**Практическое задание №1. Разработка консольной утилиты на языке BASH**

**Ссылка на задание на github:** [**https://github.com/mir3paxx/os\_labs**](https://github.com/mir3paxx/os_labs)

Для начала с помощью текстового редактора nano создается и открывается для редактирования скриптовый файл *sysinfo.sh*, в котором будет находится bash-скрипт, реализующий консольную утилиту.



Рис. 1. Создание скрипта

В начале скрипта пишется шебанг, который нужен для указания использования оболочки bash для интерпретации скрипта.

Затем пишется функция *print\_help*, которая является справкой к создаваемой утилите, она выводит список доступных ключей и информацию о том, что они делают, затем она останавливает работу. Для вывода текста в консоль используется команда *echo*. В конце, после вывода всей информации, команда *exit 0* завершает выполнение скрипта с кодом выхода 0, который означает успешное завершение работы.

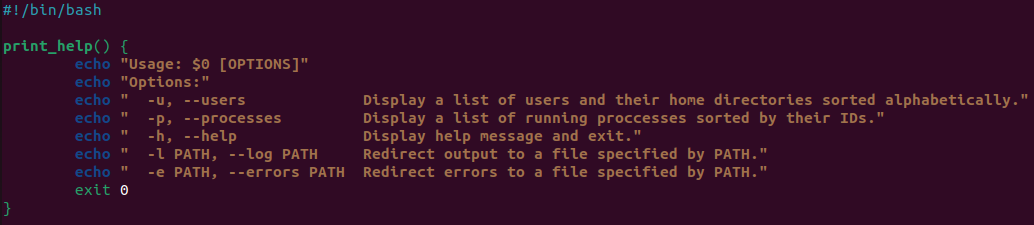


Рис. 2. Функция print\_help

Далее пишется функция *display\_users*, которая реализует вывод списка пользователей и их домашних директорий в алфавитном порядке. Сначала команда *getent passwd* извлекает информацию о пользователях из файла */etc/passwd*, затем с помощью символа *“|”* вывод этой команды передается на ввод следующей команде, Команда *cut -d: -f1,6* разделяет строки на поля по символу *“:”* и выводит только первое и шестое поле каждой строки. В файле */etc/passwd* первое поле содержит имя пользователя, а шестое поле содержит путь к домашнему каталогу пользователя. Команда *sort* используется для сортировки в алфавитном порядке.

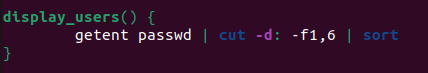


Рис. 3. Функция display\_users

Следующая функция *display\_processes* создана для вывода перечня запущенных процессов, отсортированных по их идентификатору. Команда *ps* выводит сведения о процессах. Здесь она используется с несколькими параметрами: *-eo pid,cmd* указывает *ps* вывести идентификаторы процессов (pid) и команды, запустившие процессы (cmd). *--sort=pid* сортирует вывод по идентификатору процесса.

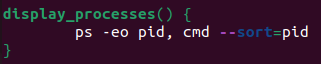


Рис. 4. Функция display\_processes

Затем реализуется функция *redirect\_output*, которая должна замещать вывод на экран выводом в файл по заданному пути. Сначала объявляется локальная переменная *path* и ей присваивается значение первого аргумента функции. Конструкция *if [ -n "$path" ]; then* проверяет, является ли переменная *path* непустой. Если она непустая, то команда *exec > "$path"* перенаправляет стандартный вывод в файл, указанный в переменной *path*, и таким образом, вывод всех последующих команд будет перенаправляться в файл. Если же переменная *path* пустая, ты выполняется следующий блок кода с *else*: выводится сообщение об ошибке в стандартный поток ошибок, затем командой *exit 1* завершается выполнение скрипта с кодом ошибки 1.

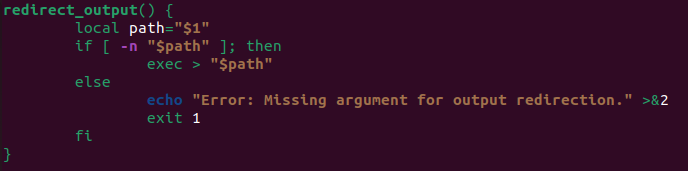


Рис. 5. Функция redirect\_output

Функция *redirect\_errors* замещает вывод ошибок из потока *stderr* в файл по заданному пути *PATH*, она реализована почти так же, как функция *redirect\_output*, единственное отличие, это то, что *exec 2> "$path"* перенаправляет стандартный поток ошибок (stderr) в файл, а *exec > "$path"* перенаправляет стандартный вывод (stdout) в файл.

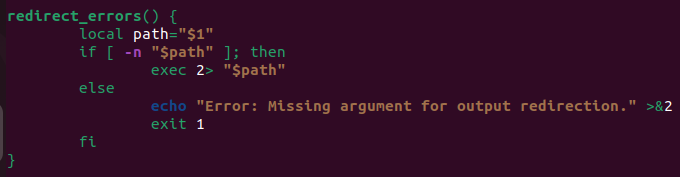


Рис. 6. Функция redirect\_errors

Далее выполняется проверка количества аргументов, переданного скрипту при его запуске, для этого используется переменная *$#*, которая содержит количество аргументов командной строки. В условии проверяется, равна ли эта переменная нулю (то есть не было передано аргументов), и если это так, то выводится сообщение об ошибке в стандартный поток ошибок. Затем команда *exit 1* завершает выполнение скрипта с ошибкой.

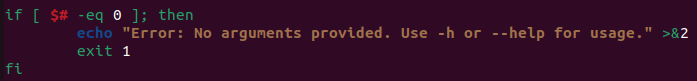


Рис. 7. Проверка на наличие аргументов

Далее для возможности работы с утилитой требуется реализовать обработку аргументов командной строки. Обработка реализуется с помощью команды *getopts*. Конструкция *getopts "uphl:e:-:" opt* используется для обработки аргументов командной строки. *"uphl:e:-:"* указывает на список опций, которые скрипт ожидает получить. Каждая буква опции в этой строке представляет собой однобуквенный флаг. Если опция требует аргумент, то после ее буквы указывается двоеточие (:), опция *-:* в списке аргументов для *getopts* указывает, что скрипт должен обрабатывать длинные опции, начинающиеся с двойного дефиса (--), *opt* - это переменная, которая будет содержать текущий обработанный аргумент.

Далее с помощью конструкции *case-esac* происходит обработка всех возможных значений опций, хранящихся в переменной *opt*. В опциях *l* и *e* *$OPTARG* - это переменная, которая содержит аргумент, переданный текущей опции, *shift* нужен для сдвига аргумента (перехода к следующему).



Рис. 8. Обработка коротких опций

*case "${OPTARG}" in*: здесь происходит проверка значения длинной опции (начинается с двойного дефиса), хранящегося в переменной *${OPTARG}*. И внутри этого case обрабатываются все возможные значения длинной опции.

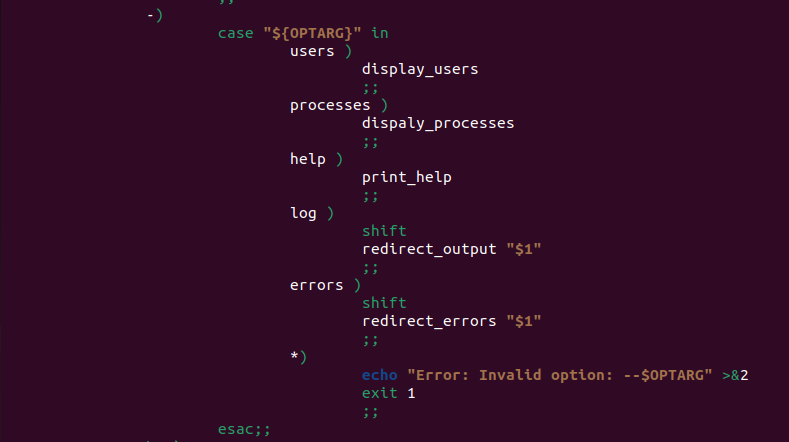


Рис. 9. Обработка длинных опций

Далее в основном кейсе обрабатывается ситуация, когда введена недопустимая опция, в этом случае выводится сообщение об ошибке и ситуация, когда опция требует аргумента, но аргумент не был передан. В этом случае также выводится сообщение об ошибке. После этого происходит выход из цикла. Последняя строка удаляет из списка аргументов командной строки все опции, которые уже были обработаны командой *getopts*, чтобы оставшиеся аргументы были доступны для дальнейшей обработки в скрипте ((*OPTIND - 1*) - это арифметическое выражение, которое вычисляет количество обработанных опций. Поскольку *OPTIND* указывает на следующий аргумент, который будет обработан, вычитание единицы дает количество уже обработанных опций, а *shift* сдвигает аргументы влево на указанное число).

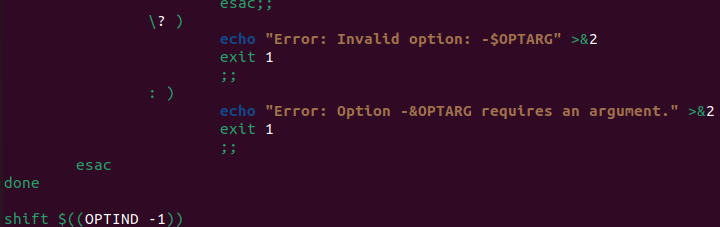


Рис. 10. Обработка ошибок

Затем файл, содержащий написанный скрипт, делается исполняемым с помощью команды *chmod +x sysinfo.sh*.

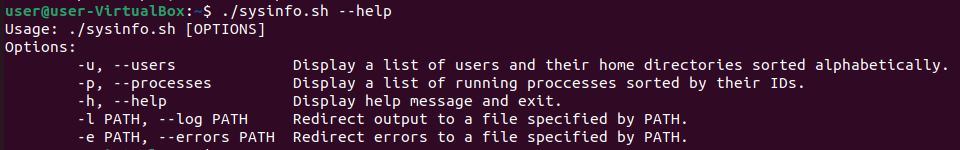


Рис. 11. Результат работы утилиты с аргументом --help

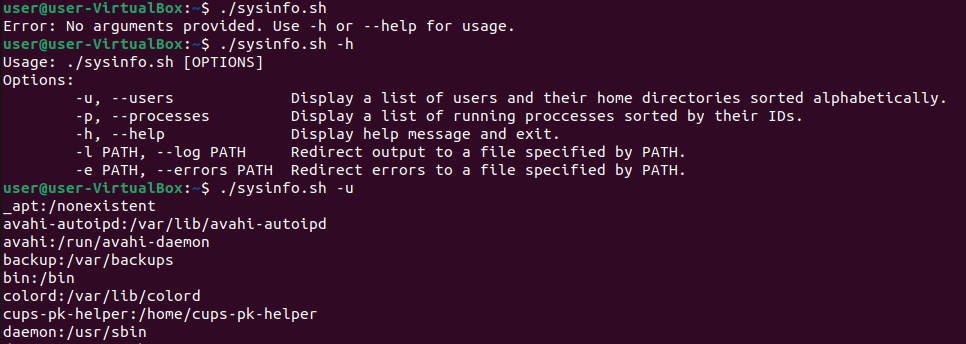


Рис. 12. Результат работы утилиты с аргументами -h и -u

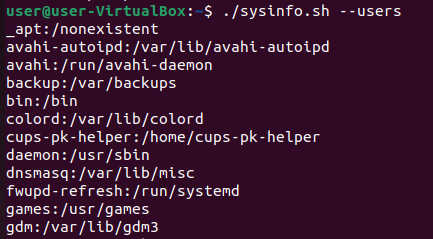


Рис. 13. Результат работы утилиты с аргументом --users

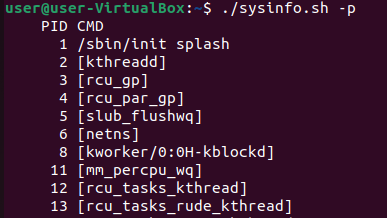


Рис. 14. Результат работы утилиты с аргументом -p

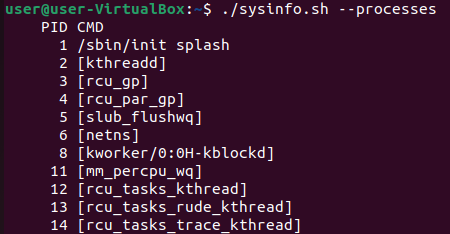


Рис. 15. Результат работы утилиты с аргументом --processes

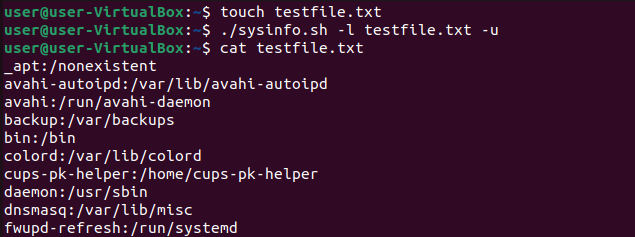


Рис. 16. Результат работы утилиты с аргументом –l

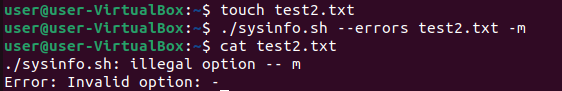


Рис. 17. Результат работы утилиты с аргументом --errors