Отчёт по лабораторной работе №11

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Цыганков Александр Романович, НПМБВ-02-20

Содержание

# Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

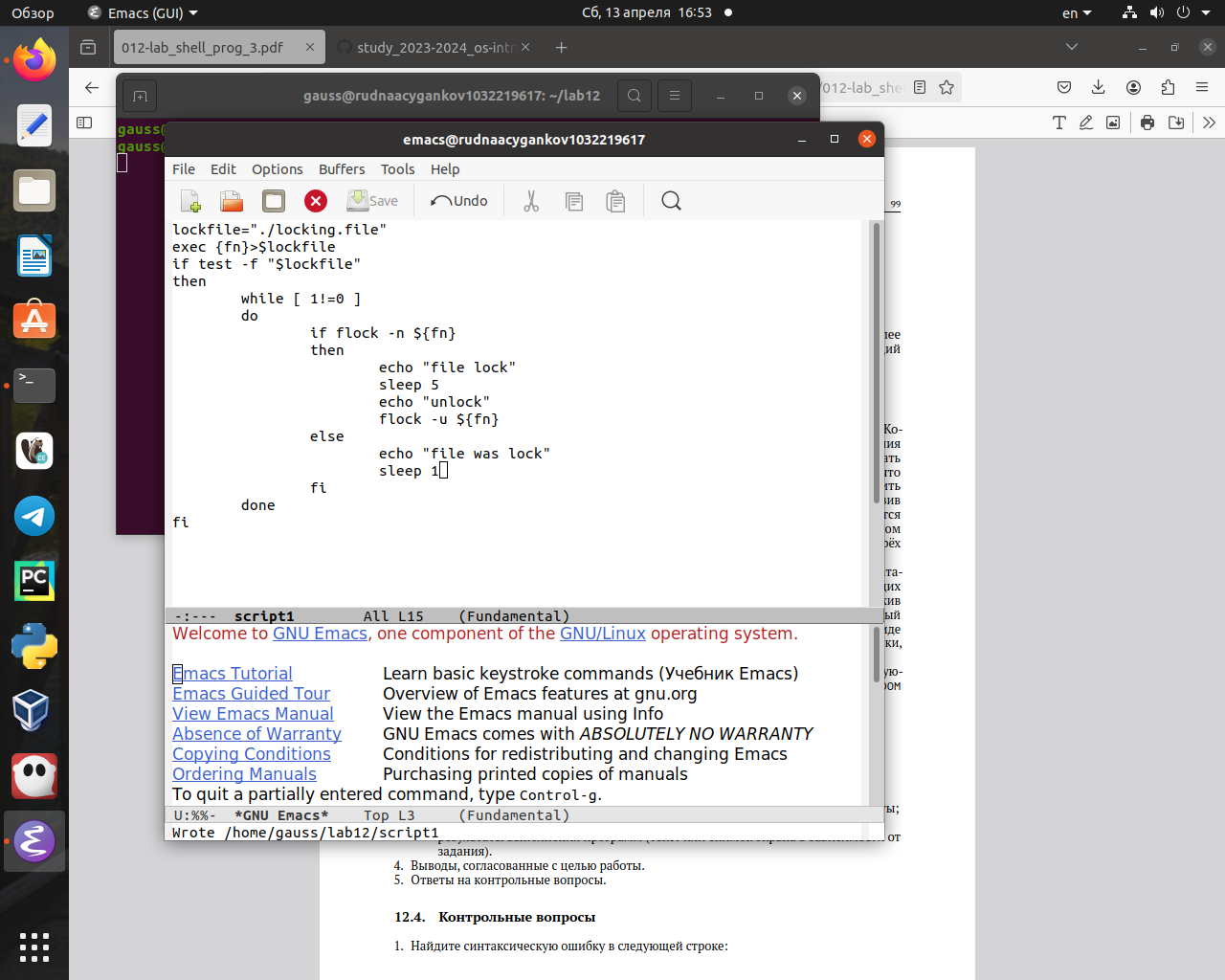
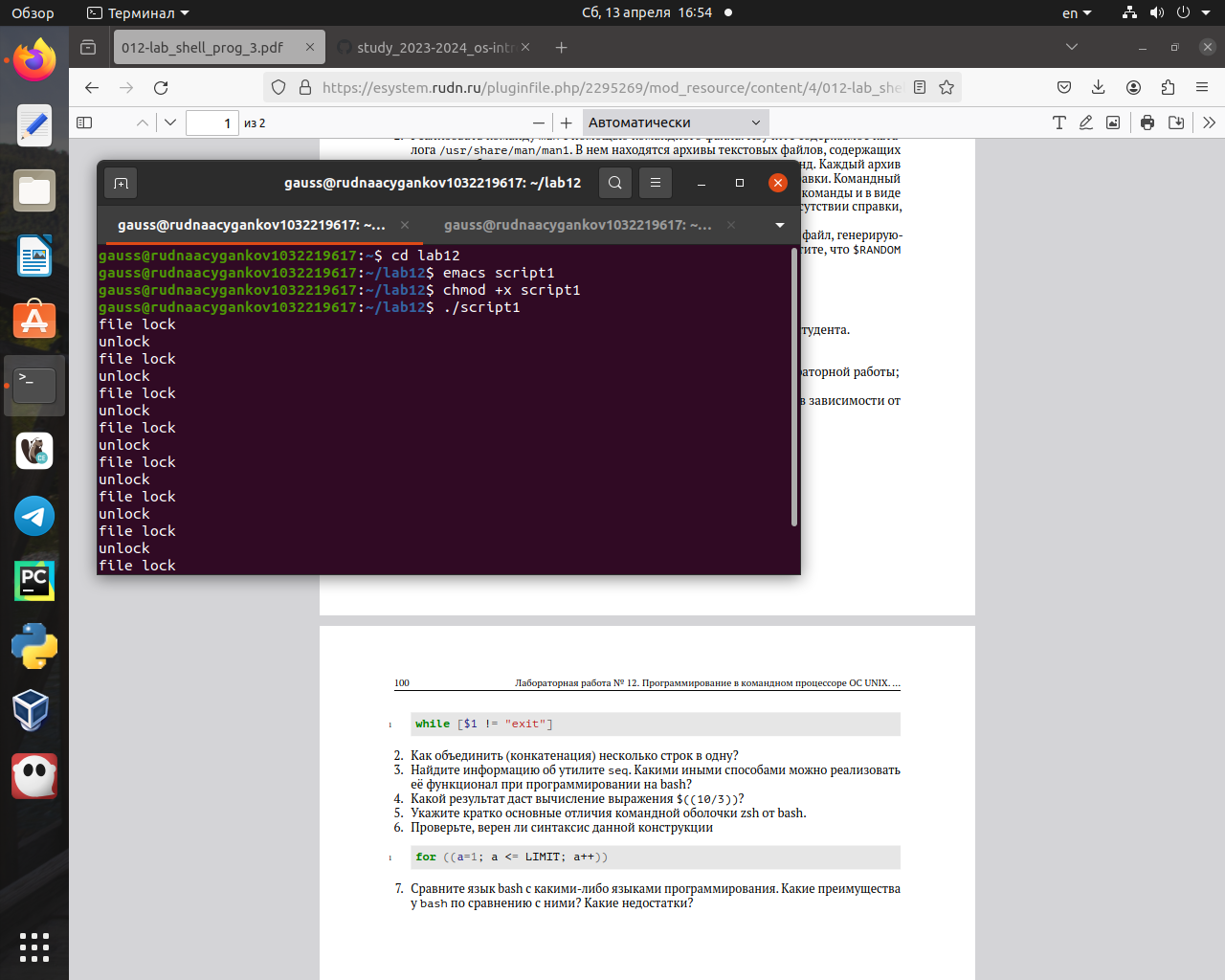
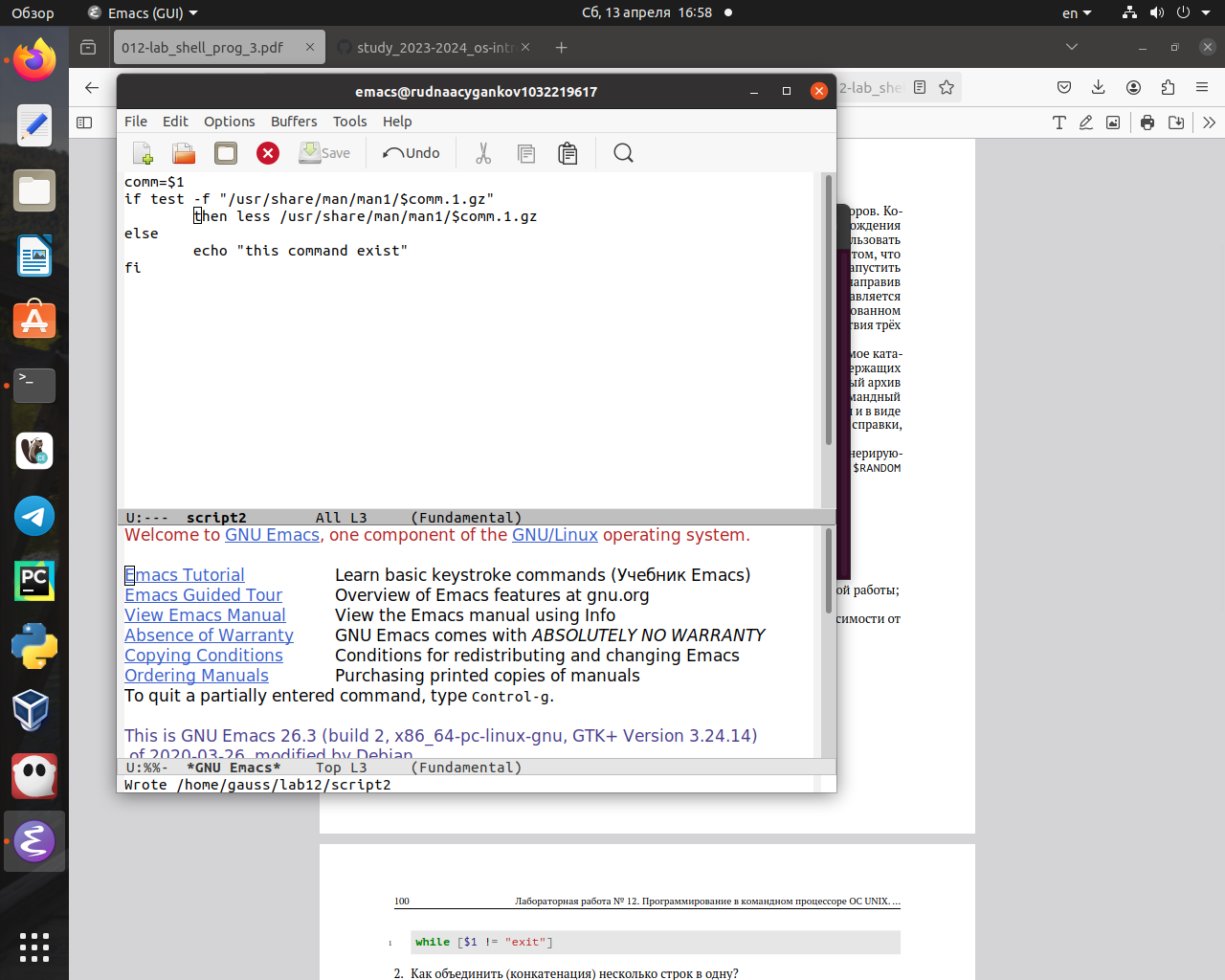
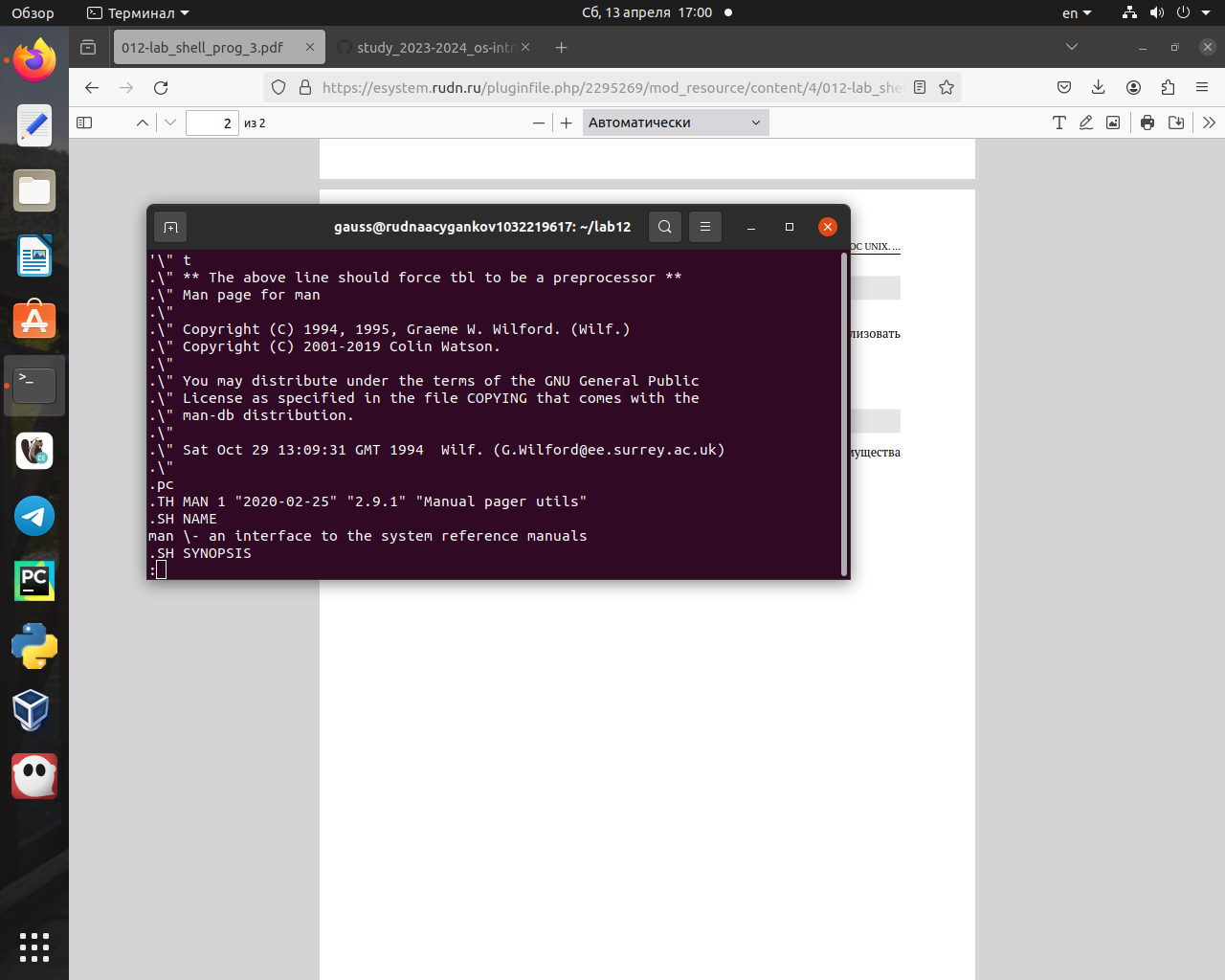
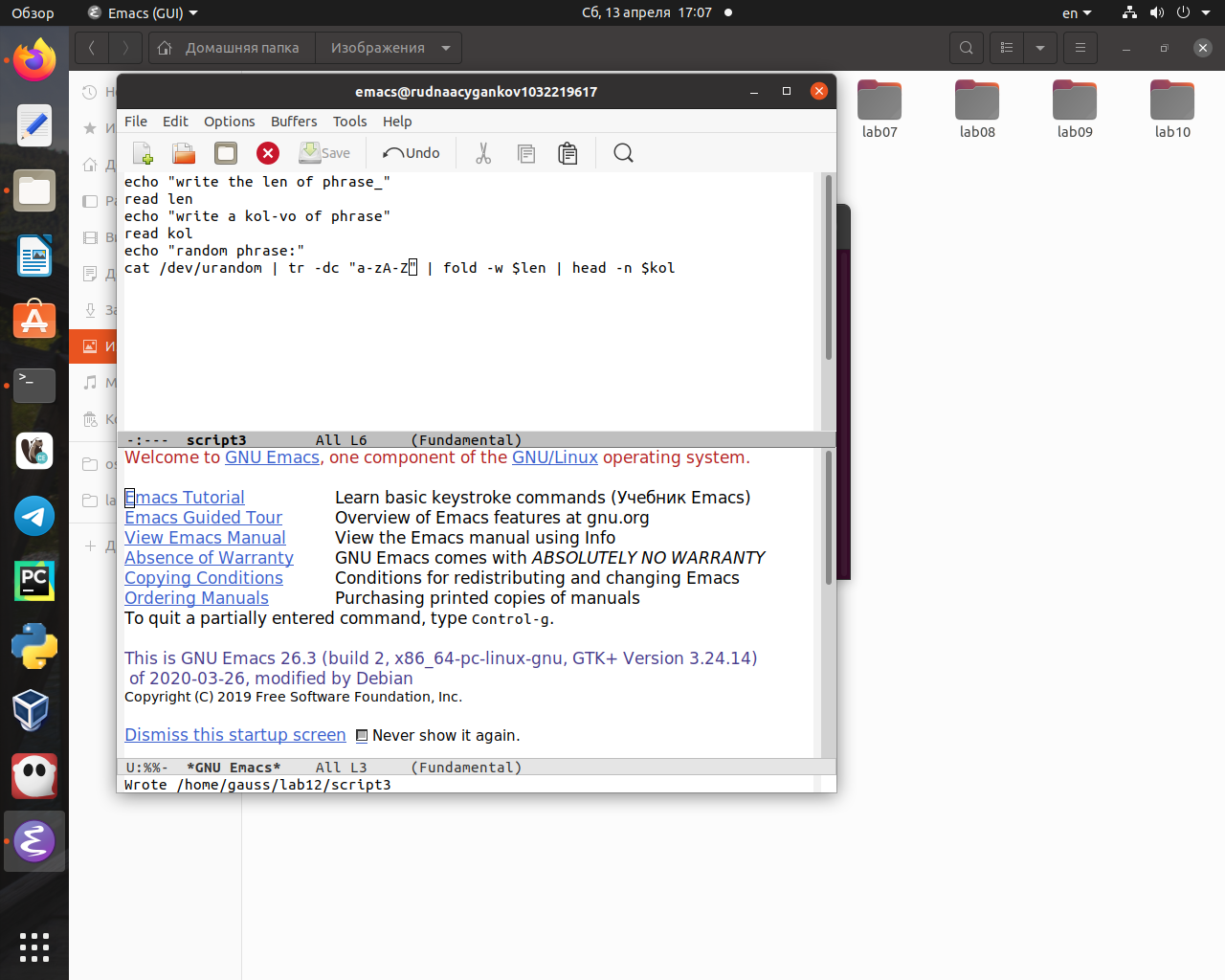
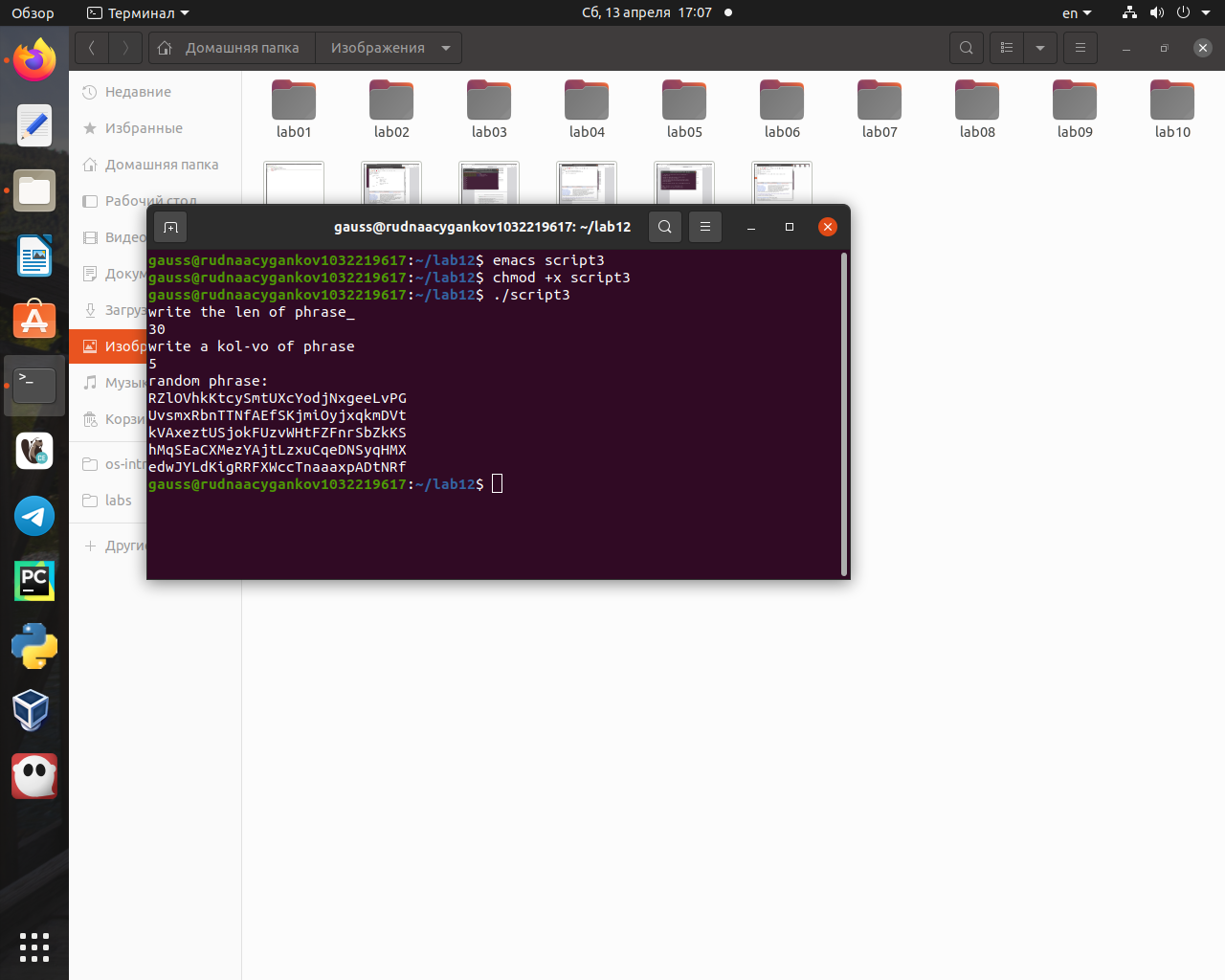
# Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. 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 до 32767.

# Теоретическое введение

* Командный процессор (командная оболочка, интерпретатор команд shell) - это программа, 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 программой совместимы с операторами оболочки Борна;
  + BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает 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 операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

# Выполнение лабораторной работы

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 (рис. @fig:001, @fig:002)
   * 
   * 
   * Результат
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. @fig:003, @fig:004)
   * 
   * 
   * Результат по команде man
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 (рис. @fig:005, @fig:006)
   * 
   * 
   * Результат

# Листинг

1. script1:

lockfile="./locking.file"  
exec {fn}>$lockfile  
if test -f "$lockfile"  
then  
 while [ 1!=0 ]  
 do  
 if flock -n ${fn}  
 then  
 echo "file lock"  
 sleep 5  
 echo "unlock"  
 flock -u ${fn}  
 else  
 echo "file was lock"  
 sleep 1  
 fi  
 done  
fi

1. script2:

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

1. script3:

echo "write the len of phrase\_"  
read len  
echo "write a kol-vo of phrase"  
read kol  
echo "random phrase:"  
cat /dev/urandom | tr -dc "a-zA-Z" | fold -w $len | head -n $kol

# Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: 1 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. Правильный вариант: while [ “$1” != “exit” ]
2. Как объединить (конкатенация) несколько строк в одну?
   * Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:
     + Первый: VAR1=“1+” VAR2=” 2” VAR3=“VAR2” echo “$VAR3”
       - Результат: 1+2
     + Второй: VAR1=“1” VAR1+=” +2 echo “$VAR1”
       - Результат: 1+2
3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?
   * Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT.
   * seq FIRST [INCREMENT] LAST: Генерирует последовательность чисел от FIRST до LAST с заданным или стандартным шагом INCREMENT.
   * seq [-w] [-f FORMAT] FIRST [INCREMENT] LAST: Аналогично предыдущему, но позволяет задать формат вывода чисел с помощью аргумента -f и дополнительно выравнивать числа по ширине с помощью аргумента -w.
   * на bash без использования seq, можно воспользоваться циклом for: for ((i=1; i<=10; i+=2)); do echo $i done
   * с использованием awk:awk BEGIN { for (i=1; i<=10; i+=2) print i }
   * с использованием perl: perl -e for ($i=1; $i<=10; $i+=2) { print "$i\n" }
4. Какой результат даст вычисление выражения $((10/3))?
   * Результатом будет 3, это целочисленное деление без остатка.
5. Укажите кратко основные отличия командной оболочки zsh от bash.
   * Автодополнение: Zsh обладает более развитой системой автодополнения, которая включает в себя автоматическое завершение имён файлов и каталогов, а также автодополнение команд и аргументов.
   * Мощные возможности расширения: Zsh предоставляет богатый набор встроенных функций и возможностей расширения, таких как темы оформления, перенаправления ввода-вывода, управление заданиями и другие.
   * Настройка и наследование: Zsh позволяет более гибко настраивать своё окружение и поведение командной оболочки. Она также поддерживает наследование настроек, что облегчает управление конфигурацией.
   * Массивы и ассоциативные массивы: Zsh поддерживает более богатый набор типов данных, включая массивы и ассоциативные массивы, что делает работу с данными более удобной.
   * Мощный синтаксис командной строки: Zsh имеет более мощный синтаксис командной строки, который включает в себя расширенные возможности обработки строк, условные выражения и циклы.
   * Совместимость с Bash: Zsh совместима с синтаксисом и скриптами Bash, что позволяет запускать большинство скриптов, написанных для Bash, без изменений.
   * Подсветка синтаксиса и подсказки по командам: Zsh предоставляет подсветку синтаксиса команд и подсказки по их использованию, что упрощает работу с командной строкой.
6. Проверьте, верен ли синтаксис данной конструкции 1 for ((a=1; a <= LIMIT; a++))
   * for ((a=1; a <= LIMIT; a++)) синтаксис верен, используя двойные круглые скобки, можно не писать $ перед переменными ().
7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?
   * Python:
     + Преимущества Bash:
     + - Простота и прямолинейность в написании коротких скриптов для автоматизации системных задач.  
        - Интеграция с системными командами и утилитами.
     + Недостатки Bash:
     + - Ограниченные возможности обработки данных и сложных структур.  
        - Отсутствие расширенных библиотек и модулей для разработки приложений.
   * Perl:
     + Преимущества Bash:
     + - Простота в написании однострочных скриптов для простых задач.  
        - Интеграция с системными командами и утилитами.
     + Недостатки Bash:
     + - Меньшая гибкость и мощь в обработке текста и регулярных выражений по сравнению с Perl.  
        - Ограниченные возможности для создания сложных приложений.

# Выводы

В ходе выполнения данной лабораторной работы я изучил расширенное программирование в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций.

# Список литературы

Руководство к лабораторной работе