отчёт по лабораторной работе 9

Программирование в командном процессоре ОС UNIX. Командные файлы

Руденко Михаил Андреевич

Содержание

# 1 Цель работы

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

# 2 Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

# 3 Теоретическое введение

10.2.1. Командные процессоры (оболочки) 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболоч- ках большинство команд будет совпадать с описанными ниже. 10.2.2. Переменные в языке программирования bash Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользо- ватель имеет возможность присвоить переменной значение некоторой строки символов. Например, команда 1 mark=/usr/andy/bin присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использо- вано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол $. Например, команда 1 mv afile ${mark} Кулябов Д. С. и др. Операционные системы 83 переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Использование значения, присвоенного некоторой переменной, называется подстанов- кой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: 1 ${имя переменной} Например, использование команд 1 b=/tmp/andy- 2 ls -l myfile > bls приведёт к подстановке в команд- ную строку значения переменной bls. Если переменной bls не было предварительно присвоено никакого значения, то её значением будет символ пробела. Оболочка bash позволяет работать с массивами. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, 1 set -A states Delaware Michigan “New Jersey” Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента. 10.2.3. Использование арифметических вычислений. Операторы let и read Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате типа radix#number, где radix (основание системы счисления) — любое чис- ло не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда let берет два операнда и присваивает их переменной. Положительным мо- ментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа let sum=x+7, и let будет искать переменную x и добавлять к ней 7. Команда let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор let-операций. Подобно С оболочка bash может присваивать переменной любое значение, а произволь- ное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для 84 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … Таблица 10.1 Арифметические операторы оболочки bash Оператор Синтаксис Результат ! !ехр Если ехр равно 0, то возвращает 1; иначе 0 != ехр1 !=ехр2 Если ехр1 не равно ехр2, то возвращает 1; иначе 0 % ехр1%ехр2 Возвращает остаток от деления ехр1 на ехр2 %= var=%exp Присваивает остаток от деления var на ехр переменной var & ехр1&ехр2 Возвращает побитовое AND выражений ехр1 и ехр2 && ехр1&&ехр2 Если и ехр1 и ехр2 не равны нулю, то возвращает 1; иначе 0 &= var &= ехр Присваивает переменной var побитовое AND var и ехр*  ехр1 \* ехр2 Умножает ехр1 на ехр2 *= var* = ехр Умножает ехр на значение переменной var и присваивает результат переменной var + ехр1 + ехр2 Складывает ехр1 и ехр2 += var += ехр Складывает ехр со значением переменной var и результат присваивает переменной var - -exp Операция отрицания exp (унарный минус) - expl - exp2 Вычитает exp2 из exp1 -= var -= exp Вычитает exp из значения переменной var и присваивает результат переменной var / exp / exp2 Делит exp1 на exp2 /= var /= exp Делит значение переменной var на exp и присваивает ре- зультат переменной var < expl < exp2 Если exp1 меньше, чем exp2, то возвращает 1, иначе возвра- щает 0 << exp1 << exp2 Сдвигает exp1 влево на exp2 бит <<= var <<= exp Побитовый сдвиг влево значения переменной var на exp <= expl <= exp2 Если exp1 меньше или равно exp2, то возвращает 1; иначе возвращает 0 = var = exp Присваивает значение exp переменной var == exp1==exp2 Если exp1 равно exp2, то возвращает 1; иначе возвращает 0 > exp1 > exp2 1, если exp1 больше, чем exp2; иначе 0 >= exp1 >= exp2 1, если exp1 больше или равно exp2; иначе 0 >> exp >> exp2 Сдвигает exp1 вправо на exp2 бит >>= var >>=exp Побитовый сдвиг вправо значения переменной var на exp ^ exp1 ^ exp2 Исключающее OR выражений exp1 и exp2 ^= var ^= exp Присваивает переменной var побитовое XOR var и exp | exp1 | exp2 Побитовое OR выражений exp1 и exp2 |= var |= exp Присваивает переменной var результат операции XOR var и exp || exp1 || exp2 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0 ~ ~exp Побитовое дополнение до exp облегчения программирования можно записывать условия оболочки bash в двойные скобки — (( )). Можно присваивать результаты условных выражений переменным, также как и исполь- зовать результаты арифметических вычислений в качестве условий. Хорошим примером Кулябов Д. С. и др. Операционные системы 85 сказанного является выполнение некоторого действия, одновременно декрементируя некоторое значение. например: 1 $ let x=5 2 $ while 3 > (( x-=1 )) 4 > do 5 > something 6 > done Этот пример показывает выполнение некоторого действия с начальным значением 5, которое декрементирует до тех пор, пока оно не будет равно нулю. При каждой итерации выполняется функция something. Наиболее распространённым является сокращение, избавляющееся от слова let в про- граммах оболочек. Если объявить переменные целыми значениями, то любое присвоение автоматически будет трактоваться как арифметическое действие. Если использовать typeset -i для объявления и присвоения переменной, то при последующем её приме- нении она станет целой. Также можно использовать ключевое слово integer (псевдоним для typeset -i) и объявлять таким образом переменные целыми. Выражения типа х=y+z будет восприниматься в это случае как арифметические. Команда read позволяет читать значения переменных со стандартного ввода: 1 echo “Please enter Month and Day of Birth ?” 2 read mon day trash В переменные mon и day будут считаны соответствующие значения, введённые с кла- виатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её. Изъять переменную из программы можно с помощью команды unset. Имена некоторых переменных имеют для командного процессора специальный смысл. Значением переменной PATH (т.е. $РАТН) является список каталогов, в которых команд- ный 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 последователь- ность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога. Если Вы сами явно не присвоите переменной PATH какое-либо значение, то стандарт- ной (по умолчанию) последовательностью поиска файла является следующая: текущий каталог, каталог /bin, каталог /usr/bin. Именно в такой последовательности командный процессор ищет файлы, содержащие программы, которые обеспечивают выполнение таких, например, команд, как echo, ls и cat. В списке каталогов, являющемся значением переменной PATH, имена каталогов от- деляются друг от друга с помощью символа двоеточия. В качестве примера приведём команду: 1 PATH=~/bin:/usr/local/bin/:/bin:/usr/bin Переменные PS1 и PS2 предназначены для отображения промптера командного про- цессора. PS1 — это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактивная программа, запущенная командным 86 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >. Другие стандартные переменные: – HOME — имя домашнего каталога пользователя. Если команда cd вводится без аргумен- тов, 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 прове- ряет 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. Значе- ние всех переменных можно просмотреть с помощью команды set. 10.2.4. Метасимволы и их экранирование При перечислении имён файлов текущего каталога можно использовать следующие символы: – \* — соответствует произвольной, в том числе и пустой строке; – ? — соответствует любому одинарному символу; – [c1-c1] — соответствует любому символу, лексикографически находящемуся между символами c1 и с2. Например, – echo \* — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls; – ls *.c — выведет все файлы с последними двумя символами, совпадающими с .c. – echo prog.? — выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.. – [a-z]* — соответствует произвольному имени файла в текущем каталоге, начинаю- щемуся с любой строчной буквы латинского алфавита. Такие символы, как ’ < > \* ? |  ” &, являются метасимволами и имеют для ко- мандного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с по- мощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавыч- ки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , “. Например, – echo \* выведет на экран символ *, – echo ab’*|*’cd выведет на экран строку ab*|*cd. 10.2.5. Командные файлы и функции Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: 1 bash командный\_файл [аргументы] Кулябов Д. С. и др. Операционные системы 87 Чтобы не вводить каждый раз последовательности символов bash, необходимо из- менить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды 1 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. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интер- претацию. Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключённых в фигур- ные скобки. Удалить функцию можно с помощью команды unset c флагом -f. 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 обо- лочек; – -fu — обозначает указанные функции как автоматически загружаемые. Автоматиче- ски загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноимёнными именами функ- ций, загружает его и вызывает эти функции. 10.2.6. Передача параметров в командные файлы и специальные переменные 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 ко- мандного 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. Рассмотрим это на примере. Пусть к командному файлу where имеется доступ по выполнению и этот командный файл содержит следующий конвейер: 1 who | grep $1. Если Вы введёте с терминала команду where andy, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем andy, в данный момент работает в ОС UNIX, то на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал ничего не будет выведено. Команда grep производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандарт- ный вывод. В примере команда grep используется как фильтр, обеспечивающий ввод со 88 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … стандартного ввода и вывод всех строк, содержащих последовательность символов andy, на стандартный вывод. В ходе интерпретации файла командным процессором вместо комбинации символов $1 осуществляется подстановка значения первого и единственного параметра andy. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем andy, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: 1 $ where andy 2 andy ttyG Jan 14 09:12 3 $ Определим функцию, которая изменяет каталог и печатает список файлов: 1 $ function clist { 2 > cd $1 3 > ls 4 > } Теперь при вызове команды clist будет изменён каталог и выведено его содержимое. Команда shift позволяет удалять первый параметр и сдвигает все остальные на места предыдущих. При использовании в командном файле комбинации символов $# вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение. Вот ещё несколько специальных переменных, используемых в командных файлах: – $* — отображается вся командная строка или параметры оболочки; – $? — код завершения последней выполненной команды; – $$ — уникальный идентификатор процесса, в рамках которого выполняется команд- ный процессор; – $! — номер процесса, в рамках которого выполняется последняя вызванная на выпол- нение 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 пере- менных; – ${name:-value} — если значение переменной name не определено, то оно будет заме- нено 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 пе- ременная определена, то подставляется value; – ${name#pattern} — представляет значение переменной name с удалённым самым коротким левым образцом (pattern); – ${#name[\*]} и ${#name[@]} — эти выражения возвращают количество элементов в массиве name. Кулябов Д. С. и др. Операционные системы 89 10.2.7. Использование команды getopts Весьма необходимой при программировании является команда getopts, которая осу- ществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: 1 getopts option-string variable [arg … ] Флаги — это опции командной строки, обычно помеченные знаком минус; Например, для команды ls флагом может являться -F. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют флаги, соответствующим образом изменяя своё поведение. Строка опций option-string — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответству- ющей переменной присваивается буква данной опции. Если команда getopts может распознать аргумент, то она возвращает истину. Принято включать getopts в цикл while и анализировать введённые данные с помощью оператора case. Предположим, необходимо распознать командную строку следующего формата: 1 testprog -ifile\_in.txt -ofile\_out.doc -L -t -r Вот как выглядит использование оператора getopts в этом случае: 1 while getopts o:i:Ltr optletter 2 do case OPTARG;; 4 i) iflag=1; ival=$OPTARG;; 5 L) Lflag=1;; 6 t) tflag=1;; 7 r) rflag=1;; 8 *) echo Illegal option $optletter 9 esac 10 done Функция getopts включает две специальные переменные среды — OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значение этого аргумента (будет равна file\_in.txt для опции i и file\_out.doc для опции o. OPTIND является числовым индексом на упомянутый аргумент. Функция getopts также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных. 10.2.8. Управление последовательностью действий в командных файлах Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некото- рого условия. Для решения подобных задач язык программирования bash предоставляет возможность использовать такие управляющие конструкции, как for, case, if и while. С точки зрения командного процессора эти управляющие конструкции являются обыч- ными командами и могут использоваться как при создании командных файлов, так и при 90 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … работе в интерактивном режиме. Команды, реализующие подобные конструкции, по су- ти, являются операторами языка программирования bash. Поэтому при описании языка программирования bash термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть ис- пользовано для принятия решения о дальнейших действиях. Команда test, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения. Так например, команда 1 test -f file возвращает нулевой код завершения (истина), если файл file существует, и ненулевой код завершения (ложь) в противном случае: – test s –— истина, если аргумент s имеет значение истина; – test -f file — истина, если файл file существует; – test -i file — истина, если файл file доступен по чтению; – test -w file — истина, если файл file доступен по записи; – test -e file — истина, если файл file — исполняемая программа; – test -d file — истина, если файл file является каталогом. 10.2.8.1. Оператор цикла for В обобщённой форме оператор цикла for выглядит следующим образом: 1 for имя [in список-значений] 2 do список-команд 3 done При каждом следующем выполнении оператора цикла for переменная имя прини- мает следующее значение из списка значений, задаваемых списком список -значений. Вообще говоря, список-значений является необязательным. При его отсутствии оператор цикла for выполняется для всех позиционных параметров или, иначе говоря, аргумен- тов. Таким образом, оператор for i эквивалентен оператору for i in $*. Выполнение оператора цикла for завершается, когда список-значений будет исчерпан. Последователь- ность команд (операторов), задаваемая списком список-команд, состоит из одной или более команд оболочки, отделённых друг от друга с помощью символов newline или ;. Рассмотрим примеры использования оператора цикла for. В результате выполнения оператора 1 for A in alpha beta gamma 2 do echo A 3 done на терминал будет выведено следующее: 1 alpha 2 beta 3 gamma Предположим, что Вы хотите найти во всех файлах текущего каталога, содержащих исходные тексты программ, написанных на языке программирования Си, все вхождения Кулябов Д. С. и др. Операционные системы 91 функции с некоторым именем. Это можно сделать с помощью такой последовательности команд: 1 for i 2 do 3 grep $i *.c 4 done Поместив эту последовательность команд в файл findref, после возможно, используя команду 1 findref ‘hash(’ ‘insert(’ ‘symbol(’, вывести на терминал все строки из всех файлов текущего каталога, имена которых оканчи- ваются символами .с, содержащие ссылки на функции hash( ), insert( ) и symbol( ). Использование символов ’ в вышеприведённом примере необходимо для снятия специ- ального смысла с символа (. 10.2.8.2. Оператор выбора case Оператор выбора case реализует возможность ветвления на произвольное число ветвей. Эта возможность обеспечивается в большинстве современных языков програм- мирования, предполагающих использование структурного подхода. В обобщённой форме оператор выбора case выглядит следующим образом: 1 case имя in 2 шаблон1) список-команд;; 3 шаблон2) список-команд;; 4 … 5 esac Выполнение оператора выбора case сводится к тому, что выполняется последователь- ность команд (операторов), задаваемая списком список-команд, в строке, для которой значение переменной имя совпадает с шаблоном. Поскольку метасимвол*  соответству- ет произвольной, в том числе и пустой, последовательности символов, то его можно использовать в качестве шаблона в последней строке перед служебным словом esac. В этом случае реализуются все действия, которые необходимо произвести, если значение переменной имя не совпадает ни с одним из шаблонов, заданных в предшествующих строках. Рассмотрим примеры использования оператора выбора case. В результате выполнения оператора 1 for A in alpha beta gamma 2 do case $A in 3 alpha) B=a;; 4 beta) В=с;; 5 gamma) В=e 6 esac 7 echo $B 8 done 92 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … на терминал будет выведено следующее: 1 а 2 c 3 e 10.2.8.3. Условный оператор if В обобщённой форме условный оператор if выглядит следующим образом: 1 if список-команд 2 then список-команд 3 {elif список-команд 4 then список-команд} 5 [else список-команд] 6 fi Выполнение условного оператора if сводится к тому, что сначала выполняется после- довательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово if. Затем, если последняя выполненная команда из этой последова- тельности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список-команд в строке, содер- жащей служебное слово then. Фраза elif проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слово else, является необязатель- ной. Если она присутствует, то последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово else, будет выполнена только при условии, что последняя выполненная команда из последовательности команд (операто- ров), которую задаёт список-команд в строке, содержащей служебное слово if или elif, возвращает ненулевой код завершения (ложь). Рассмотрим следующий пример: 1 for A in \* 2 do if test -d $A 3 then echo $A: is a directory 4 else echo -n $A: is a file and 5 if test -w $A 6 then echo writeable 7 elif test -r $A 8 then echo readable 9 else echo neither readable nor writeable 10 fi 11 fi 12 done Первая строка в приведённом выше примере обеспечивает выполнение всех последую- щих действий в цикле для всех имён файлов из текущего каталога. При этом переменная А на каждом шаге последовательно принимает значения, равные именам этих файлов. Первая содержащая служебное слово if строка проверяет, является ли файл, имя кото- рого представляет собой текущее значение переменной А, каталогом. Если этот файл является каталогом, то на стандартный вывод выводятся имя этого файла и сообщение Кулябов Д. С. и др. Операционные системы 93 о том, что файл с указанным именем является каталогом. Эти действия в приведённом выше примере обеспечиваются в результате выполнения третьей строки. Оставшиеся строки выполняются только в том случае, если проверка того, является ли файл, имя которого представляет собой текущее значение переменной А, каталогом, даёт отрицательный ответ. Это означает, что файл, имя которого представляет собой текущее значение переменной А, является обычным файлом. Если этот файл является обычным файлом, то на стандартный вывод выводятся имя этого файла и сообщение о том, что файл с указанным именем является обычным файлом. Эти действия в приведённом выше примере обеспечиваются в результате выполнения четвёртой строки. Особенностью использования команды echo в этой строке является использование флага -n, благодаря чему выводимая командой echo строка не будет дополнена символом newline (перевод строки), что позволяет впоследствии дополнить эту строку, как это, например, показано в приведённом выше примере. Вторая строка, содержащая служебное слово if, проверяет, доступен ли по записи файл, имя которого представляет собой текущее значение переменной А. Если этот файл доступен по записи, то строка дополняется соответствующим сообщением. Если же этот файл недоступен по записи, то проверяется, доступен ли этот файл по чтению. Эти дей- ствия в приведённом выше примере обеспечиваются в результате выполнения седьмой строки. Если этот файл доступен по чтению, то строка дополняется соответствующим сообщением. Если же этот файл недоступен ни по записи, ни по чтению, то строка также дополняется соответствующим сообщением. Эти действия в приведённом выше примере обеспечиваются в результате выполнения девятой строки. 10.2.8.4. Операторы цикла while и until В обобщённой форме оператор цикла while выглядит следующим образом: 1 while список-команд 2 do список-команд 3 done Выполнение оператора цикла while сводится к тому, что сначала выполняется после- довательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, а затем, если последняя выполненная команда из этой по- следовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, со- держащей служебное слово do, после чего осуществляется безусловный переход на начало оператора цикла while. Выход из цикла будет осуществлён тогда, когда послед- няя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, возвратит ненулевой код завершения (ложь). Приведённый ниже фрагмент командного файла иллюстрирует использование опера- тора цикла while. В нем реализуется ожидание события, состоящего в удалении файла с определённым именем, и только после наступления этого события производятся даль- нейшие действия: 1 while test -f lockfile 2 do sleep 30 3 echo waiting for semaphore 4 done 5 6 :create the semaphore file 94 Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. … 7 8 echo > lockfile 9 10 :further commands and after them delete the semaphore file 11 12 rm lockfile Командный файл, продемонстрированный в приведённом примере, по сути, является простейшей реализацией механизма синхронизации взаимодействующих процессов на основе семафоров. При замене в операторе цикла while служебного слова while на until условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла while и оператор цикла until идентичны. В обобщённой форме оператор цикла until выглядит следующим образом: 1 until список-команд 2 do список-команд 3 done Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования bash: это команда true, которая всегда воз- вращает код завершения, равный нулю (т.е. истина), и команда false, которая всегда возвращает код завершения, не равный нулю (т. е. ложь). Ниже приведены два примера, иллюстрирующие бесконечные циклы, которые будут выполняться до тех пор, пока ЭВМ не сломается или не будет выключена (ну, по крайней мере, до тех пор, пока Вы не нажмёте клавишу, соответствующую специальному символу INTERRUPT): 1 while true 2 do echo hello andy 3 done 1 until false 2 do echo hello mike 3 done 10.2.8.5. Прерывание циклов Два несложных способа позволяют вам прерывать циклы в оболочке bash. Команда break завершает выполнение цикла, а команда continue завершает данную итерацию блока операторов. Команда break полезна для завершения цикла while в ситуациях, когда условие пере- стаёт быть правильным. Пример бесконечного цикла while с прерыванием в момент, когда файл перестаёт существовать: 1 while true 2 do 3 if [! -f filelist[$i] 2 (( $i < ${\#filelist[\*]} ))
3 do
4 if
5 [ "$file” == “dev/null”] 6 then 7 continue 8 fi 9 action 10 done Эта программа пропускает нужное значение, но продолжает тестировать остальные

# 4 Выполнение лабораторной работы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 1).

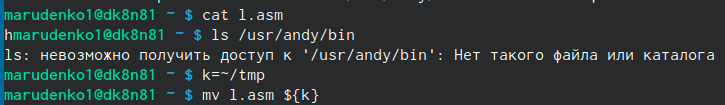


Рис. 1: присвоение строки переменной и использование

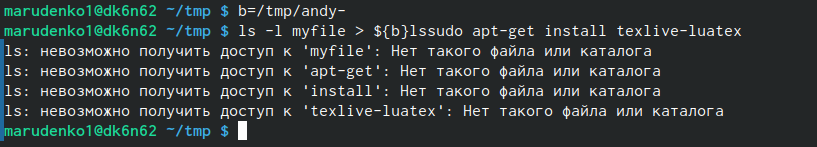


Рис. 2: то же самое

введение массива

Рис. 3: введение массива

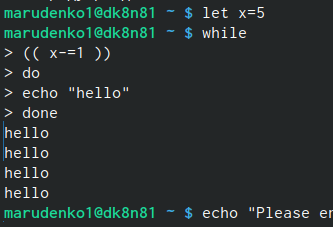


Рис. 4: цикл и присвоение переменной значения

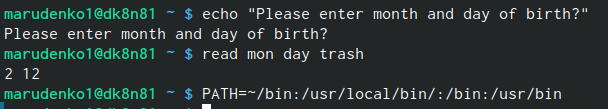


Рис. 5: присвоение с клавы значений переменным

путь

Рис. 6: путь

вывод на экран звездочки

Рис. 7: вывод на экран звездочки

вывод на экран строки

Рис. 8: вывод на экран строки

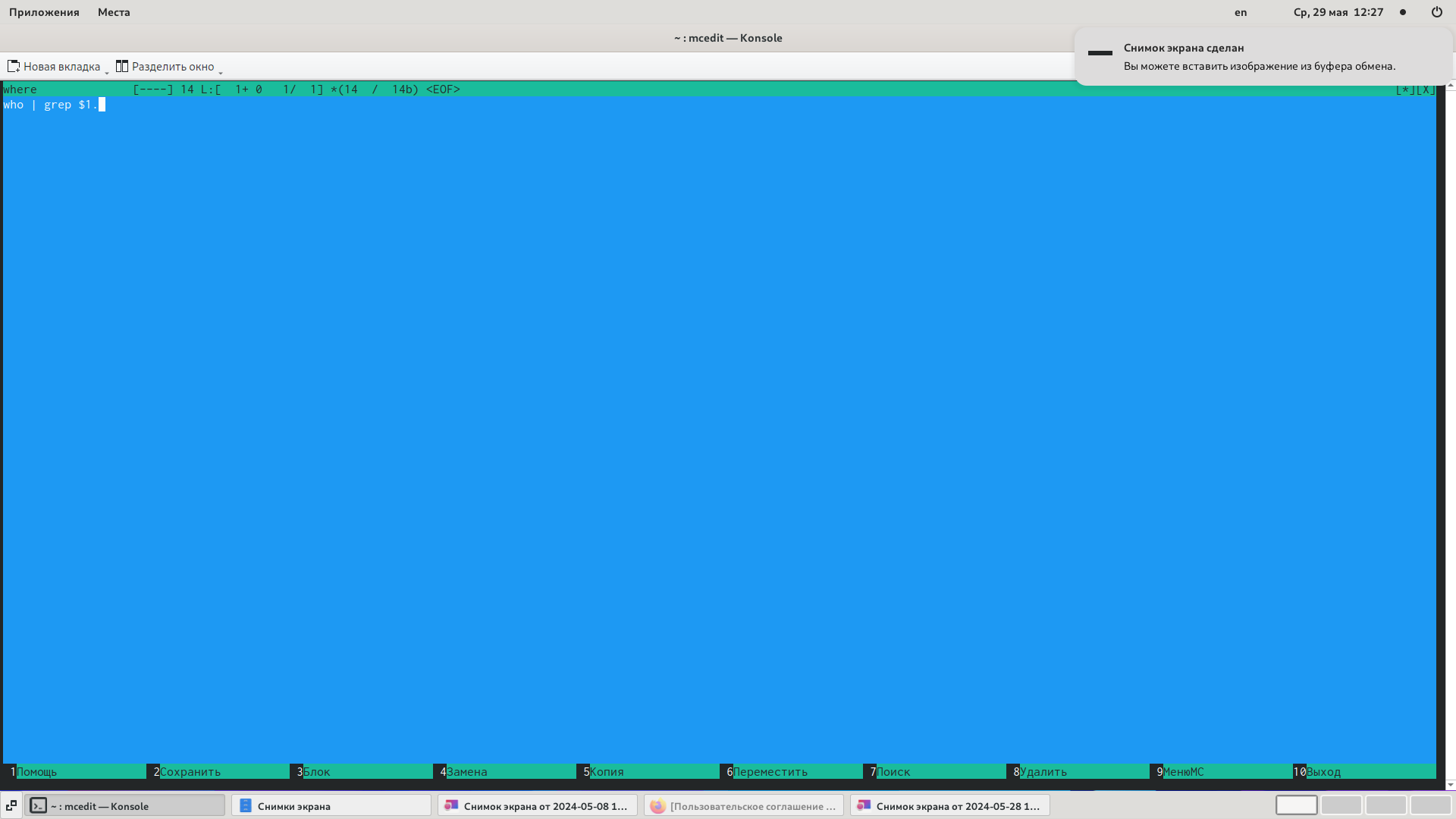


Рис. 9: запись в файл конвейера

создаем where и запускаем where

Рис. 10: создаем where и запускаем where

даем право на выполнение

Рис. 11: даем право на выполнение

ввод соответствующей команды

Рис. 12: ввод соответствующей команды

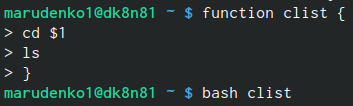


Рис. 13: функция clist

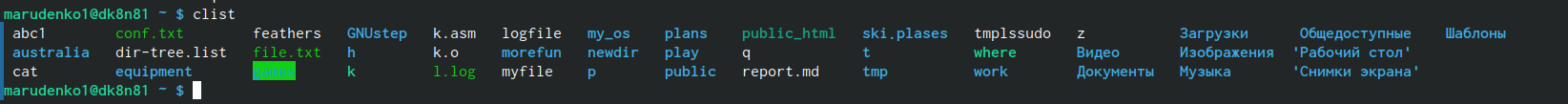


Рис. 14: вызов функции clist

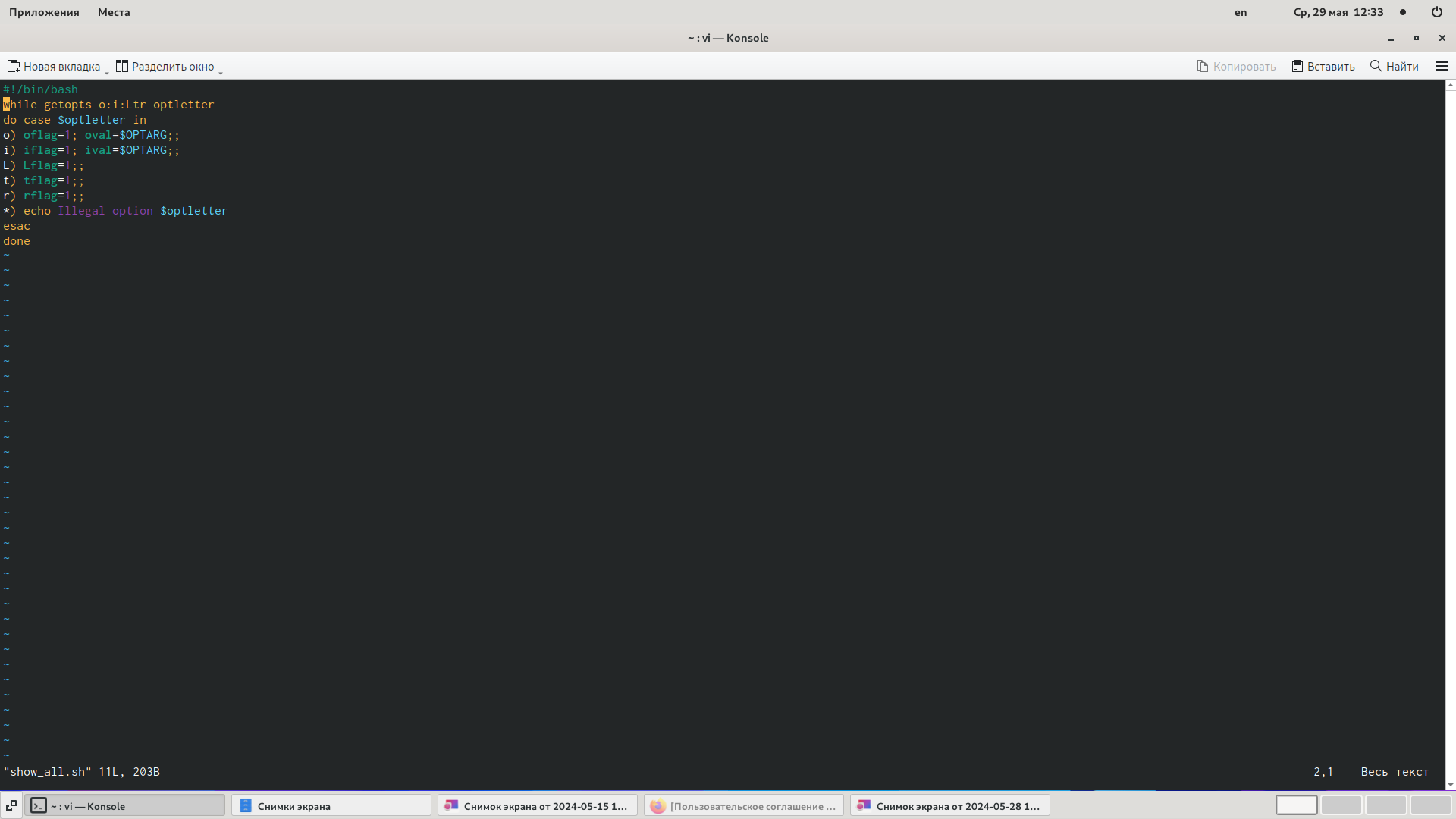


Рис. 15: функция getopts

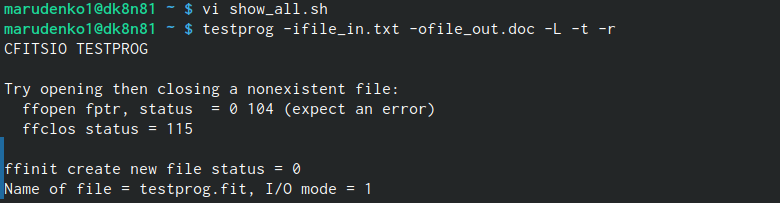


Рис. 16: запуск testprog, но как он распознает что ему с этмим файлом надо работать?

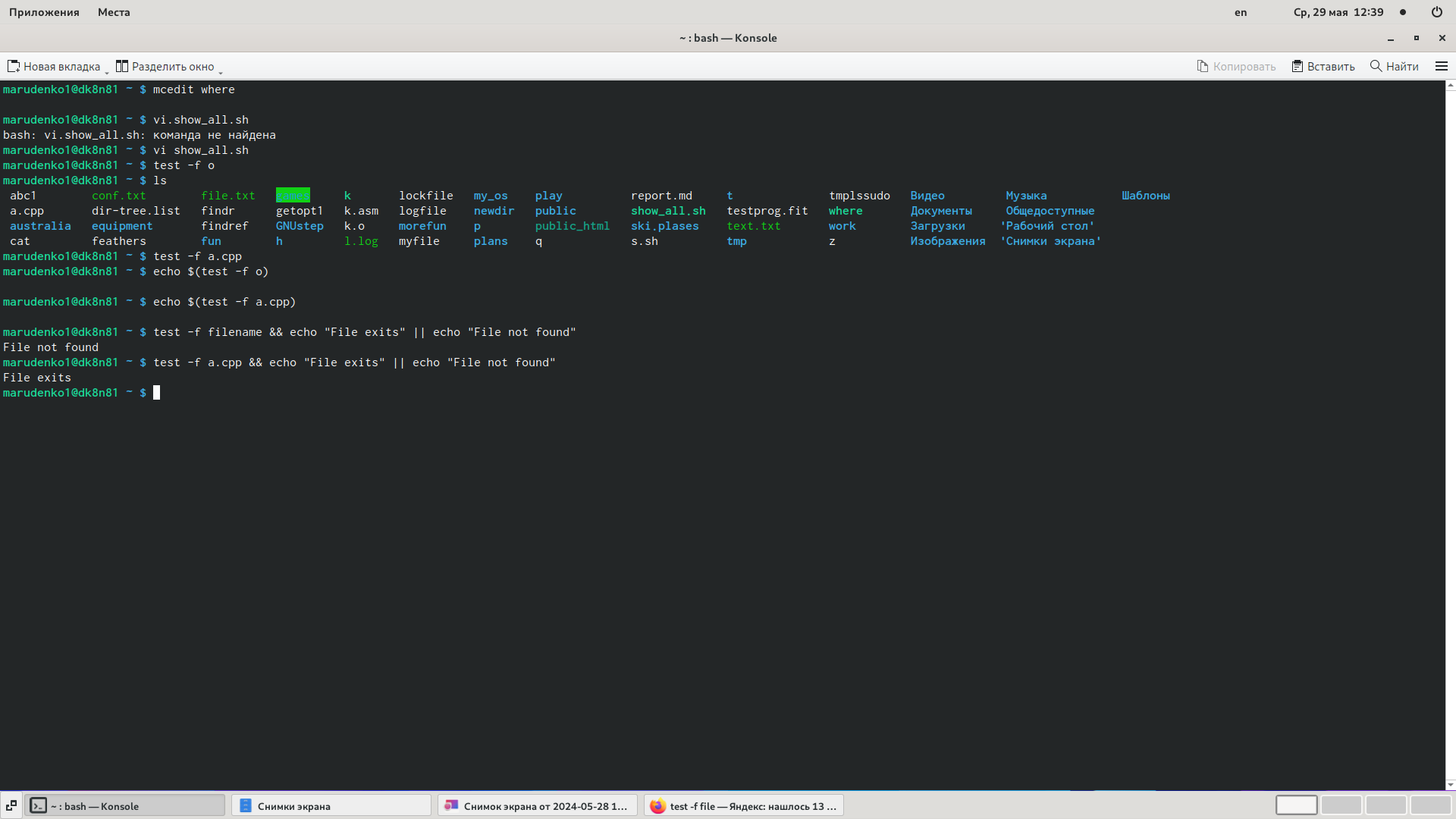


Рис. 17: test -f file

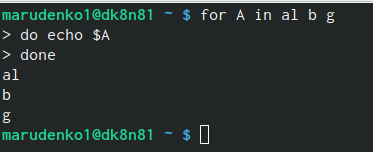


Рис. 18: вывод списка из 3х значений

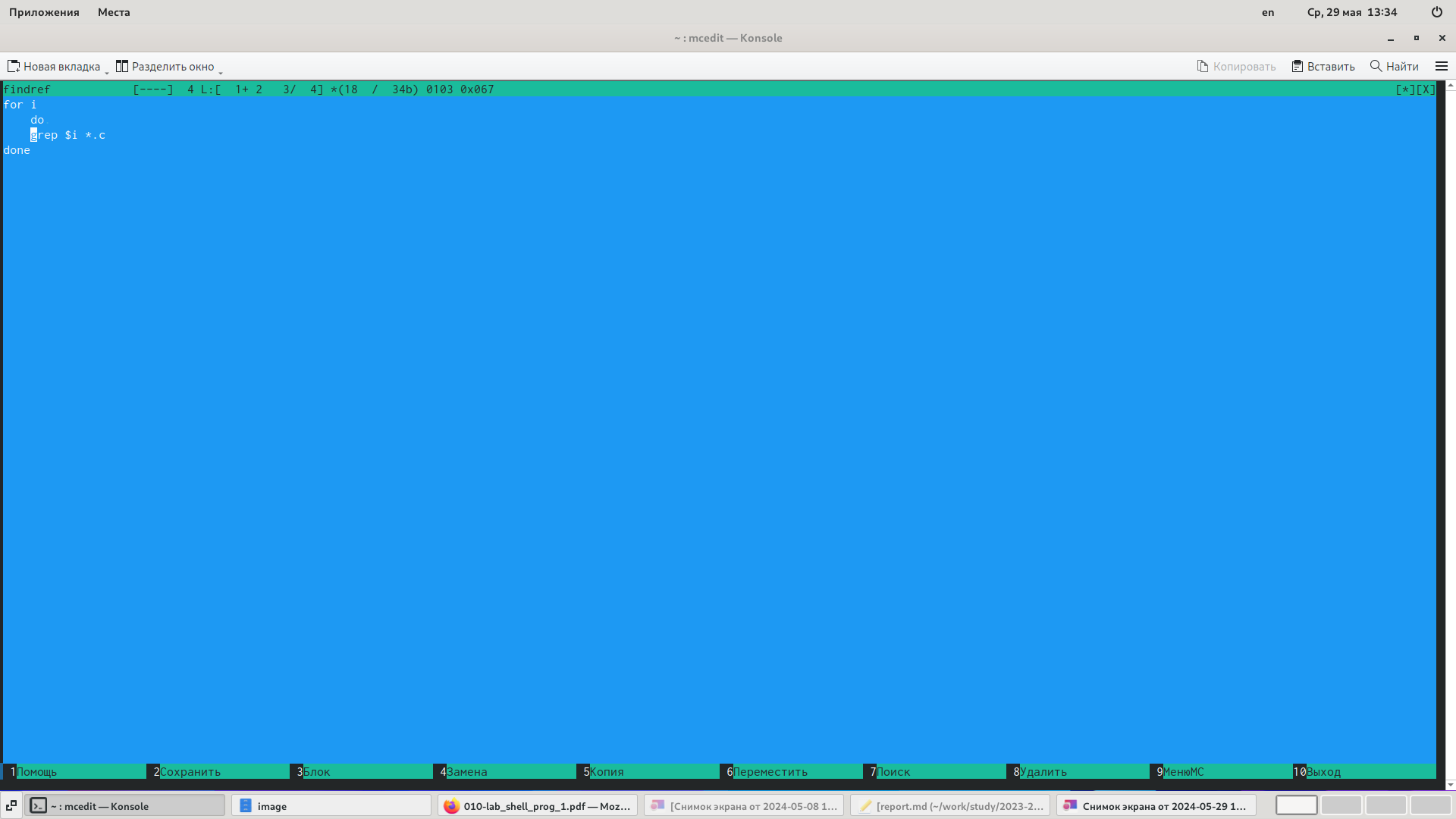


Рис. 19: findref

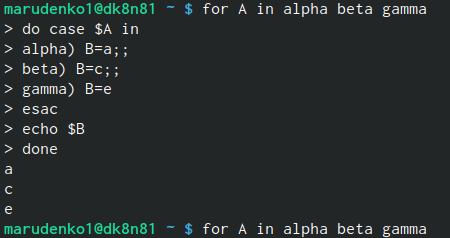


Рис. 20: использование оператора case

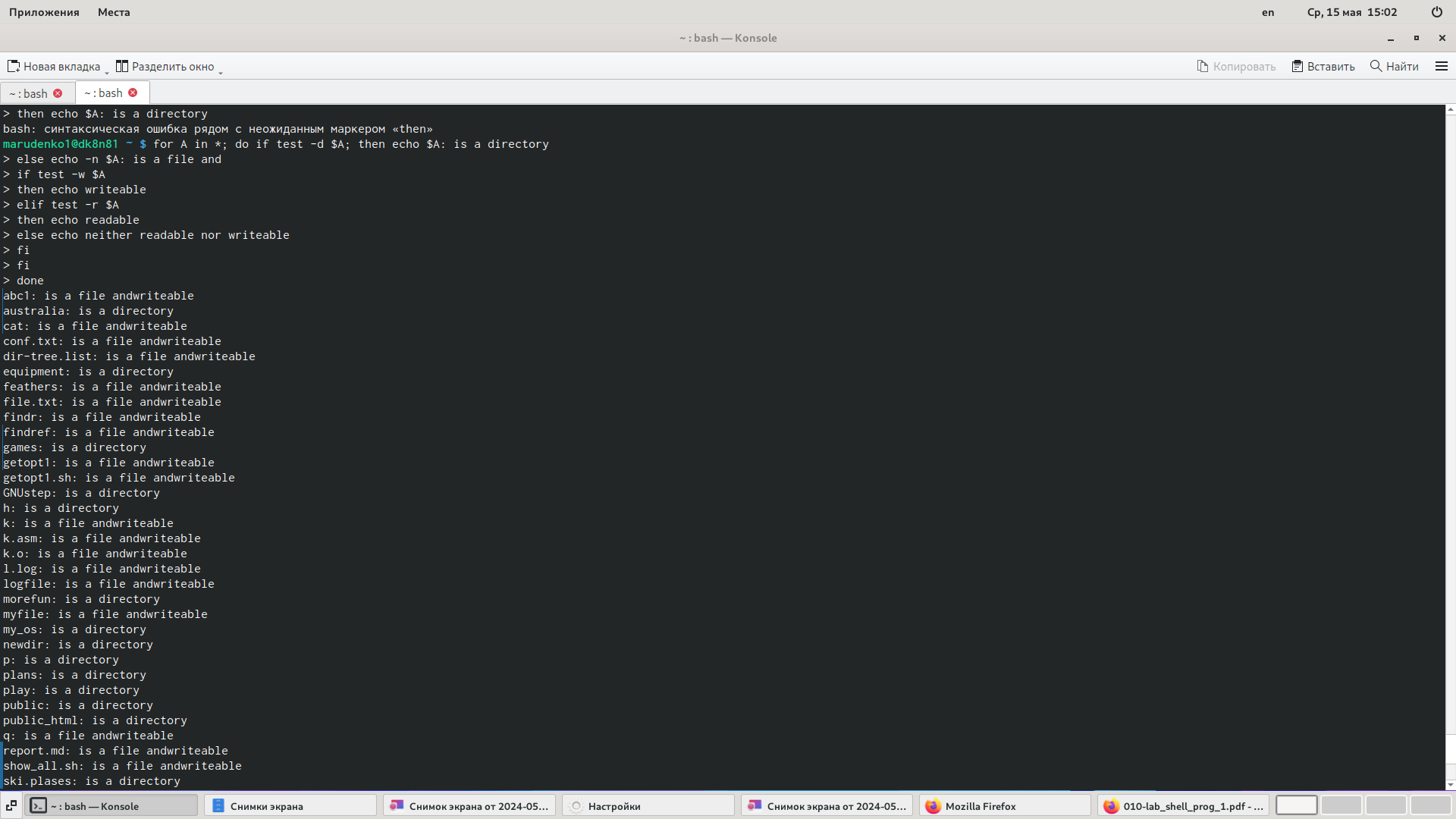


Рис. 21: читабельное и записывабельное ли

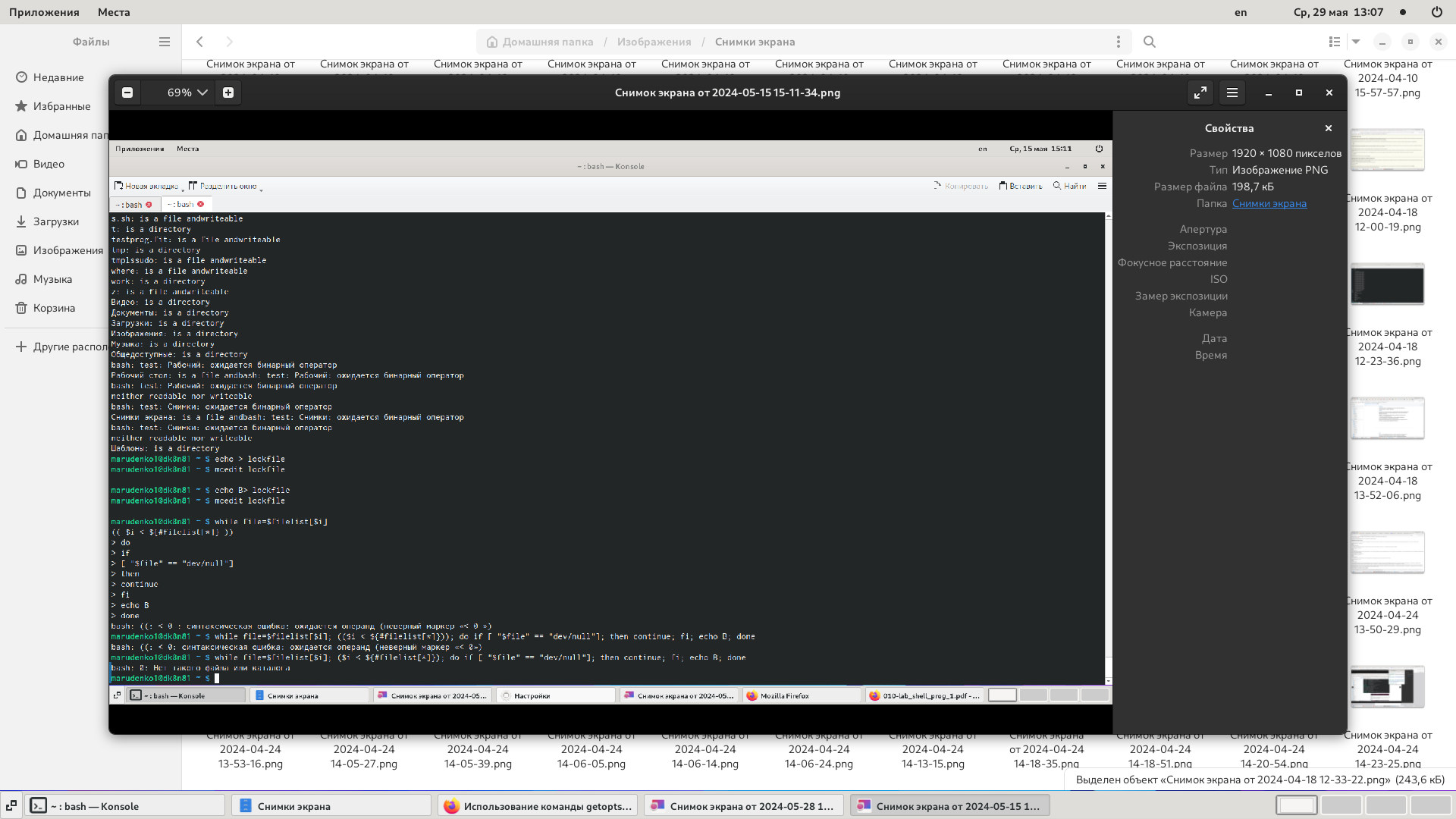


Рис. 22: игнорирование файла в произвольном списке

|  |
| --- |
| Название рисунка |

Рис. 23: Название рисунка

# 5 Выводы

Здесь кратко описываются итоги проделанной работы.