Отчёт к лабораторной работе №11

абораторная работа № 12. Программирование в командномпроцессоре ОС UNIX. Ветвления и циклы.

Кекишева Анастасия Дмитриевна

Содержание

# Цель работы

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

# Задание

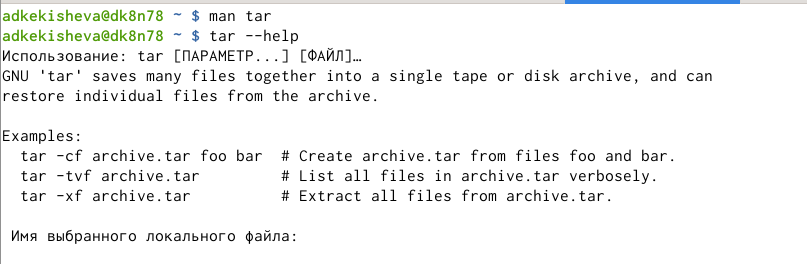
**Выполнить данныые пункты и ответить на вопросы:**

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию back up в вашем домашнем каталоге. При этом файл должен архивироваться од-ним из архиваторов на выбор zip,bzip2 или tar. Способ использования командархивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольноечисло аргументов командной строки, в том числепревышающеедесять. Напри-мер, скрипт может последовательно распечатывать значения всех переданныхаргументов.
3. Написать командный файл — аналог команды ls (без использования самой этойкоманды и команды dir). Требуется, чтобы он выдавал информацию о нужномкаталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента команднойстроки формат файла (.txt,.doc,.jpg,.pdfи т.д.) и вычисляет количествотаких файлов в указанной директории. Путь к директории также передаётся ввиде аргумента командной строки.

# Выполнение лабораторной работы

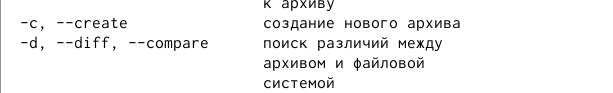
Перед выполнением лабораторной работы я хорошо ознакомилась с теоритическим материалом для её выполнения [Ссылка 1](https://vk.com/im?peers=311102850&sel=c80)

## Выполнение 1-го пункта задания

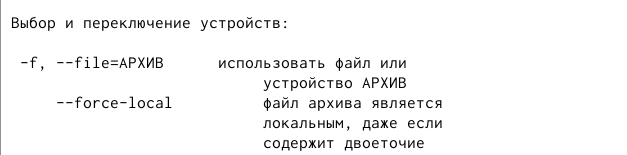


Справка tar

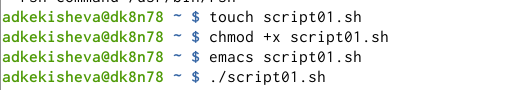
Во-первых, я изучила справку по способу использования tar. (рис. @fig:001) И нашла необходимые опции для выполнения задания (рис. @fig:002) (рис. @fig:003) :



Опция -c - создание

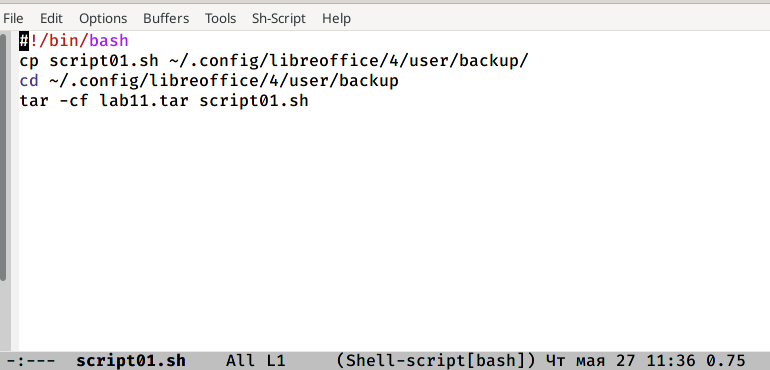


Опция -f - применеие файла или устройство АРХИВ



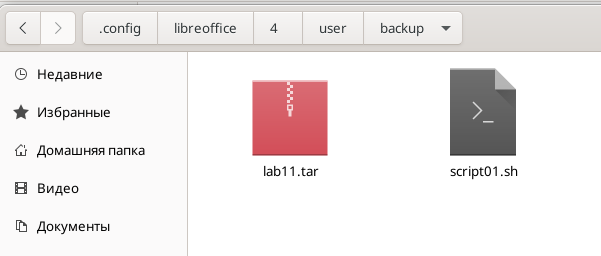
Создание файла, присвоение прав доступа

Для написания первого скрипта, я создала файл с раширение .sh, так как я буду работать в emacs. Наделила это файл правом на выполнения и присткпила к написанию. (рис. @fig:004)



Написание скрипта

Написала скрипт (рис. @fig:005) , который при запуске будет делать резервную копию самого себя в директорию backup. Для это я нашла путь к каталогу backup. В первой строке мы обязательно пишем #!/bin/bash. Далее применила команду копирования cp и скопировала скрипт в каталог backup. Далее перешла в этот каталог и после командой архивирования tar с опциями -cf сделала архив файла со скриптом с именем lab11.tar.



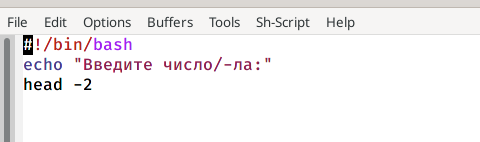
Результат

Запустила скрипт ./(название файла) (рис. @fig:004) и проверила результат (рис. @fig:006)

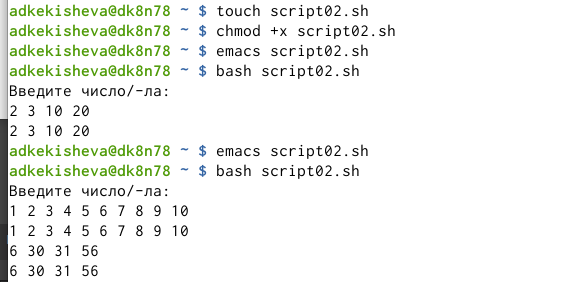
## Выполнение 2-го пункта задания

Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. На мой взляд, это можно сделать двумя способами:

1. Воспользоваться head -(количество строк для обработки) (рис. @fig:007) (рис. @fig:008).

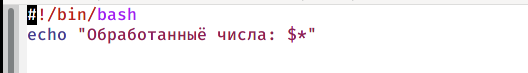


Командный файл, обрабатывающий любое произвольное число аргументов командной строки №1

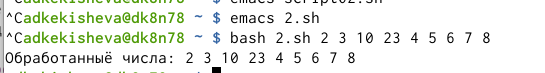


Результат работы head

1. Ввести $\*, что означает что вызывая командый файл нам не выйдет предложение ввести символы, их мы будем должны ввести сами в командной строке (рис. @fig:009) (рис. @fig:010).

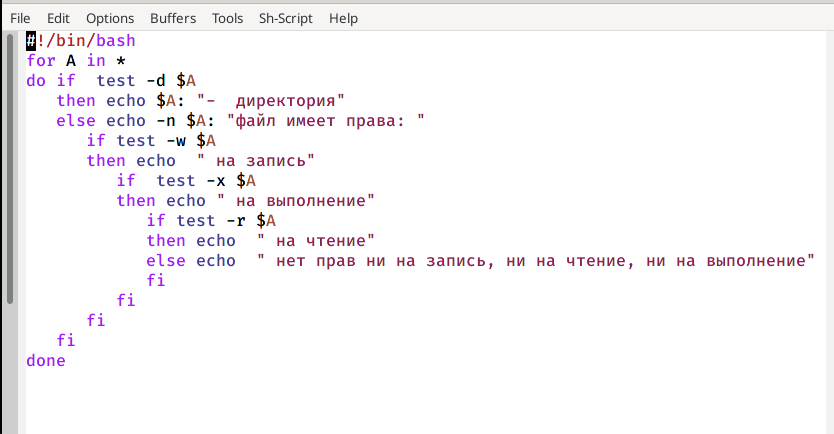


Командный файл, обрабатывающий любое произвольноечисло аргументов командной строки №2



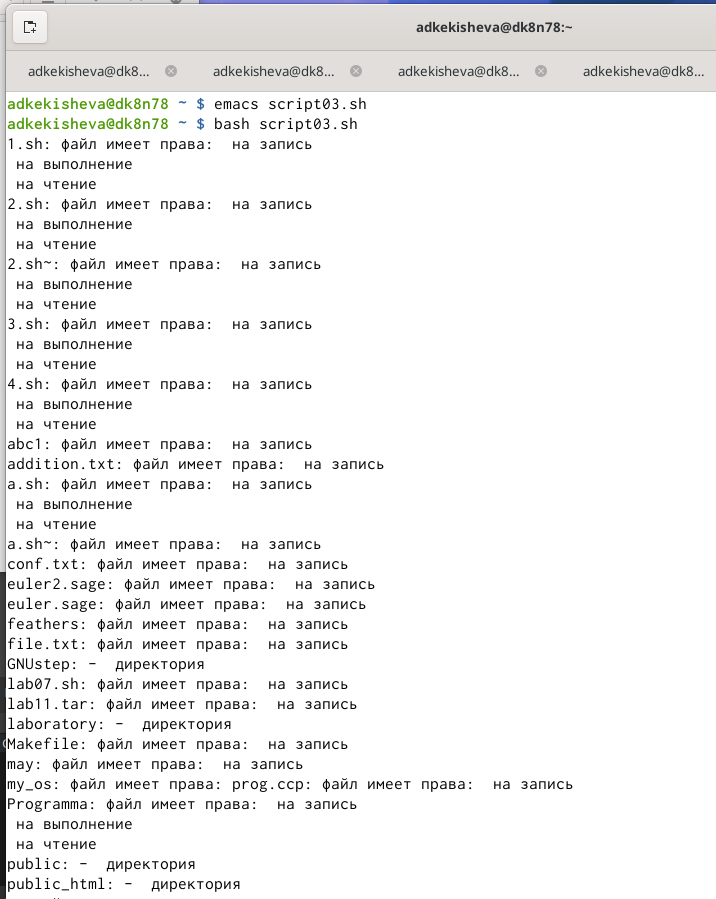
Результат работы S\*

## Выполнение 3-го пункта задания



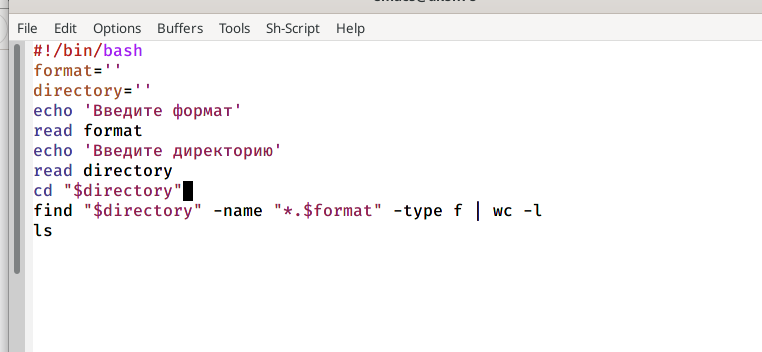
Командый файл, выводящий информацию о правах доступа

Далее написала командный файл — аналог команды ls (без использования самой этойкоманды и команды dir), который выдаёт информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога (рис. @fig:011). Для это я воспользовалась циклом for в котором переменная А будет принимать значения, равные именам этих файлов. И потом в помощью операторов условия if проверяем файл это или директория. Если директория просто выводим, что это директория, если файл, то проверяем его, на наличие прав на чтение test -r, на запись -w, на выполнение -x, в противном случае у файла нет прав. Test проверяет на истину. (рис. @fig:012)

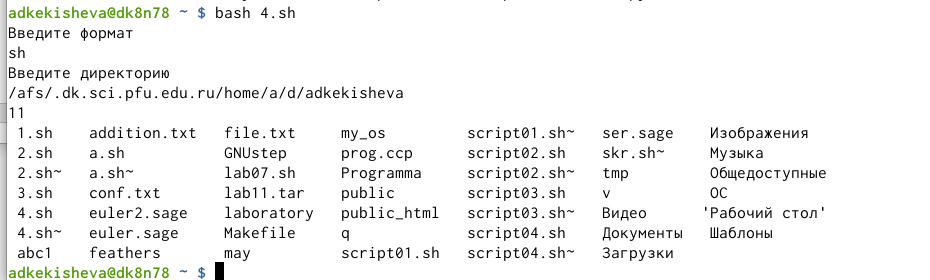


Результат выполнения командного файла

## Выполнение 4-го пункта задания

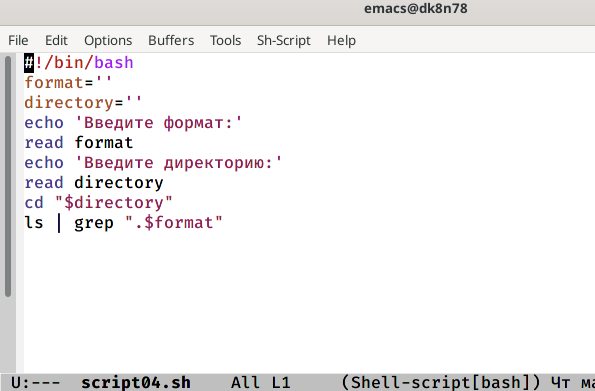


Командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории

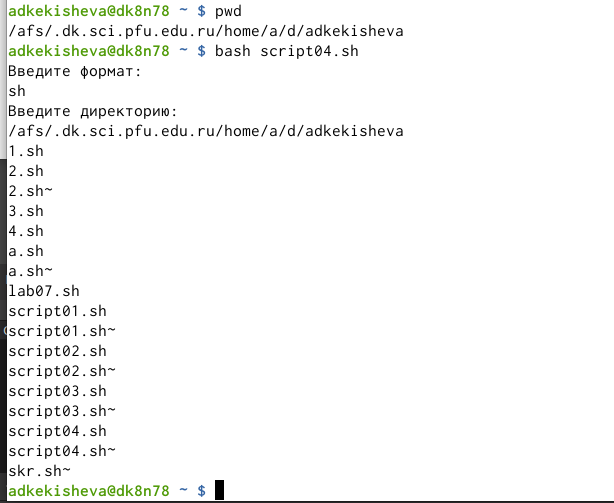


Результат выполнения командного файла

Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt,.doc,.jpg,.pdfи т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся ввиде аргумента командной строки (рис. @fig:013) (рис. @fig:014) . Здесь, я создала переменные, считывала их командой read и применяла команду find для поиска файлов с определённым расширением, а также конвеер и команду wc с опцией -l. Команда wc -l выводит количество строк в объекте.[Ссылка 2](https://losst.ru/komanda-wc-v-linux)



Командный файл, который получает в качестве аргумента командной строки формат файла и выводит их в нужной директории



Результат выполнения командного файла

Также в начале работы у меня получалось так, что командный файл выводил имена файлов, но он не вычислял количество файлов (рис. @fig:015) (рис. @fig:016)

# Вывод

Я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать командные файлы и скрипты.

# Библиография

1. [Ссылка 1](https://vk.com/im?peers=311102850&sel=c80)
2. [Ссылка 2](https://losst.ru/komanda-wc-v-linux)

**Контрольные вопросы:**

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командная оболочка— 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 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 Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

1. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

1. Как определяются переменные и массивы в языке программирования bash?

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов.

Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, set -A states Delaware Michigan “New Jersey”.

1. Каково назначение операторов let и read?

Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Команда let берет два операнда и присваивает их переменной. Положительным моментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара;

Команда read позволяет читать значения переменных со стандартного ввода.

1. Какие арифметические операции можно применять в языке программирования bash?

Сложение, вычитание, умножение, деление, а также эти же операции с присваиванием значения переменной, также операция отрицания, остаток от деления, побитовый сдвиг, побитовое дополнение и другие.

1. Что означает операция (( ))?

Операция применяется для записи условия. Далее, можно осуществлять присвоение результатов условных выражений переменным, также как и использовать результаты арифметических вычислений в качестве условий.

1. Какие стандартные имена переменных Вам известны?

-PATH (т.е. $РАТН) - 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 /.

– HOME — имя домашнего каталога пользователя. Если команда cd вводится 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 (new line).

– 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 You have mail (у Вас есть почта).

– TERM — тип используемого терминала.

– LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

1. Что такое метасимволы?

Такие символы, как ’ < > \* ? |  " &, являются метасимволами и имеют для командного процессора специальный смысл.

1. Как экранировать метасимволы?

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 \*,

– echo ab’*|*’cd выведет на экран строку ab*|*cd

1. Как создавать и запускать командные файлы?

Командный файл – это файл, в который помещена последовательность команд. Сначала командный файл можно выполнить по команде: bash командный\_файл [аргументы] и чтобы не вводить каждый раз последовательности символов bash, необходимо обеспечить доступ к этому файлу на выполнение (chmod +x имя\_файла). Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как-будто он является выполняемой программой.

1. Как определяются функции в языке программирования bash?

Для опредения функции в bash используется ключевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки.

1. Каким образом можно выяснить, является файл каталогом или обычным файлом?
2. Каково назначение команд set, typeset и unset?

Команда set- создание массива, используется команда set с флагом -A

Команда unset c флагом -f – удаление функции.

Команда typeset имеет четыре опции для работы с функциями:

* f — перечисляет определённые на текущий момент функции;
* ft — при последующем вызове функции инициирует её трассировку;
* fx — экспортирует все перечисленные функции в любые дочерние программы оболочек;
* fu — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

1. Как передаются параметры в командные файлы?

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.

1. Назовите специальные переменные языка bash и их назначение.

Специальных 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 — количество слов, 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.