比较：

awk、sed、grep更适合的方向：

- grep 更适合单纯的查找或匹配文本
- sed 更适合编辑匹配到的文本
- awk 更适合格式化文本，对文本进行较复杂格式处理

awk

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

log.txt 文本内容如下：

```
2 this is a test
3 Are 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
```

案例：

```
# 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
```

案例:

```
# 使用","分割
 $  awk -F, '{print $1,$2}'   log.txt
 ----------------------------------------
 2 this is a test
 3 Are 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 Are 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
```

案例:

```
 $ 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}
```

案例：

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

sed

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

参数说明：

-e<script>或--expression=<script> 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 后面通常不接任何咚咚；
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 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！

以行为单位的新增/删除

```
# 4 行之后追加 2 行：
sed -e '4 a newline\nnewline2' testfile

# 在第二行后(亦即是加在第三行)加上『drink tea?』字样!
[root@www ~]# nl /etc/passwd | sed '2a drink tea'
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
drink tea
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(后面省略).....

# 删除第三行到最后一行
nl /etc/passwd | sed '3,$d'
```

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?

```
[root@www ~]# nl /etc/passwd | sed '2,5c No 2-5 number'
1 root:x:0:0:root:/root:/bin/bash
No 2-5 number
6 sync:x:5:0:sync:/sbin:/bin/sync
.....(后面省略).....
```

透过这个方法我们就能够将数据整行取代了!

仅列出 /etc/passwd 文件内的第 5-7 行

```
[root@www ~]# nl /etc/passwd | sed -n '5,7p'
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6 sync:x:5:0:sync:/sbin:/bin/sync
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

数据的搜寻并显示

搜索 /etc/passwd有root关键字的行

```
nl /etc/passwd | sed '/root/p'
1  root:x:0:0:root:/root:/bin/bash
1  root:x:0:0:root:/root:/bin/bash
```

```
2  daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3  bin:x:2:2:bin:/bin:/bin/sh
4  sys:x:3:3:sys:/dev:/bin/sh
5  sync:x:4:65534:sync:/bin:/bin/sync
....下面忽略
```

如果root找到，除了输出所有行，还会输出匹配行。
使用-n的时候将只打印包含模板的行。

```
nl /etc/passwd | sed -n '/root/p'
1  root:x:0:0:root:/root:/bin/bash
```

数据的搜寻并删除

删除/etc/passwd所有包含root的行，其他行输出

```
nl /etc/passwd | sed  '/root/d'
2  daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3  bin:x:2:2:bin:/bin:/bin/sh
....下面忽略
#第一行的匹配root已经删除了
```

数据的搜寻并执行命令

搜索/etc/passwd,找到root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bash替换为blueshell，再输出这行：

```
nl /etc/passwd | sed -n '/root/{s/bash/blueshell/;p;q}'
1  root:x:0:0:root:/root:/bin/blueshell
```

最后的q是退出。

数据的搜寻并替换

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搜寻并取代。基本上 sed 的搜寻与替代的与 vi 相当的类似！他有点像这样：

```
sed 's/要被取代的字串/新的字串/g'
```

先观察原始信息，利用 /sbin/ifconfig 查询 IP

```
[root@www ~]# /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

```
[root@www ~]# /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，亦即： 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

```
[root@www ~]# /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/etc/passwd第三行到末尾的数据，并把bash替换为blueshell

```
nl /etc/passwd | sed -e '3,$d' -e 's/bash/blueshell/'
1  root:x:0:0:root:/root:/bin/blueshell
2  daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

-e表示多点编辑，第一个编辑命令删除/etc/passwd第三行到末尾的数据，第二条命令搜索bash替换为blueshell。

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，由於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：

```
[root@www ~]# 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 !

```
[root@www ~]# sed -i 's/\.$/\!/g' regular_express.txt
[root@www ~]# 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**:

```
[root@www ~]# sed -i '$a # This is a test' regular_express.txt
[root@www ~]# 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！

grep

语法

```
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...]
```

参数:

- -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。

实例

1、查找字符

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
```

2、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
```

3、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.
```