Отчет по лабораторной работе № 14

Архитектура компьютера, раздел Операционные системы

Кусоро Майова Джеймс

Содержание

# Цель

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

# Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Ко- мандный 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 ката- лога /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, генерирую- щий случайную последовательность букв латинского алфавита. Учтите, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

# Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это про- грамма, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления програм- мой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей сов- мещает свойства оболочек С и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных опера- ционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

# Выполнение лабораторной работы

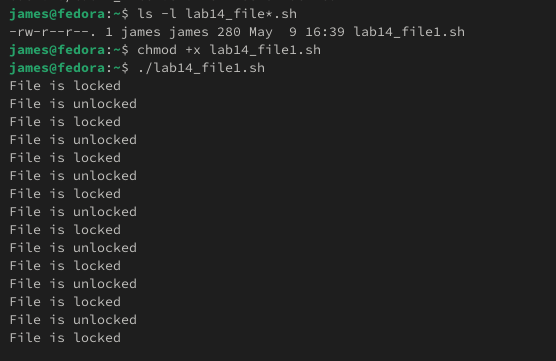
## командный файл, реализующий упрощённый механизм семафоров

Чтобы создать данный командный файл, я создала новый файл и написала в нем некоторый скрипт. Он устанавливает переменную lockfile для пути к файлу блокировки, открывает файл для записи и назначает ему дескриптор файла. Далее входит в цикл, который выполняется, пока файл блокировки существует. Пытается получить эксклусивную блокировку для файла. Если это удается, выводит “file locked”, ждет 5 секунд а затем выводит “file unlocked”:



упрощённый механизм семафоров (код)

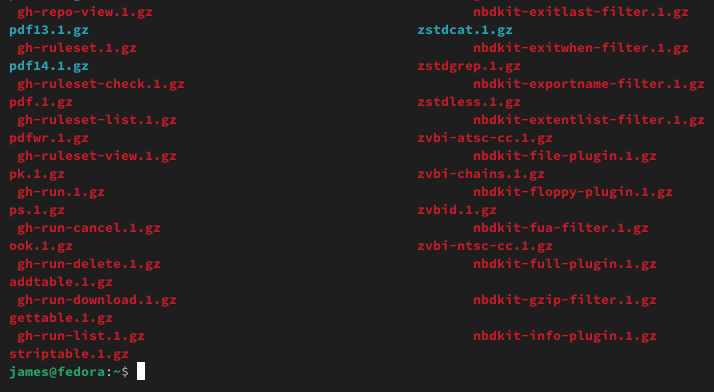
lockfile="./lock.file"  
exec {fn}>$lockfile  
  
while test -f "$lockfile"  
do  
if flock -n ${fn}  
then  
 echo "File is locked"  
 sleep 5  
 echo "File is unlocked"  
 flock -u ${fn}  
else  
 echo "File is locked"  
 sleep 5  
  
fi  
done



результаты кода

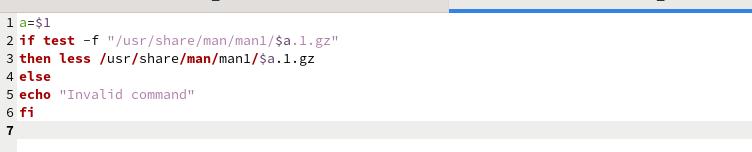
## Реализовать команду man с помощью командного файла

Я изучила содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд:



ls /usr/share/man/man1

Потом я создала файл и в нем написала скрипт реализирующий команды man. Он принимает аргумент $1, проверяет существование файла в /usr/share/man/man1, и если файл существует, использует less для отображения содержимого сжатой страницы руководства. Если файл не существует, выводит “invalid command”:

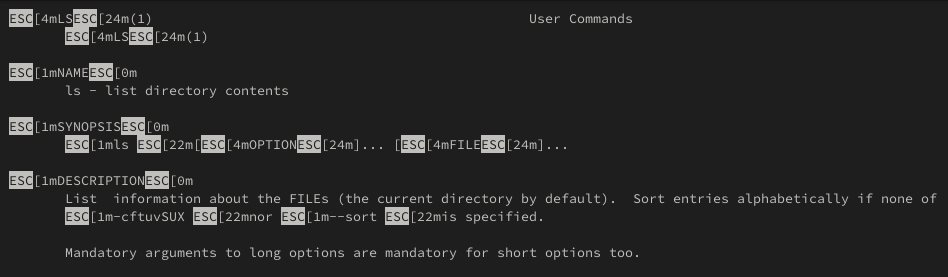


командный файл man

a=$1  
if test -f "/usr/share/man/man1/$a.1.gz"  
then less /usr/share/man/man1/$a.1.gz  
else  
echo "Invalid command"  
fi



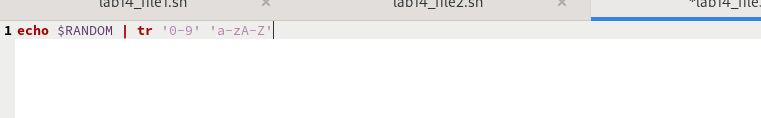
проверка командного файла man



проверка командного файла man

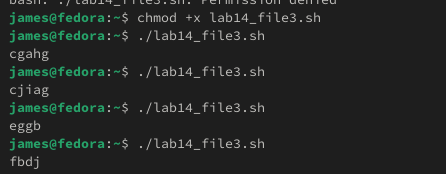
## написать командный файл, генерирующий случайную последовательность букв латинского алфавита.

Я написала скрипт который генерирует случайное число используя $RANDOM, а затем с помощью tr заменяет каждую цифру на букву от ‘a-z’ и ‘A-Z’:



командный файл, генерирующий случайную последовательность букв

echo $RANDOM | tr '0-9' 'a-zA-Z'



запуск скрипта

# Выводы

При выполнении данной работы я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Ответы на контрольные вопросы

1. 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 [ “$1” != “exit” ]
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: VAR1=“Hello,” VAR2=” World” VAR3=“VAR2” echo “$VAR3" Результат: Hello, World Второй: VAR1="Hello, " VAR1+=" World" echo "$VAR1” Результат: Hello, World
3. Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение is не выдает. seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. seq FIRST INCREMENT LAST: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод. seq -f «FORMAT» FIRST INCREMENT LAST: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными. seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными. seq -w FIRST INCREMENT LAST: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.
4. Результатом данного выражения $((10/3)) будет 3, потому что это целочисленное деление без остатка.
5. Отличия командной оболочки zsh от bash: В zsh более быстрое автодополнение для cd с помощью Тab В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала В zsh поддерживаются числа с плавающей запятой В zsh поддерживаются структуры данных «хэш» В zsh поддерживается раскрытие полного пути на основенеполных данных В zsh поддерживается замена части пути В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim
6. for ((a=1; a <= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать $ перед переменными ().
7. Преимущества и недостатки скриптового языка bash:

Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS Удобное перенаправление ввода/вывода Большое количество команд для работы с файловыми системами Linux Можно писать собственные скрипты, упрощающие работу в Linux Недостатки скриптового языка bash: Дополнительные библиотеки других языков позволяют выполнить больше действий Bash не является языков общего назначения Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на быстроте выполнения этого скрипта Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

# Список литературы

[Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2288101/mod_resource/content/4/012-lab_shell_prog_3.pdf)