使用 GNU 文本实用程序

http://www-900.ibm.com/developerWor...nux/index.shtml

关于本教程

本教程展示如何使用 GNU 文本实用程序集合来处理日志文件、文档、结构化文本数据库，以及其他文本性的数据或内容源。本集合中的实用程序经过 UNIX/Linux 开发人员几十年的改进，已证明是有用的，并且应该是您用于一般文本处理任务的第一选择。

本教程是为 Linux/UNIX 程序员和系统管理员编写的，属于初级至中级水平。

学习本教程的前提条件

对于本教程，您应该一般地熟悉一些类 UNIX 环境，特别是命令行 shell。您本身不需要是一个程序员：事实上，本教程所讲述的技术将对系统管理员和需要处理特殊报告、日志文件、项目文档以及类似内容的用户最有用（因此对正式的编程代码 处理不是那么有用）。在学习本教程的整个过程中，最好随时打开一个 shell，并试验本教程所展示的例子以及它们的一些变化形式。

基本概念将在简介：UNIX 哲学中回顾，您可以在其中复习管道、流、grep 和脚本编程的基础知识。

关于作者

David Mertz 对于处理文本具有持久的爱好。 他甚至专门为此编写了一本书 Text Processing in Python ，并且经常在他为 IBM developerWorks 撰写的文章和专栏中谈及相关的主题。

关于本教程内容的技术问题和意见，请联系 David Mertz 或者单击任一屏顶部的"反馈"。David 的 Web 站点 也是相关信息的一个很好来源。
关于本教程内容的技术问题和意见，请联系 David Mertz 或者单击任一屏顶部的"反馈"。David 的 Web 站点 也是相关信息的一个很好来源。

简介：UNIX 哲学

组合使用小型实用程序来完成大型任务

在诸如 Linux、FreeBSD、Mac OS X、Solaris、AIX 等受 UNIX 启发的操作系统中，开发环境甚至 shell 和工作环境的背后都存在一种共同的哲学。 这种哲学的要旨就是使用小型实用程序来 完满地（没有其他负面影响）完成每个小型任务，然后组合使用这些实用程序来执行复合任务。GNU 项目所产生的大多数产品都支持这种哲学——实际上特定的 GNU 实现已经移植到许多平台上，有些平台甚至传统上未被看作是 UNIX 类的。然而，Linux 内核必定是更有点单一性的软件——虽然如此，但是其内核模块、文件系统、视频驱动程序等都是相当组件化的。

对于本教程，您应该一般地熟悉一些类 UNIX 环境，特别是命令行 shell。 您本身不需要是一个程序员：事实上，本教程所讲述的技术将对系统管理员和需要处理特殊报告、日志文件、项目文档以及类似内容的用户最有用（因此对正式的编 程代码处理不是那么有用）。在学习本教程的整个过程中，最好随时打开一个 shell，并试验本教程所展示的例子以及它们的一些变化形式。

文件和流

如果这种 UNIX 哲学 具有倡导最低限度的模块化组件和协作的道义论的一面的话，它还具有本体论的一面："一切皆文件"。抽象地说，文件 只是支持一些操作的对象：首先是读取和写入字节，但是也有诸如指出其当前位置和弄清何时到达文件结尾这样的操作。UNIX 权限模型也是围绕文件的概念来建立的。

具体地说，文件可以是可记录介质上的一个具体区域（并具有由文件系统提供的关于其名称、大小、在磁盘上的位置等的标记）。但是一个文件也可以是 /dev/ 层次结构中的一个虚拟设备，或者通过 TCP/IP 或通过诸如 NFS 这样的高级协议传来远程流。重要的是，特殊文件 STDIN、STDOUT 和 STDERR 可用于读取或写到用户控制台，以及用于在实用程序之间传递数据。这些特殊文件可通过虚拟文件名称来表示，并具有特殊的语法：

STDIN 是/dev/stdin 和/或 /dev/fd/0
STDOUT 是 /dev/stdout 和/或 /dev/fd/1
STDERR 是 /dev/stderr 和/或 /dev/fd/2

UNIX 的文件本体论的优点在于，这里讨论的大多数实用程序都将统一而中立地处理各种数据源，而不管实际位于字节传输之下的存储或传输机制是什么。

重定向和管道

UNIX/Linux 实用程序的通常组合方式是使用管道和重定向。许多实用程序或者自动地或者可选地从 STDIN 接受输入，并将它们的输出发送到 STDOUT（特殊的消息则发送到 STDERR）。管道将一个实用程序的 STDOUT 发送到另一个实用程序的 STDIN（或者发送到对同一个实用程序的新的调用）。重定向或者将一个文件的内容作为 STDIN 来读入，或者将 STDOUT 和/或 STDERR 输出发送到一个指定的文件。重定向通常用于保存数据以供以后处理或重复处理（对于后者，实用程序将使用 STDIN 重定向）。

在几乎所有的 shell 中，管道都使用竖线 | 符号来执行，而重定向都使用大于号和小于号

来执行：> 和 <。为了重定向 STDERR，可使用 2>，或使用 &> 来同时将 STDOUT 和 STDERR 重定向到同一个地方。您还可以使用双大于号（>>）来将输出附加到一个现有文件的末尾。例如：

源码:

```
$ foo fname | bar - > myout 2> myerr
```

这里，实用程序 foo 可能处理名为 fname 的文件，并输出到 STDOUT。实用程序 bar 使用了一种普遍用法：当输出取自 STDIN 而不是取自指定的文件时指定一个短划线（其他某些实用程序仅接受 STDIN）。来自 bar 的 STDOUT 保存在 myout 中，它的 STDERR 则保存在 myerr 中。

文本实用程序是什么？

GNU 文本实用程序是一些用于处理和操作文本文件和流的工具集合，它们随着类 UNIX 操作系统的演进而被提炼，结果证明它们是最有用的。其中的大多数都是早期 UNIX 实现的组成部分，虽然许多已随着时间的推移而增添了附加的选项。
档案文件 textutils-2.1 中收集的实用程序包括 27 个工具；然而，GNU 项目维护者最近已决定将这些工具打包为更大的集合 coreutils-5.0 的一部分（预计可能会在后面的版本中这样做）。在从 BSD 而不是从 GNU 工具演变而来的系统上，相同的实用程序的打包方式可能稍有不同，但是仍然提供了大多数相同的实用程序。

本教程重点介绍传统上包括在 textutils 中的 27 个实用程序，偶尔会提及和使用一般在类 UNIX 系统上可用的相关工具。然而，我将跳过对实用程序 ptx（permuted index，置换索引）的介绍，因为它的用途太窄，并且包括在这里也太难于理解。

grep（通用正则表达式处理器）

这个工具本身并不是 textutils 的组成部分，但是仍然值得特别提及。实用程序 grep 是使用最广泛的 UNIX 实用程序之一，经常用于文本实用程序的管道输入或输出。

grep 所做的工作从一种意义上说非常简单，从另一种意义上说又相当复杂，难于理解。基本上，grep 识别文件中与某个正则表达式相匹配的行。它有一些开关允许您以各种方式修改输出， 比如打印周围的上下文行，给匹配行编号，或者仅识别其中出现匹配项的文件而不是识别单独的行。本质上，grep 只是用于文件中的行的过滤器（但是功能非常强大）。

grep 的复杂部分是正则表达式，您可以指定它来描述需要的匹配条件。不过这将在另一个教程（请参阅参考资料中的 developerWorks 教程"Using regular expressions"）中讲述。还有其他许多实用程序支持正则表达式，但是 grep 是其中最通用的一个工具，因而比起使用其他工具所提供的较弱的过滤器，将 grep 放进管道中通常更容易。一个简单的 grep 例子如下：

源码:

```
$ grep -c [Ss]ystem$ * 2> /dev/null | grep :[^0]$
INSTALL:1
aclocal.m4:2
config.log:1
```

```
configure:1
```

这个例子列出包含以"system"结尾的行（或许首字母是大写的）的文件；同时显示这些实例出现的次数（也就是如果不为 0 的话）。（实际上，这个例子不会处理大于 9 的计数）。

shell 脚本

虽然文本实用程序旨在以各种有用的格式产生输出（通常通过命令行开关来修改），但是有些时候能够显式地跳转和循环是很有用的。诸如 bash 这样的 shell 允许您通过控制流来组合实用程序，从而执行更复杂的任务。shell 脚本对于封装需要多次执行的复合任务特别有用，特别是那些涉及任务参数化的任务。
解释 bash 脚本编程显然超出了本教程的范围。请参阅参考资料以了解 developerWorks 的"Bash by example"系列中对 bash 的介绍。一旦理解了这些实用程序，将它们组合到已保存的 shell 脚本中是相当简单的。只是出于演示目的，下面提供一个使用 bash 进行流控制的简单例子：

源码:

```
[~/bacchus/articles/scratch/tu]$ cat flow
#!/bin/bash
for fname in `ls $1`; do
  if (grep $2 $fname > /dev/null); then
    echo "Creating: $fname.new" ;
    tr "abc" "ABC" < $fname > $fname.new
  fi
done
[~/bacchus/articles/scratch/tu]$ ./flow '*' bash
Creating: flow.new
Creating: test1.new
[~/bacchus/articles/scratch/tu]$ cat flow.new
#!/Bin/BAsh
for fnAme in `ls $1`; do
  if (grep $2 $fnAme > /dev/null); then
    eCho "CreAting: $fnAme.new" ;
    tr "ABC" "ABC" < $fnAme > $fnAme.new
  fi
done
```

这个脚本将那些由通配符指定的文件中的 a、b 和 c 字母改为大写，然后用新的名称保存更改后的版本。

面向流的过滤

cat 和 tac

最简单的文本实用程序只是将文件或流的正确内容（或许是那些内容的一部分，或者只是那些内容的重新组织）输出到 STDOUT。

实用程序 cat（conCATenate）从第一行开始，到最后一行结束。实用程序 tac （"cat"的逆向操作）以相反的顺序输出行。两个实用程序都读取作为参数来指定的每个文件，但是如果没有指定文件的话，则默认读取 STDIN。与许多实用程序一样，您可以使用特殊名称 - 来显式地指定 STDIN。下面是一些例子：

源码:

```
$ cat test2
Alice
Bob
Carol
$ tac < test3
Zeke
Yolanda
Xavier
$ cat test2 test3
Alice
Bob
Carol
Xavier
Yolanda
Zeke
$ cat test2 | tac - test3
Carol
Bob
Alice
Zeke
Yolanda
Xavier
```

head 和 tail

实用程序 head 和 tail 分别仅输出文件或流的最初部分或最后部分。这两个实用程序的 GNU 版本都支持使用开关 -c 来输出许多字节；两个实用程序都最常以面向行的模式使用，这种模式输出许多行（不管实际的行的长度如何）。head 和 tail 默认都输出 10 行。与 cat 或 tac 一样，如果没有指定文件，head 和 tail 默认都读取 STDIN 的内容。下面是一些例子：

源码:

```
$ head -c 8 test2 && echo # push prompt to new line
Alice
Bo
$ /usr/local/bin/head -2 test2
Alice
```

```
Bob
$ cat test3 | tail -n 2
Yolanda
Zeke
$ tail -r -n 2 test3 # reverse
Zeke
Yolanda
```

顺便说一下，这些（以及其他许多）实用程序的 GNU 版本都具有比 BSD 版本更灵活的开关。

tail 实用程序还具有一种特殊模式（使用开关 -f 和 -F 来指定），它继续显示写到一个"followed（后续）"文件结尾的新行。大写字母开关用于监视文件的截取和重命名，小写字母开关用于监视简单的附加操作。Follow（后续）模式对于观察另一个进程可能对日志文件执行的定期更改特别有用。

od 和 hexdump

实用程序 od 和 hexdump 分别输出文件或流的八进制、十六进制或其他编码的字节。它们对于访问或可视地检查文件中不能直接显示在终端上的字符很有用。例如，cat 或 tail 不会直接区别制表符、空格或其他空白字符——您可以使用 hexdump 来检查究竟使用了哪些字符。根据系统的类型，这两个实用程序中的任一个或者两者都可用——BSD 系统认为 od 比 hexdump 更重要，而 GNU 则相反。然而，两个实用程序都具有完全相同的用途，只不过开关稍有不同。

源码:

```
$ od test3 # default output format
0000000 054141 073151 062562 005131 067554 060556 062141 005132
0000020 062553 062412
0000024
$ od -w8 -x test3 # 8 hex digits per line
0000000 5861 7669 6572 0a59
0000010 6f6c 616e 6461 0a5a
0000020 656b 650a
0000024
$ od -c test3 # 5 escaped ASCII chars per line
0000000   X   a   v   i   e   r  \n   Y   o   l   a   n   d   a  \n   Z
0000020   e   k   e  \n
0000024
```

与其他实用程序一样，od 和 hexdump 都可以从 STDIN 或从一个或多个指定的文件接受输入。此外，od 开关 -j 和 -N 分别允许跳过最初的字节和限制读取的字节数。您甚至可以使用类似 fprintf() 的格式修饰符来进一步定制这些标准开关的输出格式。

HERE 文档

有一种特殊的重定向值得在本教程中提一下。虽然严格地讲，HERE 文档是诸如 bash 这样的 shell 的特性，而不是与文本实用程序有关的任何东西，但是它们提供了向文本实用程序发送特殊数据的有用途径。

双小于号重定向可用于从终端接收伪文件的内容。HERE 文档必须紧跟在 << 后面指定一个终止分隔符。例如：

源码:

```
$ od -c <<END
> Alice
> Bob
> END
0000000   A   l   i   c   e  \n   B   o   b  \n
0000012
```

任何字符串都可以用作分隔符；输入将在一行上遇到该字符串本身的地方终止。这样为我们提供了创建持久性文件的快捷方法：

源码:

```
$ cat  myfile <<EOF
 > Dave
 > Edna
 > EOF
 $ hexdump -C myfile
 00000000  44 61 76 65 0a 45 64 6e  61 0a           |Dave.Edna.|
 0000000a
```

面向行的过滤

作为记录的行

许多 Linux 实用程序将文件视为面向行的记录或数据的集合。这样提供了聚合那些同时对人可读和便于使用工具来处理的数据集合的便利途径。一个简单的技巧就是将每个新行字符看作是记录之间的分隔符，其中每个记录具有类似的格式。

从实用的角度出发，面向行的记录通常应该具有相对有限的长度，或许不应超过几百个字符。虽然没有哪一个文本实用程序内置了这样的限制，但是即使使用自动换 行或水平滚动，人类的眼睛也不适应观看过长的行。对于这样的情况，或者可以使用更复杂的结构化数据格式，或者可以将记录分隔到多行（或许还会作一些 grep 能够识别的标记）。作为一个简单的例子，您可以使用前缀字符来保持分层结构的多行数据格式：

源码:

```
$ cat multiline
A Alice Aaronson
B System Administrator
C 99 9th Street
A Bob Babuk
B Programmer
C 7 77th Avenue
$ grep '^A ' multiline # names only
A Alice Aaronson
A Bob Babuk
$ grep '^C ' multiline # address only
C 99 9th Street
C 7 77th Avenue
```

来自这些 grep 过滤器之一的输出是一个适用的新行字符分隔的部分记录集合，这些集合带有您感兴趣的那些字段。

cut

实用程序 cut 将文件中的字段写到标准输出上，其中每行被看作是分隔的字段集合。默认的分隔字符是制表符，但是这可以使用简短形式的选项 -d <DELIM> 或者完整形式的选项--delimiter=<DELIM>来改变。

您可以使用 -f 开关来选择一个或多个字段。-c 开关则从每行中选择特定的字符位置。两个开关都接受逗号分隔的数字或范围作为参数（包括非封闭范围）。例如，下面的文件 employees 是制表符分隔的：

源码:

```
$ cat employees
Alice Aaronson  System Administrator    99 9th Street
Bob Babuk       Programmer      7 77th Avenue
Carol Cavo      Manager 111 West 1st Blvd.
```

```
$ hexdump -n 50 -c employees
0000000   A   l   i   c   e       A   a   r   o   n   s   o   n  \t   S
0000010   y   s   t   e   m       A   d   m   i   n   i   s   t   r   a
0000020   t   o   r  \t   9   9       9   t   h       S   t   r   e   e
0000030   t  \n
0000032
$ cut -f 1,3 employees
Alice Aaronson  99 9th Street
Bob Babuk       7 77th Avenue
Carol Cavo      111 West 1st Blvd.
$ cut -c 1-3,20,25- employees
Alieministrator 99 9th Street
Bobr7th Avenue
Carlest 1st Blvd.
```

后面的例子将使用制表符之外的自定义分隔符。

expand 和 unexpand

实用程序 expand 和 unexpand 分别将制表符转换为空格和把空格转换为制表符。制表符被认为等价于特定数量的列，默认是八个列，因此对应于一个制表符的明确空格数目取决于那些空格或制表符出现在何处。除非指定 -a 选项，否则 unexpand 仅把初始的空格转换为制表符（这种默认设置对于重新编排源代码很有用）。

继续前面 employees 文件的例子，我们可以执行一些替换。注意在运行 unexpand 之后，输出中的制表符后面可能跟着一些空格，以便产生所需要的整体对齐。

源码:

```
$ cat -T employees  # show tabs explicitly
Alice Aaronson^ISystem Administrator^I99 9th Street
Bob Babuk^IProgrammer^I7 77th Avenue
Carol Cavo^IManager^I111 West 1st Blvd.
$ expand -25 employees
Alice Aaronson            System Administrator      99 9th Street
Bob Babuk                 Programmer                7 77th Avenue
Carol Cavo                Manager                   111 West 1st Blvd.
$ expand -25 employees | unexpand -a | hexdump -n 50 -c
0000000   A   l   i   c   e       A   a   r   o   n   s   o   n  \t  \t
0000010       S   y   s   t   e   m       A   d   m   i   n   i   s   t
0000020   r   a   t   o   r  \t           9   9       9   t   h       S
0000030   t   r
0000032
```

fold

fold 实用程序简单地迫使文件中的行换行。默认情况下，换行是从第 80 列之后开始，不过您可以指定其他宽度。fold 只具有有限种类的自动换行格式，但是它不会完全重新换行段落。 选项 -s 对于至少迫使在空白处换行是很有用的。下面使用我最近的一篇文章作为资料来源（使用我们前面看到的工具来剪取一个例子）：

源码:

```
$ tail -4 rexx.txt | cut -c 3-
David Mertz' fondness for IBM dates back embarrassingly many decades.
David may be reached at mertz@gnosis.cx; his life pored over at
http://gnosis.cx/publish/. And buy his book: _Text Processing in
Python_ (http://gnosis.cx/TPiP/).
$ tail -4 rexx.txt | cut -c 3- | fold -w 50
David Mertz' fondness for IBM dates back embarrass
ingly many decades.
David may be reached at mertz@gnosis.cx; his life
pored over at
http://gnosis.cx/publish/. And buy his book: _Text
 Processing in
Python_ (http://gnosis.cx/TPiP/).
$ tail -4 rexx.txt | cut -c 3- | fold -w 50 -s
David Mertz' fondness for IBM dates back
embarrassingly many decades.
David may be reached at mertz@gnosis.cx; his life
pored over at
http://gnosis.cx/publish/. And buy his book:
_Text Processing in
Python_ (http://gnosis.cx/TPiP/).
```

fmt

对于大多数用途来说，fmt 是比 fold 更有用的换行工具。实用程序 fmt 不仅换行，而且同时保留初始的缩进，并聚合行以实现段落对齐（视需要而定）。在传输或最终存储之前，fmt 对于格式化诸如电子邮件消息之类的文档很有用。

源码:

```
$ tail -4 rexx.txt  | fmt -40 -w50 # goal 40, max 50
  David Mertz' fondness for IBM dates back
  embarrassingly many decades.  David may be
  reached at mertz@gnosis.cx; his life pored
  over at http://gnosis.cx/publish/. And
  buy his book: _Text Processing in Python_
  (http://gnosis.cx/TPiP/).
$ tail -4 rexx.txt  | fold -40
  David Mertz' fondness for IBM dates ba
ck embarrassingly many decades.
  David may be reached at mertz@gnosis.c
x; his life pored over at
  http://gnosis.cx/publish/. And buy his
 book: _Text Processing in
  Python_ (http://gnosis.cx/TPiP/).
```

GNU 版本的 fmt 提供了几种用于首行和后续行缩进的选项。其中一个有用的选项是 -u，它规格化字间距和行间距（多余的空白将被删除）。

nl （和 cat）

实用程序 nl 对文件中的行编号，并具有决定编号如何出现的各种选项。cat 包含您需要用于大多数目的的行编号选项－－在 cat 满足您需要的时候请选择这个更通用的工具。只有在诸如控制前导零的显示这样的特殊情况下，才需要使用 nl（由于历史的原因，cat 并不总是包括行编号）。

源码:

```
$ nl -w4 -nrz -ba rexx.txt | head -6  # width 4, zero padded
0001    LINUX ZONE FEATURE: Regina and NetRexx
0002    Scripting with Free Software Rexx implementations
0003
0004    David Mertz, Ph.D.
0005    Text Processor, Gnosis Software, Inc.
0006    January, 2004
$ cat -b rexx.txt | head -6   # don't number bare lines
     1  LINUX ZONE FEATURE: Regina and NetRexx
     2  Scripting with Free Software Rexx implementations
     3  David Mertz, Ph.D.
     4  Text Processor, Gnosis Software, Inc.
     5  January, 2004
```

除了根据编号来讨论行更容易之外，行号还潜在地为下游进程提供排序和过滤规则。

tr

实用程序 tr 是一个用于转换出现在某个文件中的字符的强有力工具－－或转换 STDIN 中出现的字符，因为 tr 独占式地操作 STDIN ，并独占式地写到 STDOUT（当然允许重定向和管道）。
tr 具有比他的兄长 sed 更有限的功能，后者没有包括在文本实用程序中（本教程也不讨论它），但是仍然在类 UNIX 系统上几乎总是可用。虽然 sed 能够执行正则表达式的通用替换，但是 tr 仅限于替换和删除单个字符（它没有真正的上下文概念）。归根结底，tr 使用目标字符串中的字符来替换那些包含在源字符串中的 STDIN 字符。
一个简单的例子有助于说明 tr 的用法。我们可能有这样一个文件，其中的制表符和空格数量不定，我们希望规格化那些分隔符，并且使用新的分隔符来替换它们。这里的技巧是使用 -s（squeeze，压缩）标记来消除相同的连续字符序列：

源码:

```
$ expand -26 employees | unexpand -a > empl.multitab
$ cat -T empl.multitab
Alice Aaronson^I^I  System Administrator^I    99 9th Street
Bob Babuk^I^I  Programmer^I^I    7 77th Avenue
Carol Cavo^I^I  Manager^I^I    111 West 1st Blvd.
$ tr -s "\t " "| " < empl.multitab | /usr/local/bin/cat -T
Alice Aaronson| System Administrator| 99 9th Street
Bob Babuk| Programmer| 7 77th Avenue
Carol Cavo| Manager| 111 West 1st Blvd.
```

除了显式地转换所列出的字符之外，tr 还支持范围和几种指定的字符类别。例如，为了将小写字符转换为大写字符，您可以使用下面的任一种方法：

源码:

```
$ tr "a-z" "A-Z" < employees
ALICE AARONSON  SYSTEM ADMINISTRATOR    99 9TH STREET
BOB BABUK       PROGRAMMER      7 77TH AVENUE
CAROL CAVO      MANAGER 111 WEST 1ST BLVD.
```

源码:

```
$ tr [:lower:] [:upper:] < employees
ALICE AARONSON  SYSTEM ADMINISTRATOR    99 9TH STREET
BOB BABUK       PROGRAMMER      7 77TH AVENUE
CAROL CAVO      MANAGER 111 WEST 1ST BLVD.
```

如果第二个范围与第一个不一样长，第二个将使用其最后一个字符来填补：

源码:

```
$ tr [:upper:] "a-l#" < employees
alice aaronson  #ystem administrator    99 9th #treet
bob babuk       #rogrammer      7 77th avenue
carol cavo      #anager 111 #est 1st blvd.
```

这里，该脚本使用"#"字符来替换字母表后半部分中的所有大写字母。

您也可以删除 STDIN 流中的字符。您通常可以删除诸如换页之类的特殊字符或者您想要过滤掉的高位字符。不过对于这里的讨论来说，我们还是继续前一个例子：

源码:

```
$ tr -d [:lower:] < employees
A A     S A     99 9 S
B B     P       7 77 A
C C     M       111 W 1 B.
```

面向文件的过滤

使用行集合

到目前为止，我们所见的工具都是单独地处理每行。文本实用程序的另一个子集则把文件看作是行的集合，并对那些行执行全局性的操作。
类 UNIX 操作系统之下的管道就内存使用和延迟而言，能够非常高效地操作。当管道中前面的某个管道向 STDOUT 输出一行时，该行将立即对下一阶段可用。然而，下面的实用程序在它们完成（接近完成）处理之前，不会产生输出。 对于大型文件，这其中某些实用程序可能要花一些时间才能完成处理（但是它们对于所执行的任务仍然是最优化的）。

sort

实用程序 sort 仅执行其名称所暗示的任务：它对一个或多个文件中的行排序。它具有各种各样的选项，允许对文件中的字段或字符位置排序，以及允许修改比较运算（数字、日期、大小写敏感，等等）。

sort 的一种常见应用是用于组合多个文件。在我们前面的例子的基础上：

源码:

```
$ cat employees2
Doug Dobrovsky  Accountant      333 Tri-State Road
Adam Aman       Technician      4 Fourth Street
$ sort employees employees2
Adam Aman       Technician      4 Fourth Street
Alice Aaronson  System Administrator    99 9th Street
Bob Babuk       Programmer      7 77th Avenue
Carol Cavo      Manager 111 West 1st Blvd.
Doug Dobrovsky  Accountant      333 Tri-State Road
```

可以将字段和字段中的字符位置指定为排序规则，并且还可以按数字排序：

源码:

```
$ cat namenums
Alice   123
Bob     45
Carol   6
$ sort -k 2.1 -n namenums
Carol   6
Bob     45
Alice   123
```

uniq

实用程序 uniq 删除完全相同的邻近行——或者在使用某些开关的情况下，可以删除足够近似以至于可以看作是完全相同的邻近行（你可以跳过字段、字符位置，或忽略大小写）。
最通常的情况下，uniq 的输入是 sort 的输出，虽然 GNU sort 本身仅包含有限的能力使

用 -u 来消除重复的行。
uniq 的最典型应用是使用在表达式 sort list_of_things | uniq 中，用以产生一个单项列表（每行一项）。但是有些美妙的用法允许您分析重复情况，或使用不同的重复规则：

源码:

```
$ uniq -d test5 # identify duplicates
Bob
$ uniq -c test5 # count occurrences
1 Alice
2 Bob
1 Carol
$ cat test4
1       Alice
2       Bob
3       Bob
4       Carol
$ uniq -f 1 test4  # skip first field in comparisons
1       Alice
2       Bob
4       Carol
```

tsort

实用程序 tsort 在文本实用程序集合中有一点古怪。该实用程序本身的使用环境相当有限，但是它所做的工作并不是您所以为的文本处理－－tsort 对一个有向图执行拓扑排序。如果不熟悉这个概念，也用不着惊慌：简单地说，tsort 对于查找依赖关系中的适当次序很有用。例如，安装软件包可能需要在某些次序约束条件下进行，或者某些系统守护进程可能需要在其他守护进程之前初始化。
使用 tsort 实际上相当简单。您只需创建一个列出每个已知依赖关系（每个依赖关系之间用空格分隔）的文件（或流）。这个实用程序将为整个集合产生一个适当的次序（虽然不一定是唯一的）。例如：

源码:

```
$ cat dependencies # not necessarily exhaustive, but realistic
libpng XFree86
FreeType XFree86
Fontconfig XFree86
FreeType Fontconfig
expat Fontconfig
Zlib libpng
Binutils Zlib
Coreutils Zlib
GCC Zlib
Glibc Zlib
Sed Zlibc
$ tsort dependencies
Sed
Glibc
GCC
Coreutils
```

```
Binutils
Zlib
expat
FreeType
libpng
Zlibc
Fontconfig
XFree86
```

pr

pr 实用程序是一个用于文本文件的通用格式化器，它提供诸如页眉、 换行、源文本列、页边缩进以及可配置的页面和行宽等格式。然而，pr 本身并不重新换行段落，因此通常可以和 fmt 配合使用。

源码:

```
$ tail -5 rexx.txt | pr -w 60 -f | head
2004-01-31 03:22                                Page 1
  {Picture of Author: http://gnosis.cx/cgi-bin/img_dqm.cgi}
  David Mertz' fondness for IBM dates back embarrassingly many decades.
  David may be reached at mertz@gnosis.cx; his life pored over at
  http://gnosis.cx/publish/. And buy his book: _Text Processing in
  Python_ (http://gnosis.cx/TPiP/).
```

现在成了两列：

源码:

```
$ tail -5 rexx.txt | fmt -30 > blurb.txt
$ pr blurb.txt -2 -w 65 -f | head
2004-01-31 03:24              blurb.txt              Page 1
  {Picture of Author:           at mertz@gnosis.cx; his life
  http://gnosis.cx/cgi-bin/img_d  pored over at http://gnosis.cx
  David Mertz' fondness for IBM   And buy his book: _Text
  dates back embarrassingly many  Processing in Python_
  decades.  David may be reached  (http://gnosis.cx/TPiP/).
```

组合和分割多个文件

comm

实用程序 comm 用于比较已经（按字母顺序）排序的文件的内容。当文件的行被看作是项的无序集合时，这个实用程序就很有用。diff 实用程序虽然没有包括在文本实用程序中，但它是用于比较可能具有单独的修饰（但是被认为是有序的）的文件（比如源代码文件或文档）的通用方法。另一方面， 被看作是记录字段的文件没有任何内在的顺序，因而排序不会改变信息内容。

下面让我们研究一下两个排序后的名称列表之间的区别；所显示的列分别是第一个文件、第二个文件和同时存在于两个文件中的列：

源码:

```
$ comm test2b test2c
                Alice
Betsy
                Bob
Brian
        Cal
                Carol
```

引入一个打乱顺序的名称，我们看到 diff 顺利地进行了比较，而 comm 却不再能够识别重叠的情况：

源码:

```
$ cat test2d
Alice
Zack
Betsy
Bob
Carol
$ diff -U 2 test2d test2c
--- test2d      Sun Feb  1 18:18:26 2004
+++ test2c      Sun Feb  1 18:01:49 2004
@@ -1,5 +1,4 @@
 Alice
-Zack
-Betsy
 Bob
+Cal
 Carol
$ comm test2d test2c
                Alice
        Bob
        Cal
        Carol
Zack
Betsy
Bob
Carol
```

join

实用程序 join 相当有趣；它执行一些基本的关系运算（了解关系数据库理论的读者应该熟悉它们）。 简而言之，join 允许我们查找（排序的）记录集合之间的共有字段。 例如，您可能想知道哪些 IP 地址同时访问了您的 Web 站点和 FTP 站点，以及那些访问的相关信息（所请求的资源、时间，等等，这些信息将在日志文件中）。
为了提供一个简单的例子，假设您向不同的人发放彩色编码的出入证：供应商、合作伙伴、雇员。您希望了解哪些证件类型发放给了雇员。注意姓名是 employees 中的第一个字段，但是在 badges 中是第二个字段，字段之间全都用制表符分隔：

源码:

```
$ cat employees
Alice Aaronson  System Administrator    99 9th Street
Bob Babuk       Programmer      7 77th Avenue
Carol Cavo      Manager 111 West 1st Blvd.
$ cat badges
Red     Alice Aaronson
Green   Alice Aaronson
Red     David Decker
Blue    Ernestine Eckel
Green   Francis Fu
$ join -1 2 -2 1 -t $'\t' badges employees
Alice Aaronson  Red     System Administrator    99 9th Street
Alice Aaronson  Green   System Administrator    99 9th Street
```

paste

实用程序 paste 大致执行 cut 的逆向操作。也就是说，paste 将多个文件组合到列中，比如组合到字段中。默认情况下，文件之间的对应行用制表符分隔，但是您可以通过指定 -d 选项来使用不同的分隔符。
虽然 paste 能够组合不相关的文件（如果其中一个输入较长，则相应留下空字段），但是一般对同步的文件执行 paste 才最有意义。这样的一个例子是识别现有数据文件中的字段：

源码:

```
$ cut -f 1 employees > names
 $ cut -f 2 employees > titles
 $ paste -d "," titles names
 System Administrator,Alice Aaronson
 Programmer,Bob Babuk
 Manager,Carol Cavo
```

标记 -s 允许您逆转行和列的使用，这等于是将某个文件中的后续行转换为分隔的字段：

源码:

```
$ paste -s titles | cat -T
System Administrator^IProgrammer^IManager
```

split

实用程序 split 简单地将一个文件划分为多个部分，每个部分包含指定数量的行或字节（最后一个部分可能是最小的）。这些部分将写到其名称按两个后缀字母排序（默认为 xaa、xab……xzz）的一系列文件中。
虽然 split 可能仅在管理大型文件或数据集时才有用，但是它对处理更结构化的数据也很有用。例如，在作为记录的行小节中，我们看到了一个跨行分割字段的例子——如果我们希望把那些内容逐行地组合回 employees 风格的制表符分隔的字段，那该怎么办呢？下面是实现这个目的一种方法：

源码:

```
$ cut -b 3- multiline | split -l 3 - employee
$ cat employeeab
Bob Babuk
Programmer
7 77th Avenue
$ paste -s employeea*
Alice Aaronson  System Administrator    99 9th Street
Bob Babuk       Programmer      7 77th Avenue
```

csplit

实用程序 csplit 类似于 split，但是它基于文件中的上下文行来划分文件，而不是依照简单的行/字节计数。您可以在一个命令中使用一个或多个划分标准，而且可以将每个标准重复 所希望的任意多次。最有趣的规则类型是用于匹配行的正则表达式。例如下面这段 multiline 中的奇怪换行：

源码:

```
$ head -5 multiline2
Alice Aaronson
System Administrator
99 9th Street
-----
Bob Babuk
$ csplit -zq multiline2 /-----/+1 {*} # incl dashes at end, per chunk
$ cat xx01
Bob Babuk
Programmer
7 77th Avenue
-----
```

上面的划分有点不够适当，因为它没有与数据结构相对应。一种更有用的方法可能是设法分隔 行，并从头到尾地分割那些行。

文件摘要和文件识别

最简单的摘要： wc

到目前为止，我们所见的大多数工具都产生如下输出，它们在很大程度上都可以逆转以创建它们原先的形式－－或者至少每行输入都直接对输出做出了贡献。GNU 文本实用程序中的许多工具都最适合被称为是产生文件的摘要。特别是，这些实用程序的输出一般要比输入短，并且它们全都忽略输入中的大多数信息（从技术上讲，您可以将它们称作是单向功能）。
对输入文件所执行的最简单的单向功能是对行、单词和/或字节计数，这就是 wc（word count，单词计数）所做的工作。关于文件有许多有趣的事情，但是它们在不同的文件显然不是唯一的。例如：

源码:

```
$ wc rexx.txt # lines, words, chars, name
     402    2585   18231 rexx.txt
$ wc -w < rexx.txt # bare word count
    2585
$ wc -lc rexx.txt # lines, chars, name
     402   18231 rexx.txt
```

将其投入实际应用，我可以确定自己编写的哪些 developerWorks 文章字数最多。我可以使用（注意其中包含的 total，tail 的另一个管道可能会删除它）：

源码:

```
$ wc -w *.txt | sort -nr | head -4
   55190 total
    3905 quantum_computer.txt
    3785 filtering-spam.txt
    3098 linuxppc.txt
```

cksum 和 sum

实用程序 cksum 和 sum 产生文件的校验和以及块计数。后一功能只是出于历史原因才存在的，并且是一种实现得不够健壮的方法。两个实用程序都产生一个计算值，这个值在随机选择的文 件之间不大可能相同。特别是，校验和允许您在某个文件被传输或偶然修改后，确定它们没有被破坏的肯定程度。 cksum 实现了四种更健壮的后续技术，其中 -o 1 是 sum 的行为，而默认设置（没有开关）是最佳的。

源码:

```
$ cksum rexx.txt
937454632 18231 rexx.txt
$ cksum -o 3 < rexx.txt
4101119105 18231
$ cat rexx.txt | cksum -o 2
47555 36
```

```
$ cksum -o 1 rexx.txt
10915 18 rexx.txt
```

md5sum 和 sha1sum

实用程序 md5sum 和 sha1sum 在概念上类似于 cksum。顺便说一下，请注意在 BSD 派生的系统中，前一个命令的名称为 md5。然而，md5sum 和 sha1sum 分别产生 128 位和 160 位的校验和，而不是 cksum 的 16 位和 32 位输出。校验和也称为哈希码（hash）。

校验和的长度区别暗示了不同的用途。事实上，比较一个 32 位的哈希值不大可能错误地指出某个文件得到正确的传输而未经更改。但是与预防恶意欺骗者的保护措施相比较，预防意外的保护措施就是一种弱得多的标准。并且 MD5 或 SHA 哈希码是从计算上不可能欺骗的值。像 MD5 或 SHA 这样的加密哈希码的长度对于其强度来说是必要的，但是它们的设计中所考虑的远不只是长度。

设想这样一种场景：有人通过不安全的渠道向您发送了一个文件。为了确保您接收到的是真实数据，而不是某些恶意的替代品，发送者（通过另一个渠道）发布了该文件的一个 MD5 或 SHA 哈希码。攻击者无法使用这个公布的 MD5/SHA 哈希值来创建一个伪造文件——校验和从实用的目的出发唯一地识别预期的正确文件。虽然 sha1sum 从加密上说更好一点，但是由于历史原因，md5sum 使用得更为广泛。

源码:

```
$ md5sum rexx.txt
2cbdbc5bc401b6eb70a0d127609d8772  rexx.txt
$ cat md5s
2cbdbc5bc401b6eb70a0d127609d8772  rexx.txt
c8d19740349f9cd9776827a0135444d5  metaclass.txt
$ md5sum -cv md5s
rexx.txt       OK
metaclass.txt  FAILED
md5sum: 1 of 2 file(s) failed MD5 check
```

使用日志文件

weblog 的结构

weblog 文件为展示文本实用程序的实际应用提供了很好的数据源。标准的 Apache 日志文件每行包含各种各样空格分隔的字段，其中每行描述对某个 Web 资源的一次访问。遗憾的是，空格有时也会出现在引号括起的字段内部，因此处理工作并不如我们想象的那样简单（如果将分隔符排除在字段之外，或许就简单了）。然而，我们必须使用所提供给我们的东西。

下面让我们研究一下在对其执行某些任务之前，我的 weblog 中的一行内容：

源码:

```
$ wc access-log
   24422  448497 5075558 access-log
$ head -1 access-log | fmt -25
62.3.46.183 - -
[28/Dec/2003:00:00:16 -0600]
"GET /TPiP/cover-small.jpg
HTTP/1.1" 200 10146
"http://gnosis.cx/publish/programming/regular_expressions.html"
"Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
```

我们可以看到原先的文件相当大：共有 24,422 个记录。使用 fmt 来换行字段并不总是在字段边界处换行，但是引号能让您看到那些字段是什么。

提取 Web 站点访问者的 IP 地址

对 weblog 文件所执行的一个非常简单的任务就是提取所有站点访问者的 IP 地址。这将在常用的管道模式中组合一些实用程序（我们将研究前几个实用程序）：

源码:

```
$ cut -f 1 -d " " access-log | sort | uniq | head -5
12.0.36.77
12.110.136.90
12.110.238.249
12.111.153.49
12.13.161.243
```

我们可能还想知道总共究竟有多少访问者访问过该站点：

源码:

```
$ cut -f 1 -d " " access-log | sort | uniq | wc -l
    2820
```

统计出现次数

在上一节中，我们确定了有多少访问者访问过我们的 Web 站点，但是我们或许还想知道那 2820 个访问者中，每个对总的 24,422 次点击的贡献是多少。特别是，哪些访问者访问得最频繁呢？我们可以对一行执行：

源码:

```
$ cut -f 1 -d " " access-log | sort | uniq -c | sort -nr | head -5
1264 131.111.210.195
 524 213.76.135.14
 307 200.164.28.3
 285 160.79.236.146
 284 128.248.170.115
```

虽然这种方法有效，但是将柱状图部分引入可重用的 shell 脚本可能更为理想：

源码:

```
$ cat histogram
#!/bin/sh
sort | uniq -c | sort -nr | head -n $1
$ cut -f 1 -d " " access-log | ./histogram 3
1264 131.111.210.195
 524 213.76.135.14
 307 200.164.28.3
```

现在我们可以将面向行的任何项目列表管道输出到 histogram shell 脚本。我们希望显示的、出现得最频繁的项的数目是传递给该脚本的一个参数。

生成新的特殊报告

有时现有的数据文件包含我们需要的信息，但是不一定具有下游进程所需要的格式。作为一个基本的例子，假设您希望提取 weblog 的结构中显示的 weblog 的多个字段，并以不同的顺序组合它们（同时忽略不需要的字段）：

源码:

```
$ cut -f 6 -d \" access-log > browsers
$ cut -f 1 -d " " access-log > ips
$ cut -f 2 -d \" access-log | cut -f 2 -d " "
       | tr "/" ":" > resources
$ paste resources browsers ips > new.report
$ head -2 new.report | tr "\t" "\n"
:TPiP:cover-small.jpg
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
62.3.46.183
:Publish:Programming:regular_expressions.html
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
62.3.46.183
```

产生 resources 的行使用了两遍 cut，它们具有不同的分隔符。这是因为，对于 access-log 所认为的 REQUEST，它所包含的信息比我们希望 RESOURCE 提供的信息更多。

源码:

```
$ cut -f 2 -d \" access-log | head -1
GET /TPiP/cover-small.jpg HTTP/1.1
```

我们还决定将 Apache 日志中的路径分隔符改为使用冒号路径分隔符（这是旧的 Mac OS 格式，但是我们这里只是为了展示某种类型的操作） 。

cut_by_regex

接下来的这个脚本将我们在本教程中所见的许多实用程序组合到一个相当复杂的管道中。假设我们知道自己想要从某个数据文件截取一个字段，但是却不知道它的字 段位置。显而易见，通过可视化的检查能够提供答案，但是为了自动化不同数据文件类型的处理，我们可以截取与某个正则表达式相匹配的 任意一个 字段：

源码:

```
$ cat cut_by_regex
#!/bin/sh
# USAGE: cut_by_regex <pattern> <file> <delim>
cut -d "$3" -f \
  `head -n 1 $2 | tr "$3" "\n" | nl | \
   egrep $1 | cut -f 1 | head -1` \
  $2
```

实际上，我们可以这样使用：

源码:

```
$ ./cut_by_regex "([0-9]+\.){3}" access-log " " | ./histogram 3
1264 131.111.210.195
 524 213.76.135.14
 307 200.164.28.3
```

这其中有几个部分需要作进一步的说明。反引号（backtick）是 bash 中的一种特殊语法，用于将一个命令的结果看作是另一个命令的参数。特别是，反引号中的管道产生了与作为第一个参数来提供的正则表达式相匹配的 第一个字段编号。
它是如何管理这点的呢？ 首先我们仅提取数据文件的第一行；然后将指定的分隔符转换为新行字符（现在是每行一个字段）；随后我们对结果行/字段编号；接着我们搜索一个具有预期模式 的行；之后我们仅从该行截取字段编号；最后我们仅接受第一个匹配项（即使有多个匹配字段）。组合一个不错的管道需要费些思量，但是大多数管道都可以通过这 种方式来组合。

结束语和参考资料

结束语

本教程简单地介绍了使用 GNU 文本实用程序所能做的工作的一小部分。最后几个例子开始揭示出，通过创造性地使用管道和重定向，这些实用程序将是多么功能强大。将整体转换划分为有用的中 间数据的关键是，要么将中间数据保存到另一个文件中，要么将它管道输出给某个处理该数据格式的实用程序。
我要感谢同事 Andrew 在我准备本教程时所提供的帮助。

参考资料

Linux 用户很可能已经安装了他们的发行套件中的文本实用程序。但是如果还没有的话－－或者您在运行它们的旧版本，您可以从它们的 Web 站点下载最新的 27 个 GNU 文本实用程序。

最新版本的实用程序已合并到 GNU 核心实用程序（总共 88 个）中。

Windows 用户可以在 Cygwin 包中找到这些以及其他许多实用程序，而 Mac OS X 用户可以尝试 Fink。

通过在命令行键入 man utility-name 或 utility-name --help 来获得本教程介绍的任何实用程序的帮助。或者在 GNU 站点查看在线手册页。

Arnold Robbins 撰写的文章 "Opening the software toolbox"（1994 年 4 月）介绍了一些 GNU 工具，以及关于程序开发和使用的一些"软件工具"哲学。

一些常用的 UNIX 和 Linux 命令行工具已在诸如 UNIX Tutorial for Beginners 和 Basic UNIX commands 之类的在线指南中介绍过了。

Peter Seebach 的 "编写 Linux 实用程序的艺术: 开发有用的小命令行工具"（developerWorks，2004 年 1 月）提供了编写您自己的实用程序的指导方针和最佳实践。

通过也是由 Peter Seebach 撰写的 developerWorks 系列文章 "UNIX utilities"：Part 1、Part 2、Part 3 和 Part 4（developerWorks，2001 年 5 月，6月）来了解关于 UNIX 实用程序、如何使用它们以及如何编写它们的更多信息。

一旦学会了编写自己的实用程序，您就可以学习如何将它们转换为库了！通过 developerWorks 教程 "Building a cross-platform C library"（developerWorks，2001 年 6 月）的指导，这是很容易实现的。

通过阅读文章 "Basics of the Unix Philosophy" 和 "The Unix Philosophy" 来使您更了解自己的 UNIX/Linux 环境。

通过三部分的 developerWorks 系列 "Bash 实例"（developerWorks，2000 年 3 月）来开始 bash 编程入门。

David Mertz 的 "使用规则表达式" 教程（developerWorks，2000 年 9 月）是了解使用正则表达式的工具（比如 grep 和 csplit ）的很好起点。

David 的书 Text Processing in Python （Addison Wesley，2003，ISBN：0-321-11254-7）也包含了关于正则表达式的介绍，以及关于使用 Python 实现本教程中的许多技术的详细讨论。

David 的文章 "人人可用的 Rexx"（developerWorks，2004 年 2 月）介绍了一种执行简单文本处理任务的替代方法。文本实用程序的功能范围与 Rexx 编程语言的核心用途几乎完全相同。

反馈

请告诉我们本教程对您是否有帮助，以及我们如何能够做得更好。我们还想知道您希望看到其他哪些教程。

关于本教程内容的问题，请通过电子邮件 mertz@gnosis.cx 联系作者 David Mertz。