Лабораторная работа №11

Дисциплина: Операционные системы

Колчева Юлия вячеславовна

Содержание

# Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

# Выполнение лабораторной работы

Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. 1) (рис. 2) (рис. 3)

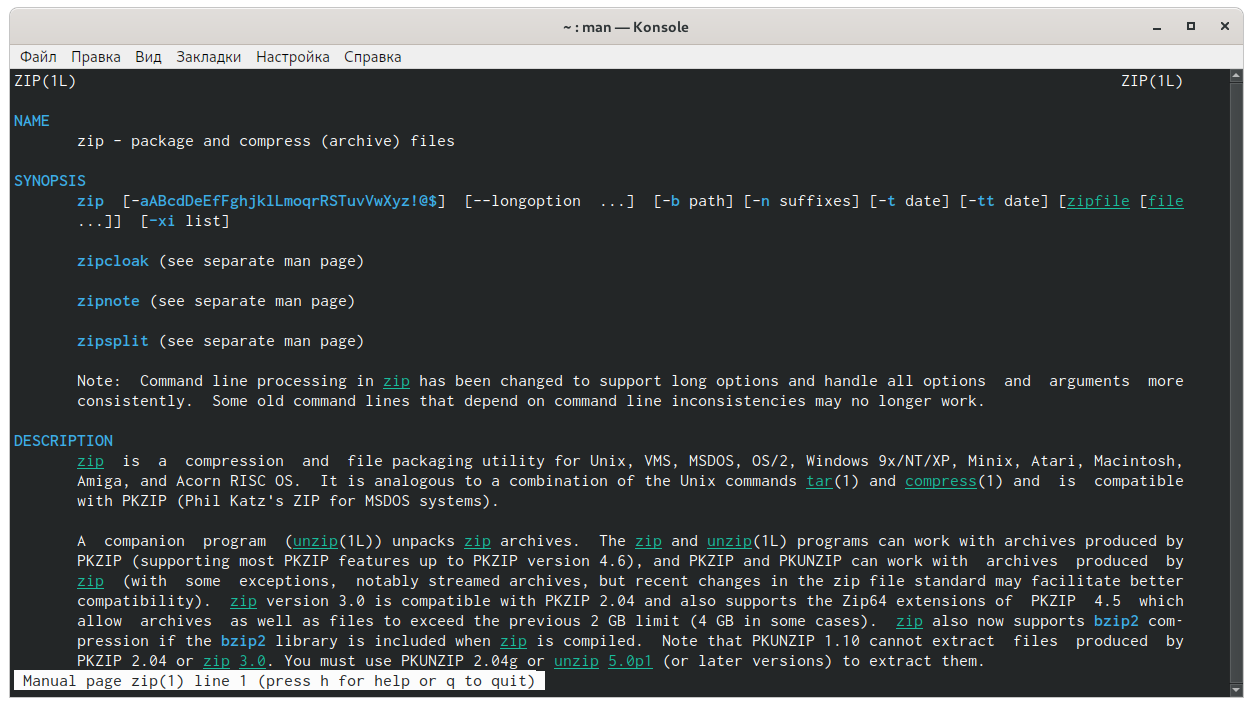


Figure 1: zip

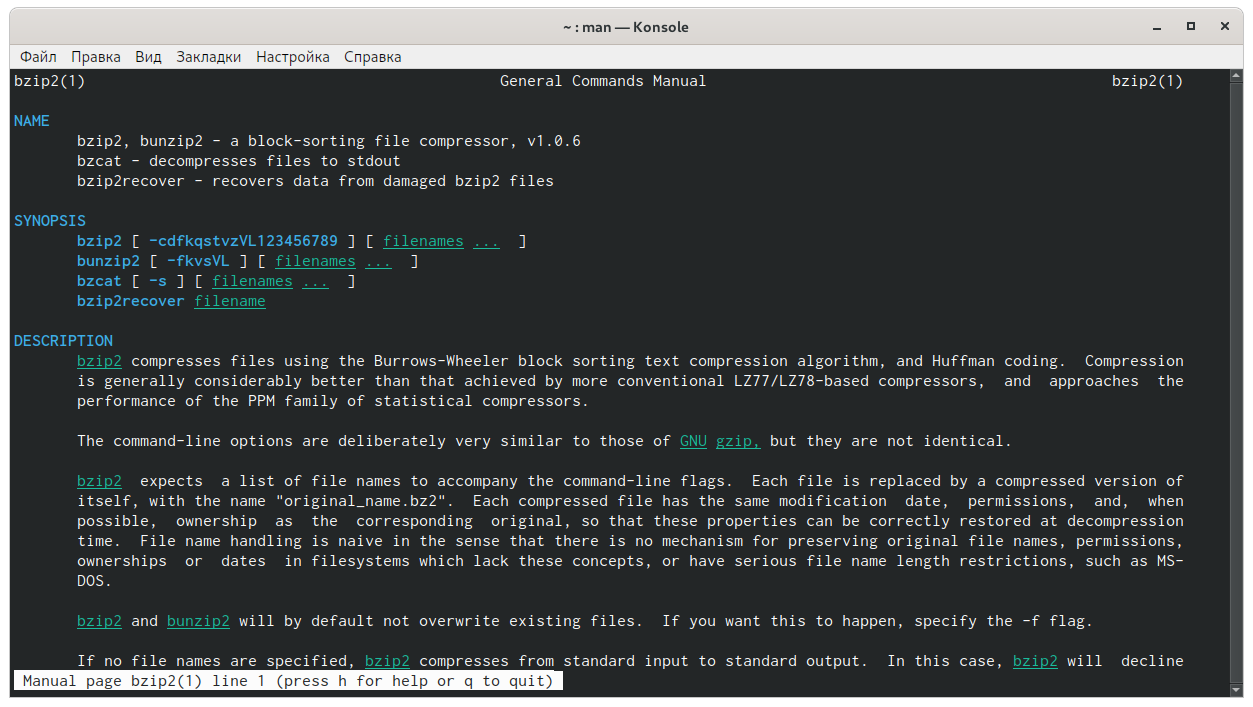


Figure 2: bzip2

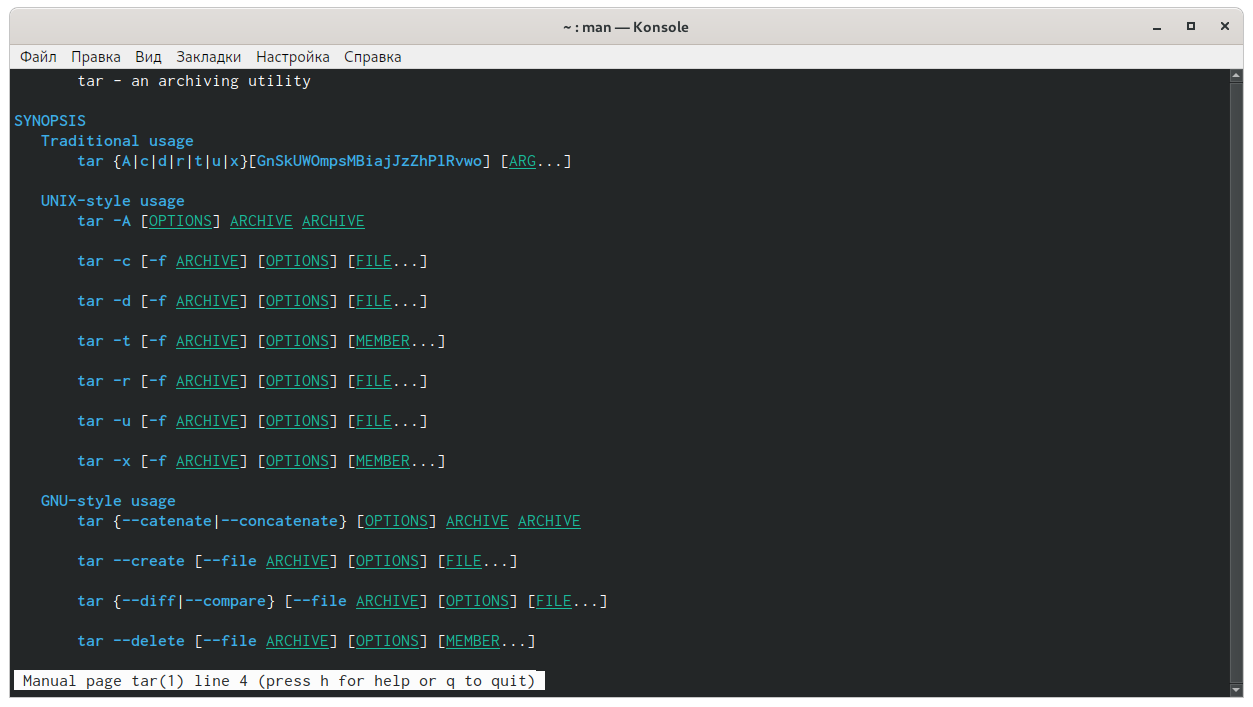


Figure 3: tar

Далее я создала файл, в котором будуписать первый скрипт, и открыла его в редакторе emacs (touch backup.sh» и emacs &) После написала скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в домашнем каталоге. При написании скрипта использовала архиватор bzip2 (рис. 4)

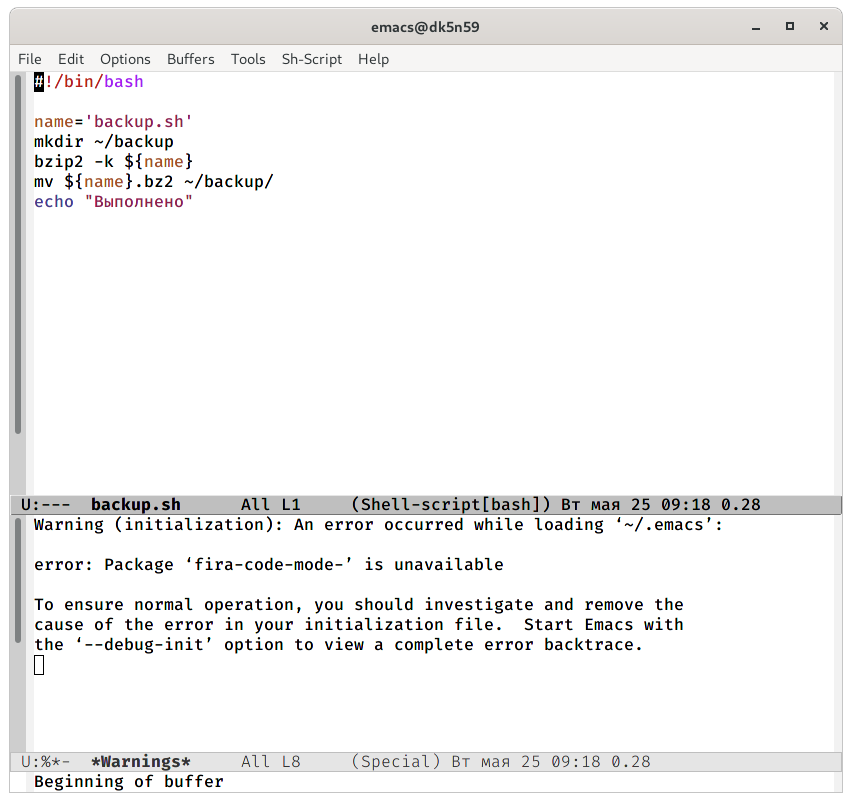


Figure 4: Первый скрипт

Проверила работу скрипта (./backup.sh), предварительно добавив для него право на выполнение (chmod+x\*.sh). Проверила, появился ли каталог backup/, перейдя в него, посмотрела его содержимое и просмотрела содержимое архива (bunzip2 -c backup.sh.bz2).Скрипт работает корректно(рис. 5)

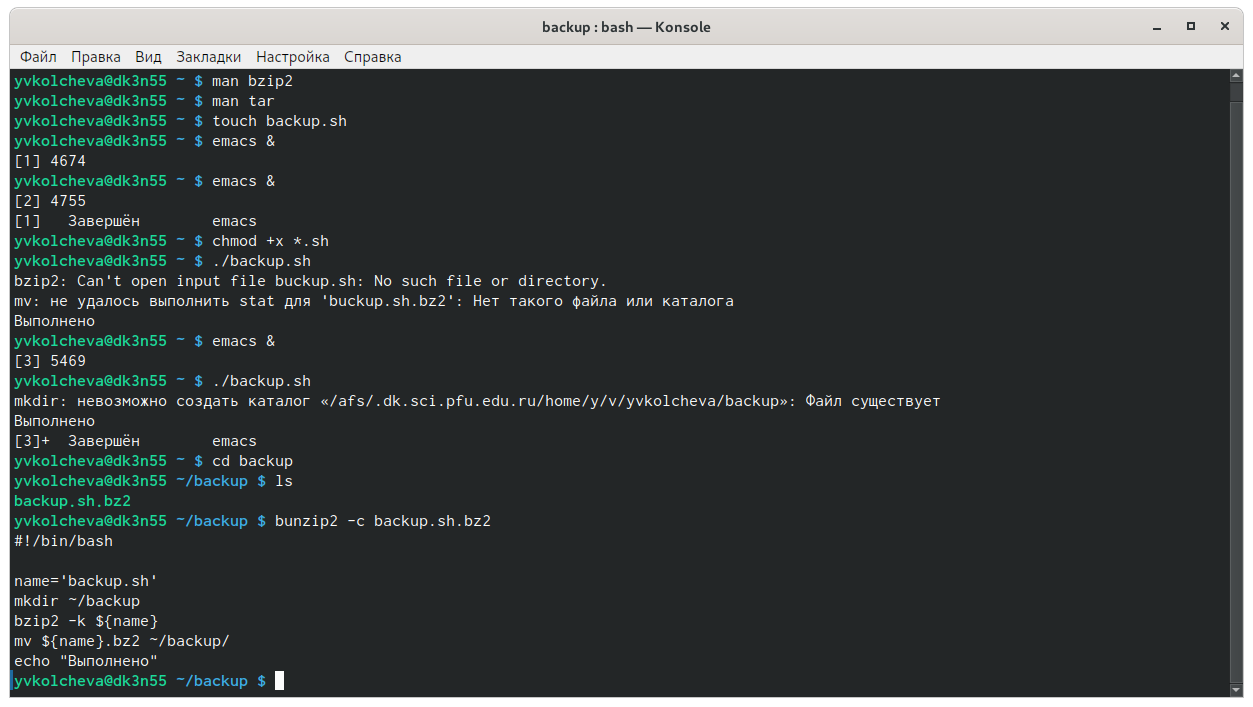


Figure 5: Проверка работы1

Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs (команды «touchp prog2.sh» и «emacs &». Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис. 6)

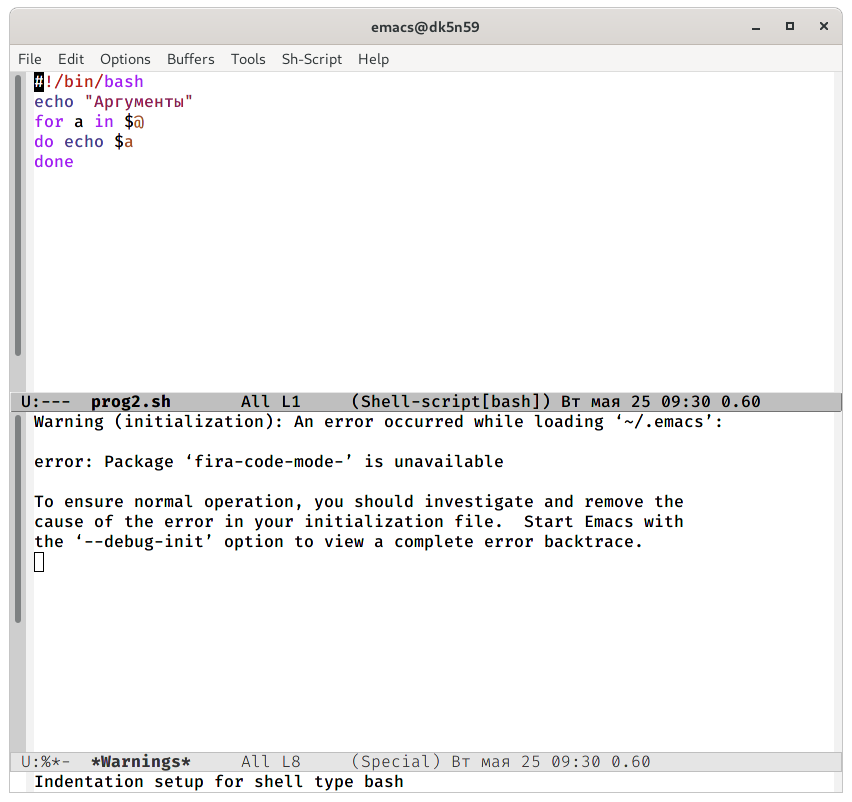


Figure 6: Второй скрипт

Проверила работу написанного скрипта (./prog2.sh0 1 2 3 4 и ./prog2.sh 0 1 2 3 45 6 7 8 9 10), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh»).Скрипт работает корректно.(рис. 7)

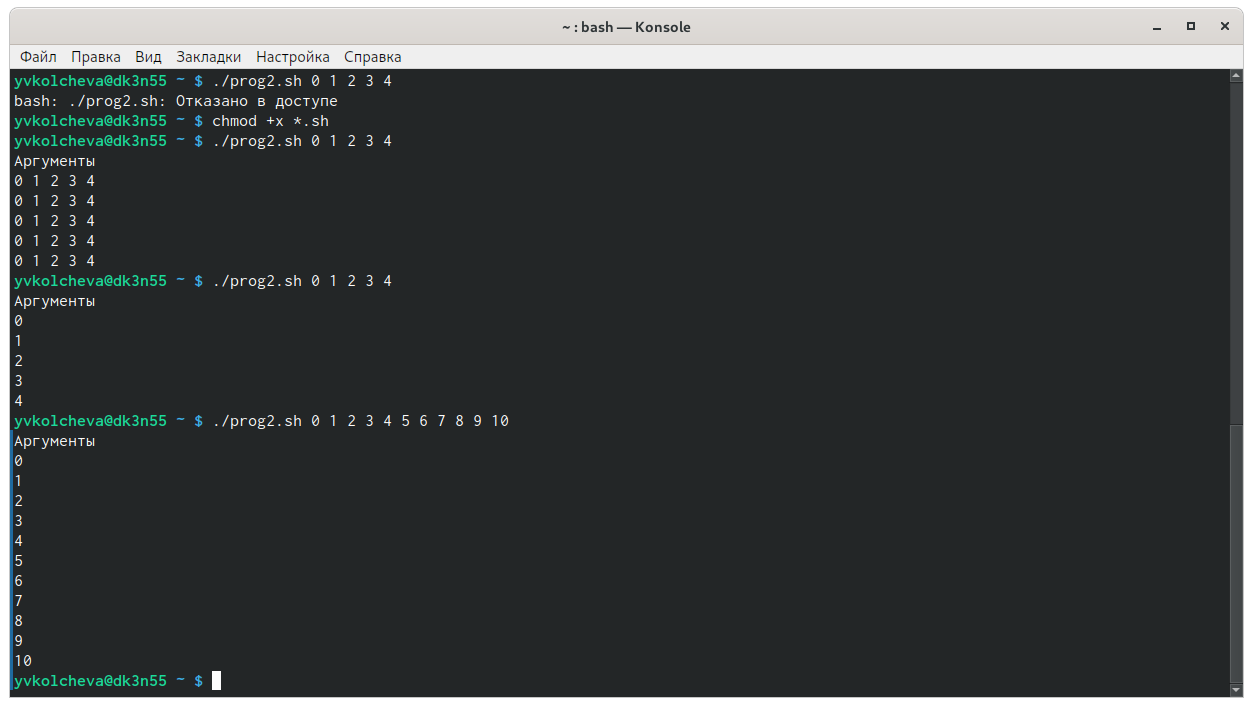


Figure 7: Проверка работы2

Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs (touch progls.sh и emacs&) Написала командный файл−аналог команды ls. (рис. 8)

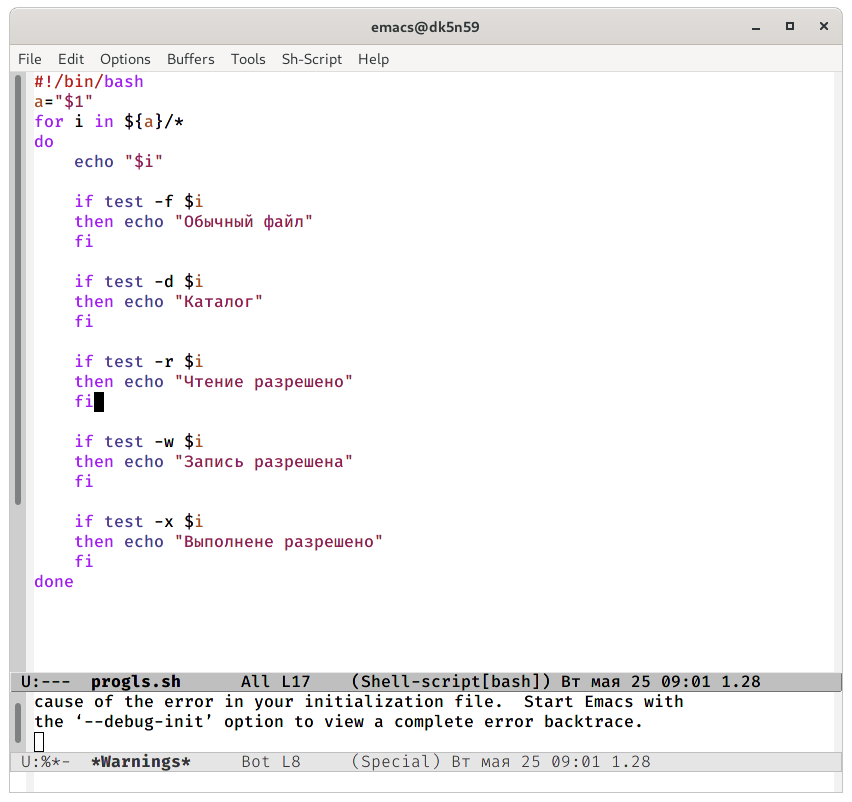


Figure 8: Третий скрипт

Далее проверила работу скрипта(./progls.sh ~), предварительно добавив для него право на выполнение (команда «chmod+x\*.sh»). Скрипт работает корректно. (рис. 9)

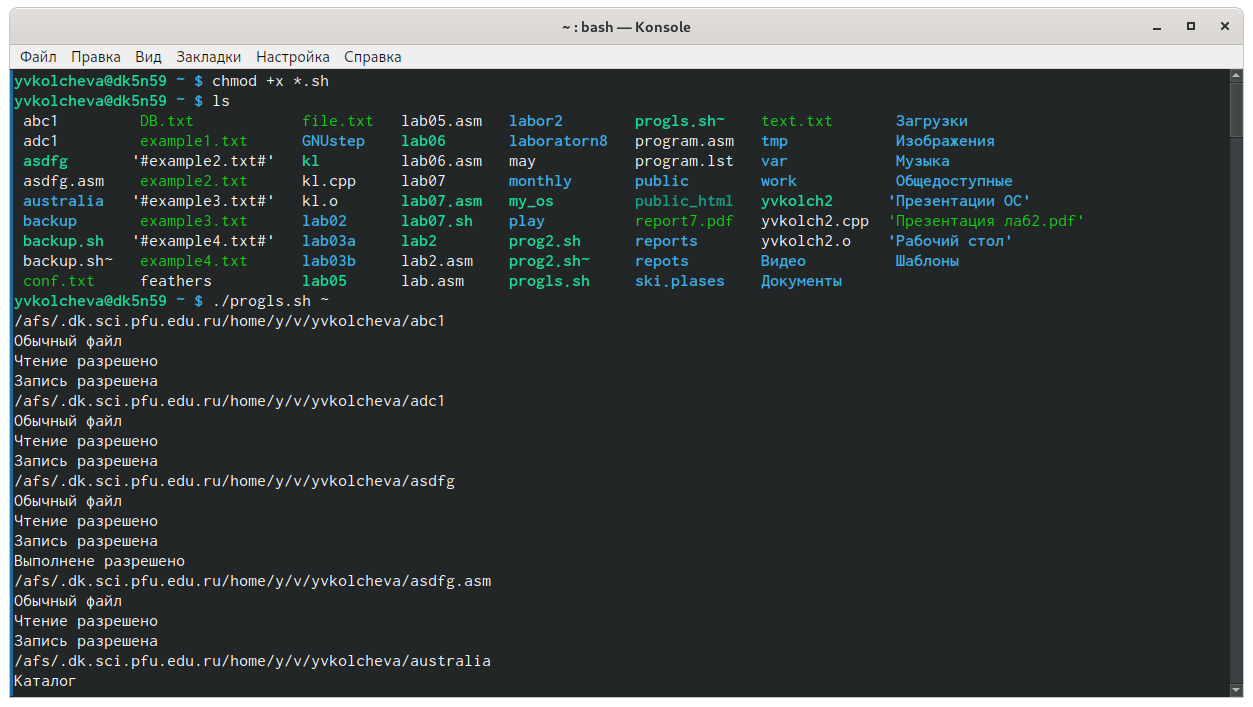


Figure 9: Проверка работы3

Для четвертого скрипта также создала файл (touch format.sh) и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs&»). Написала командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории. Путь кдиректории также передаётся в виде аргумента командной строки (рис. 10)

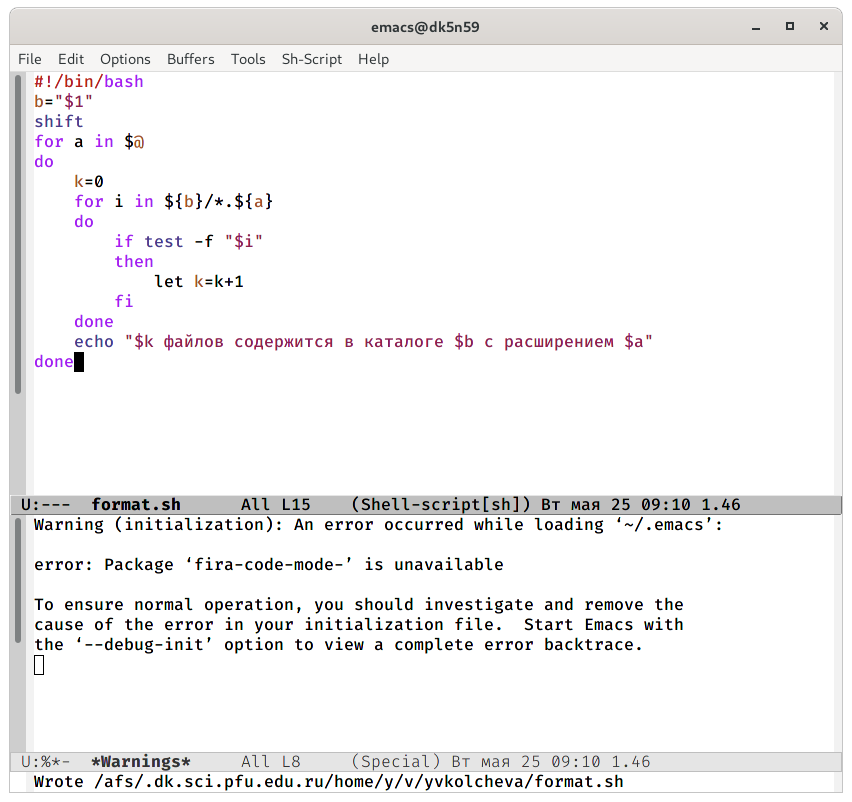


Figure 10: Четвертый скрипт

Проверила работу написанного скрипта (команда «./format.sh ~ pdf sh txt doc»),предварительно добавив для него право на выполнение (chmod+x\*.sh), а также создав дополнительные файлы с разными расширениями.Скрипт работает корректно. (рис. 11)

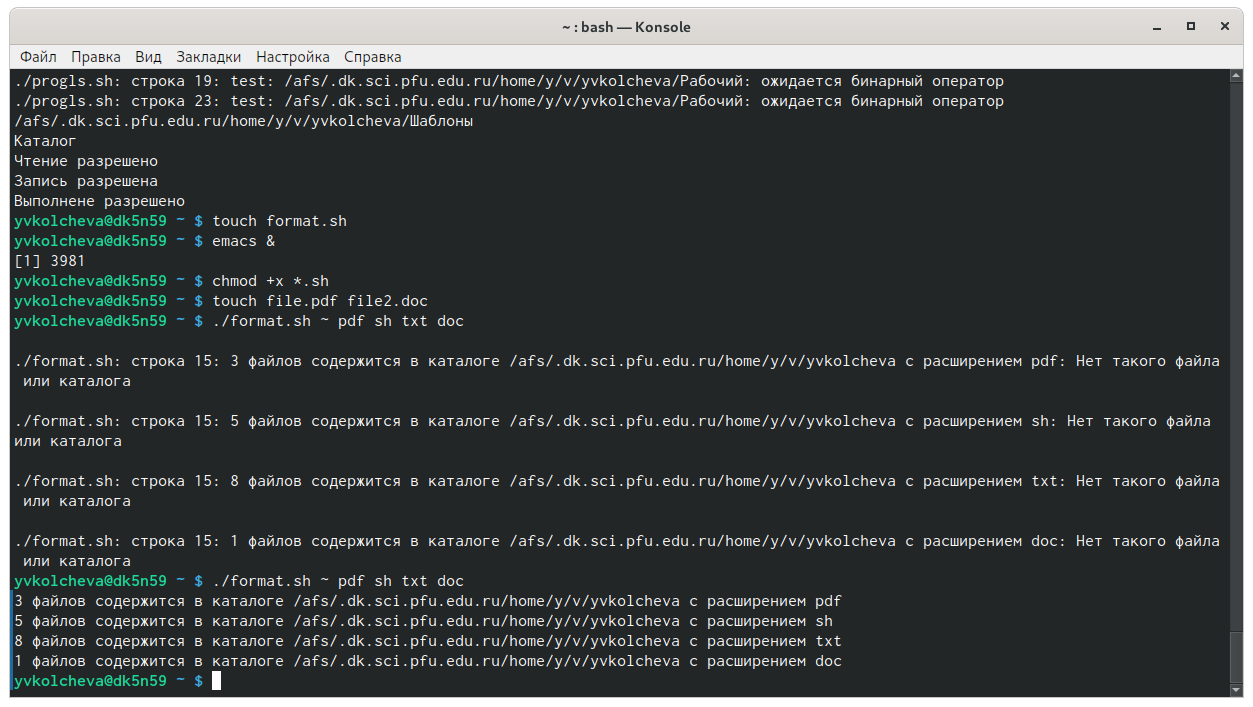


Figure 11: Проверка работы4

# Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

# Контрольные вопросы

1)Командный процессор (командная оболочка, интерпретатор команд shell) −это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linuxнаиболее часто используются следующие реализации командных оболочек: оболочка Борна (Bourneshellили sh) −стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; С-оболочка (или csh) −надстройка на оболочкой Борна, использующая Сподобный 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 BourneAgainShell(опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании FreeSoftwareFoundation).2)POSIX(PortableOperatingSystemInterfaceforComputerEnvironments) − набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIXразработаны комитетом IEEE(InstituteofElectricalandElectronicsEngineers) для обеспечения совместимости различных UNIX/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. 3)Командный процессор bashобеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «mark=/usr/andy/bin»присваивает значение строки символов /usr/andy/binпеременной markтипа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол {mark}»переместит файл afileиз текущего каталога в каталог с абсолютным полным именем /usr/andy/bin.Оболочка bashпозволяет работать с массивами. Для создания массива используется команда setс флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set-AstatesDelawareMichigan“NewJersey”»Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента. 4)Оболочка bashподдерживает встроенные арифметические функции. Команда letявляется показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение −это единичный терм (term), обычно целочисленный.Команда letберет два операнда и присваивает их переменной.Команда readпозволяет читать значения переменных со стандартного ввода: «echo “Please enter Month and Day of Birth ?”»«readmondaytrash»В переменные monи dayбудут считаны соответствующие значения, введённые с клавиатуры, а переменная trashнужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её. 5)В языке программирования bashможно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). 6)В (( ))можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат. 7)Стандартные переменные: PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневогоили текущего каталога. PS1 и PS2:эти переменные предназначены для отображения промптера командного процессора. PS1 − это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >. HOME: имя домашнего каталога пользователя. Если команда cdвводится без аргументов, то происходит переход в каталог,указанный в этой переменной. IFS:последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline). MAIL:командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение Youhavemail(у Вас есть почта). TERM: тип используемого терминала. LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. 8)Такие символы, как ’ < >*  ? | " &, являются метасимволами и имеют для командного процессора специальный смысл. 9)Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ". Например, –echo\* выведет на экран символ *, –echoab’*|*’cdвыведет на экран строку ab*|*cd. 10)Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: «bashкомандный\_файл [аргументы]»Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды «chmod+xимя\_файла»Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будтоон является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию. 11)Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unsetcфлагом -f. 12)Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «test-f путь до файла»(для проверки, является ли обычным файлом)и «test -d[путь до файла]»(для проверки, является ли каталогом). 13)Команду«set»можноиспользовать для вывода списка переменных окружения. В системах Ubuntuи Debianкоманда«set»также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду«set| more».Команда «typeset» предназначена для наложения ограничений на переменные.Команду«unset»следует использовать для удаления переменной из окружения командной оболочки. 14)При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ $ является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов $i, где 0 < i< 10, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т.е. аргумента командного файла с порядковым номером i. Использование комбинации символов $0 приводит к подстановке вместо неё имени данного командного файла. 15)Специальные переменные: $* −отображается вся командная строка или параметры оболочки; $? −код завершения последней выполненной команды; $$ −уникальный идентификатор процесса, в рамках которого выполняется командный процессор; $! −номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; $-−значение флагов командного процессора; ${#*} −возвращает целое число −количествослов, которые были результатом $*; ${#name} −возвращает целое значение длины строки в переменной name; ${name[n]} −обращение к n-му элементу массива; ${name[\*]}−перечисляет все элементы массива, разделённые пробелом; ${name[@]}−то же самое, но позволяет учитывать символы пробелы в самих переменных; ${name:-value} −если значение переменной name не определено, то оно будет заменено на указанное value; ${name:value} −проверяется факт существования переменной; ${name=value} −если name не определено, то ему присваивается значение value; ${name?value} −останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке; ${name+value} −это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value; ${name#pattern} −представляет значение переменной name с удалённым самым коротким левым образцом (pattern); ${#name[\*]} и ${#name[@]}−эти выражения возвращают количество элементов в массиве name

# Библиография

Лекция Кудрявцева https://esystem.rudn.ru/mod/url/view.php?id=718563

Лабораторная работа №11 https://esystem.rudn.ru/mod/resource/view.php?id=718604