Лабораторная работа №4. Работа с файлово-каталожной системой в ОС Linux

Рассматриваемые вопросы

1. Основные команды для работы с файлами и каталогами
2. Использование механизма ссылок
3. Прямая и косвенная адресация каталогов

Основные команды для работы с файлами и каталогами

`cd` - смена каталога
`cp` - копирование файлов
`ls` - выводит список файлов (в том числе каталогов) в указанном каталоге
`file` - выводит тип указанного файла
`find` - поиск файлов
`ln` - создание ссылок
`mkdir` - создание каталога
`mv` - перемещение файла или каталога
`pwd` – вывод имени текущего каталога
`rm` - удаления файла
`rmdir` - удаление каталога
`cat` - слияние и вывод файлов

Ссылки на файлы

В Linux существует два вида ссылок, обычно называемых жесткие ссылки и символьные, или "мягкие" ссылки. Жесткая ссылка является собственно символьным именем какого-либо файла – записью в соответствующем каталоге со ссылкой на индексный дескриптор этого файла. Таким образом, файл может иметь одновременно несколько символьных имен, в том числе в различных каталогах. Файл будет удален с диска только тогда, когда будет удалено последнее из его символьных имен. Нет такого понятия, как "настоящее" имя: все символьные имена одного файла имеют одинаковый статус.
Мягкая ссылка (или символьная ссылка, или symlink) принципиально отличается от жесткой ссылки: она является специальным файлом (с отдельным индексным дескриптором), который содержит полный путь к другому файлу. Таким образом, мягкая ссылка может указывать на файлы, которые находятся на других файловых системах, и не нуждается в наличии того файла, на который она указывает. Когда происходит попытка доступа к файлу, ядро операционной системы заменяет ссылку на тот путь, который она содержит. Однако команда **`rm`** удаляет саму ссылку, а не файл, на который она указывает. Для чтения состояния символьной ссылки, а также имени файла, на который она указывает, используется команда **`readlink`**.

Полное имя файла может задаваться как с использованием абсолютного пути, например, **`/home/user/file`**, так и с помощью относительного пути – пути, заданного относительно текущего каталога. Это особенно часто применяется в скриптах. Для этого в каждом каталоге есть два служебных каталога:
`..` – указывает на родительский каталог
`.` – указывает на текущий каталог
Например, команда **`cd ..`** позволит перейти на уровень выше, а команда **`cd .`** ничего не изменит.
Другой пример: команда **`./script.bash`** запускает скрипт именно из текущего каталога.
Наконец, если мы находимся в домашнем каталоге пользователя user, то путь к файлу
`./../../home/user/file`
будет соответствовать пути к файлу в домашнем каталоге, как и описанный выше пример абсолютного пути.
Для того, чтобы перейти к корню файловой системы можно использовать команду **`cd /`**
Для обозначения домашнего каталога активного пользователя можно использовать символ **`~`**. Тогда запись **`cd ~`** будет эквивалентна записи **`cd $HOME`**.

Задание на лабораторную работу

Создайте скрипты для перечисленных заданий и предъявите их преподавателю.
ВНИМАНИЕ! Все скрипты должны обрабатывать любые сценарии, соответствующие заданию, в том числе некорректный ввод параметров пользователем, использование имен файлов, содержащих необычные, но не запрещенные для использования в именах файлов символы, различные последовательности запусков разработанных скриптов и других действий пользователя в файловой системе. Все возникающие ошибки при выполнении скриптов, в том числе возникающие при выполнении отдельных утилит операционной системы, должны обрабатываться, и содержательные сообщения о них должны выводиться пользователю командами разрабатываемого скрипта.

1. Скрипт **`rmtrash`**
    a. Скрипту передается один параметр – имя файла в текущем каталоге вызова скрипта.

b. Скрипт проверяет, создан ли скрытый каталог **trash** в домашнем каталоге пользователя. Если он не создан – создает его.
c. После этого скрипт создает в этом каталоге жесткую ссылку на переданный файл с уникальным именем (например, присваивает каждой новой ссылке имя, соответствующее следующему натуральному числу) и удаляет файл в текущем каталоге.
d. Затем в скрытый файл **trash.log** в домашнем каталоге пользователя помещается запись, содержащая полный исходный путь к удаленному файлу и имя созданной жесткой ссылки.

2. Скрипт **untrash**
   a. Скрипту передается один параметр – имя файла, который нужно восстановить (без полного пути – только имя).
   b. Скрипт по файлу **trash.log** должен найти все записи, содержащие в качестве имени файла переданный параметр, и выводить по одному на экран полные имена таких файлов с запросом подтверждения.
   c. Если пользователь отвечает на подтверждение положительно, то предпринимается попытка восстановить файл по указанному полному пути (создать в соответствующем каталоге жесткую ссылку на файл из **trash** и удалить соответствующий файл из **trash**). Если каталога, указанного в полном пути к файлу, уже не существует, то файл восстанавливается в домашний каталог пользователя с выводом соответствующего сообщения. При невозможности создать жесткую ссылку, например, из-за конфликта имен, пользователю предлагается изменить имя восстанавливаемого файла.
3. Скрипт **backup**
   a. Скрипт создаст в **/home/user/** каталог с именем **Backup-YYYY-MM-DD**, где YYYY-MM-DD – дата запуска скрипта, если в **/home/user/** нет каталога с именем, соответствующим дате, отстоящей от текущей менее чем на 7 дней. Если в **/home/user/** уже есть «действующий» каталог резервного копирования (созданный не ранее 7 дней от даты запуска скрипта), то новый каталог не создается. Для определения текущей даты можно воспользоваться командой **date**.
   b. Если новый каталог был создан, то скрипт скопирует в этот каталог все файлы из каталога **/home/user/source/** (для тестирования скрипта создайте такую директорию и набор файлов в ней). После этого скрипт выведет в режиме дополнения в файл **/home/user/backup-report** следующую информацию: строка со сведениями о создании нового каталога с резервными копиями с указанием его имени и даты создания; список файлов из **/home/user/source/**, которые были скопированы в этот каталог.
   c. Если каталог не был создан (есть «действующий» каталог резервного копирования), то скрипт должен скопировать в него все файлы из **/home/user/source/** по следующим правилам: если файла с таким именем в каталоге резервного копирования нет, то он копируется из **/home/user/source**. Если файл с таким именем есть, то его размер сравнивается с размером одноименного файла в действующем каталоге резервного копирования. Если размеры совпадают, файл не копируется. Если размеры отличаются, то файл копируется с автоматическим созданием версионной копии, таким образом, в действующем каталоге резервного копирования появляются обе версии файла (уже имеющийся файл переименовывается путем добавления дополнительного расширения «**.YYYY-MM-DD**» (дата запуска скрипта), а скопированный сохраняет имя). После окончания копирования в файл **/home/user/backup-report** выводится строка о внесении изменений в действующий каталог резервного копирования с указанием его имени и даты внесения изменений, затем строки, содержащие имена добавленных файлов с новыми именами, а затем строки с именами добавленных файлов с существовавшими в действующем каталоге резервного копирования именами с указанием через пробел нового имени, присвоенного предыдущей версии этого файла.
4. Скрипт **upback**
   a. Скрипт должен скопировать в каталог **/home/user/restore/** все файлы из актуального на данный момент каталога резервного копирования (имеющего в имени наиболее свежую дату), за исключением файлов с предыдущими версиями.