* Для хранения исходного кода программ (и/или других документов) и совместной работы с ним несколькими людьми (если требуется).  
  Сначала тебе надо понять, что такое [система контроля версий](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8).  
  GitLab - это сервис на базе системы контроля версий Git.
* Думаю, ссылку на GitLab большинство здесь присутствующих и так знает. GitHub, GitLab, Bitbucket и т. д. - это всё web-сервисы для хостинга и управления git-репозиториями (не только Git, но другие системы контроля версий менее популярны). Движок GitLab можно установить на собственный сервер и не выкладывать код своих проектов в интернет.
* gitlab можно **бесплатно** установить себе на сервер если нет желания хранить где-то на стороне свой код.
* <https://otus.ru/nest/post/864/> - Что такое GitLab? Настройка и использование GitLab

**GitLab** — веб-приложение и система управления репозиториями программного кода для Git.

GitLab предлагает решение для хранения кода и совместной разработки масштабных программных проектов. Репозиторий включает в себя систему контроля версий для размещения различных цепочек разработки и веток, позволяя разработчикам проверять код и откатываться к стабильной версии софта в случае непредвиденных проблем.

GitLab является конкурентом GitHub, в котором среди многих других проектов размещается разработка ядра Linux Линуса Торвальдса. Поскольку GitLab разрабатывается на той же основе управления версиями (Git), принцип их работы схож. GitLab поддерживает как публичные, так и неограниченное количество частных ветвей разработки.

<https://itglobal.com/ru-ru/company/glossary/gitlab/>

GitLab устанавливают по следующим причинам:

1. Соблюдение коммерческой тайны
2. Изоляция от публичной сети (например развертывание в локальной сети учебного заведения)
3. Уменьшение финансового бремени (приватные репозитарии стоят денег)
4. Уменьшение зависимости от бизнес-модели. Например ГитХаб уважает законы других стран и реагирует на нарушения в правовом поле. ГитЛаб не несет отвественности, поскольку ответственность возглагается на того, кто его устанавливает и как им пользуется. Это значит, что вы можете захоститься где-нибудь в Корее и в репу насовать всякой дряни. ГитЛаб никак не может на это повлять, поскольку это невозможно.
5. [**Cреда**](https://docs.gitlab.com/ce/ci/README.html) непрерывной интеграции у Gitlab лучше.
6. Чтобы сказать Нет [**уродливому сине-фиолетовому цвету**](https://twitter.com/search?q=%23blue%20github&src=typd), который недавно на нем ввели.

<https://medium.com/@cakeinpanic/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-ci-%D0%BD%D0%B0-gitlab-19489cc79fe8> - Основы CI/CD на gitlab

<https://senior.ua/articles/gitlab-protiv-github>

Для жизненного цикла разработки программного обеспечения (SDLC) репозитории с контролем версий представляют собой довольно большую проблему. Они настолько важны, что в последние несколько лет оба: GitHub и GitLab позиционировали себя как инструменты DevOps, которые разработчики предпочитают больше других.

Как следует из их названий, GitLab и GitHub являются веб-хранилищами Git. Мы рассмотрим Git как системы контроля версий вместе с разъяснение GitLab и GitHub.

Услуги по управлению хранилищем контроля версий

Для начала, что такое система контроля версий? По сути, это компонент управления программным обеспечением, который обрабатывает и отслеживает изменения, внесенные в информацию, программное обеспечение и программы. Реальные проекты требуют систем контроля версий, так как обычно над программным проектом работают несколько программистов одновременно. Таким образом, система контроля версий, такая как Git, гарантирует, что в коде нет совпадений или конфликтов.

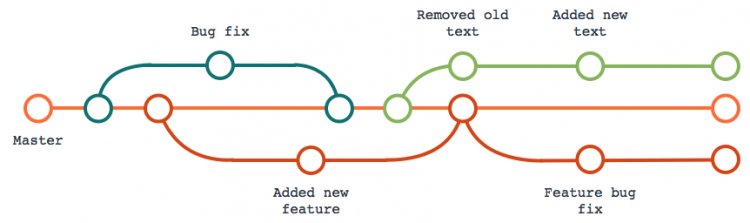
Другим ключевым аспектом систем контроля версий являются изменения в требованиях, которые часто встречаются в средах SDLC. Системы контроля версий тщательно отслеживают версии, чтобы позволить разработчикам вернуться к предыдущим версиям кода. Это особенно полезно, поскольку позволяет вносить изменения, не перерабатывая целые базы кода.

Ключевые преимущества использования системы контроля версий и ее служб управления хранилищами включают

1. рационализация процесса разработки,
2. управление кодом для нескольких проектов одновременно,
3. поддержание чистой истории для всех изменений в коде.

В системах контроля версий все изменения хранятся и отслеживаются в репозитории, что, как мы упоминали ранее, является большим плюсом для сценариев, когда разработчики хотят вернуться к предыдущей версии или исправить ошибку в коде.

Git как система контроля версий



Прежде чем углубиться в спор GitLab против GitHub, давайте выясним, что такое Git: Git - система контроля версий, это стандартное решение defacto и наиболее часто используемая система контроля версий сегодня во всем мире.

Git, с его распределенной системой управления версиями, отслеживает изменения в исходном коде во время SDLC. Созданный в 2005 году Линусом Торвальдсом (создателем Linux) и изначально разработанный как низкоуровневый движок системы контроля версий, Git теперь является полноценным хранилищем, которое является зрелым, открытым исходным кодом.

Теперь, как работает Git? Каждый раз, когда разработчик сохраняет вновь созданный фрагмент кода, Git создает коммит. Фиксация представляет собой моментальный снимок всех файлов в определенный момент времени в процессе разработки. Эти коммиты создают ссылки на другие коммиты для формирования визуального пути истории развития.

Теперь мы должны уточнить, что каждый разработчик сохраняет изменения в своем локальном репозитории кода (ветвях), который впоследствии объединяется с основной ветвью команды. Как мы уже упоминали, отличительной особенностью Git является то, что вы можете вернуться к предыдущим версиям, чтобы вносить изменения по мере необходимости.

Теперь, когда мы узнали немного больше о Git и сервисах управления репозиториями контроля версий, давайте рассмотрим GitHub и GitLab.

GitHub против GitLab

Теперь, чтобы полностью понять дилемму GitHub против GitLab, давайте рассмотрим суть вопроса: что такое GitHub и что такое GitLab?.

Что такое GitHub?

**GitHub** - это платформа разработки, которую разработчики используют для хранения проектов - отсюда и название hub, связанное с именем. Веб-хостинг GitHub, поскольку он основан на Git, является службой контроля версий, которая обеспечивает контроль доступа к таким функциям, как отслеживание ошибок, запросы функций, управление задачами, и многие другие. GitHub в основном используется для кода благодаря встроенным инструментам для его проверки и улучшения.

GitHub доступен для общественности в виде бесплатного сервиса с открытым исходным кодом, который обеспечивает удаленный доступ к репозиториям Git, обеспечивает размещение кода и помогает управлять жизненным циклом разработки программного обеспечения. Кроме того, он включает в себя такие функции, как совместное использование кода несколькими людьми, отслеживание ошибок, пространство вики и другие инструменты для так называемого социального кодирования.

Что такое GitLab?

**GitLab** - это GitHub-подобный сервис, который обеспечивает внутреннее веб-управление DevOps для управления Git-репозиториями. GitLab предлагает два варианта: бесплатную версию и платную версию для предприятий. GitLab включает в себя множество необходимых функций для успешного управления процессом разработки программного обеспечения, таких как вики, интеграция с Jira, CI Runner, управление выпусками, двоичные вложения и многое другое.

Одной из ключевых характеристик GitLab является то, что он написан на языке программирования Ruby. Он предлагает модель разработки с открытым ядром, а также решение с открытым исходным кодом. G

Благодаря более чем 100 000 организаций и пользователей, использующих GitLab, она стала 4-й наиболее быстрорастущей частной компанией по разработке программного обеспечения в 2018 году. Популярность GitLab частично объясняется его полной совместимостью с технологиями непрерывной интеграции. Интеграция помогает на разных этапах разработки упростить перекрестную ссылку на код, тесты и развертывания, предлагая полную прозрачность для разработчиков для контекста и быстрых итераций по мере необходимости.

Сходство между GitHub и GitLab

Вот некоторые из ключевых сходств между GitHub и GitLab:

1. **Отслеживание.** И GitLab, и GitHub предлагают фантастические возможности отслеживания проблем, которые позволяют изменять статус и позволяют назначать владельцев для каждой проблемы.
2. **Интеграции.** И GitLab, и GitHub имеют широкий спектр сторонних интеграций, доступных для усиления рабочих процессов и повышения производительности. GitHub предлагает сторонние интеграционные решения для специализированных инструментов и приложений. GitLab предлагает многочисленные интеграции для команд разработчиков и DevOps.
3. **Поддержка сообщества.** Хотя сообщество поддержки GitHub считается немного большим, оба решения предлагают сильное сообщество разработчиков, которые вносят вклад в обновление и поддержку GitLab и GitHub.
4. **Метки.** И GitLab, и GitHub обеспечивают легкое включение меток, чтобы обеспечить простой метод классификации проблем, запросов на слияние на основе описательных заголовков, таких как «ошибка» или «документация», которые очень полезны для отслеживания.
5. **Шаблоны описания.**Разработчики могут добавить шаблон описания к проблемам или объединить запросы в GitLab и GitHub, выбрав шаблон для эффективной коммуникации.
6. **Корпоративные решения.** GitLab и GitHub предлагают возможности уровня предприятия для групп разработчиков и организаций. Хотя GitLab лучше позиционируется в корпоративной среде, оба решения хорошо работают в корпоративных настройках.
7. **Вики-проектная документация.** И GitLab, и GitHub хранят отдельную систему для документации, которая называется Wiki и встроена в каждый проект как отдельный репозиторий Git.
8. **Правки.**Обе платформы позволяют сторонним разработчикам совместно работать над исходной веткой и редактировать их в ветвь.
9. **Вернуть коммиты.** GitHub и GitLab способны отменить коммиты или один запрос на слияние через пользовательский интерфейс каждой платформы.

Другие ключевые функции, которые совместно используются GitHub и GitLab, включают в себя запрос на извлечение, репозитории fork/clone и еще многое . Далее мы сосредоточимся на различиях между двумя платформами.

Разница между GitHub и GitLab

Вот некоторые из ключевых отличий между GitHub и GitLab:

1. **Встроенная непрерывная интеграция.**GitLab известен своей бесплатной встроенной непрерывной интеграцией, которую GitHub не предлагает. Вместо этого GitHub предлагает сторонние интеграции, которые обеспечивают непрерывную интеграцию.
2. **Аутентификация.**В GitLab разрешение предоставляется на основе ролей людей, в то время как в GitHub разработчики могут предоставлять доступ на чтение или запись к определенным репозиториям.
3. **Импорт/экспорт данных.** GitLab предлагает гораздо более подробную документацию о том, как импортировать/экспортировать данные от внешних поставщиков, в то время как документация GitHub не такая подробная. GitLab способен импортировать проекты и проблемы из большего количества источников, чем GitHub. При этом GitHub предлагает инструмент под названием GitHub Importer для ввода данных. Что касается экспорта, GitLab предлагает комплексное решение для экспорта вики, репозиториев проектов, загрузки проектов, веб-хуков и сервисов, а также проблем. С другой стороны, GitHub немного более ограничен с точки зрения возможностей экспорта.
4. **Платформа развертывания.** GitHub не поставляется со встроенной платформой развертывания и требует сторонней интеграции с внешним приложением для развертывания приложений. С другой стороны, GitLab использует Kubernetes для беспроблемного развертывания.
5. **Частные репозитории.** GitLab предлагает бесплатные частные репозитории для проектов с открытым исходным кодом, а GitHub - нет.
6. **Отслеживание комментариев.**GitHub способен предоставить полную историю обновлений комментариев - GitLab не поддерживает это.
7. **Экспорт файла CSV.** GitLab способен экспортировать файлы CSV с проблемами на адреса электронной почты уведомлений по умолчанию в виде вложений.
8. **Конфиденциальные вопросы.** Модуль GitLab Confidential Issues создает конфиденциальные проблемы, которые видны только участникам проекта с уровнем доступа Reporter или выше.
9. **Графики выгрузки.**В отличие от GitHub, GitLab предлагает Burndown Charts как часть этапов, которые позволяют разработчикам отслеживать прогресс во время спринтов или при работе над новыми версиями программного обеспечения.
10. **Циклическая аналитика.**GitLab предоставляет панель мониторинга для анализа времени, планирования и мониторинга.
11. **Ежемесячные новые функции.**GitLab известен своими ежемесячными обновлениями новых функций и улучшений, которые неукоснительно делаются 22-го числа каждого месяца.

Помимо тех, что мы только что указали, основное различие между GitHub и GitLab заключается в философии каждой платформы. GitHub больше ориентирован на высокую доступность и производительность инфраструктуры, в то время как GitLab больше ориентирован на включение как можно большего количества функций в надежную и хорошо интегрированную платформу для полного и централизованного процесса DevOps.

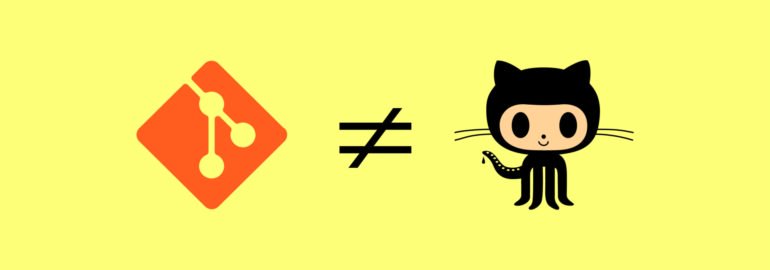
Последний штрих в обсуждении GitHub против GitLab

 В конечном счете, и GitHub, и GitLab настроены на DevOps.

Подводя итог, можно сказать, что общедоступный репозиторий GitHub - это платформа для совместной работы, которая размещает и просматривает код, управляет проектами и помогает создавать программное обеспечение - с помощью функций распределенного контроля версий и управления исходным кодом. С другой стороны, GitLab - это веб-репозиторий, который предлагает функции совместной работы, которые в значительной степени сосредоточены на наборе инструментов DevOps, и предлагает непрерывную интеграцию.

<https://tproger.ru/translations/difference-between-git-and-github/>

Git и GitHub: что это такое и в чём разница



[Alpár-Etele Méder](https://www.iconfinder.com/pocike); [Picons.me](https://www.iconfinder.com/Picons)

Перевод статьи [«Difference Between Git and GitHub»](https://medium.com/@eduoshaun/difference-between-git-and-github-807f1a57d438)

Вы пишете код, запускаете его, и всё работает как надо. Вы добавляете новую фичу, и всё перестаёт работать. Каждому разработчику знакома ситуация, когда технология не хочет ему подчиняться. Ничто не идеально, и порой что-нибудь ломается. Иногда на поиски небольшой ошибки могут потребоваться часы работы. В таких случаях на помощь приходят системы контроля версий.

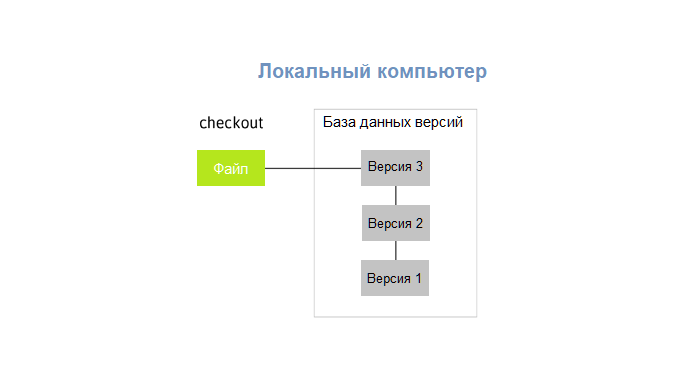
Что такое система контроля версий?

Системы контроля версий (СКВ, VCS, Version Control Systems) позволяют разработчикам сохранять все изменения, внесённые в код. Поэтому в случае, описанном выше, они могут просто откатить код до рабочего состояния вместо того, чтобы тратить часы на поиски маленькой ошибки или ошибок, ломающих весь код.

СКВ также дают возможность нескольким разработчикам работать над одним проектом и сохранять внесённые изменения, чтобы убедиться, что все могут следить за тем, над чем они работают.

Существует три типа СКВ: локальная, централизованная и распределённая.

Локальные системы контроля версий (ЛСКВ)

[](https://s3.tproger.ru/uploads/2018/07/LVCS-1.png)  
Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию, возможно даже в директорию с отметкой по времени для большего контроля. Данный подход всё ещё очень популярен и распространён. Изменения сохраняются в виде наборов патчей, где каждый патч датируется и получает отметку времени. Таким образом, если код перестаёт работать, наборы патчей можно совместить, чтобы получить исходное состояние файла.

Централизованные системы контроля версий (ЦСКВ)

[](https://s3.tproger.ru/uploads/2018/07/CVCS-1.png)  
ЦСКВ были созданы для решения проблемы взаимодействия с другими разработчиками. Такие системы имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища и там же их сохраняют. Тем не менее, такой подход имеет существенный недостаток — выход сервера из строя обернётся потерей всех данных.

Прим.перев. А ещё в таких системах может быть затруднена одновременная работа нескольких разработчиков над одним файлом.

Распределённые системы контроля версий (РСКВ)

[](https://s3.tproger.ru/uploads/2018/07/DVCS-1.png)

Недостаток ЦСКВ был исправлен в РСКВ, клиенты которых не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени), а полностью копируют репозиторий. Это значит, что у каждого клиента есть копия всего исходного кода и внесённых изменений. В этом случае, если один из серверов выйдет из строя, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Ещё одним преимуществом РСКВ является то, что они могут одновременно взаимодействовать с несколькими удалёнными репозиториями, что означает, что вы можете параллельно работать над несколькими проектами.

Теперь вы понимаете, что из себя представляют системы контроля версий и что они делают, а также знаете, каких типов они бывают. Теперь перейдём непосредственно к Git и его отличиям от GitHub.

Что такое Git?

[](https://s3.tproger.ru/uploads/2018/07/Git.jpg)  
[Git](https://git-scm.com/about) — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, для того, чтобы другие разработчики могли вносить свой вклад в ядро Linux. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Преимущества Git

* **Бесплатный и open-source**. Это значит, что его можно бесплатно скачать и вносить любые изменения в исходный код;
* **Небольшой и быстрый**. Он выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных;
* **Резервное копирование**. Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при использовании Git;
* **Простое ветвление**. В других СКВ создание веток— утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее.

Теперь пора разобраться, что такое GitHub и как он работает с Git.

Что такое GitHub?

[](https://s3.tproger.ru/uploads/2018/07/GitHub.jpg)  
[GitHub](https://github.com/) — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах.

Также GitHub может похвастаться контролем доступа, багтрекингом, управлением задачами и вики для каждого проекта. Цель GitHub — содействовать взаимодействию разработчиков.

К проекту, загруженному на GitHub, можно получить доступ с помощью интерфейса командной строки Git и Git-команд. Также есть и другие функции, такие как документация, запросы на принятие изменений (pull requests), история коммитов, интеграция со множеством популярных сервисов, email-уведомления, эмодзи, графики, вложенные списки задач, система @упоминаний, похожая на ту, что в Twitter, и т.д.

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

<https://proglib.io/p/gitlab-ili-github-kak-vybrat-resurs-pod-opredelennyy-tip-repozitoriya-2020-01-06>

GitLab или GitHub? Как выбрать ресурс под определённый тип репозитория

GitLab не так популярен, как GitHub. Но исходный код открыт, а приватные репозитории компаний бесплатны. Кратко обобщаем, для каких видов проектов какой инструмент лучше применять.

GitHub и GitLab – альтернативные системы управления репозиториями кода для [Git](https://proglib.io/p/git-for-half-an-hour/" \t "_blank). Оба ресурса предоставляют возможности для хостинга IT-проектов и их совместной разработки. В этой публикации мы кратко сравним услуги и цели, для которых вы можете использовать каждую из систем.

Отличительные черты GitHub и прогресс прошлого года

* Появилась возможность [спонсировать проекты](https://github.com/sponsors).
* GitHub [догнал](https://github.com/marketplace/category/continuous-integration) GitLab по [непрерывной интеграции](https://about.gitlab.com/blog/2016/08/05/continuous-integration-delivery-and-deployment-with-gitlab/).
* [Улучшена](https://github.com/pricing) модель ценообразования.
* Добавлены [Actions](https://proglib.io/p/github-actions" \t "_blank), [статусы](https://developer.github.com/v3/repos/statuses/), поддержку пакетов.
* Улучшена интеграция c [VSCode](https://github.com/microsoft/vscode" \t "_blank).
* Колоссальное, продолжающее расти сообщество пользователей.
* Проекты проще получают поддержку от контрибьюторов.
* Добавлена автоматическая [проверка безопасности](https://github.com/features/security) и [отчёты](https://github.com/marketplace/category/reporting).
* Ресурс стал активнее в социальных сетях ([Twitter](https://twitter.com/github" \t "_blank) и т.п.)
* Отслеживание и анализ активности пользователей.
* По-прежнему взимается плата с [закрытых репозиториев компаний](https://github.com/pricing).
* Плоская организация.
* [Скандальное](http://valleywag.gawker.com/github-cofounder-resigns-after-internal-harassment-inve-1565714008) прошлое.
* Ненулевая вероятность, что Microsoft сделает что-то дурное.

Чем отвечает GitLab

* До сих пор многие не знают, что [он существует.](https://about.gitlab.com/)
* Приватный репозиторий [создаётся из командной строки](https://docs.gitlab.com/ee/gitlab-basics/create-project.html).
* [Безопаснее](https://about.gitlab.com/security/).
* Имеет [подгруппы](https://docs.gitlab.com/ee/user/group/subgroups/) для иерархической организации.
* Закрытые репозитории бесплатны, группы не ограничены.
* Красивый и удобный интерфейс, который [можно использовать даже для GitHub](https://about.gitlab.com/solutions/github/).
* Открытый исходный код.

Личные и групповые учётные записи

В отличие от GitLab, за приватные репозитории на GitHub нужно платить. Это привело к неприятному побочному эффекту. Значимые библиотеки и модули, пакеты и целые фреймворки связаны с отдельными учётными записями. Даже если человек уже не имеет никакого отношения к обслуживанию проекта. Впрочем, GitHub обратил внимани на это упущение и стал развивать функционал [организаций и команд](https://help.github.com/en/github/setting-up-and-managing-organizations-and-teams). В репозиториях организаций теперь есть команды, можно создавать подгруппы, и для каждой указывать свои настройки и доступы к репозиториям.

Бесплатность командных приватных репозиториев повлияла на то, что компании нередко выбирают GitLab. Но функционал [subgroups](https://docs.gitlab.com/ee/user/group/subgroups/" \t "_blank) недоступен для личных учётных записей. Поэтому группы на GitLab стали важнее персональных аккаунтов. Многие разработчики называют группы своим никнеймом, а чтобы отличать имя собственной записи, добавляют в начале нижнее подчёркивание.

В свою очередь GitHub с давних пор поддерживает идею о том, что значимая работа в программировании делается людьми по отдельности. Поощряющие «награды» (форки, звёзды, лента коммитов) должны храниться в их «личном кабинете». Это привело к тому, что публичная активность на GitHub учитывается при поиске персонала, на [конференциях](https://proglib.io/events) и т. д.

Что хранить на GitLab

Кроме веб-сервиса GitLab предоставляет [открытый исходный код](https://gitlab.com/gitlab-org/gitlab-foss). Систему можно быстро и безболезненно запустить на собственном сервере. Если пользоваться веб-сервисом, то для следующих задач:

* Закрытый контент команд, компаний, организаций.
* Личные рабочие репозитории (для хранения «на всякий случай»).
* Персонализированные данные.
* Проекты учебных материалов.
* Веб-разработка (удобный инструмент – [GitLab Pages](https://docs.gitlab.com/ee/user/project/pages/" \t "_blank)).

Для всего остального есть GitHub

Весь прочий контент можно смело направлять на GitHub. Если вы хотите, чтобы код был доступен другим, чтобы им пользовались, чтобы на него ссылались. Ссылки на репозитории GitHub вызывают большее доверие и будут просматриваться чаще. Публичная активность и проекты составят портфолио, которое может повлиять на трудоустройство.

<https://www.it-blog.club/articles/gitlab/github-vs-gitlab-kto-luchshe/>

GitHub vs GitLab Кто лучше

Ещё вот в [Этой статье](https://www.it-blog.club/articles/git_main/github-vs-bitbucket-kto-luchshe/) мы уже разобрали разницу между GitHub и BitBucket. Победителем вышел второй. Но тогда я напомнил о существовании ещё и GitLab'а. Так давайте теперь рассмотрим разницу между GitHub и GitLab.

Как обычно, я бы хотел начать из далека.

Git - это платформа, на которой участники команды разработки могут хранить свой код, в случае если какой-либо или же каждый участник работает удалённо. Так же многие компании используют GIT, для контроля работы сотрудников, кто-то "накосячил" и кто где чего не до выполнил. В случаи не качественно выполненной задачи, последствия которой понесли за собой проблемы на всём проекте, GIT позволяет откатится до более раннего состояния проекта. Так что ни одна организация, команда разработчиков, ну или даже просто программист одиночка не должны пренебрегать Гитом.

Существует не мало Git платформ, но самые пожалуй самые популярные это GitHub, BitBucket и GitLab. Помимо услуг по хранению и извлечению данных, эти платформы позволяют клиентам проверять код, выяснять проблемы и предлагать решения проблем. Пожалуй это больше похоже на работу платформу программирования с расширенными функциями.

По большому счёту, современные Git платформы, по мимо простого хранения кода, позволяют полностью управлять жизненным циклом проекта. Существует не мало расширений, которые вы можете интегрировать в свою платформу, и которые будут помогать вам в различных задачах.

Различия между GitHub и GitLab

Обе эти платформы позволяют разработчикам хранить, делиться, тестировать и совместно работать над одним проектом. GitHub начал свою работу в 2008 году и на данный момент насчитывает 26 миллионов пользователей. Так же он является самым популярным среди разработчиков, которым нужно выложить свой проект в открытый доступ. Так же на GitHub имеются и приватные репозитории, но такие стоят некую сумму.

По мимо уже перечисленного, GitHub может проводить анализ проблем на проекте, отслеживать их и даже устранять, работая совместно с документацией.

GitLab начал свою работу в 2011 году и насчитывает уже более 1 миллиона активных участников, некоторые даже являются разработчиками из таких крупных компаний как IBM, Sony и NASA. Он имеет большинство функций GitHub, но так же и свои собственные уникальные особенности.

Непрерывная интеграция

Каждый раз, когда участник команды загружает код или изменяет ранее сохраненный исходный код, скрипт должен быть создан и протестирован на предмет проблем. Построение кода означает его интеграцию с уже существующим проектом. GitLab предлагает автоматическую интеграцию, которая означает каждый раз, когда в проект добавляется новый код, он перестраивает модель проекта и тестирует его проблемы учитывая нововведения. Это и называется непрерывной интеграцией или же CI. И да. Как вы уже поняли, у GitLab есть собственный CI сервис бесплатно.

GitHub же использует сторонние CI сервисы, из-за чего разработчик должен сам, вручную, интегрировать его каждый раз, чего занимает не мало времени.

Так что в удобстве работы с интеграциями побеждает GitLab, из-за автоматизированной системы CI.

Разрешение пользователя

GitHub позволяет разработчику или менеджеру проекта полностью управлять уровнями доступов для каждого участника команды. Вы можете добавить нового участника и выдать ему права, например только на чтение файлов проекта. При этом этот участник не сможет вносить изменения в файлы проекта.

НО! GitLab предлагает лучшую гибкость и контроль при управлении проектом. А всё потому что он позволяет админу проекта устанавливать более различные настройки для каждого участника. И в этих настройках вы можете найти больше чем просто Чтение и Запись.

Вы можете дать кому-то проверить проблему в исходном коде, НО запретить его изменения. Разумеется это лишь пример. Система контроля участников даёт уклон на то, что вы можете подключить человека из вне, который не отвечает за проект, но после тестирования, сможет проконсультировать вас, как стоит поступить с тем или иным участком кода.

Вы можете подумать, в чём тогда разница от доступа на Чтение. Разница в том, что этот человек может даже не иметь прав на чтение всего проекта! Он будет иметь доступ только на тестирование. Тем самым он сможешь увидеть конфликты, но не увидит весь исходный код.

Собственно думаю тут даже спорить не стоит кто зарабатывает ещё один балл. Разумеется это GitLab из-за более гибкой и продвинутой системой прав доступа к проекту.

Проблемы отслеживания и интеграции

Нет никаких сомнений в том, что обе платформы имеют надёжные средства отслеживания проблем. Вы можете назначить конкретных участников для решения проблем, тем самым просто дав им на это права. Но не забывайте, что в большинстве случаев, ваши участники команды, должны уметь работать с необходимыми решениями которые предлагают обе платформы.

По мимо просто отслеживания ошибок, GitHub и GitLab предлагают расширения для автоматических утверждений модификаций. Для этого, они оба используют сторонний трекер Tracker Usersnap. Благодаря ему, тестеры смогут проверить ваш проект и оставить свой отзыв. Менеджер же сможет собрать отзывы, отчёты и сообщения с обратной связи с пользователями, и направить команде разработчиков для решения проблем.

Но проблема в том, что для всего выше сказанного, требуются всё так же различные расширения. И всё бы хорошо, но вот только GitLab, при работе с большинством сторонних расширений, всё ещё находится словно в подростковом состоянии. По этой причине он без проблем работает только с популярными дополнениями. В тот момент как у GitHub уже имеет довольно весомый рынок всевозможных дополнений для повышения производительности работы над проектом.

Собственно в этом и заключаются проблемы отслеживания и интеграций. GitHub практически в "сухую" побеждает GitLab, наличием больших дополнений и возможностей. Так что он честно зарабатывает себе первый балл.

Импорт и экспорт данных

Когда вы работаете над проектом, документация играет важную роль, особенно когда вы нанимаете фрилансеров из разных частей мира, ну или ходя бы страны) Зачастую, при найме кого-то удалённо, возможно потребуется переход с одной платформы на другую. Необходимость может заключаться в том, что команда субподряда может работать на GitHub, а тот момент как ваш проект на GitLab. Ну или же учитывая, что GitHub силён в открытых репозиториях, а его соперник наоборот в закрытых. То в соответствиями с требованиями к проекту, вам придётся переносить проект с одной платформы на другую.

Вот этот момент знаю по себе. Так как у нас в компании, команда разработчиков, которой мы скидываем заказы, работает из другой страны, но нам повезло, что ничего переносить не пришлось. Но была так же ситуация, когда одна из компаний работала на GitLab, а субподрядчики работали с BitBucket. Вот тут как раз и пригодилось бы то, о чём сейчас пойдёт речь.

 В случае переноса проекта с одной платформы на другую, вам необходимо экспортировать и импортировать данные. GitLab предоставляет полную документацию о том, как перейти от GitLab на GitHub. GitHub же не предоставляет подобной документации, что создаёт не мало проблем в случае перехода.

По этой самой причине, GitLab снова получает дополнительный балл, благодаря подробной документации и лёгкой миграции на другие платформы.

Ценообразование и сообщество

GitHub более популярен, и его популярность можно объяснить ранним началом работы, а так же очень активным сообществом. В GitLab ещё нет большого и активного сообщества. Он начал свою деятельность немного позже. По этой причине большинство разработчиков больше знают о GitHub и начинают работу именно на нём.

Корпоративная оплата в GitHub начинается от 84 долларов за одного пользователя в год, а у GitLab от 39 долларов за участника, так же в год.

GitLab предоставляет не ограниченное количество как приватных, так и открытых репозиториев. GitHub же предоставляет не ограниченное количество только публичных.

Вот тут я пожалуй добавлю каждой платформе по баллу. Потому что у GitLab лучше цены. А у GitHub более продвинутое сообщество и он более популярен.

Итог

Несмотря на то, что популярность GitHub намного выше чем у GitLab, тренд показывает, что белее крупные команды разработчиков начинанию переходить с GitHub на GitLab, благодаря расширенным возможностям второго. Которые могут сохранить и время и деньги, там самым сделать работу над проектом более оптимизированной и эффективной.

Если вам понравилась данная статья, то обязательно подписывайтесь на обновления блога, не забывайте про группу в ВКонтакте, и поставьте ОГОНЬ! под этой статьёй!

**Практика**:

* git --version
* данные можно установить локально для какого-то проекта или глобально:

git config --global user.name Berny83

git config --global user.email [diana\_ust@mail.ru](mailto:diana_ust@mail.ru)

git config user.name - просматриваем

git config

git config --list - проверка настроек

* git - выведет все команды
* git init - инициализация гита, создание локального репозитория
* “git add css/” “git add . (git add -A)” “git add index.html” “git add \*.html” “git add css/\*.html” “git add css/\*\*/\*.html” - во всех дочерних папках все файлы с расширением html - добавление файлов в стадию ожидания; Из стадии ожидания их нужно закоммитить.
* “git add !css/style.css” - все файлы из папки css, кроме файлв style.css; “git add !\*.html” - все файлы кроме файлов с расширением html.
* git status
* git rm --cached index.html - удаляем файл из стадии ожидания.
* git commit -m “” - добавление файлов в локальное хранилище/репозиторий c комментарием.
* git log - отслеживаем все изменения в проекте; git log --oneline - все изменения в сокращенной форме
* .gitignore

Игнорирование файлов

Git имеет встроенную функцию .gitignore, с помощью которой мы можем предотвратить случайное попадание в git add ненужных файлов, папок и директорий. Очень часто в такой перечень попадают следующие данные:

* Логи;
* Артефакты систем сборки;
* Папки node\_modules в проектах node.js;
* Папки, созданные IDE, например, Netbeans или IntelliJ;
* Разнообразные заметки разработчика.

Работает функция .gitignore очень просто:

* Создаем вручную файл под названием .gitignore и сохраняем его в директорию проекта.
* Внутри файла перечисляем названия файлов/папок, которые нужно игнорировать, каждое с новой строки.
* Файл .gitignore добавляем, коммитим и отправляем на сервер, как любой другой файл в проекте. Пример исключения перечисленных выше файлов выглядит так:

\*.log

build/

node\_modules/

.idea/

my\_notes.txt

* git checkout <commit\_id> - просмотр проекта на определенной стадии коммита; git checkout master - вернуть проект на последнюю точку сохранения.
* git revert < commit\_id > - отменяет действие определенного коммита. Мы его не удаляем, мы отменяем.
* git reset <commit\_id> --hard - удаляем коммиты после указанного <commit\_id>, а также все данные будут возвращены к коммиту <commit\_id>.
* git branch <name\_of\_branch> - создание новой ветки
* git checkout <name\_of\_branch> - переход на новую ветку; git checkout master - переход на основную ветку.
* git branch -a - список всех веток
* git checkout -b admin - создание ветки и переход на нее.
* git merge <с какой веткой хочу объединить> - слияние веток
* git branch -d <branch\_name> - удалить ветку на локальном репозитории
* git push origin --delete <branchName> - удаление ветки на удаленном репозитории
* git fetch [remote-name] - git fetch origin - забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.
* git pull - если у вас есть ветка, настроенная на отслеживание удалённой ветки (обратитесь к главе 3 Ветвление в Git за более подробной информацией), то вы можете использовать команду git pull чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей. Этот способ может для вас оказаться более простым или более удобным. К тому же, по умолчанию команда git clone автоматически настраивает вашу локальную ветку master на отслеживание удалённой ветки master на сервере, с которого вы клонировали репозиторий. Название веток может быть другим и зависит от ветки по умолчанию на сервере. Выполнение git pull, как правило, извлекает (fetch) данные с сервера, с которого вы изначально склонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

Этот вариант склонирует ПОЛНОСТЬЮ весь репозиторий и просто переключится на ветку branch-name:

$ git clone http://whatever.git -b branch-name

Иногда нужно склонировать ТОЛЬКО конкретную ветку, тогда:

$ git clone --branch=branch-name http://whatever.git

Еще иногда может потребоваться склонировать репозиторий без многолетней истории его коммитов, тогда нужно использовать параметр --depth=5, где 5 - количество коммитов от последнего:

$ git clone --depth=1 --branch=branch-name <http://whatever.git>

<https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D1%83%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F%D0%BC%D0%B8>

* git init - инициализипуем локальный репозиторий
* git add - добавление файлов в стадию ожидания;
* git commit - добавление файлов в локальное хранилище;
* git remote add origin <http://whatever.git> - подключаемся к удаленному репозиторию; git remote add <shortname> <url> - <shortname> - присваиваем ему имя (origin)
* README.md - markdown - язык разметки для текст.редакторов. — облегчённый язык разметки, созданный с целью написания наиболее читаемого и удобного для правки текста, но пригодного для преобразования в языки для продвинутых публикаций (HTML, Rich Text и других). Язык текстовой разметки, созданный писателем и блогером Джоном Грубером. Он предназначен для создания красиво оформленных текстов в обычных файлах формата TXT.

<https://lifehacker.ru/chto-takoe-markdown/>

<https://www.markdownguide.org/basic-syntax/>

* git push -u origin master - git push <remote-name> <branch-name> - отправить наработки/изменения в удалённый репозиторий
* git remote - показывает к какому удаленному репозиторию подключены; выведет названия доступных удалённых репозиториев.
* git remote -v - адреса для чтения и записи, привязанные к репозиторию
* git remote show <remote> - выдаёт URL удалённого репозитория, а также информацию об отслеживаемых ветках
* git fetch - связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет.
* git branch --set-upstream-to=origin/master master - связывает tracking ветки master локального репозитория с master веткой удаленного репозитория origin.

У меня была такая же ошибка, когда я сначала создал локальный репозиторий. Потом перешел на ГитХаб и создал новый репозиторий там. Затем выполнил

$ git remote add origin <repository url>

А когда попробовал сделать push/pull, получил ту же fatal: unrelated\_histories ошибку. И вот как я поправил это:

$ git pull origin master --allow-unrelated-histories

$ git merge origin origin/master

... здесь выполняем команды add и commit ...

$ git push origin master

У вас есть там изменения, которых нет в локальном репозитории. Вы сделали не пустой репозиторий на Github, а с начальным коммитом. У локального репозитория свой первый коммит и эти коммиты, как вы понимаете, не равны (фактические у вас просто два разных репозитория).

Просто удалите папку .git локально, создайте репозиторий (git init) заново, пропишите upstream (git remote add ...) и после этого можно делать git pull. Должно помочь :)

The “fatal: refusing to merge unrelated histories” Git error occurs when two unrelated projects are merged (i.e., projects that are not aware of each other’s existence and have mismatching commit histories).

<https://www.educative.io/edpresso/the-fatal-refusing-to-merge-unrelated-histories-git-error>

* git push -u origin master - git push <remote-name> <branch-name>. Чтобы отправить вашу ветку master на сервер origin (повторимся, что клонирование обычно настраивает оба этих имени автоматически).
* git clone <url-address> <name\_of\_folder> - можно и без имени, и останется имя проекта, которое мы ему дали при создании на гитхабе.
* git pull - скопировать все изменения, готовые файлы и т.п.

# Failed to Push Some Refs | Git Error

<https://articles.assembla.com/en/articles/748133-failed-to-push-some-refs-git-error>

**Удаление и переименование удалённых репозиториев**

Для переименования удалённого репозитория можно вылолнить **git remote rename**. Например, если вы хотите переименовать pb в paul, вы можете это сделать при помощи **git remote rename**:

$ git remote rename pb paul

$ git remote

origin

paul

Стоит упомянуть, что это также изменит имена удалённых веток в вашем репозитории. То, к чему вы обращались как **pb/master**, теперь стало **paul/master**.

Если по какой-то причине вы хотите удалить удаленный репозиторий — вы сменили сервер или больше не используете определённое зеркало, или кто-то перестал вносить изменения — вы можете использовать **git remote rm**:

$ git remote rm paul

$ git remote

origin

При удалении ссылки на удалённый репозиторий все отслеживаемые ветки и настройки, связанные с этим репозиторием, так же будут удалены. Репозиторий на гитхабе остался.

# #3 - Отмена действий. Возвращение к старым версиям

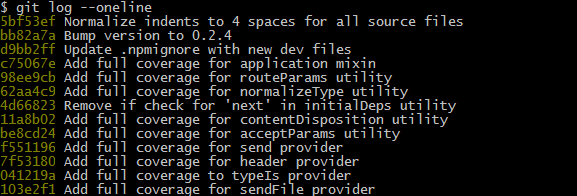
В Git есть замечательная возможность «возвращения во времени»:

* вы можете просмотреть как выглядел ваш проект на стадии определенного коммита;
* вы можете отменить определенный коммит;
* вы можете выполнить удаление коммита и возвращение к проекту на стадии определенного коммита.

### Просмотр проекта

Чтобы просмотреть проект на стадии какого-либо коммита вам необходимо прописать команду git checkout id, где id это идентификатор коммита, который хотите просмотреть.

Чтобы узнать id изначально пропишите команду git log --oneline. Тогда вы получите список всех коммитов что были сделаны в вашем проекте, а также их идентификаторы.



Чтобы просмотреть какой-либо коммит выполните комманду: git checkout 102e2f1. Здесь вы просмотрите проект на стадии первого коммита.

### Отмена коммита

Для отмены коммита используйте команду git revert id, где id это идентификатор коммита, который хотите просмотреть. После выполнения команды вы отмените определенный коммит.

### Удаление коммита

Для удаления коммита используйте команду git reset id --hard, где id это идентификатор коммита, который хотите просмотреть. После выполнения команды вы удалите определенный коммит и все данные будут возвращены к проекту что был на стадии коммита. Просто git reser id - удаляет коммиты, но оставляет файлы нетронутыми/неизмененными.

# #4 - Работа с ветками. Объединение веток

Ветки в Git существуют для комфортной работы нескольких сотрудников над одним проектом. При помощи веток вы можете создавать ответвления от главного проекта и реализовать лишь нужный для вас функционал. В уроке вы познакомитесь с Git Branch и Git Merge (слияние веток).

Так как работой над одним проектом может заниматься несколько разработчиков, то принято вносить изменения каждым из них в свою копию оригинальной версии кода. Это позволяет сохранять рабочий вариант без изменений до тех пор, пока не будет принято решение слить все версии в одну.

Каждая такая копия проекта называется веткой – она изолирована от других и имеет собственную историю.

Наличие веток позволяет обезопасить проект, сравнивать различные версии, вести параллельную разработку.

### Создание ветки

Для создания ветки используется команда git branch name, где name это название ветки, которую вы хотите создать.

Чтобы перейти на ветку используйте команду git checkout name, где name это название ветки, на которую вы хотите перейти. Чтобы перейти на основную ветку пропишите слово «master».

### Объединение веток

Для объединения веток используйте команду git merge name, где name это ветку, которую вы хотите присоединить к главной ветке.