2024

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Раздел #1

ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ

Администрирование в Linux

Лабораторная работа #15

Архивация, сжатие и резервное копирование в Linux

РАЗРАБОТАЛ

СЕРГЕЙ СТАНКЕВИЧ (SERHEY STANKEVICH)

ЛАБОРАТОРНАЯ РАБОТА #15

Архивация, сжатие и резервное копирование в Linux

Цель работы

Научиться создавать архивы, сжатие и распаковку файлов, создавать резервные копии данных, синхронизировать файлы и каталоги.

Рассмотрим несколько программ, часто используемых для управления коллекциями файлов.

Норма времени выполнения: 2 академических часа.
Оценка работы: 3 зачетных единицы.

Краткие теоретические сведения.

Одной из основных задач администратора компьютерных систем является обеспечение безопасности данных, а одним из способов решения этой задачи – своевременное создание резервных копий системных файлов. Даже если вы не являетесь системным администратором, вам все равно пригодится умение создавать копии и перемещать большие коллекции файлов из одного места в другое и с одного устройства на другое.

Архивирование

Архивирование – это процесс сбора множества файлов и упаковывание их в один большой файл. Архивирование часто применяется как один из этапов создания резервных копий системы. Оно также используется при перемещении старых данных из системы в некоторое долговременное хранилище.

tar – утилита архивирования на ленту

В мире программного обеспечения для Unix-подобных систем существует программа **tar** – классический инструмент для архивирования файлов. Ее имя, расшифровывается как *tape archive* (архив на магнитной ленте), указывает, что первоначально инструмент предназначался для создания архивов на магнитных лентах. В настоящее время он с неменьшим успехом поддерживает другие устройства хранения.

Нам часто приходится видеть имена файлов с расширением tar или tgz, которые обозначают «простые» tar-архивы и архивы, сжатые с помощью gzip соответственно. Архив может состоять из группы отдельных файлов, иерархий каталогов или и того и другого.

Команда tar имеет следующий синтаксис:

```
tar режим[параметры] путь...
```

где под *режимом* подразумевается один из нескольких режимов работы.

Здесь очень важно различать *режимы* и *параметры*.

Режим и параметр можно объединять, и при этом не требуется использовать начальный дефис. Но имейте в виду, что режим всегда должен указываться первым, перед любыми параметрами.

В программе tar используется немного непривычный способ определения параметров, поэтому рассмотрим несколько примеров ее использования.

Создадим архив всей песочницы:

```
$ tar cf playground.tar playground
```

Эта команда создаст tar-архив с именем playground.tar, включающий всю иерархию каталогов песочницы. Режим **c** – создает архив из списка файлов и/или каталогов, а параметр **f** – используется для определения имени tar-архива.

Как видите, режим и параметр можно объединять, и при этом не требуется использовать начальный дефис. Но имейте в виду, что режим всегда должен указываться первым, перед любыми параметрами.

Посмотрим содержимое архива:

```
$ tar tf playground.tar
```

Параметр **t** – вывести список содержимого архива.

Для получения более подробного списка добавим параметр **v** (verbose – подробности):

```
$ tar tvf playground.tar
```

Извлечение содержимого архива

Извлечем содержимое архива в другой каталог. Для этого создадим новый каталог с именем foo, перейдем в него и извлечем содержимое tar-архива:

```
$ mkdir foo
```

```
$ cd foo
$ tar xf ../playground.tar
$ ls
playground
```

Если внимательно исследовать содержимое ~/foo/playground, можно заметить, что в результате распаковывания архива мы получили точные копии оригинальных файлов.

Однако следует помнить: если вы не действуете от имени *суперпользователя*, файлы и каталоги, извлеченные из архива, будут принадлежать *пользователю, выполнившему восстановление*, а не первоначальному их владельцу.

Способ обработки путей в архивах

Другой интересной особенностью tar является способ *обработки путей в архивах*. По умолчанию используются относительные пути, а не абсолютные. Для этого программа tar просто удаляет начальный слеш во всех путях. Чтобы показать это, создадим снова наш архив, но на этот раз укажем абсолютный путь к архивируемому каталогу:

```
$ cd
$ tar cf playground2.tar ~/playground
```

Как вы помните, командная оболочка заменит ~/playground полным путем /home/me/playground после нажатия клавиши ENTER, благодаря этому мы получим полный путь для нашей демонстрации. Далее извлечем архив, так же как прежде, и посмотрим, что из этого получилось:

```
$ cd foo
$ tar xf ../playground2.tar
$ ls
home playground

$ ls home
me

$ ls home/me
playground
```

Как видите, здесь при извлечении архива каталог home/me/playground был воссоздан не в корневом, а в текущем рабочем каталоге ~/foo, как было бы в случае с абсолютными путями. Это может показаться странным, но такое

решение имеет свои преимущества: оно позволяет извлекать архивы в любое другое место, а не только в исходное. Повторив это упражнение с параметром, управляющим выводом дополнительных сообщений (v), можно получить более понятную картину происходящего.

Ограничение количества извлекаемых данных

При распаковке архива можно ограничить количество извлекаемых данных. Например, можно извлечь из архива единственный файл:

```
tar xf archive.tar путь_к_файлу
```

Добавление в конец команды пути к файлу гарантирует извлечение только этого файла. Можно указать несколько путей. Обратите внимание, что путь к файлу должен быть полным относительным путем в архиве. Обычно в путях к файлам нельзя использовать групповые символы; но GNU-версия tar (именно эта версия входит в состав большинства дистрибутивов Linux) поддерживает параметр –wildcards.

Например:

```
$ tar xf ../playground2.tar --wildcards 'home/me/playground/dir-*/file-A'
```

Эта команда извлечет только файлы, соответствующие указанному пути с групповым символом dir-*.

Программа tar в сочетании с find

Например, команда find используется для поиска файлов, подлежащих включению в архив:

```
$ find playground -name 'file-A' -exec tar rf playground.tar '{}' '+'
```

Здесь команда find отыскивает в каталоге playground все файлы с именем file-A и затем с помощью операции -exec вызывает tar в режиме добавления в конец (**r**), чтобы добавить найденные файлы в архив playground.tar.

Использование tar в сочетании с find предоставляет отличный способ *инкрементного резервного копирования* дерева каталогов или всей системы. Применяя find для поиска файлов, более новых, чем эталонный файл, определяющий отметку времени, можно создать архив, содержащий только более новые файлы, чем файлы предыдущего архива, при этом предполагается, что время последнего изменения эталонного файла будет изменяться сразу после создания архива.

Программа tar и стандартный ввод/вывод

Здесь программа find создает список файлов и передает его по конвейеру программе tar:

```
$ find playground -name 'file-A' | tar cf - --files-from=- | gzip > playground.tgz
```

Когда программе tar передается имя файла - (дефис), под ним подразумевается стандартный ввод или стандартный вывод, в зависимости от контекста.

Кстати, *соглашение об использовании* дефиса (-) для представления стандартного ввода/вывода используется также многими другими программами.

Параметр --files-from (который можно заменить эквивалентным параметром -T) заставляет tar читать список путей из файла, а не из командной строки.

Наконец, архив, произведенный программой tar, передается по конвейеру программе gzip, чтобы в результате получить сжатый архив playground.tgz. Расширение tgz по общепринятому соглашению используется для tar-архивов, сжатых программой gzip. В некоторых случаях используется расширение tar.gz.

В нашем примере, для сжатия архива использовалась внешняя программа gzip, однако современные GNU-версии tar поддерживают возможность gzip- и bzip2-сжатия своими встроенными средствами, для чего служат параметры **z** и **j** соответственно. Поэтому упростим наш пример:

```
$ find playground -name 'file-A' | tar czf playground.tgz -T -
```

Если, понадобится создать архив, сжатый в формате bzip2, сделаем так:

```
$ find playground -name 'file-A' | tar cjf playground.tbz -T -
```

Таким образом произведя простую замену параметра сжатия **z** на **j** (и изменив расширение выходного файла на **.tbz**, указывающее, что для сжатия использовался алгоритм bzip2), мы задействовали bzip2-сжатие.

Передача файлов между системами по сети

Поддержку стандартного ввода и вывода командой tar можно использовать для передачи файлов между системами по сети. Для этого должны использоваться компьютеры, действующие под управлением Unix-подобных систем и имеющие программы tar и ssh. В этом случае можно организовать передачу каталога из удаленной системы в локальную. Пример как это работает представлен в упражнении № 2.

Сжатие файлов

На протяжении всей истории развития вычислительных технологий не прекращались попытки размещения большего числа данных в меньшем объеме, будь то память, устройства хранения или полоса пропускания сети. Многие устройства и технологии, прочно вошедшие в обиход, такие как переносные плееры, телевидение высокой четкости или широкополосный доступ в Интернет, обязаны своим существованием эффективным *технологиям сжатия данных*.

Сжатие данных

Сжатие данных – это процесс устранения избыточных данных. Допустим, у нас есть файл, хранящий изображение абсолютно черного квадрата размером 100 на 100 пикселей. В терминах хранения данных (если предположить, что каждый пиксель представлен 24 битами, или 3 байтами) изображение занимает 30 000 байт: 100 x 100 x 3 = 30 000.

Изображение, состоящее из пикселей одного цвета, содержит массу избыточных данных. Будь мы умнее, мы могли бы закодировать данные в виде простого описания того факта, что изображение представлено блоком из 30 000 пикселей черного цвета. То есть вместо хранения блока данных с 30 000 нулей (черный цвет в файлах изображений обычно представлен нулевым значением) мы могли бы сжать данные до числа 30 000 с последующим нулем, описывающим цвет. Такая схема сжатия называется кодированием длин серий (run-length encoding) и является одной из простейших технологий сжатия.

Современные технологии не в пример сложнее и эффективнее, но главная цель осталась прежней – избавиться от избыточных данных

Алгоритмы сжатия

Алгоритмы сжатия – это *математические методики*, применяемые для осуществления сжатия данных.

Основные категории алгоритмов сжатия:

- без потерь (lossless)
- с потерями (lossy)

Сжатие без потерь гарантирует сохранность всех данных, содержащихся в оригинале. То есть после восстановления файла из сжатой версии восстановленный файл будет иметь в точности то же содержимое, что и несжатый оригинал.

Сжатие с потерями, удаляет некоторые данные во время сжатия, чтобы обеспечить более высокую степень сжатия. Восстановленный файл в этом случае не будет совпадать с оригинальной версией, скорее он будет близкой аппроксимацией оригинала. Примерами сжатия с потерями могут служить формат JPEG (для изображений) и MP3 (для музыкальных произведений).

Мы рассмотрим только сжатие без потерь, поскольку большинство данных в компьютерах потерь не допускает.

Сжатие и распаковывание файлов

Программа gzip используется для сжатия одного или нескольких файлов. Во время работы она замещает оригинальный файл его сжатой версией.

Соответствующая программа gunzip используется для восстановления сжатых файлов до исходного состояния. Например, мы создали текстовый файл с именем foo.txt, записав в него список содержимого каталога /etc:

```
$ ls -l /etc > foo.txt
$ ls -l foo.*
-rw-r--r-- 1 me    me    15738 2012-10-14 07:15 foo.txt
```

Далее мы запустили программу gzip, которая заменила оригинальный файл сжатой версией с именем foo.txt.gz. В списке содержимого каталога, который был получен с использованием шаблона foo.*, можно видеть, что исходный файл действительно был замещен сжатой версией, и эта сжатая версия получилась почти в пять раз меньше оригинала. Можно также заметить, что сжатый файл имеет такие же разрешения и время, что и оригинал.

```
$ gzip foo.txt
$ ls -l foo.*
-rw-r--r-- 1 me    me     3230 2012-10-14 07:15 foo.txt.gz
```

Далее мы запустили программу gunzip, чтобы распаковать файл. После этого сжатая версия была *замещена* оригиналом, и снова с теми же разрешениями и временем.

```
$ gunzip foo.txt
$ ls -l foo.*
-rw-r--r-- 1 me    me    15738 2012-10-14 07:15 foo.txt
```

Программа gzip имеет множество параметров (смотри справочник man). Вот пример использования некоторых из них.

```
$ gzip foo.txt
$ gzip -tv foo.txt.gz
foo.txt.gz: OK
$ gzip -d foo.txt.gz
```

Здесь мы заменили файл foo.txt его сжатой версией с именем foo.txt.gz. Затем проверили целостность сжатой версии, передав параметры -t и -v. В заключение мы распаковали файл, вернув его исходное состояние.

Программа gzip и стандартный ввод/вывод

gzip можно использовать несколько необычным способом, через стандартные ввод и вывод:

```
$ ls -l /etc | gzip > foo.txt.gz
```

Эта команда создает сжатую версию списка с содержимым каталога.

Программа gunzip, которая распаковывает файлы, сжатые с помощью gzip, предполагает, что имена файлов оканчиваются расширением .gz, поэтому его можно не указывать при условии, что имя файла в команде не соответствует существующему несжатому файлу:

```
$ gunzip foo.txt
```

Если цель только в том, чтобы просмотреть содержимое сжатого текстового файла, сделать это можно так:

```
$ gunzip -c foo.txt | less
```

Вместе с gzip обычно поставляется программа zcat, которая действует подобно программе gunzip с параметром -c. Она применяется к файлам, сжатым с помощью gzip, по аналогии с командой cat:

```
$ zcat foo.txt.gz | less
```

Высокая степень сжатия ценой скорости

Программа bzip2 похожа на программу gzip, но использует иной алгоритм, который обеспечивает более высокую степень сжатия ценой снижения скорости работы. Во многих отношениях она действует точно так же, как gzip. Файл, сжатый с помощью bzip2, получает расширение bz2:

```
$ ls -l /etc > foo.txt
```

```
$ ls -l foo.txt
-rw-r--r-- 1 me    me    15738 2012-10-17 13:51 foo.txt
$ bzip2 foo.txt
$ ls -l foo.txt.bz2
-rw-r--r-- 1 me    me     2792 2012-10-17 13:51 foo.txt.bz2
$ bunzip2 foo.txt.bz2
```

Существует также программа bzip2recover для восстановления. Поврежденных файлов формата bz2.

> НЕ ПРЕВРАЩАЙТЕСЬ В ОДЕРЖИМЫХ МАНИЕЙ СЖАТИЯ
>
> Мне иногда приходится видеть, как кто-то пытается сжать файл, уже сжатый с применением эффективного алгоритма сжатия, выполняя нечто подобное:
>
> ```
> $ gzip picture.jpg
> ```
>
> Это *напрасная трата времени и дискового пространства*! Если применить процедуру сжатия к уже сжатому файлу, зачастую получается файл большего размера. Это объясняется тем, что все методики сжатия *добавляют в файл некоторую служебную информацию*, описывающую сжатие.
> Если попытаться сжать файл, не содержащий избыточной информации, сжатие не приведет к экономии места, которая могла бы покрыть расходы на хранение служебной информации.

Упаковывание и сжатие файлов

Программа zip одновременно является и инструментом сжатия, и архиватором. Формат файлов, используемый программой, знаком пользователям Windows программа читает и создает файлы с расширением zip. Однако в Linux чаще других используется программа сжатия gzip, а второе место занимает bzip2. Пользователи Linux используют zip в основном для обмена файлами с системами Windows, а не как основной инструмент сжатия и архивирования.

В простейшем случае программа zip имеет следующий синтаксис:

zip *параметры сжатый_файл файл*

Например, ниже показано, как создать zip-архив нашей песочницы:

```
$ zip -r playground zip playground
```

Без параметра -r (отвечает за рекурсивный обход каталогов) в архив будет включен только каталог playground (без своего содержимого). Расширение .zip

добавляется к имени выходного файла автоматически, а мы включили его в пример для наглядности.

В процессе создания zip-архива программа zip обычно выводит последовательность сообщений, они показывают состояние каждого файла, добавленного в архив zip добавляет файлы в архив, используя один из двух методов: «store» (простое сохранение) – без сжатия, как в примере, приведенном выше, «deflate» – со сжатием. Числовое значение, следующее за названием метода добавления, указывает достигнутую степень сжатия.

Извлечение содержимого из zip-архива выполняется просто – с помощью программы unzip:

```
$ cd foo
$ unzip ../playground.zip
```

Важное отличие zip (от tar) состоит в том, что если указанный архив существует, он *дополняется, а не замещается*. То есть существующий архив сохраняется, новые файлы добавляются в него, а существующие – замещаются.

Программа unzip позволяет выводить информацию о файлах и выборочно извлекать их, достаточно только передать ей имя интересующего нас файла, например:

```
$ unzip -l playground.zip playground/dir-087/file-Z
```

или

```
$ cd foo
$ unzip ../playground.zip playground/dir-087/file-Z
```

При наличии параметра -l программа unzip просто выведет информацию о содержимом архива, не извлекая файл. Если имя файла (или файлов) не указано, unzip выведет список всех файлов в архиве. Для получения более подробной информации следует добавить параметр -v.

Обратите внимание, что когда при извлечении из архива обнаруживается конфликт с существующим файлом, перед его заменой у пользователя запрашивается разрешение.

Подобно программе tar zip может использовать стандартный ввод и вывод, хотя реализация этой возможности имеет меньшую практическую ценность.

Программа zip способна принимать данные со стандартного ввода, поэтому ее можно использовать для сжатия вывода других программ:

```
$ ls -l /etc/ | zip ls-etc.zip -
```

```
adding: - (deflated 80%)
```

В этом примере вывод команды ls передается по конвейеру программе zip. Так же как tar, zip интерпретирует завершающий дефис как требование «использовать стандартный ввод вместо файла».

Программа unzip позволяет направить ее результаты в стандартный вывод, для чего следует передать параметр -p (pipe – конвейер, канал):

```
$ unzip -p ls-etc.zip | less
```

Мы затронули лишь самые основные возможности программ zip и unzip. Обе они имеют множество параметров, придающих им большую гибкость, хотя некоторые из них допустимы только для определенных платформ. Для обеих программ, zip и unzip, имеются подробные страницы справочного руководства (man) с *множеством полезных примеров*.

Синхронизация файлов и каталогов

В задачах резервного копирования систем широко используется стратегия синхронизации одного или нескольких каталогов с другими каталогами, находящимися в локальной системе (обычно на некотором извлекаемом устройстве) или в удаленной. Можно, к примеру, создать локальную копию веб-сайта, находящегося в разработке, и синхронизировать ее время от времени с «рабочей» копией на удаленном веб-сервере.

В мире Unix-подобных систем для решения этой задачи широко используется инструмент rsync. Эта программа синхронизует локальные и удаленные каталоги, используя протокол rsync remote-update (протокол удаленного обновления rsync), который позволяет rsync быстро обнаруживать различия между двумя каталогами и копировать минимальный объем данных, необходимый для синхронизации.

Программа **rsync** имеет следующий синтаксис:

rsync *параметры источник приемник*

где роль *источника* и *приемника* могут играть:

- локальный файл или каталог;
- удаленный файл или каталог в форме *[пользователь@]хост:путь*;
- удаленный сервер rsync, определяемый идентификатором *URI rsync://[пользователь@]хост[:порт]/путь*

Обратите внимание, что либо источник, либо приемник должен находиться в локальной системе. Копирование из удаленной системы в удаленную систему не поддерживается.

Процесс синхронизации нескольких локальных файлов представлен в упражнении № 3.

Использование rsync для копирования по сети

Одно из самых больших достоинств **rsync** – возможность копирования файлов по сети, об этом нам «говорит» буква **r** в названии rsync, что означает remote (удаленная). Удаленную синхронизацию можно выполнить одним из двух способов.

Первый можно использовать с удаленными системами, где установлена rsync и программа удаленной командной оболочки, такая как ssh.

Второй способ использования rsync для синхронизации файлов по сети заключается в использовании сервера rysnc. Сервер rsync можно настроить на работу в режиме демона, принимающего входящие запросы на синхронизацию. Этот прием часто используется для зеркалирования удаленных систем.

Процессы синхронизации для копирования файлов по сети представлен в упражнении № 3.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Sergey Stankewich

УПРАЖНЕНИЯ

Упражнение 1

Рассмотрим гипотетический, но имеющий практическую ценность пример использования **tar**. Представим, что нужно скопировать домашний каталог со всем его содержимым в другую систему и у нас имеется жесткий диск, подключаемый к порту USB, который можно использовать для переноса файлов. В современных системах Linux такие диски «как по волшебству» автоматически монтируются в каталог /media. Допустим также, что подключаемый жесткий диск имеет том с именем BigDisk. Чтобы создать требуемый архив, выполним следующую команду:

```
$ sudo tar cf /media/BigDisk/home.tar /home
```

После записи файла следует отмонтировать диск и подключить его ко второму компьютеру. И снова он автоматически монтируется в каталог /media/BigDisk. Однако если у вас нет под рукой другого компьютера подключитесь к этому же компьютеру, но в другой учетной записью.

Чтобы извлечь архив, выполните следующие команды:

```
$ cd /
$ sudo tar xf /media/BigDisk/home.tar
```

Обратите внимание, что здесь сначала выполняется переход в каталог /, чтобы извлечение производилось относительно корневого каталога, потому что все пути в архиве – относительные.

Упражнение 2

Другой интересный пример использования поддержки стандартного ввода и вывода командой tar связан с передачей файлов между системами по сети. Нужны две машины, действующие под управлением Unix-подобных систем и имеющие программы tar и ssh. Обеспечьте связь между компьютерами. Организуем передачу каталога из удаленной системы (с именем remote-sys в этом примере) в локальную.

Здесь мы скопируем каталог Documents из удаленной системы remote-sys в каталог с именем remote-stuff в локальной системе:

```
[me@linuxbox ~]$ mkdir remote-stuff
[me@linuxbox ~]$ cd remote-stuff
```

```
[me@linuxbox remote-stuff]$ ssh remote-sys 'tar cf - Documents' | tar xf -
me@remote-sys's password:
[me@linuxbox remote-stuff]$ ls
Documents
```

Как это получилось? Мы запустили программу tar в удаленной системе с помощью команды ssh. Команда ssh позволяет выполнить программу на удаленном компьютере в сети и «увидеть» результат в локальной системе – стандартный вывод, полученный в удаленной системе, пересылается в локальную систему для обзора. Мы воспользовались этой особенностью и заставили tar создать архив (режим c) и вывести его не в файл, а в стандартный вывод (параметр f с дефисом в качестве аргумента), вследствие чего архив передается через шифрованный туннель, созданный программой ssh, локальной системе. В локальной системе мы вызвали tar с целью распаковать архив (режим x), полученный со стандартного ввода (все тот же параметр f с дефисом в качестве аргумента).

Упражнение 3

Попробуем синхронизировать несколько локальных файлов. Сначала или создадим пустой каталог foo:

```
$ rm -rf foo/*
```

Далее синхронизируем каталог playground с соответствующей копией в foo:

```
$ rsync -av playground foo
```

Мы добавили два параметра: -a (для архивирования – обеспечивает рекурсивный обход и сохранение атрибутов файлов) и -v (подробный вывод), чтобы отразить каталог playground в каталог foo. В процессе выполнения команды можно просматривать список копируемых файлов и каталогов.

В конце программа выведет итоговое сообщение, включающее общий объем скопированных данных:

```
sent 135759 bytes  received 57870 bytes  387258.00 bytes/sec
total size is 3230  speedup is 0.02
```

Запустим команду еще раз, результат будет другой:

```
$ rsync -av playgound foo
building file list ... done
```

```
sent 22635 bytes  received 20 bytes  45310.00 bytes/sec
total size is 3230  speedup is 0.14
```

Обратите внимание на отсутствие списка файлов. Это объясняется тем, что программа rsync не обнаружила различий между ~/playground и ~/foo/playground и поэтому ничего не скопировала.

Если теперь изменить файл в playground и запустить rsync еще раз, она обнаружит изменившийся файл и скопирует только его.

```
$ touch playground/dir-099/file-Z
$ rsync -av playground foo
building file list ... done
playground/dir-099/file-Z
sent 22685 bytes  received 42 bytes  45454.00 bytes/sec
total size is 3230  speedup is 0.14
```

Продолжим эксперимент. Представьте воображаемый внешний жесткий диск, использовавшийся выше в упражнении №1 с командой tar. Если после подключения такого диска к системе он снова будет смонтирован в каталог /media/BigDisk, выполним первое резервное копирование системы, для начала создав каталог /backup на внешнем устройстве, а затем вызвав rsync для копирования наиболее важных компонентов системы на внешнее устройство:

```
$ mkdir /media/BigDisk/backup
$ sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/backup
```

В этом примере мы скопировали каталоги /etc, /home и /usr/local из нашей системы на воображаемый внешний диск. Мы добавили параметр **--delete**, чтобы удалить файлы, которые могут присутствовать на устройстве с резервной копией, но отсутствовать на устройстве-источнике (этот параметр не нужен при создании резервной копии в первый раз, но является полезным дополнением в последующих операциях копирования). Периодическое повторение процедуры подключения внешнего диска и запуск этой команды rsync является неплохим (хотя и не идеальным) способом сохранения резервной копии небольшой системы.

Конечно, здесь также могло бы пригодиться создание псевдонима. Определим псевдоним и добавим его в свой файл **.bashrc**, чтобы обеспечить возможность быстрого резервного копирования:

```
alias backup='sudo rsync -av --delete /etc /home /usr/local /media/Big-Disk/backup'
```

Теперь, чтобы выполнить всю работу, достаточно просто подключить внешний диск и ввести команду **backup**.

Продолжим эксперимент.

Проведем синхронизацию для копирования файлов по сети двумя способами.

Первый способ – используем удаленные системы, где установлены программы **rsync** и **ssh**.

Допустим, что в *локальной сети* имеется другая система с огромным объемом дискового пространства и мы хотели бы использовать эту систему для хранения резервной копии вместо внешнего диска. Если допустить, что в этой системе уже имеется каталог /backup, куда можно было бы сохранить наши файлы, мы могли бы выполнить резервное копирование так:

```
$ sudo rsync -av --delete --rsh=ssh /etc /home /usr/local remotesys:/backup
```

Мы внесли два изменения в команду, чтобы обеспечить копирование по сети. Во-первых, добавили параметр --rsh=ssh, который требует от rsync использовать в качестве удаленной командной оболочки программу ssh. Благодаря этому для передачи данных из локальной системы в удаленную мы можем использовать шифрованный туннель SSH. Во-вторых, мы добавили имя удаленного узла (в данном примере remote-sys) перед именем удаленного каталога.

Второй способ – с использованием *сервера rysnc*.

Утилиту rsync можно настроить на работу в режиме демона, принимающего входящие запросы на синхронизацию. Этот прием часто используется для зеркалирования удаленных систем. Например, компания Red Hat Software поддерживает огромный репозиторий программных пакетов, разрабатываемых для ее дистрибутива Fedora.

Для специалистов, занимающихся тестированием программного обеспечения, очень удобно иметь зеркало этой коллекции в ходе этапа тестирования, предшествующего этапу выпуска дистрибутива. Поскольку файлы в репозитории обновляются достаточно часто (порой по нескольку раз в день), неплохо было бы организовать периодическую синхронизацию локального зеркала вместо копирования всего объема репозитория.

Один из таких репозиториев хранится в университете Georgia Tech; мы могли бы создать его зеркало с помощью локальной программы rsync и сервера rsync в Georgia Tech:

```
$ mkdir fedora-devel
$ rsync -av -delete rsync://rsync.gtlib.gatech.edu/fedora-linux-core/devel-opment/i386/os fedora-devel
```

В этом примере мы использовали идентификатор URI удаленного сервера rsync, включающий протокол (rsync://), имя удаленного узла (rsync gtlib gatech edu) и путь к репозиторию.

We hope you enjoy working with Linux!

ЗАДАНИЯ

Задание 1

Сделайте архивную копию вашего ранее созданного программного проекта. Перенесите копию на другое устройство (можно использовать флеш-накопитель) или другую учетную запись, и распакуйте архивную копию. Сравните размеры полученных файлов.

Затем проведите сжатие вашего ранее созданного программного проекта. Перенесите сжатую версию проекта в другое пространство имен и распакуйте проект. Сравните размеры полученных файлов, а также сравните результаты с предыдущими результатами задания.

При выполнении задания предпочтительным является передача файлов между системами по сети.

Задание 2

Проведите упаквку и сжатие вашего ранее созданного программного проекта. Перенесите сжатую версию проекта в другое пространство имен и распакуйте проект. Сравните размеры полученных файлов, а также сравните результаты с результатами предыдущих заданий.

При выполнении задания можно использовать флеш-накопитель, но предпочтительным является передача файлов между системами по сети.

Сделайте вывод на основе анализа полученных результатов.

Задание 3

Сделайте синхроизацию каталогов или файлов программы, разработанной вами ранее. Минимальным требованием является синхронизация дистрибутива в локальной системе.

Дополнительным заданием являестя синхронизация дистрибутива по сети.

Попробуйте синхронизировать каталоги с локальным репозиторием Git. При необходимости скачайте и установите систему контоля версий Git на вашу систему Linux. А затем добавьте изменения в репозитории в удаленный репозиторий.

«Easy things should be easy and hard things should be possible»
«Простые вещи должны быть простыми, а сложные вещи должны быть возможными»


Perl

Контрольные вопросы:

1) Что такое *архивирование* файлов?
2) Что такое *относительны* и *абсолютынй* путь к файлу?
3) Что такое *сжание данных*?
4) Что такое *алгоритм сжатия*?
5) Какие основные типы алгоритмов сжатия существуют? В чем их особенности?
6) Для чего используется архивирование и (или) сжатие файлов?
7) Что такое заркалирование дистрибутива, файлов и каталогов? Для чего оно используется?

Дополнительная информация

Шоттс У. «Командная строка Linux. Полное руководство.» — СПб.: Питер, 2017. — 480 с.: ил. — (Серия «Для профессионалов»).
Смотри страницы: 229-243, 334-344.

Программирование в Linux. Самоучитель. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 400 с.: ил.
Смотри страницы:

Робачевский А. М. Операционная система UNIX®. - СПб.: 2002. - 528 ил.
Смотри страницы:

Интернет источники