Linux awk 命令

Linux 命令大全

AWK 是一种处理文本文件的语言，是一个强大的文本分析工具。

之所以叫 AWK 是因为其取了三位创始人 Alfred Aho，Peter Weinberger, 和 Brian Kernighan 的 Family Name 的首字符。

语法

```
awk [选项参数] 'script' var=value file(s)
或
awk [选项参数] -f scriptfile var=value file(s)
```

选项参数说明：

- -F fs or --field-separator fs
  指定输入文件折分隔符，fs是一个字符串或者是一个正则表达式，如-F:。
- -v var=value or --asign var=value
  赋值一个用户定义变量。
- -f scripfile or --file scriptfile
  从脚本文件中读取awk命令。
- -mf nnn and -mr nnn
  对nnn值设置内在限制，-mf选项限制分配给nnn的最大块数目；-mr选项限制记录的最大数目。这两个功能是Bell实验室版awk的扩展功能，在标准awk中不适用。
- -W compact or --compat, -W traditional or --traditional
  在兼容模式下运行awk。所以gawk的行为和标准的awk完全一样，所有的awk扩展都被忽略。
- -W copyleft or --copyleft, -W copyright or --copyright
  打印简短的版权信息。
- -W help or --help, -W usage or --usage
  打印全部awk选项和每个选项的简短说明。

- -W lint or --lint
  打印不能向传统unix平台移植的结构的警告。
- -W lint-old or --lint-old
  打印关于不能向传统unix平台移植的结构的警告。
- -W posix
  打开兼容模式。但有以下限制，不识别：/x、函数关键字、func、换码序列以及当fs是一个空格时，将新行作为一个域分隔符；操作符**和**=不能代替和=；fflush无效。
- -W re-interval or --re-inerval
  允许间隔正则表达式的使用，参考(grep中的Posix字符类)，如括号表达式[[:alpha:]]。
- -W source program-text or --source program-text
  使用program-text作为源代码，可与-f命令混用。
- -W version or --version
  打印bug报告信息的版本。

基本用法

log.txt文本内容如下：

```
2 this is a test
3 Do you like awk
This's a test
10 There are orange,apple,mongo
```

用法一：

```
awk '{[pattern] action}' {filenames}   # 行匹配语句 awk ''
只能用单引号
```

实例：

```
# 每行按空格或TAB分割，输出文本中的1、4项
 $ awk '{print $1,$4}' log.txt
 ---------------------------------------------
 2 a
```

```
3 like
This's
10 orange,apple,mongo
# 格式化输出
$ awk '{printf "%-8s %-10s\n",$1,$4}' log.txt
---------------------------------------------
2        a
3        like
This's
10       orange,apple,mongo
```

用法二:

```
awk -F  #-F相当于内置变量FS, 指定分割字符
```

实例:

```
# 使用","分割
 $  awk -F, '{print $1,$2}'   log.txt
 ---------------------------------------------
 2 this is a test
 3 Do you like awk
 This's a test
 10 There are orange apple
 # 或者使用内建变量
 $ awk 'BEGIN{FS=","} {print $1,$2}'     log.txt
 ---------------------------------------------
 2 this is a test
 3 Do you like awk
 This's a test
 10 There are orange apple
 # 使用多个分隔符.先使用空格分割，然后对分割结果再使用","分割
 $ awk -F '[ ,]'  '{print $1,$2,$5}'   log.txt
 ---------------------------------------------
 2 this test
 3 Are awk
 This's a
 10 There apple
```

用法三:

```
awk -v  # 设置变量
```

实例:

```
$ awk -va=1 '{print $1,$1+a}' log.txt
-----------------------------------------------
2 3
3 4
This's 1
10 11
$ awk -va=1 -vb=s '{print $1,$1+a,$1b}' log.txt
-----------------------------------------------
2 3 2s
3 4 3s
This's 1 This'ss
10 11 10s
```

用法四:

```
awk -f {awk脚本} {文件名}
```

实例:

```
$ awk -f cal.awk log.txt
```

运算符

运算符	描述
= += -= *= /= %= ^= **=	赋值
?:	C条件表达式
||	逻辑或
&&	逻辑与
~ 和 !~	匹配正则表达式和不匹配正则表达式

运算符	描述
< <= > >= != ==	关系运算符
空格	连接
+ -	加，减
* / %	乘，除与求余
+ - !	一元加，减和逻辑非
^ ***	求幂
++ --	增加或减少，作为前缀或后缀
$	字段引用
in	数组成员

过滤第一列大于2的行

```
$ awk '$1>2' log.txt    #命令
#输出
3 Do you like awk
This's a test
10 There are orange,apple,mongo
```

过滤第一列等于2的行

```
$ awk '$1==2 {print $1,$3}' log.txt    #命令
#输出
2 is
```

过滤第一列大于2并且第二列等于'Are'的行

```
$ awk '$1>2 && $2=="Are" {print $1,$2,$3}' log.txt    #命令
#输出
3 Are you
```

内建变量

变量	描述
$n	当前记录的第n个字段，字段间由FS分隔
$0	完整的输入记录
ARGC	命令行参数的数目
ARGIND	命令行中当前文件的位置(从0开始算)
ARGV	包含命令行参数的数组
CONVFMT	数字转换格式(默认值为%.6g)ENVIRON环境变量关联数组
ERRNO	最后一个系统错误的描述
FIELDWIDTHS	字段宽度列表(用空格键分隔)
FILENAME	当前文件名
FNR	各文件分别计数的行号
FS	字段分隔符(默认是任何空格)
IGNORECASE	如果为真，则进行忽略大小写的匹配
NF	一条记录的字段的数目
NR	已经读出的记录数，就是行号，从1开始
OFMT	数字的输出格式(默认值是%.6g)
OFS	输出字段分隔符，默认值与输入字段分隔符一致。
ORS	输出记录分隔符(默认值是一个换行符)
RLENGTH	由match函数所匹配的字符串的长度
RS	记录分隔符(默认是一个换行符)
RSTART	由match函数所匹配的字符串的第一个位置
SUBSEP	数组下标分隔符(默认值是/034)

```
$ awk 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s
%4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OFS","ORS",
"RS";printf "-----------------------------------------
\n"} {printf "%4s %4s %4s %4s %4s %4s %4s %4s
%4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}'  log.txt
FILENAME ARGC  FNR   FS   NF   NR  OFS  ORS   RS
-----------------------------------------
log.txt    2    1         5    1
log.txt    2    2         5    2
log.txt    2    3         3    3
log.txt    2    4         4    4
$ awk -F\' 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s
%4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OFS","ORS",
"RS";printf "-----------------------------------------
\n"} {printf "%4s %4s %4s %4s %4s %4s %4s %4s
%4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}'  log.txt
FILENAME ARGC  FNR   FS   NF   NR  OFS  ORS   RS
-----------------------------------------
log.txt    2    1    '    1    1
log.txt    2    2    '    1    2
log.txt    2    3    '    2    3
log.txt    2    4    '    1    4
# 输出顺序号 NR, 匹配文本行号
$ awk '{print NR,FNR,$1,$2,$3}' log.txt
-----------------------------------------
1 1 2 this is
2 2 3 Are you
3 3 This's a test
4 4 10 There are
# 指定输出分割符
$  awk '{print $1,$2,$5}' OFS=" $ "  log.txt
-----------------------------------------
2 $ this $ test
3 $ Are $ awk
This's $ a $
10 $ There $
```

使用正则，字符串匹配

```
# 输出第二列包含 "th"，并打印第二列与第四列
$ awk '$2 ~ /th/ {print $2,$4}' log.txt
-----------------------------------------------
this a
```

~ 表示模式开始。// 中是模式。

```
# 输出包含 "re" 的行
$ awk '/re/ ' log.txt
-----------------------------------------------
3 Do you like awk
10 There are orange,apple,mongo
```

忽略大小写

```
$ awk 'BEGIN{IGNORECASE=1} /this/' log.txt
-----------------------------------------------
2 this is a test
This's a test
```

模式取反

```
$ awk '$2 !~ /th/ {print $2,$4}' log.txt
-----------------------------------------------
Are like
a
There orange,apple,mongo
$ awk '!/th/ {print $2,$4}' log.txt
-----------------------------------------------
Are like
a
There orange,apple,mongo
```

awk脚本

关于 awk 脚本，我们需要注意两个关键词 BEGIN 和 END。

- BEGIN{ 这里面放的是执行前的语句 }
- END {这里面放的是处理完所有的行后要执行的语句 }
- {这里面放的是处理每一行时要执行的语句}

假设有这么一个文件（学生成绩表）：

```
$ cat score.txt
Marry   2143 78 84 77
Jack    2321 66 78 45
Tom     2122 48 77 71
Mike    2537 87 97 95
Bob     2415 40 57 62
```

我们的 awk 脚本如下：

```
$ cat cal.awk
#!/bin/awk -f
#运行前
BEGIN {
    math = 0
    english = 0
    computer = 0

    printf "NAME    NO.   MATH  ENGLISH  COMPUTER
TOTAL\n"
    printf "---------------------------------------------
\n"
}
#运行中
{
    math+=$3
    english+=$4
    computer+=$5
    printf "%-6s %-6s %4d %8d %8d %8d\n", $1, $2,
$3,$4,$5, $3+$4+$5
}
```

```
#运行后
END {
    printf "-------------------------------------------
\n"
    printf "  TOTAL:%10d %8d %8d \n", math, english,
computer
    printf "AVERAGE:%10.2f %8.2f %8.2f\n", math/NR,
english/NR, computer/NR
}
```

我们来看一下执行结果：

```
$ awk -f cal.awk score.txt
NAME    NO.   MATH  ENGLISH  COMPUTER   TOTAL
---------------------------------------------
Marry  2143     78       84       77      239
Jack   2321     66       78       45      189
Tom    2122     48       77       71      196
Mike   2537     87       97       95      279
Bob    2415     40       57       62      159
---------------------------------------------
  TOTAL:       319      393      350
AVERAGE:     63.80    78.60    70.00
```

另外一些实例

AWK 的 hello world 程序为：

```
BEGIN { print "Hello, world!" }
```

计算文件大小

```
$ ls -l *.txt | awk '{sum+=$5} END {print sum}'
-----------------------------------------------
666581
```

从文件中找出长度大于 80 的行：

```
awk 'length>80' log.txt
```

打印九九乘法表

```
seq 9 | sed 'H;g' | awk -v RS=''
'{for(i=1;i<=NF;i++)printf("%dx%d=%d%s", i, NR, i*NR,
i==NR?"\n":"\t")}'
```

Linux grep 命令

Linux 命令大全

Linux grep (global regular expression) 命令用于查找文件里符合条件的字符串或正则表达式。

grep 指令用于查找内容包含指定的范本样式的文件，如果发现某文件的内容符合所指定的范本样式，预设 grep 指令会把含有范本样式的那一列显示出来。若不指定任何文件名称，或是所给予的文件名为 **-**，则 grep 指令会从标准输入设备读取数据。

语法

```
grep [options] pattern [files]
或
grep [-abcEFGhHilLnqrsvVwxy][-A<显示行数>][-B<显示列数>][-C<显示列数>][-d<进行动作>][-e<范本样式>][-f<范本文件>][--help][范本样式][文件或目录...]
```

- pattern - 表示要查找的字符串或正则表达式。
- files - 表示要查找的文件名，可以同时查找多个文件，如果省略 files 参数，则默认从标准输入中读取数据。

常用选项：：

- `-i`：忽略大小写进行匹配。
- `-v`：反向查找，只打印不匹配的行。
- `-n`：显示匹配行的行号。
- `-r`：递归查找子目录中的文件。
- `-l`：只打印匹配的文件名。

- `-c`：只打印匹配的行数。

更多参数说明：

- **-a 或 --text** : 不要忽略二进制的数据。
- **-A<显示行数> 或 --after-context=<显示行数>** : 除了显示符合范本样式的那一列之外，并显示该行之后的内容。
- **-b 或 --byte-offset** : 在显示符合样式的那一行之前，标示出该行第一个字符的编号。
- **-B<显示行数> 或 --before-context=<显示行数>** : 除了显示符合样式的那一行之外，并显示该行之前的内容。
- **-c 或 --count** : 计算符合样式的列数。
- **-C<显示行数> 或 --context=<显示行数>或-<显示行数>** : 除了显示符合样式的那一行之外，并显示该行之前后的内容。
- **-d <动作> 或 --directories=<动作>** : 当指定要查找的是目录而非文件时，必须使用这项参数，否则grep指令将回报信息并停止动作。
- **-e<范本样式> 或 --regexp=<范本样式>** : 指定字符串做为查找文件内容的样式。
- **-E 或 --extended-regexp** : 将样式为延伸的正则表达式来使用。
- **-f<规则文件> 或 --file=<规则文件>** : 指定规则文件，其内容含有一个或多个规则样式，让grep查找符合规则条件的文件内容，格式为每行一个规则样式。
- **-F 或 --fixed-regexp** : 将样式视为固定字符串的列表。
- **-G 或 --basic-regexp** : 将样式视为普通的表示法来使用。
- **-h 或 --no-filename** : 在显示符合样式的那一行之前，不标示该行所属的文件名称。
- **-H 或 --with-filename** : 在显示符合样式的那一行之前，表示该行所属的文件名称。
- **-i 或 --ignore-case** : 忽略字符大小写的差别。
- **-l 或 --file-with-matches** : 列出文件内容符合指定的样式的文件名称。
- **-L 或 --files-without-match** : 列出文件内容不符合指定的样式的文件名称。

- **-n 或 --line-number** : 在显示符合样式的那一行之前，标示出该行的列数编号。
- **-o 或 --only-matching** : 只显示匹配PATTERN 部分。
- **-q 或 --quiet或--silent** : 不显示任何信息。
- **-r 或 --recursive** : 此参数的效果和指定"-d recurse"参数相同。
- **-s 或 --no-messages** : 不显示错误信息。
- **-v 或 --invert-match** : 显示不包含匹配文本的所有行。
- **-V 或 --version** : 显示版本信息。
- **-w 或 --word-regexp** : 只显示全字符合的列。
- **-x --line-regexp** : 只显示全列符合的列。
- **-y** : 此参数的效果和指定"-i"参数相同。

 Linux 命令大全

实例

1、在文件 file.txt 中查找字符串 "hello"，并打印匹配的行：

```
grep hello file.txt
```

2、在文件夹 dir 中递归查找所有文件中匹配正则表达式 "pattern" 的行，并打印匹配行所在的文件名和行号：

```
grep -r -n pattern dir/
```

3、在标准输入中查找字符串 "world"，并只打印匹配的行数：

```
echo "hello world" | grep -c world
```

4、在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令：

```
grep test *file
```

结果如下所示：

```
$ grep test test* #查找前缀有“test”的文件包含“test”字符串的文件

testfile1:This a Linux testfile! #列出testfile1 文件中包含test字符的行
testfile_2:This is a linux testfile! #列出testfile_2 文件中包含test字符的行
testfile_2:Linux test #列出testfile_2 文件中包含test字符的行
```

5、以递归的方式查找符合条件的文件。例如，查找指定目录/etc/acpi 及其子目录（如果存在子目录的话）下所有文件中包含字符串"update"的文件，并打印出该字符串所在行的内容，使用的命令为：

```
grep -r update /etc/acpi
```

输出结果如下：

```
$ grep -r update /etc/acpi #以递归的方式查找“etc/acpi”
#下包含“update”的文件
/etc/acpi/ac.d/85-anacron.sh:# (Things like the slocate
updatedb cause a lot of IO.)
Rather than
/etc/acpi/resume.d/85-anacron.sh:# (Things like the
slocate updatedb cause a lot of
IO.) Rather than
/etc/acpi/events/thinkpad-cmos:action=/usr/sbin/thinkpad-
keys--update
```

6、反向查找。前面各个例子是查找并打印出符合条件的行，通过"-v"参数可以打印出不符合条件行的内容。

查找文件名中包含 test 的文件中不包含test 的行，此时，使用的命令为：

```
grep -v test *test*
```

结果如下所示：

```
$ grep-v test* #查找文件名中包含test 的文件中不包含test 的行
testfile1:helLinux!
testfile1:Linis a free Unix-type operating system.
testfile1:Lin
testfile_1:HELLO LINUX!
testfile_1:LINUX IS A FREE UNIX-TYPE OPTERATING SYSTEM.
testfile_1:THIS IS A LINUX TESTFILE!
testfile_2:HELLO LINUX!
testfile_2:Linux is a free unix-type opterating system.
```

Linux sed 命令

Linux 命令大全

Linux sed 命令是利用脚本来处理文本文件。

sed 可依照脚本的指令来处理、编辑文本文件。

Sed 主要用来自动编辑一个或多个文件、简化对文件的反复操作、编写转换程序等。

语法

```
sed [-hnV][-e<script>][-f<script文件>][文本文件]
```

参数说明:

- -e或--expression= 以选项中指定的script来处理输入的文本文件。
- -f<script文件>或--file=<script文件> 以选项中指定的script文件来处理输入的文本文件。
- -h或--help 显示帮助。
- -n或--quiet或--silent 仅显示script处理后的结果。
- -V或--version 显示版本信息。

动作说明:

- a ： 新增， a 的后面可以接字串，而这些字串会在新的一行出现(目前的下一行)～

- c： 取代， c 的后面可以接字串，这些字串可以取代 n1,n2 之间的行！
- d： 删除，因为是删除啊，所以 d 后面通常不接任何东东；
- i： 插入， i 的后面可以接字串，而这些字串会在新的一行出现(目前的上一行)；
- p： 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～
- s： 取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正则表达式！例如 1,20s/old/new/g 就是啦！

实例

我们先创建一个 **testfile** 文件，内容如下：

```
$ cat testfile #查看testfile 中的内容
HELLO LINUX!
Linux is a free unix-type opterating system.
This is a linux testfile!
Linux test
Google
Taobao
Runoob
Tesetfile
Wiki
```

在 **testfile** 文件的第四行后添加一行，并将结果输出到标准输出，在命令行提示符下输入如下命令：

```
sed -e 4a\newLine testfile
```

使用 **sed** 命令后，输出结果如下：

```
$ sed -e 4a\newLine testfile
HELLO LINUX!
Linux is a free unix-type opterating system.
This is a linux testfile!
Linux test
newLine
Google
Taobao
Runoob
Tesetfile
Wiki
```

以行为单位的新增/删除

将 **testfile** 的内容列出并且列印行号，同时，请将第 2~5 行删除!

```
$ nl testfile | sed '2,5d'
     1  HELLO LINUX!
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

sed 的动作为 **2,5d**，那个 **d** 是删除的意思，因为删除了 2-5 行，所以显示的数据就没有 2-5 行了， 另外，原本应该是要下达 sed -e 才对，但没有 -e 也是可以的，同时也要注意的是， sed 后面接的动作，请务必以 **'...'** 两个单引号括住喔!

只要删除第 2 行:

```
$ nl testfile | sed '2d'
     1  HELLO LINUX!
     3  This is a linux testfile!
     4  Linux test
     5  Google
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

要删除第 3 到最后一行：

```
$ nl testfile | sed '3,$d'
     1  HELLO LINUX!
     2  Linux is a free unix-type opterating system.
```

在第二行后(即加在第三行) 加上**drink tea?** 字样：

```
$ nl testfile | sed '2a drink tea'
     1  HELLO LINUX!
     2  Linux is a free unix-type opterating system.
drink tea
     3  This is a linux testfile!
     4  Linux test
     5  Google
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

如果是要在第二行前，命令如下：

```
$ nl testfile | sed '2i drink tea'
     1  HELLO LINUX!
drink tea
     2  Linux is a free unix-type opterating system.
     3  This is a linux testfile!
     4  Linux test
     5  Google
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

如果是要增加两行以上，在第二行后面加入两行字，例如 **Drink tea or ..... 与 drink beer?**

```
$ nl testfile | sed '2a Drink tea or ......\
drink beer ?'
```

```
1  HELLO LINUX!
     2  Linux is a free unix-type opterating system.
Drink tea or ......
drink beer ?
     3  This is a linux testfile!
     4  Linux test
     5  Google
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

每一行之间都必须要以反斜杠 **** 来进行新行标记。上面的例子中，我们可以发现在第一行的最后面就有** ** 存在。

以行为单位的替换与显示

将第 **2-5** 行的内容取代成为 **No 2-5 number** 呢?

```
$ nl testfile | sed '2,5c No 2-5 number'
     1  HELLO LINUX!
No 2-5 number
     6  Taobao
     7  Runoob
     8  Tesetfile
     9  Wiki
```

透过这个方法我们就能够将数据整行取代了。

仅列出 testfile 文件内的第 5-7 行:

```
$ nl testfile | sed -n '5,7p'
     5  Google
     6  Taobao
     7  Runoob
```

可以透过这个 sed 的以行为单位的显示功能， 就能够将某一个文件内的某些行号选择出来显示。

数据的搜寻并显示

搜索 testfile 有 **oo** 关键字的行:

```
$ nl testfile | sed -n '/oo/p'
     5  Google
     7  Runoob
```

如果 root 找到，除了输出所有行，还会输出匹配行。

数据的搜寻并删除

删除 testfile 所有包含 **oo** 的行，其他行输出

```
$ nl testfile | sed  '/oo/d'
     1  HELLO LINUX!
     2  Linux is a free unix-type opterating system.
     3  This is a linux testfile!
     4  Linux test
     6  Taobao
     8  Tesetfile
     9  Wiki
```

数据的搜寻并执行命令

搜索 testfile，找到 **oo** 对应的行，执行后面花括号中的一组命令，每个命令之间用分号分隔，这里把 **oo** 替换为 **kk**，再输出这行：

```
$ nl testfile | sed -n '/oo/{s/oo/kk/;p;q}'
     5  Gkkgle
```

最后的 **q** 是退出。

数据的查找与替换

除了整行的处理模式之外， sed 还可以用行为单位进行部分数据的查找与替换<。

sed 的查找与替换的与 **vi** 命令类似，语法格式如下：

```
sed 's/要被取代的字串/新的字串/g'
```

将 testfile 文件中每行第一次出现的 oo 用字符串 kk 替换，然后将该文件内容输出到标准输出:

```
sed -e 's/oo/kk/' testfile
```

g 标识符表示全局查找替换，使 sed 对文件中所有符合的字符串都被替换，修改后内容会到标准输出，不会修改原文件：

```
sed -e 's/oo/kk/g' testfile
```

选项 **i** 使 sed 修改文件:

```
sed -i 's/oo/kk/g' testfile
```

批量操作当前目录下以 **test** 开头的文件：

```
sed -i 's/oo/kk/g' ./test*
```

接下来我们使用 /sbin/ifconfig 查询 IP：

```
$ /sbin/ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:90:CC:A6:34:84
inet addr:192.168.1.100 Bcast:192.168.1.255
Mask:255.255.255.0
inet6 addr: fe80::290:ccff:fea6:3484/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
.....(以下省略).....
```

本机的 ip 是 192.168.1.100。

将 IP 前面的部分予以删除：

```
$ /sbin/ifconfig eth0 | grep 'inet addr' | sed
's/^.*addr://g'
192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0
```

接下来则是删除后续的部分，即：**192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0**。

将 IP 后面的部分予以删除:

```
$ /sbin/ifconfig eth0 | grep 'inet addr' | sed
's/^.*addr://g' | sed 's/Bcast.*$//g'
192.168.1.100
```

多点编辑

一条 sed 命令，删除 testfile 第三行到末尾的数据，并把 HELLO 替换为 RUNOOB :

```
$ nl testfile | sed -e '3,$d' -e 's/HELLO/RUNOOB/'
     1  RUNOOB LINUX!
     2  Linux is a free unix-type opterating system.
```

-e 表示多点编辑，第一个编辑命令删除 testfile 第三行到末尾的数据，第二条命令搜索 HELLO 替换为 RUNOOB。

直接修改文件内容(危险动作)

sed 可以直接修改文件的内容，不必使用管道命令或数据流重导向！ 不过，由于这个动作会直接修改到原始的文件，所以请你千万不要随便拿系统配置来测试！ 我们还是使用文件 regular_express.txt 文件来测试看看吧!

regular_express.txt 文件内容如下:

```
$ cat regular_express.txt
runoob.
google.
taobao.
facebook.
zhihu-
weibo-
```

利用 sed 将 regular_express.txt 内每一行结尾若为 . 则换成 !

```
$ sed -i 's/\.$/\!/g' regular_express.txt
$ cat regular_express.txt
runoob!
google!
taobao!
facebook!
zhihu-
weibo-
```

:q:q

利用 sed 直接在 regular_express.txt 最后一行加入 **# This is a test**:

```
$ sed -i '$a # This is a test' regular_express.txt
$ cat regular_express.txt
runoob!
google!
taobao!
facebook!
zhihu-
weibo-
# This is a test
```

由於 $ 代表的是最后一行，而 a 的动作是新增，因此该文件最后新增 **# This is a test**!

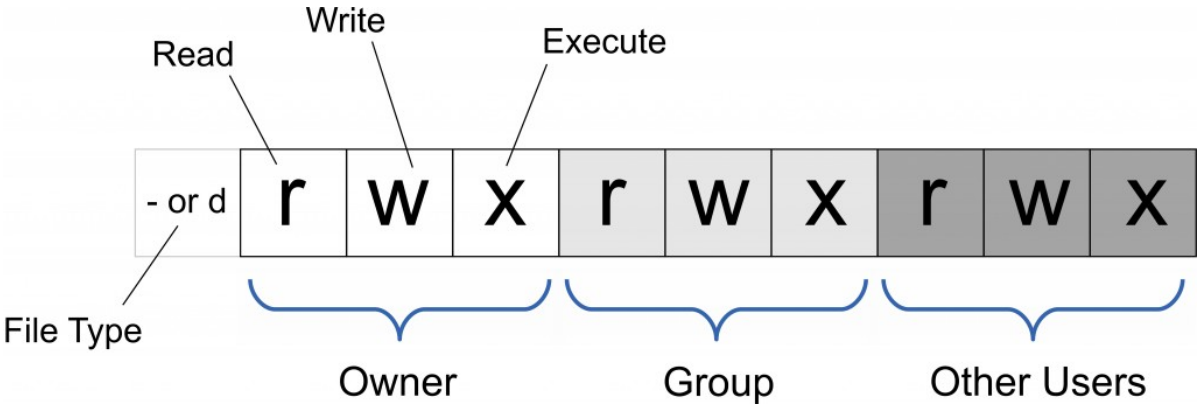
sed 的 **-i** 选项可以直接修改文件内容，这功能非常有帮助！举例来说，如果你有一个 100 万行的文件，你要在第 100 行加某些文字，此时使用 vim 可能会疯掉！因为文件太大了！那怎办？就利用 sed 啊！透过 sed 直接修改/取代的功能，你甚至不需要使用 vim 去修订！

Linux chmod命令

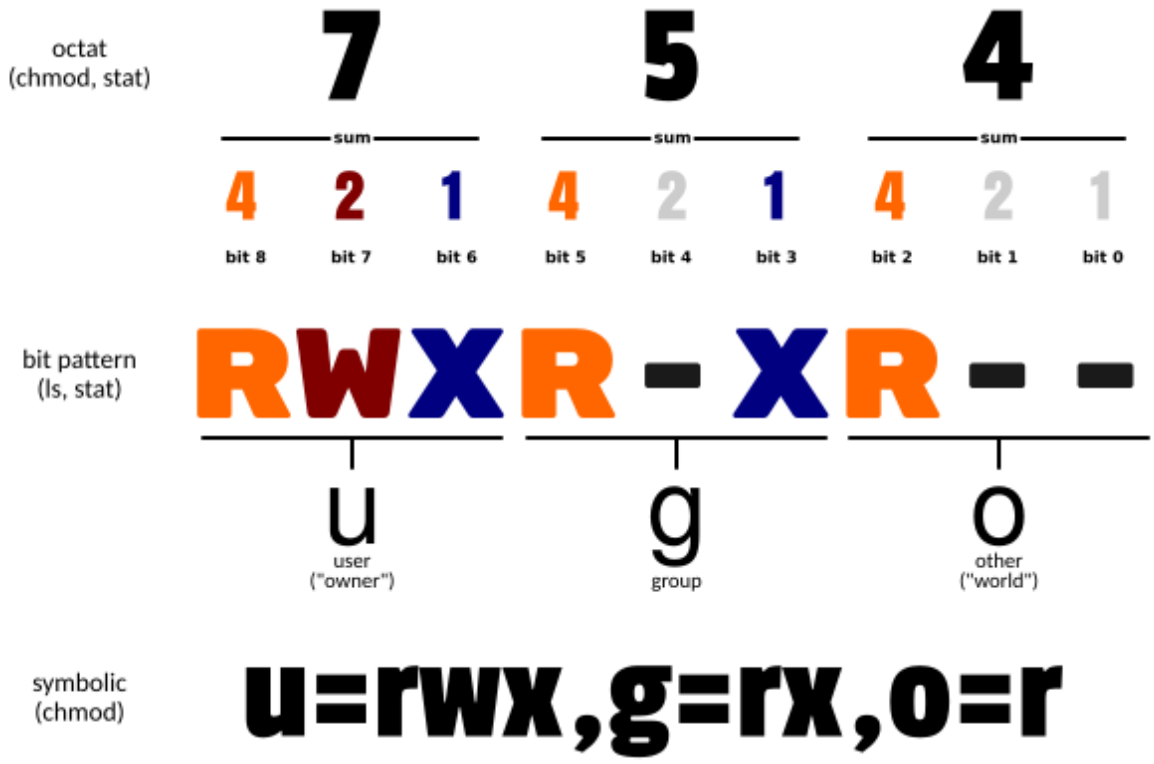
 Linux 命令大全

Linux chmod（英文全拼：change mode）命令是控制用户对文件的权限的命令

Linux/Unix 的文件调用权限分为三级 : 文件所有者（Owner）、用户组（Group）、其它用户（Other Users）。



只有文件所有者和超级用户可以修改文件或目录的权限。可以使用绝对模式（八进制数字模式），符号模式指定文件的权限。



使用权限：所有使用者

语法

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

参数说明

mode：权限设定字串，格式如下：

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

其中：

- u 表示该文件的拥有者，g 表示与该文件的拥有者属于同一个群体(group)者，o 表示其他以外的人，a 表示这三者皆是。
- + 表示增加权限、- 表示取消权限、= 表示唯一设定权限。
- r 表示可读取，w 表示可写入，x 表示可执行，X 表示只有当该文件是个子目录或者该文件已经被设定过为可执行。

其他参数说明：

- -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

符号模式

使用符号模式可以设置多个项目：who（用户类型），operator（操作符）和 permission（权限），每个项目的设置可以用逗号隔开。 命令 chmod 将修改 who 指定的用户类型对文件的访问权限，用户类型由一个或者多个字母在 who 的位置来说明，如 who 的符号模式表所示:

who	用户类型	说明
u	user	文件所有者
g	group	文件所有者所在组
o	others	所有其他用户
a	all	所有用户, 相当于 *ugo*

operator 的符号模式表:

Operator	说明
`+`	为指定的用户类型增加权限
`-`	去除指定用户类型的权限
`=`	设置指定用户权限的设置，即将用户类型的所有权限重新设置

permission 的符号模式表:

模式	名字	说明
`r`	读	设置为可读权限
`w`	写	设置为可写权限
`x`	执行权限	设置为可执行权限
`X`	特殊执行权限	只有当文件为目录文件，或者其他类型的用户有可执行权限时，才将文件权限设置可执行
`s`	setuid/gid	当文件被执行时，根据who参数指定的用户类型设置文件的setuid或者setgid权限
`t`	粘贴位	设置粘贴位，只有超级用户可以设置该位，只有文件所有者u可以使用该位

八进制语法

chmod命令可以使用八进制数来指定权限。文件或目录的权限位是由9个权限位来控制，每三位为一组，它们分别是文件所有者（User）的读、写、执行，用户组（Group）的读、写、执行以及其它用户（Other）的读、写、执行。历史上，文件权限被放在一个比特掩码中，掩码中指定的比特位设为1，用来说明一个类具有相应的优先级。

#	权限	rwx	二进制
7	读 + 写 + 执行	rwx	111

#	权限	rwx	二进制
6	读 + 写	rw-	110
5	读 + 执行	r-x	101
4	只读	r--	100
3	写 + 执行	-wx	011
2	只写	-w-	010
1	只执行	--x	001
0	无	---	000

例如， 765 将这样解释：

- 所有者的权限用数字表达：属主的那三个权限位的数字加起来的总和。如 rwx ， 也就是 4+2+1 ， 应该是 7。
- 用户组的权限用数字表达：属组的那个权限位数字的相加的总和。如 rw- ， 也就是 4+2+0 ， 应该是 6。
- 其它用户的权限数字表达：其它用户权限位的数字相加的总和。如 r-x ， 也就是 4+0+1 ， 应该是 5。

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
```

为 ex1.py 文件拥有者增加可执行权限:

```
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
```

其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。

r=4，w=2，x=1

- 若要 rwx 属性则 4+2+1=7;
- 若要 rw- 属性则 4+2=6;
- 若要 r-x 属性则 4+1=5。

```
chmod a=rwx file
```

和

```
chmod 777 file
```

效果相同

```
chmod ug=rwx,o=x file
```

和

```
chmod 771 file
```

效果相同

若用 **chmod 4755 filename** 可使此程序具有 root 的权限。

更多说明

命令	说明
`chmod a+r *file*`	给file的所有用户增加读权限
`chmod a-x *file*`	删除file的所有用户的执行权限
`chmod a+rw *file*`	给file的所有用户增加读写权限
`chmod +rwx *file*`	给file的所有用户增加读写执行权限
`chmod u=rw,go= *file*`	对file的所有者设置读写权限，清空该用户组和其他用户对file的所有权限（空格代表无权限）
`chmod -R u+r,go-r *docs*`	对目录docs和其子目录层次结构中的所有文件给用户增加读权限，而对用户组和其他用户删除读权限
`chmod 664 *file*`	对file的所有者和用户组设置读写权限, 为其其他用户设置读权限
`chmod 0755 *file*`	相当于`u=rwx (4+2+1),go=rx (4+1 & 4+1)`。`0` 没有特殊模式。
`chmod 4755 *file*`	`4`设置了设置用户ID位，剩下的相当于u=rwx (4+2+1),go=rx (4+1 & 4+1)。
`find path/ -type d -exec chmod a-x {} \;`	删除可执行权限对path/以及其所有的目录（不包括文件）的所有用户，使用'-type f'匹配文件
`find path/ -type d -exec chmod a+x {} \;`	允许所有用户浏览或通过目录path/

Linux diff 命令

Linux 命令大全

Linux diff 命令用于比较文件的差异。

diff 以逐行的方式，比较文本文件的异同处。如果指定要比较目录，则 diff 会比较目录中相同文件名的文件，但不会比较其中子目录。

语法

```
diff [-abBcdefHilnNpPqrstTuvwy][-<行数>][-C <行数>][-D <巨集名称>][-I <字符或字符串>][-S <文件>][-W <宽度>][-x <文件或目录>][-X <文件>][--help][--left-column][--suppress-common-line][文件或目录1][文件或目录2]
```

参数:

- -<行数>　指定要显示多少行的文本。此参数必须与-c或-u参数一并使用。
- -a或--text　diff预设只会逐行比较文本文件。
- -b或--ignore-space-change　不检查空格字符的不同。
- -B或--ignore-blank-lines　不检查空白行。
- -c　显示全部内文，并标出不同之处。
- -C<行数>或--context<行数>　与执行"-c-<行数>"指令相同。
- -d或--minimal　使用不同的演算法，以较小的单位来做比较。
- -D<巨集名称>或ifdef<巨集名称>　此参数的输出格式可用于前置处理器巨集。
- -e或--ed　此参数的输出格式可用于ed的script文件。
- -f或-forward-ed　输出的格式类似ed的script文件，但按照原来文件的顺序来显示不同处。
- -H或--speed-large-files　比较大文件时，可加快速度。
- -I<字符或字符串>或--ignore-matching-lines<字符或字符串>　若两个文件在某几行有所不同，而这几行同时都包含了选项中指定的字符或字符串，则不显示这两个文件的差异。
- -i或--ignore-case　不检查大小写的不同。
- -l或--paginate　将结果交由pr程序来分页。

- -n或--rcs　将比较结果以RCS的格式来显示。
- -N或--new-file　在比较目录时，若文件A仅出现在某个目录中，预设会显示：
- Only in目录：文件A若使用-N参数，则diff会将文件A与一个空白的文件比较。
- -p　若比较的文件为C语言的程序码文件时，显示差异所在的函数名称。
- -P或--unidirectional-new-file　与-N类似，但只有当第二个目录包含了一个第一个目录所没有的文件时，才会将这个文件与空白的文件做比较。
- -q或--brief　仅显示有无差异，不显示详细的信息。
- -r或--recursive　比较子目录中的文件。
- -s或--report-identical-files　若没有发现任何差异，仍然显示信息。
- -S<文件>或--starting-file<文件>　在比较目录时，从指定的文件开始比较。
- -t或--expand-tabs　在输出时，将tab字符展开。
- -T或--initial-tab　在每行前面加上tab字符以便对齐。
- -u,-U<列数>或--unified=<列数>　以合并的方式来显示文件内容的不同。
- -v或--version　显示版本信息。
- -w或--ignore-all-space　忽略全部的空格字符。
- -W<宽度>或--width<宽度>　在使用-y参数时，指定栏宽。
- -x<文件名或目录>或--exclude<文件名或目录>　不比较选项中所指定的文件或目录。
- -X<文件>或--exclude-from<文件>　您可以将文件或目录类型存成文本文件，然后在=<文件>中指定此文本文件。
- -y或--side-by-side　以并列的方式显示文件的异同之处。
- --help　显示帮助。
- --left-column　在使用-y参数时，若两个文件某一行内容相同，则仅在左侧的栏位显示该行内容。

- --suppress-common-lines　在使用-y参数时，仅显示不同之处。

实例1：比较两个文件

```
[root@localhost test3]# diff log2014.log log2013.log
3c3
< 2014-03
---
> 2013-03
8c8
< 2013-07
---
> 2013-08
11,12d10
< 2013-11
< 2013-12
```

上面的"3c3"和"8c8"表示log2014.log和log20143log文件在3行和第8行内容有所不同；"11,12d10"表示第一个文件比第二个文件多了第11和12行。

实例2：并排格式输出

```
[root@localhost test3]# diff log2014.log log2013.log  -y -
W 50
2013-01                 2013-01
2013-02                 2013-02
2014-03               | 2013-03
2013-04                 2013-04
2013-05                 2013-05
2013-06                 2013-06
2013-07                 2013-07
2013-07               | 2013-08
2013-09                 2013-09
2013-10                 2013-10
2013-11               <
2013-12               <
[root@localhost test3]# diff log2013.log log2014.log  -y -
W 50
2013-01                 2013-01
```

```
2013-02                    2013-02
2013-03                  | 2014-03
2013-04                    2013-04
2013-05                    2013-05
2013-06                    2013-06
2013-07                    2013-07
2013-08                  | 2013-07
2013-09                    2013-09
2013-10                    2013-10
                         > 2013-11
                         > 2013-12
```

说明：

- "|"表示前后2个文件内容有不同
- "<"表示后面文件比前面文件少了1行内容
- ">"表示后面文件比前面文件多了1行内容

Linux vi/vim

所有的 Unix Like 系统都会内建 vi 文书编辑器，其他的文书编辑器则不一定会存在。

但是目前我们使用比较多的是 vim 编辑器。

vim 具有程序编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程序设计。

相关文章：[史上最全Vim快捷键键位图 — 入门到进阶]

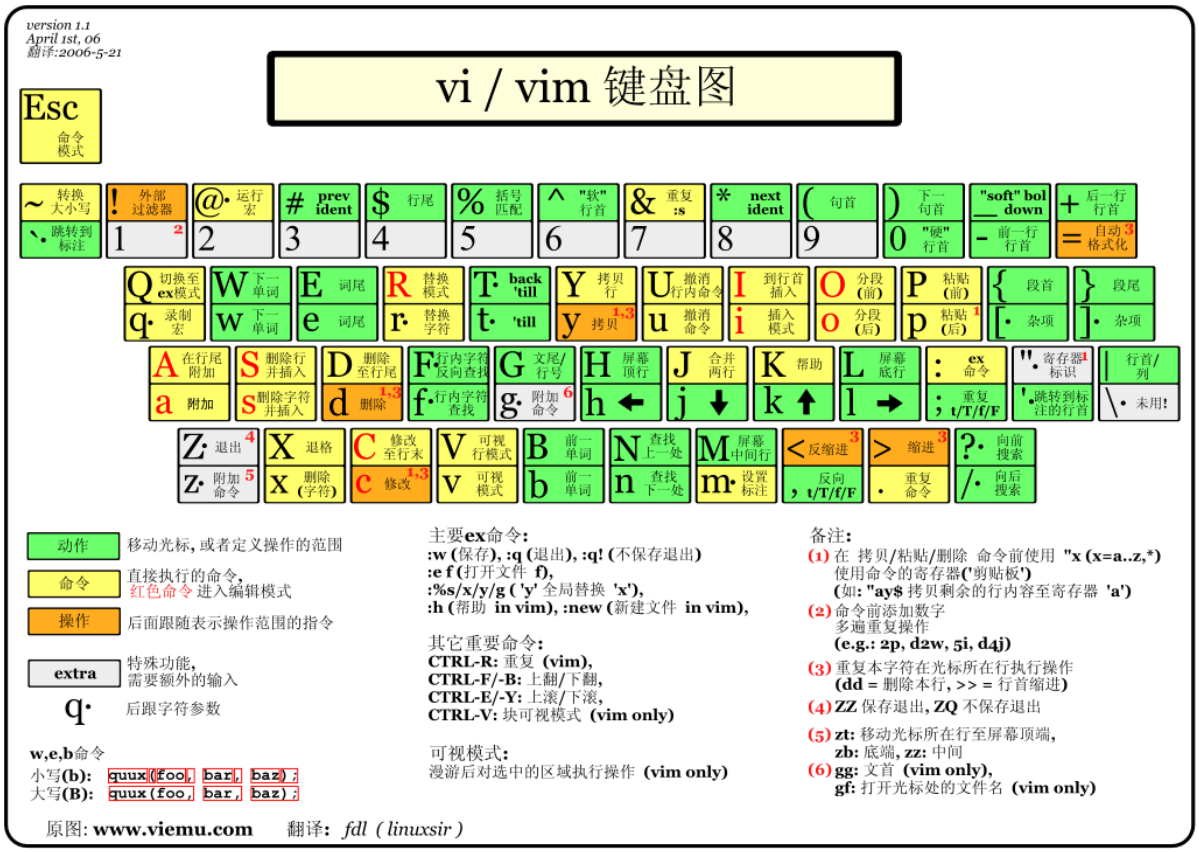
什么是 vim?

Vim 是从 vi 发展出来的一个文本编辑器。代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。

简单的来说， vi 是老式的字处理器，不过功能已经很齐全了，但是还是有可以进步的地方。 vim 则可以说是程序开发者的一项很好用的工具。

连 vim 的官方网站 (https://www.vim.org/) 自己也说 vim 是一个程序开发工具而不是文字处理软件。

vim 键盘图



vi/vim 的使用

基本上 vi/vim 共分为三种模式，**命令模式（Command Mode）、输入模式（Insert Mode）和命令行模式（Command-Line Mode）**。

命令模式

用户刚刚启动 vi/vim，便进入了命令模式。

此状态下敲击键盘动作会被 Vim 识别为命令，而非输入字符，比如我们此时按下 **i**，并不会输入一个字符，**i** 被当作了一个命令。

以下是普通模式常用的几个命令：

- **i** -- 切换到输入模式，在光标当前位置开始输入文本。
- **x** -- 删除当前光标所在处的字符。
- **:** -- 切换到底线命令模式，以在最底一行输入命令。

- **a** -- 进入插入模式，在光标下一个位置开始输入文本。
- **o**：在当前行的下方插入一个新行，并进入插入模式。
- **O** -- 在当前行的上方插入一个新行，并进入插入模式。
- **dd** -- 剪切当前行。
- **yy** -- 复制当前行。
- **p**（小写） -- 粘贴剪贴板内容到光标下方。
- **P**（大写）-- 粘贴剪贴板内容到光标上方。
- **u** -- 撤销上一次操作。
- **Ctrl + r** -- 重做上一次撤销的操作。
- **:w** -- 保存文件。
- **:q** -- 退出 Vim 编辑器。
- **:q!** -- 强制退出Vim 编辑器，不保存修改。

若想要编辑文本，只需要启动 Vim，进入了命令模式，按下 **i** 切换到输入模式即可。

命令模式只有一些最基本的命令，因此仍要依靠**底线命令行模式**输入更多命令。

输入模式

在命令模式下按下 **i** 就进入了输入模式，使用 **Esc** 键可以返回到普通模式。

在输入模式中，可以使用以下按键：

- **字符按键以及Shift组合**，输入字符
- **ENTER**，回车键，换行
- **BACK SPACE**，退格键，删除光标前一个字符
- **DEL**，删除键，删除光标后一个字符
- **方向键**，在文本中移动光标
- **HOME/END**，移动光标到行首/行尾
- **Page Up/Page Down**，上/下翻页

- **Insert**，切换光标为输入/替换模式，光标将变成竖线/下划线
- **ESC**，退出输入模式，切换到命令模式

底线命令模式

在命令模式下按下：（英文冒号）就进入了底线命令模式。

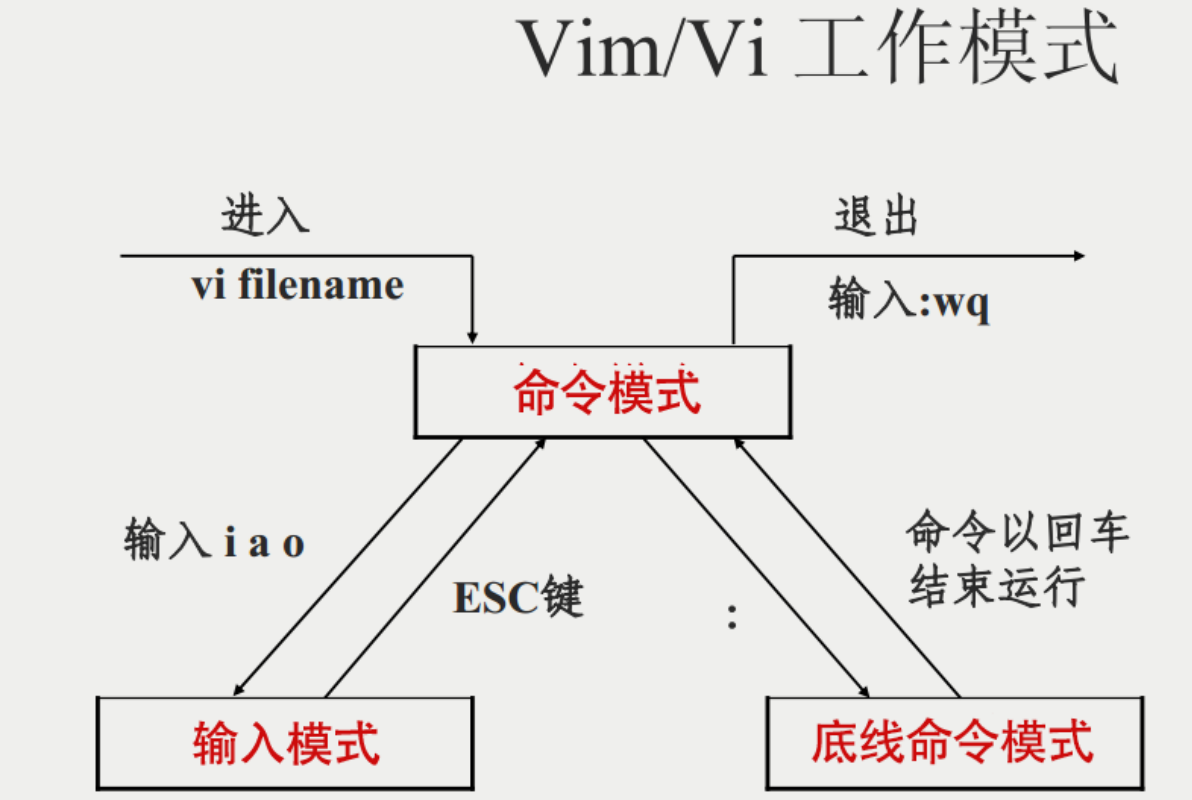
底线命令模式可以输入单个或多个字符的命令，可用的命令非常多。

在底线命令模式中，基本的命令有（已经省略了冒号）：

- `:w`：保存文件。
- `:q`：退出 Vim 编辑器。
- `:wq`：保存文件并退出 Vim 编辑器。
- `:q!`：强制退出Vim编辑器，不保存修改。

按 **ESC** 键可随时退出底线命令模式。

简单的说，我们可以将这三个模式想成底下的图标来表示：



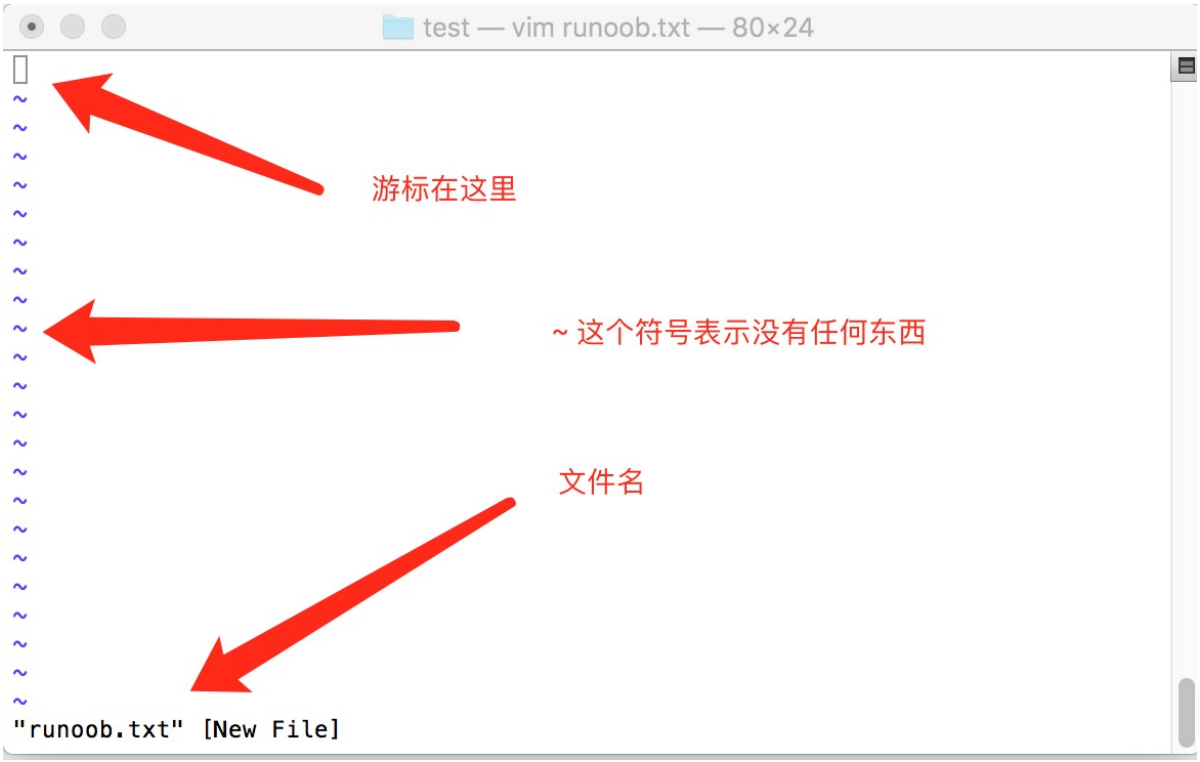
vi/vim 使用实例

使用 vi/vim 进入一般模式

如果你想要使用 vi 来建立一个名为 runoob.txt 的文件时，你可以这样做：

```
$ vim runoob.txt
```

直接输入 **vi 文件名** 就能够进入 vi 的一般模式了。请注意，记得 vi 后面一定要加文件名，不管该文件存在与否！



按下 i 进入输入模式(也称为编辑模式)，开始编辑文字

在一般模式之中，只要按下 i, o, a 等字符就可以进入输入模式了！

在编辑模式当中，你可以发现在左下角状态栏中会出现 –INSERT- 的字样，那就是可以输入任意字符的提示。

这个时候，键盘上除了 **Esc** 这个按键之外，其他的按键都可以视作为一般的输入按钮了，所以你可以进行任何的编辑。

按下 ESC 按钮回到一般模式

好了，假设我已经按照上面的样式给他编辑完毕了，那么应该要如何退出呢？是的！没错！就是给他按下 **Esc** 这个按钮即可！马上你就会发现画面左下角的 – INSERT – 不见了！

在一般模式中按下 :wq 储存后离开 vi

OK，我们要存档了，存盘并离开的指令很简单，输入 **:wq** 即可保存离开！

OK! 这样我们就成功创建了一个 runoob.txt 的文件。

---

vi/vim 按键说明

除了上面简易范例的 i, Esc, :wq 之外，其实 vim 还有非常多的按键可以使用。

第一部分：一般模式可用的光标移动、复制粘贴、搜索替换等

移动光标的方法	
h 或 向左箭头键(←)	光标向左移动一个字符
j 或 向下箭头键(↓)	光标向下移动一个字符
k 或 向上箭头键(↑)	光标向上移动一个字符
l 或 向右箭头键(→)	光标向右移动一个字符
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键， 亦即加上想要进行的次数(数字)后，按下动作即可！	
[Ctrl] + [f]	屏幕『向下』移动一页，相当于 [Page Down]按键 (常用)
[Ctrl] + [b]	屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
[Ctrl] + [d]	屏幕『向下』移动半页
[Ctrl] + [u]	屏幕『向上』移动半页
+	光标移动到非空格符的下一行

移动光标的方法	
-	光标移动到非空格符的上一行
n	那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20 则光标会向后面移动 20 个字符距离。
0 或功能键[Home]	这是数字『 0 』：移动到这一行的最前面字符处 (常用)
$ 或功能键[End]	移动到这一行的最后面字符处(常用)
H	光标移动到这个屏幕的最上方那一行的第一个字符
M	光标移动到这个屏幕的中央那一行的第一个字符
L	光标移动到这个屏幕的最下方那一行的第一个字符
G	移动到这个档案的最后一行(常用)
nG	n 为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行(可配合 :set nu)
gg	移动到这个档案的第一行，相当于 1G 啊！ (常用)
n	n 为数字。光标向下移动 n 行(常用)
搜索替换	
/word	向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串，就输入 /vbird 即可！ (常用)
?word	向光标之上寻找一个字符串名称为 word 的字符串。

移动光标的方法	
n	这个 n 是英文按键。代表重复前一个搜寻的动作。举例来说， 如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！
N	这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。 例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。
使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词！	
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2 ！ 举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则：『:100,200s/vbird/VBIRD/g』。(常用)
:1,$s/word1/word2/g 或 **:%s/word1/word2/g**	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！(常用)
:1,$s/word1/word2/gc 或 **:%s/word1/word2/gc**	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)
删除、复制与贴上	

移动光标的方法	
x, X	在一行字当中，x 为向后删除一个字符(相当于 [del] 按键)， X 为向前删除一个字符(相当于 [backspace] 亦即是退格键) (常用)
nx	n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符，『10x』。
dd	剪切游标所在的那一整行(常用)，用 p/P 可以粘贴。
ndd	n 为数字。剪切光标所在的向下 n 行，例如 20dd 则是剪切 20 行(常用)，用 p/P 可以粘贴。
d1G	删除光标所在到第一行的所有数据
dG	删除光标所在到最后一行的所有数据
d$	删除游标所在处，到该行的最后一个字符
d0	那个是数字的 0 ，删除游标所在处，到该行的最前面一个字符
yy	复制游标所在的那一行(常用)
nyy	n 为数字。复制光标所在的向下 n 行，例如 20yy 则是复制 20 行(常用)
y1G	复制游标所在行到第一行的所有数据
yG	复制游标所在行到最后一行的所有数据
y0	复制光标所在的那个字符到该行行首的所有数据
y$	复制光标所在的那个字符到该行行尾的所有数据

移动光标的方法	
p, P	p 为将已复制的数据在光标下一行贴上，P 则为贴在游标上一行！ 举例来说，我目前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后， 那 10 行数据会贴在原本的 20 行之后，亦即由 21 行开始贴。但如果是按下 P 呢？那么原本的第 20 行会被推到变成 30 行。 (常用)
J	将光标所在行与下一行的数据结合成同一行
c	重复删除多个数据，例如向下删除 10 行，[10cj]
u	复原前一个动作。(常用)
[Ctrl]+r	重做上一个动作。(常用)
这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次～ 利用这两个功能按键，你的编辑，嘿嘿！很快乐的啦！	
.	不要怀疑！这就是小数点！意思是重复前一个动作的意思。 如果你想要重复删除、重复贴上等等动作，按下小数点『.』就好了！ (常用)

第二部分：一般模式切换到编辑模式的可用的按钮说明

进入输入或取代的编辑模式	
i, I	进入输入模式(Insert mode)： i 为『从目前光标所在处输入』， I 为『在目前所在行的第一个非空格符处开始输入』。(常用)
a, A	进入输入模式(Insert mode)： a 为『从目前光标所在的下一个字符处开始输入』， A 为『从光标所在行的最后一个字符处开始输入』。(常用)
o, O	进入输入模式(Insert mode)： 这是英文字母 o 的大小写。o 为在目前光标所在的下一行处输入新的一行； O 为在目前光标所在的上一行处输入新的一行！(常用)
r, R	进入取代模式(Replace mode)： r 只会取代光标所在的那一个字符一次；R会一直取代光标所在的文字，直到按下 ESC 为止；(常用)

进入输入或取代的编辑模式	
上面这些按键中，在 vi 画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样。 由名称就知道该动作了吧！！特别注意的是，我们上面也提过了，你想要在档案里面输入字符时， 一定要在左下角处看到 INSERT 或 REPLACE 才能输入喔！	
[Esc]	退出编辑模式，回到一般模式中(常用)

第三部分：一般模式切换到指令行模式的可用的按钮说明

指令行的储存、离开等指令	
:w	将编辑的数据写入硬盘档案中(常用)
:w!	若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊！
:q	离开 vi (常用)
:q!	若曾修改过档案，又不想储存，使用 ! 为强制离开不储存档案。
注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～	
:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)
ZZ	这是大写的 Z 喔！如果修改过，保存当前文件，然后退出！效果等同于(保存并退出)

指令行的储存、离开等指令	
ZQ	不保存，强制退出。效果等同于 **:q!**。
:w [filename]	将编辑的数据储存成另一个档案（类似另存新档）
:r [filename]	在编辑的数据中，读入另一个档案的数据。亦即将 『filename』 这个档案内容加到游标所在行后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个档案。
:! command	暂时离开 vi 到指令行模式下执行 command 的显示结果！例如 『:! ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！
vim 环境的变更	
:set nu	显示行号，设定之后，会在每一行的前缀显示该行的行号
:set nonu	与 set nu 相反，为取消行号！

特别注意，在 vi/vim 中，数字是很有意义的！数字通常代表重复做几次的意思！ 也有可能是代表去到第几个什么什么的意思。

举例来说，要删除 50 行，则是用 『50dd』 对吧！ 数字加在动作之前，如我要向下移动 20 行呢？那就是『20j』或者是『20↓』即可。