**1.Работа с командной оболочкой**

* Работа выполняется: на хостовой или виртуальной машине.
* Навыки: выполнение команд, редактирование команд, просмотр истории, работа с аргументами команд.
* Изучаемые команды: bash, man, whoami, cal, history, clear, date, echo.

Команды, которые необходимо выполнить в терминале, будут обозначаться следующим образом:

$ whoami

Доллар в начала команды лишь говорит о том, что команду нужно выполнить от имени обычного пользователя, вводить символ доллара как часть команды не нужно. Если первый символ - решётка - это означает, что команду нужно ввести от имени системного администратора (root), что потребуется в некоторых последующих заданиях.

1. Откройте эмулятор терминала: Приложения - Системные - Терминал (Applications - System - Terminal).
2. Введите команду whoami и нажмите Enter.

$ whoami

1. Нажмите стрелку вверх - появится история набранных команд.
2. Нажмите стрелку вниз, чтобы стереть команду с экрана.
3. Выполните команду с опечаткой: whoaim. Что произошло?
4. Используя историю команд измените команду whoaim на whoami и снова выполните её.
5. Выведите на экран текущие дату и время.
6. Выведите календарь.
7. Прочитайте справку по команде cal и найдите информацию о том, как вывести календарь за три месяца:

$ man cal

Для того, чтобы закочнить чтение справки по команде, нажмите клавишу q. Так можно прочитать справку по любой команде, имеющейся в системе.

1. Выведите календарь для текущего месяца, предыдущего и следующего.
2. Используя команду echo выведите в терминал фразу "Hello World!":

$ echo "Hello World!"

В данном примере аргумент взят внутрь кавчек, таким образом, фраза "Hello World!" станет единым аргументов. Если ввести просто echo Hello World! - bash воспримет это как два самостоятельных агумента: Hello и World!, определяя их по пробелу.

1. Посмотрите историю своих команд.
2. Очистите окно терминала.

**Перемещение по файловой системе**

* Работа выполняется: на хостовой или виртуальной машине.
* Навыки: просмотр содержимого каталогов, перемещение по дереву каталогов.
* Изучаемые команды: pwd, cd, ls.

1. Установите, в каком каталоге вы сейчас находитесь.

pwd

1. Перейдите в корневой каталог (/).

cd /

1. Попробуйте перейти на каталог уровнем выше. Проверьте текущий каталог. Изменился ли текущий каталог? Почему?

cd ..

pwd

1. Выведите список файлов текущего каталога.

ls

1. Выведите список файлов каталога /var.

ls /var

1. Вернитесь в домашний каталог.

cd ~

или просто

cd

1. Поднимитесь на один каталог вверх. Какой стал текущий каталог?
2. Выведите список файлов из текущего каталога.
3. Выведите список файлов из домашнего каталога так, чтобы можно было определить владельца файлов. Используйте справку по команде ls (команда *man ls*).

**Работа с файловой системой**

* Работа выполняется: только на виртуальной машине.
* Навыки: создание файлов и каталогов, перемещение и переименование объектов файловой системы, копирование файлов и каталогов, просмотр содержимого файлов, удаление файлов.
* Изучаемые команды: touch, mkdir, mv, cp, cat, rm, rmdir.

1. Перейдите в домашний каталог.
2. Создайте в домашнем каталоге подкаталог fruits.

mkdir fruits

1. Перейдите в каталог /. Находясь в каталоге /, создайте в домашнем каталоге подкаталог animals.
2. Создайте во временном каталоге (/tmp) пустой файл temp.

touch /tmp/temp

1. Перейдите в ранее созданный каталог fruits.
2. В текущем каталоге создайте пустые файлы apple, banana, pineaple, lion.
3. Находясь в каталоге fruits, в ранее созданном каталоге animals создайте пустые файлы cat.txt, dog.txt, elephant.txt.
4. Определите дату создания файла apple.
5. Выведите из текущего каталога (fruits) файлы, имя которых начинается на b.

ls b\*

1. Выведите из текущего каталога файлы, имя которых оканчивается на a.
2. Выведите из текущего каталога файлы, имя которых содержит буквы b или i.
3. Скопируйте файл /etc/passwd в домашний каталог:

cp /etc/passwd ~

1. Выведите содержимое файла /etc/issue на экран.
2. Скопируйте файл /etc/issue на файл apple так, чтобы заменить файл apple.
3. Выведите содержимое файла apple на экран. Оно должно совпадать с содержимым файла /etc/issue.
4. Переместите файл lion в каталог animals.
5. В имени файла pineaple есть опечатка. Переименуйте файл pineaple в pineapple.
6. Подсчитайте количество строк в файле /etc/passwd.
7. Создайте в домашнем каталоге символьную ссылку с именем passwd\_link, указывающую на /etc/passwd.
8. Создайте в домашнем каталоге жёсткую ссылку с именем passwd\_hard, указывающую на /etc/passwd.
9. Отследите изменение количества жёстких ссылок у файла /etc/passwd и сравните его с другими файлами.
10. Удалите файл fruits/apple
11. Попробуйте удалить каталог fruits:

rmdir fruits

Возникнет ошибка. Почему?

1. Создайте пустой каталог rmme и удалите его командой rmdir. Почему команда выполнилась без ошибок?
2. Удалите каталог fruits командой rm -fr.

rm -fr fruits

rm -fr удаляет все каталоги с подкаталогами, не спрашивая и не помещая их в корзину. Этой командой надо пользоваться очень внимательно и только в крайних случаях, так как можно случайно удалить важные данные навсегда.

1. Выведите на экран содержимое файлов /etc/passwd и /varlog/messages одновременно, используя команду cat.
2. Скопируйте каталог /etc/init.d/ в домашний каталог:

cp /etc/init.d/ ~

Данная команда не выполнится, так как cp по умолчанию копирует файлы по одному. Чтобы скопировать каталог целиком, включая все подкаталоги, используйте команду:

cp -r /etc/init.d/ ~

1. Удалите каталог init.d из домашнего каталога.

**Конвейеры. Обработка текстовых файлов**

Выполнить следующие задания. Объяснить каждый шаг конвейера. Вместо знаков вопроса подставьте нужную команду/аргумент.

* Работа выполняется: на виртуальной машине.
* Навыки: построение конвейеров из команд, сортировка, фильтрация, поиск.
* Изучаемые команды: cut, grep, sort, wc, tr, uniq, head, tail, fold.

1. Вывести на экран всех пользователей системы с сортировкой по алфавиту:

$ cut -d: -f1 /etc/passwd | ???

1. Изменить предыдущую команду таким образом, чтобы помимо имени пользователя на экране также выводился его идентификатор. *Подсказка:* посмотрите содержимое файла /etc/passwd, чтобы найти, в каком из полей, разделенных двоеточием, находится UID.
2. Подсчитать количество пользователей, у которых командой оболочкой является bash:

$ grep :/bin/bash /etc/passwd | wc -l

1. Вывести на экран всех пользователей, использующих командную оболочку /bin/bash, отсортировав их по алфавиту в обратном порядке:

$ grep :/bin/bash /etc/passwd | ??? | ???

1. Выполнить сортировку пакетов по размеру (большего к меньшему) и вывести первые 25 пакетов:

$ rpm -qa --qf '[%{NAME}:]' | sort -rn -k2,2 | ???

1. Вывести все файлы из пакета man, подсчитать для каждого из них размер, игнорируя ошибки, и выполнить сортировку по размеру:

$ rpm -ql man | xargs -l du 2> /dev/null | sort -n

1. При помощи программы top вывести список работающих процессов и отсортировать их по имени процесса (аргумент команды sort заполнить самостоятельно):

$ top -b -n1 | sort ???

1. Используя команду awk, изменить вывод команды top таким образом, чтобы в первой колонке отображалось имя процесса, во второй - объём занимаемой памяти:

$ top -b -n1 | awk '{printf "%35s %s\n", $12, $6}'

1. Вывести файл /etc/passwd в отсортированном по пользователям виде, заменив знаки двоеточия пробелами:

$ ??? | tr ":" " "

1. Вывести файл /etc/passwd в отсортированном по пользователям виде в две колонки - в первой - имя пользователя, во второй - командная оболочка пользователя:

$ ??? | awk -F: '{printf "%15s %s\n", $1, $7}'

1. На примере файла user-manual.txt из документации git исследовать конвейеры.

$ cd /usr/share/doc/git-\*

1. Выполнить анализ частоты появления каждого слова в файле user-manual.txt. Конвейер нужно строить по шагам, наблюдая за изменением результата. Определить этап получения первичной информации, этап обработки и этап представления результатов.

$ cat user-manual.txt | tr ' ' '\012' | tr '[:upper:]' '[:lower:]' | tr -d '[:punct:]' | grep -v '[^a-z]' | sort | uniq -c | sort -rn | head -5

Или:

$ cat user-manual.txt | tr -cs '[:alpha:]' '\n' | tr '[:upper:]' '[:lower:]' | sort | uniq -c | sort -rn | head -5

Почему различаются результаты работы конвейеров?

1. Выполнить анализ частоты появления каждого символа.

$ tr '[:lower:]' '[:upper:]' < user-manual.txt | tr -d '[:punct:] \n\t' | fold -w1 | sort | uniq -c | sort -rn | head -5

1. Вывести самые часто встречаемые слова, в которых количество символов больше семи:

$ tr -cs '[:alpha:]' '\n' < user-manual.txt | less | grep -E ........ | sort | uniq -c | sort -rn | head

**Управление доступом к файлам**

Выполните следующие задания от имени **обычного** пользователя (не root!).

* Работа выполняется: только на виртуальной машине.
* Навыки: управление доступом к файлам.
* Изучаемые команды: chmod, chown, chgrp, umask, su.

1. Перейдите в домашний каталог.
2. Создайте каталог test и перейдите в него.
3. Создайте файл file, определите его права доступа.
4. Запретите любые действия с файлом file.
5. Попробуйте записать строку "тест" в файл file. В доступе должно быть отказано.
6. Попробуйте прочитать файл file. В доступе должно быть отказано.
7. Измените права доступа для file так, чтобы только текущий пользователь мог осуществлять в него запись. Команда ll должна показывать права --w-------.
8. Запишите строку "test" в файл file. Попробуйте прочитать содержимое файла - в доступе должно быть отказано.
9. Разрешите **группе** текущего пользователя читать содержимое файла. Группа текущего пользователя обычно совпадает с именем пользователя. Команда ll должна показывать права --w-r-----.
10. Попробуйте прочитать содержимое файла file. В доступе теперь должно быть *отказано*.
11. Разрешите текущему пользователю читать содержимое файла file. Команда ll должна показывать права -rw-r-----.
12. Попробуйте прочитать содержимоей файла file - должна появиться ранее добавленная туда строка test.
13. Создайте каталог dir. Создайте в этом каталоге файл new\_file, запишите в него текст "file 2".
14. Просмотрите содержимое каталога dir. Должен появиться файл new\_file.
15. Измените права доступа на каталог dir так, чтобы пользователи не имели права на исполнение (x). Попробуйте просмотреть содержимое файла new\_file в каталоге dir. В доступе должно быть отказано. Попробуйте удалить этот файл - в доступе также должно быть отказано.
16. Попробуйте изменить владельца файла file на root. Изменить группу владельца на root. В обоих операциях должно быть отказано - обычный пользователь не может изменить владельца файла.
17. При помощи команды umask задайте такую маску, чтобы при создании файла никто не имел к нему никакого доступа, кроме владельца. При создании нового файла (команда touch file1) права доступа данного файла будут следующими: -rw-------.
18. При помощи команды umask задайте такую маску, чтобы при создании файла все имели к данному файлу полный доступ. При создании нового файла (команда touch file2) права доступа данного файла будут следующими: -rw-rw-rw-.
19. Откройте второе окно с терминалом и выполните вход от имени системного администоратора (root):

su

1. От имени системного администратора измените владельца файла file на root. Измените права доступа так, чтобы только владелец мог прочитать файл.
2. Попробуйте прочитать файл file от имени **обычного** пользователя. В доступе должно быть отказано.
3. От имени системного администратора измените права доступа на файл так, чтобы группа могла прочитать данный файл.
4. Попробуйте прочитать файл file от имени **обычного** пользователя. В доступе должно быть разрешено.

**Управление процессами и заданиями**

* Работа выполняется: на хостовой или виртуальной машине.
* Навыки: работа с процессами, задачами, фоновый режим.
* Изучаемые команды: ps, top, bg, fg, pstree, kill.

1. Выполните команды:

$ ps

$ ps -u

$ ps -ux

$ ps -aux

$ ps -e

Найдите отличия в работе команды ps в зависимости от переданных аргументов.

1. Выведите список процессов в виде дерева:

$ pstree

1. Для того, чтобы найти какой-либо процесс, используйте конвейер:

$ pstree | less

В интерактивном режиме программы less напечатайте:

/syslog

и нажмите Enter. Выйдите из программы нажатием клавиши q

1. Bash обладает полезным свойством выполнять задачи в фоновом режиме, для этого после выполняемой команды нужно поставить символ амперсанд - "&". Запустите команду. #: которая будет работать "вечно":

$ yes

1. Завершите команду, нажав сочетание Ctrl+C
2. Запустите эту же команду в фоновом режиме, перенаправив стандартный вывод:

$ yes > /dev/null &

1. Появится надпись вроде следующей: [1] 18335. В квадратных скобках указан номер задания, следом - PID процесса, который может не совпадать с примером.
2. Убедитесь при помощи команды jobs, что в фоне работает одна задача
3. Запустите ещё одну задачу:

$ yes n > /dev/null &

И ещё одну:

$ yes yes > /dev/null &

1. Откройте команду top и отметьте две запущенные ранее задачи, потребляющие больше всего ресурсов:

$ top

1. Уничтожьте одну из них, набрав в интерактивном режиме:

k

Здесь - идентификатор процесса, который можно посмотреть в команде top

1. Выполните выход из команды top клавишей q
2. Выясните PID остальных задач:

$ jobs -l

1. Уничтожьте одну из задач командой kill, используя полученный PID

$ kill

1. Уничтожьте оставшуюся задачу по номеру задания, используя команду:

$ kill %

Здесь - номер задачи

1. Убедитесь, что в фоне теперь не выполняется ни одной задачи.
2. Существует ещё один метод перевода задачи в фоновый режим. Запустите команду:

$ yes > /dev/null

1. Остановите задачу, отправив ей сигнал HUP (для этого достаточно нажать сочетание клавиш Ctrl+Z).
2. Переведите остановленную задачу в фоновый режим:

$ bg

1. Откройте команду top и убедитесь, что процесс yes работает (занимает 100% процессорного времени)
2. Завершите команду yes, используя команду killall. Данная команда может остановить процесс по его имени:

$ killall yes

1. Проверьте состояние фоновых заданий, убедитесь, что в фоне нет ни одной задачи.

### 2.Ваш первый сценарий

В предыдущей работе для того чтобы вывести текст на экран, использовалась команда echo. В данном примере мы напишем для вывода фразы "Hello world!" специальный сценарий.

1. Создайте файл hello.sh.
2. Откройте данный файл для редактирования в текстовом редакторе.
3. Запишите первую строку файла: #!/bin/bash. Это так называемый "шебанг" - специальная инструкция, сообщающая операционной системе, что данный файл нужно воспринимать именно как Bash-сценарий и использовать для его выполнения командую оболочку Bash.
4. Запишите вторую строку в файл: echo "Hello world!". Данная команда выводит на экран фразу "Hello world!".
5. Выполните полученный сценарий: bash hello.sh
6. Для того чтобы сценарий можно было выполнить как обычную программу, нужно сделать файл сценария исполняемым: chmod +x hello.sh
7. Попробуйте запустить сценарий как обычную программу: ./hello.sh.
8. Попробуйте запустить сценарий без указания пути: hello.sh. Данная команда не работает, так как она отсутствует в перечне путей в переменной $PATH. $PATH содержит список каталогов, разделённых :, в которых командная оболочка последовательно ищет исполняемый файл.
9. Создайте в домашнем каталоге каталог bin и переместите туда файл hello.sh.
10. Попробуйте запустить сценарий без указания пути: hello.sh. Так как сценарий теперь расположен по пути, который есть в $PATH, команда должна сработать и вывести на экран "Hello world!".

Примечание. Данный сценарий будет доступен только текущему пользователю. Для того чтобы сценарий работал у всех пользователей системы, его нужно скопировать в каталог /usr/local/bin (для этого потребуются права root).

### Порядок выполнения работы

Начиная со следующего раздела вам будет необходимо самостоятельно разработать несколько сценариев на языке Bash. Создайте рабочий каталог, в котором будут содержаться ваши сценарии:

$ mkdir ~/scripts

$ cd ~/scripts

Установите программу git:

$ su

<ввести пароль root>

# yum install -y git

# exit

Инициализируйте в данном каталоге git-репозиторий:

$ git init

Создайте файл README, в котором напишите:

1. ФИО студента
2. Номер группы

И сделайте первый коммит с данным файлом. Не забудьте предварительно настроить git.

$ git config --global user.name "ФИО"

$ git config --global user.email "Адрес электронной почты"

$ git add README

$ git commit -m 'Added README' README

Дальнейшие правила выполнения задания:

1. **Все** сценарии должны содержаться в каталоге scripts.
2. Каждое задание - отдельный файл со сценарием. Название сценария придумать самостоятельно.
3. После написания и тестирования **каждого** сценария его нужно добавить в git аналогично файлу README и выполнить git commit с комментарием: "<порядковый номер задания> - <краткое описание сценария>".
4. Исправление ошибок должно также сопровождаться коммитом с описанием изменений.
5. Задания (примеры) без номера заводить в git не нужно.

Пример. Завершён сценарий № 1.1 раздела "1. Объединение команд", его название - make\_shared.sh.

$ git add make\_shared.sh

$ git commit -m '1.1 - создание общего каталога' make\_shared.sh

### 1. Автоматизация выполнения команд

В данном разделе рассмотрим простейшие примеры использования сценариев Bash - объединение нескольких однотипных команд. Это простейшие сценарии, которые не принимают аргументов, не содержат сложных условий, циклов и т.п.

Например, для создания каталога, доступного всем пользователям на чтение и запись, необходимо выполнить два действия:

* создать каталог;
* изменить права доступа для каталога.

1.1. Разработать сценарий, который создаёт каталог /tmp/shared и устанавливает на него права доступа rwxrwxrwx.

Пример решения данной задачи на языке сценариев Bash:

#!/bin/bash

DIR=/tmp/shared

mkdir -p "$DIR"

chmod 777 "$DIR"

1.2. Вывести количество файлов в домашнем каталоге, которые заканчиваются на .txt. Создайте несколько таких файлов для тестирования.

1.3. Вывести текущие переменные окружения в отсортированном по алфавиту порядке.

1.4. Разработать программу "Good morning", которая:

1. Пожелает пользователю доброго утра.
2. Выведет текущее время и календарь на текущий месяц.
3. Выведет список дел из файла TODO домашнего каталога пользователя.

1.5. Найти и вывести пути до файлов из каталога /usr (включая подкаталоги), размер которых больше 20 Мб. Подсказка: man find.

1.6. Подсчитать количество файлов, количество скрытых файлов в домашнем каталоге текущего пользователя и вывести результат в формате:

Домашний каталог пользователя

<User>

содержит обычных файлов:

XX

скрытых файлов:

YY

1.7. Вывести на экран дату, время, список зарегистрировавшихся пользователей, uptime системы.

1.8. Вывести количество процессов, запущенных от имени текущего и от имени пользователя root в формате:

Процессов пользователя:

<User>

XX

Процессов пользователя root:

YY

1.9. Найти и вывести 5 процессов, потребляющих больше всего памяти в системе. Подсказка: man ps

1.10. Вывести файлы и каталоги из домашнего каталога пользователя, упорядочив их по размеру. Подсказка: использовать команды du и sort.

1.11. Разработать сценарий, который выводит файлы из текущего каталога в следующем порядке:

1. Каталоги.
2. Обычные файлы.
3. Символьные ссылки.
4. Символьные устройства.
5. Блочные устройства.

Формат вывода:

Каталоги:

drwxr-xr-x 2 root root 560 сен 13 01:34 block/

drwxr-xr-x 2 root root 120 сен 13 01:34 bsg/

drwxr-xr-x 3 root root 60 июн 19 06:41 bus/

drwxr-xr-x 2 root root 3680 сен 13 01:34 char

...

Обычные файлы:

...

Символьные ссылки:

...

Символьные устройства:

...

Блочные устройства:

...

Когда сценарий будет готов, скопируйте его в каталог ~/bin для тестирования. Далее протестируйте сценарий для каталогов /, /dev, /tmp.

Подсказка: ls -l /dev | grep ^b

### 2. Перенаправление стандартного ввода/вывода

Полезная шпаргалка: <http://www.catonmat.net/download/bash-redirections-cheat-sheet.pdf>

Краткий справочник:

* > - записать стандартного вывод (stdout) в файл (содержимое файла будет безвозвратно утеряно)
* 1> - полностью аналогично предыдущему
* 2> - аналогично, но для вывода ошибок (stderr)
* &> - аналогично, но для двух стандартных выводов сразу - stderr и stdout
* >> - **до**писать стандартный вывод (stdout) в файл (содержимое файла будет сохранено)
* 2>> - аналогично, но для stderr
* &>> - аналогично, но для двух стандартных выводов сразу - stderr и stdout
* < file - перенаправить файл в стандартный ввод (stdin)
* > &2 - всё, что программа выводит в stdout, будет перенаправлено в stderr
* |& - поток, объединяющий stderr и stdout (по умолчанию потоки работают только для stdout)
* /dev/null - "чёрная дыра", в неё можно отправить любой ненужный вывод

Обычно перенаправление вывода размещают после команды - наиболее логичным способом:

echo error >&2

ls > /tmp/list

Однако так тоже правильно, и иногда удобно, чтобы абстрагироваться от перенаправляемого файла и сосредоточиться на аргументах команды:

>&2 echo error

> /tmp/list ls -l

< /tmp/list grep 1

Основная сложность проявляется в том, что и stderr и stdout по умолчанию выводятся на экран и никак не отличаются друг от друга. Проверить stderr это или stdout можно перенаправлением в /dev/null.

Например:

ls /non-existent

ls: невозможно получить доступ к /non-existent: Нет такого файла или каталога

# попробуем спрятать ошибку. Так не работает, так как ошибка выводится в stderr

ls /non-existent > /dev/null

# теперь пробуем перенаправить поток ошибок:

ls /non-existent 2> /dev/null

# получилось! Теперь на экран ничего не выводится

# можно вообще заблокировать любой вывод программы

# это нужно, когда мы просто хотим узнать код возврата

ls /non-existent &> /dev/null

# данный способ годится только для Bash

# в более старых системах приходилось делать так:

ls /non-existent 2> /dev/null >&2

Частая ошибка - попытка найти что-то в выводе программы с помощью grep, но вывод осуществляется "мимо" stdout, в итоге мы получим весь вывод просто на экран.

# вы можете использовать grep, чтобы найти нужный параметр в справке

ls --help | grep ссыл

следовать по символьным ссылкам в командной

следовать по всем символьным ссылкам в командной

-L, --dereference показывая информацию для символьной ссылки,

показывать информацию о файле, на который ссылка

ссылается

# но с ошибками не так просто! Эта команда никак не поможет с поиском:

ls /{1..100} | grep 99

# чтобы найти ошибку в гуще других, нужно чуть изменить команду

ls /{1..100} |& grep 99

Удобство перенаправления потоков в Bash (по сравнению с такими языками как C) компенсируется тем, что символы >, <, |, которые нередко встречаются в повседневной работе, нужно экранировать.

Вы можете создавать сценарии, которые также принимают на стандартный ввод (stdin) некий ввод и обрабатывает его с помощью стандартных команд. Для этого нужно просто не указывать файл для обрабатывающей команды. Например, нужно разработать сценарий (mygrep.sh), который принимает на стандартный ввод текст и выводит строки, совпадающие с заранее запрограммированным шаблоном.

#!/bin/bash

grep -i bash

Данный сценарий будет фактически упрощённым аналогом команды grep, где все необходимые аргументы уже заданы. Его можно использовать следующим образом:

cat /etc/passwd | ./mygrep.sh

< /etc/passwd ./mygrep.sh

Для тестирования следующих заданий выполните подготовительные действия:

* создайте в домашнем каталоге файл numbers.txt, в который запишите 10 000 натуральных чисел (см. команду seq);
* создайте в домашнем каталоге файл users.txt, в который запишите имена всех пользователей системы (используйте cut и /etc/passwd);
* создайте в домашнем каталоге файл bash.txt, в который запишите содержимое двоичного файла /bin/bash в текстовом виде (используйте od);
* создайте в домашнем каталоге файл services.txt, который будет идентичным файлу /etc/services (скопировать файл с новым именем).

2.1. Разработать сценарий, который ведёт в файле /tmp/run.log последовательный журнал запусков:

* в конец журнала добавляет строку с датой и временем запуска сценария (используйте команду date для фиксации даты и времени запуска сценария);
* в стандартный вывод (stdout) - выводит фразу "Hello, World!"
* в стандартный вывод ошибок (stderr) - выводит количество предыдущих запусков программы (для этого достаточно подсчитать количество строк в журнале).

Убедиться в правильности работы программы и выводе различных сообщений в различные потоки вывода:

2.1.sh > /dev/null # должен вывести счётчик запусков, счётчик должен увеличиваться

2.1.sh 2> /dev/null # должен вывести Hello, World!

2.2. Разработать сценарий, который запускает сценарий, разработанный в предыдущем задании, а также открывает журнал запусков предыдущего сценария в программе less.

2.3. Разработать сценарий, который для bash.txt (созданного ранее):

* сохранит строки, которые содержат сочетание символов 000000 в файл /tmp/zeros;
* сохранит строки, которые не содержат сочетания символов 000000 - в файл /tmp/nozeros;
* выведет 10 первых и 10 последних строк от каждого из файлов /tmp/zeros и /tmp/nozeros.

2.4. Разработать сценарий, который считывает построчно стандартный ввод и выводит только те строки, которые содержат слово bin целиком в стандартный вывод ошибок. Для проверки сценария используйте конвейер с командой 'ls /.

2.5. Разработать сценарий, который для всех файлов с расширением txt в домашнем каталоге пользователя:

* выведет список таких файлов;
* выведет суммарный размер в байтах и строках для файлов с расширением txt.

Подсказка. Для решении этой задачи создайте временный файл в каталоге /tmp, по окончании работы сценария удалите его.

### 3. Аргументы командной строки и переменные

В данном разделе необходимо разработать сценарии, которые не только автоматизируют какие-то действия, но и принимают некоторые параметры от пользователя. Таким образом, созданные сценарии можно повторно использовать для различных входных данных.

Аргументы командной строки в сценарий Bash передаются также как и в обычную программу вроде grep или wc. Внутри сценария используются специальные переменные $1, $2 и т.п. - по порядку аргументов.

$ ./myscript.sh foo bar 1 2 3

Первые два аргумента: foo bar

Сумма третьего, четвёртого и пятого аргументов: 6

$ cat myscript.sh

#!/bin/bash

echo "Первые два аргумента: $1 $2"

echo "Сумма третьего, четвёртого и пятого аргументов: $(($3+$4+$5))"

Исследуйте, как ведут себя специальные переменные Bash самостоятельно:

$1, $2 - первый аргумент, второй аргумент и т.п.; $# - количество аргументов командной строки, переданные сценарию; $\* - все аргументы, переданные сценарию, объединённые в один; $@ - все аргументы, переданные сценарию, по отдельности.

При обработке аргументов всегда используйте экранирующие кавычки. Это обезопасит код от странных ошибок, связанные с тем, что во входных данных могут содержаться пробелы и специальные символы:

# НЕПРАВИЛЬНО!

echo $1 $2 $3

grep foo $1

echo $\*

du $@

# Правильно:

echo "$1 $2 $3"

grep foo -- "$1"

echo "$\*"

du -- "$@"

# исключение составляет переменная $#, это всегда число:

grep -m$# /etc/passwd -- "$1"

# при передаче аргумента (напр., имени файла) в другую команду, всегда отделяйте

# пользовательские данные от аргументов команды символами --. Это гарантирует,

# что ваша команда будет работать даже для файлов, которые начинаются с символа -.

grep "$1" /etc/passwd # неправильно, будет ошибка, если $1 == --help

grep /etc/passwd -- "$1" # правильно

Переменные в Bash аналогичны аргументам. Однако, в отличие от аргументов, переменную, прежде чем использовать, необходимо объявить:

NAME=Вася

# если в данных содержится пробел, их надо экранировать

FIO="Иванов Иван Иванович"

# в переменную удобно записывать вывод другой команды:

USERS=$(grep /bin/bash /etc/passwd | cut -d: -f1)

# при копировании одной переменной в другую экранирование не требуется:

FIO\_COPY=$FIO # экранирование не нужно

# аналогично, не нужно экранирование при записи в переменную

# результата работы какой-либо команды

FILES=$(ls) # экранирование не нужно

# однако при объединении нескольких команд без экранирования не обойтись:

FIO\_AND\_ID="$FIO $(id)" # требуется экранировать, так как объединяем результат

Некоторые полезные переменные:

$USER # имя текущего пользователя

$HOST # имя узла

$HOME # путь до домашнего каталога текущего пользователя

$RANDOM # случайное число

$LANG # текущие настройки локализации (язык и кодировка)

$$ # PID текущего процесса (сценария)

$? # код возврата предыдущей команды

3.1. Разработать сценарий, который выводит на экран количество переданных ему аргументов. Скопировать его в $HOME/bin для дальнейшего использования его в других сценариях.

3.2. Разработать сценарий, который вызывает предыдущий сценарий дважды: первый раз с объединённым полным списком аргументов, второй раз - со списком всех переданных ему аргументов по отдельности.

3.3. Разработать сценарий-тестировщик, который будет тестировать ваши сценарии на различные входные данные. Первым аргументом данный сценарий должен принимать имя файла сценария, который необходимо протестировать. Запрограммируйте следующие сценарии тестирования:

* с аргументами "1" "2" "3";
* с пятью случайными числами (см. переменную $RANDOM);
* с аргументами "foo" "bar" "foobar" "foo bar";
* с аргументами "foo" "--foo" "--help" "-l".

Протестируйте с помощью разработанного сценария-тестировщика два предыдущих сценария.

3.4. Перепишите задание 1.11. так, чтобы исследуемый каталог передавался через аргумент командной строки.

3.5. Разработать сценарий, который вызывает команду grep и принимает следующие аргументы:

* текст, который нужно найти;
* файл, в котором нужно найти этот текст;
* максимальное количество строк, которое нужно вывести на экран.

Вывод команды grep отсортировать и пронумеровать.

3.6. Разработать сценарий, который выводит в одну строку имя пользователя, его домашний каталог, а также количество символов в этих двух переменных. Например: root /root 9. Подсказка: изучите аргументы команды echo, wc, математические вычисления в Bash - $(()).