sed 命令

> 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。

sed 的基本使用

语法

```
sed [-ihnV][-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(脚本)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(脚本文件)来处理输入的文本文件。
- -i 直接修改文件。
- -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 的动作可以搭配正规表示法！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：

```
[root@localhost sedtest]# cat testfile
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：

```
[root@localhost sedtest]# 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！

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，即执行过一次 sed 命令后就结束，不再进入循环。

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 ip 命令查询 IP：

```
$ ip a s eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 00:0c:29:12:1d:e0 brd ff:ff:ff:ff:ff:ff
    inet 172.16.100.22/24 brd 172.16.100.255 scope global eth0
    inet6 fe80::20c:29ff:fe12:1de0/64 scope link
       valid_lft forever preferred_lft forever
```

本机的 ip 是 172.16.100.22

先只显示存在 IP 的一行：

```
$ ip a s eth0 | sed -n '/inet /p'
    inet 172.16.100.22/24 brd 172.16.100.255 scope global eth0
```

注意，单引号里面 inet 后面有空格

接下来则是删除后续的部分以及前面的 inet：

```
$ ip a s eth0 | sed -n '/inet /{s/brd.*$//;s/inet //;p}'
    172.16.100.22/24
```

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-
```

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
```

由于 $ 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！

追加行的说明

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

a 动作是在匹配的行之后追加字符串，追加的字符串中可以包含换行符（实现追加多行的情况）。

追加一行的话前后都不需要添加换行符 **\n**，只有追加多行时在行与行之间才需要添加换行符(最后一行最后也无需添加，添加的话会多出一个空行)。

man sed 信息：

```
Append text, which has each embedded newline preceded by a backslash.
```

例如：

4 行之后添加一行：

```
sed -e '4 a newline' testfile
```

4 行之后追加 2 行：

```
sed -e '4 a newline\nnewline2' testfile
```

4 行之后追加 3 行(2 行文字和 1 行空行)

```
sed -e '4 a newline\nnewline2\n' testfile
```

4 行之后追加 1 行空行：

```
#错误：sed -e '4 a \n' testfile
sed -e '4 a \ ' testfile 实际上
```

实际上是插入了一个含有一个空格的行

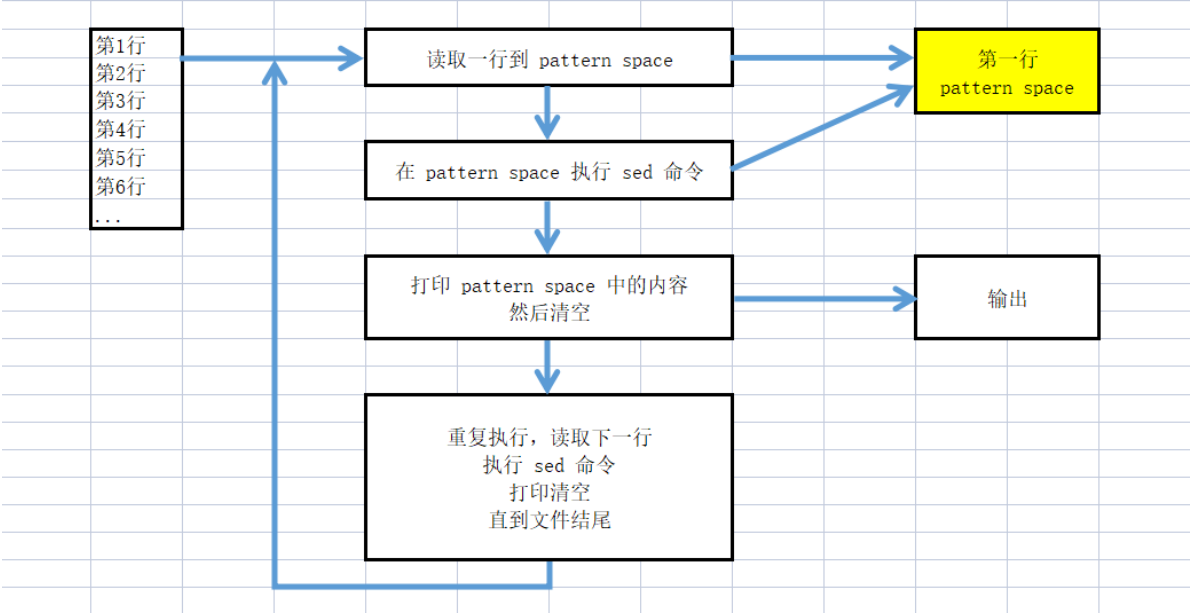
添加空行：

```
# 可以添加一个完全为空的空行
sed '4 a \\'

# 可以添加两个完全为空的空行
sed '4 a \\n'
```

sed 的高级使用

首先需要理解下 sed 的运行机制，sed 具有两个数据缓存区，一个所谓 pattern 区，一个所谓 hold 区，一行数据过来在 pattern 区执行 sed 处理命令，然后默认输出 pattern 区的内容（除非 -n 消除自动打印），然后会清空 pattern 区（默认），进行下一行数据操作（下一次循环）。hold 区可以在循环间保持住本区数据不删除，sed 的一些命令可以在 pattern 区和 hold 区之间 move data。

图解



多行命令的使用

动作说明（注意，以下命令都为大写）：

- N 将数据流中的下一行加进来创建一个多行组来处理
- D 删除多行组中的一行
- P 打印多行组中的一行

N 多行操作命令

> N 命令会将下一行文本内容添加到 pattern 区已有数据之后（之间用换行符分隔），从而使前后两个文本行同时位于 pattern 区中，sed 命令会将这两行数据当成一行来处理。

实例1

```
[root@localhost sedtest]# cat test
This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
[root@localhost sedtest]# sed '/first/{N;s/\n/ /}' test
This is the header line.
This is the first data line. This is the second data line.
This is the last line.
```

在这个例子中，sed 命令查找含有单词 first 的那行文本。找到该行后，它会用 N 命令将下一行合并到那行，然后用替换命令 s 将换行符替换成空格。结果是，文本文件中的两行在 sed 的输出中成了一行。

实例2

```
[root@localhost sedtest]# cat test2
On Tuesday, the Linux System
Administrator's group meeting will be held.
All System Administrators should attend.
Thank you for your attendance.
[root@localhost sedtest]# sed 'N;s/System.Administrator/Desktop User/' test2
On Tuesday, the Linux Desktop User's group meeting will be held.
All Desktop Users should attend.
Thank you for your attendance.
```

用 N 命令将发现第一个单词的那行和下一行合并后，即使短语内出现了换行，你仍然可以找到它，这是因为，替换命令在 System 和 Administrator 之间用了通配符（.）来匹配空格和换行符这两种情况。但当它匹配了换行符时，它就从字符串中删掉了换行符，导致两行合并成一行。这可能不是你想要的。

要解决这个问题，可以在 sed 脚本中用两个替换命令，一个用来匹配短语出现在多行中的情况，一个用来匹配短语出现在单行中的情况，比如：

```
[root@localhost ~]# sed 'N
> s/System\nAdministrator/Desktop\nUser/
> s/System Administrator/Desktop User/
> ' test2
On Tuesday, the Linux Desktop
User's group meeting will be held.
All Desktop Users should attend.
Thank you for your attendance.
```

D 多行删除命令

> sed 不仅提供了单行删除命令（d），也提供了多行删除命令 D，其作用是只删除 pattern 区中的第一行，也就是说，D 命令将 pattern 区中第一个换行符（包括换行符）之前的内容删除掉。

实例1

```
[root@localhost sedtest]# sed 'N;/System\nAdministrator/D' test2
Administrator's group meeting will be held.
All System Administrators should attend.
Thank you for your attendance.
```

文本的第二行被 N 命令加到了 pattern 区，因此 sed 命令第一次匹配就是成功，而 D 命令会将 pattern 区中第一个换行符之前（也就是第一行）的数据删除，所以，得到了如上所示的结果。

实例2

```
[root@localhost sedtest]# cat test3

This is the header line.
This is a data line.

This is the last line.
[root@localhost ~]# sed '/^$/{N;/header/D}' test3
This is the header line.
This is a data line.

This is the last line.
```

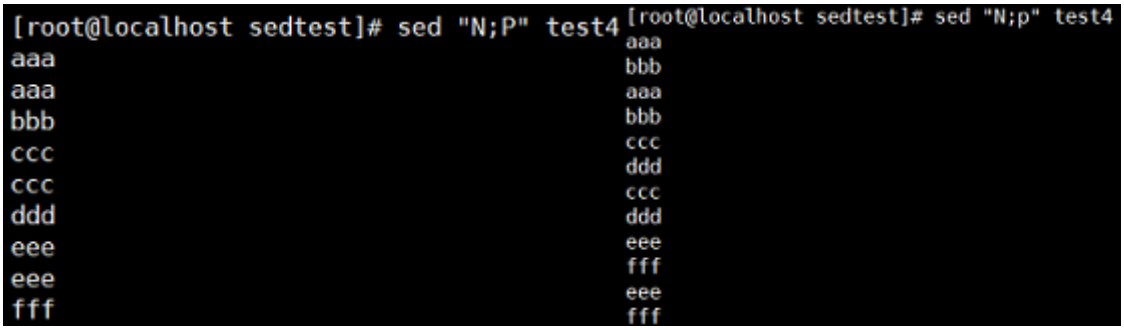
sed 会查找空白行，然后用 N 命令来将下一文本行添加到 pattern 区。此时如果 pattern 区的内容中含有单词 header，则 D 命令会删除 pattern 区中的第一行。

P 多行打印命令

> 同 d 和 D 之间的区别一样，P（大写）命令和单行打印命令 p（小写）不同，对于具有多行数据的 pattern 区来说，它只会打印 pattern 区中的第一行，也就是首个换行符之前的所有内容。

实例

```
[root@localhost sedtest]# cat test4
aaa
bbb
ccc
ddd
eee
fff
```



第一个 sed 命令，每次都使用 N 将下一行内容追加到 pattern 区内容的后面（用换行符间隔），也就是说，第一次时 pattern 区中的内容为 aaa\nbbb，但 P（大写） 命令的作用的打印换行符之前的内容，也就是 aaa，之后则是 sed 在自动输出功能输出 aaa 和 bbb（sed 命令会自动将 \n 输出为换行），依次类推，就输出了所看到的结果。

第二个 sed 命令，使用的是 p （小写）单行打印命令，它会将 pattern 区中的所有内容全部打印出来（\n 会自动输出为换行），因此，出现了看到的结果。

sed 保持空间

前面我们一直说，sed 命令处理的是 pattern 区中的内容，其实这里的 pattern 区，应称为模式空间。值得一提的是，模式空间并不是 sed 命令保存文件的唯一空间。sed 还有另一块称为保持空间的缓冲区域，它可以用来临时存储一些数据，一般称作 hold 区。

动作说明：

- h 将模式空间中的内容复制到保持空间
- H 将模式空间中的内容附加到保持空间
- g 将保持空间中的内容复制到模式空间
- G 将保持空间中的内容附加到模式空间
- x 交换模式空间和保持空间中的内容

> 通常，在使用 h 或 H 命令将字符串移动到保持空间后，最终还要用 g、G 或 x 命令将保存的字符串移回模式空间。保持空间最直接的作用是，一旦我们将模式空间中所有的文件复制到保持空间中，就可以清空模式空间来加载其他要处理的文本内容.

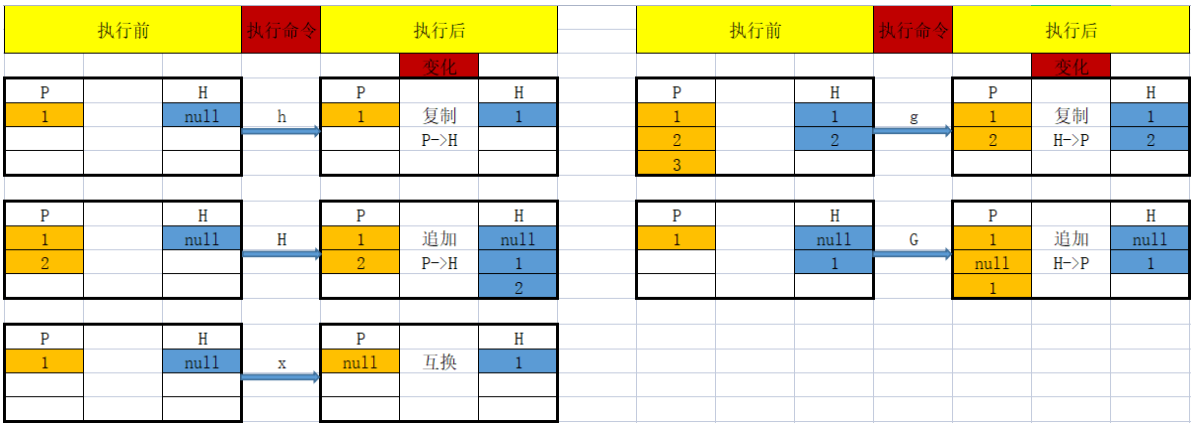
图解 分析过程

P : Pattern Space

H : Hold Space

蓝色 : Hold Space 中的数据

橙色 : Pattern Space 中的数据



test :

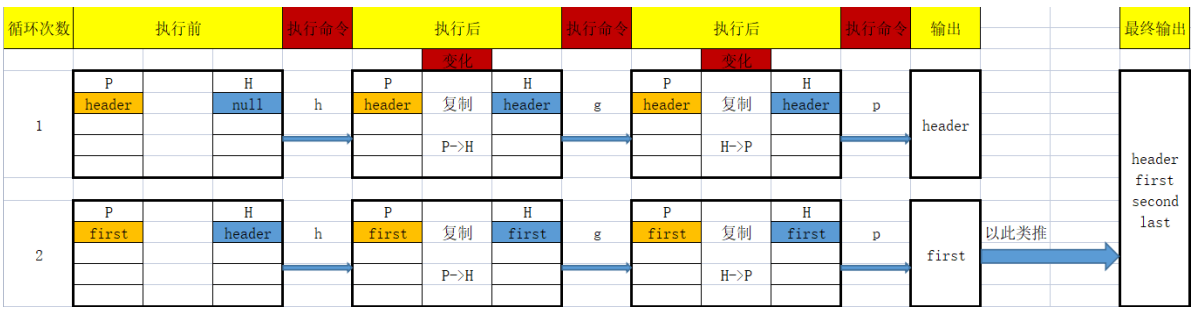
```
[root@localhost sedtest]# cat test
This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
```

实例1: h，g 配合使用

```
[root@localhost sedtest]# sed -n 'h;g;p' test
This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
```

执行 h 后 pattern 区的内容与 hold 区的内容已经相同，再执行 g 其实是多此一举，p 输出的是 pattern 区的全部内容，然后默认执行一次 d 命令清空 pattern 区的全部内容，但不会操作 hold 区的内容，然后进行下一行数据操作（下一次循环）

图解

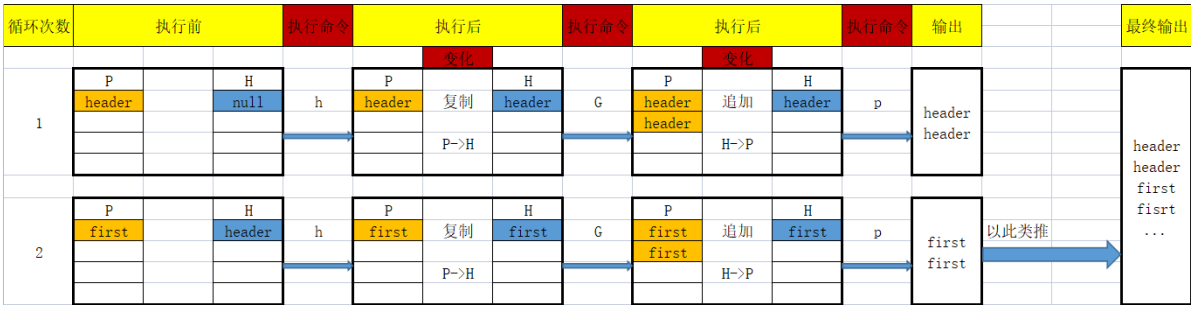


实例2: h，G 配合使用

```
[root@localhost sedtest]# sed -n 'h;G;p' test
This is the header line.
This is the header line.
This is the first data line.
This is the first data line.
This is the second data line.
This is the second data line.
This is the last line.
This is the last line.
```

执行 h 后 pattern 区的内容与 hold 区的内容变得相同，再执行 G 命令，则输出的内容，相当于把 pattern 区的内容再复制一行放在下一行进行输出，只是在 hold 区中多了 执行前 pattern 区的数据，但是在下一次循环执行 g 命令后，之前 hold 区中有没有内容已经关系不大了，都会被第二次循环时 pattern 区中的内容所复制过去进行覆盖

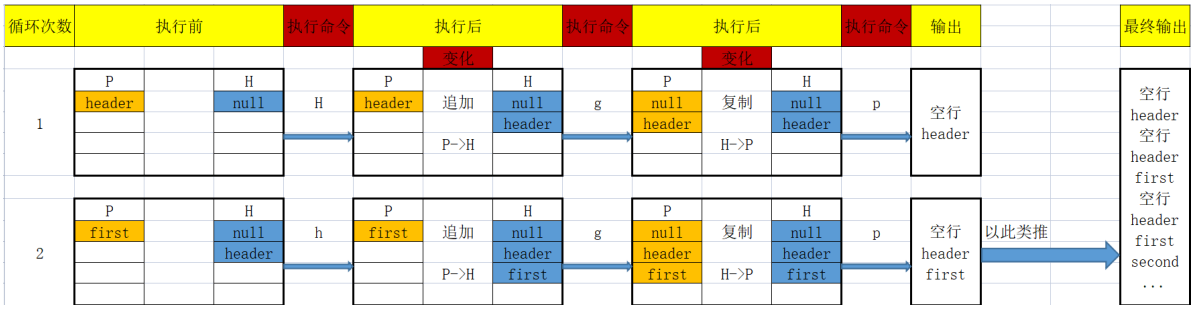
图解



实例3: H，g 配合使用

```
[root@localhost sedtest]# sed -n 'H;g;p' test

This is the header line.

This is the header line.
This is the first data line.

This is the header line.
This is the first data line.
This is the second data line.

This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
```

先看图解

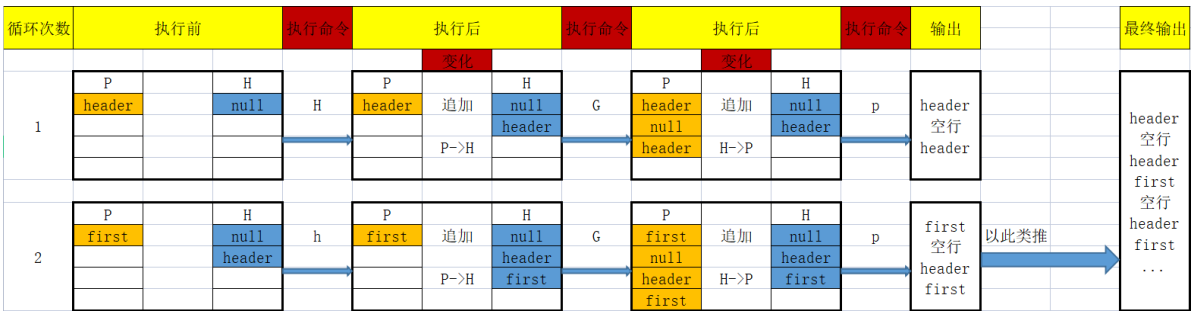


hold 区中默认初始状态是为空，但是当最开始执行 H 命令将 pattern 区中的内容追加到 hold 区的时候，会追加到一个空行下面，我们权当 hold 区中默认有个空行就行了，而在执行完第一次循环后，hold 区中的内容是不会清除的，所以第二次追加输出会是上图所示，以此类推

实例4: H，G 配合使用

```
[root@localhost sedtest]# sed -n 'H;G;p' test
This is the header line.

This is the header line.
This is the first data line.

This is the header line.
This is the first data line.
This is the second data line.

This is the header line.
This is the first data line.
This is the second data line.
This is the last line.

This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
```

这个空行可能会扰乱你的思路，**先看图解**



错误思路：

为什么空行容易扰乱思路呢，因为可能有人直接看输出的时候，可能会将 **header 空行** 当成第一次循环的输出，**header second 空行** 当成第二次循环的输出，最后又多出四行最后还没空格，然后就蒙了

正确思路：

在 H;g 用法的每次循环结束输出的内容上多加一行当前循环 pattern 区开始读取的内容，如下图所示

```
[root@localhost sedtest]# sed -n 'H;G;p' test
This is the header line.

This is the header line.
This is the first data line.

This is the header line.
This is the first data line.
This is the second data line.

This is the header line.
This is the first data line.
This is the second data line.
This is the last line.

This is the header line.
This is the first data line.
This is the second data line.
This is the last line.
```

豁然开朗有木有

实例5: x 的使用

```
[root@localhost sedtest]# seq 1 | sed -n "p;x;p;G;p"
1


1
```

先看图解

配合 G 命令的简单使用，先输出了一次 pattern 区的内容，执行 x 命令后将 pattern 区的内容与 hold 区的内容交换后再输出，此时输出的是空行，而上一行的 “1” 是上一次的输出，再执行 G 命令把 hold 区的内容追加到 pattern 区中进行输出，输出的是 “空行和1”，前面的内容是前面已经输出的内容，最终输出这样的结果

sed 改变指定流程

b 分支命令

通常，sed 程序的执行过程会从第一个脚本命令开始，一直执行到最后一个脚本命令（b 命令是个例外，它会强制 sed 返回到脚本的顶部，而不读取新的行）。sed 提供了 b 分支命令来改变命令脚本的执行流程，其结果与结构化编程类似。

b 分支命令基本格式为：

```
[address]b [label]
```

其中，address 参数决定了哪些行的数据会触发分支命令，label 参数定义了要跳转到的位置。

需要注意的是，如果没有加 label 参数，跳转命令会跳转到脚本的结尾，比如：

```
[root@localhost ~]# sed '{2,3b;s/This is/Is this/;s/line./test?/}' test
Is this the header test?
This is the first data line.
This is the second data line.
Is this the last test?
```

可以看到，因为 b 命令未指定 label 参数，因此数据流中的第2行和第3行并没有执行那两个替换命令。

如果我们不想直接跳到脚本的结尾，可以为 b 命令指定一个标签（也就是格式中的 label，最多为 7 个字符长度）。在使用此该标签时，要以冒号开始（比如 :label2），并将其放到要跳过的脚本命令之后。这样，当 sed 命令匹配并处理该行文本时，会跳过标签之前所有的脚本命令，但会执行标签之后的脚本命令。

比如说：

```
[root@localhost ~]# sed '{/first/b jump1;s/This is the/No jump on/
> :jump1
> s/This is the/Jump here on/}' test
No jump on header line
Jump here on first data line
No jump on second data line
No jump on last line
```

在这个例子中，如果文本行中出现了 first，程序的执行会直接跳到 jump1 标签之后的脚本行。如果分支命令的模式没有匹配，sed 会继续执行所有的脚本命令。

b 分支命令除了可以向后跳转，还可以向前跳转，例如：

```
[root@localhost ~]# echo "This, is, a, test, to, remove, commas." | sed -n
'{
> :start
> s/,//1p
> /,/b start
> }'
This is, a, test, to, remove, commas.
This is a, test, to, remove, commas.
This is a test, to, remove, commas.
This is a test to, remove, commas.
This is a test to remove, commas.
This is a test to remove commas.
```

在这例子中，当缓冲区中的行内容中有逗号时，脚本命令就会一直循环执行，每次迭代都会删除文本中的第一个逗号，并打印字符串，直至内容中没有逗号。

t 测试命令

类似于 b 分支命令，t 命令也可以用来改变 sed 脚本的执行流程。t 测试命令会根据 s 替换命令的结果，如果匹配并替换成功，则脚本的执行会跳转到指定的标签；反之，t 命令无效。

测试命令使用与分支命令相同的格式：

```
[address]t [label]
```

跟分支命令一样，在没有指定标签的情况下，如果 s 命令替换成功，sed 会跳转到脚本的结尾（相当于不执行任何脚本命令）。例如：

```
[root@localhost ~]# sed '{
> s/first/matched/
> t
> s/This is the/No match on/
> }' test
No match on header line
This is the matched data line
No match on second data line
No match on last line
```

此例中，第一个替换命令会查找模式文本 first，如果匹配并替换成功，命令会直接跳过后面的替换命令；反之，如果第一个替换命令未能匹配成功，第二个替换命令就会被执行。

再举个例子：

```
[root@localhost ~]# echo "This, is, a, test, to, remove, commas. " | sed -n '{
> :start
> s/,//1p
> t start
> }'
This is, a, test, to, remove, commas.
This is a, test, to, remove, commas.
This is a test, to, remove, commas.
This is a test to, remove, commas.
This is a test to remove, commas.
This is a test to remove commas.
```

附: sed 的帮助文档

```
[root@localhost sedtest]# sed --help
用法: sed [选项]... {脚本(如果没有其他脚本)} [输入文件]...

  -n, --quiet, --silent
                 取消自动打印模式空间
  -e 脚本, --expression=脚本
                 添加“脚本”到程序的运行列表
  -f 脚本文件, --file=脚本文件
                 添加“脚本文件”到程序的运行列表
  --follow-symlinks
                 follow symlinks when processing in place; hard links
                 will still be broken.
  -i[SUFFIX], --in-place[=SUFFIX]
                 edit files in place (makes backup if extension supplied).
                 The default operation mode is to break symbolic and hard
links.
                 This can be changed with --follow-symlinks and --copy.
  -c, --copy
                 use copy instead of rename when shuffling files in -i mode.
                 While this will avoid breaking links (symbolic or hard),
the
                 resulting editing operation is not atomic.  This is rarely
                 the desired mode; --follow-symlinks is usually enough, and
                 it is both faster and more secure.
```

```
  -l N, --line-length=N
                指定“l”命令的换行期望长度
  --posix
                关闭所有 GNU 扩展
  -r, --regexp-extended
                在脚本中使用扩展正则表达式
  -s, --separate
                将输入文件视为各个独立的文件而不是一个长的连续输入
  -u, --unbuffered
                从输入文件读取最少的数据，更频繁的刷新输出
      --help    打印帮助并退出
      --version  输出版本信息并退出

如果没有 -e, --expression, -f 或 --file 选项，那么第一个非选项参数被视为
sed脚本。其他非选项参数被视为输入文件，如果没有输入文件，那么程序将从标准
输入读取数据。
```

参考来源：Linux sed 命令 | 菜鸟教程 (runoob.com)

Linux sed命令高级用法精讲 (biancheng.net)

sed之G、H、g、h使用 - fhefh - 博客园 (cnblogs.com)