Linux 三剑客 grep、awk、sed

在后处理数字规整开发过程中，发现需要进行很多文本格式转换。Linux 三剑客在文本格式处理上很强大，可惜很久不用，不太会使了，目前是用 python 来处理。重新整理下三剑客的用法，希望能提高日后开发效率。

grep

简介

grep 与 sed、awk 并称 Linux 中的三剑客，其全称是 Global search Regular Expression and Print out the line。

语法格式：

```
grep [options] [pattern [file]
```

常用参数：

参数	说明
-c	计算找到的行数
-o	指出匹配的内容
-i	不区分大小写
-n	显示匹配内容的行号
-r	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命令将回报信息并停止动作
-v	反向选择，即没有'搜索字符串'内容的行
-l	列出文件内容符合指定的范本样式的文件名称
-E	egrep，可以使用扩展正则表达式
-F	fgrep, 不支持正则表达式
--color	高亮查找词

正则表达式种类繁多且复杂，POSIX 将正则表达式进行了标准化，并把实现方法分为了两大类：

- 基本正则表达式（BRE）
- 扩展正则表达式（ERE）

两者的区别，更多的是元字符的区别。在基本正则表达式（BRE）中，只承认`^`、`$`、`.`、`[`、`]`、`*`这些是元字符，所有其他的字符都被识别为普通字符。而在扩展正则表达式（ERE）中，则在BRE的基础上增加了`（`、`）`、`{`、`}`、`?`、`+`、`|`等元字符。

示例

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
(中间省略数十行)
apache:x:48:48:Apache:/var/www:/sbin/nologin
test:x:502:502::/home/test:/bin/bash
leo:x:503:503::/home/leo:/bin/bash
roc:x:504:504::/home/roc:/bin/bash
```

```
# 搜索包含leo字符串的行:
$ grep leo /etc/passwd
leo:x:503:503::/home/leo:/bin/bash
$ grep -v leo /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
(中间省略数十行)
apache:x:48:48:Apache:/var/www:/sbin/nologin
test:x:502:502::/home/test:/bin/bash
roc:x:504:504::/home/roc:/bin/bash

# 使用选项 n
$ grep -n leo /etc/passwd
29:leo:x:503:503::/home/leo:/bin/bash
# 使用-c选项
$ grep -c leo /etc/passwd
1
$ grep -i "LEO" /etc/passwd
leo:x:503:503::/home/leo:/bin/bash
```

grep 命令可以一次搜索很多个文件，从大量的文件中找出含有特定字符的文件.

```
# 当前目录下有三个文件
$ ll
total 12
-rw-rw-r-- 1 roc roc 58 Mar 15 17:47 1.txt
-rw-rw-r-- 1 roc roc 59 Mar 15 17:51 2.txt
-rw-rw-r-- 1 roc roc 58 Mar 15 17:52 3.txt
# 1.txt文件的内容如下
$ cat 1.txt
this first file
this file contain some import infomation.
# 2.txt文件的内容如下
$ cat 2.txt
this second file
this file contain some import infomation.
# 3.txt文件的内容如下
$ cat 3.txt
```

```
this third file
this file contain some import infomation.
```

找出内容中含有first单词的文件

```
$ grep -l "first" *.txt
1.txt
```

```
# 找出不含 first 单词的文件都有哪些
$ grep -L "first" *.txt
2.txt
3.txt
```

grep支持正则：

```
# 搜索/etc/passwd文件中开头是 leo 的行
$ grep '^leo' /etc/passwd
leo:x:503:503::/home/leo:/bin/bash
# 使用<bin>来表示bin是一个词，避免匹配到sbin
$ grep '\<bin\>' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
cloud-user:x:500:500::/home/cloud-user:/bin/bash
test:x:502:502::/home/test:/bin/bash
leo:x:503:503::/home/leo:/bin/bash
roc:x:504:504::/home/roc:/bin/bash
```

egrep 可以实现多条件查找

```
# 查找以root为首或以bash为尾的行
$ egrep '^root|bash$' passwd
root:x:0:0:root:/root:/bin/bash
cloud-user:x:500:500::/home/cloud-user:/bin/bash
test:x:502:502::/home/test:/bin/bash
leo:x:503:503::/home/leo:/bin/bash
roc:x:504:504::/home/roc:/bin/bash

```

假如搜索字符串中包含了不少特殊字符，而这些特殊字符恰好又是正则表达式预留的字符，就可以使用 fgrep 来避免烦琐的转义了，在 fgrep 的眼里，没有特殊字符，都是普通字符。

```
# 我们的roc.txt文件中有几个^和$
$ cat roc.txt
^this third file
^$this file contain some import infomation.

# grep会尝试去找开头为this的行，但并未找到
$ grep '^this' roc.txt

# fgrep会老老实实地去找^this字符串，它找到了
$ fgrep '^this' roc.txt
^this third file
```

awk

`awk` 是处理文本文件的一个应用程序，它依次处理文件的每一行，并读取里面的每一个字段。对于日志、CSV 那样的每行格式相同的文本文件，`awk` 可能是最方便的工具。

基本用法

`awk` 的基本用法就是下面的形式。

```
$ awk 动作 文件名
# 示例
$ awk '{print $0}' demo.txt
```

上面示例中，`demo.txt` 是 `awk` 所要处理的文本文件。前面单引号内部有一个大括号，里面就是每一行的处理动作 `print $0`。其中，`print` 是打印命令，`$0` 代表当前行，上面命令的执行结果，就是把每一行原样打印出来。

`awk` 会根据空格和制表符，将每一行分成若干字段，依次用 `$1`、`$2`、`$3` 代表第一个字段、第二个字段、第三个字段等等。

```
$ echo 'this is a test' | awk '{print $3}'
a
```

下面，为了便于举例，我们把 `/etc/passwd` 文件保存成 `demo.txt`。

```
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

这个文件的字段分隔符是冒号（`:`），所以要用 `-F` 参数指定分隔符为冒号。然后，才能提取到它的第一个字段。

```
$ awk -F ':' '{ print $1 }' demo.txt
root
daemon
bin
sys
sync
```

变量

除了 `$ + 数字` 表示某个字段，`awk` 还提供其他一些变量。

变量 `NF` 表示当前行有多少个字段，因此 `$NF` 就代表最后一个字段。

```
$ echo "this is a test" | awk '{print NF}'
4
$ echo 'this is a test' | awk '{print $NF}'
test
```

`$(NF-1)` 代表倒数第二个字段。

```
$ awk -F ':' '{print $1, $(NF-1)}' demo.txt
root /root
daemon /usr/sbin
bin /bin
sys /dev
sync /bin
```

上面代码中，`print`命令里面的逗号，表示输出的时候，两个部分之间使用空格分隔。

变量`NR`表示当前处理的是第几行。

```
$ awk -F ':' '{print NR ") " $1}' demo.txt
1) root
2) daemon
3) bin
4) sys
5) sync
```

上面代码中，`print`命令里面，如果原样输出字符，要放在双引号里面。

`awk`的其他内置变量如下。

- `FILENAME`：当前文件名
- `FS`：字段分隔符，默认是空格和制表符。
- `RS`：行分隔符，用于分割每一行，默认是换行符。
- `OFS`：输出字段的分隔符，用于打印时分隔字段，默认为空格。
- `ORS`：输出记录的分隔符，用于打印时分隔记录，默认为换行符。
- `OFMT`：数字输出的格式，默认为`%.6g`。

函数

`awk`还提供了一些内置函数，方便对原始数据的处理。

函数`toupper()`用于将字符转为大写。

```
$ awk -F ':' '{ print toupper($1) }' demo.txt
ROOT
DAEMON
BIN
SYS
SYNC
```

上面代码中，第一个字段输出时都变成了大写。

其他常用函数如下。

- `tolower()`：字符转为小写。
- `length()`：返回字符串长度。
- `substr()`：返回子字符串。
- `sin()`：正弦。
- `cos()`：余弦。
- `sqrt()`：平方根。
- `rand()`：随机数。

条件

`awk` 允许指定输出条件，只输出符合条件的行。输出条件要写在动作的前面。

```
$ awk '条件 动作' 文件名
```

请看下面的例子。

```
$ awk -F ':' '/usr/ {print $1}' demo.txt
root
daemon
bin
sys
```

上面代码中，`print` 命令前面是一个正则表达式，只输出包含 `usr` 的行。

下面的例子只输出奇数行，以及输出第三行以后的行。

```
# 输出奇数行
$ awk -F ':' 'NR % 2 == 1 {print $1}' demo.txt
root
bin
sync
# 输出第三行以后的行
$ awk -F ':' 'NR >3 {print $1}' demo.txt
sys
sync
```

下面的例子输出第一个字段等于指定值的行。

```
$ awk -F ':' '$1 == "root" {print $1}' demo.txt
root
$ awk -F ':' '$1 == "root" || $1 == "bin" {print $1}' demo.txt
root
bin
```

if 语句

`awk` 提供了 `if` 结构，用于编写复杂的条件。

```
$ awk -F ':' '{if ($1 > "m") print $1}' demo.txt
root
sys
sync
```

上面代码输出第一个字段的第一个字符大于 `m` 的行。

`if` 结构还可以指定 `else` 部分。

```
$ awk -F ':' '{if ($1 > "m") print $1; else print "---"}' demo.txt
root
---
---
sys
sync
```

sed

基本用法

sed英文全称是stream editor，遵循简单的工作流：读取（从输入中读取某一行），执行（在某一行上执行sed命令）和显示（把结果显示在输出中）。通常sed命令这样被调用：

```
sed [options] 'command' file
```

options是sed可以接受的参数，command是sed的命令集。command的一般形式是：

```
[addr]X[options]
```

X是一个单个字母sed命令。[addr]是一个可选的行地址。如果[addr]是被指定的，那么命令X在匹配的行将被执行。[addr]可以使用一个单独的行号、正则表达式、或行的范围。附加[options]被用于一些sed命令。

常用参数：

```
-e script 、--expression=script：运行指定脚本，当只有一个命令时可以省略
-f script-file、--file=script-file：运行指定脚本文件
-n、--quiet、--silent：只输出脚本处理过的内容
-i：直接修改原文件（默认不修改原文件，需要重定向）
```

命令说明：

```
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！
```

删除

```
$ cat Sed.txt
That is line 1,That is First line
That is line 2,the Second line,Empty line followed

That is line 4,That is Third line
That is line 5,That is Fifth line
```

```
# 将file的第一行删除后输出到屏幕
$ sed '1d' Sed.txt
this is line 2,the Second line,Empty line followed

this is line 4,this is Third line
this is line 5,this is Fifth line
# 由于sed默认不修改原文件，如果希望保存修改后的文件则需要用重定向
sed '1d' Sed.txt > saved_file
# 如果想直接修改文件，使用-i参数，不会有任何输出，而是直接修改了原文件，删除了第一行
# Sed-i '1d' file
```

```
11
12 #删除指定范围的行（第一行到最后行）
13 $ sed '1,$d' Sed.txt
14 $ sed '$d' Sed.txt
15 That is line 1,That is First line
16 That is line 2,the Second line,Empty line followed
17
18 That is line 4,That is Third line
19 #删除除指定范围以外的行（只保留第5行）
20 $ sed '5!d' Sed.txt
21 this is line 5,this is Fifth line
22
23 #删除所有包含Empty的行
24 $ sed '/Empty/d' Sed.txt
25 this is line 1,this is First line
26
27 this is line 4,this is Third line
28 this is line 5,this is Fifth line
29
30 #删除空行
31 $ sed '/^$/d' Sed.txt
32 this is line 1,this is First line
33 this is line 2,the Second line,Empty line followed
34 this is line 4,this is Third line
35 this is line 5,this is Fifth line
```

查找替换

```
1  # s命令用于替换文本，本例中使用LINE替换line，默认情况下只替换第一次匹配到的内容
2  $ sed 's/line/LINE/' Sed.txt
3  this is LINE 1,this is First line
4  this is LINE 2,the Second line,Empty line followed
5
6  this is LINE 4,this is Third line
7  this is LINE 5,this is Fifth line
8
9  # 要想每行最多匹配2个line，并改为LINE，可用如下方式，到第2行中有3个line，前两个被替换了
10 $ sed 's/line/LINE/2' Sed.txt
11 this is line 1,this is First LINE
12 this is line 2,the Second LINE,Empty line followed
13
14 this is line 4,this is Third LINE
15 this is line 5,this is Fifth LINE
16 # s命令利用g选项，可以完成所有匹配值的替换
17 $ sed 's/line/LINE/g' Sed.txt
18 this is LINE 1,this is First LINE
19 this is LINE 2,the Second LINE,Empty LINE followed
20
21 this is LINE 4,this is Third LINE
22 this is LINE 5,this is Fifth LINE
23
24 # 只替换开头的this为that
25 [root@localhost ~]# sed 's/^this/that/' Sed.txt
26 that is line 1,this is First line
27 that is line 2,the Second line,Empty line followed
28
29 that is line 4,this is Third line
```

```
that is line 5,this is Fifth line

# y命令可进行字符转换，其作用为将一系列字符逐个地变换为另外一系列字符
# 将数字1转换为A，2转换为B，4转换为C，5转换成D
$ sed 'y/1245/ABCD/' Sed.txt
this is line A,this is First line
this is line B,the Second line,Empty line followed

this is line C,this is Third line
this is line D,this is Fifth line
```

插入

使用i或a命令插入文本，其中i代表在匹配行之前插入，而a代表在匹配行之后插入

```
# 使用i在第二行前插入文本
$ sed '2 i Insert' Sed.txt
this is line 1,this is First line
Insert
this is line 2,the Second line,Empty line followed

this is line 4,this is Third line
this is line 5,this is Fifth line

# 使用a在第二行后插入文本
$ sed '2 a Insert' Sed.txt
this is line 1,this is First line
this is line 2,the Second line,Empty line followed
Insert

this is line 4,this is Third line
this is line 5,this is Fifth line

# 在匹配行的上一行插入问题
[root@localhost ~]# sed '/Second/i Insert' Sed.txt
this is line 1,this is First line
Insert
this is line 2,the Second line,Empty line followed

this is line 4,this is Third line

# 使用r命令可从其他文件中读取文本，并插入匹配行之后
# 将/etc/passwd中的内容读出放到Sed.txt空行之后
$ sed  '/^$/r /etc/passwd' Sed.txt
this is line 1,this is First line
this is line 2,the Second line,Empty line followed

root:x:0:0:root:/root:/bin/bash
......(略去内容)......
apache:x:48:48:Apache:/var/www:/sbin/nologin
this is line 4,this is Third line
this is line 5,this is Fifth li
```

打印和写文件

```
# 打印出文件中指定的行
$ sed-n '1p' Sed.txt    # 一定要加-n参数，表示不打印没关系的行
this is line 1,this is First line

#使用p命令，则只打印实际处理过的行，简化了输出（使用-n参数）
[root@localhost ~]# sed-n 's/the/THE/p' Sed.txt
this is line 2,THE Second line,Empty line followed

# 使用w命令将结果保存到外部指定文件
$ sed-n '1,2 w output' Sed.txt
# 这里没有任何输出，因为输出被重定向到文件了
$ cat output
this is line 1,this is First line
this is line 2,the Second line,Empty line followed
```