

**ОСНОВЫ GIT**

**ДЛЯ ТЕСТИРОВЩИКОВ**

Оглавление

[СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ (СКВ) 3](#_Toc94024781)

[ЛОКАЛЬНАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ 4](#_Toc94024782)

[ЦЕНТРАЛИЗОВАННАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ 5](#_Toc94024783)

[РАСПРЕДЕЛЕННАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ 7](#_Toc94024785)

[ВВЕДЕНИЕ В GIT 9](#_Toc94024786)

[СОСТОЯНИЕ ФАЙЛОВ 10](#_Toc94024787)

[Установка Git 12](#_Toc94024788)

[GitHub 13](#_Toc94024789)

[Установка SSH ключей 14](#_Toc94024790)

СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ (СКВ)

Давайте рассмотрим пример. Тестовая документация на проекте храниться в Excel таблице в определенной папке. Получив удаленный доступ к файлу, первый тестировщик начал вносить свои изменения в смоук тесты проекта. Одновременно с ним второй тестировщик также начал редактировать файл со смоук тестами внося изменения в тестовые данные.

Когда первый тестировщик закончил свою работу, он сохранил файл. Таким образом на сервере исходный файл с тестами получил версию 1 первого тестировщика.

Закончив вносить изменения, второй тестировщик также сохранил файл. Так как никакой сиcтемы контроля версий не было, изменения второго тестировщика перезаписали файл, и он получил версию 2 второго тестировщика, тем самым не сохранив изменения первого. Первый тестировщик потратил время зря.

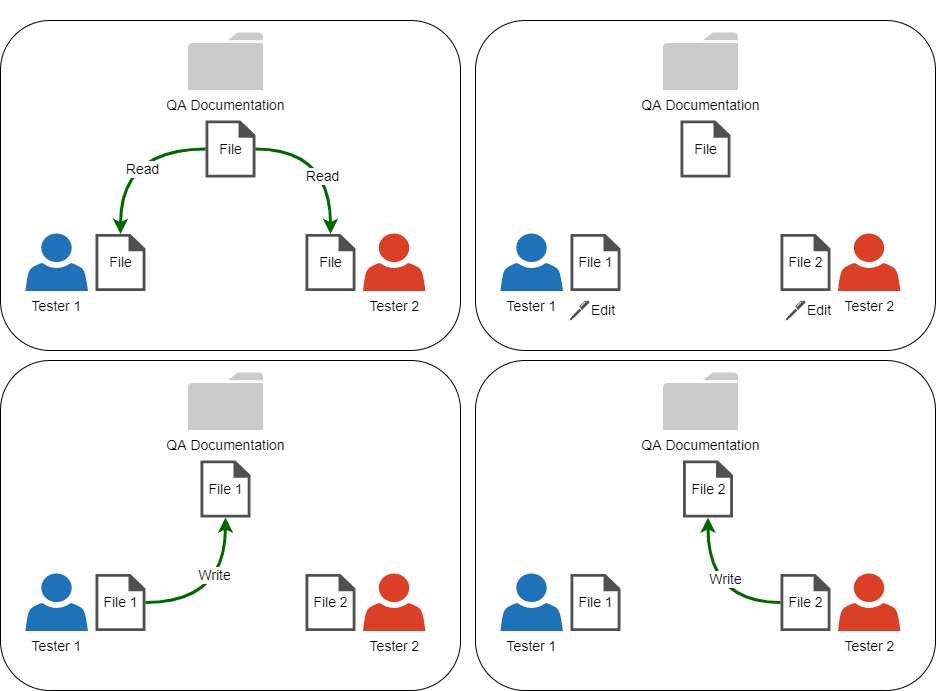
****

Рисунок 1 - Пример одновременной работы над одним файлом

В таком случае нам поможет система контроля версий. Система управления версий позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение.

Другими словами, СКВ позволяет изменять одни и те же файлы нескольким разработчикам одновременно и без создания локальных копий на их компьютерах. При этом все варианты изменений сохраняются отдельно, что позволяет сделать разные варианты одного и того же файла с учетом разных правок от разных людей. Если же несколько изменений затрагивают один и тот же фрагмент документа, то система предложит выбрать нужный вариант.

Некоторые системы, такие как Subversion, отслеживают историю отдельных файлов. Другие, такие как Git и Mercurial, отслеживают историю целых репозиториев.

Управление версиями подобно системе безопасности. Если вы внесли изменения, которые позже вызвали проблемы, можно будет вернуть файл или весь проект к определенной точке вместо того, чтобы начинать все с нуля.

**Система контроля версий** — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Система контроля версий защищает исходный код от катастрофических сбоев, от случайных ухудшений, вызванных человеческим фактором, а также от непредвиденных последствий.

ЛОКАЛЬНАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ

Локальная систем контроля версий представляет из себя простейшую базу данных, которая хранит записи обо всех изменениях в файлах. Не предназначена для коллективного использования.

Одним из примеров таких систем является система контроля версий RCS, разработанная в 1985 году. Она хранит изменения в файлах (патчи), осуществляя контроль версий. Набор этих изменений позволяет восстановить любое состояние файла.

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

— высокая скорость выполнения восстановления (база данных четко структурирована, поэтому сложностей при поиске не возникает, сетевая задержка отсутствует, поскольку данные хранятся непосредственно на рабочем компьютере).

Недостатки:

— возможность потери данных вследствие возникновения физических поломок оборудования;

— отсутствие возможности совместной разработки.

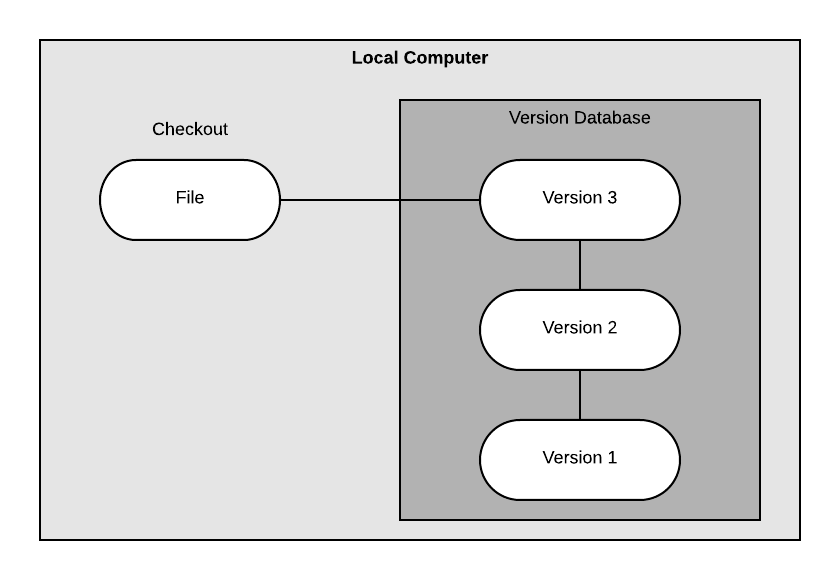


Рисунок 2 - Схема локальной системы контроля версий

ЦЕНТРАЛИЗОВАННАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ

Для организации централизованной системы контроля версий используется сервер, который содержит все версии файлов. Клиенты, обращаясь к этому серверу, получают файлы из этого централизованного хранилища. Применение централизованных систем контроля версий на протяжении многих лет являлась стандартом. К ним относятся CVS, Subversion, Perforce.

Такими системами легко управлять из-за наличия единственного сервера. Но при этом наличие централизованного сервера приводит к возникновению единой точки отказа в виде этого самого сервера. В случае отключения этого сервера разработчики не смогут использовать файлы. Самым худшим сценарием является физическое уничтожение сервера - он приведет к потере всего кода проекта.

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

— возможность ведения командной разработки проекта;

Недостатки:

— отсутствие доступа к данным при сбое работы сервера;

— довольно низкая скорость работы (из-за возникновения сетевых задержек).

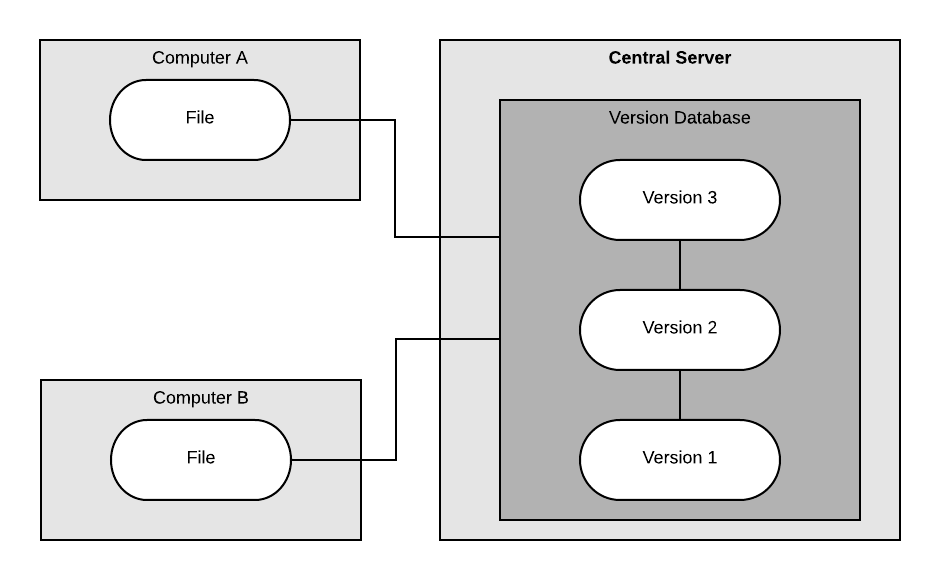


Рисунок 3 - Схема централизованной системы контроля версий

|  |  |
| --- | --- |
| https://i1.wp.com/devpractice.ru/wp-content/uploads/2017/03/git-for-beginners-part1-3.jpg | *Subversion* (SVN) – система контроля версий, созданная на замену CVS. SVN была разработана в 2004 году и до сих пор используется. Несмотря на многие преимущества по сравнению с CVS, у SVN есть недостатки: проблемы с переименованием, невозможность удаления данных из хранилища, проблемы в операции слияния ветвей и т.д. В целом SVN был (и остается) значительном шагом вперед по сравнению с CVS. |

|  |  |
| --- | --- |
| https://i0.wp.com/devpractice.ru/wp-content/uploads/2017/03/git-for-beginners-part1-2.jpg | *CVS*(*Concurrent Versions System*, Система одновременных версий) одна из первых систем получивших широкое распространение среди разработчиков. Возникла в конце 80-х годов прошлого века. В настоящее время не развивается, это в первую очередь связано с рядом ключевых недостатков, таких как невозможность переименования файлов, неэффективное их хранение, практически полное отсутствие контроля целостности. |

|  |  |
| --- | --- |
|  | *Perforce* (P4) — коммерческая система управления версиями. Разработана компанией Perforce Software, основанной в 1995 году. Система имеет клиент-серверную архитектуру. Сервер Perforce может одновременно иметь несколько репозиториев, называемые «депо». Система предоставляет множество других возможностей — извещения (notification), создание ветвей проекта (branching), слияние веток (merging), точки отката. |

РАСПРЕДЕЛЕННАЯ СИСТЕМА КОНТРОЛЯ ВЕРСИЙ

В распределенной системе контроля версий (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий.

**Репозиторий** (repository) — совокупность файлов, состояние которых отслеживается, и история их изменений. Репозиторий— это проект, над которым ведется работа, и все его изменения.

В этом случае, если один из серверов, через который разработчики обменивались данными, выйдет из строя, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных

Все копии являются равноправным и могут синхронизироваться между собой.

К данному виду систем контроля версий относятся Mercurial, Bazaar и Git. Последняя система контроля версий и будет рассмотрена более детально.

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

— возможность ведения командной разработки проекта;

— при сбое работы сервера система сохраняет данные в локальном репозитории, что позволяет эффективно вести процесс разработки, а после восстановления работы сервера, передать все изменения в удаленный репозиторий;

— при физической поломке сервера данные можно легко перенести в новый удалённый репозиторий с любого локального репозитория;

— высокая скорость работы (в ходе работы данные записываются и получаются из локального репозитория, поэтому сетевые задержки отсутствуют).

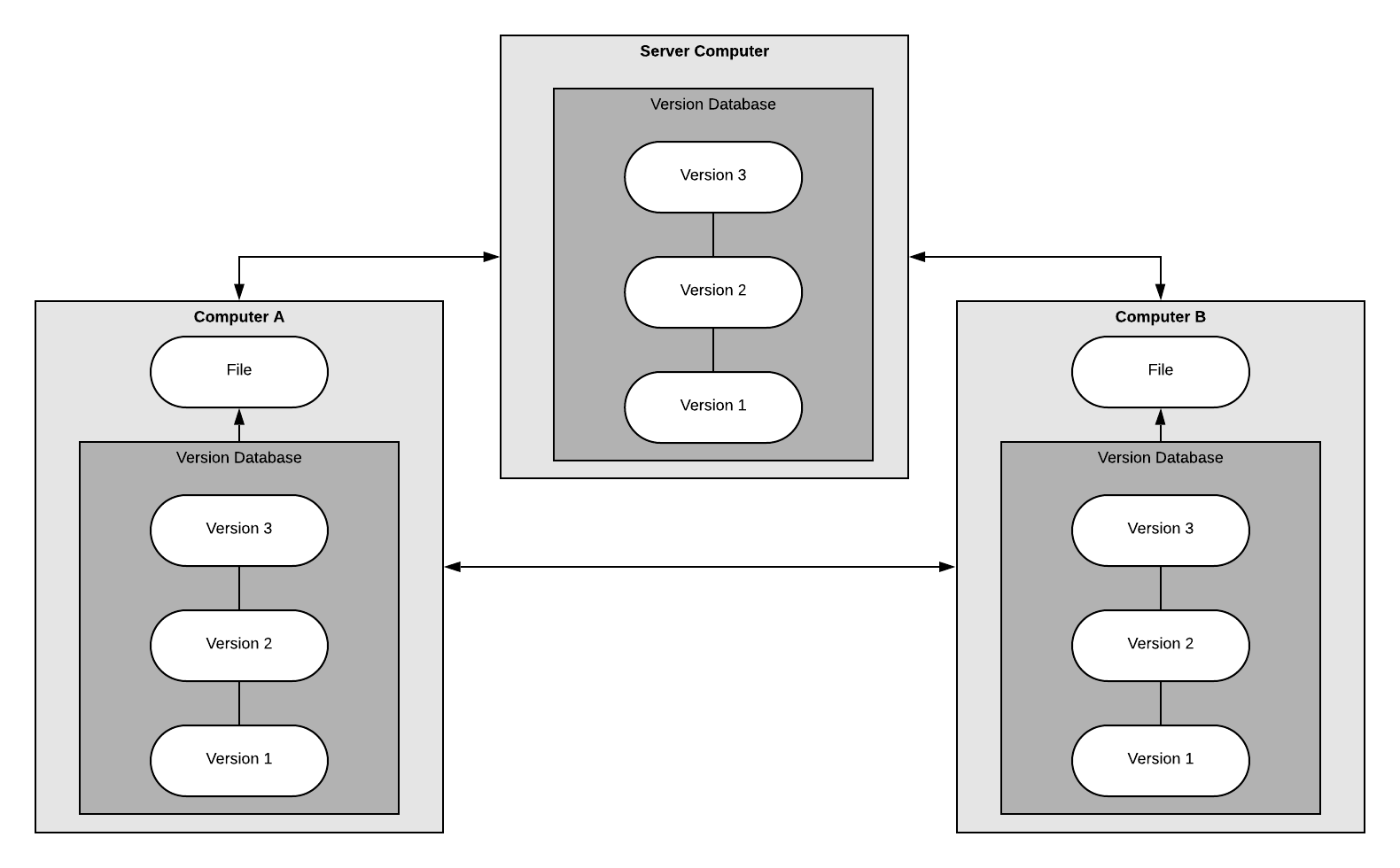


Рисунок 4 - Схема распределенной системы контроля версий

|  |  |
| --- | --- |
| Логотип программы Bazaar | *Bazaar* — распределённая система управления версиями. Команды Bazaar очень похожи на команды, используемые в CVS или SVN. В отличие от чисто распределённых систем контроля версий, которые не используют центральный сервер, Bazaar поддерживает работу как с сервером, так и без него. Возможно даже использовать оба метода одновременно для одного и того же проекта. |

|  |  |
| --- | --- |
|  | *Mercurial* представляет собой распределенную систему контроля версий, которая построена таким образом, что в ней отсутствует понятие центрального репозитория. Mercurial обладает всеми возможностями системы контроля версий, такими как ветвление, слияние, синхронизация с другими репозиториями. Данный проект используют и поддерживают большое количество крупных разработчиков, среди них Mozilla, OpenOffice, OpenJDK и многие другие. |

ВВЕДЕНИЕ В GIT

|  |  |
| --- | --- |
|  | *Git –* распределенная система контроля версий, разработанная Линусом Торвальдсом для работы над ядром операционной системы Linux. По своим базовым возможностям Git схож с Mercurial, но благодаря ряду достоинств (высокая скорость работы, возможность интеграции с другими VCS, удобный интерфейс) и очень активному сообществу, сформировавшемуся вокруг этой системы, Git вышел в лидеры рынка распределенных систем контроля версий. |

Git — система управления версиями с распределенной архитектурой. Основное отличие Git от любой другой системы контроля версий — это подход к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях).

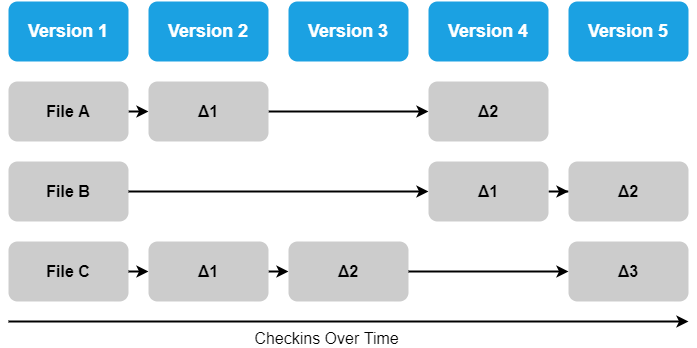


Рисунок 5 - Хранение данных как набора изменений относительно первоначальной версии каждого из файлов

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

**Commit** (коммит) — фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.

Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как поток снимков.

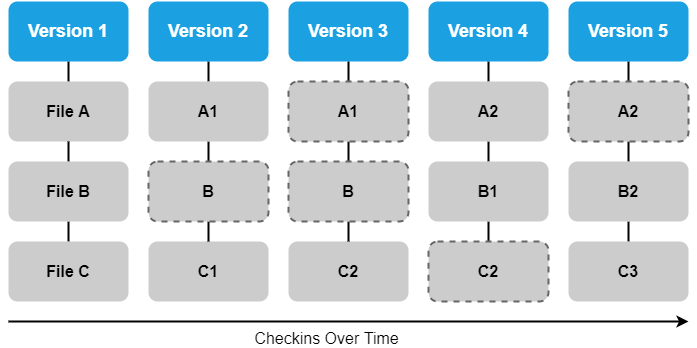


Рисунок 6 - Хранение данных как снимков проекта во времени

Это очень важное отличие между Git и почти любой другой системой контроля версий. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем. Это делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, надстроенными над ней, нежели просто на систему контроля версий.

СОСТОЯНИЕ ФАЙЛОВ

В Git есть три основных состояния, в которых может находится файл:

- изменён (modified)

- индексирован (staged)

- зафиксирован (committed)

К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

Индексированный — это изменённый файл в его текущей версии, отмеченный для включения в следующий коммит.

Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

Также в Git существуют основные секции проекта:

* рабочая директория (working tree)
* область индексации (staging area)
* репозиторий Git (Git directory).

Рабочая директория является снимком одной версии проекта. Эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск, для того чтобы их можно было использовать или редактировать.

Область индексации — это файл, обычно находящийся в каталоге Git, в нём содержится информация о том, что попадёт в следующий коммит. Её техническое название на языке Git — «индекс».

Репозиторий Git — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git и это та часть, которая копируется при клонировании репозитория с другого компьютера.

Процесс работы над файлами выглядит следующим образом:

1. Изменение файлов в рабочей директории – файл в состоянии «*изменен*». (Изменяете файлы вашей рабочей копии).
2. Добавление изменений в индекс (только те, которые войдут в следующий коммит) – файл в состоянии «*индексирован*». (Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки **только** этих изменений в индекс).
3. Коммит (добавление измененных файлов в репозиторий) – файл в состоянии «*зафиксирован*». (Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш репозиторий Git).

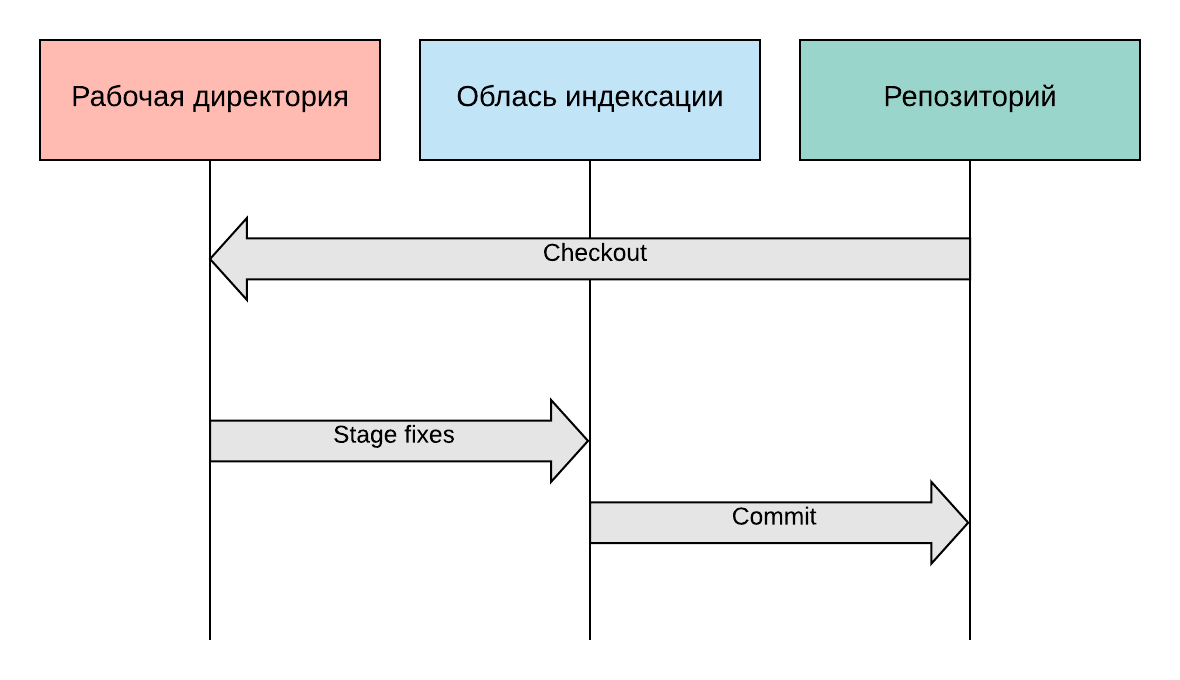


Рисунок 7 - Рабочая директория, область индексации и репозиторий Git

Установка Git

Чтобы установить Git на Windows, нужно скачать установщик с сайта Git: <https://git-scm.com/downloads>. Там же можно найти версии и для других операционных систем.

Выбираем чекбокс Windows Explorer Integration. Остальные настройки оставляем по умолчанию.

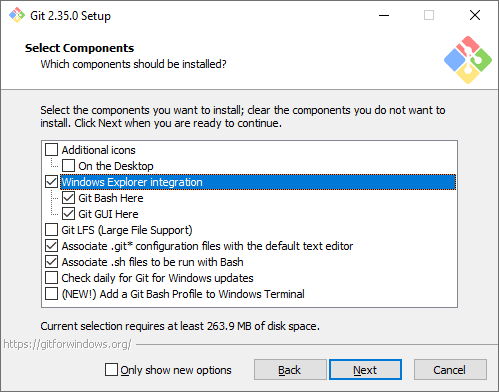


Рисунок 8 - Установка Git

После того, как все действия по установке завершены, убедимся, что Git появился в системе компьютера. Откройте командную строку Windows (для этого нажмите Win+R, в появившемся окне введите cmd) и введите git --version, должна появиться текущая версия программы на вашей машине. Эта проверка подходит для всех операционных систем.

Когда Git установлен в вашей системе, нужно настроить среду для работы с Git под себя. Это нужно сделать только один раз — при обновлении версии Git настройки сохранятся.

Первое, что вам следует сделать после установки Git — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена.

Следует открыть командную строку Git (Git CMD) и ввести следующие данные:

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

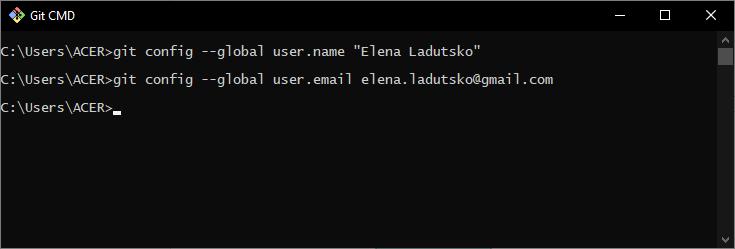


Рисунок 9 - Пример установки имени пользователя и емейла

После установки проверим установленные значения:

git config user.name

git config user.email

Обратите внимание, что в командах, указанных выше, есть опция --global. Это значит, что такие данные будут сохранены для всех ваших действий в Git и вводить их больше не надо. Если вы хотите менять эту информацию для разных проектов, то в директории проекта вводите эти же команды, только без опции --global.

GitHub

Для продолжения работы с Git потребуется регистрация на GitHub.

GitHub — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом. Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL).



**Git —** это инструмент, позволяющий реализовать распределённую систему контроля версий, а **GitHub** — это сервис для проектов, использующих Git.

Git-репозиторий, загруженный на GitHub, доступен с помощью интерфейса командной строки Git и Git-команд. Также есть и другие функции: документация, запросы на принятие изменений (pull requests), история коммитов, интеграция со множеством популярных сервисов, email-уведомления, эмодзи, графики, вложенные списки задач, система упоминаний и т.д.

Кроме GitHub есть другие сервисы, которые используют Git —Bitbucket и GitLab и т.д. Вы можете разместить Git-репозиторий на любом из них.

Регистрируемся на сайте GitHub <https://github.com/>. Вводим обязательные данные: имя пользователя, адрес электронной почты и пароль.

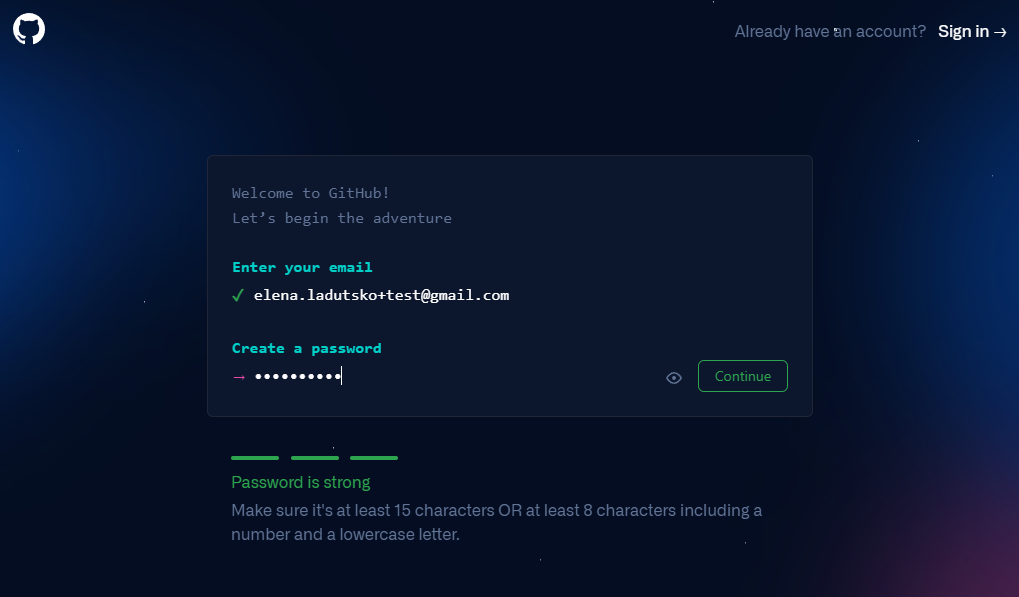


Рисунок 10 - Ввод имени пользователя и пароля для GitHub

После заполнения полей проходим верификацию, проходим небольшой опрос. Выбираем бесплатный план – выбираем “Continue for Free”.

На первой странице GitHub предложит создать новый репозиторий, организацию или узнать больше о GitHub. Пропустим эти пункты и перейдем в профиль для дальнейшей установки.

Установка SSH ключей

Для связи Git и GitHub для дальнейшей работы с проектом необходим SSH ключ. Он нужен для того, чтобы иметь доступ к проектам, хранящимся на сервисе и выполнять команды в консоли без постоянного подтверждения пароля.

Каждый SSH-ключ содержит пару: открытый (публичный) и закрытый (приватный) ключ. Открытый ключ отправляется на сервер, закрытый ключ — секретная часть, он храниться у вас.

Вы отправляете какую-то информацию на сервер, где хранится ваш открытый ключ, сервер понимает, что вы это вы, то есть идентифицирует именно вас, и даёт вам какой-то ответ. И только вы можете расшифровать этот ответ, потому что только у вас есть подходящий закрытый ключ.

Чтобы пройти авторизацию по SSH-ключу, его надо сгенерировать или найти уже ранее созданный ключ на своём компьютере.

Вначале проверим, есть ли сгенерированные ключи на компьютере. Они находятся в папке C:\Users\Username\.ssh

По умолчанию, имена файлов публичных ключей будут такими:

id\_rsa.pub

id\_ecdsa.pub

id\_ed25519.pub

Если папка пуста или не существует – ключей нет. Если же папка уже содержит данные ключи, мы их обновим.

Создаем (или обновляем существующий) SSH ключ:

1. Открываем Git CMD.
2. Вставляем в консоль текст ниже, заменив электронный адрес на ваш email из учётной записи в GitHub, созданной ранее.

ssh-keygen -t rsa -b 4096 -C "[johndoe@example.com](mailto:johndoe@example.com)"

1. Когда вам будет предложено "Enter a file in which to save the key", нажмите «Enter» для создания файла в каталоге, предложенном по умолчанию.

Enter a file in which to save the key (/Users/Username/.ssh/id\_rsa): [Нажмите Enter]

1. В командном процессоре введите кодовую фразу. Если вы не хотите вводить и запоминать кодовую фразу, просто дважды нажмите «Enter»: пустое значение также допустимо.

Enter passphrase (empty for no passphrase): [Введите кодовую фразу]

Enter same passphrase again: [Повторно введите кодовую фразу]

1. Снова проверьте наличие папки C:\Users\Username\.ssh и ключей в ней.

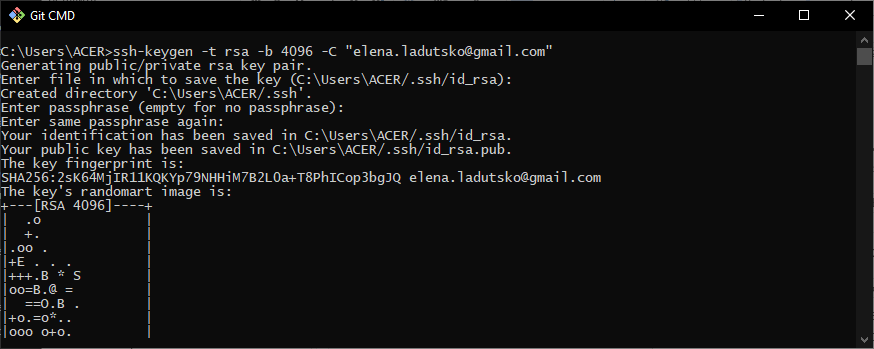


Рисунок 11 - Генерация SSH ключа

Если папки не было изначально, она должна отобразиться со всеми новыми ключами в ней.

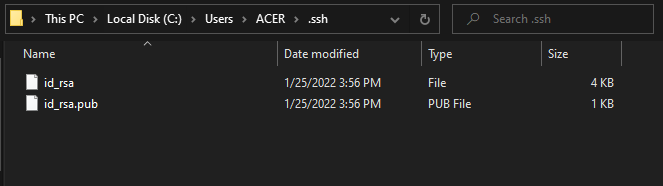


Рисунок 12 - Папка со сгенерированными ключами

Чтобы настроить учетную запись GitHub для использования нового (или существующего) ключа SSH, вам также необходимо добавить его в свою учетную запись GitHub.

Открываем файл id\_rsa.pub в текстовом редакторе и копируем его содержимое. Далее переходим на страницу редактирования ключей в GitHub. В правом верхнем углу любой страницы GitHub кликните на свою фотографию профиля, затем нажмите выберите Settings (<https://github.com/settings/keys>).

На вкладке “SSH and GPG keys” выбираем “New SSH key”.

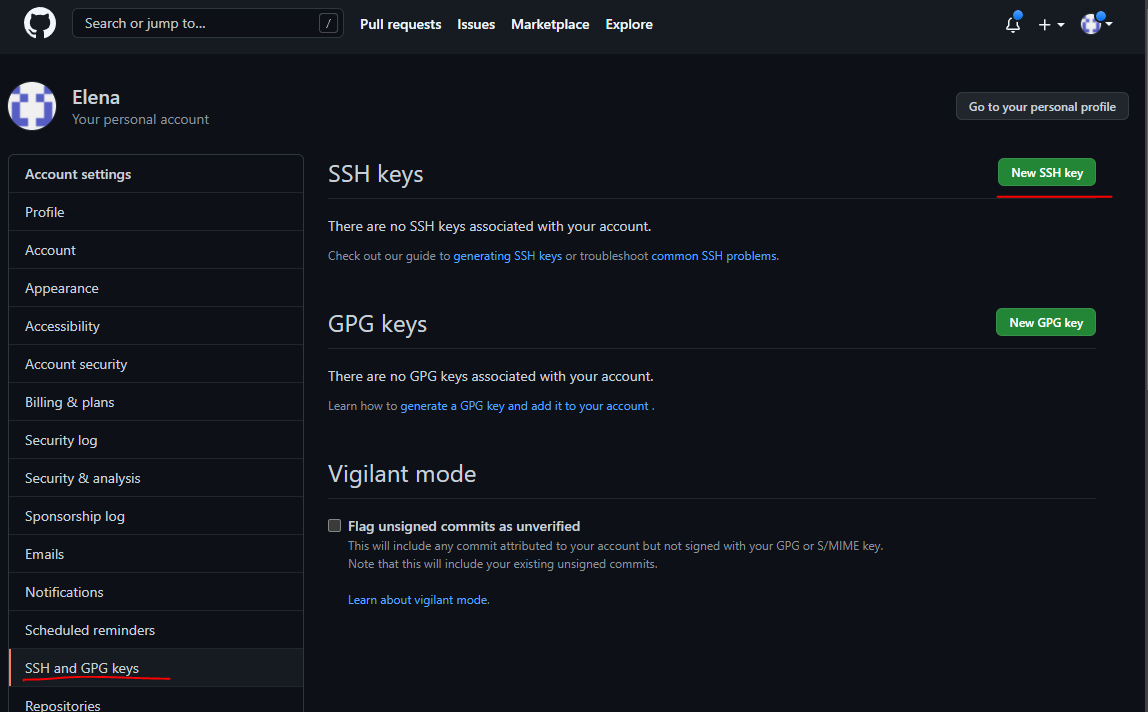


Рисунок 13 - SSH and GPG keys

Вводим имя ключа (можно придумать абсолютно любое) в поле Title (название), а в Key (ключ) вставляем сам ключ из буфера обмена. Нажимаем “Add SSH key” (добавить SSH-ключ).

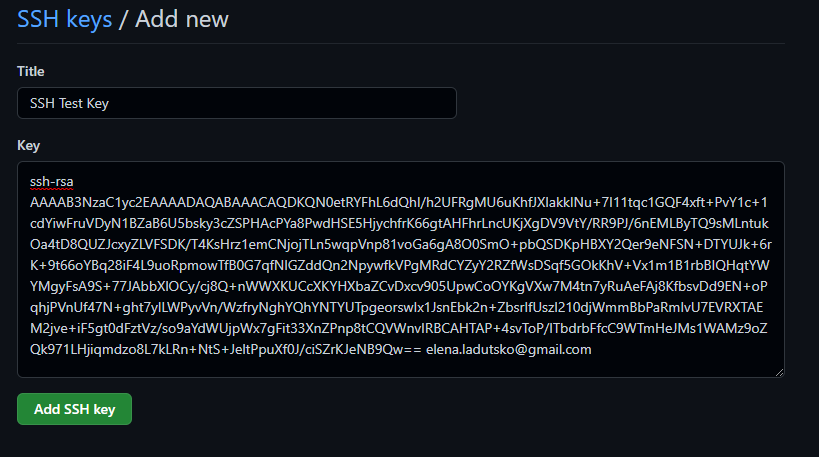


Рисунок 14 - Добавление SSH ключа

После установки ключа проверяем соединение с GitHub.

1. Открываем Git CMD.
2. Вводим команду:

$ ssh -T git@github.com

1. После сообщения

The authenticity of host 'github.com (IP ADDRESS)' can't be established.

RSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.

Are you sure you want to continue connecting (yes/no)?

1. Вводим yes.
2. Получаем сообщение, что соединение успешно установлено.
3. В сообщении проверяем, что оно содержит имя пользователя, которое мы устанавливали выше.

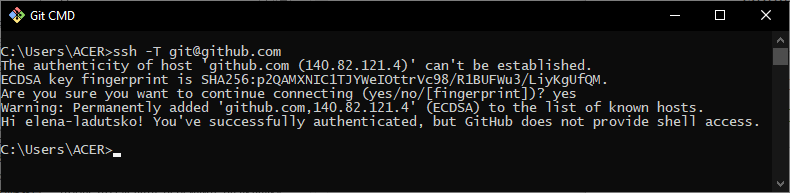


Рисунок 15 - Установка соединения с GitHub

Создание Git репозитория

Создать Git репозиторий можно двумя способами:

1. Превратить в Git репозиторий любой локальный каталог, который в настоящее время не находится под версионным контролем.
2. Клонировать существующий репозиторий Git из любого места.

В обоих случаях вы получите готовый к работе Git репозиторий на вашем компьютере.

Для превращения любого каталога в репозиторий Git нужно для начала создать этот каталог. Для примера используем папку Git, в которой храниться документ Word и один HTML файл.

Допустим, что в HTML файле храниться код проекта, а в Word документе – тестовая документация.

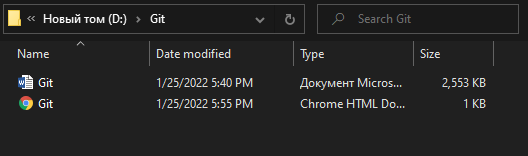


Рисунок 16 - Каталог с файлами

Для создание репозитория в существующем каталоге для начала нужно перейти в него. В Git CMD выполняем следующую команду:

cd C:\Git – для перехода между папками в пределах одного диска

D: – для перехода на другой диск

Затем, находясь в нужном каталоге, выполняем команду gin init.

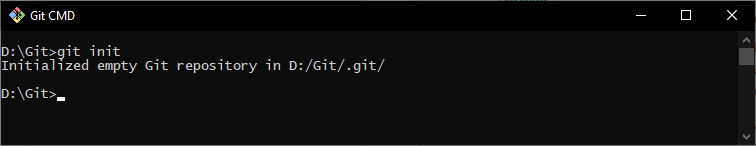


Рисунок 17 - Создание репозитория

Эта команда создаёт в текущем каталоге новый подкаталог с именем. git, содержащий все необходимые файлы репозитория. На этом этапе проект ещё не находится под версионным контролем.

Выполнив команду git status мы можем увидеть, какие файлы нашего репозитория в каком состоянии находятся. Если выполнить эту команду сейчас, результат будет примерно таким:

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

Git.docx

Git.html

nothing added to commit but untracked files present (use "git add" to track)

Мы видим, что наши файлы все еще находятся в состоянии untracked.

Если мы хотим добавить под версионный контроль существующие файлы (документ и html-файл), следует добавить их в индекс и осуществить первый коммит изменений.

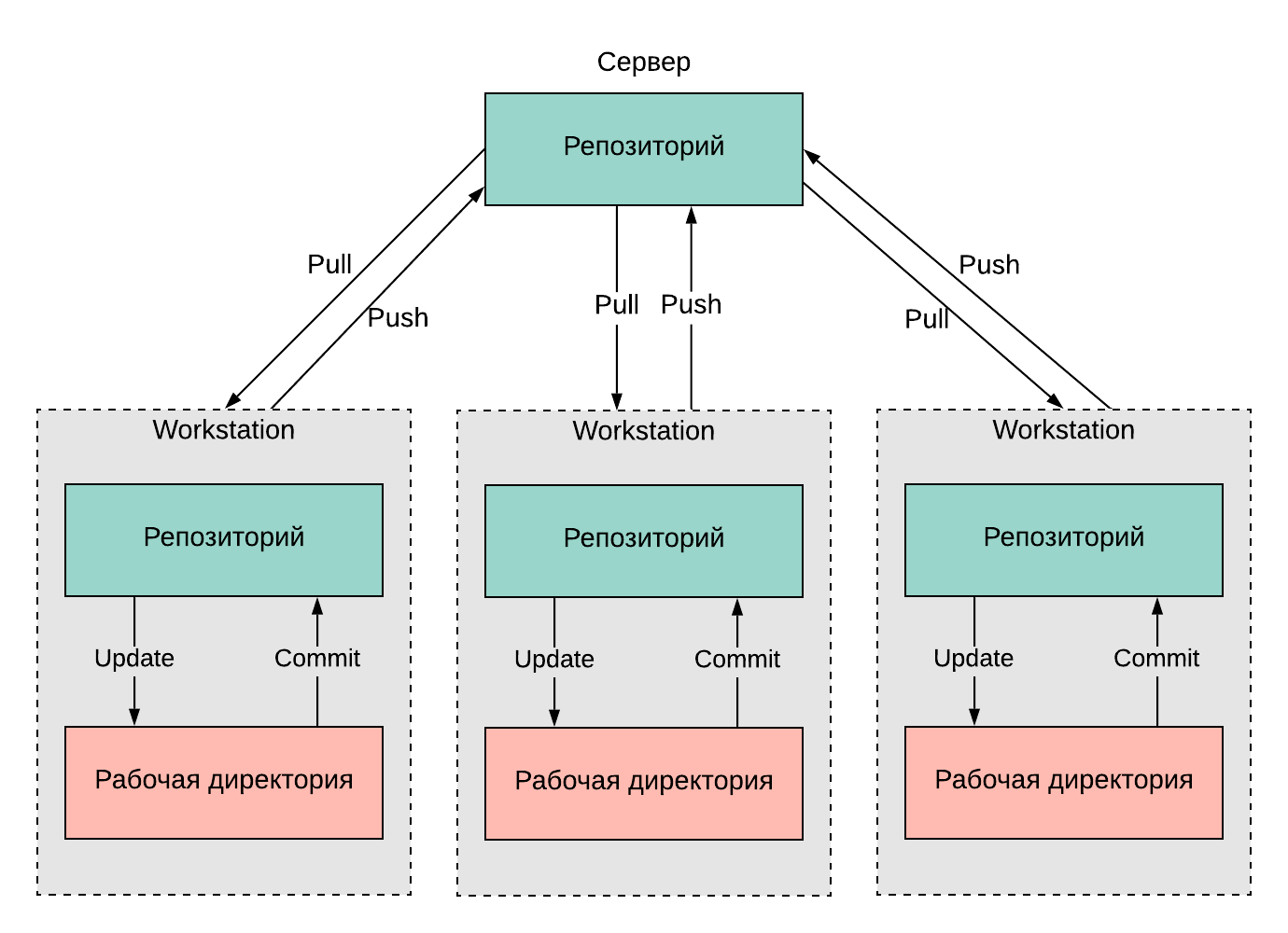
Для этого выполняем команду git add несколько раз, указав индексируемые файлы, а затем выполняем git commit.



Рисунок 18 - Добавление файлов и коммит

**Pull** (пул) — получение последних изменений с удалённого сервера репозитория.

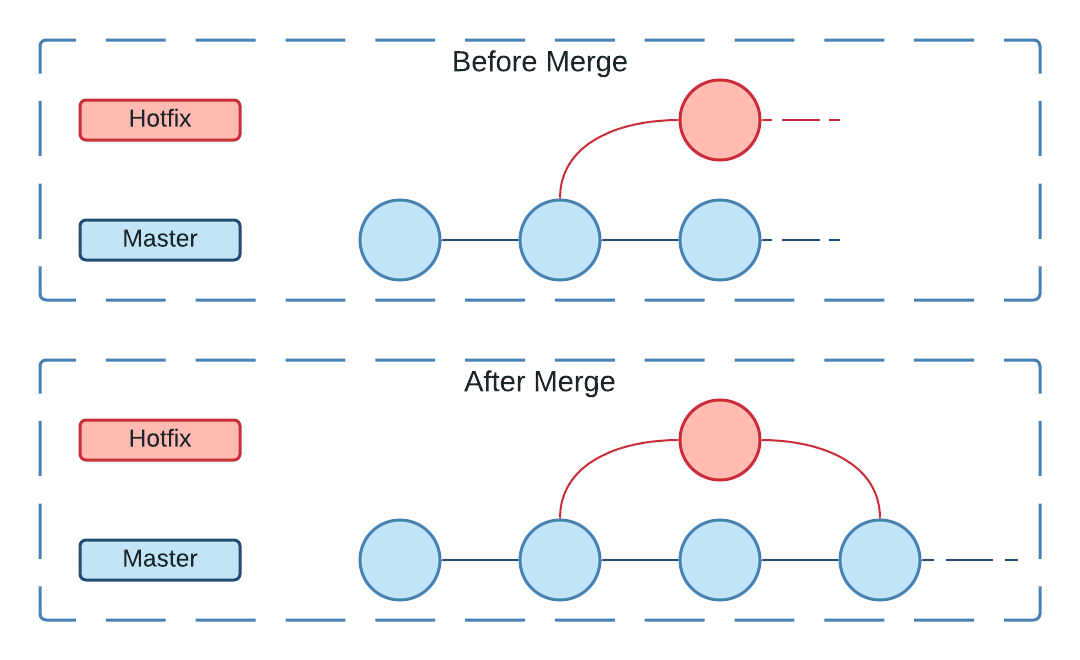
**Push** (пуш) — отправка всех неотправленных коммитов на удалённый сервер репозитория.



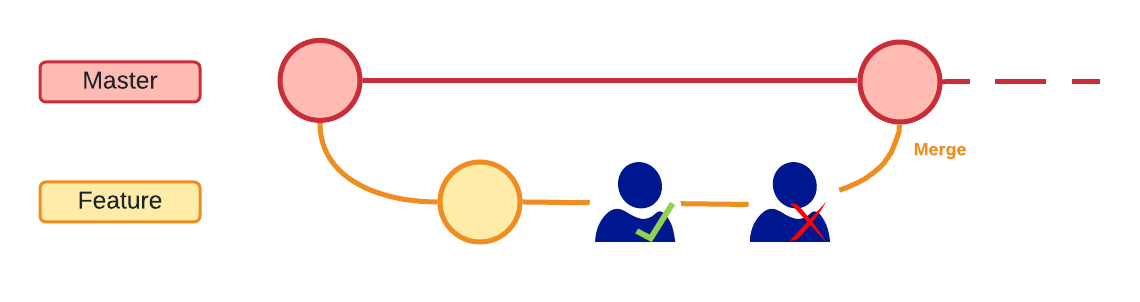
**Branch** (ветка) — копия проекта, его параллельная версия. Ветви имеют свою собственную историю и изолированы друг от друга. Разработчики вносят изменения с свои ветки, не влияя на основную. Когда все изменения внесены, ветку можно объединить с главной.

**Master** (мастер) — главная или основная ветка репозитория. Обычно это полностью готовое к релизу состояние со всеми пройденными тестами.

**Merge** (мерж) – слияние двух или более веток.



* **Pull Request** (пулреквест) — запрос на слияние локального репозитория с основным репозиторием.
* **Pull Request** обеспечивает удобный веб-интерфейс для обсуждения предлагаемых изменений до их включения в официальный репозиторий проекта.



* **Deploy** (деплой) — процесс «разворачивания» приложения в рабочем окружении.
* **Environment** (энвайромент) – рабочее окружение, место, где приложение запускается и доступно для запросов.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **QA** | **UAT** | **PROD** |
| **Branch** | develop | develop | master |
| **Deploy** | automatic | manual | manual |
| **Data** | testing data | production sanitized | production |
| **Domain** | qa.project.com | uat.project.com | project.com |

**Словарь:**

* **Git** — система управления версиями файлов с распределенной архитектурой.
* **GitHub** — веб-сервис для размещения репозиториев и совместной разработки проектов.
* **Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге. Именно в нём происходит разработка и фиксация изменений, которые отправляются на удалённый репозиторий.
* **Удалённый репозиторий** — репозиторий, находящийся на удалённом сервере. Это общий репозиторий, в который приходят все изменения и из которого забираются все обновления.
* **Branch** (ветка) — копия проекта, его параллельная версия.
* **Master** (мастер) — главная или основная ветка репозитория.
* **Merge** – слияние двух или более веток.
* **Commit** (коммит) — фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.
* **Pull** (пул) — получение последних изменений с удалённого сервера репозитория.
* **Push** (пуш) — отправка всех неотправленных коммитов на удалённый сервер репозитория.
* **Pull Request** (пулреквест) — запрос на слияние локального репозитория с основным репозиторием.
* **Deploy** (деплой) — процесс «разворачивания» приложения в рабочем окружении.
* **Environment** (энвайромент) – рабочее окружение, место, где приложение запускается и доступно для запросов.