1. grep，awk 的用法

```
2 #First you can use grep (-n) to find the number of lines of string.
3 #Then you can use awk to separate the answer.
4 grep -n -i "$2" ./"$1" > ref
5 awk -F ":" '{print $1}' ./ref > ./"$3"
```

Awk:

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
```

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
```

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
```

Grep:

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。

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.
```

2.shell 脚本的书写和参数传递

```
#!/bin/bash
a=1
while [ $a -le 100 ]
do
        if [ $a -gt 70 ]          #if loop variable is greater than 70
        then
                rm -r ./"file$a"

        elif [ $a -gt 40 ]        # else if loop variable is great than 40
        then
                mv ./"file$a" ./" newfile$a"

        fi
        let a=a+1
                #don't forget change the loop variable
done
```

3.gcc 编译，Makefile

```
run:
	gcc -c fibo.c -I ../include
	gcc -c main.c -I ../include

```

```
run:
	cd ./code
	make
	gcc -o fibo ./code/fibo.o ./code/main.o

clean:
	rm ./code/fibo.o
	rm ./code/main.o
~
~
```

4.sed 的用法

```
#!/bin/bash
sed -i "s/$2/$3/g" ./$1
```

```
#!/bin/bash
sed -n "8p" $1 >> $2
sed -n "32p" $1 >> $2
sed -n "128p" $1 >> $2
sed -n "512p" $1 >> $2
sed -n "1024p" $1 >> $2

```

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.
```

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
```

```
$ 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*
```

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-
```

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
```