## Утилиты обработки текста и текстовые редакторы

### Цель работы

Ознакомиться со стандартными средствами UNIX-подобных операционных систем, предназначенными для обработки текстовой информации.

### Задания к работе

1. Войти в систему с собственной учетной записью.
2. Вывести расширенный список процессов (ps aux) и сохранить эту информацию в файл ps.out
3. Выполнить сортировку файла ps.out по имени пользователя-владельца процесса, сохранить результат в файл sorted.ps
4. Разделить сортированную информацию из файла sorted.ps по нескольким файлам, в каждом файле - один ключ сортировки (например, файл root.ps - содержит строки, начинающиеся со слова root, user.ps - со слова user и т.д.)
5. Добавить в начало каждой строки созданных файлов текст <li> (тег разметки списков языка HTML), результаты сохранить в файлах исходное\_имя.html (например).
6. Подсчитать количество строк в каждом из полученных файлов (это значение будет соответствовать числу процессов, запущенных пользователем). Результаты объединить со строкой вида:

<p><b>Итого процессов: NN</b>

1. где NN - количество процессов, запущенных пользователем и сохранить в отдельные файлы (например root.total, user.total и т.п).
2. Создать файл index.html следующего содержания:

<html>  
<title>Статистика процессов</title>  
<body>  
<h1>Распределение процессов по пользователям</h1>

1. Командами обработки текста последовательно добавить в файл index.html ранее созданные файлы, разделенные строкой "<hr noshade>" (например в таком порядке: файл root.html, строка-разделитель, файл root.totlal, user.html, строка-разделитель, user.total и т.д.). Завершить формирование файла добавлением строки вида:

</body></html>

1. Просмотреть полученный файл (index.html) в веб-браузере.
2. Создать скрипт, который автоматизирует проделанные операции.

### Методические указания

Редактирование текстовых файлов одна из наиболее часто выполняемых работ на компьютере. Соответственно и программ для обработки текста разработано множество. Они отличаются друг от друга в той или иной степени, однако есть существенное различие, связанное с двумя типами текстовых файлов, которые могут быть созданы такими программами.

Первый тип - это простые ASCII-файлы, использующие код ASCII для представления символов. К этому же классу отнесем и те файлы, которые содержат специальные служебные символы или последовательности символов кода ASCII, используемые для форматирования текста при выводе на экран и принтер. Но существенно то, что эти форматирующие последовательности (почти) не мешают Вам прочитать текст, содержащийся в файле с помощью любого средства просмотра или простейшего текстового редактора. Примерами таких файлов могут служить файлы, создаваемые редакторами типа встроенного редактора программы Midnight Commander, файлы в формате .html, файлы, создаваемые программой notepad в Windows и vi в UNIX.

Второй тип - это файлы, использующие собственный формат для представления текста (в которых символы текста представлены специальными последовательностями). Текст в таких файлах невозможно прочитать без той программы, в которой файл создавался, или специальной программы-просмотрщика. Примеры: файлы в форматах .doc (MS Word), .rtf (Reach text format), .odt (текстовый документ OpenOffice Writer).

В ЮНИКС-системах традиционно большое число узкоспециализированных утилит, среди которых немало и команд обработки текста. Однако, прежде чем перейти к рассмотрению их возможностей, необходимо сделать отступление в сторону регулярных выражений - универсального инструмента обработки текстов.

#### Регулярные выражения

Регулярные выражения - это высокоуровневое средство обработки информации, представленной в виде символьных последовательностей: строк и текстов. С помощью регулярных выражений можно формировать шаблоны поиска и замены фрагментов текста любой сложности. Построение таких шаблонов в общем случае задача не совсем тривиальная и требует определенной практики. О подробностях использования языка регулярных выражений можно прочитать в специальной литературе, здесь же приведем список исходных шаблонов, а ниже - несколько примеров использования.

|  |  |
| --- | --- |
| **Шаблон** | **Назначение** |
| . | Заменяет любой символ. Выражение "п.уть" находит "путь" и "пить" |
| ^Петя | Находит слово, только если оно расположено в начале абзаца. |
| Петя$ | Находит слово, только если оно расположено в конце абзаца. |
| \* | Предыдущий символ может повторяться много раз (от нуля до бесконечности): например, "Аб\*в" находит "Ав", "Абв", "Аббв", "Abbbc" и так далее. Комбинация .\* может использоваться для любого символа. |
| + | Предыдущий символ должен присутствовать хотя бы один раз или в неограниченных количествах: "AX.+4" находит "AX 4", но не "AX4"Всегда находится наиболее длинный текст внутри абзаца. Если абзац содержит текст AX 4 AX4, то будут найдены от первой A до последней 4. |
| ? | Символ перед ? может отсутствовать, либо присутствовать только один раз. "Тексты?" находит слова "Текст" и "Тексты". |
| \C | Абсолютно точно находит этот символ (не цифру!), в данном случае C (если, например, нужно найти знак доллара в регулярных выражениях: \$) |
| \n | Находит жесткий разрыв строк, вставленный с помощью Shift+Enter. |
| \t | Находит символ табуляции |
| \> | Элемент поиска должен присутствовать в конце слова: "нот\>" находит "блокнот", но не "ноты". |
| \< | Элемент поиска должен присутствовать в начале слова: "\>нот" находит "ноты", но не "блокнот". |
| ^$ | Ищет пустые абзацы. |
| ^. | Ищет первый символ абзаца. |
| & | Указывает на найденный текст. Используется при замене. |
| [abc123] | Находит все символы в скобках |
| [a-e] | Находит все символы между a и e |
| [a-eк-о] | Находит все символы между буквами a-e и к-о. |
| [^a-в] | Находит все символы, кроме a-в |
| \xXXXX | Находит все символы с четырехзначным шестнадцатеричным кодом XXXX.Номер символа кода зависит от используемого шрифта. |
| этот|тот | Находит все "этот" и все "тот". |
| {2} | Символ перед открывающей скобкой должен присутствовать столько раз, сколько указано в скобках. 8{2} находит 88. |
| {1,2} | Символ перед открывающей скобкой должен присутствовать столько раз, сколько указано в скобках. 8{1,2} находит 8 и 88. |
| ( ) | Круглые скобки следует использовать для задания символов внутри скобок, как ссылок. После этого можно ссылаться на первую ссылку в текущем выражении с \1, на вторую ссылку с \2 и так далее. Если текст содержит число 13487889 и вы ищете регулярное выражение (8)7\1\1, то будет найдено число 8788. |
| [:digit:]? | Находит число (0 до 9). [:digit:]\* находит последовательность цифр. |
| [:space:]? | Находит пропуски: пробелы и символы табуляции. |
| [:print:]? | Находит печатаемые символы. |
| [:cntrl:]? | Находит непечатаемые символы. |
| [:alnum:]? | Находит буквенно-цифровые символы (цифры и буквы). |
| [:alpha:]? | Находит буквенные символы (буквы). |
| [:lower:]? | Находит буквы строчные буквы. |
| [:upper:]? | Находит буквы прописные буквы. |

Для логического объединения выражений с помощью операторов И/ИЛИ, необходимо использовать скобки. Например, " ((a[A-Я]\*)|(аб[А-Я]\*)|(б[А-Я]\*))$" находит элементы, начинающиеся с пробела и идущих затем "a" или "ab" или "b" и присутствующих в конце абзаца.

#### Специализированные команды

Специализированные команды обработки текста, такие как cat, cut, sort, split и пр., предназначены для решения специфичных задач, например для сортировки, фильтрации или объединения строк. Приведем несколько примеров использования таких команд (подробное описание в man имя\_команды или имя\_команды --help).

#### Команда cat

Использование: cat [КЛЮЧ] [ФАЙЛ]...

Сцепляет ФАЙЛ(ы) или стандартный ввод на стандартный вывод. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

aag@stilo:~> **cat errors.log** // *вывести файл на экран*  
Mar 2 09:13:37 stilo kernel: ACPI: PCI Interrupt 0000:00:1d.0[A] -> Link [LNKA] -> GSI 5 (level, low) -> IRQ 5  
Mar 2 09:13:37 stilo kernel: PCI: Setting latency timer of device 0000:00:1d.0 to 64  
Mar 2 09:13:37 stilo kernel: usb usb1: root hub lost power or was reset  
Mar 2 09:13:37 stilo kernel: ACPI: PCI Interrupt 0000:00:1d.1[B] -> Link [LNKD] -> GSI 11   
  
aag@stilo:~> **cat -n errors.log** // *включить нумерацию строк при выводе*  
1 Mar 2 09:13:37 stilo kernel: ACPI: PCI Interrupt 0000:00:1d.0[A] -> Link [LNKA] -> GSI 5 (level, low) -> IRQ 5  
2 Mar 2 09:13:37 stilo kernel: PCI: Setting latency timer of device 0000:00:1d.0 to 64  
3 Mar 2 09:13:37 stilo kernel: usb usb1: root hub lost power or was reset  
4 Mar 2 09:13:37 stilo kernel: ACPI: PCI Interrupt 0000:00:1d.1[B] -> Link [LNKD] -> GSI 11   
...

#### Команда cut

Использование: cut [КЛЮЧ]... [ФАЙЛ]...

Печатает выбранные части строк из каждого ФАЙЛА на стандартный вывод. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

aag@stilo:~> **cut -c 1-16 errors.log** // *вывести первые 16 символов каждой строки*  
Mar 3 15:43:53   
Mar 3 16:22:08   
Mar 3 17:09:05   
Mar 3 17:09:14   
...  
aag@stilo:~> **cut -c 23- errors.log** //*вывести строки, начиная с 23-го символа*  
kernel: scsi 2:0:0:0: Direct-Access USB 2.0 Flash Disk 0.00 PQ: 0 ANSI: 2  
kernel: sd 2:0:0:0: [sdb] 2015231 512-byte hardware sectors (1032 MB)  
kernel: sd 2:0:0:0: [sdb] Write Protect is off  
kernel: sd 2:0:0:0: [sdb] Mode Sense: 00 00 00 00  
kernel: sd 2:0:0:0: [sdb] Assuming drive cache: write through  
kernel: usb-storage: device scan complete  
hald: mounted /dev/sdb1 on behalf of uid 1000  
syslog-ng[2237]: STATS: dropped 0  
...  
//*выбрать символы с 8 по 16 и с 23 до конца строки, использовать "пробел" как разделитель*   
aag@stilo:~> **cut -c 8-16,23- errors.log --output-delimiter=' '**  
15:43:50 kernel: scsi 2:0:0:0: Direct-Access USB 2.0 Flash Disk 0.00 PQ: 0 ANSI: 2  
15:43:50 kernel: sd 2:0:0:0: Attached scsi generic sg2 type 0  
15:43:50 kernel: usb-storage: device scan complete  
15:43:53 hald: mounted /dev/sdb1 on behalf of uid 1000  
16:22:08 syslog-ng[2237]: STATS: dropped 0  
...  
// *вывести только 1,3 и 4 поля, разделенные пробелом;*   
// *использовать "->" в качестве нового разделителя*  
aag@stilo:~> **cut -f 1,3,4 errors.log --delimiter=' ' --output-delimiter='->'**  
Mar->3->15:43:50  
Mar->3->15:43:53  
Mar->3->16:22:08  
Mar->3->17:09:05  
Mar->3->17:09:14  
...

#### Команда sort

Использование: sort [КЛЮЧ]... [ФАЙЛ]...

Печатает сортированное слияние всех ФАЙЛ(ов) на стандартный вывод. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

// *сортировать файл по убыванию, отбрасывая повторы строк;*  
// *результат записать в новый файл*  
aag@stilo:~> **sort -ur errors.log -o e.log**  
aag@stilo:~> **cat e.log** // *показать созданный файл*  
Dec 10 21:35:33 stilo kernel: ACPI: AC Adapter [AC] (on-line)  
Dec 10 21:35:31 stilo auditd[2920]: Init complete, auditd 1.2.6 listening for events  
Dec 10 21:35:30 stilo syslog-ng[2511]: Changing permissions on special file /dev/xconsole  
Dec 10 21:35:30 stilo auditd: Config file /etc/audit/auditd.conf doesn't exist, skipping  
Dec 10 21:35:29 stilo network: Starting the NetworkManagerDispatcher  
Dec 10 21:35:29 stilo dhcdbd: Started up.  
Dec 10 21:35:28 stilo syslog-ng[2511]: syslog-ng version 1.6.11 starting

#### Команда split

Использование: split [КЛЮЧ] [ФАЙЛ [ПРЕФИКС]]

Выводит фиксированного размера части ФАЙЛА в файлы ПРЕФИКСaa, ПРЕФИКСab, ...; по умолчанию размер части равен 1000 строк, а ПРЕФИКС равен `x'. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

aag@stilo:~> **split e.log log\_** // разделить по 1000 строк в файлы с префиксом "log\_"   
aag@stilo:~> **ls log\*** // показать список файлов  
log\_aa log\_ac log\_ae log\_ag log\_ai log\_ak log\_am log\_ao log\_aq log\_as  
log\_ab log\_ad log\_af log\_ah log\_aj log\_al log\_an log\_ap log\_ar   
  
aag@stilo:~> **split -d e.log err\_** // использовать числовые суффиксы в именах  
aag@stilo:~> ls err\*  
err\_00 err\_02 err\_04 err\_06 err\_08 err\_10 err\_12 err\_14 err\_16 err\_18   
err\_01 err\_03 err\_05 err\_07 err\_09 err\_11 err\_13 err\_15 err\_17

#### Команда strings

Использование: strings [КЛЮЧ] [ФАЙЛ(ы)]

Выводит строки из ФАЙЛА(ов) (stdin по умолчанию)

// *вывести строки, длиннее 350 байт из всех файлов с именем*  
// *начинающимся с "err", при выводе строк показывать имя файла*  
// *приведен сокращенный фрагмент результата обработки*  
aag@stilo:~> **strings -fn 350 err\***   
errors.log: 2462225 Jan 9 10:08:58 stilo suse\_register[4313]: Argument Dump: $VAR1 = ...  
err\_04.log: 2462225 Jan 9 10:08:58 stilo suse\_register[4313]: Argument Dump: $VAR1 = ...  
errors.log~: 3477727 Jan 9 10:08:58 stilo suse\_register[4313]: Argument Dump: $VAR1 = ...

#### Команда tail

Использование: tail [КЛЮЧ]... [ФАЙЛ]...

Печатает последние 10 строк каждого из ФАЙЛОВ на стандартный вывод. Если задано несколько ФАЙЛОВ, сначала печатает заголовок с именем файла. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

// *вывести 2 последние строки с указанием имени файла;*  
// *обновлять информацию по мере записи в файл*  
stilo:/var/log/apache2 # **tail -fv -n2 access\_log**  
==> access\_log <== // *имя файла*  
192.168.0.191 - - [04/Mar/2008:14:02:38 +0600] "GET /nettech/work01/ HTTP/1.1" 200 13776 "-" "Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.8.0.3) Gecko/20060524 ASPLinux/1.5.0.3-0.110am Firefox/1.5.0.3 pango-text"  
192.168.0.199 - - [04/Mar/2008:14:03:05 +0600] "GET /tasks.shtml HTTP/1.1" 200 18692 "http://aag.asoiu/tasks.shtml" "Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.8.0.3) Gecko/20060524 ASPLinux/1.5.0.3-0.110am Firefox/1.5.0.3 pango-text"  
...

#### Команда head

Использование: head [КЛЮЧ]... [ФАЙЛ]...

Печатает первые 10 строк каждого ФАЙЛА на стандартный вывод. Если задано несколько ФАЙЛОВ, сначала печатает заголовок с именем файла. Если ФАЙЛ не задан или задан как -, читает стандартный ввод.

aag@stilo:~> **head -vn 3 err\_\*** // *вывести по 3 первых строки из файлов с именем err\_любыесимволы*  
==> err\_00 <==  
Oct 16 21:35:53 stilo gconfd (aag-5419): Обнаружен разрешённый адрес   
Nov 8 07:56:43 stilo syslog-ng[2537]: syslog-ng version 1.6.11 going down  
Nov 8 07:56:42 stilo kernel: Kernel logging (proc) stopped.  
==> err\_01 <==  
Nov 27 21:13:00 stilo gconfd (aag-5423): Обнаружен разрешённый адрес "xml:readonly:/etc/opt/gnome/gconf/gconf.xml.mandatory" к источнику конфигурации только-для-чтения в позиции 0  
...

Самостоятельно рассмотреть назначение и параметры команд uniq и wc.

#### Команда grep

Синтаксис

grep [КЛЮЧ(и)] ОБРАЗЕЦ [ФАЙЛЫ(ы)...]  
grep [КЛЮЧ(и)] [-e ОБРАЗЕЦ | -f ФАЙЛЫ] [ФАЙЛЫ(ы)]

Утилита grep выполняет поиск образца в текстовых файлах и выдает все строки, содержащие этот образец. Она использует компактный недетерминированный алгоритм сопоставления.

Будьте внимательны при использовании в списке\_образцов символов $, \*, [, ^, |, (, ) и \, поскольку они являются метасимволами командного интерпретатора. Лучше брать весь список\_образцов в одиночные кавычки '... '.

Если имя\_файла не указано, grep предполагает поиск в стандартном входном потоке. Обычно каждая найденная строка копируется в стандартный выходной поток. Если поиск осуществлялся в нескольких файлах, перед каждой найденной строкой выдается имя файла.

Опции -E и -F влияют на способ интерпретации списка\_образцов программой grep. Если указана опция -E, программа grep интерпретирует образцы в списке как полные регулярные выражения. Если же указана опция -F, grep интерпретирует список\_образцов как фиксированные строки. Если ни одна из этих опций не указана, grep интерпретирует элементы списка\_образцов как простые регулярные выражения.

**Ключи команды**

|  |  |
| --- | --- |
| **-b** | Предваряет каждую строку номером блока, в котором она была найдена. Это может пригодиться при поиске блоков по контексту (блоки нумеруются с 0). |
| **-c** | Выдает только количество строк, содержащих образец. |
| **-h** | Предотвращает выдачу имени файла, содержащего сопоставившуюся строку, перед собственно строкой. Используется при поиске по нескольким файлам. |
| **-i** | Игнорирует регистр символов при сравнениях. |
| **-l** | Выдает только имена файлов, содержащих сопоставившиеся строки, по одному в строке. Если образец найден в нескольких строках файла, имя файла не повторяется. |
| **-n** | Выдает перед каждой строкой ее номер в файле (строки нумеруются с 1). |
| **-s** | Подавляет выдачу сообщений о не существующих или недоступных для чтения файлах. |
| **-v** | Выдает все строки, за исключением содержащих образец. |
| **-w** | Ищет выражение как слово, как если бы оно было окружено метасимволами \< и \>. |
| **-e список\_образцов** | Задает один или несколько образцов для поиска. Образцы в **списке\_образцов**должны разделяться символами новой строки. Пустой образец можно задать, введя два символа новой строки подряд. Если одновременно с этой опцией не указана опция **-E** или **-F**, каждый образец будет рассматриваться как простое регулярное выражение. Утилита **grep** воспринимает несколько опций **-e** и **-f**. При поиске строк, соответствующих образцу, используются все заданные образцы, но порядок сопоставления не определен. |
| **-E** | Сопоставлять с полными регулярными выражениями. Рассматривать каждый заданный образец как *полное регулярное выражение*. Если любое из полных регулярных выражений-образцов сопоставляется с входной строкой, строка считается соответствующей. Пустое полное регулярное выражение соответствует любой строке. Каждый образец будет интерпретироваться как полное регулярное выражение, за исключением метасимволов **\(** и **\)**, причем:   1. Полное регулярное выражение, за которым идет **+**, соответствует одному или более вхождениям полного регулярного выражения. 2. Полное регулярное выражение, за которым идет **?**, соответствует 0 или одному вхождению полного регулярного выражения. 3. Полным регулярным выражениям, разделенным символами **|** или символами новой строки, соответствуют строки, сопоставляющиеся с любым из указанных выражений. 4. Полные регулярные выражения можно брать в круглые скобки **()** для группировки.   Максимальный приоритет имеют операторы **[]**, затем **\*?+**, конкатенация, и наконец, оператор **|** и символ новой строки. |
| **-f файл\_образцов** | Читает один или несколько образцов из файла с указанным полным именем **файл\_образцов**. Образцы в **файле\_образцов** завершаются символом новой строки. Пустой образец можно задать с помощью пустой строки в**файле\_образцов**. Если только вместе с этой опцией не указана опция **-E** или **-F**, каждый образец считается простым регулярным выражением. |
| **-F** | Задает сопоставление с фиксированными строками. Каждый образец ищется как строка, а не как регулярное выражение. Если входная строка содержит любой из образцов в качестве подряд идущих байтов, такая строка считается соответствующей образцу. Пустая строка-образец соответствует любой строке. (Подробнее см. man grep). |
| **-q** | Немногословный режим. В стандартный выходной поток не выдается ничего, кроме сопоставившихся строк. Если одна из входных строк соответствует образцу, возвращается статус выхода 0. |
| **-x** | Считает сопоставившимися только строки, все символы которых использованы при сопоставлении с фиксированной строкой или регулярным выражением. |

Применение

Можно задавать несколько опций -e и -f. При этом утилита grep использует все заданные образцы при сопоставлении с входными строками. (Учтите, что порядок проверки не задается. Если реализация находит среди образцов пустую строку, она может искать сначала именно ее, тем самым, сопоставление будет найдено для каждой строки, а остальные образцы, по сути, - проигнорированы.)

Опция -q дает средства простого определения, находится ли образец (или строка) в группе файлов. При поиске в нескольких файлах она обеспечивает более высокую производительность (поскольку позволяет завершить работу, как только будет найдено первое соответствие) и не требует дополнительных усилий пользователя при формировании набора файлов-аргументов (поскольку grep вернет нулевой статус выхода при обнаружении соответствия даже если при работе с предыдущими операндами-файлами произошла ошибка доступа или чтения.)

Примеры использования

//*Найти все вхождения слова "network", вывести номер каждой строки*  
aag@stilo:~> **grep -i -n network errors.log**  
20085:Mar 2 14:28:27 stilo network: Starting the DHCP DBUS Daemon  
20086:Mar 2 14:28:27 stilo network: Starting the NetworkManagerDispatcher  
20087:Mar 2 14:28:27 stilo network: Starting the NetworkManager  
...  
//*Поиск пустых строк*  
aag@stilo:~> **grep ^$ errors.log**  
  
// *Поиск строк, содержащих фиксированные подстроки:*  
// *вывести все строки, содержащие подстроки warning, IPv6 или и ту, и другую*  
aag@stilo:~> **grep -E 'warning|IPv6' errors.log**  
Nov 27 21:12:02 stilo ifup-wireless: eth0 warning: using NO encryption  
Nov 27 21:12:06 stilo kernel: IPv6 over IPv4 tunneling driver  
Nov 27 21:12:13 stilo kernel: eth0: no IPv6 routers present  
Jan 15 12:28:15 stilo python: [7979]: warning: Unable to set locale.  
Feb 28 09:21:51 stilo SuSEfirewall2: Warning: ip6tables does not support state matching. Extended IPv6 support disabled.  
...  
// *Поиск строк, соответствующих образцу:*  
// *вывести все строки, начинающиеся с "Mar 3 09:22:26" или "Jan 9 18:28:33"*  
aag@stilo:~> **grep -E '^(Mar 3 09:22:26)|(Jan 9 18:28:33).{1,}$' errors.log**  
Jan 9 18:28:33 stilo avahi-dnsconfd[3689]: Got SIGTERM, quitting.  
Jan 9 18:28:33 stilo sshd[4123]: Received signal 15; terminating.  
Jan 9 18:28:33 stilo avahi-daemon[3465]: Got SIGTERM, quitting.  
Mar 3 09:22:26 stilo gconfd (root-3243): starting (version 2.20.0), pid 3243 user 'root'  
Mar 3 09:22:26 stilo gconfd (root-3243): Resolved address "xml:readonly:/etc/gconf/gconf.xml.mandatory" ...  
Mar 3 09:22:26 stilo gconfd (root-3243): Resolved address "xml:readwrite:/root/.gconf" t...  
Mar 3 09:22:26 stilo gconfd (root-3243): Resolved address "xml:readonly:/etc/gconf/gconf...  
Mar 3 09:22:26 stilo gconfd (root-3243): Resolved address "xml:readonly:/etc/gconf/gconf.xml...

#### Потоковый редактор sed

sed (от английского Stream EDitor) — потоковый редактор. Простая, но мощная программа, выполняющая преобразования последовательного потока текстовых данных. Команда sed получает входной поток (обычно, файл) построчно, редактирует каждую строку, согласно правилам, определенным в собственном языке (sed-скрипт), и выводит результат в выходной поток.

sed часто считают не интерактивным текстовым редактором. Однако, он отличается от обычных текстовых редакторов «инвертированностью» по отношению к тексту и набору команд для его редактирования. Обычные текстовые редакторы вначале загружают весь текст документа, а затем применяют к нему команды по одной, в то время как sed вначале загружает набор команд, а затем применяет его к каждой строке текста. Так как одновременно в памяти находится только одна строка, sed может обработать произвольно большие текстовые файлы.

Следующий пример демонстрирует типичное использование sed:

sed -e 's/oldstuff/newstuff/g' inputFileName > outputFileName

Здесь s — команда замены; g — глобально, что означает «в во всей строке». Строка oldstuff — образец искомого текста (на основе регулярного выражения), строка newstuff - новый текст, которым нужно заменить oldstuff. Команда замены (s///) безусловно является самой мощной и часто используемой командой sed.

В Unix sed часто используется в виде фильтра при конвейерной обработке (|, или pipe):

generate\_data | sed -e 's/x/y/'

Несколько внутренних команд sed могут быть записаны в файле (sed-скрипт, имя файла - любое) и затем применены в виде:

sed -f *имя\_файла\_с\_командами* inputFileName > outputFileName

Помимо замены, возможны и другие формы простой обработки. Например, следующий сценарий удаляет пустые строки или строки, которые содержат только пробелы:

sed -e '/^ \*$/d' inputFileName

Комплексные конструкции sed возможны до такой степени, что он может быть представлен как высоко специализированный, хотя и простой, язык программирования.

Синтаксис

Существует два варианта запуска sed:

sed [options] 'command' file(s)  
sed [options] — f scriptfile file(s)

В первом варианте возможно задание команды редактирования sed (заключенной в одинарные кавычки) в командной строке. Во втором варианте задается файл сценария scriptfile, содержащий команды sed. Если не заданы обрабатываемые файлы, происходит чтение со стандартного ввода.

Доступны следующие параметры командной строки:

-е cmd

Следующий аргумент является инструкцией редактирования; параметр необходим только при задании более чем одной инструкции.

-f scriptfile

Аргумент является файлом, содержащим команды редактирования.

-n

Подавить вывод по умолчанию; sed отображает строки только по команде p или при установленном ключе р команды s.

-V

Отобразить номер версии sed.

--quiet

Идентично -n.

--expression=*cmd*

Идентично -е.

--file=*file*

Идентично —f .

Перечень команд sed по группам

Простое редактирование

Команда Действие  
а\ Добавление текста после строки  
с\ Замена текста (обычно области текста)  
i\ Вставка текста перед строкой  
d Удаление строк  
s Замена  
y Преобразование символов

Информация о строках

Команда Действие  
= Отобразить порядковый номер строки  
1 Отображать управляющие символы в кодах ASCII  
p Отобразить строку

Обработка ввода/вывода

Команда Действие  
n Пропустить текущую строку и перейти к следующей  
r Послать на ввод sed содержимое другого файла  
w Записать исходные строки в другой файл  
q Завершить работу сценария sed (конец вывода)

Копирование и вставка

Команда Действие  
h Создать дубликат пространства шаблонов в специальном буфере;   
 предыдущее содержимое буфера уничтожается  
H Добавить дубликат пространства шаблонов к содержимому специального буфера  
g Перезаписать пространство шаблонов содержимым буфера  
G Добавить к пространству шаблонов содержимое буфера  
x Обменять содержимое пространства шаблонов и специального буфера

Обработка многострочного ввода

Действие Команда  
N Прочитать следующую строку из ввода (с добавлением символа новой строки)  
D Удалить все до символа новой строки  
P Отобразить символы до символа новой строки

Примеры использования

Замена текста  
  
# заменить одинарные междустрочные интервалы на двойные  
sed G   
  
# удалить двойные междустрочные интервалы   
sed 'n;d'   
  
# нумерация непустых строк файла (по конвейеру передается предварительно обработанный файл)  
sed '/./=' file | sed '/./N; s/n/ /' file  
  
# подсчет количества строк (аналог "wc -l")  
sed -n '$='   
  
# добавление отступа в начало каждой строки   
sed 's/^/\t /'   
  
# удаление отступов (пробелы,табуляции) с начала каждой строки  
sed 's/^[ \t]\*//'   
  
# удаляем отступы, пробелы, табуляции и с конца, и с начала строки  
sed 's/^[ t]\*//;s/[ t]\*$//'   
  
# центрируем весь текст посредине при ширине колонки 79 символов.В первом способе  
# пробелы в начале строки нужны, а пробелы в конце строки дополняются до конца строки.  
# Второй способ, пробелы в начале строки отбрасываются в центр строки, и нет  
# завершающих пробелов до конца строки.  
sed -e :a -e 's/^.{1,77}$/ & /;ta' # Способ 1  
sed -e :a -e 's/^.{1,77}$/ &/;ta' -e 's/( \*)1/1/' # Способ 2  
  
# подстановка (найти и заменить) "foo" на "bar" в каждой строке  
sed 's/foo/bar/' # заменяет только первое вхождение в строке  
sed 's/foo/bar/4' # заменяет только 4 вхождение в строке  
sed 's/foo/bar/g' # заменяет ВСЕ вхождения в строке  
sed 's/(.\*)foo(.\*foo)/1bar2/' # заменяет друг за другом  
sed 's/(.\*)foo/1bar/' # заменяет только завершающее слово  
  
# ЗАМЕНЯЕТ "foo" на "bar" ТОЛЬКО для строк ,содержащих "base"  
sed '/base/s/foo/bar/g'  
  
# меняет "red","green","blue" на "purple"  
sed 's/red/purple/g;s/green/purple/g;s/blue/purple/g'   
  
# обратный порядок строк  
sed '1!G;h;$!d' # способ 1  
sed -n '1!G;h;$p' # способ 2  
  
# обратный порядок символов в строке   
sed '/n/!G;s/(.)(.\*n)/&21/;//D;s/.//'  
  
# соединяет строки (аналог "paste")  
sed '$!N;s/n/ /'  
  
# добавляем запятые к числовым строкам, меняя "1234567" на "1,234,567"  
sed -e :a -e 's/(.\*[0-9])([0-9]{3})/1,2/;ta' # остальные sed  
  
# добавляем пустую строку каждые пять 5 строк (после строк 5, 10, 15, 20, итд.)  
sed 'n;n;n;n;G;'   
  
# печать первых 10 строк файла (аналог "head")  
sed 10q  
  
Вывод строк  
  
# печать только строк, которые совпадают с regexp (аналог "grep")  
sed -n '/regexp/p' # способ 1  
sed '/regexp/!d' # способ 2  
  
# печать только строк, НЕ совпадающих с regexp (как "grep -v")  
sed -n '/regexp/!p' # способ 1, соответствует вышеприведенному  
sed '/regexp/d' # способ 2, простейший синтаксис  
  
# grep для AAA и BBB и CCC (в любом порядке)  
sed '/AAA/!d; /BBB/!d; /CCC/!d'  
  
# grep для AAA и BBB и CCC (в таком же порядке)  
sed '/AAA.\*BBB.\*CCC/!d'  
  
# grep для AAA или BBB или CCC (как в "egrep")  
sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d   
  
# печать строк длинее N символов (в примере - 65)  
sed -n '/^.{65}/p'  
  
# печать строк короче N символов (в примере - 65)  
sed -n '/^.{65}/!p' # способ 1, соответствует вышеприведенному  
  
# печать фрагмента файла от заданного регулярного выражения до конца файла  
sed -n '/regexp/,$p'  
  
# печать фрагмента файла, основанная на номерах строк (в примере - строки с 8 по 12)  
sed -n '8,12p' # способ 1  
sed '8,12!d' # способ 2  
  
# печать только выбранной строки (в примере - 52)  
sed -n '52p' # способ 1  
sed '52!d' # способ 2  
sed '52q;d' # способ 3, эффективно для больших файлов  
  
# печать каждой 7-ой строки начиная со строки 3  
sed -n '3,${p;n;n;n;n;n;n;}'   
  
# печать части файла между двумя регулярными выражениями  
sed -n '/regexp1/,/regexp2/p'   
  
Удаление строк  
  
# печать всего файла , КРОМЕ части между двумя регулярными выражениями  
sed '/regexp1/,/regexp2/d'  
  
# удаление последовательно повторяющихся строк файла (аналог "uniq").  
sed '$!N; /^(.\*)n1$/!P; D'  
  
# удалить непоследовательные повторяющиеся строки файла.   
sed -n 'G; s/n/&&/; /^([ -~]\*n).\*n1/d; s/n//; h; P'  
  
# удаление первых 10 строк файла  
sed '1,10d'  
  
# удалить последнюю строку файла  
sed '$d'  
  
# удалить последние 10 строк файла  
sed -e :a -e '$d;N;2,10ba' -e 'P;D'  
  
# удалить все пустые строки файла (также как и "grep '.' ")  
sed '/^$/d' # способ 1  
sed '/./!d' # способ 2  
  
# удалить все ПОСЛЕДОВАТЕЛЬНЫЕ пустые строки из файла кроме первых двух:  
sed '/^$/N;/n$/N;//D'  
  
# удалить все пустые строки с начала файла:  
sed '/./,$!d'  
  
# удалить все пустые строки в конце файла  
sed -e :a -e '/^n\*$/{$d;N;ba' -e '}' # рaботает на всех sed  
  
# удалить последнюю строку каждого параграфа  
sed -n '/^$/{p;h;};/./{x;/./p;}'  
  
# удалить теги HTML (включая многострочные теги)  
sed -e :a -e 's/<[^>]\*>//g;/</N;//ba'

Как видно из примеров, sed – своего рода оболочка для регулярных выражений, что позволяет, несмотря на ограниченный набор встроенных команд, использовать sed для сложной обработки текстов.

#### Язык обработки шаблонов awk

Язык сценариев awk – это мощный инструмент для обработки текстовых файлов при помощи шаблонов (регулярных выражений). Файлы могут состоять из записей фиксированной или переменной длины. Границы записей обозначаются разделителем (по умолчанию это символ новой строки). awk может использоваться как из командной строки, так и в awk-сценариях. Обычно эту программу можно вызвать, введя команду awk (или gawk для GNU-версии) в командной строке интерпретатора.

Язык сценариев awk позволяет:

* С легкостью обрабатывать текстовый файл, как если бы он состоял из полей и записей текстовой базы данных
* Использовать переменные для внесения изменений в базу данных
* Выполнять команды интерпретатора из сценария
* Выполнять арифметические и строковые операции
* Использовать конструкции программирования, такие как циклы и условные операторы
* Определять пользовательские функции
* Обрабатывать вывод команд интерпретатора
* Обрабатывать аргументы командной строки постепенно
* Отображать оформленные отчеты

#### Текстовые редакторы

#### Редактор mcedit

Встроенный редактор файлового менеджера Midnight Commander вызывается во время работы в этой программе нажатием клавиши [F4] при условии, что в инициализационном файле Midnight Commander-а установлена опция "use\_internal\_edit". Однако его можно вызвать независимо от MC просто из командной строки, командой mcedit.

Этот редактор обеспечивает выполнение большинства функций редактирования, присущих полноэкранным редакторам текста. С его помощью можно редактировать файлы практически любого размера, поскольку верхняя граница для размера редактируемого файла составляет 16 Мегабайт.

Поддерживаются следующие возможности: копирование, перемещение, удаление, вырезание и вставка блоков текста; отмена предыдущих операций (по комбинации клавиш [Ctrl]-[u]); выпадающие меню; вставка файлов; макро-определения; поиск и замена по регулярным выражениям (а также собственный вариант операций поиска и замены, основанный на функциях scanf-printf); выделение текста по комбинации клавиш [Shift]-[стрелки] в стиле MS Windows - MAC (только для linux-консоли); переключение между режимами вставки-замены символа; а также операция обработки блоков текста командами оболочки shell.

Редактор очень прост. Для того, чтобы узнать, какие клавиши вызывают выполнение определенных действий, достаточно просмотреть выпадающие меню, которые вызываются нажатием клавиши [F9] в окне редактора.

Если Вы работаете в консоли Linux, то можно использовать следующие комбинации клавиш, не перечисленные в меню:

* [Shift]-[клавиши стрелок] - выделение блока текста.
* [Ctrl]-[Ins] - копирует блок в файл cooledit.clip.
* [Shift]-[Ins] - производит вставку последнего скопированного в cooledit.clip блока в позицию курсора.
* [Shift]-[Del] - удаляет выделенный блок текста, запоминая его в файле cooledit.clip.
* По клавише [Enter] вставляются символы конца строки и перевода каретки, причем на следующей строке автоматически устанавливается отступ.

Редактор поддерживает макросы. Для того, чтобы определить макрос, нажмите [Ctrl]-[R], после чего введите строки команд, которые должны быть выполнены. После завершения ввода команд снова нажмите [Ctrl]-[R] и свяжите макрос с какой-нибудь клавишей или комбинацией клавиш, нажав эту клавишу (комбинацию). Макрос будет вызываться нажатием [Ctrl]-[A] и назначенной для него клавиши. Макрос можно также вызвать нажатием любой из клавиш [Meta] ([Alt]), [Ctrl] или [Esc] и назначенной макросу клавиши, при условии, что данная комбинация не используется для вызова какой-либо другой функции.

Макро-команды после определения записываются в файл cedit/cooledit.macros в Вашем домашнем каталоге. Вы можете удалить макрос удалением соответствующей строки в этом файле.

Перенос фрагментов текста из одного файла в другой при работе в консоли выполняется следующим образом:

1. отмечаем начало блока с помощью клавиши [F3];
2. перемещаем курсор к концу блока;
3. отмечаем конец блока с помощью клавиши [F3];
4. набираем комбинацию [Ctrl]-[Insert];
5. закрываем этот файл, открываем другой;
6. ставим курсор туда, куда хотим вставить данный фрагмент и нажимаем комбинацию клавиш [Shift]-[Insert].

При работе из эмулятора терминала (в графическом режиме) та же задача решается по-иному, через перенос фрагмента текста через другой файл (по умолчанию используется файл ~/.cedit/cooledit.clip). Выделите фрагмент текста, выберите пункт меню "File/Copy to file" и нажмите [Enter]. Затем переходите в другой файл, ставите курсор туда, куда надо вставить фрагмент, и выбираете пункт меню "File/Insert file".

Более подробную информацию об использовании редактора можно получить в руководстве (man mcedit).