昵称： 肖邦linux
园龄： 8年1个月
粉丝： 239
关注： 17
关注成功

< 2023年8月 >

日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

积分与排名

积分 - 164836
排名 - 7288

随笔分类

肖邦Linux - 01.Apache(1)
肖邦Linux - 02.Python(11)
肖邦Linux - 03.Linux(13)
肖邦Linux - 04.Shell(6)
肖邦Linux - 05.MySQL(10)
肖邦Linux - 06.C语言(9)
肖邦Linux - 07.Linux命令(1)
肖邦Linux - 08.服务搭建(2)
肖邦Linux - 09.集群架构(4)
肖邦Linux - 10.系统监控(1)
更多

友情链接

我的个人博客

阅读排行榜

1. 使用LVS实现负载均衡原理及安装配置详解(122458)
2. sed入门详解教程(47597)
3. C语言基本类型之long long int(20305)

sed入门详解教程

sed 是一个比较古老的，功能十分强大的用于文本处理的流编辑器，加上正则表达式的支持，可以进行大量的复杂的文本编辑操作。sed 本身是一个非常复杂的工具，有专门的书籍讲解 sed 的具体用法，但是个人觉得没有必要去学习它的每个细节，那样没有特别大的实际意义。网上也有很多关于 sed 的教程，我也是抱着学习的心态来学习 sed 的常见的用法，并进行系统的总结，内容基本覆盖了 sed 的大部分的知识点。文中的内容比较简练，加以实际示例来帮助去理解 sed 的使用。

一、写在前边

1、sed介绍

sed 全名为 stream editor，流编辑器，用程序的方式来编辑文本，功能相当的强大。是贝尔实验室的 Lee E.McMahon 在 1973 年到 1974 年之间开发完成，目前可以在大多数操作系统中使用，sed 的出现作为 grep 的继任者。与vim等编辑器不同，sed 是一种非交互式编辑器(即用户不必参与编辑过程)，它使用预先设定好的编辑指令对输入的文本进行编辑，完成之后再输出编辑结构。sed 基本上就是在玩正则模式匹配，所以，玩sed的人，正则表达式一般都比较强。

2、sed工作原理

sed会一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，成为"模式空间"，接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。文件内容并没有改变，除非你使用重定向存储输出。

4. CentOS 7 网卡命名修改为eth0格式(19106)
5. kvm 使用入门详解(17188)

评论排行榜

1. 使用LVS实现负载均衡原理及安装配置详解(22)
2. 网站Web业务架构从小到大演变(6)
3. 数据库MySQL调优实战经验总结(6)
4. sed入门详解教程(5)
5. MySQL架构由小变大的演变过程(4)

推荐排行榜

1. 使用LVS实现负载均衡原理及安装配置详解(30)
2. 数据库MySQL调优实战经验总结(17)
3. MySQL架构由小变大的演变过程(8)
4. sed入门详解教程(8)
5. Linux 库函数与系统调用的关系与区别(7)

目录导航

3、正则表达式概念

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具，换句话说，正则表达式就是记录文本规则的代码。许多程序设计语言都支持利用正则表达式进行字符串操作。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。

4、正则表达式的匹配过程

简单描述一下正则表达式的匹配过程，就是拿正则表达式所表示的字符串去和原文字符串内容去匹配，直到匹配到原文内容字符串中的一个完整子串就表示匹配成功。举个例子，有一行文件内容"this is better desk"，这里用"esk"去匹配，匹配过程是这样的：首先拿e去匹配文件行内容，从this开始，直到better的e，第一个字符匹配成功，接着s去匹配better字符e后边的t字符，没有匹配成功；然后重新拿esk中的e去和better的第二个t去匹配，没有成功，接着原始内容的下一个字符，直到desk中的e字符，逐个匹配s，k字符，到此为止，esk成功匹配，正则表达式匹配完毕，整个过程就是这样，即使再复杂的正则表达式的匹配过程也是按照此过程来进行的。

二、基本正则表达式

关于正则表达式的内容挺多的，掌握好下文中提及的内容就能满足正常工作中的需要，如果是专门做正则编程的，可以去买本正则表达式的书籍来看好了^_^。只有多动手多练习，才是学开发编程的最好姿势。

1. 符号"."

匹配任意一个字符，除了换行符，但是需要注意的是，在sed中不能匹配换行符，但是在awk中可以匹配换行符。类似shell通配符中的"?"，匹配一个任意字符。

2. 符号"*"

"*"表示前边字符有0个或多个。".*"表示任意一个字符有0个或多个，也就是能匹配任意的字符。类似shell通配符中的"*"，可以匹配任意字符。

3. 符号"[]"

"[ ]"中括号中可以包含表示字符集的表达式。使用方法大概有如下几种
[a-z]：表示a-z字符中的一个，也就是小写字母。
[0-9]：表示0-9字符中的一个，也就是表示数字。

[A-Z]：表示大写字母。
[a-zA-Z]：表示字符集为小写字母或者大写字母。

```
[a-zA-Z0-9]：表示普通字符，包括大小写字母和数字。
[abc]：表示字符a或者字符b或者字符c。
[^0-9]：表示非数字类型的字符，^表示取反意思，只能放在中括号的开
[-cz]：表示字符-或者字符c或者字符z，注意与[c-z]的区别，因为-符
```

4. 符号"^"

"^"表示行首的意思，也就是每一行的开始位置。在这里并不是上边字符范围中取反的意思，^符号只有在"[]"符号的开头处才能表示字符取反。

```
^abc：表示以abc开头的字符串abc。
^abc.*：表示以abc开头的字符串abcxxx。
```

5. 符号"$"

"$"表示行尾的意思，也就是每一行的结尾位置，很好理解，和"^"正好相反。

```
world$：表示以world结尾的字符串world，如果该行中间有world字符
^$：表示空行。行首和行尾没有内容，可不就是空行嘛。
```

6. 符号"\"

"\"表示是转义字符，和其它语言中用到的转义字符意义基本上是一样的。其实简单理解，就是把元字符转义为普通字符，比如"\\"表示普通符号"\"，把普通字符转换为特殊意义符号，比如"\n"表示把普通字符n转义为换行符。

7. 符号"{}"

"{}"表示前边字符的数量范围，大概有三种用法，其实容易理解，看例子就知道了，但是必须注意要加上转义字符"\"，否则不生效，表示为普通字符"{"或"}"。

```
\{2\}：表示前边字符的重复次数是2。
\{2,\}：表示前边字符的重复次数至少是2，也就是大于等于2。
\{2,9\}：表示前边字符的重复次数大于2但小于9。
```

8. 符号"\<"和"\>"

"\<"表示匹配条件为词首的位置，理解上可以对比 "^" 行首。举个例子，"nihao 1hello 2hello3 hello4"有这么内容的一行内容。

"\<hello"匹配结果"nihao 1hello 2hello3 hello4"；"hello\>"匹配结果"nihao 1hello 2hello3 hello4"，这种匹配方式用的不是太多，用到会用就OK。

三、扩展正则表达式

扩展正则表达式是在基本正则表达式中扩展出来的，内容不是很多，使用频率上可能没有基本正则表达式那么高，但是扩展正则依然很重要，很多情况下没有扩展正则是搞不定的。sed命令使用扩展正则需要加上选项-r。

1. 符号"?"

"?"：表示前置字符有0个或1个。

2. 符号"+"

"+"：表示前置字符有1个或多个。

3. 符号"|"

"|"：表示指明两项之间的一个选择。

abc|ABC：表示可以匹配abc或者ABC。

4. 符号"()"

"()"表示分组，类似算数表达式中的()。子命令表达式中可以通过\1，\2，\3等来表示分组匹配到的内容。其实"()"也可以在基本正则表达式中使用的。

```
(a|b)b：表示可以匹配ab或者bb字串
([0-9])|([0][0-9])|([1][0-9])：表示匹配0-9或者00-09或者1(
```

5. 符号"{}"

这里的"{}"和基本正则表达式中的大括号意义是一样的，只不过在使用时不用加"\"转义符号。

四、正则表达式的分类和应用

字符类

```
[Ww]hat  \.H[12345]
```

字符的范围

```
[a-z] [0-9] [Cc]hapter[1-9] [-+*/] [0-1][0-9][-/]
[0-3][0-9][-/][0-9][0-9]
```

排除字符类

```
[^0-9]
```

重复出现的字符

```
[15]0*  [15]00
```

字符的跨度

*与\{n,m\}

目录导航

电话号码的匹配

```
[0-9]\{3\}-[0-9]\{7,8\}
```

分组操作

```
compan(y|ies)
```


欢迎关注微信公众号，及时获取新的文章。

五、sed语法和常用选项

1、语法

sed [选项] 'command' 文件名称

选项部分，常见选项包括-n，-e，-i，-f，-r选项。

command部分包括：[地址1，地址2] [函数] [参数(标记)]

2、常用选项

选项-n

sed默认会把模式空间处理完毕后的内容输出到标准输出，也就是输出到屏幕上，加上-n选项后被设定为安静模式，也就是不会输出默认打印信息，除非子命令中特别指定打印选项，则只会把匹配修改的行进行打印。

例子1：

```
echo -e 'hello world\nnihao' | sed 's/hello/A/'
结果：
A world
nihao
```

例子2：

```
echo -e 'hello world\nnihao' | sed -n 's/hello/A/'
结果：加-n选项后什么也没有显示。
```

例子3：

```
echo -e 'hello world\nnihao' | sed -n 's/hello/A/p'
结果：A world/
说明：-n选项后，再加p标记，只会把匹配并修改的内容打印了出来。
```

选项-e

如果需要用sed对文本内容进行多种操作，则需要执行多条子命令来进行操作。

例子1：

```
echo -e 'hello world' | sed -e 's/hello/A/' -e 's/wor
结果：A B
```

例子2：

```
echo -e 'hello world' | sed 's/hello/A/;s/world/B/'
结果：A B
```

说明：例子1和例子2的写法的作用完全等同，可以根据喜好来选择，如果需要的子命令操作比较多的时候，无论是选择-e选项方式，还是选择分号的方式，都会使命令显得臃肿不堪，此时使用-f选项来指定脚本文件来执行各种操作会比较清晰明了。

选项-i

sed默认会把输入行读取到模式空间，简单理解就是一个内存缓冲区，sed子命令处理的内容是模式空间中的内容，而非直接处理文件内容。因此在sed修改模式空间内容之后，并非直接写入修改输入文件，而是打印输出到标准输出。如果需要修改输入文件，那么就可以指定-i选项。

例子1：

```
cat file.txt
hello world
[root@localhost]# sed 's/hello/A/' file.txt
A world
[root@localhost]# cat file.txt
hello world
```

例子2：

```
[root@localhost]# sed -i 's/hello/A/' file.txt
[root@localhost]# cat file.txt
A world
```

例子3：

```
[root@localhost]# sed -i.bak 's/hello/A/' file.txt
```

说明：最后一个例子会把修改内容保存到file.txt，同时会以file.txt.bak文件备份原来未修改文件内容，以确保原始文件内容安全性，防止错误操作而无法恢复原来内容。

选项-f

还记得 -e 选项可以来执行多个子命令操作，用分号分隔多个命令操作也是可以的，如果命令操作比较多的时候就会比较麻烦，这时候把多个子命令操作写入脚本文件，然后使用 -f 选项来指定该脚本。
例子1：

```
echo "hello world" | sed -f sed.script
结果：A B
```

sed.script脚本内容：

```
s/hello/A/
s/world/B/
```

说明：在脚本文件中的子命令串就不需要输入单引号了。

选项-r

sed命令的匹配模式支持正则表达式的，默认只能支持基本正则表达式，如果需要支持扩展正则表达式，那么需要添加-r选项。
例子1：

```
echo "hello world" | sed -r 's/(hello)|(world)/A/g'
A A
```

六、数字定址和正则定址

1、关于定址的概念

默认情况下sed会对每一行内容进行匹配、处理、输出，某些情况不需要对处理的文本全部编辑，只需要其中的一部分，比如1-10行，偶数行，或者是包含"hello"字符串的行，这种情况下就需要我们去定位特定的行来处理，而不是全部内容，这里把这个定位指定的行叫做"定址"。

2、数字定址

数字定址其实就是通过数字去指定具体要操作编辑的行，数字定址有几种方式，每种方式都有不同的应用场景，下边以举例的方式来描述每种数字定址的用法。

例子1：

```
sed -n '4s/hello/A/' message
```

说明：将第4行中hello字符串替换为A，其它行如果有hello也不会被替换。

例子2：

```
sed -n '2,4s/hello/A/' message
```

说明：将第2-4行中hello字符串替换为A，其它行如果有hello也不会被替换。

例子3：

```
sed -n '2,+4s/hello/A/' message
```

说明：从第2行开始，再接着往下数4行，也就是2-6行，这些行会把hello字符替换为A。

例子4：

```
sed -n '4,~3s/hello/A/' message
```

说明：第4行开始，到第6行。解释6的由来，"4,~3"表示从4行开始到下一个3的倍数，这里从4开始算，那就是6了，当然9就不是了，因为是要求3的第一个超过前边数字4的倍数，感觉这种适用场景不会太多。

例子5：

```
sed -n '4~3s/hello/A/' message
```

说明：从第4行开始，每隔3行就把hello替换为A。比如从4行开始，7行，10行等依次+3行。这个比较常用，比如3替换为2的时候，也就是每隔2行的步调，可以实现奇数和偶数行的操作。

例子6：

```
sed -n '$s/hello/A/' message
```

说明：$符号表示最后一行，和正则中的$符号类似，但是第1行不用^表示，直接1就行了。

例子7：

```
sed -n '1!s/hello/A/' message
```

说明：!符号表示取反，该命令是将除了第1行，其它行hello替换为A，上述定址方式也可以使用！符号。

3、正则定址

正则定址使用目的和数字定址完全一样，使用方式上有所不同，是通过正则表达式的匹配来确定需要处理编辑哪些行，其它行就不需要额外处理。

例子1：

```
sed -n '/nihao/d' message
说明：将匹配到nihao的行执行删除操作。
```

例子2：

```
sed -n '/^$/d' message
说明：删除空行
```

例子3：

```
sed -n '/^TS/,/^TE/d' message
说明：匹配以TS开头的行到TE开头的行之间的行，把匹配到的这些行删除
```

4、数字定址和正则定址混用

其实数字定址和正则定址可以配合使用，参考下边的例子。

例子1：

```
sed -n '1,/^TS/d' message
说明：匹配从第1行到TS开头的行，把匹配的行删除。
```

5、关于定址的分组命令

例子1：

```
/^TS/,/^TE/{
s/CN/China/
s/Beijing/BJ/
}
```

说明：该命令表示将从TS开头的行到TE开头的行之间范围的行内容中CN替换为China，并且把Beijing替换为BJ，类似于多命令之间用分号的那种方式，不过这样定址代码只写了一遍，相当于执行了一条子命令。

例子2：

```
sed -n '2,3s{/cn/china/;/a/b/}' message
```

说明：效果类似例子1，有点数学上的乘法分配率的意思。

6、sed定址的总结

sed 默认的命令执行范围是全局编辑的，如果不明确指定行的话，命令会在所有输入行上执行，如果想仅对其中部分行执行命令，

可以使用地址限制。如果给了 2 个地址，即地址对(地址范围)，则命令匹配的这个地址范围内执行，但是需要注意的是：对于像 "addr1，addr2" 这种形式的地址匹配，如果addr1 匹配，则匹配成功，"开关"打开，在该行上执行命令，此时不管 addr2 是否匹配，即使 addr2 在 addr1 这一行之前；接下来读入下一行，如果addr2 匹配，则执行命令，同样开关"关闭"；如果 addr2 在 addr1 之后，则一直处理到匹配为止，换句话说，如果 addr2 一直不匹配，则开关一直不关闭，因此会持续执行命令到最后一行。

目录导航

七、基本子命令

1、子命令a

子命令a表示在指定行下边插入指定行的内容。

例子1：

```
sed 'a A' message
说明：将message文件中每一行下边都插入添加一行内容是A。
```

例子2：

```
sed '1,2a A' message
说明：将message文件中1-2行的下边插入添加一行内容是A
```

例子3：

```
sed '1,2a A\nB\nC' message
说明：将message文件中1-2行的下边分别添加3行，3行内容分别是A、
```

2、子命令i

子命令i和a使用上基本上一样，只不过是在指定行上边插入指定行的内容。

例子1：

```
sed 'i A' message
说明：将message文件中每一行上边都插入添加一行内容是A。
```

例子2：

```
sed '1,2i A' message
说明：将message文件中1-2行的上边插入添加一行内容是A
```

例子3：

```
sed '1,2i A\nB\nC' message
说明：将message文件中1-2行的上边分别添加3行，3行内容分别是A、
```

3、子命令c

子命令c是表示把指定的行内容替换为自己需要的行内容。

例子1：

```
sed 'c A' message
说明：将message文件中所有的行内容都分别替换为A行内容。
```

例子2：

```
sed '1,2c A' message
```

说明：将message文件中1-2行的内容替换为A，注意这里说的是将1-2行所有的内容只替换为一个A内容，也就是1-2行内容编程了一行，定址如果连续就是这种情况。如果想把1-2行分别替换为A，可以参考下个例子的方式。

例子3：

```
sed '1,2c A\nA' message
说明：将message中1-2行内容分别替换为了A，需要在替换内容上手动
```

4、子命令d

子命令d表示删除指定的行内容，比较简单，更容易理解。

例子1：

```
sed 'd' message
说明：将message所有行全部删除，因为没有加定址表达式，所以平时如
```

例子2：

```
sed '1,3d' message
说明：将message文件中1-3行内容删除。
```

5、子命令y

子命令y表示字符替换，可以替换多个字符，只能替换字符不能替换字符串，且不支持正则表达式，具体使用方法看例子。

例子1：

```
sed 'y/ab/AB/' message
说明：把message中所有a字符替换为A符号，所有b字符替换为B符号。
```

强调一下，这里的替换源字符个数和目的字符个数必须相等；字符不支持正则表达式；源字符和目标字符每个字符需要一一对应。

6、子命令=

目录导航

子命令=，可以将行号打印出来。

例子：

```
sed '1,2=' message
```

结果：

```
1
nihao
2
hello world
```

说明：将指定行的上边显示行号。

7、子命令r

子命令r，类似于a，也是将内容追加到指定行的后边，只不过r是将指定文件内容读取并追加到指定行下边。

例子1：

```
sed '2r a.txt' message
```

说明：将a.txt文件内容读取并插入到message文件第2行的下边。

8、子命令s

子命令s为替换子命令，是平时sed使用的最多的子命令，没有之一。因为支持正则表达式，功能变得强大无比，下边来详细地说说子命令s的使用方法。

基本语法：

[address]s/pattern/replacement/flags

s字符串替换，替换的时候可以把/换成其它的符号，比如=，replacement部分用下列字符会有特殊含义：

```
>>>  &：用正则表达式匹配的内容进行替换
>>>  \n：回调参数
>>>  \(\)：保存被匹配的字符以备反向引用\n时使用，最多9个标签，
```

Flags

```
>>>  n：可以是1-512，表示第n次出现的情况进行替换
>>>  g：全局更改
>>>  p：打印模式空间的内容
>>>  w file：写入到一个文件file中
```

实例用法

测试文件：

```
# cat message
hello 123 world
```

目录导航

例子1：

```
sed 's/hello/HELLO/' message
说明：将message每行包含的第一个hello的字符串替换为HELLO，这是
```

例子2：

```
sed -r 's/[a-z]+ [0-9]+ [a-z]+/A/' message
结果：A
说明：使用了扩展正则表达式，需要加-r选项。
```

例子3：

```
sed -r 's/([a-z]+)( [0-9]+ )([a-z]+)/\1\2\3/' message
结果：hello 123 world
说明：再看下一个例子就明白了。
```

例子4：

```
sed -r 's/([a-z]+)( [0-9]+ )([a-z]+)/\3\2\1/' message
结果：world 123 hello
说明：\1表示正则第一个分组结果，\2表示正则匹配第二个分组结果，\
```

例子5：

```
sed -r 's/([a-z]+)( [0-9]+ )([a-z]+)/&/' message
结果：hello 123 world
说明：&表示正则表达式匹配的整个结果集。
```

例子6：

```
sed -r 's/([a-z]+)( [0-9]+ )([a-z]+)/111&222/' messag
结果：111hello 123 world222
说明：在匹配结果前后分别加了111、222。
```

例子7：

```
sed -r 's/.*/111&222/' message
说明：在message文件中每行的首尾分别加上111、222。
```

例子8：

```
sed 's/i/A/g' message
说明：把message文件中每行的所有i字符替换为A，默认不加g标记时只
```

例子9：

```
sed 's/i/A/2' message
说明：把message文件中每行的第2个i字符替换为A。
```

例子10：

```
sed -n 's/i/A/p' message
说明：加-p标记会把被替换的行打印出来，再加上-n选项会关闭模式空间
```

例子11：

```
sed -n 's/i/A/w b.txt' message
说明：把message文件中内容的每行第一个字符i替换为A，然后把修改内
```

例子12：

```
sed -n 's/i/A/i' message
说明：把message文件中每一行的第一个i或I字符替换为A字符，也即是
```

八、sed工作模式

1、模式空间和保持空间

模式空间初始化为空，处理完一行后会自动输出到屏幕并清除模式空间；保持空间初始化为一个空行，也就是默认带一个\n，处理完后不会自动清除。模式空间和保持空间，从程序的角度去看，其实就是sed在工作的时候占用了一些内存空间和地址，sed工作完毕就会把内存释放并归还给操作系统。

2、sed工作流程

大概简单描述一下sed的工作流程，读取文件的一行，存入模式空间，然后进行所有子命令的处理，处理完后默认会将模式空间的内容输出打印到标准输出，也就是在屏幕上显示出来，接着清空模式空间的内存，继续读取下一行的内容到模式空间，继续处理，依次循环处理。

3、模式空间和保持空间的置换

h：把模式空间内容覆盖到保持空间中
H：把模式空间内容追加到保持空间中
g：把保持空间内容覆盖到模式空间中
G：把保持空间内容追加到模式空间中
x：交换模式空间与保持空间的内容

4、实例用法

测试文件：

```
# cat test.txt
11111
22222
```

目录导航

```
33333
44444
```

例子1：

```
sed '{1h;2,3H;4G}' test.txt
结果：
11111
22222
33333
44444
11111
22222
33333
解释说明：略。懒得写了。
```

例子2：

```
sed '{1h;2x;3g;$G}' test.txt
结果：
11111
11111
22222
44444
22222
解释说明：略。
```

例子3：

```
sed '{1!G;h;$!d}' test.txt
结果：
44444
33333
22222
11111
```

九、高级子命令

高级子命令比较少，但是比较复杂，平时用的也会相对少些，却也很重要，有的内容处理不用高级子命令是完成不了的。

n：读入下一行到模式空间，例：'4{n;d}' 删除第5行。

N：追加下一行到模式空间，再把当前行和下一行同时应用后面的命令。

P：输出多行模式空间的第一部分，直到第一个嵌入的换行符位置。在执行完脚本的最后一个命令之后，模式空间的内容自动输出。P命令经常出现在N命令之后和D命令之前。

D：删除模式空间中第一个换行符的内容。它不会导致读入新的输入行，相反，它返回到脚本的顶端，将这些指令应用与模式空间剩余的内容。这3个命令能建立一个输入、输出循环，用来维护两行模式空间，但是一次只输出一行。

例子1：

```
sed 'N;$!P;D' a.txt
#说明：删除文件倒数第二行
```

例子2：

```
sed 'N;$!P;$!D;$d' a.txt
# 说明：删除文件最后两行
```

十、分支和测试

分支命令用于无条件转移，测试命令用于有条件转移。

1、分支branch

跳转的位置与标签相关联。

如果有标签则跳转到标签所在的后面行继续执行。

如果没有标签则跳转到脚本的结尾处。

标签：以冒号开始后接标签名，不要在标签名前后使用空格。

2、跳转到标签指定位置

测试文件：

```
grep seker /etc/passwd
seker:x:500:500::/home/seker:/bin/bash
```

例子1：

```
grep seker /etc/passwd | sed ':top;s/seker/blues/;/se
结果：blues:x:55500:500::/home/blues:/bin/bash
```

选择执行

例子2：

```
grep 'seker' /etc/passwd | sed 's/seker/blues/;/seker
结果：blues:x:66600:500::/home/seker:/bin/bash
```

测试命令，如果前一个替换命令执行成功则跳转到脚本末尾(case结构)

例子3：

```
grep 'seker' /etc/passwd | sed 's/seker/ABC;t;s/home/
结果：ABC:x:500:500::/home/seker:/bin/bash
```

例子4：

```
grep 'zorro' /etc/passwd | sed 's/seker/ABC/;t;s/home
结果：zorro:x:500:500::/DEF/zorro:/bin/bash
```

与标签关联，跳转到标签位置。

例子5：

```
grep 'seker' /etc/passwd | sed 's/seker/ABC/;t end;s/
结果：ABC:x:500:500::/home/seker:/bin/XYZ
```

十一、sed实战练习

实例1：删除文件每行的第二个字符。

```
sed -r 's/(.*)(.)$/\1/'
```

实例2：删除文件每行的最后一个字符。

```
sed -r 's/(.*)(.)$/\1/'
```

实例3：删除文件每行的倒数第2个单词。

```
sed -r 's/(.*)([^a-Z]+)([a-Z]+)([^a-Z]+)([a-Z]+)([^a-
```

实例4：交换每行的第一个字符和第二个字符。

```
sed -r 's/(.)(.)(.*)/\2\1\3/' /etc/passwd
```

实例5：交换每行的第一个单词和最后一个单词。

```
sed -r 's/([a-Z]+)([^a-Z]+)(.*)([^a-Z]+)([a-Z]+)([^a-
```

实例6：删除一个文件中所有的数字。

```
sed 's/[0-9]//g' /etc/passwd
```

实例7：用制表符替换文件中出现的所有空格。

```
sed -r 's/ +/\t/g' /etc/passwd
```

实例8：把所有大写字母用括号()括起来。

```
sed -r 's/([A-Z])/(\1)/g' /etc/passwd
```

实例9：打印每行3次。

```
sed 'p;p' /etc/passwd
```

实例10：隔行删除

```
sed '0~2{=;d}' /etc/passwd
```

实例11：把文件从第22行到第33行复制到56行后面。

```
sed '22h;23,33H;56G' /etc/passwd
```

实例12：把文件从第22行到第33行移动到第56行后面。

```
sed '22{h;d};23,33{H;d};56g' /etc/passwd
```

实例13：只显示每行的第一个单词。

```
sed -r 's/([a-Z]+)([^a-Z]+)(.*)/\1/' /etc/passwd
```

实例14：打印每行的第一个单词和第三个单词。

```
sed -r 's/([a-Z]+)([^a-Z]+)([a-Z]+)([^a-Z]+)([a-Z]+)(
```

实例15：将格式为mm/yy/dd的日期格式换成 mm;yy;dd

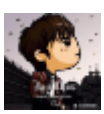
```
date '+%m/%y/%d' | sed 's/\//;/g'
```

欢迎关注微信公众号，及时获取新的文章。

分类: 肖邦Linux - 04.Shell

标签: sed

好文要顶 关注成功 收藏该文



肖邦linux
粉丝 - 239 关注 - 17

目录导航

关注成功

9 推荐　0 反对

支持成功

« 上一篇： Linux系统监控命令之iotop

» 下一篇： CentOS7安装图形界面和修改运行级别

posted on 2016-07-05 16:33 肖邦linux 阅读(47605) 评论(5) 编辑 收藏 举报

目录导航

评论：

默认 | **按时间** | 按支持数

#1楼 2019-02-12 21:21 | 滞销书读者

受教了，谢谢博主，还有请问博主这个目录导航怎么弄出来呢?

支持(0) 反对(0)

回复 引用

#2楼 2019-03-13 20:26 | iOS小熊

sed -i 's/hello/A/' file.txt

会直接报错 sed :1: "file.txt" :invalid command code f

环境是mac os

必须要 sed -i '' 's/hello/A/' file.txt 这样才行，中间要多个''，不知道是为什么?

原因找到了，确实是mac os的锅。

-i extension

Edit files in-place, saving backups with the specified extension. If a zero-length extension is given, no backup will be saved. It is not recommended to give a zero-length

extension when in-place editing files, as you risk corruption or partial content in situa-

tions where disk space is exhausted, etc.

-i 操作后面要跟一个extension参数，明确备份的文件

so 可以是空串 也可以是备份文件后缀字符串

支持(0) 反对(0)

回复 引用

#3楼 2020-02-29 10:43 | 铁皮石斛

你好，我想请教一个问题，说一个文件有数G之大，现在的需求是修改前30行中的 abc 变成 ABC,如

何用 sed 实现呢?

类似下面的操作，他要把整个文件都遍历一遍，而不是仅仅处理到30行。

```
sed -i 's/abc/ABC/' file.txt
```

或者说用别的什么方法?

支持(0) 反对(0

回复 引用

#4楼 2020-07-06 05:22 | 张缔

@铁皮石斛
sed -i '1,30s/abc/ABC/' file.txt 这样不行吗

支持(0) 反对(0)

回复 引用

#5楼 2020-09-07 14:40 | 铁皮石斛

@张缔
这样可以一半，这个与不加 1,30 的区别就是他只会在 1到30号进行替换，别的行他就不去匹配了。

但是，但是，剩下的行他一样要“走”一遍，而不是处理完这30号就停下了脚步，在文件有数G之大时，非常慢。

支持(0) 反对(0)

回复 引用

刷新评论 刷新页面 返回顶部

发表评论 升级成为园子VIP会员

编辑 预览

支持 Markdown

自动补全

提交评论 退出 订阅评论 我的博客

[Ctrl+Enter快捷键提交]

目录导航

目录导航

编辑推荐：
· 服务端不回应客户端的 syn 握手，连接建立失败原因排查
· 超强的 Anchor Positioning 锚点定位
· [MAUI] 在 .NET MAUI 中实现可拖拽排序列表
· 使用 MediatR 实现 CQRS
· 记一次 .NET某报关系统 非托管泄露分析

阅读排行：
· 开源项目自荐：截图工具（小、快、功能丰富）
· 使用C#创建安装Windows服务程序(干货)
· 记录一次内网渗透过程
· Redis专题-秒杀
· 服务端不回应客户端的syn握手，连接建立失败原因排查