ЛАБОРАТОРНАЯ РАБОТА № 2. УПРАВЛЕНИЕ КОДОМ С ПОМОЩЬЮ GIT И ДОКУМЕНТИРОВАНИЕ ПРОЕКТА

Содержание

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

CLI — Command-Line Interface

Работу ОС LINUX можно представить в виде функционирования множества взаимосвязанных процессов. При загрузке системы сначала запускается ядро (процесс 0), которое в свою очередь запускает командный интерпретатор shell (процесс 1).

Взаимодействие пользователя с системой LINUX происходит в интерактивном режиме посредством командного языка. **Командная оболочка** операционной системы или **командный интерпретатор** — **shell** — интерпретирует вводимые команды, запускает соответствующие программы (процессы), формирует и выводит ответные сообщения.

Shell — это интерфейс, обеспечивающий взаимодействие между ядром и пользователем. Интерфейс shell очень прост. Обычно он состоит из приглашения в командной строке, в которой пользователь вводит команды. Shell не только интерпретирует команды, но и создает среду, которую вы можете конфигурировать и программировать. У командной оболочки есть свой язык программирования, который позволяет писать программы, содержащие достаточно сложные последовательности команд Unix-подобных систем, таких как Linux, FreeBSD и macOS. Язык программирования командной оболочки (shell) обладает многими свойствами обычного языка программирования, в частности в нем предусмотрено использование циклов и условных переходов.

Каждому пользователю системы предоставляется свой собственный пользовательский интерфейс или shell. Пользователи могут модифицировать свои командные оболочки в соответствии с конкретными потребностями. В этом смысле shell пользователя функционирует скорее как операционная среда, которой пользователь может управлять по своему усмотрению.

Разработано несколько разновидностей shell, например Bourne, Korn, Bash, Z shell и C-shell. Bourne-shell был разработан в Bell Labs для System V. C-shell разработан для версии BSD. Kom-shell — это усовершенствованный вариант Bourne-shell. Bash (от англ. Bourne again shell) — усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Bash — одна из наиболее популярных современных разновидностей командной оболочки; особенно популярна в среде Linux, где она часто используется в качестве предустановленной командной оболочки. Z shell, zsh — одна из современных командных оболочек, которая является расширенным вариантом bourne shell с большим количеством улучшений. Начиная с macOS Catalina 10.15 zsh является предустановленной командой оболочке. Ранее в macOS по умолчанию использовалась командная оболочка bash. В современных версиях Unix, включая Linux и macOS, представлены все вышеназванные shell, что даёт пользователю возможность выбора.

Файловая структура: каталоги и файлы

В операционной системе UNIX все файлы организованы в каталоги, которые, в свою очередь, иерархически соединены друг с другом, образуя одну общую

файловую структуру. При обращении к файлу необходимо указывать не только его имя, но и место, которое он занимает в этой файловой структуре. Можно создавать любое количество новых каталогов, добавляя их к файловой структуре.

Команды работы с файлами в UNIX-подобных ОС могут выполнять сложные операции, например, перемещение и копирование целых каталогов вместе с их подкаталогами. Такие команды, как find, cp, mv и ln, позволяют находить файлы, копировать их и перемещать из одного каталога в другой, а также создавать ссылки.

Файлы в UNIX-подобных ОС организованы в иерархическую систему каталогов. Каталог может содержать файлы и другие каталоги. В этом смысле каталоги выполняют две важные функции. Во-первых, в каталоге хранятся файлы, подобно папкам в ящике картотеки, а во-вторых, каталог соединяется с другими каталогами, как ветвь дерева соединяется с другими ветвями.

Именно таким образом каталог может соединяться с другим каталогом. Из-за сходства с деревом такую структуру часто называют древовидной структурой. Данную структуру можно назвать структурой «родители-потомки».

Аналогичным образом любой каталог является подкаталогом другого каталога. Каждый каталог может содержать множество подкаталогов, но сам должен быть потомком только одного родительского каталога. Вверху файловой системы находится корневой каталог (обозначается символом «косая черта»), от которого ответвляются другие каталоги. Каждый каталог может содержать несколько других каталогов или файлов, но родительский каталог у него всегда бывает только один.

Файловая структура UNIX-подобных ОС разветвляется на несколько каталогов, начиная с корневого, /. В корневом каталоге имеется несколько системных каталогов, которые содержат файлы и программы, относящиеся к самой ОС. Корневой каталог, кроме того, содержит каталог home, который может содержать начальные каталоги всех пользователей системы. Начальный каталог каждого пользователя, в свою очередь, будет включать в себя каталоги, который пользователь создаёт для своих нужд. Каждый из этих каталогов тоже может содержать каталоги. Все эти вложенные каталоги ответвляются от начального каталога пользователя.

Получить доступ к каталогу можно либо по имени, либо сделав его каталогом по умолчанию. Каждому каталогу при создании присваивается имя. Этим именем можно пользоваться для доступа к файлам, находящимся в данном каталоге. Если при проведении какой-либо операции над файлами имена каталогов не указываются, то используется каталог по умолчанию, который называют рабочим каталогом. В этом смысле рабочий каталог — это каталог, в котором вы в данный момент работаете. При регистрации в системе в качестве рабочего принимается ваш начальный каталог, имя которого обычно совпадает с вашим регистрационным именем.

Структура файловой системы macOS

Домены (Domains)

В многопользовательской системе доступ к различным ресурсам системы становится неоднородным. Часть ресурсов нужна большинству пользователей, например прикладные программы. Доступ к ним должен быть открыт для всех пользователей. Некоторые ресурсы являются критически важными для работы самой системы, поэтому желательно ограничить доступ к ним со стороны обычных пользователей. Помимо общих ресурсов, каждому пользователю хотелось бы хранить и использовать некоторые ресурсы индивидуально, чтобы другие пользователи не имели к ним доступа, например персональные сообщения электронной почты.

Поэтому macOS заметно строже относится к расположению файлов в файловой системе. Практически каждый файл имеет свое заранее определенное место. Для файлов различного типа и назначения определены различные стандартные места. Не рекомендуется сохранять файлы в произвольных местах и без необходимости перемещать или переименовывать файлы. Изменение местоположения или переименование системных файлов может привести к полной или частичной потере работоспособности системы. Алгоритм работы Finder'a также рассчитан на наличие определённого порядка в размещении информации.

Чтобы разделить «сферы полномочий» и навести во всем этом порядок, в macOS введено понятие домена файловой системы или просто домена.

Домен (domain) — это некоторая область файловой системы, имеющая чётко выраженную, заранее определённую структуру, что облегчает поиск доменов и работу с ними. Домены различаются наполнением — конкретными файлами, хранящимися в определённых местах, и степенью доступности этих файлов. Всего имеется четыре домена:

User, Local, Network, System

User (Пользователь). Это домен пользователя, который в данный момент зарегистрировался в системе. Папка верхнего уровня этой структуры называется домашней папкой данного пользователя. Домашняя папка пользователя может находиться как на загрузочном диске, так и на другом компьютере в сети. Пользователь сам решает, что и как хранить в своём домене.

Local (Локальный). Это домен программ, документов и других ресурсов, которые не являются критически важными для работы системы, но должны быть доступны всем пользователям данного компьютера. Администраторы системы могут добавлять, удалять или модифицировать элементы этого домена. Домен всегда располагается на загрузочном диске данного компьютера.

Network (Сеть). Это домен для программ, документов и других ресурсов, которые должны быть доступны всем пользователям локальной компьютерной сети. Обычно ресурсы этого домена помещаются на один из серверов сети и находятся в ведении администратора сети.

System (Система). Это домен системного программного обеспечения, установленного фирмой Apple. Всегда находится на загрузочном диске. Содержит программы, критически важные для работы всей системы. Пользователь не может добавлять, удалять или изменять содержимое этого домена. Только «главный администратор» — `root` — имеет исключительные права на внесение изменений в содержимое этого домена.

Системные каталоги в Linux и macOS

Корневой каталог, являющийся началом файловой структуры ОС Linux, содержит ряд системных каталогов. Системные каталоги содержат файлы и программы, служащие для управления системой и её сопровождения. Многие из этих каталогов содержат подкаталоги с программами, предназначенными для выполнения конкретных задач.

/bin

bin — это сокращённо от `binaries' (т.е. двоичные или выполняемые файлы). Здесь находится много важных системных программ. Большинство основных команд Unix-подобных ОС находятся в этом каталоге.

/dev

"Файлы" в dev известны как драйверы устройств. Они используются для доступа к устройствам и ресурсам системы, таким как диски, модемы, память и т.д. Например, вы можете читать данные из файла, точно также вы можете читать входные сигналы от мыши, имея доступ к /dev/mouse. Имена файлов, начинающиеся на fd, — это дисководы гибких дисков. fd0 — первый дисковод, fd1 — второй. Различные /dev/ttys, /dev/cua устройства используются для доступа к последовательным портам. Например, /dev/ttys0 относится к `COM1' под MS-DOS.

Устройства, имена которых начинаются с hd или sd, обеспечивают доступ к жёстким дискам. /dev/hda относится ко всему первому жёсткому диску, а hda1 только к первому разделу /dev/hda. Аналогично и для sda. sda — весь жёсткий диск, подключенный как Primary Master, sda1 — первый раздел на данном жёстком диске. Отличие между hda и sda заключается в том, что hda — это подключение дисков по шине IDE, а sda — по интерфейсу SATA.

Устройства с именами /dev/tty относятся к "виртуальным консолям" вашей системы (доступ путем нажатия alt-F1, alt-F2 и т.д.). /dev/tty1 соответствует первой, /dev/tty2 соответствует второй и т.д.

Устройства, чьи имена начинаются на /dev/pty, это «псевдотерминалы». Они используются для входа с удалённых "терминалов". Например, если ваша машина в сети, вход к вам по telnet будет использовать одно из устройств /dev/pty.

/etc

etc содержит файлы конфигурации системы. Например /etc/passwd (файл паролей), /etc/groups (файл групп), /etc/rc (командный файл инициализации) и т.д.

/sbin

В sbin находятся важные исполняемые системные файлы, используемые системным администратором.

/home

home содержит домашние каталоги пользователей.

/lib

lib содержит образы разделяемых библиотек (shared library images). Эти файлы содержат код, который могут использовать многие программы. Вместо того, чтобы каждая программа имела свою собственную копию этих выполняемых файлов, они хранятся в одном общедоступном месте — в /lib. Это позволяет сделать выполняемые файлы меньше и сэкономит место в системе.

/proc

proc — это "виртуальная файловая система" procfs, в которой файлы хранятся в памяти, а не на диске. Они связаны с различными процессами, происходящими в системе, и позволяют получить информацию о том, что делают программы и процессы в указанное время.

/tmp

Многие программы нуждаются в создании рабочих файлов, которые нужны короткое время. Каноническое место для этих файлов в /tmp (там обычно чаще проводится уборка мусора).

/usr

usr — состоит из ряда подкаталогов, которые в свою очередь содержат наиболее важные и полезные программы и файлы конфигурации, используемые системой. Различные каталоги, описанные выше, необходимы для нормального функционирования системы.

/usr/bin — для различных программ UNIX. Он содержит большинство выполняемых программ, которых нет ни в каких других местах, например, в том же /bin их нет.

/usr/etc — также как и /etc, содержит всевозможные системные программы и конфигурационные файлы.

/usr/include — содержит include-файлы (header - файлы) для компилятора Си.

/usr/lib — содержит библиотеки-"заглушки" и "статические" библиотеки, эквивалентные файлам из /lib. При компиляции программа "связывается" с библиотеками, находящимися в /usr/lib, которые в свою очередь направляют

программы обращаться в /lib, если им нужен актуальный код. Кроме того, многие другие программы хранят в /usr/lib свои конфигурационные файлы.

/usr/local — в большой степени похож на /usr — он содержит различные программы и файлы, несущественные для системы.

/usr/man — содержит страницы руководств пользователям по различным утилитам и командам.

/usr/src — содержит исходные коды (неоткомпилированные программы) для различных программ вашей системы. Наиболее важная вещь здесь это каталог */usr/src/linux*, в котором содержатся исходные коды ядра Linux.

/var

var содержит каталоги, которые часто меняются в размере или имеют тенденцию быстро расти.

ОСНОВНЫЕ КОМАНДЫ

Для выполнения заданий этого раздела Вам потребуется знание следующих команд:

- `who` – вывод информации об активных пользователях;
- `whoami`, who am i – вывод информации о пользователе терминала;
- `uname` – вывод информации о версии операционной системы;
- `echo` – вывод сообщений на терминал;
- `banner` – вывод сообщений на терминал прописными буквами;
- `man` – вызов оперативной справочной системы;
- `date` – вывод текущей даты;
- `cal` – календарь;
- `write` – передача сообщений на терминал другого пользователя;
- `wall` – передача сообщений на все терминалы;
- `mesg` – разрешение/запрет вывода сообщений от других пользователей;
- `mail` – отправка/получение почты;
- `ps` – печать списка запущенных программ;
- `clear` – очистка экрана;
- `login` – регистрация в системе;
- `logout` – выход из системы;
- `exit` – завершение текущего `shell`.

Если команда читает текст с терминала, как, например, команды передачи сообщений, конец текста следует обозначить символом `^D`. Тем самым программе передается символ «конца файла».

Справочная система man может содержать одноимённые статьи в разных разделах. Например, umask из раздела 1 – это команда, из раздела 2 – системный

вызов. Если раздел не указан, по умолчанию отображается, как правило, статья раздела 1. Для доступа к разделу 2 следует ввести:

```
man 2 umask
```

или

```
man -S 2 umask
```

В некоторых системах поддерживаются оба варианта, в некоторых какой-либо один.

Строка-аргумент команды echo может содержать управляющие символы, обозначения которых аналогичны используемым в программах на языке С. Это позволяет передавать одной командой форматированный текст. Такой текст следует заключать в двойные кавычки и указывать опцию -e. Ознакомление с другими опциями оставлено в качестве упражнения (посмотрите man).

Пример.

С помощью команды echo вывести в столбик цифры от 0 до 9.

Решение.

```
echo -e "0\n1\n2\n3\n4\n5\n6\n7\n8\n9\n"
```

КОМАНДЫ РАБОТЫ С ФАЙЛАМИ И КАТАЛОГАМИ

Для выполнения приведённых ниже заданий рекомендуется использовать следующие:

- pwd – вывод абсолютного маршрутного имени текущего рабочего каталога;
- cd – изменение рабочего каталога;
- ls – вывод информации о содержимом каталога;
- mkdir – создание каталога;
- rmdir – удаление каталога;
- touch – обновление временной метки файла;
- cp – копирование файлов;
- mv – перемещение или переименование файла;
- cat – объединение и вывод на экран содержимого файлов;
- more – постраничный просмотр содержимого файла;
- rm – удаление файла;
- alias – создание псевдонима.

Для выполнения ряда заданий потребуется знание шаблонов генерации имен файлов:

- ? (вопросительный знак) соответствует любому одному символу, кроме первой точки;
- [ ] (квадратные скобки) определяют группу символов (выбирается один символ из группы);
- - (знак «минус») определяет диапазон допустимых символов;
- ! (восклицательный знак) отвергает следующую за ним группу символов;

- * (символ «звёздочка») соответствует любому количеству символов, кроме первой точки.

Именно интерпретатор команд заменяет метасимволы *, ?, [ ] на соответствующие им имена файлов. Программа вместо аргументов, содержащих метасимволы, получит сформированный интерпретатором список — результат подстановки, а число аргументов может измениться. Вот почему от команды

```
10cp
*.cpp
*.exe
```

не следует ожидать попарной подстановки, привычной на персональных ЭВМ, где она реализуется средствами самой программы.

С другой стороны, подход UNIX позволяет единообразно применять символы экранирования “, ‘ и, тем самым, более гибко управлять поведением программы.

Пример 1.

Вывести на экран содержимое всех файлов, имеющих в имени символ e.

```
cat *e*
```

Файлы, имя которых начинается с символа . (точка), не подпадают под указанный шаблон, поэтому если требуется отобразить и такие файлы, команду следует дополнить:

```
cat *e* .*e*
```

Пример 2.

Вывести в столбик список файлов, в именах которых второй символ – пробел.

```
ls -1 ?\ *
```

Пример 3.

Переместить файл listing из каталога work в каталог Development_And_Research. Других каталогов и файлов, имя которых начинается на букву D в текущем каталоге нет.

```
mv work/listing D*/
```

СТРУКТУРА GIT

Git — распределенная система контроля версий. Git представляет собой серверную часть, которая отвечает за версионное хранение документов и обработку запросов, клиентскую часть, позволяющую формировать запросы к серверу Git и набора конфигурационных файлов, хранящих настройки пользователя и позволяющих адаптировать Git для самых разных задач.

Git состоит из набора консольных утилит и файла конфигурации. За счёт такой архитектуры достигается переносимость на разные платформы и позволяет создавать графические клиенты для работы с проектами.

Структура репозитория (отдельного проекта) Git представляет собой каталог файловой системы, в котором хранятся конфигурационные файлы, журналы —

файлы изменений репозитория, файлы индексов, ускоряющие поиск по репозиторию и файлы проекта. Дерево файлов в репозитории не отражает реальное хранение файлов, так как Git изменяет расположение файлов для ускорения доступа. Если при работе с проектом файл изменяется, то в репозитории Git создаётся новый файл. Таким образом при изменения Git не изменяет старые файлы, а создаёт новые. Пользователь же видит «снимок» файловой системы на определённый момент времени (под снимком здесь понимается информация на определённый период времени, т. е. информация, созданная до запрашиваемого периода и изменённая после этого периода). Из репозитория файлы не удаляются. Они могут быть лишь помечены как удалённые.

По умолчанию конфигурация репозитория хранится в каталоге “.git”, корневого каталога проекта. Таким образом достаточно легко превратить любой каталог в репозиторий или импортировать существующий репозиторий вызвав соответствующую утилиту, которая создаст подкаталог с необходимыми файлами или исправит существующие файлы в случае импорта.

Для каждого объекта в репозитории считается SHA-1 хеш, и именно он становится именем файла, содержащего данный объект в директории `.git/objects`. Для оптимизации работы с файловыми системами, не использующими деревья для директорий, первый байт хэша становится именем поддиректории, а остальные — именем файла в ней, что снижает количество файлов в одной директории

В классическом обычном сценарии в репозитории `git` есть три типа объектов — файл, дерево и коммит. Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных поддиректорий, коммит — дерево + некая дополнительная информация (например, родительский(е) коммит(ы), а также комментарий).

В репозитории иногда производится сборка мусора, во время которой устаревшие файлы заменяются на «дельты» между ними и файлами современными, после чего данные «дельты» складываются в один большой файл, к которому строится индекс. Это снижает требования по месту на диске.

Репозиторий `git` бывает локальный и удалённый. Локальный репозиторий — это поддиректория `.git`, создаётся (в пустом виде) командой `git init` и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой `git clone`.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным.

Работа с Git из командной строки

Для работы с Git-репозиторием необходимо установить утилиту `git`. Данная утилита находиться в сети интернет в свободном доступе.

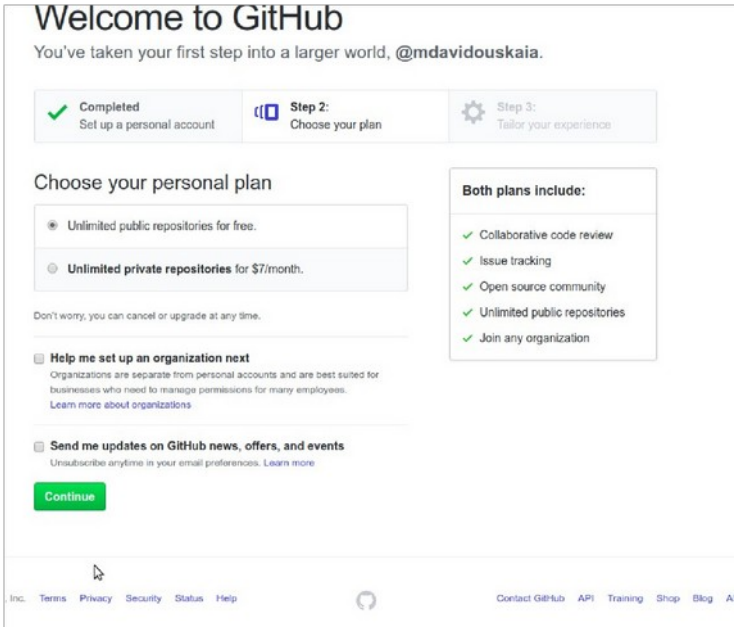
Для настройки конфигурации Git используются следующие команды:

Команда	**Комментарий**
`git config --global user.name Имя`	Задает имя пользователя
`git config --global user.email email@example.com`	Задает email пользователя
`git init`	Если проекта нет, его можно инициализировать командой
`git status`	Состояние: что было отредактировано, что добавлено в индекс до коммита
`git add .`	добавить в индекс все изменения
`git add file.txt`	добавить содержимое файла в индекс
`git commit`	Коммит проекта
`git commit -am "comment" - add + commit`	Коммит без добавления новых файлов
`git branch`	список веток
`git checkout имя_ветки`	Переход в другую ветку
`git checkout -f`	отметить незакоммиченные изменения
`git diff ветка1 ветка2`	Сравнить ветка1 и ветка2
`git merge новая_ветка`	Объединение текущей ветки и новая_ветка
`git reset --hard HEAD`	Вернуть ветку в состояние до слияния
`git push pb master`	Отправить код в ветку master удаленного репозитория pb
`git fetch pb`	Извлечь информацию из репозитория pb

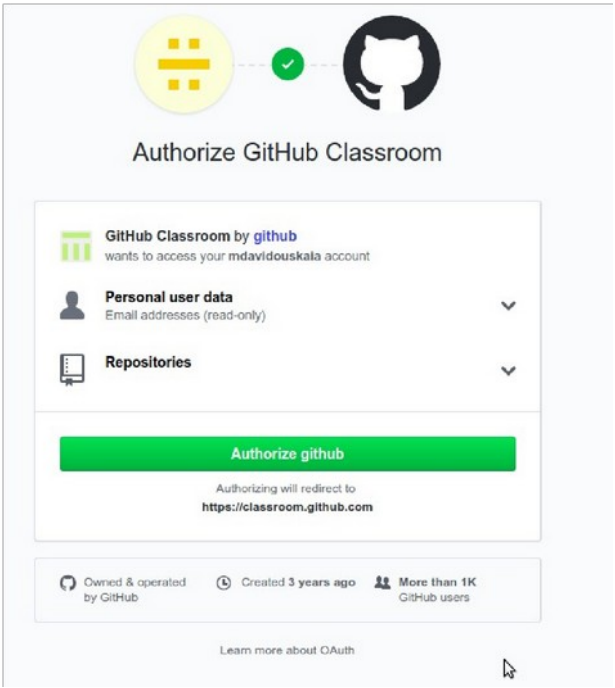
Подключение к проекту на GitHub Education

Для подключения к проекту курса на GitHub выполните следующие действия:

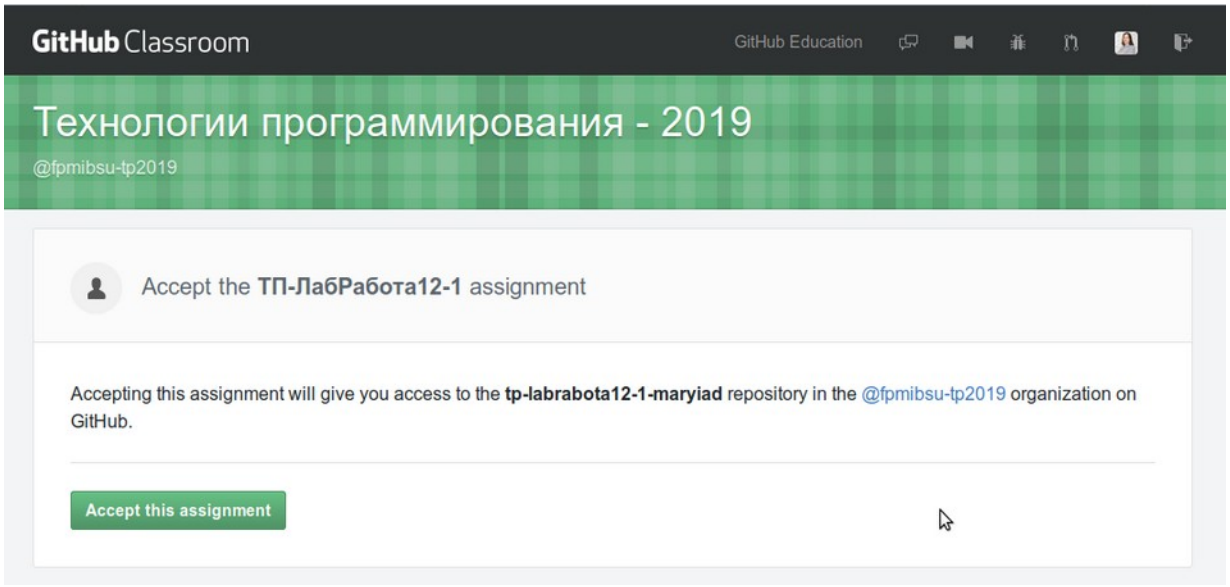
1. Создайте учётную запись на Github (см. рис. ниже)



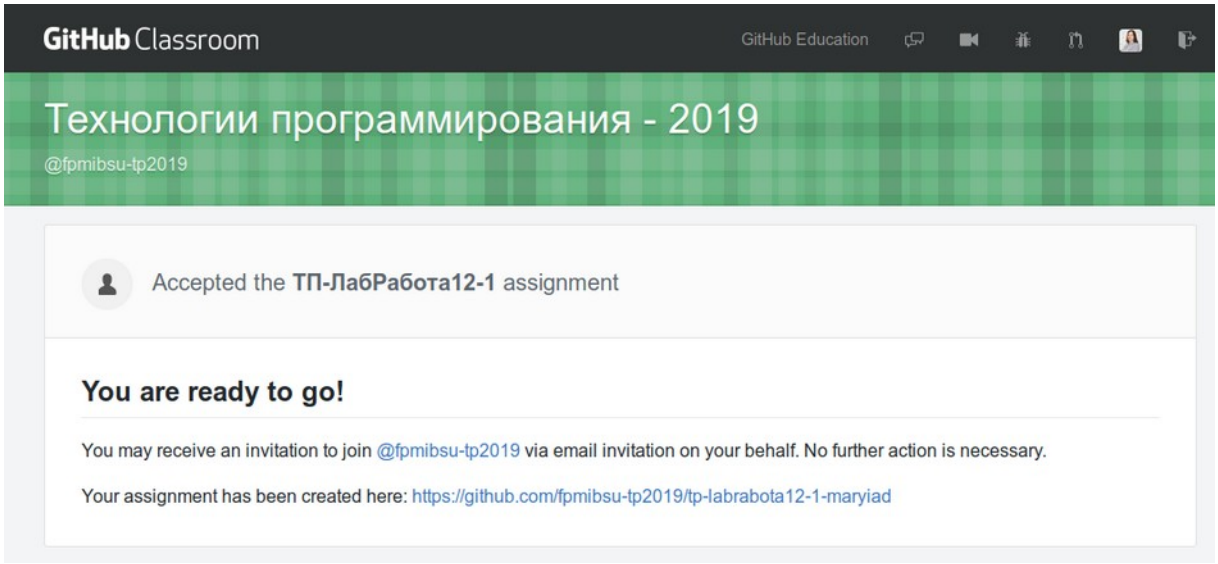
2. Оставаясь авторизованным на github, перейдите по ссылке, опубликованной в курсе «Технологии программирования для мобильных приложений», и разрешите доступ к Вашей учётной записи на Github для приложения GitHub Classroom (см. рис. ниже).



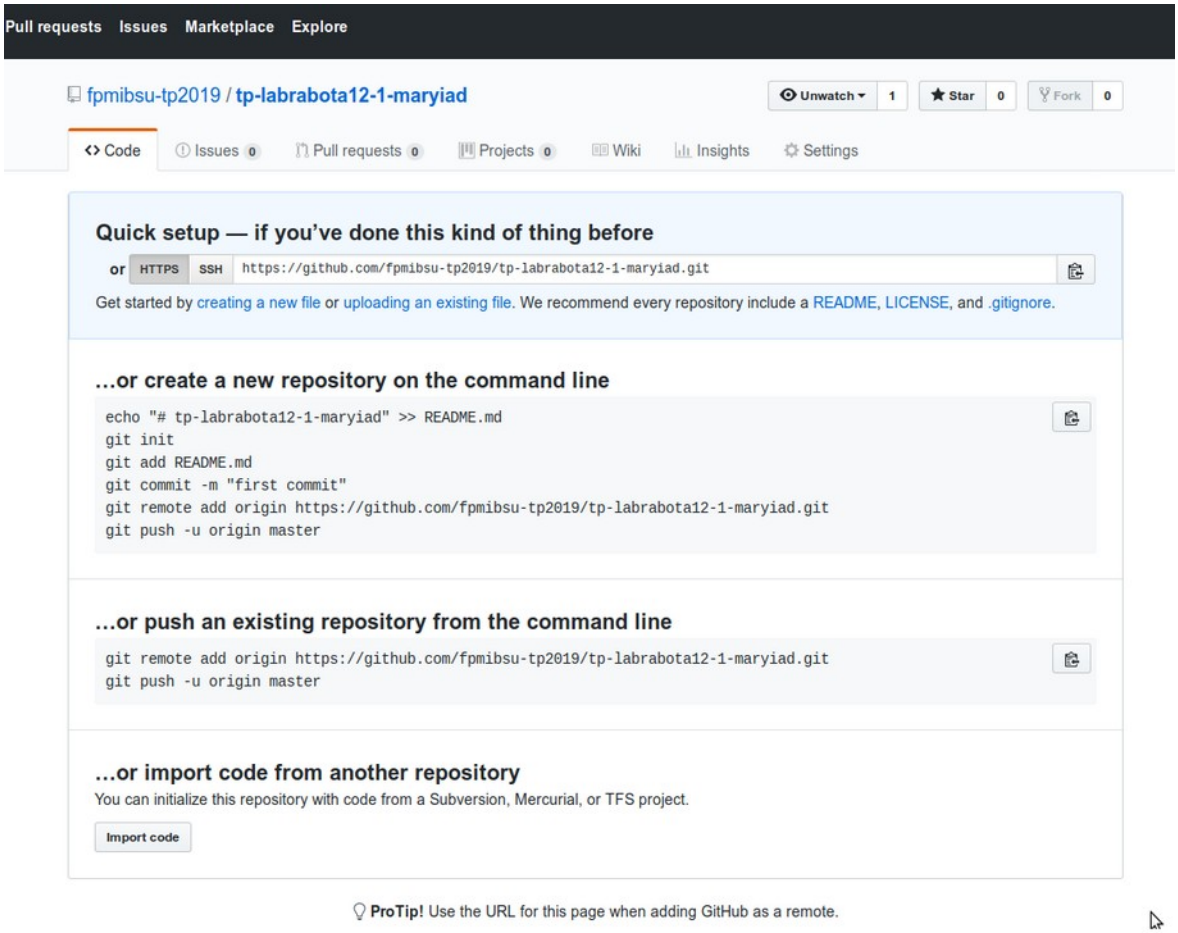
3. Подтвердите получение доступа к Вашему личному репозиторию для проектов, который создан в организации @fpmi-tp2024.



4. На последнем шаге активации доступа Вы получите ссылку на Ваш приватный репозиторий для выполнения задания по лабораторной работе №2. Перейдите по ссылке.



5. В результате Вы будете перенаправлены в Ваш приватный (личный репозиторий) для выполнения задания по лабораторной работе № 2: (см. рис. ниже).



ЗАДАНИЯ

Требования к отчёту

В файле `Readme` проекта на `github` должна быть ссылка на отчёт. Отчет опубликовать в репозитории в папке `docs` или во внешнем хранилище, добавив ссылку на него в файл `Readme` репозитория. Отчёт должен содержать скриншоты и описания этапов выполнения лабораторной работы с примерами команд и ответы на контрольные вопросы.

Критерии оценивания

Оценка 4

Выполнены **Задание 1** (№1-9), **Задание 2** (№1-9), **Задание 3** (№1-8), **Задание 4** (№1-9). Отчёт содержит скриншоты и описание команд. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт может содержать ошибки. Максимальная задержка на предоставление отчёта, публикацию в репозитории и задании курса, а также его защиту — 2 недели.

Оценка 5-6

Выполнены **Задание 1** (№ 1-12), **Задание 2** (№ 1-9), **Задание 3** (№1-8), **Задание 4** (№1-14) ответы на контрольные вопросы (№ 1-4, 6-10). Отчёт содержит скриншоты и описание команд. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и ответы на контрольные вопросы может содержать ошибки. Максимальная задержка на предоставление отчёта, публикацию в репозитории и задании курса, а также его защиту — 1 неделя.

Оценка 7-8

Выполнены **Задание 1** (№ 1-15), **Задание 2** (№ 1-25), **Задание 3** (№1-8), **Задание 4** (№1-20), ответы на контрольные вопросы (№ 1-17). Отчёт содержит скриншоты и описание команд. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и ответы на контрольные вопросы может содержать незначительные ошибки. Максимальная задержка на предоставление отчёта, публикацию в репозитории и задании курса, а также его защиту — 1 неделя.

Оценка 9

Выполнены **Задание 1** (№ 1-15), **Задание 2** (№ 1-25), **Задание 3** (№1-8), **Задание 4** (№1-21), ответы на контрольные вопросы (№ 1-21) на отличном уровне. Отчёт содержит скриншоты и описание команд и ответы на контрольные вопросы без ошибок. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Лабораторная работа опубликована в репозитории и в задании курса в срок, а так же защищена в срок.

ЗАДАНИЕ 1. РАБОТА В КОМАНДНОЙ СТРОКЕ

1. Выведите сообщение «Имя Фамилия», указав Ваше имя и фамилию, в виде нескольких строк с помощью команд `echo` и `banner`. При необходимости установите требуемое приложение.
2. Прочитайте статью справочной системы `man` о пользовании справочной системой.
3. Прочитайте статью справочной системы о команде `uname`. Из какого раздела справочника Вы прочитали статью?
4. Определите имя машины, название и версию операционной системы, с которой Вы работаете. Каков аппаратный тип системы?
5. Выведите дату в форматах `dd-mm-yy, mm-dd-yy hh:mm:ss`.
6. Выведите дату в две строки: на первой – день, месяц, год, на второй – текущее время, снабдив вывод комментарием с Вашим именем и фамилией.
7. Выведите календарь на текущий месяц, на месяц Вашего рождения текущего года.
8. Выведите календарь на будущий год. В каком столбце отображаются воскресенья? Какую команду использовать, чтобы неделя начиналась с понедельника? С воскресенья?

9. Определите порядковый номер текущего дня с начала года.
11. Используя команду `write`, пошлите сообщение на консоль.
12. Используя команду mesg, определите, разрешены ли сообщения на Ваш терминал. Запретите сообщения. Какова будет реакция системы, если кто-нибудь попытается передать Вам сообщение?
13. Прочитайте свою почту в консоли.
14. Привести несколько примеров команды конвертации файла `*.doc` в формат `*.html`.
15. Привести несколько примеров изменения кодировки файла, например из `cp1251` в `utf-8`.

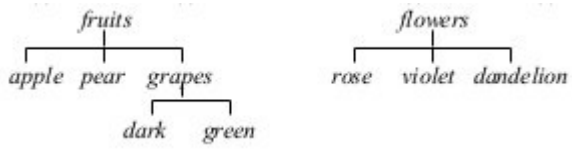
ЗАДАНИЕ 2. РАБОТА С ФАЙЛАМИ

cat

1. Создавать файл `dirlist.txt`, состоящий из 13 строк. Нечётные строки — пустые, чётные — содержат 2) Ваше ФИО, 4) место (город/посёлок) проживания, 6) улица проживания, 8) область проживания, 10) список Ваших хобби и интересов.
2. вывести содержимое файла `dirlist.txt`:
   a в обратном порядке
   b с нумерацией непустых строк
   c с нумерацией всех строк
   d схлопывая подряд идущие пустые строки в одну

Работа с файлами

3. Просмотрите содержимое Вашего домашнего каталога.
4. Определите имя своего домашнего каталога и его родительского каталога.
5. Определите полный путь до Вашего домашнего каталога.
6. Создайте в своём домашнем каталоге подкаталоги вида и выведите их структуру в виде дерева:



7. Находясь в домашнем каталоге, с помощью команды `touch` создайте файл `macintosh` в имеющемся каталоге `apple` и несколько файлов в каталогах `dark` и `green`.
8. Войдите в каталог `flowers`. Находясь в каталоге `flowers`, скопируйте все подкаталоги `fruits` вместе с находящимися в них файлами в специально созданный каталог `basket`.
9. Находясь в каталоге flowers, удалите каталог `fruits`.
10. Создайте в домашнем каталоге директорию `manyfiles`. Создайте в ней с помощью команды и регулярных выражений 100 файлов с именами `a1, a2, a3, .... a100`. **Цикл не используем!**
    Удалите только файлы с нечётными номерами.

11. Находясь в своём домашнем каталоге, создайте ВСЕ подкаталоги из списка в виде дерева, используя **регулярные выражения с помощью ОДНОЙ команды**:
    - A/B/C;
    - A/B;
    - A/B/C/D;
    - A/E.

Находясь в своем домашнем каталоге, удалите все подкаталоги каталога **B** и выведите дерево каталога **A**.

grep

12. Обновите в редакторе nano текстовый файл `dirlist.txt` так, чтобы он содержал в каждой строке как минимум одно название месяца. Вывести строки файла `dirlist.txt`, содержащие строки с определенным месяцем (месяц определяем по дате рождения) и записать их в файл `grep_month_name.txt`
13. Записать строки, не содержащие этот месяц, в файл `grep_other_monthes.txt`.
14. Создать папку grep, переместить в нее файлы созданные в пунктах 12 и 13.
15. Создать папку mac_os_lab и вложенные папки (не менее 3-х уровней вложения и 5 вложенных папок). Создать текстовые файлы в каждой из директорий. Ряд файлов должны содержать слово «`root`». Находясь в папке `mac_os_lab` найти все файлы в этой директории и ее поддиректориях, в которых встречается подстрока root, вывести строки с указанием их номеров.

find

16. Найти все файлы в системе, содержащие в имени «`sh`» (с помощью команд `find` и `locate`). Какая из команд работает быстрее, почему? Доп: Проверить совпадают ли результаты выполнения обеих команд, чем они отличаются, почему?
17. Найти файлы, изменённые за последний час.
18. Удалить весь каталог `manyfiles` со всеми файлами.

Математические операции в консоли

19. Используя редактор `nano`, создайте текстовый файл следующего содержания:

```
25454+11745232
866*12434
1587%12
45225/45*1128
4^37
sqrt(625)
```

Посчитайте все примеры из файла с помощью одной команды. (Вариантов команды существует несколько, засчитывается каждый из них). **Использовать встроенную команду!**

Команды для работы с текстом

20. Используя редактор Vi/Vim, создайте текстовый документ, содержащий список студентов группы. Структура каждой строки: символ табуляции, фамилия, пробел, инициалы.
21. Выведите первые 10 строк файла, последние 15 строк файла.
22. Используя команду `expand`, замените символы табуляции в начале строки на соответствующее количество пробелов и сохраните с новым именем.
23. Используя команды `diff`, `diff3` сравните файлы. В каких случаях используется каждая из команд.
24. Для каких задач используется команды `cmp`, `comm`
25. Используя команду split разбейте файл `dirlist.txt` из 10 строк на два файла по 5 строк.

ЗАДАНИЕ 3. Изучение GitHub

1. Для работы с `Git`-репозиторием необходимо установить утилиту `git`, если она не установлена.
2. Задать данные пользователя (имя, адрес электронной почты), в консоли с помощью команд:

   a **на своем компьютере на глобальном уровне**:

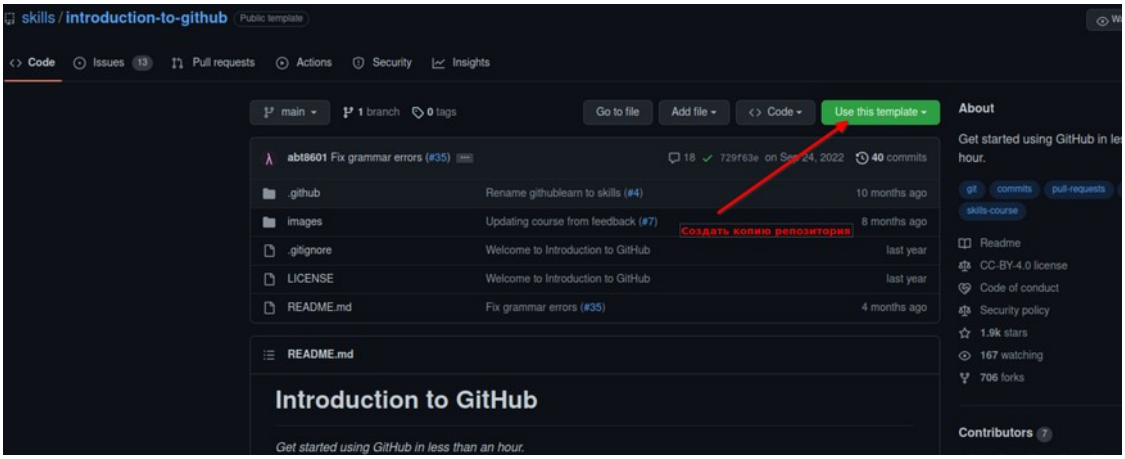
```
git config --global user.name "Ваше имя для подписи коммитов"
git config --global user.email "Ваш емейл, с которым зарегистрированы на github"
```

   b на **компьютере в классе** только в каталоге **КОНКРЕТНОГО** репозитория, когда создадите его копию локально. Данный пункт выполняется для каждого репозитория, если параметры не заданы глобально:

```
git config user.name "Ваше имя для подписи коммитов"
git config user.email "Ваш емейл, с которым зарегистрированы на github"
```

3. Изучить основные команды работы с `git` в консоли (см. `СТРУКТУРА GIT`) или с помощью интерактивного тренажера https://learngitbranching.js.org/.
4. Создать учётную запись на проекте `github`.
5. Откройте в браузере сайт https://skills.github.com
6. Активируйте курс Introduction to GitHub.

7. Прочитайте внимательно инструкцию и создайте копию репозитория с курсом с помощью кнопки `Use this template` и новый репозиторий назовите `lab2-intro-git`

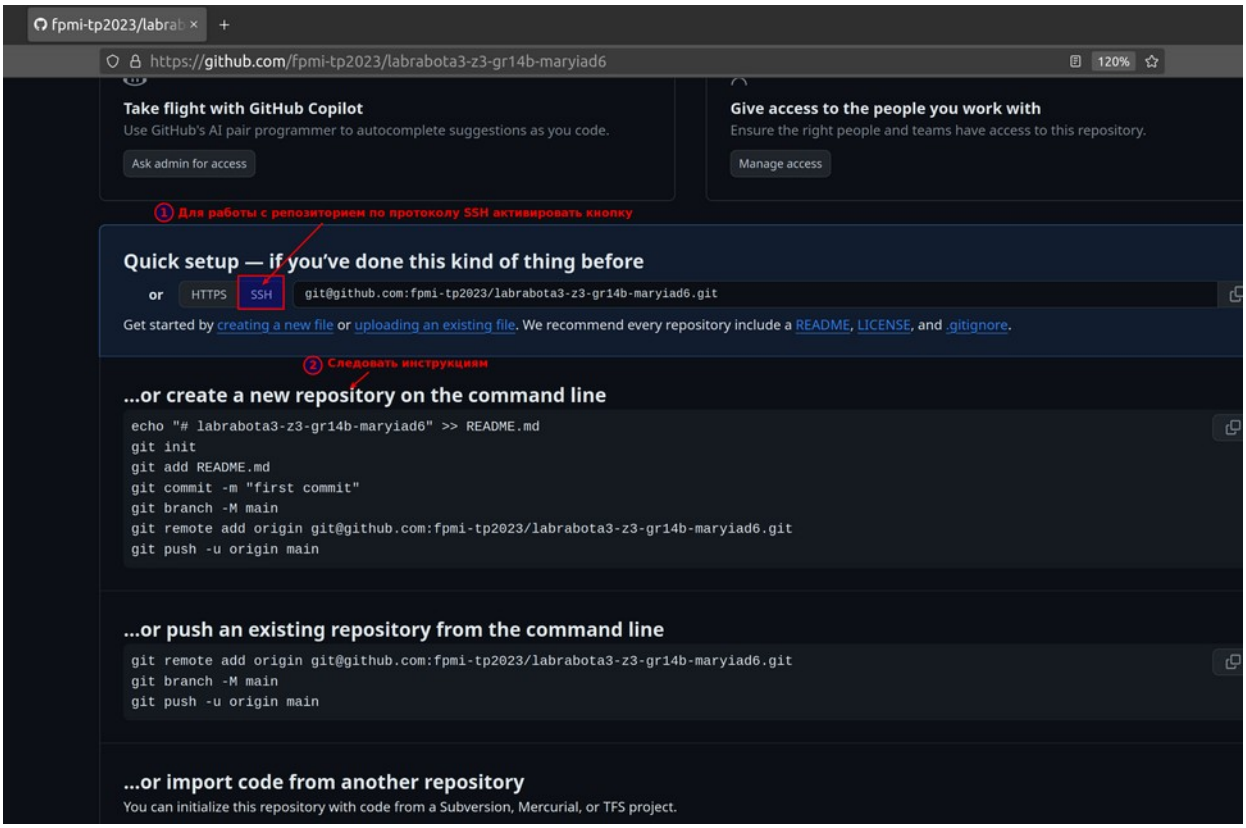


8. Выполните все задания курса.
9. Репозиторий после выполнения не закрывайте и не удаляйте. Ссылку на данный репозиторий необходимо будет добавить в задании 4.
10. Откройте в браузере сайт https://skills.github.com
11. Активируйте курс Communicate using Markdown.
12. Прочитайте внимательно инструкцию и создайте копию репозитория с курсом с помощью кнопки `Use this template` и новый репозиторий назовите `lab2-intro-markdown`. При необходимости задайте переменные окружения для репозитория, определив имя пользователя и email (см. п.2. задания 3 выше).
13. Выполните все задания курса.
14. Репозиторий после выполнения не закрывайте и не удаляйте. Ссылку на данный репозиторий необходимо будет добавить в задании 4.

ЗАДАНИЕ 4. РАБОТА С GIT-РЕПОЗИТОРИЕМ

1. Перейти по ссылке-приглашению, опубликованному для Вашей подгруппы в теме 2 курса на сайте https://edufpmi.bsu.by/.
2. Открыть приложение terminal, в котором будут выполняться дальнейшие команды.
3. Изучить документацию Генерация открытого SSH-ключа в macOS и добавление в Github account. Сгенерировать ssh-ключ и добавить публичный ключ в свою учетную запись на Github. Если ключ генерируется на компьютер в классе, то имя файла для его хранения сохраните с указанием фамилии.

4. Создать каталог репозитория для задания 4, например **`tp-lab2-gr11a`** в домашнем каталоге или каталоге, где размещается свои выполненные задания.
5. Перейти в каталог репозитория.
6. Выполнить инструкции на странице репозитория в консоли macOS. При подключении использовать ссылку для репозитория (см. рис. ниже).



7. Создать файл `Readme` согласно рекомендациям на странице репозитория (см. рис. выше).
8. Добавить файл `Readme` в список отслеживаемых (`git add`) и зафиксировать в репозитории (закоммитить) (`git commit`). Данная последовательность команд обеспечивает создание ветки `Master`[1]. **Если не опубликовать первый коммит и попробовать создать ветку, то ветка `Main` не будет создана!** Опубликовать изменения из локального репозитория во внешнем репозитории на `github`, выполнив команду git push, т. е. запушить.
9. Отправьте изменения во внешний репозиторий. Требуется ли ввод пароля после добавления ssh-ключа?
10. Создайте в локальном репозитории ветку `Development` в консоли и переключитесь в неё.

1 После изменений политик создания веток первая ветка, которая сейчас создается в репозитории носит название `Main`.

11. Добавьте в локальный репозиторий проект из задания 6 лабораторной работы 1.
12. Добавьте в файл `Readme` ссылку на ваши репозитории из задания 3, оформив ссылки, используя синтаксис markdown, например [Пройденный курс Introduction to GitHub](указать URL вашего репозитория).
13. Добавьте изменения в отслеживаемые и выполните `commit`.
14. После внесения изменений слейте с помощью команды `git merge` ветку `Development` и главную ветку `Main`.
15. Отправьте изменения во внешний репозиторий.
16. Переключитесь в ветку `Development.`
17. Выполните несколько изменений файлов (такими изменениями может быть улучшение читаемости кода, добавление комментариев и т. д.). После каждого атомарного изменения выполняйте команду `commit` в репозиторий в консоли. К каждому коммиту добавить осмысленный комментарий, чтобы при необходимости можно было опознать, какие были выполнены изменения.
18. После внесения изменений слейте ветку `Development` и главную ветку `Main`.
19. Отправьте изменения во внешний репозиторий.
20. Выберите git-клиент с графическим интерфейсом `SourceTree/SmartGit/GitKraken/GitHub Desktop`. И ознакомьтесь с документацией, например GitKraken Git Client Tutorial For Beginners, GitKraken Git Client – пример, Методическое руководство по работе с Git в консоли и в утилите SourceTree.
21. Настройте репозиторий в `SourceTree/SmartGit/GitKraken/GitHub Desktop`, указав путь к локальной и внешней версии репозитория.
22. Создайте ветку `Live` и перейдите в неё.
23. Внесите изменения в проект (добавьте новую функциональность) (см. п. 10 текущего задания) и после каждого атомарного изменения передавайте коммит в репозиторий в `SourceTree/SmartGit/GitKraken/GitHub Desktop`. К каждому коммиту добавлять комментарий.
24. После внесения изменений слейте ветку `Live` и главную ветку `Main`. Проиллюстрируйте слияние двух веток в отчёте в `SourceTree/SmartGit/GitKraken /GitHub Desktop`.
25. Отправьте изменения во внешний репозиторий.
26. В отчёте проиллюстрируйте историю коммитов, работы с ветками.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Ответы на контрольные вопросы сохранить в файл `control-questions.md`, оформить с помощью синтаксиса Markdown и опубликовать в репозиторий задания 4. Продемонстрировать использование заголовков и нумерованных списков.

1. Что такое «командная оболочка» / «командный интерпретатор»?
2. Как называется командный интерпретатор и какая его версия в используемой операционной системе?
3. Для каких задач предназначено приложение `Terminal` в macOS?
4. Приведите полный путь к домашней папке пользователя, под которым вы авторизованы в системе. Привести пример команды, позволяющей определить путь к домашнему (текущему) каталогу в консоли.
5. Какой командой можем вывести содержимое каталога системных конфигурационных файлов?
6. С помощью какой команды, можем изменить командный интерпретатор с zsh на bash?
7. В каких случаях выполняется инициализация репозитория? Какой командой?
8. Что такое коммит?
9. Почему перед созданием новой ветки важно создать коммит?
10. В каких состояниях может находиться файл в репозитории? Как происходит изменение состояния файла?
11. Что такое ветка?
12. Какое условие является обязательным для формирования ветки `master` после клонирования репозитория или его инициализации?
13. Что такое HEAD?
14. Способы создания веток.
15. Как узнать текущую ветку?
16. Как переключаться между ветками?
17. С помощью какой команды можно слить две ветки?
18. Как подключить к инициализированному (созданному) локальному репозиторию внешний репозиторий? Какой командой?
19. Какая команда позволяет отменить последний коммит?
20. Для чего предназначена команда rebase?
21. В каких случаях требуется создание форка? Приведите способы создания форка, включая консольную команду.