**Версионирование**

Чтобы лучше понять проблему версионирования, рассмотрим пример дизайнера, который закончил работать над проектом и отправил финальную версию заказчику. У дизайнера есть папка, в которой хранится финальная версия проекта:

source/

barbershop\_index\_final.psd

Всё хорошо, дизайнер закончил работу, но заказчик прислал в ответ правки. Чтобы была возможность вернуться к старой версии проекта, дизайнер создал новый файл barbershop\_index\_final\_2.psd, внёс изменения и отправил заказчику:

source/

barbershop\_index\_final.psd

barbershop\_index\_final\_2.psd

Этим всё не ограничилось, в итоге структура проекта разрослась и стала выглядеть так:

source/

barbershop\_index\_final.psd

barbershop\_index\_final\_2.psd

…

barbershop\_index\_final\_19.psd

…

barbershop\_index\_latest\_final.psd

barbershop\_index\_latest\_final\_Final.psd

Вероятно, многие уже сталкивались с подобным, например, при написании курсовых работ во время учёбы. В профессиональной разработке использование новых файлов для версионирования является плохой практикой. Обычно у разработчиков в папке проекта хранится множество файлов. Также над одним проектом может работать несколько человек. Если каждый разработчик для версионирования будет создавать новый файл, немного изменяя название предыдущей версии, то в скором времени в проекте начнётся хаос и никто не будет понимать какие файлы нужно открывать.

**Git**

Для решения проблемы с сохранением новой версии файлов удобно использовать системы контроля версий. Одна из самых популярных — Git. Работу Git можно сравнить с процессом сохранения и загрузки в компьютерных играх:

* если впереди ждёт тяжёлое сражение, то перед этим лучше заранее сохраниться;
* чтобы это сделать, нужно выполнить специальную команду;
* после чего сохранение попадает в специальную папку и содержит состояние игры;
* теперь при необходимости всегда есть возможность вернуться к предыдущей версии игры.

Папка, содержащая данные игры, могла бы выглядеть так:

SomeGame/

| - saves

| | - save001.sav

| | - save002.sav

| | …

| | папка с сохранениями

|

| - game.exe

| ...файлы игры

Файлы, необходимые для работы приложения, хранятся в рабочей области. В папке saves хранится история всех сохранений игры. Git сохраняет код вашего проекта по такому же принципу: сохранения попадают в специальную скрытую папку, а рабочей областью является содержимое корневой папки.

**Основные понятия**

[Список терминов](https://htmlacademy.ru/blog/81-git-and-github-glossary), которые будут вам полезны.

Репозиторий

Проект, в котором была инициализирована система Git, называется репозиторием. При инициализации в проект добавляется скрытая папка .git. Репозиторий хранит все рабочие файлы и историю их изменений.

Рабочая область и хранилище

barbershop/

| - .git

| | - bea0f8e

| | - d516600

| | Хранилище

|

| - css

| - index.html

| Рабочая область

Корневая папка проекта — это рабочая область. В ней находятся все файлы и папки, необходимые для его работы.

Хранилище — это содержимое скрытой папки .git. В этой папке хранятся все версии рабочей области и служебная информация. Этим версиям система автоматически даёт название, состоящее из букв и цифр. В примере выше — это bea0f8e и d516600. Не стоит проводить манипуляции с папкой .gitвручную. Вся работа с системой производится командами через специальные приложения или консоль.

Коммит

Точно так же, как и в игре, в системе контроля версий Git можно сохранить текущее состояние проекта. Для этого есть специальная команда — commit. Она делает так, что новая версия проекта сохраняется и добавляется в хранилище. В файле с сохранением отображаются: все изменения, которые происходили в рабочей области, автор изменений и краткий комментарий, описывающий суть изменений. Каждый коммит хранит полное состояние рабочей области, её папок и файлов проекта.

В итоге проект работает так:

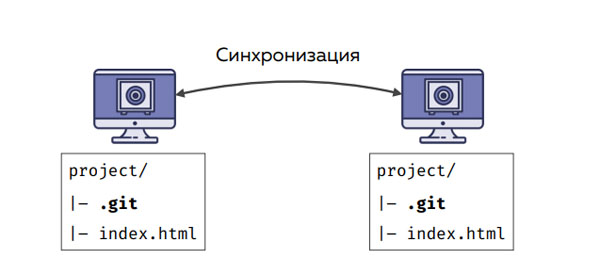
1. Репозиторий хранит все версии проекта. В случае передачи этого проекта другому человеку, он увидит всё, что с ним происходило до этого.
2. Ничего не теряется и не удаляется бесследно. При удалении файла в новой версии добавляется запись о том, что файл был удалён.
3. Всегда можно вернуться к любой из версий проекта, загрузив её из хранилища в рабочую область.

**Система контроля версий Git**

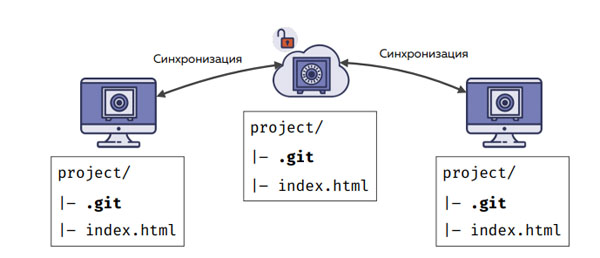
Git — это распределённая и децентрализованная система управления версиями файлов. Децентрализованная система означает, что у каждого разработчика есть личный репозиторий проекта с полным набором всех версий. А все необходимые для работы файлы находятся на компьютере. При этом постоянное подключение к сети не требуется, поэтому система работает быстро. При командной разработке нужна синхронизация репозиториев, так как проект — один и его состояние должно быть у всех одинаковым.

**Работа в команде**

Как синхронизовать данные репозиториев между разработчиками? Изначально Git репозитории сами могут синхронизироваться от пользователя к пользователю. Дополнительные программы для этого не нужны. Есть специальные команды в консоли, позволяющие передавать данные из одного репозитория в другой.

Репозитории можно синхронизировать между пользователями.

Этот способ сложный и редко используется. Чаще всего разработчики синхронизируют локальный репозиторий с удалённым. Удалённый репозиторий — это тот же репозиторий, только его данные находятся в облаке.

Синхронизация через удалённый репозиторий.

**Этапы синхронизации**

Как сделать так, чтобы разработчик смог передать актуальную версию проекта коллеге?

Для взаимодействия с системой Git в консоль вводятся специальные команды. Не пугайтесь, работу с консолью можно будет заменить на работу с одной из программ, о которых расскажем ниже. Но чтобы лучше понимать суть, придётся запомнить несколько команд. Все они начинаются с ключевого слова git. Для синхронизации есть две основных команды: pull (англ. «тянуть») и push (англ. «толкать»).

Pull

Если работа над проектом ведётся в команде, то перед тем как начать писать код нужно получить последнюю версию проекта. Для этого нужно выполнить команду pull. Так мы забираем все изменения, которые были совершены со времени последней синхронизации с удалённым репозиторием. Теперь они у нас в репозитории на локальном компьютере.

Push

Чтобы отправить коллегам последнюю версию проекта выполняем команду push. Если в удалённом репозитории с момента последней синхронизации не было никаких изменений, то все сохранённые изменения успешно загрузятся в облако и коллеги получат последнюю версию проекта, выполнив команду pull. Если же были изменения, то Git попросит вас перед отправкой подтянуть последние версии, сделав pull.

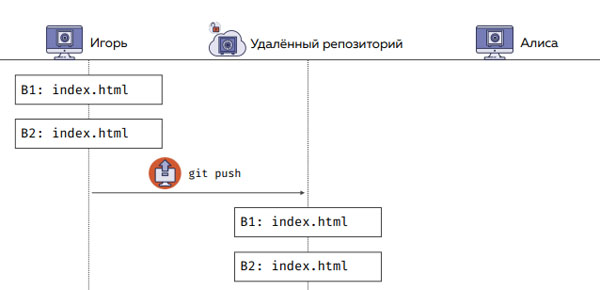
Синхронизация (push и pull) между локальными и удалённым репозиториями.

**Типовой рабочий процесс с использованием Git**

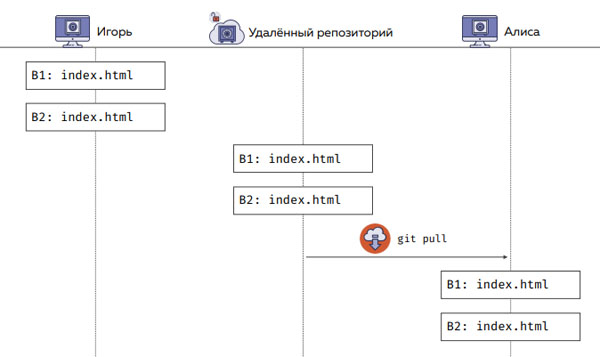
Разберём типовой процесс разработки сайта в команде. Представим, что Игорь и Алиса — разработчики на одном проекте. Игорь начал верстать проект и сделал первые коммиты, в которых зафиксировал изменения в файле index.html. Для схематичности названия коммитов будут простые: B1 и B2.

Коммиты B1 и B2.

После того как Игорь сделал два коммита, он захотел отправить свои изменения в удалённый репозиторий. Чтобы их передать, Игорь выполнил команду git push. После чего в облаке появилось две версии проекта. То есть Игорь отправил не только финальную версию проекта, но и все сохранённые изменения.

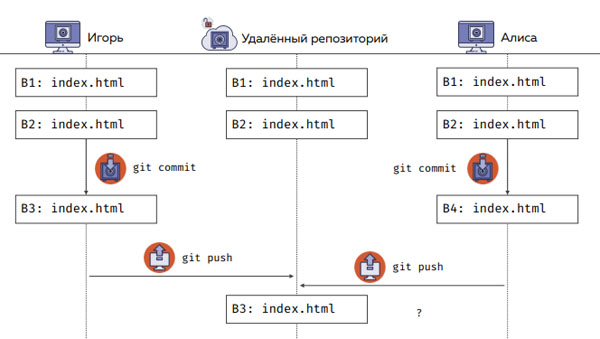
Игорь запушил свои коммиты.

После пуша данные синхронизировались с удалённым репозиторием. Но как Алисе теперь получить изменения? Для этого она выполняет команду git pull и получает все изменения из облака к себе на компьютер. Таким образом, состояние проекта у Игоря и Алисы синхронизировались и они могут дальше продолжить работать над ним.

Данные у обоих разработчиков синхронизировались.

Параллельные изменения

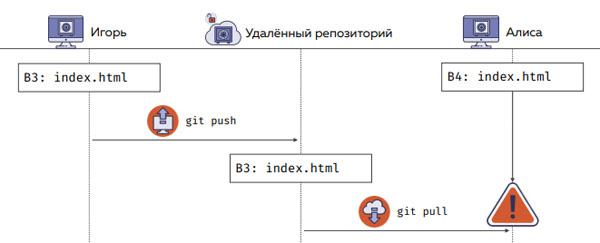
Что произойдёт, если разработчики изменят одинаковый файл и сделают push? Предположим, что Игорь и Алиса изменили файл index.html, сделали коммит с изменениями и запушили его. Игорь оказался быстрее Алисы и сделал push первым.

Два пуша в одно время?

В этом случае Git сообщит Алисе, что нельзя пушить свои изменения, потому что она не делала pull. Дело в том, что после того как Игорь синхронизировался с удалённым репозиторием, версия проекта Алисы стала отличаться от той, что находится на удалённом репозитории, и Git это видит. Система сообщает, что перед тем, как выполнить команду push, нужно выполнить pull, чтобы забрать изменения. Алиса делает pull и ей вновь приходит уведомление от Git. В этот раз он сообщает Алисе о том, что произошёл конфликт.

Конфликт

Дело в том, что Игорь и Алиса изменили одинаковый файл и теперь Алисе предстоит решить конфликт.

Конфликт

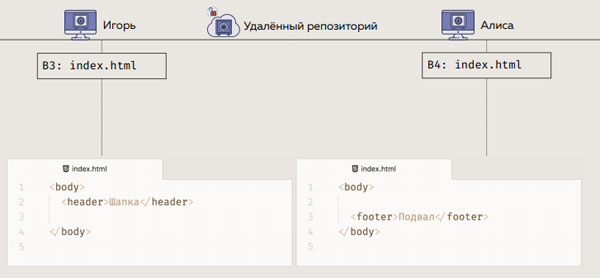
Существуют два вида конфликтов:

1. Автоматически разрешаемый конфликт.
2. Конфликт, который нужно разрешить вручную.

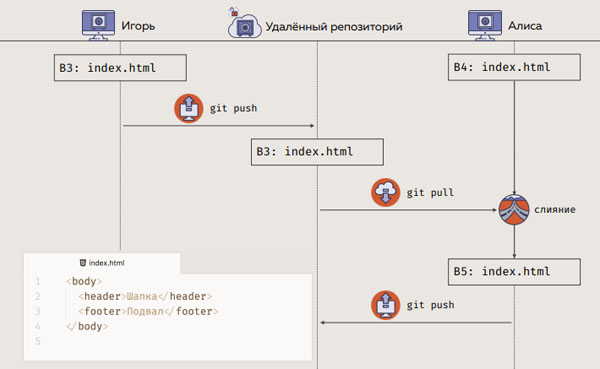
Ниже рассмотрим оба варианта.

Слияние

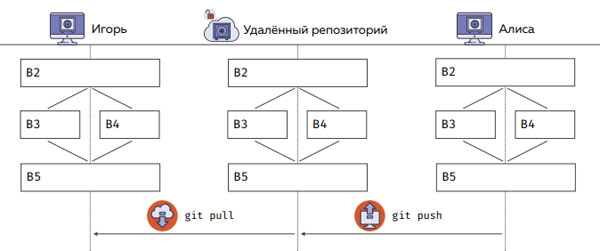
Допустим, что на третьей строке Игорь добавил в проект шапку, а на четвёртой Алиса добавила футер.

Игорь сделал шапку и отправил коммит, а Алиса добавила подвал.

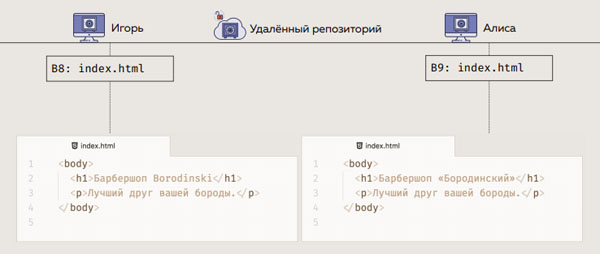
Git видит, что произведённые изменения не затрагивают друг друга. Он сам объединит две версии проектов в одну, совершив слияние. После этого Алиса спокойно синхронизируется с удалённым репозиторием, отправив новую версию проекта.

Изменения в проекте не пересекались и Git выполняет слияние сам.

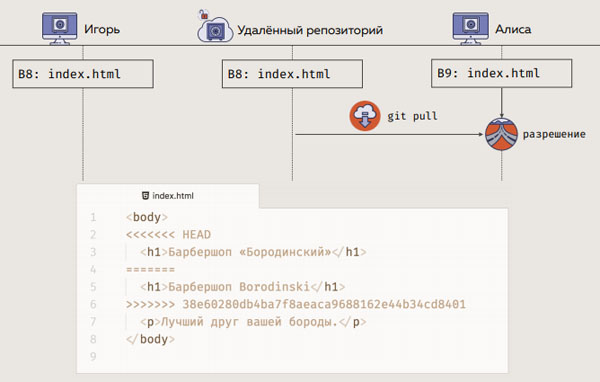
Во время слияния Git не знает в каком порядке расположить коммит В3 Игоря и коммит В4 Алисы, из-за которых случился конфликт. Поэтому Git разрешает существовать нескольким версиям проекта одновременно. Как раз для этого и нужен следующий коммит В5, в котором происходит слияние предыдущих параллельных версий. После того как Алиса запушит изменения, она отправляет все версии проектов на удалённый репозиторий. В следующий раз, когда Игорь сделает pull, он получит полную историю со слиянием конфликта.

Слияние.

Допустим, что Игорь и Алиса продолжили работать над проектом, но в этот раз изменили одинаковую строку в файле index.html. Вновь Игорь оказался быстрее и первым синхронизировал свои изменения с удалённым репозиторием. Алиса сделала pull и получила сообщение о конфликте.

Алиса и Игорь изменили один и тот же блок.

В таком случае Git не знает чья версия проекта правильная и поступает очень просто. Он изменяет файл index.html, добавляя в него изменения и Игоря и Алисы. После этого предупреждает Алису о конфликте и просит выбрать правильный вариант.

Версии файла.

Версии проектов разделяются строками второй, четвёртой и шестой. Их нужно удалить и оставить правильный вариант заголовка. После того как Алиса это сделает, она сможет закоммитить изменения и запушить их на удалённый репозиторий. Игорь же при следующей синхронизации с облаком получит тот вариант заголовка, который выбрала Алиса.

**Окружение Git**

Git — удобная система. Плюсом является то, что вокруг него создано множество сервисов, которые позволяют сделать работу с ним удобнее. Расскажем о тех, что будут вам полезны в начале работы.

**GitHub**

[GitHub](https://github.com/) — это сайт, сервис и то самое облако, в котором можно хранить удалённые репозитории и через которое коллеги могут синхронизировать свои версии проектов. Как зарегистрироваться мы рассказали в [этой статье](https://htmlacademy.ru/blog/85-register-on-github-work-with-github-desktop).

**GUI**

Облегчить работу с Git и GitHub могут [специальные программы](https://git-scm.com/downloads/guis/). Такие программы в удобной форме показывают изменения в коде, список коммитов и обладают другими удобными возможностями. Обычно в подобных программах есть возможность выполнять стандартные Git команды: pull, push, commit прочие, — просто нажав на кнопку.