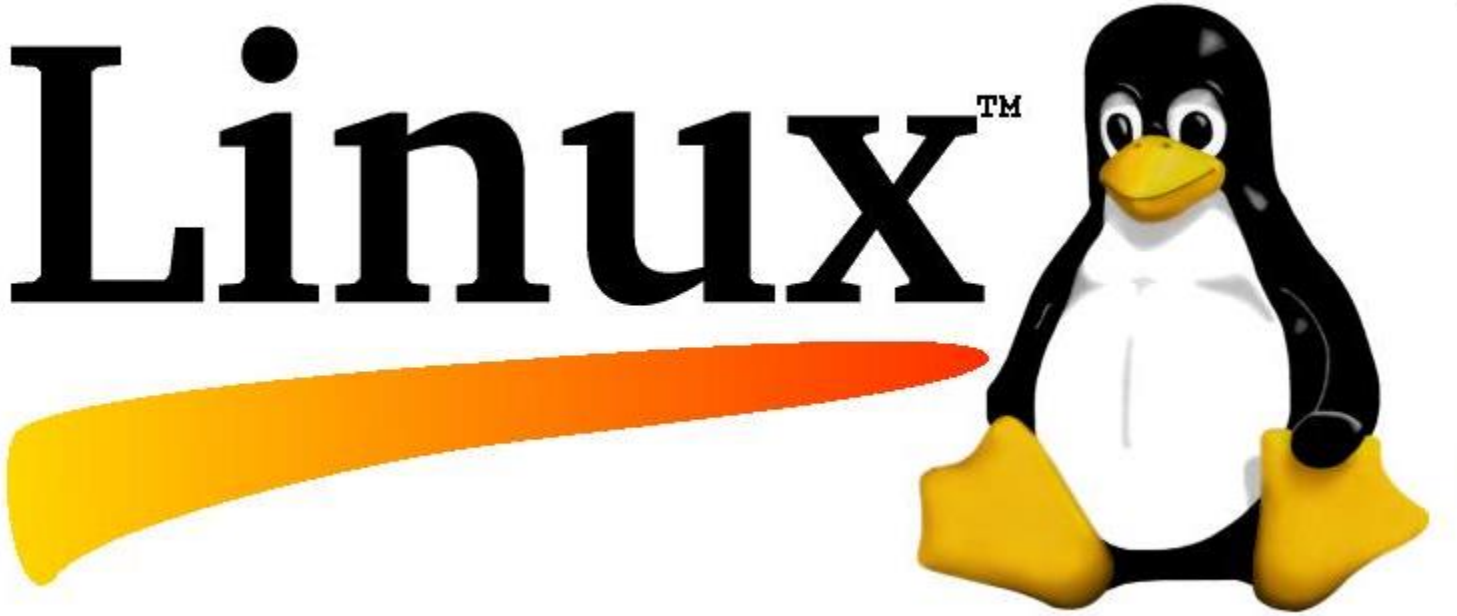

Введение. История. Основные файлы и папки Unix.

UNIX семейство переносимых, многозадачных и многопользовательских операционных систем.

Идеи, заложенные в основу UNIX, оказали огромное влияние на развитие компьютерных операционных систем. В настоящее время UNIX - системы признаны одними из самых исторически важных ОС.

Операционная система linux

Linux – это современная версия UNIX для ПК и рабочих станций. Она была разработана в начале 90-х годов студентом хельсинкского университета Линусом и названа созвучно его имени.

Сейчас Linux реализуется практически для всех типов процессоров и ПК на их основе.

Linux обеспечивает полный набор протоколов TCP/IP для сетевой работы. Достаточно часто на компьютерах, работающих под управлением Linux, реализуют сервера для защиты локальных сетей при работе в Интернет, почтовые серверы, сервер DHCP.

Основные характеристики ОС UNIX

ОС UNIX имеет следующие основные характеристики:

- *вытесняющая многозадачность* на основе процессов, работающих в изолированных адресных пространствах в виртуальной памяти;
- поддержка одновременной работы многих пользователей;
- поддержка асинхронных процессов;
- иерархическая файловая система;
- поддержка независимых от устройств операций ввода-вывода (через специальные файлы устройств);
- стандартный интерфейс для программ (программные каналы, IPC) и пользователей (командный интерпретатор, не входящий в ядро ОС);
- встроенные средства учета использования системы.

Введение в bash

После входа в систему вас приветствует приглашение, которое выглядит примерно так:

```
$
```

На практике приглашение, которое вы видите, может немного отличаться. Например, оно может содержать имя хоста, имя текущей рабочей директории, или все вместе. Не зависимо от того, как выглядит ваше приглашение, есть одна несомненная вещь: программа, которая выводит это приглашение, называется оболочка интерпретатора команд (от англ. shell — оболочка, она же командная строка или терминал — прим. пер.), и, вероятнее всего, вашей командной оболочкой будет 'bash'.
Вы можете убедиться, что используете bash, набрав:

```
$ echo $SHELL
/bin/bash
```

Если строчка выше выдает ошибку, или ответ не соответствует, возможно, что вы запустили другую оболочку.

О bash

Bash- это акроним от Bourne-again-shell, от англ. «ещё-одна-командная-оболочка-Борна» или «рождённая-вновь-командная оболочка» и является оболочкой по умолчанию для большинства Linux-систем. Задача оболочки получать от вас команды, через которые вы взаимодействуете с Linux-системой. После того, как вы закончили ввод команд, вы можете выйти из оболочки (exit) или закончить сеанс (logout), в этом случае вы увидите приглашение входа в систему.
Кстати, вы также можете выйти из оболочки bash нажав control-D в приглашении.

Использование «cd»

В приглашении введите следующую команду (без $):

$ cd /

Вы только что сообщили bash, что хотите работать в директории /, также известной, как корневая; все директории в системе имеют форму дерева, и / является его вершиной, т.е. корнем (в информатике деревья растут наоборот, корень вверху, а ветки спускаются вниз — прим. пер.). Cd устанавливает директорию, в которой вы в данный момент работаете, также известную как «текущая рабочая директория».

Пути

Чтобы узнать текущую рабочую директорию в bash нужно набрать:

$ pwd
/

В примере с cd, аргумент / называется путь. Он сообщает cd куда мы хотим отправиться. В частности, аргумент / это абсолютный путь, что значит, что он задает расположение относительно корня дерева файловой системы.

Абсолютные пути

Ниже несколько из них:

/dev
/usr
/usr/bin
/usr/local/bin

У всех абсолютных путей есть одна общая черта, они начинаются с /. Указывая, допустим, /usr/local/bin в качестве аргумента для cd, мы сообщаем, что хотим попасть в / директорию, затем в usr директорию внутри нее, и так далее в local и bin, вниз по дереву.
Абсолютные пути всегда отсчитываются начиная от / сперва.

Относительные пути

Другой тип пути называется «относительный путь». bash, cd, и другие команды всегда интерпретируют их относительно текущей директории.
Относительные пути НИКОГДА не начинаются с /. Так, если мы сначала переместимся в /usr:

$ cd /usr

То, затем мы можем использовать относительный путь local/bin, чтобы попасть в директорию /usr/local/bin:

$ cd local/bin
$ pwd
/usr/local/bin

Использование

Относительные пути могут также содержать одну или более "..« директории. Директория ".." специальная; она указывает на родительскую директорию. Так, продолжая с примера выше:

$ pwd
/usr/local/bin
$ cd ..
$ pwd
/usr/local

Как видно, наша текущая директория теперь /usr/local. Мы смогли переместиться «назад» на одну директорию относительно текущей, где были до того.
Кроме того, мы также можем использовать ".." в существующем относительном пути, позволяющем нам переместиться в директорию «рядом» с той, в которой находимся:

```
$ pwd
/usr/local
$ cd ../share
$ pwd
/usr/share
```

cd и домашняя директория

Если бы мы хотели переместиться в нашу домашнюю директорию, то могли бы набрать:

```
$ cd
```

Без каких либо аргументов cd переместит в вашу домашнюю директорию, которая будет /root для суперпользователя, или обычно /home/username (где username — имя пользователя в системе) для любого другого пользователя. Но, что если мы хотим указать файл в нашей домашней директории? Может быть мы хотим передать путь к файлу в качестве аргумента нашей программе myprog.

Если файл расположен в нашей домашней директории, мы можем набрать:

$ ./myprog /home/drobbins/myfile.txt

Однако, использования абсолютного пути вроде этого, не всегда удобно. Мы можем использовать символ ~ (тильда), чтобы проделать то же самое:

$ ./myprog ~/myfile.txt

Другие домашние директории пользователей

Bash воспримет одиночную ~ как указатель на вашу домашнюю директорию, но вы также можете использовать её для указания на домашние директории других пользователей. Например, если мы хотели сослаться на файл под названием fredsfile.txt в домашней директории пользователя fred, то могли бы набрать:

$ ./myprog ~fred/fredsfile.txt

Использование команд Linux

Знакомство с ls

Пройдемся по команде **ls**. Набрав **ls** вы получите список содержимого текущей рабочей директории:

```
$ cd /usr
$ ls
X11R6      doc             i686pclinuxgnu      lib             man
bin        gentoox86       include             libexec         portage
distfiles  i686linux       info local          portage.old     src
```

Указав опцию -a, вы можете увидеть полный список, включая скрытые файлы и директории, начинающиеся с ".". Как видно в следующем примере, ls -a выводит также особые связывающие директории "." и "..":

```
$ ls a
.    bin        gentoox86       include    libexec    portage         share
     tmp
..   distfiles  i686linux       info local             portage.old    src
X11R6           doc             i686pclinuxgnu         lib            man
```

Развернутые списки директорий

Вы также можете задать одну и более директорий или файлов в командной строке с ls. Если вы укажите файл, то ls покажет вам только этот файл. А если зададите директорию, то ls выдаст ее содержимое. Опция -l очень удобна, когда необходимо посмотреть права доступа, владельца, время последнего изменения и размер в списке содержимого директории.

Смотрим на директории

Иногда вы захотите взглянуть на директорию, а не внутрь нее. В этом случае вы можете указать опцию -d, которая скажет ls рассматривать любую директорию, как внутреннюю:

```
$ ls dl /usr /usr/bin /usr/X11R6/bin ../share
drwxrxrx       4 root      root       96 Dec 18 18:17 ../share
drwxrxrx       17 root     root        576 Dec 24 09:03 /usr
drwxrxrx       2 root      root       3192 Dec 26 12:52 /usr/X11R6/bin
drwxrxrx    2 root     root          14576 Dec 27 08:56 /usr/bin
```

Рекурсивный и инодный списки

Вы можете использовать -d чтобы смотреть на директорию, но также можно использовать -R для противоположного: не только лишь глянуть внутрь директории, но и рекурсивно посмотреть все директории с файлами внутри нее! Опция -i может использоваться для отображения числа инодов для объектов в списке файловой системы:

```
$ ls i /usr
1409 X11R6        314258 i686linux          43090 libexec  13394 sbin
1417 bin          1513 i686pclinuxgnu       5120 local     13408
share
8316 distfiles    1517 include              776 man        23779 src
43 doc            1386 info                 93892 portage  36737 ssl
70744 gentoox86   1585 lib                  5132 portage.old   784 tmp
```

Понятие инода

Каждому объекту файловой системы назначен уникальный индекс, называемый номером инода. Рассмотрим например ссылки "." и "..", которые появляются в каждой директории. Чтобы полностью понять, чем на самом деле является директория "..", мы сперва взглянем на номер инода у /usr/local:

$ ls id /usr/local
5120 /usr/local

У директории /usr/local номер инода равен 5120. А теперь посмотрим номер инода у /usr/local/bin/..:

$ ls id /usr/local/bin/..
5120 /usr/local/bin/..

Как видно, директория /usr/local/bin/.. имеет такой же номер, как у /usr/local! В прошлом мы полагали, что /usr/local сама является директорией. Теперь же, мы обнаружили, что фактически директория — это инод с номером 5120, и нашли, по меньшей мере, два элемента (называемых «ссылками»), которые указывают на данный инод. И /usr/local, и /usr/local/bin/.. — ссылки на 5120-ый инод. Хотя этот инод и существует только в одном месте на диске, тем не менее на него может быть множество ссылок.
На самом деле, мы даже можем увидеть общее количество ссылок ведущих на этот, 5120 инод, используя команду ls -dl:

$ ls dl /usr/local
drwxrxrx 8 root root 240 Dec 22 20:57 /usr/local

mkdir

Пройдемся по команде mkdir, которая используется для создания новых директорий. Следующий пример создает три новых директории, tic, tac, и toe, все внутри /tmp:

```
$ cd /tmp
$ mkdir tic tac toe
```

По умолчанию, команда mkdir не создает для вас родительские директории; весь путь вплоть до последнего (создаваемого) элемента должен существовать. Так, если вы захотите создать вложенные директории won/der/ful, вам придется выполнить три отдельные команды mkdir:

```
$ mkdir won/der/ful
mkdir: cannot create directory `won/der/ful': No such file or
directory
$ mkdir won
$ mkdir won/der
$ mkdir won/der/ful
```

Однако, у mkdir есть очень удобная опция -p, которая говорит mkdir создавать любые отсутствующие родительские директории, как можете увидеть тут:

$ mkdir p easy/as/pie

Чтобы узнать больше о команде mkdir наберите man mkdir и прочитайте инструкцию. Это же касается почти всех команд, рассмотренных здесь (например man ls), исключая cd, которая встроена в bash.

touch

Рассмотрим команды cp и mv, используемые для копирования, переименования и перемещения файлов и директорий. Но начнем обзор воспользовавшись командой touch, чтобы создать файл в /tmp:

$ cd /tmp
$ touch copyme

Команда touch обновляет «mtime» (время последней модификации) файла, если тот существует. Если файл не существует, то новый, пустой файл будет создан. Сейчас у вас должен быть файл /tmp/copyme с нулевым размером.

echo

Теперь, когда файл существует, давайте добавим немного данных в него. Можно сделать это с помощью команды echo, которая принимает аргументы и печатает их на стандартный вывод. Сперва, команда echo сама по себе:

```
$ echo "firstfile"
firstfile
```

А сейчас, та же команда echo, но с перенаправлением вывода:

```
$ echo "firstfile" > copyme
```

Знак «больше» сообщает оболочке записывать вывод echo в файл по имени copyme. Этот файл будет создан, если не существовал, или перезаписан, если существует. Набрав ls -l, увидим, что файл copyme имеет размер в 10 байт, так как содержит слово firstfile и символ новой строки:

```
$ ls l copyme
rwrr    1 root     root          10 Dec 28 14:13 copyme
```

cat и cp

Чтобы вывести содержимое файла на терминал, используйте команду cat:

$ cat copyme
firstfile

Сейчас, мы можем воспользоваться основным вызовом команды cp для создания файла copiedme из оригинального copyme:

$ cp copyme copiedme

Ниже проверим, что это действительно разные файлы; у них отличаются номера инодов:

$ ls i copyme copiedme
648284 copiedme 650704 copyme

mv

А сейчас давайте воспользуемся командой mv для переименования copiedme в movedme. Номер иноды останется прежний; однако, имя файла, указывающее на инод, изменится.

```
$ mv copiedme movedme
$ ls i movedme
648284 movedme
```

Номер инода у перемещаемого файла остается прежним до тех пор, пока файл назначения находится в той же файловой системе, что и исходный файл. **mv**, помимо возможности переименовать файлы, позволяет перемещать один или более файлов в другое место в иерархии директорий. Например, чтобы переместить /var/tmp/myfile в директорию /home/user, я наберу:

```
$ mv /var/tmp/myfile /home/user
```

После этого myfile будет перемещен в /home/user/myfile. И если /home/uset располагается в другой файловой системе, нежели /var/tmp, команда mv скопирует myfile в новую файловую систему и удалит его из старой.

Когда myfile перемещается между файловыми системами, то myfile на новом месте получает новый номер инода. Это все потому, что у каждой файловой системы свой независимый набор номеров инодов.
Мы также можем воспользоваться mv для перемещения нескольких файлов в одну директорию. К примеру, чтобы переместить myfile1t и myarticle3 в /home/user, потребуется набрать:

$ mv /var/tmp/myfile1 /var/tmp/myarticle3 /home/user

Создание ссылок и удаление файлов

Жесткие ссылки

Мы уже упоминали термин «ссылка», когда рассказывали о взаимоотношениях между директориями (их именами) и инодами (индексным номерами, лежащими в основе файловой системы, которых мы не замечаем). Вообще в Linux существует два типа ссылок. Тип, о котором мы уже говорили ранее, называется «жесткие ссылки». Каждый инод может иметь произвольное число жестких ссылок. Когда уничтожается последняя жесткая ссылка, и не одна программа не держит файл открытым, то Linux автоматически удаляет его.

Новые жесткие ссылки можно создать воспользовавшись командой ln:

```
$ cd /tmp
$ touch firstlink
$ ln firstlink secondlink
$ ls i firstlink secondlink
15782 firstlink    15782 secondlink
```

Как видите, жесткие ссылки работают на уровне инодов, для указания конкретного файла. В Linux системах, для жестких ссылок есть несколько ограничений. В частности, можно создавать жесткие ссылки только на файлы, не на директории. Хотя "." и ".." являются созданными системой жесткими ссылками на директории, вам (даже от имени пользователя «root») не разрешается создавать любые свои собственные. Второе ограничение жестких ссылок состоит в том, что нельзя связать ими несколько файловых систем. Это значит, что у вас не получится создать жесткую ссылку с /usr/bin/bash на /bin/bash и если ваши директории / и /usr находятся в разных файловых системах (разделах).

Символьные ссылки

В практике, символьные ссылки (или символические, иногда «симлинки» — от англ.) используются гораздо чаще, чем жесткие. Симлинки — это файлы особого типа, которые ссылаются на другие файлы по имени, а не прямо по номеру инода. Они не спасают файлы от удаления; если файл, на который указывает ссылка, исчезает, то симлинк перестает работать, ломается.
Символические ссылки можно создать передав для ln опцию -s.

$ ln s secondlink thirdlink
$ ls l firstlink secondlink thirdlink
rwrwr 2 agriffis agriffis 0 Dec 31 19:08 firstlink
rwrwr 2 agriffis agriffis 0 Dec 31 19:08 secondlink
lrwxrwxrwx 1 agriffis agriffis 10 Dec 31 19:39 thirdlink>secondlink

В выводе ls -l символьные ссылки можно отличить тремя способами. Во-первых, обратите внимание на символ l в первой колонке. Во-вторых, размер символической ссылки равен количеству символов в ней (secondlink в нашем случае). В-третьих, последняя колонка в выводе показывает куда ведет ссылка с помощью интуитивного обозначения "->".

rm

Итак, мы знаем как использовать cp, mv и ln, настало время узнать о том, как можно удалять объекты из файловой системы. Обычно это делается с помощью команды rm. Чтобы удалить файлы, просто укажите их в командной строке:

```
$ cd /tmp
$ touch file1 file2
$ ls l file1 file2
rwrr    1 root     root          0 Jan  1 16:41 file1
rwrr    1 root     root          0 Jan  1 16:41 file2

$ rm file1 file2
$ ls l file1 file2
ls: file1: No such file or directory
ls: file2: No such file or directory
```

Имейте ввиду, что под Linux, однажды удаленный файл, обычно исчезает на века. Поэтому многие начинающие системные администраторы используют опцию -i, когда удаляют файлы. Опция -i сообщает rm удалять файлы в интерактивном режиме — это значит спрашивать перед удалением любого файла.

Например:

```
$ rm i file1 file2
rm: remove regular empty file `file1'? y
rm: remove regular empty file `file2'? y
```

В примере выше команда rm запрашивает подтверждение на удаление каждого из указанных файлов. В случае согласия, я должен был вводить «y» и нажать enter, дважды. Если бы я ввел «n», то файл бы остался цел. Или, если я сделал что-нибудь не так, я мог бы нажать Control-C и сбросить выполнение команды rm -i целиком — всяко до того, как это могло нанести какой-нибудь ущерб моей системе.

Может быть полезным добавить при помощи вашего любимого текстового редактора следующую строку в ваш файл ~/.bashrc, и затем выйти (logout) и войти (login) в систему вновь. После этого, всякий раз, когда вы наберете rm, оболочка bash преобразует ее автоматически в команду rm -i. Таким образом, rm будет всегда работать в интерактивном режиме:

```
alias rm="rm i"
```

rmdir

Для удаления директорий у вас имеется два варианта. Вы можете удалить все объекты внутри директории и затем воспользоваться rmdir для удаления самой директории:

```
$ mkdir mydir
$ touch mydir/file1
$ rm mydir/file1
$ rmdir mydir
```

Самый лучший способ удалить директорию состоит в использовании опций «рекурсивного принуждения» (recursive force) команды rm, чтобы приказать ей удалять указанную директорию, также как и объекты содержащиеся внутри:

```
$ rm rf mydir
```

Обычно, rm -rf является наиболее предпочтительным методом для удаления древа директорий.

FHS и поиск файлов

Стандарт иерархии файловой системы

Стандарт иерархии файловой системы (Filesystem Hierarchy Standard или сокр. FHS) — это документ который определяет схему директорий в Linux-системах. FHS разработан чтобы представить общую схему для упрощения независимой от дистрибутива разработки программного обеспечения, поскольку так все необходимое располагается одинаково в большинстве дистрибутивов. FHS определяет следующее дерево директорий (взято непосредственно из спецификации):

- / (корневая директория)
- /boot (статичные файлы загрузчика)
- /dev (файлы устройств)
- /etc (специфические для хоста конфигурационные файлы)
- /lib (основные разделяемые библиотеки и модули ядра)
- /mnt (точка монтирования для временных нужд)

- /mnt (точка монтирования для временных нужд)
  - /pt (дополнительные пакеты ПО)
  - /sbin (основные системные программы)
  - /tmp (временные файлы)
  - /usr (вторичная иерархия)
  - /var (изменяемые данные)

Две независимые классификации в FHS

Спецификация FHS основывается на идее существования двух независимых классификаций файлов: разделяемые и неразделяемые, а также изменяемые и статичные. Разделяемые данные могут распределятся на несколько хостов; неразделяемые специфичны для конкретного хоста (как, например, конфигурационные файлы). Изменяемые данные могут изменяться; статичные не изменяются (за исключением установки и обслуживания системы).

Нижеследующая табличка резюмирует четыре возможные комбинации, с примерами директорий, которые попадают в данные категории. Опять же, эта таблица прямо из спецификации:

	разделяемые	неразделяемые
статичные	/usr	/etc
	/opt	/boot
изменяемые	/var/mail	/var/run
	/var/spool/news	/var/lock

Вторичная иерархия в /usr

Внутри /usr вы обнаружите вторичную иерархию, которая выглядит очень похоже на корневую файловую систему. Для /usr не критично существование во время включения машины, она может быть общим сетевым ресурсом (разделяема) или примонтирована с CD-ROM (статична).

Большинство конфигурация Linux не используют «разделяемость» /usr, но ценно понимать полезность отличия между основной иерархией в корневой директории и вторичной иерархией в /usr.

Поиск файлов

Linux-системы зачастую содержат сотни тысяч файлов. Вероятно, что временами вам потребуется помощь для нахождения какого-либо файла. Для этого в Linux есть несколько разнообразных средств.

PATH

Когда вы запускаете программу из командной строки, bash начинает просматривать список директорий в поисках программы которую вы указали. Например, когда вы вводите ls, bash в действительности не знает, что программа ls находится в /usr/bin. Вместо этого, он ссылается на переменную окружения называемую PATH, которая содержит список директорий разделенных двоеточием. Мы можем проверить значение PATH:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin.
```

С таким значением PATH (у вас оно может быть другим) bash сначала проверит директорию /usr/local/bin, затем /usr/bin в поисках программы ls.
Скорее всего, ls находится в /usr/bin, тогда на этой директории bash прекратит поиск.

Изменение PATH

Вы можете расширять переменную PATH, присваивая ей новое значение в командой строке:

```
$ PATH=$PATH:~/bin
$ echo $PATH
/
usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin:/home/agri
ffis/bin
```

Вы также можете удалять элементы из PATH, хотя это не так просто, поскольку вы не можете ссылаться в команде на существующий $PATH.
Лучший вариант — это просто заново указать в PATH то, что вам нужно:

```
$ PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:~/bin
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/agriffis/bin
```

Чтобы сделать ваши изменения PATH доступными для процессов, которые будут запускаться в командной оболочке, необходимо «экспортировать» их используя команду export:

```
$  export PATH
```

О команде «which»

Вы можете проверить, есть ли конкретная программа в вашем PATH используя which. В следующем примере мы видим, в каталогах PATH нашей системы, программы с названием sense нет:

```
$ which sense
which: no sense in (/usr/local/bin:/usr/bin:/bin:/usr/sbin:
/sbin:/usr/X11R6/bin)
```

В этом примере, ls успешно находится:

```
$ which ls
/usr/bin/ls
```

```
which a
```

Наконец, вы должны знать о флаге -a, который укажет which показать вам все экземпляры программы в PATH:

```
$ which a ls
/usr/bin/ls
/bin/ls
```

whereis

Если вам необходимо больше информации о программе, чем просто ее расположение, вы можете воспользоваться командой whereis:

$ whereis ls
ls: /bin/ls /usr/bin/ls /usr/share/man/man1/ls.1.gz

Здесь мы видим что ls находится в двух каталогах с общими исполняемыми файлами, /bin и /usr/bin. Кроме того, нам сообщили что есть документация, которая находится в /usr/share/man. Это man-страница которую вы увидите, если введете man ls.

Программа whereis может использоваться для поиска расположения исходников и нестандартного поиска (имеется ввиду возможность искать файлы для которых отсутствуют маны, исходники или бинарники. Также ей можно указать альтернативные пути для поиска. Обратитесь к man-странице для получения дополнительной информации.

find

Команда find это другой удобный инструмент в вашем арсенале. Используя find вы не ограничены лишь поиском программ; вы можете искать любые типы файлов, используя различные критерии поиска. Например, поищем в директории /usr/share/doc, файл который называется README:

$ find /usr/share/doc name README
/usr/share/doc/ion20010523/README
/usr/share/doc/bind9.1.3r6/dhcpdynamicdnsexamples/README
/usr/share/doc/sane1.0.5/README

find и шаблоны

Вы можете использовать glob-шаблоны для аргументов -name, при условии что вы экранируете их кавычками или обратным слешем (таким образом они будут переданы команде в нетронутом виде, иначе они сначала будут развернуты bash'ем и уже после переданы команде). Давайте поищем все файлы README с расширением:

```
$ find /usr/share/doc name README\*
/usr/share/doc/iproute22.4.7/README.gz
/usr/share/doc/iproute22.4.7/README.iproute2+tc.gz
/usr/share/doc/iproute22.4.7/README.decnet.gz
/usr/share/doc/iproute22.4.7/examples/diffserv/README.gz
/usr/share/doc/pilotlink0.9.6r2/README.gz
/usr/share/doc/gnomepilotconduits0.8/README.gz
/usr/share/doc/gimp1.2.2/README.i18n.gz
/usr/share/doc/gimp1.2.2/README.win32.gz
/usr/share/doc/gimp1.2.2/README.gz
/usr/share/doc/gimp1.2.2/README.perl.gz
[еще  578 строк  опущено ]
```

Игнорирование регистра в find

Конечно, вы можете игнорировать регистр при поиске:

$ find /usr/share/doc name '[Rr][Ee][Aa][Dd][Mm][Ee]*'

Или, намного проще:

$ find /usr/share/doc iname readme*

Как видно, для поиска без учета регистра можно использовать опцию -iname.

find и регулярные выражения

Если вы знакомы с регулярными выражениями, вы можетеиспользовать опцию -regex для поиска файлов с именами соответствующими шаблону. А также опцию похожую на -iname, которая называется -iregex и заставляет find игнорировать регистр в шаблоне. Пример:

$ find /etc iregex '.*xt.*'
/etc/X11/xkb/types/extra
/etc/X11/xkb/semantics/xtest
/etc/X11/xkb/compat/xtest
/etc/X11/appdefaults/XTerm
/etc/X11/appdefaults/XTermcolor

Однако в отличии от большинства программ, find требует чтобы регулярное выражение указывалось для всего пути, а не только его части.
По этой причине, стоит в начале и конце шаблона ставить .*; простого использования xt в качестве шаблона будет недостаточно.