Система управления версиями (от англ. Version Control System, VCS) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями 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, и многое другое.

Такая система фиксирует изменения, что позволяет вам в случае чего откатиться к любой предыдущей версии файла. Кроме отката изменений, система контроля версий позволяет сравнивать версии одного и того же файла, чтобы найти в нем изменения, видеть, кто эти изменения внес, когда это было сделано и что могло вызвать проблему.

Первые две вещи, которые вам нужно сделать, это установить git и создать бесплатную учетную запись GitHub.

**Инструкция по установке git для windows**

Официальная сборка доступна для скачивания на сайте Git. Просто перейдите по ссылке https://git-scm.com/download/win , и загрузка начнется автоматически. Далее запустите установочный файл, везде выбираем далее, дополнительных опций, кроме тех, что установлены по умолчанию, выбирать не требуется.

**Инструкция по установке git для Mac**

Установщик macOS Git поддерживается и доступен для загрузки на веб-сайте Git по адресу https://git-scm.com/download/mac . Далее запустите установочный файл, везде выбираем далее, дополнительных опций, кроме тех, что установлены по умолчанию, выбирать не требуется.

Затем откроем консоль - сочетание клавиш win + r и в окне набираем *cmd.* Для проверки корректности установки git требуется прописать в консоли

*git --version*

Итак, мы установили git, теперь нужно добавить немного настроек. Есть довольно много опций, с которыми можно играть, но мы настроим самые важные: наше имя пользователя и адрес электронной почты. Откройте терминал и запустите команды:

*git config --global user.name "My Name"*

*git config --global user.email myEmail@example.com*

Теперь каждое наше действие будет отмечено именем и почтой. Таким образом, пользователи всегда будут в курсе, кто отвечает за какие изменения — это вносит порядок.

Git хранит весь пакет конфигураций в файле .gitconfig, находящемся в вашем локальном каталоге. Чтобы сделать эти настройки глобальными, то есть применимыми ко всем проектам, необходимо добавить флаг –global. Если вы этого не сделаете, они будут распространяться только на текущий репозиторий.

Для того, чтобы посмотреть все настройки системы, используйте команду:

*git config --list*

Далее создадим учетную запись GitHub на сайте https://github.com/

**Основы git**

Затем нужно перейти в папку проекта на локальном компьютере, нажать правой кнопкой мыши и выбрать опцию “открыть в терминале”. Чтобы инициализировать репозиторий git в корне папки, выполните команду

*git init*

Добавьте новый файл в проект, используя любой текстовый редактор, который вам нравится. Как только вы добавили или изменили файлы в папке, содержащей репозиторий git, git заметит, что файл существует внутри репозитория. Но git не будет отслеживать файл, если вы не укажете это явно. После создания нового файла вы можете использовать команду

*git status*, чтобы увидеть, какие файлы известны git.

Одна из самых запутанных частей при первом изучении git — это концепция промежуточной среды и ее связь с фиксацией.

Коммит — это запись о том, какие изменения вы внесли с момента последнего коммита. По сути, вы вносите изменения в свой репозиторий (например, добавляете файл или изменяете его), а затем говорите git поместить эти изменения в фиксацию.

Коммиты составляют суть вашего проекта и позволяют вам переходить к состоянию проекта при любом другом коммите.

Итак, как вы сообщаете git, какие файлы помещать в коммит? Здесь в дело вступает промежуточная среда или индекс. Как видно из предыдущего шага, когда вы вносите изменения в репозиторий, git замечает, что файл изменился, но ничего с ним не делает (например, добавляет его в коммит).

Чтобы добавить файл в коммит, сначала нужно добавить его в промежуточную среду. Для этого вы можете использовать команду

*git add имя_файла*.

Если нам нужно добавить все, что находится в директории, мы можем использовать

*git add -A*

Если вы повторно запустите команду *git status*, вы увидите, что git добавил файл в тестовую среду.

Запустите команду

*git commit -m “сообщение коммита”*

Сообщение в конце коммита должно быть связано с тем, что содержит коммит — может быть, это новая функция, может быть, это исправление ошибки, может быть, это просто исправление опечатки. Не размещайте сообщение типа «asdfadsf» или «foobar». Это огорчает других людей, которые видят вашу фиксацию. Очень печально. Коммиты вечно

хранятся в репозитории, поэтому, если вы оставите четкое объяснение ваших изменений, это может быть чрезвычайно полезно для будущих программистов, которые попытаются понять, почему некоторые изменения были внесены годы спустя.

**Ветвление git**

Теперь, когда вы сделали новый коммит, давайте попробуем что-нибудь более сложное.

Допустим, вы хотите создать новую функцию, но беспокоитесь о внесении изменений в основной проект при разработке этой функции. Вот тут-то и появляются ветки git.

Ветки позволяют вам перемещаться вперед и назад между «состояниями» проекта. Официальная документация git описывает ветки следующим образом: «Ветка в Git — это просто легковесный подвижный указатель на один из этих коммитов». Например, если вы хотите добавить новую страницу на свой веб-сайт, вы можете создать новую ветку только для этой страницы, не затрагивая основную часть проекта. Когда вы закончите со страницей, вы можете объединить свои изменения из своей ветки в основную ветку. Когда вы создаете новую ветку, Git отслеживает, от какой фиксации ваша ветка «ответвилась», поэтому он знает историю всех файлов.

Допустим, вы находитесь в основной ветке и хотите создать новую ветку для разработки своей веб-страницы. Вот что вы будете делать: Запустите

*git checkout -b <имя_новой_ветки>* .

В данном проекте мы назовем ветку new_feature и команда будет выглядеть таким образом:

*git checkout -b new_feature*

Эта команда автоматически создаст новую ветку, а затем «проверит вас» на ней, что означает, что git переместит вас в эту ветку за пределы основной ветки.

После выполнения вышеуказанной команды вы можете использовать команду

*git branch*

, чтобы подтвердить, что ваша ветка создана.

Сейчас, если мы запустим branch, мы увидим две доступные опции:

*$ git branch*

*new_feature*

** master*

**Слияние веток**

Добавим в новую ветку файл под названием feature.txt. Мы создадим его, добавим и закоммитим:

*$ git add feature.txt*

*$ git commit -m "New feature complete."*

Изменения завершены, теперь мы можем переключиться обратно на ветку master.

*$ git checkout master*

Теперь, если мы откроем наш проект в файловом менеджере, мы не увидим файла feature.txt, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться merge для объединения веток.

*$ git merge new_feature*

Теперь ветка master актуальна. Ветка new_feature больше не нужна, и ее можно удалить.

*$ git branch -d new_feature*

**Разрешение конфликтов при слиянии**

Конфликты слияния возникают, если вносятся конкурирующие изменения в ту же строку файла или если один пользователь изменяет файл, а другой этот же файл удаляет.

Конфликты регулярно возникают при слиянии ветвей или при отправке чужого кода. Иногда конфликты исправляются автоматически, но обычно с этим приходится разбираться вручную — решать, какой код остается, а какой нужно удалить.

Если вы запустили git merge и возник конфликт слияния, ваш терминал или командная строка ответит вам сообщением:

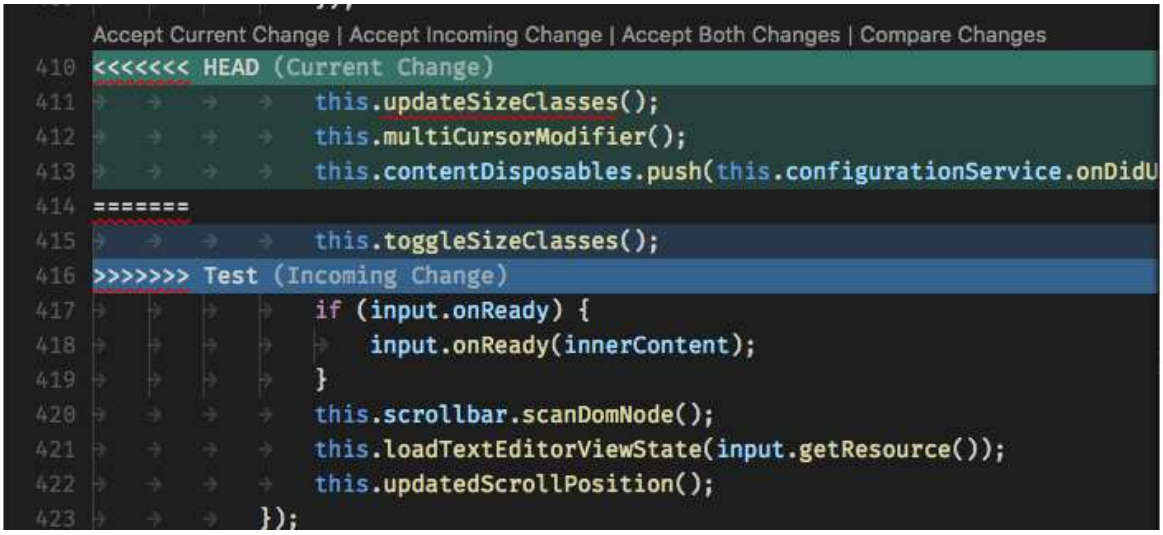
*CONFLICT (content): Merge conflict in [filename]*

Это сообщение говорит нам, в каком конкретно файле возник конфликт.

Найдите конфликт

Откройте файл, на который указал Git, и прокрутите его, пока не найдете конфликт. Ваша IDE может подсказать вам нужное место при помощи подсветки. В примере ниже показано, как это выглядит в VS Code. Редактор подсвечивает текущее изменение и входящее.

- **Текущее изменение** (англ. current change) также иногда называют исходящим. Оно представляет изменения в коде, которые вы сделали в вашей локальной ветке.
- **Входящее изменение** (англ. incoming change) представляет изменения в коде, которые вы вытягиваете (pull) из базовой ветки, или изменения, внесенные другими разработчиками.




Решите, какие изменения нужно применить

То, хотите ли вы принять текущие изменения, входящие изменения или все изменения, зависит от ваших целей. При принятии решения вы ориентируетесь на свое понимание этих изменений.

Если вы не знаете, как поступить, лучше всего посоветоваться с командой или тем разработчиком, который написал входящие изменения.

Вы можете принять изменения, не делая коммит, и локально протестировать программу на работоспособность.

Удалите все длинные последовательности символов ==== , <<<< или >>>>

Эти символы используются для того, чтобы помочь вам определить, где возник конфликт слияния. При принятии выбранных изменений они обычно исчезают, но порой случаются сбои. Проследите за тем, чтобы случайно не включить их в коммит: это может привести к багам в программе.

Что, если я ошибся?

Если вы допустили ошибку или не уверены в том, какие изменения нужно принять, вы можете остановить процесс слияния, запустив следующую команду:

*git merge --abort*

Если вы уверены, что конфликт разрешен, сделайте коммит изменений.

После принятия нужных изменений вы можете сделать коммит. Проделайте следующие шаги:

- Сохраните файлы, в которые были внесены изменения.
- Запустите git status и проверьте, что изменения коснулись правильных файлов.
- Добавьте выбранные файлы в стейджинг: git add [имя файла].
- Сделайте коммит изменений: git commit -m «[ваше сообщение коммита]».
- Запустите git push.

**Репозиторий на GitHub**

Если вы хотите отслеживать только свой код локально, вам не нужно использовать GitHub. Но если вы хотите работать в команде, вы можете использовать GitHub для совместного изменения кода проекта.

Чтобы создать новый репозиторий на GitHub, войдите в систему и перейдите на домашнюю страницу GitHub. Вы можете найти опцию «New repository» под знаком «+» рядом с изображением вашего профиля в правом верхнем углу панели навигации.

После нажатия кнопки GitHub попросит вас назвать репозиторий и предоставить краткое описание.

Когда вы закончите заполнять информацию, нажмите кнопку «Создать репозиторий», чтобы создать новый репозиторий.

GitHub спросит, хотите ли вы создать новый репозиторий с нуля или хотите добавить репозиторий, который вы создали локально. В этом случае, поскольку мы уже создали новый репозиторий локально, мы хотим отправить его на GitHub, поэтому следуйте разделу «...or push an existing repository from the command line»:


После команды

*git push -u origin main*

GitHub попросит Вас войти под своей учетной записью, и только после этого добавит все изменения Вашего проекта.

Теперь мы отправим фиксацию в вашей ветке в новый репозиторий GitHub. Это позволяет другим людям видеть внесенные вами изменения. Если они одобрены владельцем репозитория, изменения могут быть объединены в основную ветвь.

**Получение изменений на свой компьютер**

Чтобы получить самые последние изменения, которые вы или другие пользователи объединили на GitHub, используйте основную команду git pull origin (при работе с основной веткой). В большинстве случаев это можно сократить до «git pull».

**Клонирование репозитория**

Клонирование - это когда вы копируете удаленный репозиторий к себе на локальный ПК. Это то, с чего обычно начинается любой проект. При этом вы переносите себе все файлы и папки проекта, а также всю его историю с момента его создания. Чтобы склонировать проект, сперва, необходимо узнать, где он расположен и скопировать ссылку на него.

*git clone адрес_гит_репозитория*

**Настройка .gitignore**

В большинстве проектов есть файлы или целые директории, в которые мы не хотим (и, скорее всего, не захотим) коммитить. Мы можем удостовериться, что они случайно не попадут в git add -A при помощи файла .gitignore

Создайте вручную файл под названием .gitignore и сохраните его в директорию проекта.

Внутри файла перечислите названия файлов/папок, которые нужно игнорировать, каждый с новой строки.

Файл .gitignore должен быть добавлен, закоммичен и отправлен на сервер, как любой другой файл в проекте.

Вот хорошие примеры файлов, которые нужно игнорировать:

Логи

Артефакты систем сборки

Папки node_modules в проектах node.js

Папки, созданные IDE, например, Netbeans или IntelliJ

Разнообразные заметки разработчика.

Файл .gitignore, исключающий все перечисленное выше, будет выглядеть так:

```
*.log
build/
node_modules/
.idea/
my_notes.txt
```

Символ слэша в конце некоторых линий означает директорию (и тот факт, что мы рекурсивно игнорируем все ее содержимое). Звездочка, как обычно, означает шаблон.