Работа в Bash

Процессы

Основной задачей любой операционной системы является управление процессами. Операционная система должна распределять между ними ресурсы, предоставлять им возможность совместно использовать информацию и обмениваться ею, защищать ресурсы, используемые одним процессом, от их использования другими процессами, а также обеспечивать возможность синхронной работы процессов. Для этого операционная система должна поддерживать для каждого процесса свою структуру данных, в которой задается состояние данного процесса и указываются ресурсы, которыми он владеет. Это позволяет операционной системе осуществлять управление процессами.

Процесс — это в выполняемая в данный момент программа. Выполнение процесса должно осуществляться последовательно. Процесс определяется как сущность, представляющая основную единицу работы, которая должна быть реализована в системе.

Операционная система контролирует следующую деятельность, связанную с процессами:

- создание и удаление процессов
- планирование процессов
- синхронизация процессов
- коммуникация процессов
- разрешение тупиковых ситуаций

Создание процесса

Простейшей ОС не требуется создание новых процессов, так как внутри них работает всего одна программа, запускаемая во время включения устройства. В более сложных системах надо создавать новые процессы. Обычно они создаются:

- При запуске ОС;
- При появлении запроса на создание процесса.

Маски файлов

Маской имени файла называется комбинация специальных символов, позволяющая легко производить массовый отбор файлов по каким-то общим признакам. Например, файлов одного типа или с похожим именем. Возможен также одновременный отбор по названным критериям.

Для чего это нужно?

Во времена MS-DOS и подобных ей операционных систем, маски имен файлов использовались повсеместно. Это сегодня мы просто выделяем нужные файлы и копируем их, куда нужно. Тогда же для каждого действия вручную вбивалась соответствующая команда и применение масок имен позволяло значительно облегчить этот рутинный труд.

С появлением и развитием первых файловых менеджеров потребность в использовании масок сильно сократилась, но они используются и сегодня. Например, с помощью масок легко осуществлять поиск необходимых файлов. Несмотря на то, что поиск в Windows и так неплохо ищет файлы хоть по типу, хоть по имени, с помощью маски можно задать более конкретные условия, что сделает поиск более точным.

Также возможность отбор файлов по маскам используется в некоторых приложениях.

Ну и конечно же эти маски используются при работе с командной строкой в консоли.

Для отбора файлов по маске используются символы «?» и «*», где «*» — набор произвольных символов, а «?» — один произвольный символ.

Работа с потоками ввода-вывода

Существуют stdout, stdin, stderr потоки для ввода-вывода.

Ввод-вывод использует механизм перегрузки операций, гарантирующий вызов нужной функции-операции для указанного типа данных.

Классы потоков

Класс streambuf управляет буфером потока, обеспечивая базовые операции заполнения, опорожнения, сброса и прочих манипуляций с буфером.

Класс ios является базовым классом потоков ввода-вывода.

Классы istream и ostream — производные от ios и обеспечивают работу потоков соответственно ввода и вывода.

Класс iostream является производным от двух предыдущих и предусматривает функции как для ввода, так и для вывода.

Классы ifstream, of stream и f stream предназначены для управления файловым вводом-выводом.

Классы istrstream и ostrstream управляют резидентными потоками (форматированием строк в памяти). Это устаревшая методика, оставшаяся в C++Builder в качестве пережитка.

Каждый процесс открывает потоки (помните, что в UNIX файл — это и есть поток данных?) для ввода и вывода данных, а также вывода сообщений об ошибках и другой диагностической информации. Эти потоки:

- ? 0 — стандартный ввод (**stdin**),
- ? 1 — стандартный вывод (**stdout**),
- ? 2 — стандартный поток сообщений об ошибках (**stderr**).

Ссылаться на эти потоки можно по их *файловым дескрипторам*. 0, 1 и 2 — это и есть такие дескрипторы.

По умолчанию потоки ввода-вывода связываются с консолью, с которой запущен процесс: стандартный ввод — с клавиатурой, другие два потока — с экраном.

Все потоки можно перенаправить в другой файл. Это может быть файл на диске, файл устройства (например, принтер или /dev/null) или стандартный поток другого процесса.

Для перенаправления стандартного вывода команды используется символ >(«больше»). Если местом назначения служит файл, то можно его не перезаписывать, а присоединить (*append*) выводимые данные в его конец. Для такого перенаправления применяется символ >>.

Стандартный ввод перенаправляется символом < («меньше»).

Для перенаправления стандартного потока ошибок используется конструкция 2>. Чтобы присоединить **stderr** к **stdout** и перенаправить их вместе, пользуйтесь переадресацией 2>&1.

Для направления стандартного вывода одной команды на стандартный ввод другой применяется символ |

Перенаправление стандартных потоков ввода вывода

Перенаправление обычно осуществляется вставкой специального символа >между командами.

Обычно синтаксис выглядит так: команда1 > файл1 — выполняет команду1, помещая стандартный вывод в файл1; команда1 < файл1 выполняет команду1, используя в качестве источника ввода файл1 (вместо клавиатуры).

На каждый запрос ввода программы считывается одна строка текста из файла. Конструкция команда1 < файл1 > файл2 совмещает два предыдущих варианта: выполняет команду1 ввода из файла1 и вывода в файл2.

Создание конвейеров

Конвейеры — это возможность нескольких программ работать совместно, когда выход одной программы непосредственно идет на вход другой без использования промежуточных временных файлов. Синтаксис: команда1 | команда2, выполняет команду1 используя её поток вывода как поток ввода при выполнении команды2, что равносильно использованию двух перенаправлений и временного файла:

```
команда1 > ВременныйФайл
команда2 < ВременныйФайл
rm ВременныйФайл
```

Хороший пример командных конвейеров — это объединение echo с другой командой для получения интерактивности в неинтерактивных средах, к примеру:

```
echo -e "ИмяПользователя\nПароль" | ftp localhost
```

Здесь запускается ftp-клиент, который подключается к локальному узлу (localhost). На запрос программы водит первую строку ИмяПользователя, затем на следующий запрос ввода считывает строку Пароль. Строки в команде echoразделены через \n.

Редакторы

vi/vim— серия текстовых редакторов операционных систем семейства UNIX.

gedit — поставляется по умолчанию в окружении рабочего стола GMOME.

Nano— простой и удобный консольный текстовый редактор Linux.

История vi

Первая версия была написана Биллом Джоем в 1976 году.

В то время наиболее распространённым был редактор ed. Поскольку он был довольно сложным для «простого смертного», George Coulouris разработал редактор em (editor for mortals —

редактор для смертных). Билл Джой модифицировал редактор em и назвал его en, а позже на его основе создал ex, в котором появился визуальный режим, вызывавшийся командой vi. Так как пользователи больше времени проводили в визуальном режиме, ex 2.0, ставший частью 2BSD, сразу запускался уже в нём. Так появился vi, бывший в то время всего лишь жёсткой ссылкой на ex.

Редактор vi имеет три режима:

1. Командный - в этом режиме можно перемещаться по файлу и выполнять редактирующие команды над текстом. Команды вызываются ОБЫЧНЫМИ ЛАТИНСКИМИ БУКВАМИ.

2. Ввода текста - в этом режиме обычные латинские буквы будут вставляться в текст.

3. Режим строчного редактора ED используется для управления файлами (типа сохранить файл, зачитать файл и т.д.)

Как выйти из vi

чтобы выйти из файла без сохранения, нажмите:

ESC : q ! Enter

чтобы выйти из файла, сохранив изменения, нажмите:

ESC : w ! Enter

ESC : q Enter

выйти из файла с сохранением, одной командой:

ESC : wq Enter

Кратко о скриптах

Итак, что же такое скрипт ?

Как правило, *скриптом называется программа или программный файл сценарий*. Ну а если быть предельно точным, то скриптом будет называться практически любая исполняемая процедура.

Если говорить об Интернет-технологиях, то понятие "скрипт" уже несколько сужается и его можно охарактеризовать, как исполняемую процедуру, написанную на каком либо языке, которая запускается на выполнение со стороны сервера по запросу поступившему с конкретно определенной веб-страницы.

Сфера применения скриптов огромна. Например:

- seo-скрипты, помогающие продвигать сайты на чистом PHP
- сюда же можно отнести скрипты автоматизации бизнеса
- cap-скрипты - скрипты активной рекламы
- popup-скрипты - скрипты всплывающей информации
- скрипты наблюдающие за статистикой посещений (счетчики посещаемости)
- скрипты - гостевые книги
- скрипты - системы комментариев к понравившимся статьям
- на скриптах основаны все cms и форумы
- скрипты помогают динамическому отображению веб-сайта
- при их помощи пользователь получает возможность обращаться к базам данных
- скрипты позволяют организовать изменения части сайта без перегрузки всей страницы
- popup-скрипты и другие полезные скрипты.

Скриптовый язык удобен в следующих случаях:

- Если нужно обеспечить программируемость без риска дестабилизировать систему.
- Если важен выразительный код.
- Скрипты известны низким порогом вхождения, на скриптовом языке может писать даже низкоквалифицированный программист.
- Если требуется кроссплатформенность.
- Скриптовые языки применяются для написания программ, не требующих оптимальности и быстроты исполнения.
- Также скриптовые языки хороши для визуализации данных: создания сложных графиков и презентаций, а также простых программ .

Основы программирования bash

BASH — Bourne-Again SHell (что может переводится как «перерожденный шел», или «Снова шел Борна(создатель sh)»), самый популярный командный интерпретатор в юниксоподобных системах, в особенности в GNU/Linux.

Любой bash-скрипт должен начинаться со строки:

#!/bin/bash

в этой строке после #! указывается путь к bash-интерпретатору, поэтому если он у вас установлен в другом месте(где, вы можете узнать набрав whereis bash) поменяйте её на ваш путь.

Коментарии начинаются с символа # (кроме первой строки).

В bash переменные не имеют типа(о них речь пойдет ниже)

Встроенные команды

- **break** выход из цикла **for, while** или **until**
- **continue** выполнение следующей итерации цикла **for, while** или **until**
- **echo** вывод аргументов, разделенных пробелами, на стандартное устройство вывода
- **exit** выход из оболочки
- **export** отмечает аргументы как переменные для передачи в дочерние процессы в среде
- **hash** запоминает полные имена путей команд, указанных в качестве аргументов, чтобы не искать их при следующем обращении
- **kill** посылает сигнал завершения процессу
- **pwd** выводит текущий рабочий каталог
- **read** читает строку из ввода оболочки и использует ее для присвоения значений указанным переменным.\
- **return** заставляет функцию оболочки выйти с указанным значением
- **shift** перемещает позиционные параметры налево
- **test** вычисляет условное выражение
- **times** выводит имя пользователя и системное время, использованное оболочкой и ее потомками
- **trap** указывает команды, которые должны выполняться при получении оболочкой сигнала
- **unset** вызывает уничтожение переменных оболочки
- **wait** ждет выхода из дочернего процесса и сообщает выходное состояние.

Переменные Bash

- **$DIRSTACK** - содержимое вершины стека каталогов
- **$EDITOR** - текстовый редактор по умолчанию
- **$EUID** - Эффективный UID. Если вы использовали программу su для выполнения команд от другого пользователя, то эта переменная содержит UID этого пользователя, в то время как...
- **$UID** - ...содержит реальный идентификатор, который устанавливается только при логине.
- **$FUNCNAME** - имя текущей функции в скрипте.
- **$GROUPS** - массив групп к которым принадлежит текущий пользователь
- **$HOME** - домашний каталог пользователя
- **$HOSTNAME** - ваш hostname
- **$HOSTTYPE** - архитектура машины.
- **$LC_CTYPE** - внутренняя переменная, котороя определяет кодировку символов
- **$OLDPWD** - прежний рабочий каталог
- **$OSTYPE** - тип ОС
- **$PATH** - путь поиска программ
- **$PPID** - идентификатор родительского процесса
- **$SECONDS** - время работы скрипта(в сек.)
- **$#** - общее количество параметров переданных скрипту
- **$*** - все аргументы переданыне скрипту(выводятся в строку)
- **$@** - тоже самое, что и предыдущий, но параметры выводятся в столбик
- **$!** - PID последнего запущенного в фоне процесса
- **$$** - PID самого скрипта

Циклы Bash

Циклы позволяют выполнять один и тот же участок кода необходимое количество раз. В большинстве языков программирования существует несколько типов циклов. Большинство из них поддерживаются оболочкой Bash. Мы рассмотрим их все в сегодняшней статье, но сначала поговорим какими они бывают:

- **for** - позволяет перебрать все элементы из массива или использует переменную-счетчик для определения количества повторений;
- **while** - цикл выполняется пока условие истинно;
- **until** - цикл выполняется пока условие ложно.