

# **Лабораторная работа № 12.**

**Программирование в командном процессоре ОС UNIX. Расширенное  
программирование**

**ОЗЬЯС Стев Икнэль Дани**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Задание 1 . . . . .	9
4.2	Задание 2 . . . . .	10
4.3	Задание 3 . . . . .	10
<b>5</b>	<b>Выводы</b>	<b>12</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>13</b>
	<b>Список литературы</b>	<b>15</b>

## Список иллюстраций

4.1	Командный файл №1 . . . . .	9
4.2	Командный файл №2 . . . . .	10
4.3	Командный файл №3 . . . . .	11

# Список таблиц

# 1 Цель работы

Цель работы — изучить основы программирования в оболочке ОС UNIX. Научится 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  $t_1$  дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени  $t_2 < t_1$ , 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 до 32767.

## 3 Теоретическое введение

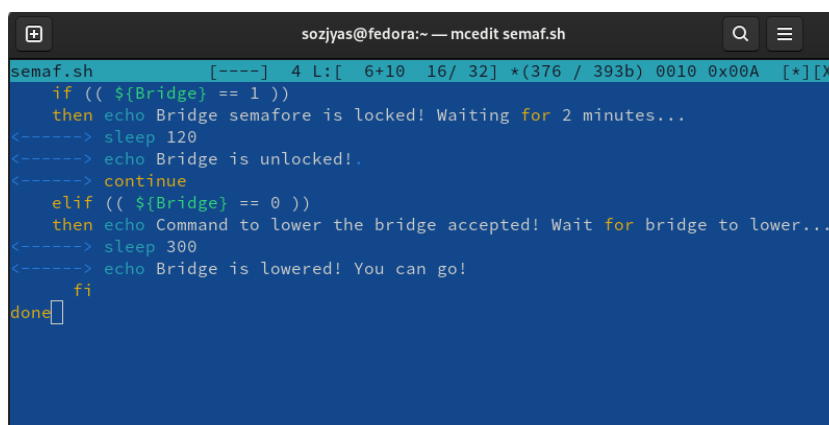
- Командный файл (скрипт или сценарий) - это текстовый файл, состоящий из команд интерпретатора. При запуске этого файла последовательно выполняются все команды, содержащиеся в нем.
- Для создания командного файла:
  - Запустите текстовый редактор.
  - Последовательно запишите команды, располагая каждую команду на отдельной строке.
  - Сохраните этот файл, сделайте его исполняемым, применив команду:
    1. `chmod +x имя_файла.`
  - Запустите созданный файл и проверьте правильность выполнения команд.
  - В случае нахождения ошибки, в текстовом редакторе внесите изменения в командный файл, сохраните его и проверьте еще раз.
- Более подробно об Unix см. в [1–6].



## 4 Выполнение лабораторной работы

### 4.1 Задание 1

1. Написал командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени  $t_1$  дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени  $t_2 < t_1$ , 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. 4.1)

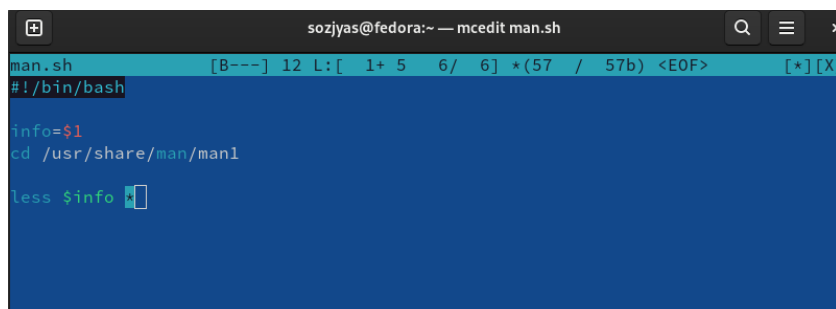


```
soziyas@fedora:~ — mcedit semaf.sh
semaf.sh [----] 4 L:[ 6+10 16/ 32] *(376 / 393b) 0010 0x00A [*][X]
  if (( ${Bridge} == 1 ))
  then echo Bridge semaphore is locked! Waiting for 2 minutes...
<-----> sleep 120
<-----> echo Bridge is unlocked!
<-----> continue
  elif (( ${Bridge} == 0 ))
  then echo Command to lower the bridge accepted! Wait for bridge to lower...
<-----> sleep 300
<-----> echo Bridge is lowered! You can go!
  fi
done
```

Рис. 4.1: Командный файл №1

## 4.2 Задание 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`. (рис. 4.2)



```
sozjyas@fedora:~ — mcedit man.sh
man.sh [B---] 12 L: [ 1+ 5 6/ 6] *(57 / 57b) <EOF> [*] [X]
#!/bin/bash

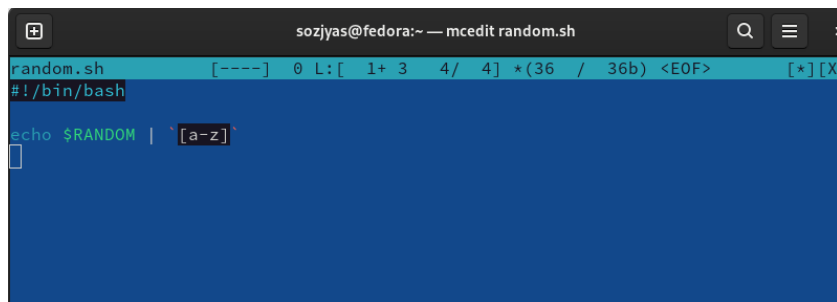
info=$1
cd /usr/share/man/man1

less $info *
```

Рис. 4.2: Командный файл №2

## 4.3 Задание 3

- Используя встроенную переменную `$RANDOM`, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтил, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. 4.3)



```
sozjyas@fedora:~ — mcedit random.sh
random.sh [----] 0 L:[ 1+ 3 4/ 4] *(36 / 36b) <EOF> [*][X]
#!/bin/bash
echo $RANDOM | tr -dc 'a-z'
```

Рис. 4.3: Командный файл №3

## **5 Выводы**

Я изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 6 Контрольные вопросы

Контрольные вопросы 1. \$1 следует внести в кавычки. 2. С помощью знака >| можно объединить несколько строк в одну. 3. Эта утилита выводит последовательность целых чисел с заданным шагом. Также можно реализовать с помощью утилиты jot. 4. Результатом вычисления выражения  $\$(10/3)$  будет число 3. 5. В zsh можно настроить отдельные сочетания клавиш так, как вам нравится. Использование истории команд в zsh ничем особенным не отличается от bash. Zsh очень удобен для повседневной работы и делает добрую половину рутины за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в zsh после for обязательно вставлять пробел, нумерация массивов в zsh начинается с 1, чего совершенно невозможно понять. Так, если вы используете shell для повседневной работы, исключая написание скриптов, используйте zsh. Если вам часто приходится писать свои скрипты, только bash! Впрочем, можно комбинировать. 6. Синтаксис конструкции for ((a=1; a <= LIMIT; a++)) верен. 7. Язык bash и другие языки программирования: - Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на си/си++, скомпилированных с максимальной оптимизацией; - Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам; - Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя

большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ; - Скорость кодов, генерируемых компилятором языка си фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM; - Скорость ассемблерных кодов x86-64 может меньше, чем аналогичных кодов x86, примерно на 10%; - Оптимизация кодов лучше работает на процессоре Intel; - Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, awk (gawk, mawk) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах; - Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (gcc, icc, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром; - В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета ask(5,2,3)

## Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016.  
URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.