Александр Прокудин

Искать везде, искать всегда

Использовать для поиска информации несколько различных средств совсем неэффективно. Проще всего обратиться за помощью к программам, которые сумеют отыскать необходимое как в файлах и документах, так и в почтовой базе.

Исторически сложилось так, что пользователи Linux и других Unix-подобных систем редко имеют дело с документами в бинарных форматах, если не считать музыки и видео. Вместо DOC – LaTeX, HTML или вообще TXT, вместо XLS — Gnumeric на основе XML и т. д. Кодировка у создаваемых файлов до недавних пор обычно совпадала с кодировкой системной локали, вследствие чего процедура поиска любых нужных данных не вызывала затруднений — подходил любой поисковый механизм, умеющий работать с текстом. Однако стоило Linux двинуться в сторону десктопа, как требования пользователей тут же выросли. Теперь они хотят искать документы внутри архивов и по тегам музыкальных файлов, и чтобы все это происходило быстро и как можно более удобно. Кроме того, русскоязычному пользователю, который постоянно сталкивается с проблемой дуализма кодировок (кто-то привык работать в CP1251, а кто-то — в KOI8-R), обычно хочется, чтобы поиск происходил в обеих кодировках.

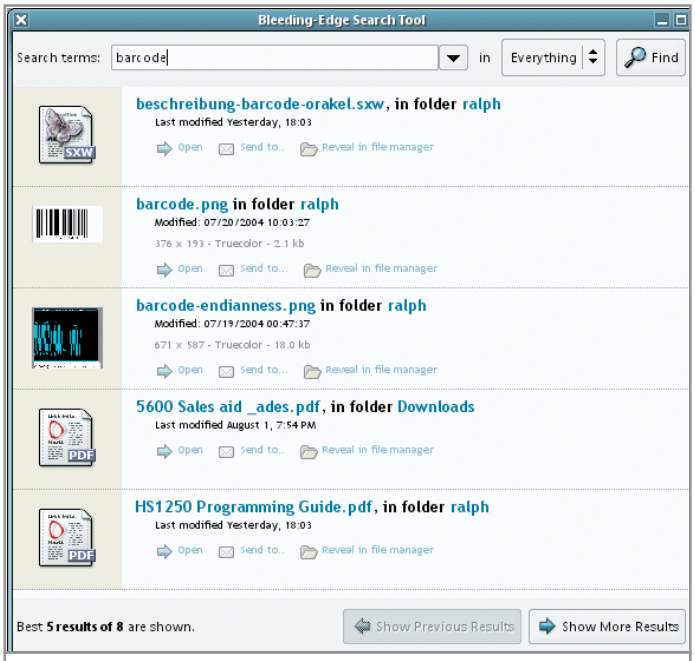
С одной стороны, у нас есть повод позавидовать пользователям Windows и Mac OS X: ведь они обладают такими средствами как Google Desktop и Spotlight. С другой стороны, решить поставленную задачу можно с помощью старых и новых наработок свободных систем. Для начала попробуем разобраться с тем, как устроены поисковые механизмы и по каким принципам они работают.

Принцип действия

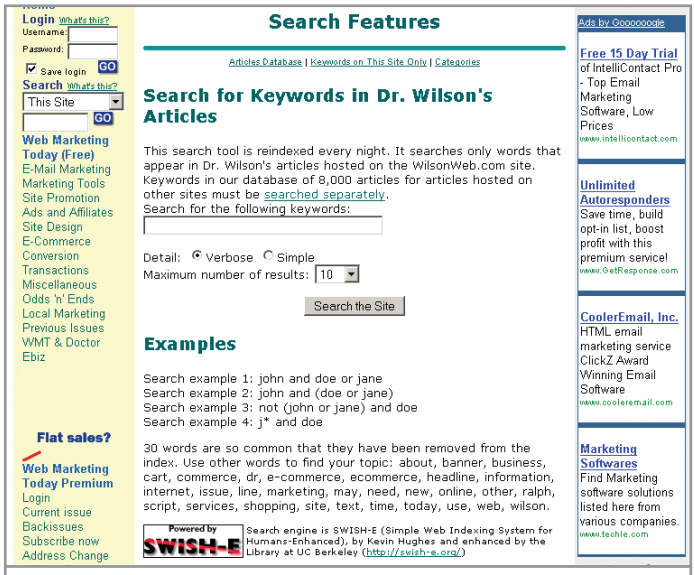
Практически все умные поисковые системы работают одинаково. Сначала запускается индексатор – программа, которая проходит по локальной или удаленной файловой системе и собирает всю информацию о находящихся в ней данных, которая затем будет использоваться при поиске. Эта информация записывается в базу данных, которая обычно сохраняется в бинарном виде для экономии дискового пространства и ускорения доступа к ней. После создания основного индекса с некоторой периодичностью, определяемой пользователем либо заданной в системном планировщике cron, запускается индексатор.

Поскольку индексаторы имеют тенденцию отъедать массу системных ресурсов, их обычно запускают в такое время суток, когда компьютер не используется. К примеру, в Fedora Core программа updatedb, составляющая индекс файлов на всем жестком диске, автоматически запускается в двенадцать часов ночи. Непосредственно для поиска среди всех проиндексированных данных используется другая программа — с текстовым, графическим или веб-интерфейсом.

Но это устаревший подход к поиску данных. Существует более продвинутая концепция, используемая в новейших разработках. Заключается она в том, что вместо индексатора, запускаемого по расписанию или же отдельному указанию пользователя, в системе работает демон индексации. Одним



Beagle осуществляет поиск не только по директориям, но и по почтовой базе, а также кешу веб-браузера



Swish-e предоставляет пользователю удобный веб-интерфейс, при помощи которого осуществлять поиск гораздо легче

из примеров, использующих данную разработку, является локальная поисковая система Beagle.

Beagle

Эта многообещающую разработку IT-ресурс Ars Technica назвал самым ожидаемым продуктом 2005 года. После того как разрабатывающая Beagle компания Ximian была выкуплена Novell, проект получил значительную финансовую поддержку и стал развиваться не в пример быстрее. Сейчас Beagle можно считать одним из ключевых программных продуктов Novell, написанных в среде Mono — своеобразной альтернативе Microsoft .NET.

Beagle состоит из двух частей. Первой частью является специализированный демон beagled. Запустившись в первый раз, он проходит по системе и создает индекс — базу данных, в которой в сжатом виде хранится информация обо всех существующих файлах и их содержании. Создав первоначальный индекс, демон прячется в засаду и ловит пробегающие мимо него сообщения ядра о появлении в системе новых файлов или изменении уже существующих. За отправку этих сообщений отвечает модуль ядра inotify. Поймав сообщение, демон производит переиндексацию изменившегося файла.

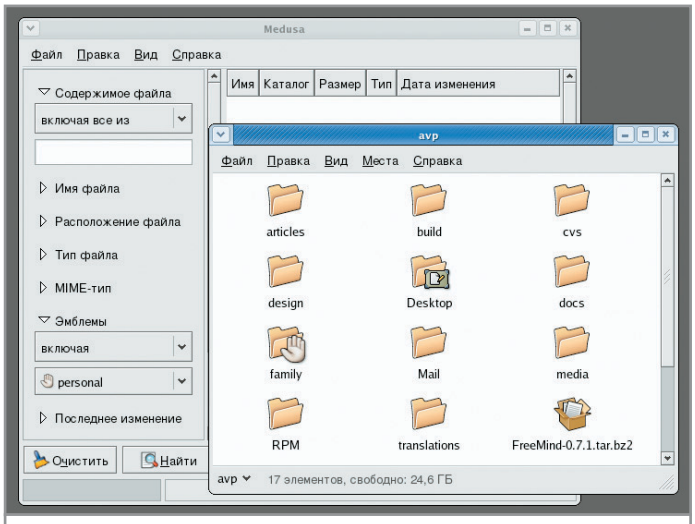
Beagle понимает массу самых различных форматов: текстовые — DOC, XLS, PPT, SXW PDF, RTF; графические — JPG, PNG; музыкальные — MP3, OGG Vorbis, FLAC, MIDI, Musepack, Monkey Audio; файлы с исходным текстом — C, C++, C#, Java, Python; а также файлы Texinfo и LyX. Помимо этого, он может читать кеш-файлы RSS-агрегаторов Blam и liferead и историю переписки в интернет-пейджере Gaim.

Beagled по своей сути является не чем иным, как популярным индексатором Lucene, портированным с Java на Mono. Но одним лишь портированием дело не ограничилось. Все перечисленные выше возможности по индексированию данных реализованы уже авторами Beagle, причем модульно. Это значит, что нужный фильтр к не поддерживаемому пока типу файлов можно дописать самому. Для взаимодействия с другими приложениями Beagle использует все более популярный сервер DBUS, прочитать о котором можно, к примеру, на сайте разработчиков: www.freedesktop.org/Software/dbus.

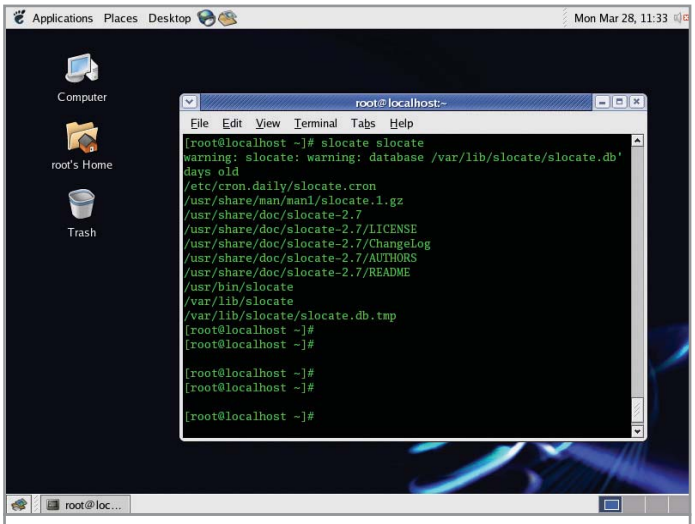
Собственно поиск производится из программы с графическим интерфейсом под названием B.E.S.T, которое расшифровывается как Bleeding-Edge Search Tool. Надо сказать, что расходы на содержание пары дизайнеров и специалистов по проектированию пользовательских интерфейсов весьма благотворно сказались на облике программы. Результаты поиска выдаются по релевантности, причем не сплошным текстом, а приятными глазу блоками. Кроме того, все найденные файлы можно прямо из B.E.S.T открыть в соответствующей программе, тут же ответить на письмо, а из найденного фрагмента разговора в ICQ повторно выйти на собеседника.

Над сборкой программы придется немного потрудиться. Начнем с того, что inotify не входит в стандартную поставку ядер, поэтому вам надо либо устанавливать специально собранное для вашего дистрибутива ядро (если такое есть), либо брать патч и собирать ядро самостоятельно. Признаюсь, за добрых пять лет работы с Linux я ни разу не пересобрал ядро. Поленился и в этот раз, потому зашел на beaglewiki.org и обнаружил там ссылку на уже собранное ядро для моего дистрибутива. Дальнейшая процедура сборки потребует крепких нервов. Вам понадобятся следующие пакеты: gtk-sharp, glade-sharp, gecko-sharp, gnome-sharp, dbus-sharp, gconf-sharp, gmime-sharp и mono как минимум версии 1.0.4. Начиная с версии 0.0.6.1 Beagle требует еще и dbus версии 0.23.1 вместо CVS dbus.

Когда все эти пакеты будут установлены, выведите справку к сценарию конфигурации сборки ./configure --help и прочитайте в ней информацию о приведенных в разделе



С помощью системы Medusa можно осуществлять поиск по иконкам файлов или директорий



Текстовые консольные утилиты пока что слишком рано списывать со счетов

«Optional Features» ключах. Так, передав ./configure ключ –enable-epiphany-extension, вы разрешите сборку фильтра Beagle, который будет индексировать закешированные браузером Epiphany веб-страницы. Также в корневом каталоге с исходными текстами Beagle расположен каталог mozilla-extension, в котором находится расширение для браузера Firefox, позволяющее демону beagled на лету индексировать просматриваемые вами страницы.

Swish-e

Проектов с общим названием Swish на самом деле два: Swish-e и Swish++, который, по сути, является тем же самым Swish-e, только немного переписанным одним из недовольных пользователей оригинала. Непосредственно Swish-e работает по следующему принципу: сначала все индексирует, потом спрашивает, что надо найти. При этом как для индексирования, так и для выполнения запросов используется один и тот же бинарный файл.

Если Beagle требовал терпения при сборке, то Swish предполагает наличие у пользователей практических познаний в языках сценариев, особенно Perl. Сама же сборка заключается в выполнении стандартной последовательности действий: ./configure; make; sudo make install.

После установки нужно выбрать для хранения индекса такой каталог, доступ к которому будут иметь все пользователи данного компьютера. Как вариант можно попросту создать в корневом каталоге директорию indeces (в которой будут находиться конфигурационный файл swish.conf и файлы индексов), а затем разрешить читать и писать данные из нее всем пользователям.

В самом простом варианте конфигурационный файл для данного поискового сервера выглядит примерно следующим образом:

```
IndexDir /home/user
IndexOnly .html
IndexFile ./user_html.index
```

Вместо «user» достаточно подставить свое имя пользователя в системе, чтобы Swish-e смог проиндексировать все документы с расширением HTML в вашем домашнем каталоге. Количество индексируемых программой каталогов можно произвольно увеличивать, добавляя туда, например, каталоги локальной сети, монтируемые с параметрами -t nfs или -t smbfs. Теперь имея на руках готовый конфигурационный файл, запустим процесс индексирования документов:

```
$ swish-e -c user-html.conf
```

После создания индекса можно приступать к поиску:

```
swish-e -f avp_html.index -w beagle
# SWISH format: 2.4.3
# Search words: beagle
# Removed stopwords:
# Number of hits: 5
# Search time: 0.001 seconds
# Run time: 0.034 seconds
1000 /home/avp/articles/search/beagle/msg00047.html "Beagle roadmap." 18582
1000 /home/avp/Desktop/data/search/beagle/msg00047.html "Beagle roadmap." 18582
791 /home/avp/articles/search/beagle/msg00039.html "Beagle Networking" 5236
791 /home/avp/Desktop/data/search/beagle/msg00039.html "Beagle Networking" 5236
608 /home/avp/cvs/beagle/Tiles/template-mockup.html "Beagle List" 3713
```

Поиск из консоли даже при обращении к памяти вводившихся команд не слишком удобен, поэтому стоит использовать входящие в комплект готовые сценарии на Perl, создающие привычную для пользователей веб-страницу, на которой можно вводить запрос и получать результат непосредственно в окне браузера.

Ценность Swish-e заключается в том, что его возможности по индексированию при наличии опыта программирования можно легко расширять. Поскольку на вход индексатор принимает либо HTML, либо TXT, можно написать сценарий для предварительного разбора документов вроде .DOC или .PDF в один из этих форматов. Фактически по нахождении файла с заданной маской расширения, скажем *.pdf, Swish-e вызовет сценарий для разбора этого файла в понятный ему формат, проиндексирует результат и сохранит его в своей базе данных. Обилие приложений, позволяющих получать нужные данные из бинарных файлов (например, тот же catdoc для чтения файлов .DOC и .XLS), в руках знакомого с программированием человека способно сделать Swish-e достойной заменой пресловутому Google Desktop.

Medusa

Medusa — система поиска, сильно завязанная на среде GNOME. Она состоит из индексатора, запускаемого командой medusa-indexd, с указанием URI корневого индексируемого каталога в качестве аргумента, и графического приложения к поиску по созданному индексу. На сайте разработчиков данного поисковика (http://members.cox.net/sinzui/medusa/) в качестве текущей версии указана 6.1, однако в дереве разработки находится уже 6.3, и, пожалуй, стоит поработать именно с ней. Выкачать срез дерева разработки можно серией команд:

```
export
CVSROOT = :pserver:anonymous@anoncvs.gnome.org:/cvs/gnome
cvs login
```

В ответ на предложение ввести пароль просто нажмите ввод и затем наберите команду:

```
cvs -z3 -d :pserver:anonymous@anoncvs.gnome.org:/cvs/gnome
co medusa
```

В дальнейшем для обновления среза будет достаточно заходить в каталог medusa и давать команду:

```
cvs update -Pd
```

Для сборки вам понадобятся средства разработки GNOME и пакет gnome-common, содержащий сценарий gnome-autogen.sh, без которого собрать что-либо из CVS для GNOME просто не получится. После того как сценарий отработает, можно уточнить параметры сборки запуском другого сценария с нужными ключами — ./configure . Скажем, если вас не устраивает, что по умолчанию Medusa установится в каталог /usr/local, можно указать сценарию ключ -perfix = /правильный_путь, например: -prefix = /opt.

В текущей версии как индексатор, так и непосредственно графический клиент запускаются с правами обычного пользователя. Так безопаснее, хотя индексы можно создавать и с правами суперпользователя. Созданные индексные файлы по умолчанию хранятся в каталоге ~/.medusa домашней папки пользователя.

Графический клиент medusa-gui весьма удобен тем, что критерии поиска можно уточнить при помощи сразу десятка параметров, из которых мы чуть подробнее остановимся на следующих двух:

- Тип файла. Можно выбрать между приложением, текстовым документом, музыкой, просто файлом или каталогом. Medusa полагается на MIME-типы в GNOME, по которым и определяет, среди каких данных искать или не искать нужные документы.

- Эмблемы. По этому критерию видно, что графический клиент Medusa, как это и было задумано в начале разработчиками, ориентирован на использование в качестве дополнения к файловому менеджеру Nautilus. Поскольку в Nautilus файлам и каталогам присваивается эмблема — значок, несущий некую смысловую нагрузку, то при поиске можно выставить логический переключатель: может искомый документ иметь ту или иную эмблему, или же наоборот, обязан ее не иметь. Один из таких примеров вы можете видеть на приведенном скриншоте на странице слева.

При необходимости (скажем, с целью использования в сценариях или в процессе доступа с удаленного компьютера по ssh) вместо msearch-gui можно воспользоваться его консольным аналогом — msearch.

Текстовые средства

Несмотря на то что все больше пользователей отдают предпочтение средствам поиска с графическим интерфейсом, консольные утилиты все-таки пока не стоит сбрасывать со счетов, они еще играют серьезную роль при выполнении различных каждодневных задач.

slocate

При помощи программы slocate поиск файлов происходит не по содержанию, а по названию. Понятно, что скорость при этом обратно пропорциональна удобству, поскольку в индексе, создаваемом программой updatedb, хранятся только имена файлов и их расположение. Slocate (secure locate) сейчас полностью заменила собой устаревшую locate. Разница состоит в том, что locate показывала пользователям местонахождение даже тех файлов, о существовании которых им знать было не положено. В slocate же поведение программы можно настраивать: для отключения безопасного режима при индексировании достаточно передать updatedb ключ -l 0. Если индексирование файлов по расписанию планируется запускать в рабочее время, стоит изменить значение аргумента команды «renice», указанного в файле /etc/cron.daily/slocate.cron, на меньший, поскольку по умолчанию сценарий выглядит так:

```
#!/bin/sh
./etc/updatedb.conf
renice +19 -p $$ >/dev/null 2>&1
/usr/bin/updated
```

Одного взгляда на него достаточно, чтобы понять, почему многие совсем отключают индексацию по расписанию. Значения «+5» будет вполне достаточно для того, чтобы комфортно работать в момент индексирования, которое бы при этом не длилось слишком долго. При поиске с помощью slocate можно пользовать регулярные выражения. Например, по команде:

```
slocate *search*.sxw
```

Результат может быть следующим:

```
/home/avp/articles/search/search_v2.sxw
/home/avp/articles/linux_search/linux_search_0.3.sxw
```

find

Программа find также используется для поиска файлов по имени. Обычный запрос с использованием find выглядит следующим образом:

```
find /home/fc3/ -name '*gnomeprint*.rpm' | sort
/home/fc3/cd2/Fedora/RPMS/libgnomeprint15-0.37-
10.i386.rpm
/home/fc3/cd4/Fedora/RPMS/libgnomeprintui22-devel-2.8.0-
1.i386.rpm
```

В данном случае мы просим отыскать внутри каталога /home/fc3 все файлы, имя которых попадает в маску *gnomeprint*. Программу удобно использовать в паре с другими приложениями. К примеру, немного преобразовав введенную выше строку, мы получим команду, по запуску которой все найденные объекты с именем, попадающим в маску *gnomeprint*.rpm, будут автоматически устанавливаться в систему:

```
sudo rpm -Uvh `find /home/fc3/ -name '*gnomeprint*.rpm'`
```

Для find можно задать несколько дополнительных опций, например ограничить поиск файлами с датой последнего доступа или изменения. Так, ключом -amin n задается время в минутах, прошедшее с последнего открытия искомого файла. Таким образом, все файлы внутри текущего каталога (включая подкаталоги), открывавшиеся десять минут назад, будут выведены по команде:

```
find . -amin 10 '*'
```

Для отсчета дней удобнее всего будет воспользоваться ключом -atime -n, где n – не что иное, как количество дней. Кроме того, find понимает разницу между обычным файлом, каталогом и символической ссылкой. Этот критерий поиска задается ключом -type, которому передается аргумент типа искомых данных. Например, по указанной ниже команде выводятся все символические ссылки по маске *win* внутри текущего каталога:

```
find . -type l '*win*'
```

grep

Программа grep (и ее варианты egrep и fgrep) обычно используется для контекстного поиска с использованием регулярных выражений, но с ее помощью можно искать и файлы. Правда, стоит иметь в виду, что grep изначально не умеет читать архивы и теги аудиофайлов.

Однако и внутри бинарных данных grep может откопать полезную вам информацию. Более того, по умолчанию она все файлы принимает за бинарные (это поведение можно изменить, указав программе ключ -binary-files = text, после чего бинарные файлы будут рассматриваться как текстовые). Например, запустив следующую команду, мы просим grep найти все файлы, в которых встречается слово «Genesis» или «genesis», после чего отсортировать их в алфавитном порядке командой «sort»:

```
grep '[Gg]enesis' -r ./ | sort
Бинарный файл ./Genesis/1970_Trespass/03 — Genesis —
Angels.ogg совпадает
Бинарный файл ./Genesis/1970_Trespass/04 — Genesis —
Stagnation.ogg совпадает
```

Иногда очень удобно использовать grep благодаря способности командной оболочки Linux перенаправлять вывод команд в другие команды. Например, мы хотим найти все файлы в директории /root, содержащие в названии последовательность символов «conf». Для этого введем следующую команду:

```
ls /root | grep conf
```

Также можно использовать grep вместе с командой «find»:

```
vi $(find) / -print | grep fooname
```

Таким образом, «find» ищет все файлы и каталоги начиная с корня файловой системы, а затем передает результат на обработку grep, фильтрующей результат и оставляющей только те файлы, названия которых содержат «fooname». Эти файлы открываются в редакторе vi (если их несколько, то они открываются по очереди).

Делаем выводы

Подводя итог, стоит отметить, что ускорение доступа к данным разработчики пробуют реализовать несколькими способами. Один из них заключается в использовании поиска с предварительным индексированием, который был описан выше. Второй способ — это создание надстройки над стандартной файловой системой, которая хранит в специализированной базе данных информацию о метаданных имеющихся в ней документов, что позволяет быстро находить документы по их описанию, а не по фактическому содержанию (причина гигантского размера большинства индексов) или местоположению. В мире свободного программного обеспечения такими разработками являются GNOME Storage и DBFS. Вполне возможно предположить, что два подхода рано или поздно сольются в один, и тогда в мире наступит полное и окончательное технологическое пресыщение.