Использование SHELL

Определение слова Linux зависит от контекста, в котором оно используется. Под Linux понимается ядро системы, которое является центральным контроллером всего, что происходит на компьютере. Люди, которые говорят, что их компьютер «работает под управлением Linux», обычно ссылаются на ядро и набор инструментов, которые поставляются с ним (так называемый дистрибутив). Если кто-то говорит, что у него «опыт работы с Linux», он, скорее всего, говорит о самих программах. Тем не менее, они могут также говорить о том, чтобы знать, как добавить и разбить новый диск или даже точно настроить ядро. Каждый из этих компонентов будет исследован, чтобы вы точно поняли, какую роль играет каждый из них.

Что насчет UNIX? Изначально UNIX была операционной системой, разработанной в AT & T Bell Labs в 1970-х годах. Он был изменен и разветвлен (то есть люди его модифицировали, и эти модификации послужили основой для других систем), так что теперь существует множество различных вариантов UNIX. Тем не менее, UNIX теперь является торговой маркой и спецификацией, принадлежащей отраслевому консорциуму под названием Open Group. Только программное обеспечение, сертифицированное Open Group, может называться UNIX. Несмотря на принятие всех требований спецификации UNIX, Linux не был сертифицирован, поэтому Linux на самом деле не UNIX! Это просто ... как UNIX.

Ядро

Три основных компонента операционной системы - это ядро, оболочка и файловая система. Ядро операционной системы похоже на авиадиспетчер в аэропорту. Ядро определяет, какая программа получает какие части памяти, запускает и убивает программы и обрабатывает отображение текста на мониторе. Когда приложению необходимо выполнить запись на диск, оно должно попросить операционную систему завершить операцию записи. Если два приложения запрашивают один и тот же ресурс, ядро решает, кто его получит, а в некоторых случаях убивает одно из приложений, чтобы сохранить остальную часть системы.

Ядро также управляет переключением приложений. Компьютер будет иметь небольшое количество процессоров и ограниченный объем памяти. Ядро заботится о выгрузке одной задачи и загрузке новой задачи, если задач больше, чем процессоров. Когда текущая задача выполняется достаточно времени, процессор приостанавливает задачу, чтобы выполнить другую. Это называется упреждающей многозадачностью. Многозадачность означает, что компьютер выполняет несколько задач одновременно, а упреждающий означает, что ядро решает, когда переключать фокус между задачами. С быстрым переключением задач кажется, что компьютер делает много вещей одновременно.

Каждое приложение может думать, что оно имеет большой блок памяти в системе, но это ядро поддерживает эту иллюзию, переназначая меньшие блоки памяти, разделяя блоки памяти с другими приложениями или даже заменяя блоки, которые не были затронуты. на диск.

Приложения

Как и авиадиспетчер, ядро бесполезно без чего-либо для управления. Если ядро - башня, приложения - самолеты. Приложения отправляют запросы ядру и получают ресурсы, такие как память, процессор и диск, взамен. Ядро также абстрагирует сложные детали от приложения. Приложение не знает, находится ли блок диска на твердотельном диске от производителя A, вращающемся металлическом жестком диске от производителя B или даже на общем сетевом ресурсе. Приложения просто следуют интерфейсу прикладного программирования (API) ядра и, в свою очередь, не должны беспокоиться о деталях реализации.

Когда мы, как пользователи, думаем о приложениях, мы склонны думать о текстовых процессорах, веб-браузерах и почтовых клиентах. Ядро не заботится, работает ли оно с чем-то, с чем сталкивается пользователь, с сетевой службой, которая общается с удаленным компьютером, или с внутренней задачей. Итак, из этого мы получаем абстракцию, называемую процессом. Процесс - это всего лишь одна задача, которая загружается и отслеживается ядром. Приложению может даже потребоваться несколько процессов для работы, поэтому ядро позаботится о запуске процессов, их запуске и остановке в соответствии с запросом и раздаче системных ресурсов.

Открытый исходный код

Linux начался в 1991 году как проект хобби Линуса Торвальдса. Он сделал источник свободно доступным, и другие присоединились к этой новой операционной системе. Он был не первой системой, разработанной группой, но, поскольку это был проект, созданный с нуля, первые пользователи имели возможность влиять на направление проекта и следить за тем, чтобы ошибки других UNIX не повторялись.

Программные проекты принимают форму исходного кода, который представляет собой удобочитаемый набор компьютерных инструкций. Исходный код может быть написан на любом из сотен различных языков программирования, Linux просто написан на C, который является языком, который делится историей с оригинальной UNIX.

Исходный код не понимается непосредственно компьютером, поэтому он должен быть скомпилирован в машинные инструкции компилятором. Компилятор собирает все исходные файлы и генерирует что-то, что может быть запущено на компьютере, например ядро Linux.

Исторически сложилось так, что большинство программ было выпущено по лицензии с закрытым исходным кодом, что означает, что вы получаете право использовать машинный код, но не можете видеть исходный код. Часто в лицензии конкретно говорится, что вы не будете пытаться выполнить обратный инжиниринг машинного кода обратно в исходный код, чтобы выяснить, что он делает!

Открытый исходный код ориентирован на источник программного обеспечения. Философия открытого исходного кода заключается в том, что у вас есть право на получение программного обеспечения и его изменение для собственного использования. Linux принял эту философию с большим успехом. Люди взяли источник, внесли изменения и поделились ими с остальной частью группы.

Наряду с этим был проект GNU (GNU, а не UNIX). В то время как GNU (произносится как «ga-noo») создавал свою собственную операционную систему, они были гораздо эффективнее в создании инструментов, которые работают вместе с операционной системой UNIX, таких как компиляторы и пользовательские интерфейсы. Источник был в свободном доступе, Таким образом, Linux смог нацелить свои инструменты и предоставить полную систему. Таким образом, большинство инструментов, являющихся частью системы Linux, происходят из этих инструментов GNU.

Дистрибутивы Linux

Возьмите Linux и инструменты GNU, добавьте еще несколько пользовательских приложений, таких как почтовый клиент, и у вас будет полноценная система Linux. Люди начали собирать все это программное обеспечение в дистрибутив почти сразу, как только Linux стал пригодным для использования. Дистрибутив заботится о настройке хранилища, установке ядра и установке остальной части программного обеспечения. Полнофункциональные дистрибутивы также включают инструменты для управления системой и менеджер пакетов, которые помогут вам добавлять и удалять программное обеспечение после завершения установки.

Как и в UNIX, существует много разных дистрибутивов. В наши дни существуют дистрибутивы, которые ориентированы на работу серверов, настольных компьютеров или даже отраслевых инструментов, таких как дизайн электроники или статистические вычисления. Основные игроки на рынке можно проследить до Red Hat или Debian. Наиболее заметным отличием является менеджер пакетов программного обеспечения, хотя вы найдете и другие различия во всем - от расположения файлов до политических соображений.

Red Hat начинался как простой дистрибутив, в котором появился Red Hat Package Manager (RPM). В конечном итоге разработчик создал компанию, которая пыталась коммерциализировать рабочий стол Linux для бизнеса. Со временем Red Hat начала уделять больше внимания серверным приложениям, таким как веб-сервис и обслуживание файлов, и выпустила Red Hat Enterprise Linux, которая была платной услугой в долгом цикле выпуска. Цикл выпуска диктует, как часто программное обеспечение обновляется. Бизнес может ценить стабильность и хотеть длинные циклы выпуска, любитель или стартап могут хотеть новейшее программное обеспечение и выбрать более короткий цикл выпуска. Чтобы удовлетворить последнюю группу, Red Hat спонсирует проект Fedora, который создает персональный рабочий стол, включающий новейшее программное обеспечение, но при этом основанный на той же основе, что и версия для предприятий.

Поскольку все в Red Hat Enterprise Linux является открытым исходным кодом, появился проект CentOS, который перекомпилировал все пакеты RHEL и раздал их бесплатно. CentOS и

другие подобные ему (такие как Scientific Linux) в значительной степени совместимы с RHEL и интегрируют некоторые более новые программы, но не предлагают платную поддержку, которую делает Red Hat.

Debian - это скорее работа сообщества, и поэтому он также способствует использованию программного обеспечения с открытым исходным кодом и соблюдению стандартов. Debian разработал собственную систему управления пакетами, основанную на формате файлов .deb. В то время как Red Hat не поддерживает платформы Intel и AMD для производных проектов, Debian поддерживает многие из этих платформ напрямую.

Ubuntu - самый популярный дистрибутив Debian. Это создание Canonical, компании, которая была создана для дальнейшего развития Ubuntu и зарабатывания денег путем предоставления поддержки.

Аппаратные платформы

Linux начинался как нечто, что будет работать только на компьютере, подобном Linus: 386 с конкретным контроллером жесткого диска. Диапазон поддержки вырос, так как люди создали поддержку для другого оборудования. В конце концов, Linux начал поддерживать другие чипы, в том числе аппаратное обеспечение, созданное для запуска конкурентоспособных операционных систем!

Типы аппаратных средств выросли от скромного чипа Intel до суперкомпьютеров. Позже были разработаны чипы меньшего размера, поддерживаемые Linux, для встраивания в потребительские устройства, называемые встроенными устройствами. Поддержка Linux стала повсеместной, так что зачастую проще создать оборудование для поддержки Linux, а затем использовать Linux в качестве плацдарма для собственного программного обеспечения, чем для создания собственного аппаратного и программного обеспечения с нуля.

Со временем сотовые телефоны и планшеты начали работать под управлением Linux. Компания, позже купленная Google, предложила платформу Android, которая представляет собой комплект Linux и программное обеспечение, необходимое для работы телефона или планшета. Это означает, что усилия по выводу телефона на рынок значительно меньше. Вместо долгой разработки новой операционной системы компании могут тратить свое время на инновации в программном обеспечении, ориентированном на пользователя. Android сейчас является одним из лидеров на рынке телефонов и планшетов.

Помимо телефонов и планшетов, Linux можно найти во многих потребительских устройствах. Беспроводные маршрутизаторы часто работают под управлением Linux, потому что он имеет богатый набор сетевых функций. TiVo - это потребительский цифровой видеомагнитофон на базе Linux. Хотя в основе этих устройств лежит Linux, конечные пользователи не должны знать об этом. Пользовательское программное обеспечение взаимодействует с пользователем, а Linux обеспечивает стабильную платформу.

Оболочка (Shell)

Операционная система предоставляет по крайней мере одну оболочку или интерфейс; это позволяет вам сказать компьютеру, что делать. Оболочку иногда называют интерпретатором, потому что она принимает команды, которые выдает пользователь, и интерпретирует их в форму, которую ядро затем может выполнить на оборудовании компьютера. Два наиболее распространенных типа оболочек - это графический интерфейс пользователя (GUI) и интерфейс командной строки (CLI).

В Windows® обычно используется оболочка с графическим интерфейсом, в основном с помощью мыши, чтобы указать, что вы хотите сделать. Хотя использование операционной системы таким способом может считаться простым, использование CLI имеет много преимуществ, в том числе:

Повторение команд: в оболочке с графическим интерфейсом нет простого способа повторить предыдущую команду. В CLI есть простой способ повторить (а также изменить) предыдущую команду.

Гибкость команд: оболочка графического интерфейса обеспечивает ограниченную гибкость при выполнении команды. В CLI параметры указываются с помощью команд, чтобы обеспечить гораздо более гибкий и мощный интерфейс.

Ресурсы. Оболочка графического интерфейса обычно использует огромное количество ресурсов (RAM, CPU и т. Д.). Это связано с тем, что для отображения графики требуется много вычислительной мощности и памяти. В отличие от этого CLI использует очень мало системных

ресурсов, что позволяет большему количеству этих ресурсов быть доступными для других программ.

Сценарии: в оболочке с графическим интерфейсом для выполнения нескольких задач часто требуется несколько щелчков мышью. С помощью CLI можно создать сценарий для выполнения многих сложных операций, введя всего одну «команду»: имя сценария. Скрипт представляет собой серию команд, помещенных в один файл. При выполнении сценарий выполняет все команды в файле.

Удаленный доступ: хотя можно выполнять команды удаленно в оболочке с графическим интерфейсом, эта функция обычно не устанавливается по умолчанию. С помощью командной строки CLI получить доступ к удаленной машине легко и обычно доступно по умолчанию.

Разработка. Обычно для разработки программ на основе графического интерфейса требуется больше времени по сравнению с программами на основе интерфейса командной строки. В результате на типичной ОС Linux обычно есть тысячи программ CLI, в то время как в ОС на основе GUI, например Microsoft Windows®, всего пара сотен программ. Больше программ означает больше мощности и гибкости.

Операционная система Microsoft Windows® была разработана, чтобы в первую очередь использовать интерфейс GUI из-за ее простоты, хотя также доступно несколько интерфейсов CLI. Для простых команд есть диалоговое окно «Выполнить», в котором вы можете ввести или просмотреть команды, которые хотите выполнить. Если вы хотите набрать несколько команд или увидеть вывод команды, вы можете использовать командную строку, также называемую оболочкой DOS. Недавно Microsoft осознала, насколько важно иметь мощную среду командной строки, и в результате представила Powershell.

Linux также имеет интерфейс командной строки и интерфейс командной строки. В отличие от Windows, Linux позволяет легко изменять оболочку графического интерфейса пользователя (также называемую средой рабочего стола), которую вы хотите использовать. Двумя наиболее распространенными средами рабочего стола для Linux являются GNOME и KDE, однако есть много других доступных оболочек графического интерфейса.

Чтобы получить доступ к CLI из GUI в операционной системе Linux, пользователь может открыть программу, называемую терминалом. Linux также можно настроить для запуска только CLI без графического интерфейса; Обычно это делается на серверах, которым не требуется графический интерфейс, прежде всего для освобождения системных ресурсов.

Bash Shell

Мало того, что операционная система Linux предоставляет несколько оболочек GUI, также доступны несколько оболочек CLI. Обычно эти оболочки создаются на основе одной из двух старых оболочек UNIX: Bourne Shell и C Shell. На самом деле оболочка bash получила свое название от Bourne Shell: Bourne Again SHell. В этом курсе вы узнаете, как использовать CLI для Linux с оболочкой bash, возможно, самой популярной в Linux. Оболочка bash имеет множество встроенных команд и функций, которые вы изучите, в том числе:

Псевдонимы: дайте команде другое или более короткое имя, чтобы сделать работу с оболочкой более эффективной.

Повторное выполнение команд: для сохранения повторного ввода длинных командных строк.

Подстановочные знаки: используются специальные символы, такие как?, * И [], чтобы выбрать один или несколько файлов в качестве группы для обработки.

Перенаправление ввода / вывода: использует специальные символы для перенаправления ввода <или << и вывода>.

Трубы: используются для подключения одной или нескольких простых команд для выполнения более сложных операций.

Фоновая обработка: позволяет программам и командам работать в фоновом режиме, пока пользователь продолжает взаимодействовать с оболочкой для выполнения других задач. Например:

```
sort red.txt &
```

Оболочка, используемая вашей учетной записью по умолчанию, устанавливается во время создания вашей учетной записи. По умолчанию многие дистрибутивы Linux используют bash для оболочки нового пользователя. Администратор может использовать команду usermod, чтобы указать другую оболочку по умолчанию после создания учетной записи.

Доступ к shell

То, как вы получаете доступ к оболочке командной строки, зависит от того, обеспечивает ли ваша система вход в систему через GUI или CLI:

- Системы на основе графического интерфейса: если система настроена на представление графического интерфейса, вам необходимо найти программное приложение под названием «Terminal». В среде рабочего стола GNOME приложение «Terminal» можно запустить Applications -> System Tools->Terminal.
- Системы на основе CLI. Многие системы Linux, особенно серверы, по умолчанию не настроены на предоставление графического интерфейса, поэтому вместо этого они представляют CLI. Если система настроена на представление интерфейса командной строки, то после входа в систему автоматически запускается приложение терминала.

В первые дни вычислений терминальными устройствами были большие машины, которые позволяли пользователям вводить данные через клавиатуру, а отображать вывод печатали на бумаге. Со временем терминалы эволюционировали, и их размеры сократились в нечто похожее на настольный компьютер с монитором видеодисплея для вывода и клавиатурой для ввода.

В конечном итоге, с появлением персональных компьютеров, терминалы стали программными эмуляторами реального оборудования. Все, что вы вводите в терминале, интерпретируется вашей оболочкой и переводится в форму, которая затем может быть выполнена ядром операционной системы.

Если вы находитесь в удаленном месте, то псевдотерминальные соединения также могут быть установлены через сеть с использованием нескольких методов. Небезопасные соединения могут быть установлены с использованием протоколов, таких как telnet, и программ, таких как rlogin, тогда как безопасные соединения могут быть установлены с использованием программ, таких как putty, и протоколов, таких как ssh.

Файловые системы

Помимо ядра и оболочки, другим важным компонентом любой операционной системы является файловая система. Для пользователя файловая система - это иерархия каталогов и файлов с корневым каталогом / в верхней части дерева каталогов. Для операционной системы файловая система представляет собой структуру, созданную в разделе диска, состоящем из таблиц, определяющих расположение каталогов и файлов.

Команда

Команда - это программа, которая при выполнении в командной строке выполняет действие на компьютере.

Когда вводится команда, процесс запускается операционной системой, которая может читать ввод, манипулировать данными и производить вывод. С этой точки зрения команда запускает процесс в операционной системе, который затем заставляет компьютер выполнять задание.

Однако есть и другой способ взглянуть на то, что является командой: посмотреть на ее источник. Источник - это то, откуда команда «взялась», и в оболочке вашего CLI есть несколько различных источников команд:

- Команды, встроенные в саму оболочку: хорошим примером является команда **cd**, так как она является частью оболочки **bash**. Когда пользователь вводит команду **cd**, оболочка **bash** уже выполняется и знает, как интерпретировать эту команду, не требуя запуска дополнительных программ.
- Команды, которые хранятся в файлах, которые ищет оболочка: если вы введете команду **ls**, оболочка выполнит поиск по каталогам, перечисленным в переменной **PATH**, чтобы найти файл с именем **ls**, который она может выполнить. Эти команды также можно выполнить, введя полный путь к команде.
- Псевдонимы (Aliases): псевдоним может переопределять встроенную команду, функцию или команду, найденную в файле. Псевдонимы могут быть полезны для создания новых команд, созданных из существующих функций и команд.

Например:

```
alias mycal="cal 2014"
mycal
```

- Функции: Функции также могут быть построены с использованием существующих команд для создания новых команд, переопределения команд, встроенных в оболочку, или команд, хранящихся в файлах. Псевдонимы и функции обычно загружаются из файлов инициализации при первом запуске оболочки.

Команды, хранимые в файлах

Команды, которые хранятся в файлах, могут быть в нескольких формах. Большинство команд написаны на языке программирования C, который изначально хранится в удобочитаемом текстовом файле. Эти исходные текстовые файлы затем компилируются в читаемые компьютером двоичные файлы, которые затем распространяются в виде командных файлов.

Пользователи, которые заинтересованы в том, чтобы увидеть исходный код скомпилированного лицензионного программного обеспечения GPL, могут найти его на сайтах, где оно было создано, например на kernel.org. Лицензионный код GPL также заставляет распространителей скомпилированных двоичных файлов, таких как RedHat и Debian, сделать исходный код доступным. Часто его можно найти в репозиториях дистрибьюторов.

Командные файлы могут также содержать читабельный текст в форме файлов сценариев (скриптов). Файл сценария (скрипта) представляет собой набор команд, который обычно выполняется в командной строке.

Возможность создавать свои собственные файлы сценариев является очень мощной функцией CLI. Если у вас есть ряд команд, которые вы регулярно вводите для выполнения какой-либо задачи, вы можете легко создать сценарий оболочки bash для выполнения этих нескольких команд, введя только одну команду: имя файла сценария. Вам просто нужно поместить эти команды в файл и сделать его исполняемым (более подробная информация об этом будет представлена в следующем разделе).

Большинство команд следуют простому шаблону синтаксиса:

command **[*options*…] [*arguments*…]**

Другими словами, вы вводите команду, за которой следуют один или несколько параметров (которые не всегда требуются) и один или несколько аргументов, прежде чем нажимать клавишу Enter. Хотя в Linux есть некоторые команды, которые не полностью соответствуют этому синтаксису, большинство команд используют этот синтаксис.

При вводе команды, которая должна быть выполнена, первым шагом является ввод имени команды. Название команды часто основано на том, что она делает, или на то, что разработчик, который думает, что разработчик команды, лучше всего описывает функцию команды.

Например, команда ls отображает список информации о файлах. Связывание имени команды с чем-то мнемоническим для того, что оно делает, может помочь вам легче запомнить команды.

Вы должны помнить, что каждая часть команды обычно чувствительна к регистру, поэтому LS неверен и завершится ошибкой, но ls верен и завершится успешно.

В следующем примере команда ls выполняется без каких-либо параметров или аргументов, что приводит к отображению текущего содержимого каталога. Многие команды, такие как команда ls, могут успешно выполняться без каких-либо опций или аргументов, но имейте в виду, что есть команды, которые требуют от вас вводить больше, чем просто одна команда.

Если вам необходимо добавить параметры, они могут быть указаны после имени команды. Короткие параметры (однобуквенные ключи) указываются через дефис, за которым следует один символ. Короткие параметры - это то, как параметры были определены традиционно.

Часто персонаж выбирается мнемоническим по своему назначению, например, выбирая букву «a» для «всех».

Несколько отдельных параметров могут быть заданы как отдельные параметры, например -a -l -r, или как -alr.

В следующем примере опция -l предоставляется команде ls, что приводит к выводу "long display":

```
ls -l
```

Можно ввести имя команды с несколькими короткими опциями. Вывод всех этих примеров одинаков, -l выдаст длинный список, а -r обратный порядок отображения результатов:

```
ls -l -r
ls -rl
ls -lr
```

Обычно короткие варианты можно комбинировать с другими короткими параметрами в любом порядке. Исключением является случай, когда для опции требуется аргумент.

Например, опция -w команды ls указывает желаемую ширину вывода и поэтому требует аргумента. В сочетании с другими параметрами опция -w может быть указана последней, за которой следует ее аргумент, и она по-прежнему будет действительна, как в ls -rtw 40, которая задает ширину вывода 40 символов. В противном случае параметр -w не может быть объединен с другими параметрами и должен быть указан отдельно.

Если используется несколько опций, требующих аргументов, объединять их нельзя. Например, опция -T также требует аргумента. Чтобы учесть оба аргумента, каждый параметр задается отдельно:

```
ls -w 40 -T 12
```

Некоторые команды поддерживают дополнительные параметры, длина которых превышает один символ. Длинным опциям для команд предшествует двойной дефис - и значением опции обычно является имя опции, например --all. Например:

```
ls --all
```

Для команд, которые поддерживают как длинные, так и короткие опции, можно использовать длинные и короткие опции одновременно:

```
ls --all --reverse -t
```

Команды, которые поддерживают длинные параметры, также часто поддерживают аргументы, которые могут быть указаны с одинаковым символом или без него (выходные данные обеих команд одинаковы:

```
ls --sort time
ls --sort=time
```

Существует специальная опция «одинокий» двойной дефис --, которая может использоваться для указания конца всех опций команды. Это может быть полезно в некоторых обстоятельствах, когда неясно, должен ли некоторый текст, следующий за параметрами, интерпретироваться как дополнительная опция или как аргумент команды.

Указание аргументов

После команды и любой из ее опций многие команды будут принимать один или несколько аргументов. Команды, использующие аргументы, могут требовать одного или нескольких из них. Например, команда **touch** требует как минимум один аргумент, чтобы указать имя файла для действия.

Команда **ls**, с другой стороны, позволяет указать аргумент имени файла, но это не требуется. Некоторые команды, такие как команда **cp** (копировать файл) и команда **mv** (переместить файл), требуют как минимум два аргумента, исходный и целевой файл.

Аргументы, содержащие необычные символы, такие как пробелы или не алфавитно-цифровые символы, обычно необходимо заключать в кавычки, заключая их в двойные или одинарные кавычки. Двойные кавычки не позволят оболочке интерпретировать некоторые из этих специальных символов; одинарные кавычки не позволяют оболочке интерпретировать любые специальные символы.

В большинстве случаев одинарные кавычки являются «более безопасными» и, вероятно, должны использоваться всякий раз, когда есть аргумент, содержащий символы, которые не являются буквенно-цифровыми.

Команда exec

Единственным исключением из основного синтаксиса команды является команда **exec**, которая принимает в качестве аргумента другую команду для выполнения. Что особенного в командах, которые выполняются с exec, так это то, что они заменяют текущую работающую оболочку.

Обычно команда **exec** используется в сценариях-оболочках. Если целью сценария является простая настройка и запуск другой программы, то он называется сценарием-оболочкой.

В скрипте-обертке последняя строка скрипта часто использует **exec program** (где **program** - это имя другой программы для выполнения) для запуска какой-либо другой программы. Сценарий, написанный таким образом, позволяет избежать продолжения работы оболочки во время работы запущенной программы, в результате этот метод экономит ресурсы (например, ОЗУ).

exec может использоваться для вызова перенаправления для одного или нескольких операторов в сценарии.

Команда uname

Команда **uname** отображает системную информацию. Эта команда выводит Linux по умолчанию, когда она выполняется без каких-либо опций.

Опции uname

Short Option	Long Option	Prints
`-a`	`--all`	All information
`-s`	`--kernel-name`	Kernel name
`-n`	`--node-name`	Network node name
`-r`	`--kernel-release`	Kernel release
`-v`	`--kernel-version`	Kernel version
`-m`	`--machine`	Machine hardware name
`-p`	`--processor`	Processor type or unknown
`-i`	`--hardware-platform`	Hardware platform or unknown
`-o`	`--operating-system`	Operating system
	`--help`	Help information
	`--version`	Version information

Команда **uname** полезна по нескольким причинам, в том числе, когда вам необходимо определить имя компьютера, а также текущую версию используемого ядра.