Поиск файлов

Команды **locate** и **find** полезны для поиска файла в файловой системе. Хотя обе команды выполняют одинаковые задачи, каждая из них выполняет свою методику, со своими собственными отличительными преимуществами и недостатками.

Команда locate

Команда **locate** использует базу данных. Эта база данных может обновляться вручную администратором с помощью команды **updatedb**, хотя обычно эта команда запускается автоматически каждый день.

Команда **updatedb** создает базу данных всех файлов, найденных на компьютере, для быстрого поиска. Команде **updatedb** можно запретить искать конкретное имя, путь или файловую систему, изменив соответствующую строку в файле **/etc/update.conf.** Вот файл по умолчанию во всей его полноте:

Этот файл может редактировать пользователь root. Любое имя, путь или файловая система, указанные в соответствующей строке, не будут добавлены в базу данных. Любая строка, начинающаяся с символа #, будет игнорироваться.

Команда **locate** принимает строку поиска в качестве аргумента. Например, чтобы найти файл с именем **passwd**, используйте следующую команду:

```
locate passwd
```

При использовании команды locate необходимо учитывать следующее:

- Команда locate вернет результаты файлов, к которым у текущего пользователя есть доступ.
- Команда locate отобразит все файлы с поисковым запросом в любом месте имени файла. Например, команда locate passwd будет соответствовать и /etc/passwd, и /etc/ thishaspasswdinit.
- Как и большинство вещей в Linux, команда locate чувствительна к регистру. Например, команда locate passwd не будет соответствовать файлу с именем / etc / PASSWD. Чтобы команда locate не учитывала регистр символов, используйте параметр -i.

Поскольку команда **locate** работает с базой данных, она может очень быстро работать даже в системе с сотнями тысяч файлов. Однако, если вы хотите использовать команду locate для поиска файла, который был создан совсем недавно, он не сможет найти файл, если база данных не обновлялась с момента создания файла.

Аналогично, база данных может содержать устаревшую информацию о файлах, которые могли существовать в самом недавнем прошлом, поэтому команда сообщит о них неправильно, поскольку они все еще существуют.

В следующем примере демонстрируются последствия, возникающие, когда база данных не обновляется в режиме реального времени:

1. Новый файл с именем **lostfile** изначально не был найден командой **locate**.

```
cd ~
touch lostfile
locate lostfile
```

2. После обновления базы данных командой **updatedb** команда **locate** может найти файл

```
updatedb
locate lostfile
```

3. После удаления файла потерянного файла команда locate все равно сообщит, что файл «существует».

```
rm lostfile
locate lostfile
```

Команда find

Если необходимо искать файлы, которые в данный момент находятся в файловой системе, то следует использовать команду **find**. Команда **find** работает медленнее, чем команда **locate**, потому что она ищет каталоги в режиме реального времени; однако, это не страдает от проблем, связанных с устаревшей базой данных.

Команда **find** ожидает каталог в качестве первого аргумента. Это будет отправной точкой поиска. Команда **find** будет искать этот каталог и все его подкаталоги. Если каталог не указан, команда **find** начнет поиск в текущем каталоге.

Обратите внимание, что текущий каталог упоминается как . в следующем примере, но также может упоминаться как ./ Команда **find** использует опцию **-name** для поиска файлов по имени.

```
find / -name passwd
find / -name 'pass*'
```

Команда find также предлагает множество опций для поиска файлов, в отличие от команды locate, которая ищет только файлы на основе имени файла. Следующая таблица иллюстрирует некоторые из наиболее часто используемых параметров и критериев:

ОСНОВНЫЕ ПАРАМЕТРЫ КОМАНДЫ FIND

- **-P** никогда не открывать символические ссылки
- **-L** — получает информацию о файлах по символическим ссылкам. Важно для дальнейшей обработки, чтобы обрабатывалась не ссылка, а сам файл.
- **-maxdepth** — максимальная глубина поиска по подкаталогам, для поиска только в текущем каталоге установите 1.
- **-depth** — искать сначала в текущем каталоге, а потом в подкаталогах
- **-mount** искать файлы только в этой файловой системе.
- **-version** — показать версию утилиты find
- **-print** — выводить полные имена файлов
- **-type f** — искать только файлы
- **-type d** — поиск папки в Linux

КРИТЕРИИ

- **-name** — поиск файлов по имени
- **-iname -** поиск без учета регистра по имени
- **-perm** — поиск файлов в Linux по режиму доступа
- **-user** — поиск файлов по владельцу
- **-group** — поиск по группе
- **-mtime** — поиск по времени модификации файла
- **-atime** — поиск файлов по дате последнего чтения
- **-nogroup** — поиск файлов, не принадлежащих ни одной группе
- **-nouser** — поиск файлов без владельцев
- **-newer** — найти файлы новее чем указанный
- **-size** — поиск файлов в Linux по их размеру

Если указано несколько критериев, то все критерии должны совпадать, так как команда поиска автоматически принимает логические условия «и» между критериями (что можно явно указать с помощью **-a** между критериями). Например, папка «options» также должна принадлежать sysadmin для поиска, чтобы получить результат:

```
find / -user sysadmin -a -name options
```

Будьте осторожны, чтобы перед скобками ставить обратную косую черту или использовать одинарные кавычки вокруг них, чтобы оболочка не пыталась интерпретировать их как специальные символы. Также, как упоминалось выше, используйте кавычки вокруг любых шаблонов, которые должна интерпретировать команда find вместо оболочки, как показано в следующем примере:

```
find . -iname 'desk*' -o \( -name Downloads -a -user sysadmin \)
```

В обычном тексте команду **find** просят найти файлы в текущем каталоге . с не чувствительным к регистру именем, начинающимся с настольных файлов ИЛИ с точным именем **opt**, принадлежащие **root**.

По умолчанию команда **find** просто печатает имена найденных файлов. Чтобы сгенерировать вывод, аналогичный команде **ls -l,** вы можете указать опцию **-ls**:

```
find / -iname 'desk*' -o \( -name opt -a -user root \)
```

```
[root@ISP ~]# find / -iname 'desk*' -o \( -name opt -a -user root \)
/etc/opt
/var/opt
/usr/lib/tuned/desktop
/usr/share/desktop-directories
/usr/share/gnome-background-properties/desktop-backgrounds-default.xml
/usr/share/kde4/apps/desktoptheme
/opt
```

-o – ИЛИ (OR)

-a – И (AND)

Чтобы вывод был точно таким же, как вывод команды **ls -l**, используйте параметр **-exec** для выполнения **ls -l** для каждого найденного файла. Например:

```
find -name 'passwd' -exec ls -l {} \;
```

```
[root@ISP ~]# find / -iname 'passwd' -exec ls -l {} \;
total 0
-r--r--r--. 1 root root 0 Apr 30 01:47 index
dr-xr-xr-x. 2 root root 0 Apr 30 01:47 perms
-r--r--r--. 1 root root 0 Apr 30 01:47 /sys/fs/selinux/class/passwd/perms/passwd
-rw-r--r--. 1 root root 888 Jan 22 23:45 /etc/passwd
-rw-r--r--. 1 root root 188 Jun 10  2014 /etc/pam.d/passwd
-rwsr-xr-x. 1 root root 27832 Jun 10  2014 /usr/bin/passwd
```

В предыдущем примере команда **find** выполняет команду **ls -l** для каждого найденного файла. Пара фигурных скобок **{}** является заполнителем для имени каждого найденного файла; обратите внимание, что между ними нет места . **\;** это точка с запятой, которая добавляется между каждой командой, так что несколько команд могут выполняться последовательно.

Чтобы команда **find** подтвердила выполнение команды для каждого найденного файла, используйте опцию действия **-ok** вместо **-exec**:

```
cd ~
touch one two three
find . -mmin -2 -ok rm {} \;
```

```
[root@ISP ~]# find . -mmin -2 -ok rm {} \;
< rm ... . > ? y
rm: cannot remove '.': Is a directory
< rm ... ./one > ? y
< rm ... ./two > ? y
< rm ... ./three > ? y
```

Команда whereis

Операционной системе доступно множество команд, и иногда полезно знать, где они находятся. Команду whereis можно использовать для поиска в вашем PATH двоичного файла. Он также способен искать страницу руководства и исходные файлы для любой заданной команды.

Получим значение переменной PATH:

```
echo $PATH
```

Чтобы узнать, где находится команда grep, используйте whereis без каких-либо опций:

```
whereis grep
grep: /bin/grep /usr/share/man/man1/grep.1.gz
```

Из выходных данных команды whereis команда grep находится в каталоге / bin / grep. Но есть также путь, указанный для справочной страницы grep по адресу /usr/share/man/man1/grep.1.gz. Без опции **whereis** предоставляет обе части информации.

Чтобы просмотреть расположение двоичного файла отдельно от страницы руководства, используйте параметры **-b** и **-m** соответственно:

```
whereis -b grep
grep: /bin/grep
whereis -m grep
grep: /usr/share/man/man1/grep.1.gz
```

Опция **-s** может использоваться для поиска исходного кода, который был установлен для данной команды. В случае с grep в данный момент установлен только двоичный файл:

```
whereis -s grep
```

grep:

Обратите внимание, что результаты не возвращаются, поскольку исходный код для **grep** не установлен.

Опция **-u** может использоваться для определения команд, у которых нет записи для запрошенного атрибута. Чтобы узнать, какие команды в каталоге **/bin** не имеют справочных страниц, используйте следующую команду:

```
cd /bin
whereis -m -u *
kmod:
plymouth:
plymouth-upstart-bridge:
running-in-container:
```

Команда which

Иногда команда whereis возвращает более одного результата для команды. В этом случае администратор хотел бы знать, какая команда используется. Команда bash является примером, который обычно дает два результата:

```
whereis -b bash
bash: /bin/bash /etc/bash.bashrc
```

Чтобы узнать, какой результат bash является настоящей командой, используйте команду which:

```
which bash
/bin/bash
```

Опция **-a** может использоваться для поиска нескольких исполняемых файлов. Было бы полезно узнать, был ли исполняемый скрипт вставлен злонамеренно для переопределения существующей команды.

```
which -a ping
/bin/ping
```

Используя это, администратор может быть уверен, что единственный исполняемый файл с именем **ping** находится в каталоге **/bin**.

Команда type

Команда **type** может использоваться для определения информации о различных командах. Некоторые команды происходят из определенного файла:

```
type which
which is /usr/bin/which
```

Этот вывод будет аналогичен выводу команды which:

```
which which
/usr/bin/which
```

Команда type также может идентифицировать команды, встроенные в оболочку bash (или другую):

```
type echo
echo is a shell builtin
```

Этот вывод значительно отличается от вывода команды which:

```
which echo
/bin/echo
```

Используя опцию -a, команда type также может указать путь к другой команде:

```
type -a echo
echo is a shell builtin
echo is /bin/echo
```

Команда type также может идентифицировать псевдонимы для других команд:

```
type ll
ll is aliased to `ls -alF'
type ls
ls is aliased to `ls --color=auto'
```

Выходные данные этих команд указывают, что ll является псевдонимом для ls -alF, и даже ls является псевдонимом для ls --color = auto. Опять же, вывод значительно отличается от команды which:

```
which ll
which ls
/bin/ls
```

Команда type поддерживает другие параметры и может искать несколько команд одновременно. Чтобы отобразить только одно слово, описывающее echo, ll и какие команды, используйте параметр -t:

```
type -t echo ll which
builtin
alias
file
```

Иерархическая файловая система

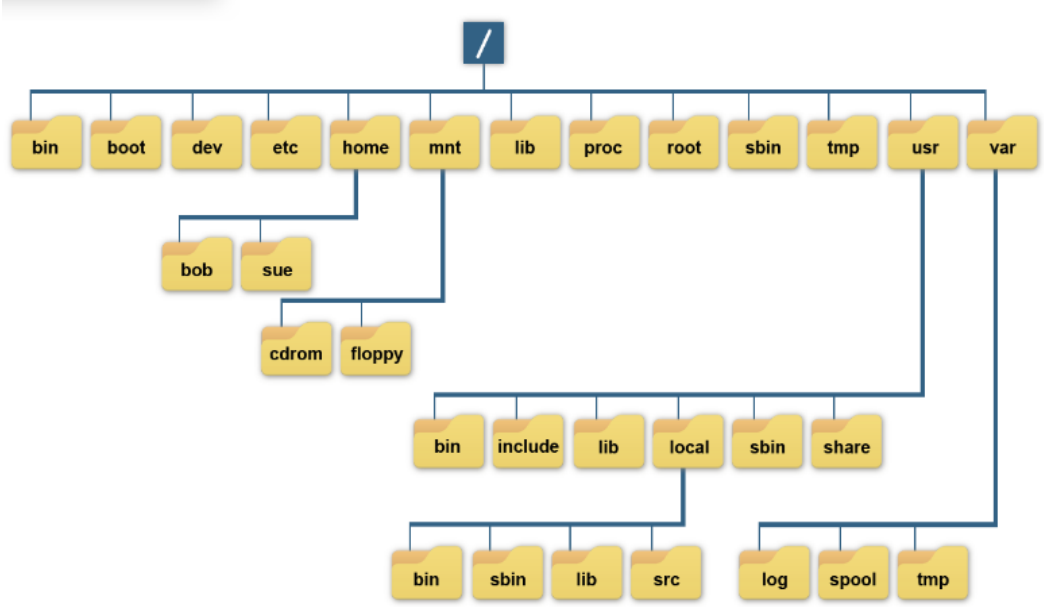
Лицензирование с открытым исходным кодом для многих компонентов Linux позволяет любому создавать собственный дистрибутив. Большинство людей начинают с известного дистрибутива, вносят изменения и выпускают «Форк» оригинала. Иллюстрацию, содержащую многие дистрибутивы и их происхождение, можно найти по адресу http://futurist.se/gldt/wp-content/uploads/12.02/gldt1202.svg.

Поскольку существует так много дистрибутивов Linux, можно было бы ожидать, что разные люди будут менять имена файлов и папок, что в конечном итоге сделает их несовместимыми. Это делает базовое соглашение, необходимое для именования и расположения важных системных файлов и каталогов.

Стандарт иерархии файловой системы (FHS) - это соглашение о стандартизации имен и местоположений каталогов и их содержимого для использования в большинстве файловых систем Linux. Это помогает узнать, какие каталоги ожидать и что в них найти. Что еще более важно, это позволяет программистам писать программы, которые смогут работать в самых разных системах, соответствующих этому стандарту.

Во время разработки первой серии этого стандарта с 1994 по 1995 год он был известен как Стандарт файловой системы (FSSTND). Когда вторая серия была запущена в 1997 году, она была переименована в Стандарт иерархии файловой системы (FHS). Окончательная версия 2.3 этой второй серии этого стандарта FHS была опубликована в 2004 году по адресу http://refspecs.linuxfoundation.org/fhs.shtml. В 2011 году черновая версия третьей серии этого стандарта была опубликована по адресу http://www.linuxbase.org/betaspecs/fhs/fhs.html.

Файловая структура Linux лучше всего визуализируется в виде дерева с разветвленными каталогами и файлами из корневого каталога /. В то время как фактический стандарт содержит гораздо больше каталогов, чем указано ниже, изображение и таблица выделяют некоторые наиболее важные из них.



Directory	Purpose
/	The root of the primary filesystem hierarchy.
/bin	Contains essential user binary executables.
/boot	Contains the kernel and bootloader files.
/dev	Populated with files representing attached devices.
/etc	Configuration files specific to the host.
/home	Common location for user home directories.
/lib	Essential libraries to support /bin and /sbin executables.
/mnt	Mount point for temporarily mounting a filesystem.
/opt	Optional third party add-on software.
/root	Home directory for the root user.
/sbin	Contains system or administrative binary executables.
/srv	May contain data provided by services of the system.
/tmp	Location for creating temporary files.
/usr	The root of the secondary filesystem hierarchy.
/usr/bin	Contains the majority of the user commands.
/usr/include	Header files for compiling C-based software.
/usr/lib	Shared libraries to support /usr/bin and /usr/sbin.
/usr/local	The root of the third filesystem hierarchy for local software.
/usr/sbin	Non-vital system or administrative executables.
/usr/share	Location for architecturally-independent data files.
/usr/share/dict	Word lists.
/usr/share/doc	Documentation for software packages.
/usr/share/info	Information pages for software packages.
/usr/share/locale	Locale information.
/usr/share/man	Location for man pages.
/usr/share/nls	Native language support files.

Поставщики дистрибутивов Linux продолжают вносить некоторые изменения, хотя новая версия стандарта не была опубликована более десяти лет назад. Два заметных новых дополнения включают каталог / run и каталог / sys. Каталог / run рассматривается для использования в следующих версиях FHS для хранения изменчивых данных, которые изменяются во время выполнения. Ранее предполагалось, что эти данные будут храниться в каталоге / var / run, но из-за недоступности этого каталога во время загрузки эти данные могут разбросаться в других местах, таких как скрытые файлы в каталоге / dev.

Каталог **/sys** в некоторых традиционных системах UNIX использовался для хранения файлов, связанных с ядром. В современных системах Linux каталог **/sys** используется для монтирования псевдофайловой системы sysfs. Эта файловая система используется для экспорта информации об объектах ядра и их отношениях друг с другом. Объекты ядра представлены каталогами, а файлы, которые они содержат, названы для атрибутов этих объектов. Содержимое файлов представляет значение для этого атрибута. Символические ссылки используются для представления отношений между объектами.

Еще одно заметное изменение, которое вносят некоторые дистрибутивы Linux, - это преобразование каталогов **/bin**, **/sbin** и **/lib** в символические ссылки, которые указывают на / usr / bin, /usr /sbin и /usr /lib соответственно. Все пользовательские исполняемые файлы теперь находятся в каталоге / usr / bin, исполняемые файлы администратора теперь находятся в каталоге / usr / sbin, а библиотеки для поддержки всех этих исполняемых файлов теперь находятся в каталоге / usr / lib.

Слияние каталогов /bin, /sbin и /lib с каталогами /usr/bin, /usr/sbin и /usr/lib было несколько спорным. Многим администраторам не по себе с давними подразделениями этих файлов на разные каталоги.

Поскольку способ загрузки UNIX, каталоги /bin, /sbin и /lib должны были быть частью корневой файловой системы, поскольку они содержат критические загрузочные исполняемые файлы. Некоторые разработчики утверждают, что причина их разделения больше не действительна. В ранних версиях UNIX разработчики имели две файловые системы размером около 1,5 МБ каждая на двух отдельных дисках для корневой файловой системы и файловой системы / usr. Когда корневая файловая система начала заполняться, разработчики решили переместить некоторые исполняемые файлы, которые были в каталогах / bin и / sbin, которые не были необходимы для загрузки системы, в соответствующие каталоги / usr / bin и / usr / sbin. (в отдельной файловой системе /usr).

По соображениям безопасности FHS разделяет файлы на основе 2 критериев:

Shareable / Unshareable

Совместно используемые файлы могут храниться на одном хосте и использоваться на других. Например, /var/www часто используется в качестве корневого каталога веб-сервера, который обменивается файлами с другими хостами. Другой пример - домашние каталоги пользователей.

Неразделимые файлы не должны быть разделены между хостами. К ним относятся состояния процессов в каталоге /**var/run** и **/boot.**

Переменная / Статическая

Статические файлы, как правило, не меняются, включая библиотечные файлы и страницы документации. Примером являются информационные страницы, расположенные в **/usr/share /info.**

Переменные файлы обычно изменяются во время выполнения программ. Каталог **/var/run** содержит файлы, которые могут быть как переменными, так и недоступными.

В таблице ниже приведены основные различия между типами файлов:

	Shareable	Unshareable
Static	/usr	/etc
	/opt	/boot
Variable	/var/mail	/var/run
	/var/spool/news	/var/lock