以，但是要复制多个文件时，最后一个参数必须是目录。

选项

- **-a** 相当于 -d、-p、-r 选项的集合，这几个选项我们一一介绍；
- **-d** 如果源文件为软链接（对硬链接无效），则复制出的目标文件也为软链接；（**e.g.2**）
- **-i** 询问，如果目标文件已经存在，则会询问是否覆盖；（**e.g.3**）
- **-l** 把目标文件建立为源文件的硬链接文件，而不是复制源文件；（**e.g.4**）
- **-s** 把目标文件建立为源文件的软链接文件，而不是复制源文件；（**e.g.4**）
- **-p** 复制后目标文件保留源文件的属性（包括所有者、所属组、权限和时间）；(**e.g.5**)
- **-r** 递归复制，用于复制目录；（**e.g.6**）
- **-u** 若目标文件比源文件有差异，则使用该选项可以更新目标文件，此选项可用于对文件的升级和备用；
- **-v** 显示详实的命令操作信息。（**e.g.7**）

参数

- **source_file** 表示源文件名；
- **target_file** 表示目标文件名；

- **path** 表示要将文件复制到该目录下。

样例

- **e.g.1** 基本用法。

```
# 当前目录下的文件
vagrant@vagrant:~$ ls -F
foo1.txt  hello.py  test/

# 将源文件不改名复制到指定目录下
cp foo1.txt test/foo1.txt
cp foo1.txt test

# 将源文件改名后复制到指定目录下
cp foo1.txt test/foo2.txt

# 将多个源文件复制到指定目录
cp foo1.txt hello.py test
```

- **e.g.2** 复制软链接文件。

```
# 建立一个测试软链接文件/tmp/cangls_slink
[root@localhost ~]# ln -s /root/cangls /tmp/cangls_slink

# 源文件本身就是一个软链接文件
[root@localhost ~]# ll /tmp/cangls_slink
lrwxrwxrwx 1 root root 12 6月 14 05:53 /tmp/cangls_slink -> /root/cangls

# 复制软链接文件，但是不加"-d"选项
[root@localhost ~]# cp /tmp/cangls_slink /tmp/cangls_t1

# 复制软链接文件，加入"-d"选项
[root@localhost ~]# cp -d /tmp/cangls_slink /tmp/cangls_t2

# 会发现不加"-d"选项，实际复制的是软链接的源文件，而如果加入了"-d"选项，则会复制软链接文件
[root@localhost ~]# ll /tmp/cangls_t1 /tmp/cangls_t2
-rw-r--r-- 1 root root 0 6月 14 05:56 /tmp/cangls_t1
lrwxrwxrwx 1 root root 12 6月 14 05:56 /tmp/cangls_t2-> /root/cangls
```

> 注意，"-d" 选项对硬链接是无效的。

- **e.g.3** 如果目标文件已存在，询问是否覆盖文件。

```
vagrant@vagrant:~$ cp -i foo1.txt test
cp: overwrite 'test/foo1.txt'? y
```

- **e.g.4** 复制为软连接文件和硬链接文件

```
#建立测试文件
[root@localhost ~]# touch bols

#源文件只是一个普通文件，而不是软链接文件
[root@localhost ~]# ll -i bols
```

```
262154 -rw-r--r-- 1 root root 0 6月 14 06:26 bols

#使用"-l" 和"-s"选项复制
[root@localhost ~]# cp -l /root/bols /tmp/bols_h
[root@localhost ~]# cp -s /root/bols /tmp/bols_s

#目标文件 /tmp/bols_h 为源文件的硬链接文件
#目标文件 /tmp/bols_s 为源文件的软链接文件
[root@localhost ~]# ll -i /tmp/bols_h /tmp/bols_s
262154 -rw-r--r-- 2 root root 0 6月 14 06:26  /tmp/bols_h
932113 lrwxrwxrwx 1 root root 10 6月 14 06:27  /tmp/bols_s -> /root/bols
```

> 这两个选项和 "-d" 选项是不同的，"d" 选项要求源文件必须是软链接，目标文件才会复制为软链接；而 "-l" 和 "-s" 选项的源文件只需是普通文件，目标文件就可以直接复制为硬链接和软链接。

- **e.g.5** 保留源文件属性复制

我们发现，在执行复制命令后，目标文件的时间会变成复制命令的执行时间，而不是源文件的时间。例如：

```
# 不适用 -p 选项复制文件
[root@localhost ~]# cp /var/lib/mlocate/mlocate.db /tmp/

# 注意源文件的时间和所属组
[root@localhost ~]# ll /var/lib/mlocate/mlocate.db
-rw-r-----1 root slocate2328027 6月 14 02:08 /var/lib/mlocate/mlocate.db

# 由于复制命令由root用户执行，所以目标文件的所属组为了root，而且时间也变成了复制命令的执行时间
[root@localhost ~]#ll /tmp/mlocate.db
-rw-r----- 1 root root2328027 6 月 14 06:05 /tmp/mlocate.db
```

而当我们执行备份、日志备份的时候，这些文件的时间可能是一个重要的参数，这就需执行 "-p" 选项了。这个选项会保留源文件的属性，包括所有者、所属组和时间。例如：

```
#使用"-p"选项
[root@localhost ~]# cp -p /var/lib/mlocate/mlocate.db /tmp/mlocate.db_2

#源文件和目标文件的所有属性都一致，包括时间
[root@localhost ~]# ll /var/lib/mlocate/mlocate.db /tmp/mlocate.db_2
-rw-r----- root slocate 2328027 6月 14 02:08 /tmp/mlocate.db_2
-rw-r----- root slocate 2328027 6月 14 02:08 /var/lib/mlocate/mlocate.db
```

- **e.g.6** 复制目录

```
# 当前目录下的目录
vagrant@vagrant:~$ ls -d */
folder1/  folder2/  test/

# 将源目录不改名复制指定目录下
cp -r folder1 test

# 将源目录改名复制指定目录下
cp -r folder1 test/folder

# 将多个目录复制到指定目录下
cp -r folder1 folder2 test
```

> "-a" 选项相当于 "-d、-p、-r" 选项，这几个选项我们已经分别讲过了。所以，当我们使用 "-a" 选项时，目标文件和源文件的所有属性都一致，包括源文件的所有者，所属组、时间和软链接性。使用 "-a" 选项来取代 "-d、-p、-r" 选项更加方便。

- **e.g.7** 显示详实的操作信息

```
vagrant@vagrant:~$ cp -rv folder1 folder2 test
'folder1' -> 'test/folder1'
'folder2' -> 'test/folder2'
```

- mv

作用
移动文件，或者给文件重命名。

语法

```
mv [options] [source_file] [target_file]
mv [options] [source_files] [path]
```

> 这两种用法都可以，但是要复制多个文件时，最后一个参数必须是目录。

选项

- **-f** 强制覆盖，如果目标文件已经存在，则不询问，直接强制覆盖（ubuntu18 默认选项）；
- **-i** 交互移动，如果目标文件已经存在，则询问用户是否覆盖；
- **-n** 如果目标文件已经存在，则不会覆盖移动，而且不询问用户；
- **-v** 显示文件或目录的移动过程；
- **-u** 若目标文件已经存在，但两者相比，源文件更新，则会对目标文件进行升级。

样例

- **e.g.1** 基本用法

> 类似 cp 的基本用法，并且 mv 移动目录不用像 cp 命令那样加 -r 参数。

```
# 当前目录下的文件
vagrant@vagrant:~$ ls -F
foo1.txt  hello.py  test/

# 将源文件不改名移动到指定目录下
mv foo1.txt test/foo1.txt
mv foo1.txt test

# 将源文件改名后移动到指定目录下
mv foo1.txt test/foo2.txt

# 将多个源文件移动到指定目录
mv foo1.txt hello.py test
```

- **e.g.2** 改名

```
# 将文件名 foo1.txt 改成 foo2.txt
mv foo1.txt foo2.txt
```

- mkdir

作用
创建目录

语法

```
mkdir [options] [directories]
```

选项

- **-m** 选项用于手动配置所创建目录的权限，而不再使用默认权限；
- **-p** 选项递归创建所有目录。

样例

- e.g.1 基本用法

```
# 创建一个目录
mkdir project

# 创建多个目录
mkdir project1 project2
```

- **e.g.2** 手动配置权限

```
# 创建一个权限为 711 的目录
vagrant@vagrant:~$ mkdir -m 711 test

# 查看 test 目录的权限
vagrant@vagrant:~$ ls -l
total 4
drwx--x--x 2 vagrant vagrant 4096 May  7 00:50 test
```

- **e.g.3** 递归创建目录

```
mkdir -p /test1/test2/test3
```

- rm

作用
移除文件

语法

```
rm [options] [file_names]
```

选项
-f 和 -i 相反，强制删除（force），系统将不再询问，而是直接删除目标文件或目录。
-i 和 -f 相反，在删除文件或目录之前，系统会给出提示信息，可以有效防止不小心删除有用的文件或目录。
-r 递归删除，主要用于删除目录，可删除指定目录及包含的所有内容，包括所有的子目录和文件。

样例

- **e.g.1** 基本用法

```
# 删除一个普通文件
rm foo.txt

# 删除多个普通文件
rm foo1.txt hello.py
```

- **e.g.2** 递归删除

要删除目录，必须使用 -r 参数，并且会删除该目录下的文件（普通文件及目录等）。

```
 rm -r test
```

- ln

作用
用于给文件创建链接

语法

```
ln [options] [source_file] [target_file]
```

选项
-s 建立软链接文件。如果不加 "-s" 选项，则建立硬链接文件；
-f 强制。如果目标文件已经存在，则删除目标文件后再建立链接文件。

样例

- **e.g.1** 创建硬链接

```
[root@localhost ~]# touch cangls
[root@localhost ~]# ln /root/cangls /tmp
#建立硬链接文件，目标文件没有写文件名，会和原名一致
#也就是/tmp/cangls 是硬链接文件
```

- **e.g.2** 创建软连接

```
[root@localhost ~]# touch bols
[root@localhost ~]# In -s /root/bols /tmp
#建立软链接文件
```

> 这里需要注意的是，软链接文件的源文件必须写成绝对路径，而不能写成相对路径（硬链接没有这样的要求）；否则软链接文件会报错。这是初学者非常容易犯的错误。

- alias

作用
定义或显示别名。

- 简化较长的命令。
- 定义一个或多个别名。
- 修改一个或多个已定义别名的值。
- 显示一个或多个已定义别名。
- 显示全部已定义的别名。

语法

```
alias [option] [name[=value] ……]
```

选项

- **-p** 显示全部已定义的别名。

参数

- **name** 指定要（定义、修改、显示）的别名。
- **value** 别名的值。

样例

- e.g.1 显示所有别名

```
alias
alias -p
```

- e.g.2 显示指定别名

```
alias ls
alias ls grep
```

- e.g.3 修改或定义别名

```
alias ls="ls --color=auto"  # 修改
alias less="less -N"        # 定义
```

- cat

作用

连接多个文件并打印到标准输出。

- 显示文件内容，如果没有文件或文件为-则读取标准输入。
- 将多个文件的内容进行连接并打印到标准输出。
- 显示文件内容中的不可见字符（控制字符、换行符、制表符等）。

语法

```
cat [options] [file1 [file2 [……]]]
```

选项

- **-b, --number-nonblank** 只对非空行编号，从1开始编号，覆盖"-n"选项。
- **-n, --number** 对所有行编号，从1开始编号。
- **-s, --squeeze-blank** 压缩连续的空行到一行。
- **-v, --show-nonprinting** 使用"^"和"M-"符号显示控制字符，除了LFD（line feed，即换行符'\n'）和TAB（制表符）。
- **-T, --show-tabs** 使用"^I"表示TAB（制表符）。
- **-E, --show-ends** 在每行的结尾显示'$'字符。
- **-e** 等价于"-vE"组合选项。

- **-t**　等价于"-vT"组合选项。
- **-A, --show-all**　等价于"-vET"组合选项。

样例

- **e.g.1 显示单个文件**

```
cat file
```

- e.g.2 将多个文件连接起来

```
# 方法一
cat file_1 file_2 file_3 > file

# 方法二
cat file_* > file
```

- history

作用
显示或操作历史命令列表。

语法

```
history [-c] [-d offset] [n]
history -anrw [filename]
history -ps arg [arg...]
```

选项

- **-c**　　清空历史列表。
- **-d offset**　根据offset删除记录。如果是正数则表示offset位置的记录，如果为负数则表示从结尾向前offset位置的记录。
- **-a**　　将当前终端的历史记录行添加到历史记录文件。
- **-n**　　将尚未从历史文件中读取的历史行追加到当前历史列表中。
- **-r**　　读取历史文件，并将其内容附加到历史列表中。
- **-w**　　将当前历史记录列表附加到历史记录文件中并且附加它们到历史列表中。
- **-p**　　在每个arg上执行历史记录扩展并在标准输出上显示结果，而不将结果存储在历史记录列表中。
- **-s**　　将每个arg作为单个条目附加到历史记录列表。

样例

- **打印最近使用的10条命令**

```
history 10
```

- **搜索历史命令**

在自己曾经用过的命令中，找出和/usr/bin这一目录相关的：

```
vagrant@vagrant:~$ history | grep /usr/bin
 1013  ls /usr/bin/*
 1017  file $(ls /usr/bin/* 2> /dev/null | grep zip)
 1051  history | grep /usr/bin
```

- **使用历史命令**

从上面搜索历史命令中，我们可以看到我们搜索到的每条命令前都有一个编号，我们可以通过这个编号再次调用指定的命令：

```
!1017  # 执行第1017条命令 file $(ls /usr/bin/* 2> /dev/null | grep zip)
```

历史命令展开

序列	行为
!!	重复最后一次执行的命令。可能按下上箭头按键和 enter 键更容易些。
!number	重复历史列表中第 number 行的命令。
!string	重复最近历史列表中，以这个字符串开头的命令。
!?string	重复最近历史列表中，包含这个字符串的命令。

历史命令

按键	行为
Ctrl-p	移动到上一个历史条目。类似于上箭头按键。
Ctrl-n	移动到下一个历史条目。类似于下箭头按键。
Alt-<	移动到历史列表开头。
Alt->	移动到历史列表结尾，即当前命令行。
Ctrl-r	**反向增量搜索。从当前命令行开始，向上增量搜索。**
Alt-p	反向搜索，非增量搜索。（输入要查找的字符串，按下 Enter来执行搜索）。
Alt-n	向前搜索，非增量。
Ctrl-o	执行历史列表中的当前项，并移到下一个。如果你想要执行历史列表中一系列的命令，这很方便。

> 输入 Ctrl-r来启动增量搜索， 接着输入你要寻找的字。当你找到它以后，你可以敲入 Enter 来执行命令， 或者输入 Ctrl-j，从历史列表中复制这一行到当前命令行。

- id

作用
打印真实以及有效的用户和所在组的信息

语法

```
id [options] [user1 [user2 [……]]]
```

选项

- **-a**　　兼容性选项，没有实际作用。
- **-Z, --context**　只打印进程的安全上下文。
- **-g, --group**　　只打印有效的组ID。
- **-G, --groups**　　打印全部组ID。
- **-u, --user**　　只打印有效的用户ID。
- **-z, --zero**　　使用空字符代替默认的空格来分隔条目。

样例

```
vagrant@vagrant:~$ id
uid=1000(vagrant) gid=1000(vagrant)
groups=1000(vagrant),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lxd),1
13(lpadmin),114(sambashare)
```

- chmod

作用
用来变更文件或目录的权限

语法

```
chmod [options] [mod] [file]
```

选项

- **--reference=RFILE**　使用参考文件或参考目录RFILE的权限来设置目标文件或目录的权限。
- **-R, --recursive**　对目录以及目录下的文件递归执行更改权限操作。

样例

```
# 添加组用户的写权限。
chmod g+w ./test.log
```

```
# 删除其他用户的所有权限。
chmod o= ./test.log
# 使得所有用户都没有写权限。
chmod a-w ./test.log
# 当前用户具有所有权限，组用户有读写权限，其他用户只有读权限。
chmod u=rwx, g=rw, o=r ./test.log
# 等价的八进制数表示：
chmod 754 ./test.log
# 将目录以及目录下的文件都设置为所有用户拥有读写权限。
# 注意，使用'-R'选项一定要保留当前用户的执行和读取权限，否则会报错！
chmod -R a=rw ./testdir/
# 根据其他文件的权限设置文件权限。
chmod --reference=./1.log  ./test.log
```

基础概念

下面这九个字符叫做文件模式，代表着文件所有者、文件组所有者和其他人的读、写和执行权限：

rwxrwxrwx

属性	文件	目录
r	允许打开并读取文件内容。	允许列出目录中的内容，前提是目录必须设置了可执行属性（x）。
w	允许写入文件内容或截断文件。但是不允许对文件进行重命名或删除，重命名或删除是由目录的属性决定的。	允许在目录下新建、删除或重命名文件，前提是目录必须设置了可执行属性（x）。
x	允许将文件作为程序来执行，使用脚本语言编写的程序必须设置为可读才能被执行。	允许进入目录，例如：cd directory。

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

- u符号代表当前用户。
- g符号代表和当前用户在同一个组的用户，以下简称组用户。
- o符号代表其他用户。
- a符号代表所有用户。
- r符号代表读权限以及八进制数4。

- w符号代表写权限以及八进制数2。
- x符号代表执行权限以及八进制数1。
- s符号代表设置权限suid和sgid，使用权限组合u+s设定文件的用户的ID位，g+s设置组用户ID位。
- t符号代表只有目录或文件的所有者才可以删除目录下的文件。
- +符号代表添加目标用户相应的权限。
- -符号代表删除目标用户相应的权限。
- =符号代表添加目标用户相应的权限，删除未提到的权限。

- umask

作用
显示或设置创建文件的权限掩码。

语法

```
umask [options] [mode]
```

选项

- **-p**　当没有参数时指定该选项，执行产生的输出格式可复用为输入；
- **-S**　以符号组合的方式输出创建文件的权限掩码，不使用该选项时以八进制数的形式输出。

样例

```
# 以八进制数的形式输出创建文件的权限掩码。
umask -p
# 执行结果：
umask 0022
# 以符号组合的方式输出创建文件的权限掩码。
umask -S
# 执行结果：
u=rwx,g=rx,o=rx
```

```
# 添加权限：
# 为组用户添加写权限。
umask g+w
# 删除权限：
# 删除其他用户的写、执行权限
umask o-wx
# 赋值权限：
# 赋值全部用户所有权限，等价于umask u=rwx,g=rwx,o=rwx
umask a=rwx
# 清除其他用户的读、写、执行权限。
umask o=
```

- chown

作用
用来变更文件或目录的拥有者或所属群组

语法

```
chown [options] [owner] [:[group]] file
```

选项

- **-h或--no-dereference**　只对符号连接的文件作修改，而不更改其他任何相关文件；
- **-R或--recursive**　递归处理，将指定目录下的所有文件及子目录一并处理；
- **--reference=<参考文件或目录>**　把指定文件或目录的拥有者与所属群组全部设成和参考文件或目录的拥有者与所属群组相同。

样例

参数	结果
bob	把文件所有者从当前属主更改为用户 bob。
bob:users	把文件所有者改为用户 bob，文件用户组改为用户组 users。
:admins	把文件用户组改为组 admins，文件所有者不变。
bob:	文件所有者改为用户 bob，文件用户组改为用户 bob 登录系统时所属的用户组。

- find

作用
用来在指定目录下查找文件。任何位于参数之前的字符串都将被视为欲查找的目录名。如果使用该命令时，不设置任何参数，则find命令将在当前目录下查找子目录与文件。并且将查找到的子目录和文件全部进行显示。

语法

```
find [path] [option1] [argument1] [option2] [argument2] ...
```

参数说明

- **path**　告诉 find 去哪里找东西
- **options**
  - **-name**　按照文件名查找文件 (e.g.1)
  - **-iname**　同上，但忽略大小写 (e.g.1)
  - **-path**　按照路径查找目录或文件 (e.g.2)
  - **-type**　按照文件类型查找文件 (e.g.4)
  - **-maxdepth**　指定文件搜索的最大深度 (e.g.5)

    - **-mindepth** 指定文件搜索的最小深度 (e.g.5)
    - **-amin** 查找在指定时间曾被存取过的文件或目录，单位以分钟计算 (e.g.6)
    - **-atime** 查找在指定时间曾被存取过的文件或目录，单位以天计算 (e.g.6)
    - **-mmin** 查找在指定时间曾被更改过的文件或目录，单位以分钟计算 (e.g.6)
    - **-mtime** 查找在指定时间曾被更改过的文件或目录，单位以天计算 (e.g.6)
    - **-cmin** 查找在指定时间之时被更改过的文件或目录，单位以分钟计算 (e.g.6)
    - **-ctime** 查找在指定时间之时被更改的文件或目录，单位以天计算 (e.g.6)
    - **-size** 查找符合指定的文件大小的文件 (e.g.7)
    - **-perm** 按照文件权限查找文件 (e.g.8)
    - **-user** 根据文件拥有者查找文件 (e.g.8)
    - **-group** 根据文件用户组查找文件 (e.g.8)
    - **-exec** 假设find指令的回传值为True，就执行该指令 (e.g.9)
    - **-ok** 此参数的效果和指定"-exec"类似，但在执行指令之前会先询问用户，若回答"y"或"Y"，则放弃执行命 (e.g.9)
    - **-regex** 指定字符串作为寻找文件或目录的范本样式
- **pattern** 匹配模式，可以用通配符或正则表达式

样例

- **e.g.1** 按照文件名查找文件

```
# 查询当前路径下文件名为 settings.py 的文件
find . -name settings.py

# 查询当前路径下以 ".py" 结尾的文件
find . -iname "*.py"

# 查询当前路径下第一个字符为数字的文件
find . -name "[0-9]*"

# 查询当前路径下以 ".txt" 和 ".pdf" 结尾的文件
find . -name "*.txt" -o -name "*.pdf"
```

- **e.g.2** 按照路径查找文件或目录

```
# 当前目录的结构如下所示
.
├── test1
|   └── folder
|       ├── 1.txt
|       ├── 2.txt
|       ├── 3.txt
|       └── 4.txt
└── test2
    └── folder
        └── hello.py

# 由于没有路径为 folder, 所以没有返回结果
vagrant@vagrant:~$ find . -path "folder"
vagrant@vagrant:~$

# 只返回目录
vagrant@vagrant:~$ find . -path "*folder"
./test1/folder
```

```
./test2/folder

# 返回目录和文件
vagrant@vagrant:~$ find . -path "*folder*"
./.pycharm_helpers/pycharm/_jb_create_folder.py
./test1/folder
./test1/folder/3.txt
./test1/folder/2.txt
./test1/folder/1.txt
./test1/folder/4.txt
./test2/folder
./test2/folder/hello.py

# 只返回文件
vagrant@vagrant:~$ find . -path "*folder/*"
./test1/folder/3.txt
./test1/folder/2.txt
./test1/folder/1.txt
./test1/folder/4.txt
./test2/folder/hello.py
```

- **e.g.3** 否定参数

```
# 找出当前路径下不是以.txt结尾的文件
find . ! -name "*.txt"
```

- **e.g.4** 按照文件类型查找文件。类型参数列表：
  - f 普通文件
  - l 符号连接
  - d 目录
  - c 字符设备
  - b 块设备
  - s 套接字
  - p Fifo

```
# 查找当前路径下的普通文件
find -type f
```

- **e.g.5** 设定文件的搜索深度

```
# 当前目录目录结构如下
# 当前目录为深度1，其子目录 proj1 为深度2，其子目录的子目录深度为3
.
├── depth_1.txt
├── proj1
|   ├── app1
|   |   └── depth_3.txt
|   ├── app2
|   └── depth_2.txt
└── proj2
    ├── app1
    └── app2

# 设置最大的搜索深度为1
```

```
vagrant@vagrant:~/test$ find -maxdepth 1 -type f
./depth_1.txt

# 设置最大的搜索深度为2
vagrant@vagrant:~/test$ find -maxdepth 2 -type f
./proj1/depth_2.txt
./depth_1.txt

# 设置最小的搜索深度为 2
vagrant@vagrant:~/test$ find -mindepth 2 -type f
./proj1/depth_2.txt
./proj1/app1/depth_3.txt

# 设置最小的搜索深度为 3
vagrant@vagrant:~/test$ find -mindepth 3 -type f
./proj1/app1/depth_3.txt
```

- **e.g.6** 根据文件时间戳进行搜索

```
# 搜索最近七天内被访问过的所有文件
find . -type f -atime -7

# 搜索恰好在七天前被访问过的所有文件
find . -type f -atime 7

# 搜索超过七天内被访问过的所有文件
find . -type f -atime +7

# 搜索访问时间超过10分钟的所有文件
find . -type f -amin +10

# 找出修改时间在 file.log 之后的所有文件
find . -type f -newer file.log
```

- **e.g.7** 根据文件大小进行匹配。文件大小单元：
  - b —— 块（512字节）
  - c —— 字节
  - w —— 字（2字节）
  - k —— 千字节
  - M —— 兆字节
  - G —— 吉字节

```
# 搜索大于10KB的文件
find . -type f -size +10k

# 搜索小于10KB的文件
find . -type f -size -10k

# 搜索等于10KB的文件
find . -type f -size 10k
```

- **e.g.8** 根据文件权限/所有权进行匹配

```
# 当前目录下搜索出权限为777的文件
find . -type f -perm 777

# 找出当前目录下权限不是644的 py 文件
find . -type f -name "*.py" ! -perm 644

# 找出当前目录用户tom拥有的所有文件
ind . -type f -user tom

# 找出当前目录用户组sunk拥有的所有文件
find . -type f -group sunk
```

- **e.g.9** 借助 `-exec` 选项于其他命令结合使用
  - `{}` 表示由 `find` 命令找到的内容
  - `\;` 表示命令到这里结束

```
# 找出当前目录下所有 root 的文件，并把所有权更改为用户 tom
find .-type f -user root -exec chown tom {} \;

# 找出自己家目录下所有的.txt文件并删除
find $HOME/. -name "*.txt" -ok rm {} \;

# 查找当前目录下所有.txt文件并把他们拼接起来写入到all.txt文件中
find . -type f -name "*.txt" -exec cat {} \;> /all.txt

# 将30天前的.log文件移动到old目录中
find . -type f -mtime +30 -name "*.log" -exec cp {} old \;
```