Для создания Git-репозитория существуют два основных подхода. **Первый** подход — импорт в Git уже существующего проекта или каталога. **Второй** — клонирование уже существующего репозитория с сервера.

**Первый – импорт в гит уже существующего проекта или каталога:**

1. **git config - -global user.name “Your Name”**, а также: **git config - -global user.email “sl1mvsshady@gmail.com”** — информация об ав торе (это значит, что мы обращаемся к конфигурационному глобальному файлу гита и к полю user.email)
2. **git сonfig - -global core.autocrlf true** и **git config - -global core.safecrlf true —** поскольку в файлах разных операционных систем окончания строк разные (разные символы используются) и чтобы у нас по этому поводу не было конфликтов нам нужно настроить гит таким образом, чтобы он понимал в какой операционке мы работаем и добавлял/заменял правильно окончания строк, чтобы не было разницы в символах в конце строк. Первый 2 пункта нужно сделать в начале установке гита и все. Также можно проверить все настройки с помощью команды **git config - - list**, также можно проверять значение каждого ключа в отдельности, например: **git config user.name/user.email**
3. **mkdir** – make directory (создать директорию/каталог/папку(это путь хранения файла.))
4. Перейдем в эту папку т.е. сменим директорию – **cd test\_git/** (change directory)
5. *Т.к. Git – это распределенная система контроля версий основное отличие которой от централизированной системы контроля версия заключается в том, что каждый разработчик имеет на своем ПК репозиторий в виде скрытой папки(кцкц.git),где и будет хранится вся информация об истории коммитов,тегах—ходе разработки проекта. Локальный репозиторий состоит из временного (индекса) и полноценного хранилищ. Сначала идет сохранения в индекс, а уже из индекса мы переносим в полноценное хранилище с помощью команды git commit(как бы окончательно фиксируем все изменения).* С помощью команды **git init** создадим репозиторий. Есть другой способ создания репозитория — команда **git clone**
6. **git status —** команда, которая показывает статус нашего репозитория/проекта, измененные и не добавленные файлы. Она выводит информацию обо всех изменениях, кроме того git status указы вает файлы с неразрешенными конфли ктами слияния и файлы, игнорируемые git.
7. **git add .** — команда, которая позволяет внести в индекс – временное хранилище – изменения, который затем войдут в полноценное хранилище с помощью команды git commit. *Применим данную команду (git add), дабы сохранить во временном хранилище(.git) сделанные нами изменения в файле index.html (созданным в папке test\_git, в которой в свою очередь создано наше временное хранилище(индекс) с окончанием .git)*. Можно написать git add index.html (выбираем именно этот файл и добавляем в индекс), но обычно пишут git add . (это значит, что все файлы, которые у нас есть в папке test\_git будут добавлены во временное хранилище (индекс) включая новые). Теперь, прописав git status мы увидем зеленую надписать index.html, что будет означать, что index.html добавлен в временное хранилище «индекс» (.git).
8. **git commit —** команда с помощью которой мы переносим сохраненные нами изменения в файлах(коде) (в нашем примере это index.html) из временное хранилища (индекса) в постоянное (т.е. как бы фиксируем наши изменения окончательно) *Следует заметить, что коммиты – базовое понятия во всех системах контроля версий. Т.е. коммиты – это некие фиксаторы состояний нашего кода (точки в истории, по которым происходит отслеживание изменений нашего кода) Т.е. то, что мы изменили в нашем файлы (index.html) мы поместили во временное хранилище (индекс)с помощью команды git.add . , а теперь то, что есть в индексе – «закомитим» с помощью команды git commit (тем самым сделав уже полноценный коммит, точку в истории изменения нашего кода).* Удобно писать **git commit -m** (messenge), т.е. описываем наш коммит, нашу точки в истории изменений нашего кода (дабы потом было понятно, что именно изменилось в нашем файле(коде), некая хронология событий нашего файла(кода)). *Например: мы написали header, потом сделали следующие действия: git add . 🡪 git commit -m “added header” (т.е. мы отмечаем какой этам работы был проделан)*. Также очень удобно использовать **git commit - -amend.** Данную команда используется, когда мы закомитили какие-то изменения, но вспомнили, что не дописали какую-то деталь, которая не нуждается в отдельном коммите. Тогда мы пишем её дописываем, а потом используем git commit - -amend, после чего нас выбрасывает из консоли в стандартный блокнот, где вверху золотым написал наш прежний коммит, который мы можем изменить (переключив в блокноте режим insert кнопкой «о» на клавиатуре тем самым зафиксировав то, что мы добавили некую деталь, а потом нажимаем esc, потом двоиточья(:), а потом wq (т.е. write quit). Возращаемся в наш консоль, прописываем git log и убеждаемся, что новый коммит не был создан, но старый изменен. Также для того, чтобы каждый раз не писать git add . , можно прописывать команду **git commit -am “fsfsfsfs”** (add + message). То есть это означает, что мы сразу все что есть в нашей папке добавляем во временное хранилище ( индекс) и сразу коммитим одной командой, но эта команда не работает для новых файлов. (т.е. если мы только создали файл и написали в нем что-то то все равно придется использовать git add . , но уже при следующем дополнении( кода в файле) можно использовать git commit -am “fsfsf”
9. **git log —** разнообразная информация о коммитах в целом, по отдельным файлам и различной глубины погружения в историю. Иногда требуется получить информацию об истории коммитов, коммитах, изменивших

отдельный файл; коммитах за определенный отрезок времени и так далее. Для этих

целей используется команда git log. Также для к этой команде применяются ключи: **git log –p**(подробнейшая информация о каждом коммите), **git log - -stat** (cтатистика изменения файлов, вроде числа измененных файлов, внесенных в них строк, удаленных файлов, **git log --summary** (за информацию по созданиям, переименованиям и правам доступа файлов).

1. **git** **reset —** сбросить все временное хранилище (индекс).Также удалить из временного хранилища (индекса) изменения определенного файла можно с помощью git reset index.html (наш пример).
2. **git diff —** (от слова difference) команда, с помощью которой можно увидеть изменения не внесенные в индекс (временное хранилище) (*в нашем примере, когда мы добавили информацию(код) в наш файл (index.html), который мы создали в папке git\_test в которой в свою очередь находится наш локальный репозиторий .git).* Команду git diff можно считать подмножеством команды git log, определяющую изменения между состоянием нашего кода ( что добавилось и что убралось). Красным – то, что убралось, зеленым – то, что добавилось, при чем нужно понимать, что учитывается и знах переноса строки, потому если мы продолжаем наш код, то самая последняя строка нашего старого кода будет вначале красная т.к. в ней не было знака переноса строки, но потом она опять зеленая т.к. перенос строки добавился из-за того, что мы уже начали что-то добавлять в новой строке и наш файл изменился, а команда git diff как раз призвана для того, чтобы фиксировать все изменения.

Команда **git diff - -cached** ( с ключом –сashed) покажет изменения внесенные в индекс.

**Второй:клонирование из гит уже существуюего репозитория с сервера.**

**git clone url** (url задается либо с помощью HTTPS либо с помощью SSH протокола) — команда с помощью которой может склонировать (скопировать) из гит уже существующий репозиторий к себе на машину просто введя эту команду в гит баше. Если нужно указать не стандартный пусть сохранения этого репозитория, то пишется например /k/welcome/test , где test – это папка которой не существует, но которую создаст гит и в которую он поместит служебную папку .git, в которой будет находится временное локальное хранилище (индекс), а также помимо папки .git в папке test будет наш файлик с кодом ( например index.html) и текстовый файл readme). После того, как мы склонировали репозиторий с гита на свою машину, создаться папка.

**cd папка с нашим клонированным репозиторием —** переход в папку (change directory) нашего склонированного репозитория. (название папки совпадает с тем репозиторем, который мы качали с гита)

**ls -la** — показать список всех файлов в директории, которой я нахожусь.

**cd ../** — возврат назад, в нашу первоначальную директорию

**rm -rf** — удалить папку, напр имер: rm -rf fronted\_test

**Ветвление**

Операция ветвления и слияния — сердце и душа git. именно эти возможности делают такой удобной работу с системой.

В основе операций ветвления лежат ветки, которые представляют собой очередность коммитов, т.е. такая «линия» состоящая из «точек» которые и есть наши коммиты. Так образно можно описать ветку. По умолчанию основная ветка – это ветка master. Когда над проектом работает большая команда, то правильно создавать новую ветку, работать над своими изменениями/дополнениями кода, а потом обратно «сливать» созданную новую ветку в мастер (в основную, первоначальную ветку). Такой «слив» можно сделать а) на гите, через „pull request“ и в самом баше т.е. локанльно с помощью команды git merge footer , где footer – название ветки.

**git checkout -b “название”** — (branch) cоздание ветки, в которую и произойдет переключение. Если в текущей ветке были какие-то изменения по сравнению с последним коммитом в ветке(HEAD), то команда откажется производить переключение, дабы не потерять произведенную работу. (*Итак, мы создали новую ветвь ”header” и перешли в неё, зашли в наш файлик (index.html) и прописали какой-то div, закомитили изменения, попытались спушить изменения, но не учли следующий нюанс: когда мы создаем создаем локанально ветвь, нам нужно эту же ветвь еще как бы «отправить» на гитхаб, чтобы как бы синхронизировать обе ветви, установить между ними связь. Изначально между нашей базовой, первоначальной веткой master (локальной) и той, что на гите была установлена связь,, а вот если создаем новую, то нужно эту связь обязательно установить.*

**git branch** — список доступных веток.

**git push origin header/git push origin -u header (при первом пуше)**  — с помощью этой команды мы как бы создаем такую же ветвь (такую же ветвь header) только на гите, тем самым устанавливая между веткой созданной нами локально (header) и той, что мы создали на гите (с помощью команды git push origin header) связь, синхронизацию. *Дальше, введя эту команду мы переходим на гитхаб, обновляемся, видим информацию о новой ветке. Нажимаем по зеленой кнопке (Сompare & pull request), которая означает запрос на сливание нашей ветки (header) в мастер, дальше ознакомившись со списком изменений(кода) можно подтвердить запрос и дальше также можно перевести этот наш запрос на конкретного человека (в команде) с помощью Assignee (если работаешь не сам). также тот кто проверяет может ознакомиться с изменениями/ добавлениями кода ( переходим в files changes), прокомментить что-то если нужно и наконец нажать зеленую кнопку (Merge pull request) тем самым сливаю ветку (в нашем случае header) в нашу первоначальную, основную ветку master. После этого, можно зайти в наш код (находясь в гите) и увидить, что он изменился (добавился/исправился) при чем в ветке мастер.*

**git merge footer(название ве тки) —** команда, которая позволяет слить ветку в master. В данном случает это ветвь footer. (была создана в нашем примере, после чего в index.html был дописан код, закомичен, после осуществился переход с ветки footer в нашу master с помощью команды git checkout master и прописана команда git merge footer (дословно слить футер)).

Также существуют конфликты между ветками. Это происходит тогда, когда 2 разработчика создают 2 ветки работая над одним куском кода. Тогда Гит в файле с кодом добавляет свои символы тем самым разделяя две конфликтующих куска кода, он как бы не знает какой из них более правильный. В этом случае, мы должны выбрать тот кусок кода, который нас больше устраивает и убрать все лишнее ( тот кусок который не устраивает, и добавленный гитом символы), потом просто закомитить изменения.