tuicool.com

info sed 中文不完全文档 - 推酷

小提示：

（1）“自定备注”后的内容是自己添加上去用来提醒自己的，非原文内容。

（2）对于存疑的地方我添加了原文。

（3）测试环境：centos 6.5 + GNU sed 4.2.1

正文：

该文档记录 GNU 4.2.1 版本的流编辑器：sed

****

菜单：

* Introduction::

* Invoking sed:: 调用方法；

* sed Pragrams::

* Examples:: 一些示例脚本；（尚未翻译）

* Limitations:: GNU sed 的一些限制和非限制；

* Other Resources:: 一些关于了解 sed 的其他资源；

1.Introduction

****

sed 是一个流编辑器。一个流编辑器被用来执行基于输入流（一个文件或者来自管道的输入）的基础文本转化。虽然在某些方面和一个允许脚本化编辑的编辑器（例如 ed）相同，sed 通过使得只有一行越过输入来工作，因此更加的有效（'sed' works by making only one pass over input(s), and is consequently more efficient）。但是在一个管道中过滤文本是 sed 的能力，这明确的使得它和其他类型的编辑器区分开来。

2.Invoking sed

****

通常 sed 的调用是：

sed SCRIPT INPUTFILE...

调用 sed 的完整格式是：

sed OPTIONS... [SCRIPT] [INPUTFILE]

--version

打印当前正在运行的 sed 程序的版本以及 copyright 注意事项，然

后退出；

--help

打印简洁的用法，这些用法总结了这些命令行选项，并给出了 bug-reporting 地址，然后退出；

-n

--quiet

--silent

在默认情况，sed 在每一个周期的结束时通过脚本打印出 pattern space（见 How sed works: Execution Cycle）。这些选项禁用了自动打印，sed 仅仅当被 'p' 命令显式的告知时才处理输出。

-e script

--expression=script

当处理输入时，将包含在 script 中的命令添加到将要被运行的命令集合中；

自定备注：

（1）在同时使用 -e 和 -n 选项时，-n 选项应该放在前面，即 -ne 而不是 -en ；从另一个角度来讲，-e 选项后面必须要立即接 script。

（2）可以使用多个 -e 选项来分割不同的命令，例如：

$ sed -e '1,3' -e 's/test/testing/' file-name

注意：sed 对于每一行按照从左到右的方法对每一个脚本的触发条件进行测试，所以前面的脚本执行结果可能会影响到后面的脚本执行结果。

-f script-file

--file=script-file

当处理输出时，将包含在 script-file 中的命令添加到将被执行的命令集合中；

-i[suffix]

--in-place[=suffix]

这个选项声明了这个文件将被合适的编辑。GNU sed 达成这一点是通过创建一个临时文件，然后将输出送至这个文件中而不是送至标准输出中（脚注 1）。

这个选项隐含了 -s 选项；

当到达文件结尾的时候，这个临时文件被重命名为输出文件的原始名字。这个拓展如果被应用，在重名临时文件之前，将被用来修改旧文件的名字，因此对旧文件做了一个备份复制（see "How sed works?"）（脚注 2）。

这个规则将被遵循：如果这个拓展并没有包含一个 * ，然后它会被添加到当前文件名的后面作为一个后缀；如果这个拓展包含了一个或者多个 * 字符，然后每一个 * 被取代为当前的文件名（正在处理的

文件）。这就允许你添加一个前缀给备份文件，而不是一个后缀，或者甚至是将原文件的备份副本放到其他目录（提供了已经存在的目录）。

如果 suffix 没有被应用，原文件将被覆盖而没有制作备份；

自定备注：

（1）这里说的 “拓展” 指的是 suffix。

（2）不能够通过 -i 将原文件的备份文件存储在不同分区上。

实例：

$ cat test

hello, linux

hello, sed

$ sed -i/home/tom/*.old -e '$a this is a test' test

$ cat /home/tom/test.old

hello, linux

hello, sed

this is a test

-l N

--line-length=N

声明默认的换行长度。长度为 0 表示从不换行。如果没有被指定，其值将被视为 70

自定备注：

未测试成功。

--posix

GNU sed 命令包含了几个针对 POSIX sed 的拓展。为了简化编写可移植性的脚本，这个选项禁用了所有在这个手册中的拓展，包括额外的命令。大多数的这些拓展接受 sed 程序中超出 POSIX 授权的语法，但有一些（例如在 Reporting Bugs 中描述的 N 命令的行为, see "Reporting Bugs"）实际上违背了标准。如果你仅仅想禁用随后的拓展类型，你能够设置 POSIXLY_CORRECT 变量为一个非空的值。

自定备注：

应该将该变量设置为环境变量。

-b

--binary

这个选项在每一个平台上可用，但是仅仅在那些操作系统区分文本文件和二进制文件的地方才有效。这种情况下区分被产生 -- 对于 MS-DOS, Windows, Cygwin -- 文本文件由被回车和换行字符分隔的行组成，sed 不能够看到结尾位置的 CR 字符。当这个选项被指

定，sed 将使用二进制模式打开输入文件，因此不要求对于结束于 LF 字符的行做特定的处理和考虑。

自定备注：

该选项尚未测试

--follow-symlinks

当处于支持符号链接的平台上时这个选项是可用的，当且仅当 -i 选项被声明时它将生效。在这个前提下，如果在命令行上声明的文件是一个符号链接，sed 将沿着链接，去编辑链接的最终目标。默认的行为是中断符号链接，所以链接目标进将不会被编辑。

自定备注：

在不完全测试中，如果不使用 -i 选项，sed 会跟随符号链接而不是中断链接。

-r

--regexp-extended

使用拓展正则表达式(regular expressions)而不是基础正规表达式(basic regular expressions)，拓展正则表达式是那些能够被 egrep 接受的表达式，它们能够更加清晰因为使用了更少的反斜杠，但是它们是一个 GNU 拓展因此使用这些表达式的脚本是不可移植的。（see Extended regexps）

-s

--separate

默认的，sed 将那些在命令行上声明的文件视为一个单独的持续的长流。这个 GNU sed 拓展允许使用者将它们视为分隔的文件：范围内的处理（例如 '/abc/,/def/'）是不被允许跨越几个文件的，行号和每一个文件的开头关联，'$' 指代每一个文件的最后一行，由 R 命令调用的文件被绕回到每一个文件的开头。

-u

--unbuffered

输入和输出缓冲都是保证实用的最小值（as minimally as practical）。（如果输入来自类似 tail -s 的命令，这是特别有用的，并且你希望能够尽可能快的看到转化后的输出。）

自定备注：

未测试成功

如果没有 -e, -f, --expression 或者 --file 选项在命令行被给出，然后第一个在命令行上的非选项参数（non-option argument）将被视为脚本（script）来被执行。

如果任何命令行参数在如上处理后被保留，这些参数将被解释为输入文件的文件名来被处理。一个 - 的文件名指代标准输入流。如果没有文件名被指定那么将被处理标准输入。

脚注:

（1）这里应用于命令例如 =, a, c, i, l, p（This applies to

commands such as '=', 'a', 'c', 'i', 'l', 'p'.）。你仍然能够通过使用 w 或者 W 联合 /dev/stdout 特殊文件来写到标准输出中。

（2）注意 GNU sed 无论任何输出是否被实际的更改了，都制作备份文件（Note that GNU 'sed' creates the backup file whether or not any output is actually changed）。

自定备注：

指没有使用 -i 选项的前提下。

3. sed Programs

****

一个 sed 程序包括一个或者多个 sed 命令，通过一个或者更多 -e, -f, --expression, 和 --file 选项传入，在这些选项没有被使用的情况下，则是第一个非选项参数（non-option argument）被使用。本文挡提及 sed 脚本；这是关于所有被传入的脚本和脚本文件的有序并置的含义的理解（This document will refer to 'the' 'sed' script; this is understood to mean the in-order catenation of all of the SCRIPTs and SCRIPT-FILEs passed in.）。

每一个 sed 命令包括一个可选的定位或者定位范围（addresses or addresses range），被一个 one-character 命令名字和任何可选的 command-speific 代码所跟随。

3.1 How sed works?

==========

sed 维护两个数据缓冲区：一个活动的 pattern space，和一个临时的 hold space。它们初始时全部为空。

sed 通过执行如下针对每一行输入的周期来运作的：

no.1：sed 从输入流中读取一行，移除任何尾随的新行符，然后将它放入 pattern 空间。

no.2：命令将被执行，每一个命令能够拥有一个和它相关联的定位：定位是一种条件代码，一个命令只有在条件被鉴别通过后才能被执行。

当抵达这个脚本的结尾时，除非 -n 选项被使用，否则 pattern 空间中的内容将被打印到输出流，如果尾随新行符被移除了那么它将会被加回来。然后针对下一个输入行的周期开始。

除非特殊的命令（例如 D）被使用，在两个周期之间，pattern 空间的内容将被删除。另一方面，hold 空间在周期之间保持它的数据（h/H/x/g/G 命令在两个周期之间将移除全部的缓冲区）。

自定备注：

对于 h/H/x/g/G 命令，应该是没有移除缓冲区内容的，只不过是对缓冲区内容进行了操作。

----------Footnotes----------

（1）实际上，如果 sed 打印一行但是没有遇到终止的新行符，尽管如此仍然会打印丢失的新行符，只要有更多的文本被发送到相同的输出流中，这将会给出最意想不到的惊喜，即使它没有准确的使用等同于 cat 的命令，例如 sed -n p。

自定备注：

不理解上述的脚注

3.2 Selecting lines with sed

==========

在一个 sed 脚本中的定位操作可以是以下形式中的任何一个：

'NUMBER'

声明一个行号将会仅仅匹配位于输入中相对应的行。（注意 sed 持续并跨越所有的文件来计算行数，除非 -i 或者 -s 选项被声明。）

'FIRST~STEP'

这个 GNU 拓展匹配第 FIRST 行开始的每一个 STEP 倍的行号的行。典型的，当存在一个非负数 N 使得当前行号等于 FIRST + (N * STEP) 的行时，该行将被选择。因此，要选择奇数行号的行，一个方法是使用 '1~2'；要选择第二行开始的每一个三倍的行号的行，'2~3'应该被使用；要选择第五行开始的每一个五倍的行，使用 '10~5'；'50~0' 仅仅 '50' 的另一种表达方式。

'$'

这个选项匹配最后一个输入文件的最后一行，或者每一个文件的最后一行当使用 -i 或者 -s 选项被声明的时候。

'/REGEXP/'

这将会选择任何和正则表达式 REGEXP 匹配的行。如果 REGEXP 自身包含任何 '/' 字符，每一个都必须被反斜杠 '\' 跳脱。

空的正则表达式 '//' 重复上一个（last）正则表达式匹配（如果空的正则表达式传递给 's' 命令则受到相同的限制）。注意当正则表达式被编译的时候正则表达式的修饰符（modifiers）将被计算，因此将它们和空的正则表达式一起声明是不可行的。

自定备注：

对于 // ，这里有一个实例：

$ cat ttt

127.0.0.1 localhost

hello, sed

hello, world!

$ sed -ne '/sed/p' -e '//p' ttt

hello, sed

hello, sed

从上述实例可以得知：// 是在重复前面的 /sed/ 这个正则表达式。

'\%REGEXP%'

（% 可以被任何单个字符替换）

这里也是匹配正则表达式 REGEXP ，但是允许使用一个不同的定界符（delimiter）而不是 '/'。当正则表达式自身包含大量的斜杠字符时这是特别有用的，从此它就避免对每一个 '/' 冗长乏味的跳脱。如果 REGEXP 自身包含任何定界符（delimiter）字符，每一个应该被一个反斜杠 '\' 跳脱。

'/REGEXP/I'

'\%REGEXP%I'

用于正则表达式匹配的 'I' 修饰符（modifier）是一个 GNU 拓展，将会导致 REGEXP 按照大小写不敏感的方式被匹配。

'/REGEXP/M'

'\%REGEXP%M'

对于正则表达式的 'M' 修饰符是一个 GNU sed 拓展，将会导致 '^' 和 '$' 来分别（除了一般的行为）匹配在一个换行符之后的空字符串和之前的空字符串。存在着特殊的字串 '\`' 和 '\'' ，总是匹配 buffer 的开头或者结尾。'M' 表示多行（multi-line）。

自定备注：

不理解。

如果没有给出定位操作，将被匹配所有的行；如果给出了一个定位，仅仅选择和定位相匹配的行。

可以通过声明两个被逗号 ',' 分隔的定位来指定一个定位范围。一个定位范围匹配那些从被第一个定位匹配的行开始，持续到第二个定位

被匹配的行（被包含）。

如果第二个定位是一个正则表达式，然后对于结束匹配的检查将会从和第一个定位匹配的行之后的那一行开始：范围将会总是跨越至少两行（当然排除输入流结束的情形）。

如果第二个定位是一个小于等于匹配第一个定位的行号的数字，将会仅仅匹配一行。

自定备注：

第一个定位也可以是一个正则表达式。

GNU sed 也支持一些特殊的双定位（two-adress）形式；所有的这些都是 GNU 拓展：

'0,/REGEXP/'

行号 0 能够被使用到一个类似 '0,/REGEXP/' 的定位声明中，以至于 sed 也会尝试在输入的第一行中对 REGEXP 进行匹配。换而言之，'0,/REGEXP/' 类似于 '1,/REGEXP/'，但是除了这种情况：如果 ADDR2 匹配输入的最开始的行，'0,/REGEXP/' 将认为它是范围的结束，相反的 '1,/REGEXP/' 形式将会匹配它的范围的开始，因此使得范围跨越到正则表达式第二次发生匹配的位置。

注意这是唯一的一个对于 '0' 定位生效的地方；不存在第 0 行，通过使用任何其他方式被给予 '0' 定位的命令将会给出一个错误。

自定备注：

举例：

$ cat test.txt

welcome to use stream editor(sed)

2you are right

I will become webmaster and linux-master

wow, haha

$ sed -n '0,/w/p' test.txt

welcome to use stream editor(sed)

$ sed -n '1,/w/p' test.txt

welcome to use stream editor(sed)

2you are right

I will become webmaster and linux-master

从上述实例可以得知：

使用第一种方法，将仅仅匹配第一行，而使用第二行，则在正则表达式第二次被匹配的地方停止。

'ADDR1, +N'

匹配 ADDR1 和接下来的 N 行。

'ADDR1, ~N'

匹配 ADDR1 和接下来的行直到下一行的输入行号是 N 的倍数。

自定备注：

和 ADDR1 匹配的行一定会被显示，然后由是否满足第二个条件来确定另一个范围边界。

附加 '!' 字符到一个定位声明的结尾处将会对匹配结果进行取反。那就是，如果 '!' 字符跟随一个定位范围，然后只有不匹配定位范围的行将被选择。这对于单个的定位也有效，同时，也许是违背常情（perversely）的，对于空定位一样有效。

自定备注：

什么是空定位？

推测：指没有使用定位声明的情况，在这种情况下，所有的行都将被匹配。

3.3 Overview of Regular Expression Syntax

==========

为了知道如何使用 sed，人们应该理解正则表达式（regexp 作为缩写）。一个正则表达式是一个依赖于从左到右被匹配的主题字串的 pattern 。大多数的字符是普通的：在一个 pattern 中它们代表它们自身，同时匹配在主题中相一致的字串。作为 一个琐碎的例子，pattern:

The quick brown fox

匹配一个和它自身一致的主题字串的一部分。正则表达式的力量来自于在 pattern 中包含的可替换和重复的能力。这些是通过特殊字符的使用来编码在 pattern 中，它们并不代表自身而是通过一些特殊的方式被解释。这里是对于在 sed 中使用的正则表达式的简要描述。

'CHAR'

一个匹配它自己的单独的普通字符。

'*'

为前面的正则表达式匹配一个 0 个字符长度或者更大长度相匹配的序列（Matches a sequence of zero or more instance of matches for the preceding regular expression），序列必须是这些情况之一：一个普通字符，一个经过 '\' 预处理的特殊字符，一个 '.'，一个群组正则表达式（grouped regexp）（见下面），或者一个括号表达式。作为一个 GNU 拓展，一个有后缀的正则表达式（a postfixed regexp）也应该能够被 '*' 跟随；例如，'a**' 和 'a*'是等价的。POSIX 1002.1-2001 声明当 '*' 出现在一个正则表达式或者子表达式的开头时代表它自身，但是许多的 nonGNU 实现不支持这点，一个可移植的脚本应该在这些上下文中使用 '*' 作为替代（portable scripts should instead use '*' in these contexts）。

自定备注：

推测：有后缀的表达式是指类似于这种的 '\(REGEXP\)'* 的表达式。

'\+'

如同 '*'，但是匹配一个或者更多。它是一个 GNU 拓展。

'?'

如同 '*'，但是只匹配 0个 或者 1个。它是一个 GNU 拓展。

'\{I\}'

如同 '*'，但是准确的匹配 I 次序列（I 是一个十进制整数）；为了可移植性，保持它在 0 到 255 之间。

自定备注：

指重复前面一个正则表达式 I 次。

例如：

$ sed -ne '/^a\{1\}/p' file-name

$ sed -ne '/^a\{2\}/p' file-name

第一个命令表示匹配位于 file-name 文件中开头含有一个 a 字符的行并打印。

第二个命令表示匹配位于 file-name 文件中开头含有两个 a 字符的行并打印。

'\{I,J\}'

匹配在 I 次和 J 次之间的序列（I/J 类似于上一条的 I）。

自定备注：

指重复前面一个正则表达式 I 到 J 范围内任意一个整数次数。

'\{I,\}'

匹配大于等于 I 的序列。

'\(REGEXP\)'

组织内嵌的 REGEXP 作为一个整体，这被用来：

* 应用于后缀操作符（postfix operators），例如 '\(abcd\)*':这将会查 0 个或者更多个 abcd 整体序列，然而 'abcd*' 将会查找被跟随有 0 或者更多 d 的 abc 。注意对于 '\(abcd\)*' 的支持被 POSIX 1993.1-2001 所要求，但是许多 nonGNU 实现并没有支持它，因此它不是普遍可移植的。

* 使用返回参考（use back references）（见下面）。

'.'

匹配任何字符，包括新行符。

'^'

匹配位于 pattern space 开头的空字符串，例如.在 ^（circumflex）之后所显示的正则表达式一定会出现在 pattern space 空间的开头。

在大多数的脚本中，pattern space 被初始化为每一行的内容（参考 How sed works: Execution Cycle）。所以，它是一个有用的简化想到用 '^#include' 来匹配那些以 '#include' 作为开头字符串的行--例如，如果存在位于它之前的空格，那么匹配失败。这个简化是有效的只要 pattern space 的原始内容没有被修改过，例如使用 's' 命令。

自定备注：

's' 命令和这个简化有什么关系？

仅仅当 '^' 位于正则表达式或者子表达式的开头时它作为一个特殊字符（那是指，紧接 '\(' 或者 '\|' 之后）。不过，因为 POSIX 允许实现在那种上下文中将 '^' 看为普通字符，可移植的脚本应该避免 '^' 位于一个子表达式的开头（Portable scripts should avoid '^' at the beginning of a subexpression, though, as POSIX allows implementations that treat '^' as an ordinary characters in that context）。

自定备注：

上面最后一段的最后一句很别扭，可能出错。

'$'

和 '^' 一样，但是代表 pattern space 的结尾。'$' 作为一个特殊字符也是仅仅当它位于正则表达式或者子表达式的开头（那是指，在 '\(' 或者 '\|' 之后），在一个子表达式的结尾处使用是不可移植的。

自定备注：

在我的测试中，

如果使用 sed -ne '/$xxx/p' file-name，则 $ 字符被当作普通字符来匹配；

如果使用 sed -ne '/xxx$/p' file-name，则 $ 字符表示 pattern space 的结尾。

在定位操作中，例如使用 sed -ne '1,$s/xxx/yyy/p' file-name，则 $ 字符表示file-name 的最后一行。

如果使用 sed -ne '/^xxx/p' file-name，则 ^ 字符表示 pattern space 的开头。

如果使用 sed -ne '/xxx^/p' file-name，则 ^ 字符表示普通字符。

如果将 $ 字符或者 ^ 字符放在 xxx 的中间，比如 xx^xx 或者 x$xxx ，则同样都代表普通字符。

等价用法：

我在一些博客上看到：可以使用 \< 和 \> 字串分别表示 pattern space 的开头和结尾；作用和当 ^ 和 $ 用作特殊字符时等同。

另外，可以使用如下方法来在当前行中追加内容：

$ sed -i -e '1s/$/192.168.1.1/' file-name

上述脚本在 file-name 文件的第一行最后一个字符后面添加 ip；另外，不要担心在第一行的后面没有 $ 结尾字符，该字符在添加完毕

后会自动添加上去。

'[LIST]'

'[^LIST]'

匹配任何位于 LIST 中的单个字符：例如，'[aeiou]' 匹配所有的元音字母。一个列表可以包含类似于 'CHAR1-CHAR2' 的序列，这将会匹配任何位于 CHAR1 和CHAR2 之间的字符。

一个领头的 '^' 为与 LIST 相反的含义，所以它匹配任何单个没有位于 LIST 中的字符。为了在列表中包含 ']' ，使得它是第一个字符（如果需要 '^' 的话应该在 '^' 之后），为了在列表中包含 '-'，使得它作为第一个或者最后一个字符。为了包含 '^' 字符，可以将 '^' 放在第一个字符的后面。

字符 '$'，'*'，'.'，'[' 和 '\' 通常在 LIST 中是非特殊的。例如，'[*]' 匹配 '\' 或者 '*' 两者中的一个，因为 '\' 在这里不是特殊字符。然而，类似于 '[.ch.]'，'[=a=]' 和 '[:space:]' 在 LIST 中是特殊的，各自代表收集符号（symbols），等价的类别，和字符类别，因此当 '[' 被 '.'，'='，或者 ':' 所跟随时它在 LIST 是特殊的。同样，当没有处于 'POSIXLY_CORRECT' 模式中时，在 LIST 中的特殊的字符比如 '\n' 和 '\t' 被识别出来。（见 Escapes）。

'REGEXP1\REGEXP2'

匹配 REGEXP1 和 REGEXP2 两者之一。通过插入语来使用复杂的可替换的正则表达式。匹配进程从左到右依次尝试每一个选择，第一个成功的将被使用。它是一个 GNU 拓展。

自定备注：

不理解

'REGEXP1REGEXP2'

匹配 REGEXP1 和 REGEXP2 的连结。连结绑定比 '\|','^' 和 '$' 更加牢固，但是要低于其他正则表达式操作符。

自定备注：

不理解

'DIGIT'

匹配位于正则表达式中的第 DIGIT 个 '\(...\)' 插入子表达式。这被称为 'back reference' 。子表达式通过从左到右计算 '\(' 的发生次数隐含的被标上序号。

自定备注：

例如：

$ sed -ne 's/\(love\)able/\1rs/p' file-name

上述脚本将匹配位于 file-name 文件中的含有 loveable 行，使用 lovers 来替换它，然后将结果打印出来。

注意：

（1）\1 表示前面的脚本中，使用 \(...\) 包含起来的第一个式子，在这里则为 love 。

（2）1 需要使用 \ 来修饰，这和文档中所说的似乎有所不同。

'\n'

匹配新行符。

自定备注：

未匹配成功。

'\CHAR'

匹配 CHAR ，CHAR 是 ‘$’,'*','.','[','\',或者 '^'。注意你可以去轻松假设被解释的类 C 反斜杠只有 '\n' 和 '\\'；特别的是 '\t' 是不可移植的，在大多数的实现下是匹配一个 't' 而不是一个 tab 字符。

自定备注：

在 GNU sed 中 \t 是可以被使用的。

注意正则表达式匹配器是贪婪的（greedy），例如，匹配是从左到右被尝试，如果以相同字符开头的两个或者多个匹配是可能的，他将匹配最长的那一个。

自定备注：

不理解

部分实例：

'.*'

'.\+'

这两个都匹配一个字串中的所有字符；然而，第一个匹配每一个字串（包括空字串），然而第二个仅仅匹配包含至少一个字符的字串。

'.\{9\}A$'

匹配这样的行：该行里面有一个 A ，并且在 A 之前至少有 9 个字符。

自定备注：

这里是没有错的，只要在一行中存在满足上述条件的字串，该行就是被匹配。

'^.\{15\}A'

匹配这样一个行：它开头包含由 16 个字符的字符串，前 15 个任意，最后一个是 A

自定备注：

"/$var/"

匹配和变量 var 内容相同的行。

注意：需要使用 " 而不是 ' 。

3.4 Often-Used Commands

==========

如果你全部使用 sed ，你将会相当可能想要知道这些命令。

'#'

不允许定位。

'#'字符是一个注释的开头；这个注释持续到下一个新行符。

如果你关心可移植性，注意一些 sed 的实现可能支持一个单行注释（a single one-line comment），然后仅仅当脚本的最前面一个字符是 '#'（if you are concerned about portability, be aware that some implementations of sed may only support a single one-line comment, and then only when the very first character of the script is a '#'）。

警告：

如果 sed 脚本的头两个字符是 '#n' ，然后 -n （no-autoprint）选项将被附加上去。如果将一个注释放进你的脚本的第一行，注释是以字符 'n' 开头，同时你不想要这种行为，然后应该确保使用一个大写字符 'N'，或者在 'n' 之前放入至少一个空格。

'q [EXIT-CODE]'

这个命令只接受一个单个的定位。

退出 sed 而不处理任何更多的命令或者输入。注意当前的 pattern space 将被打印如果自动打印没有被 -n 选项禁用的话。这个从 sed 脚本返回一个退出代码的能力是一个 GNU 拓展。

自定备注：

一个实例：

```
$ cat ttt
hello
this is a test
$ sed -ne '1{n; q 3; p}' ttt
$ echo $?
3
$ sed -ne '1{n; p; q 4}' ttt
this is a test
$ echo $?
4
$ sed -ne '1q' ttt
$ sed -e '1q' ttt
hello
```

从上述实例可以看出：

(1)n 会跳过当前的行，即舍弃当前的 pattern space 中的内容，然

后开始下一个周期；

(2)sed 命令在遇到 q 命令时，会直接退出该程序，然后返回退出代码；

(3)如果自动打印没有被 -n 禁用，则直接使用 'Nq' 或者 'Nq EXIT-CODE' 会直接打印 N 行号的行，后者会返回一个退出代码。

'd'

删除 pattern space ；立刻开始下一个周期。

'p'

打印 pattern space (到标准输出)。这个命令通常只和命令行选项 -n 结合。

'n'

如果自动打印没有被禁用，打印 pattern space，然后不管任何情况，使用下一行输入写入 pattern space。如果没有更多的输入，sed 将退出而不执行任何更多的命令。

'{ COMMANDS }'

一个组命令可以被 '{' 和 '}' 字符封装起来。这是尤其有用的当你想要通过一个单独定位（或者定位范围）被匹配让一组命令被触发时。

自定备注：

在 {} 中，可以使用；号来将不同的命令分隔。

3.5 The 's' Command

==========

's' （as in substitute）命令的语法是 's/REGEXP /REPLACEMENT/FLAGS'。'/'字符可以被任何在 's'命令中给出的其他单个字符所替代。'/' 字符（或者其他被用来代替它的字符）能够出现在 REGEXP 或者 REPLACEMENT 中当且仅当它被 '\' 字符预处理。

's' 命令也许是 sed 命令中最重要的，同时也有着许多的不同的选项。它最基础的概念是简单的：'s' 命令依赖所提供的 REGEXP 来尝试匹配 pattern space ；如果匹配成功，pattern space 中被匹配的部分将被 REPLACEMENT 替换。

REPLACEMENT 能够包含 '\N'（N 是一个从 1 到 9 的数字），它引用匹配内容中位于在第 N 个 '\（' 和对应的 '\)' 之间的部分（The REPLACEMENT can contain '\N' (N being a number from 1 to 9, inclusive) references, which refer to the portion of the match which is contained between the Nth '\(' and its matching '\)'）。另外，REPLACEMENT 能够包含未被跳脱的 '&' 字符，这是指 pattern space 中整个被匹配的部分。最后，作为一个 GNU sed 拓展，你能够引用一个由反斜杠和这些字符（'L', 'l', 'U', 'u' 或者 'E'）所组成的特殊序列。含义如下：

'\L'

将 REPLACEMENT 部分转化为小写直到一个 '\U' 或者 '\E' 被发现。

'\l'

将下一个字符转化小写。

'\U'

将 REPLACEMENT 部分转化为大写直到一个 '\L' 或者 '\E' 选项被发现。

'\u'

将下一个字符转化大写。

'\E'

停止由 \L' 或者 '\U' 开始的大小写转化。

为了包含一个在最后的替代部分中的文字：'\','&'，或者一个新行符，确保在 REPLACEMENT 中通过使用一个 '\' 来预处理所要求的 '\','&'，或者一个新行符。

自定备注：

（1）使用 & 可以表示在 pattern space 中被匹配的部分内容，通常情况下，就是前面的 REPLACEMENT 部分。

例如：

$ sed -ne 's/love/**&**/p'

上述脚本将会把所匹配到的 love 字段提换为 **love**，这时 & 就表示被匹配的字段 love。

如果是这种情况：sed -ne 's/\(love\)rable/**&**/p' file-name

\(和 \) 字段会被去掉，也就是说 & 表示 loverable 字段而不是 \(love\)rable

（2）

对于上述 \L \l \U \u \E 特殊序列的几个实例：

$ cat ttt

hello, sed

$ sed -ne 's/hello/\Uhe\LLLo/p' ttt

HEllo, sed

$ sed -ne 's/sed/\lS\uUn\LYO\E\ng/p' ttt

hello, sunyong

$ var=sunyong

$ sed -ne "s/sed/\U$var/p" ttt

hello, SUNYONG

注意：对于最后一个脚本，灵活性是很高的。

（3）

使用其他字符替换位于 s 命令用来分隔的 / 字符，这在搜索含有路径的关键词非常有用：

's' 命令能够被 0 个或者更多的如下的 FLAGS 所跟随：

'g'

将替换应用到所有和 REGEXP 匹配的地方，不仅仅是第一个。

'NUMBER'

仅仅替换 REGEXP 中的第 NUMBER 个匹配内容。

注意：POSIX 标准并没有声明应该发生什么当混淆 'g' 和 NUMBER 修饰符的时候，当前没有一个统一的含义跨越 sed 实现之上。对于 GNU sed ，这个相互作用被定义为：忽略在第 NUMBER 个之前的匹配，然后匹配和取代在第 NUMBER 个以及之后的所有匹配。

'p'

如果替换发生了，就打印新的 pattern space 。

注意：当 'p' 和 'e' flag 都被声明的时候，这两个之间的相对顺序产生非常不同的结果。一般情况下，'ep'（evaluate then print）是你所想要的，但是反过来操作对于 debugging 可以非常有用的。由于这个原因，当前的 GNU sed 版本特意都解释了在 ‘e' 之前和之后的 'p' 选项，即为在计算之前或者之后打印 pattern space ，然而一般情况下对于 's' 命令的 flags 只展示一次它们的影响。这个行为尽管被记载了，可能会在将来的版本中改变。

'w FILE-NAME'

如果替换产生了，然后将结果写入指定的文件中。作为一个 GNU 拓展，支持两个特殊的文件名：/dev/stderr，将结果写入标准错误中，/dev/stdout，将结果写入标准输出中（1）。

'e'

这个命令允许来自 shell 命令的信息从管道输入进入 pattern space（This command allows one to pipe input from a shell command into pattern space）。如果替换发生了，在 pattern space 中找到的命令将被执行，同时 pattern space 的内容将被该命令的输出所替换。尾随的新行符被禁止；如果将被执行的命令中包含一个 NUL 字符，那么结果是未定义的。这是一个 GNU 拓展。

自定备注：

举例：

$ cat ttt

haha

this is a test

kkkk

ddd

ls

$ sed -n 's/ls/pwd/e' ttt

haha

this is a test

kkkk

ddd

/tmp/test

从上述结果可以得知：

1>REGEXP 部分要在文件中存在（也就是要被匹配）；

2>被执行的命令放在命令行的后面；

3>e flag 可以使得前面的那一个命令被 sed 读入 pattern space 中，并且取代和 REGEXP 匹配的部分；

4>sed 将执行在 pattern space 中的命令；

5>所得到的结果将代替和 REGEXP 匹配的部分，在允许的情况下，在一个周期的结束将被打印到标准输出中。

'I'

'i'

对于正则表达式匹配的 'I’修饰符是一个 GNU 拓展，它使得 sed 以大小写不敏感的方式来匹配。

'M'

'm'

对于正则表达式匹配的 'M' 修饰符是一个 GNU sed 拓展，将会导致 '^' 和 '$' 来分别（除了一般的行为）匹配在一个换行符之后的空字符串和之前的空字符串。存在着特殊的字串 '\`' 和 '\'' ，总是匹配 buffer 的开头或者结尾。'M' 表示多行（multi-line）。

自定备注：

不理解。

----------Footnotes----------

(1)它是和 'p' 等价的除非 '-i' 选项被使用。

自定备注：

使用 p 和使用 w 在这些情况是等价的：

$ cat ttt

localhost

hello, sed

this is a test

$ sed -e 's/sed/sunyong/p' ttt

localhost

hello, sunyong

hello, sunyong

this is a test

$ sed -e 's/sed/sunyong/w /dev/stdout' ttt

localhost

hello, sunyong

hello, sunyong

this is a test

而对于使用了 -i 选项，w 可以指定写入的文件，但是除非 -i 选项后面添加了后缀，否则 p 则只能写入原文件中。

使用 s 命令对将所匹配的内容删除：

$ sed -ne 's/ip//g' file-name

能够将所有的 ip 字符删除，实现了对一行中的部分内容的删除，而不是整行删除。

3.6 Less Frequently-Used Commands

==========

尽管也许这里的命令比之前的小节中的命令更少使用，但一些小而有用的 sed 脚本能够通过这些命令来构建。

'y/SOURCE-CHARS/DEST-CHARS/'

（在任何给出 y 命令中的 / 字符能够被任何其他字符统一的替换掉。）

通过与 SOURCE-CHARS 字符相关联的 DEST-CHARS 字符来直译任何位于 pattern space 中的和 SOURCE-CHARS 相匹配的字符。

/ , \ 或者新行符的实例能够出现在 SOURCE-CHARS 或者 DEST-CAHRS 字符列表中，规定每一个实例要被一个 \ 字符跳脱。SOURCE-CHARS 和 DEST-CHARS 字符列表必须相同的字符数目（在去除跳脱之后）。

自定备注：

$ cat ttt

a

bc

cddd

abcd

$ sed -e '1,$y/abcd/ABCD/' ttt

A

BC

CDDD

ABCD

上述脚本对 ttt 文件的所有行进行匹配，如果遇到 abcd 字符列表中的任何一个，都将其映射为与之对应的 ABCD 字符列表中的那一个字符。

'a\'

'TEXT'

作为一个 GNU 拓展，这个命令接受两个定位。

当处于当前周期的结尾或者当下一输入行被读入时，对跟随这个命令后的文本按行排序输出（每一行以 \ 字符结尾，\ 将被从输出中移除）。

作为一个 GNU 拓展，如果位于 a 和新行符之间存在一个非“空白符-\”序列，那么这一行文本（以 a 命令之后的第一个非空白字符开始的），被视为 TEXT 块的第一行。（这允许对添加一行编辑动作的简化。）这个拓展同样能和 i 和 c 命令一起工作。

'i\'

'TEXT'

作为一个 GNU 拓展，这个命令接受两个定位。

立刻输出跟随在这个命令之后的文本行（每一行以一个 \ 字符结尾，\ 将被从输出中移除）。

'c\'

'TEXT'

删除和定位或者定位范围匹配的行，然后在上一行的位置（如果没有声明定位，就是在每一行的位置）输出位于这个命令之后的文本行（每一行以一个 \ 字符结尾，\ 将被从输出中移除。）。在这个命令完成之后，一个新的周期将开始，因为 pattern space 将已经被删除。

'='

作为一个 GNU 拓展，这个命令接受两个定位。

打印当前输出行的行号和一个尾随的新行符。

'l N'

打印 pattern space 使用一个清楚的形式：

非打印字符按照 C 风格的跳脱形式打印；

长的行被分割，使用尾随的 \ 字符来表明分割动作；

每一行的结尾使用 $ 字符来标记。

N 声明了自动换行的长度；长度为 0 意味着不对长的行自动换行。如果被忽略，默认使用在命令行上指定的值。N 参数是一个 GNU 拓展。

'r FILENAME'

作为一个 GNU 拓展，这个命令接受两个定位。

按序读取 FILENAME 的内容并当处于当前周期的结尾或者下一行被读入的时候插入输出流中。注意如果 FILENAME 不可读，则将它视为一个空文件，不显示任何错误。

作为一个 GNU 拓展，特殊值 /dev/stdin 被支持作为 FILENAME ，这将会从标准输入中读入内容。

'w FILENAME'

将 pattern space 内容写入 FILENAME 。作为一个 GNU sed 拓展，两个特殊值被支持：/dev/stderr, /dev/stdout 。将结果分别写入标准错误流或者标准输出流中。（1）

在第一个输入行被读入之前文件将被创建或者被截断；所有引用相同文件名的 w 命令（包括成功执行的 s 命令中的 w flag）输出内容但不关闭和重新打开文件（all 'w' commands which refer to the same FILENAME are output without closing and reopening the file）。

自定备注：

在写入文件时，如果文件一开始就存在，那么该文件就会被截断；如果文件不存在，则会被创建；在一次多个脚本同时对一个文件执行写

入操作时，使用追加的方式写入的。就如上面所说的！

'D'

删除 pattern space 中的文本直到遇到第一个换行符。如果有任何剩余的文本被留下，使用合并的 pattern space (resultant pattern space) 重新开始周期（没有从输入中读入新的一行），否则开始一个正常的周期。

'N'

向 pattern space 中添加一个新行符，然后将下一个输入行附加到 pattern space 中。如果没有更多的输入，sed 将会退出而不处理任何更多的命令。

'P'

打印 pattern space 中的部分内容直到遇到第一个新行符。

'h'

拷贝当前 pattern space 的内容到 hold space 中。

'H'

在 hold space 的内容中追加一个新行符，然后将当前 pattern space 中的内容追加到 hold space 中。

'g'

拷贝当前 hold space 的内容到 pattern space 中。

'G'

在 pattern space 的内容中追加一个新行符，然后将当前 hold space 中的内容追加到 pattern space 中。

'x'

交换 hold space 和 pattern space 中的内容。

----------Footnotes----------

（1）和 'p' 等价除非 -i 选项正在被使用。

自定备注：

（1）注意：在本节中介绍的均为 command ，也就是说应该使用 adress 匹配相应的行，然后在触发这些 command 。

（2）

一些实例：

'a\TEXT'

'a \TEXT'

'a TEXT'

将 TEXT 字串添加作为所有所匹配的行的下面一行。

'c\TEXT'

'c \TEXT'

'c TEXT'

将 TEXT 字串取代所有所匹配的行。

'i\TEXT'

'i \TEXT'

'i TEXT'

将 TEXT 字串插入到所有所匹配的行的上面。

h\H\g\G 用法举例：

$ cat ttt

this is a test

/tmp/test

welcome to use sed

you are right

$ cat ttt | sed -e '1h' -e '$g'

this is a test

/tmp/test

welcome to use sed

this is a test

上面将第 1 行复制到 hold space 中，然后当处理结尾的行时，将 hold space 中的内容复制到 pattern space 中，覆盖掉了其中原来的属于结尾一行的内容。

$ sed -e '1H' -e '2H' -e '$G' ttt

this is a test

/tmp/test

welcome to use sed

you are right

this is a test

/tmp/test

上面先将第 1 行复制到 hold space 中，然后将第 2 行追加复制到 hold space 中，然后将 holp space 中的内容追加到最后一行的后面。

注意：在 you are right 和 this is a test 这两行之间存在一行空行，尚不知道原因！推测可能是和 pattern space 一开始就被清空有关。

延伸正规表示法：

+ ：表示重复一个或者一个以上的前一个字符；

? ：表示零个或者一个的前一个字符；

| ：表达逻辑意义或，即如果两个条件中有一个被满足，则整个条件被满足；

() : 表示一个群组的字符串，例如 g(la|oo)d 表示 glad 和 good 这两个字串；

()+: () 和 + 的组合，将 () 组合的字串看成一个“字符”，+ 的意义则和原来的相同；

注意：延伸正规表示法的关键字对比于基础正规表示法的关键字是经过拓展了的。但是基础正规表示法则不支持所拓展的关键字；

例如，在基础正规表示法中使用括号的时候一般都要使用跳脱字符来对括号进行跳脱。

5. GNU 'sed''s Limitations and non-Limitations

****

对于那些想要写出可移植性的 sed 脚本的人，要注意一些实现已经被了解到限制行的长度（对于 pattern 和 hold 空间）不超过 4000 字节。POSIX 标准声明了合格的 sed 实现应该支持至少 8192 个字节的行长度。GNU sed 对于行长度没有内部限制；只要它能够 'malloc' 更多的（虚拟的）内存，你能够随心所欲填充（feed）或者构建行。

然而，递归被用来处理子模式（subpatterns）和无限的重复。这意

味着可用的栈空间可能通过一些模式来限制能够被处理的缓冲区大小（This means that the available stack space may limit the size of the buffer that can be processed by certain patterns）。

6.Other Resources for Learning About 'sed'

****

另外有几本书籍已经写到了 sed（在书中讨论 shell 编程时特别的或者作为一个章节写到），一个能够找出更多的关于 sed 信息（包括几本书的建议）的地方是 sed-users 邮件列表的 FAQ ，这里可用：

http://sed.sourceforge.net/sedfaq.html

有兴趣的可以访问：

http://www.student.northpark.edu/pemente/sed/index.htm 和 http://sed.sf.net/grabbag ，包含一些 sed 指导和一些其他的 sed 相关的糖果。

sed-users 邮件列表自身被 Sven Guckes 维护。要订阅的话，访问 http://groups.yahoo.com 并搜索 sed-users 邮件列表。