Отчёт по лабораторной работе №11:

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Касакьянц Владислав Сергеевич

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# 2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (>/dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду man с помощью командного файла. Изучите содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.
3. Используя встроенную переменную $RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 д 32767.

# 3 Выполнение лабораторной работы

1. Напишем командный файл (рис. 1), реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (>/dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов (рис. 2):

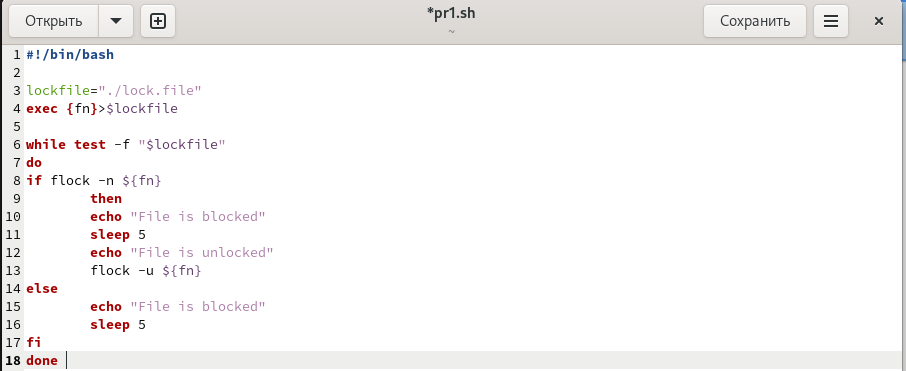


Рис. 1: Командный файл №1

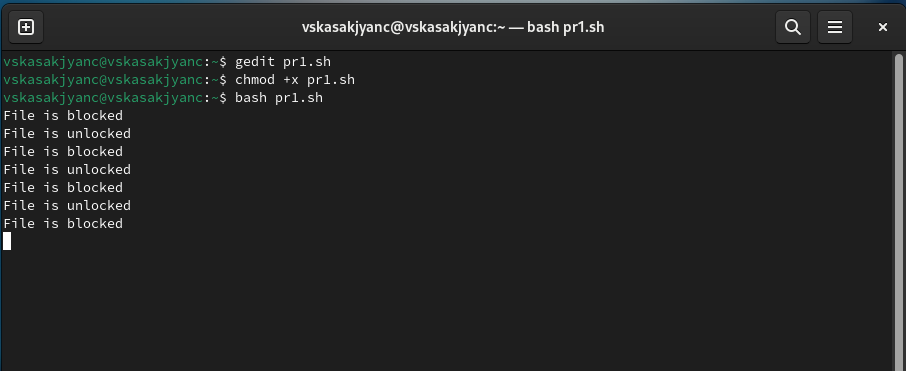


Рис. 2: Работа кода №1

1. Реализуем команду man с помощью командного файла. Изучим содержимое каталога /usr/share/man/man1 (рис. 3). В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл (рис. 4) получает в виде аргумента командной строки название команды и в виде результата выдаёт справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1 (рис. 5) :

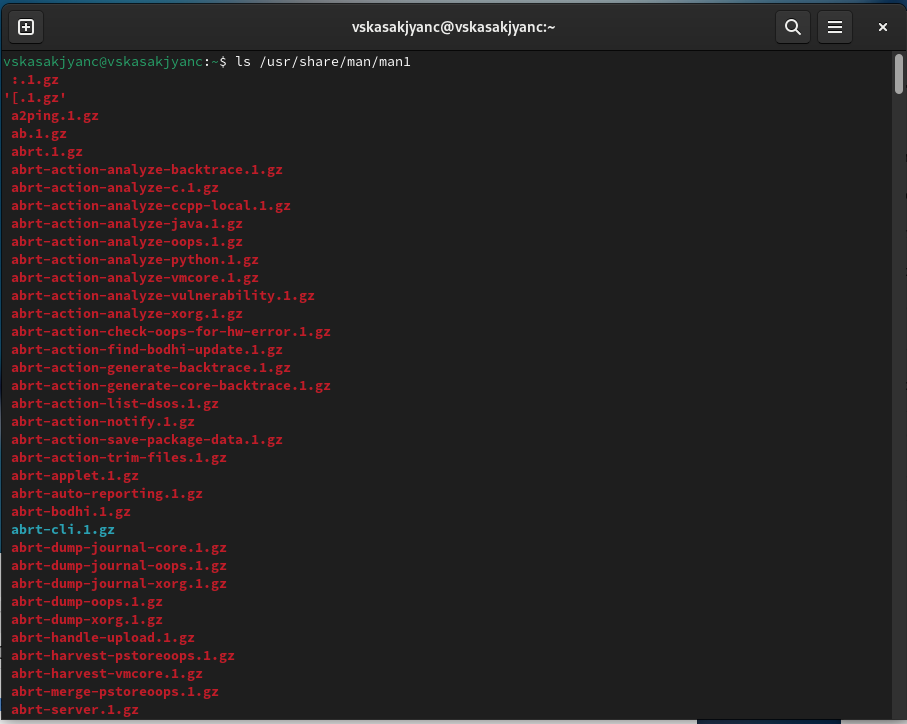


Рис. 3: Каталог man1

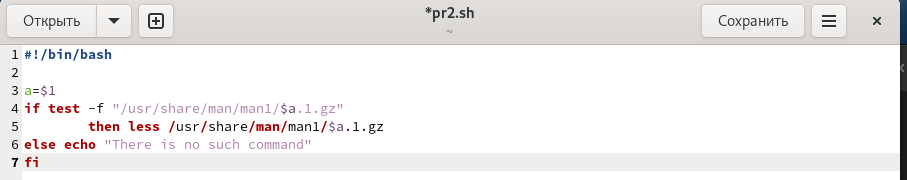


Рис. 4: Командный файл №2

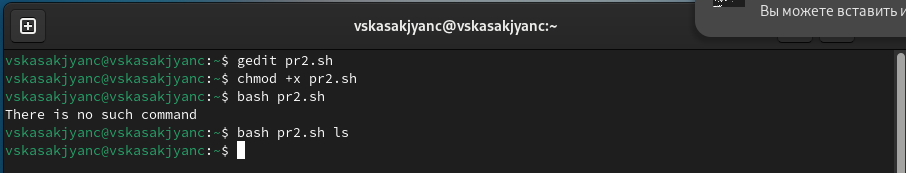


Рис. 5: Работа программы №2

Получаем справку команды ls (рис. 6).

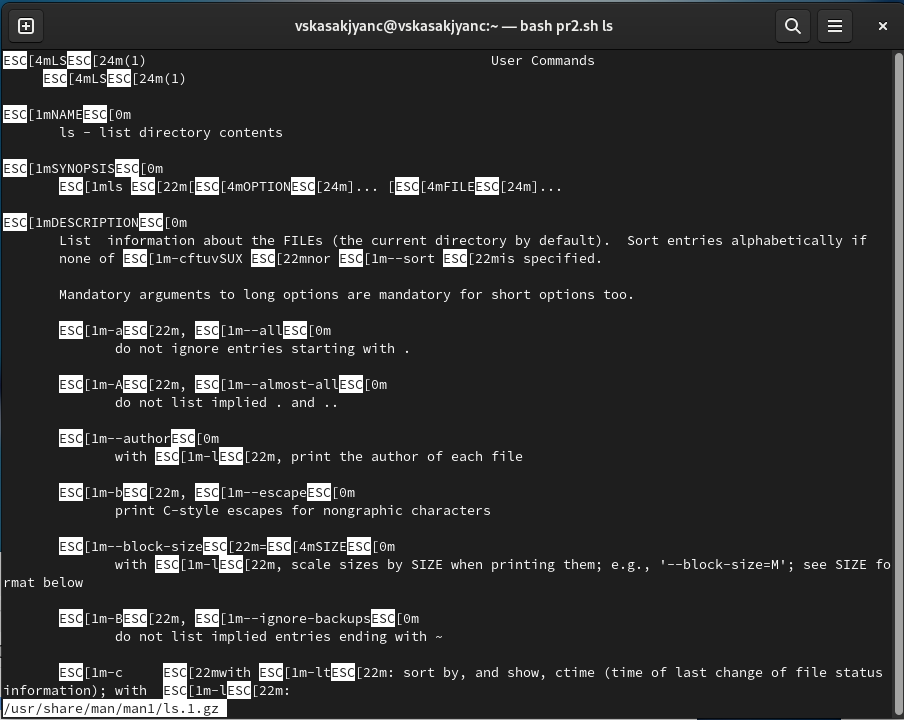


Рис. 6: Справка команды ls

1. Используя встроенную переменную $RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767 (рис. 7), (рис. 8):

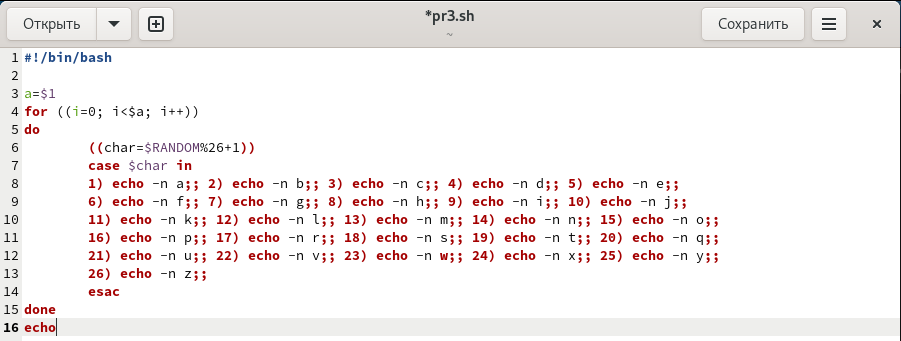


Рис. 7: Командный файл №3

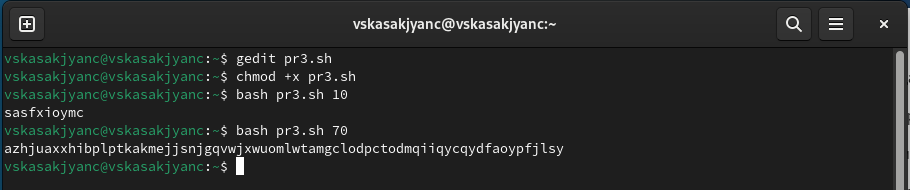


Рис. 8: Результат выполнения командного файла №3

# 4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: while [$1 != "exit"]

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [ и перед второй скобкой ] выражение $1 необходимо взять в “”, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так:

while [ "$1" != "exit" ]

1. Как объединить (конкатенация) несколько строк в одну?

Для объединения нескольких строк в одну в bash-скриптах можно использовать следующие методы:

* Просто написать строки одну за другой без разделителей:

str="Строка1""Строка2""Строка3"

* Использовать оператор конкатенации (+=), чтобы добавить к строке дополнительные данные:

str="Строка1"  
str+="Строка2"  
str+="Строка3"

* Использовать команду printf для форматирования и объединения строк:

printf -v str "%s%s%s" "Строка1" "Строка2" "Строка3"

* Использовать команду echo с опцией -n для предотвращения добавления новой строки после каждого вызова:

str=$(echo -n "Строка1"; echo -n "Строка2"; echo -n "Строка3")

1. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?

Утилита seq в bash используется для генерации последовательности чисел. Она позволяет указать начальное число, шаг и конечное число для создания последовательности. Например, команда seq 1 2 10 выведет числа от 1 до 10 с шагом 2. Чтобы реализовать функционал seq без использования самой утилиты, можно применить следующие подходы:

* Использование цикла for:

for ((i=1; i<=10; i+=2)); do  
 echo $i  
done

* Использование цикла while:

i=1  
while [ $i -le 10 ]; do  
 echo $i  
 i=$((i+2))  
done

* Использование brace expansion {}:

echo {1..10..2}

1. Какой результат даст вычисление выражения $((10/3))?

Данное выражение в bash приведёт к выполнению целочисленного деления, так как bash не поддерживает вещественную арифметику в арифметических операциях. В результате, вы получите целую часть от деления 10 на 3, то есть **3**.

1. Укажите кратко основные отличия командной оболочки zsh от bash.

Командная оболочка zsh часто сравнивается с bash, так как обе они широко используются и имеют много общего. Однако между ними есть ряд отличий:

* **Интерактивность**: zsh предлагает более продвинутые возможности для интерактивной работы, включая улучшенное автодополнение и коррекцию ошибок.
* **Темы и плагины**: zsh поддерживает темы и плагины через фреймворк oh-my-zsh, что позволяет легко настраивать внешний вид и функциональность оболочки.
* **Синтаксис**: В zsh есть улучшения синтаксиса, такие как более гибкие глобальные выражения и расширенные возможности для работы со строками и массивами.
* **Совместимость**: zsh обычно совместим с bash, но включает дополнительные функции, которые могут не работать в bash.
* **Конфигурация**: Файлы конфигурации для zsh (~/.zshrc) и bash (~/.bashrc и ~/.bash\_profile) различаются, и zsh предоставляет более обширные настройки по умолчанию.

Эти отличия делают zsh популярным выбором для пользователей, которые ищут более богатый пользовательский интерфейс и гибкость в настройке своей командной оболочки. Однако bash остаётся стандартом для многих систем и скриптов из-за его широкой доступности и предсказуемости.

1. Проверьте, верен ли синтаксис данной конструкции for ((a=1; a <= LIMIT; a++)).

Синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать $ перед переменными ().

1. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Сравнение bash с другими языками программирования можно провести по нескольким критериям:

**Преимущества bash:**

* **Специализация**: bash идеально подходит для автоматизации рутинных задач в Unix-подобных системах.
* **Встроенная поддержка**: Практически в каждой Unix-подобной системе есть bash, что делает его универсальным инструментом для системного администрирования.
* **Простота использования**: Для написания простых скриптов не требуется сложная настройка или компиляция.
* **Интеграция с системой**: bash обладает прямым доступом к системным вызовам и утилитам командной строки.

**Недостатки bash:**

* **Ограниченная область применения**: bash не подходит для сложных программных систем, где требуется высокая производительность и масштабируемость.
* **Отсутствие современных функций**: В bash нет многих возможностей, доступных в полноценных языках программирования, таких как объектно-ориентированное программирование.
* **Сложность**: Сложные скрипты на bash могут быть трудночитаемыми и поддерживаемыми из-за ограничений синтаксиса и структуры языка.

Сравнивая bash с языками программирования, такими как **Python** или **Java**, можно отметить, что эти языки предлагают более широкие возможности для разработки программного обеспечения, включая библиотеки для научных расчетов, машинного обучения, веб-разработки и многого другого. Однако для системного администрирования и быстрой автоматизации задач bash остается предпочтительным выбором из-за его простоты и тесной интеграции с операционной системой.

# 5 Выводы

В данной лабораторной работе мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Список литературы