[1) Что такое Git?](https://www.lenakso.top/chto-takoe-git/)

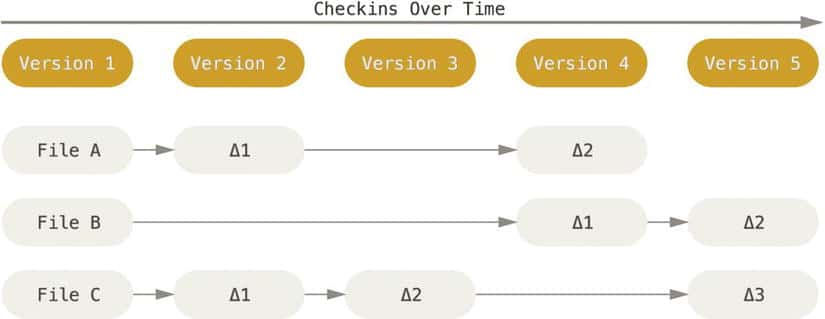
Автор: [wpmaster](https://www.lenakso.top/author/lenakso/" \o "Записи wpmaster) | 13.03.2020

[Нет комментариев](https://www.lenakso.top/chto-takoe-git/#respond)

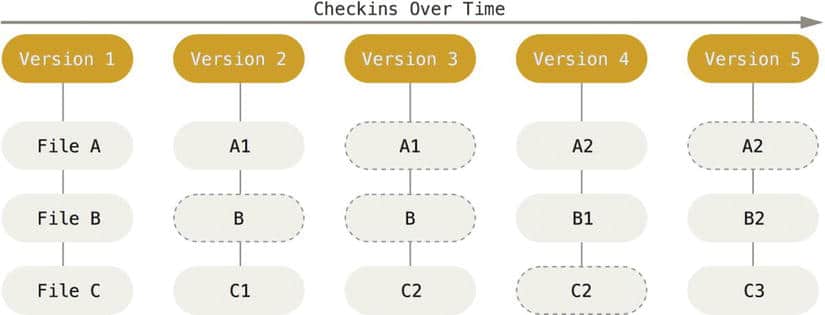
Git-это распределенная система управления версиями с открытым исходным кодом. В распределенной системе управления версиями локальная копия сохраняется на каждом узле (компьютере человека, который является частью проекта). Существует также централизованный сервер, где все члены команды продвигают свои изменения. Таким образом, Git устойчив к сбоям, поскольку каждый узел имеет свою собственную копию исходного дерева.

**Как Git хранит изменения**

Git хранит изменения в файлах иначе, чем другие системы управления версиями, такие как SVN и CVS. Это одна из важных концепций Git, которую вы должны усвоить как можно раньше. Системы контроля версий хранят разницу между двумя версиями. Например, рассмотрим файл А, который менялся три раза. Первая версия файла будет сохранена как есть, в смысле сохранится полный файл. По мере появления новых версий будут сохраняться только отличия от предыдущей версии. Это станет яснее, когда мы посмотрим на изображение ниже, это изображение показывает, как в течение нескольких ревизий изменения сохраняются для трех файлов.



Другие CVS хранят дельту (изменения) в исходном файле с течением времени. Git, напротив, сохраняет моментальный снимок измененного файла. Например, если вы внесли изменения в файл A, сохраняется полный снимок измененного файла. Если файл не изменился между двумя версиями, Git сохранит ссылку на исходный файл вместо того, чтобы копировать его снова в новой версии. Ниже изображение суммирует, как Git внутренне хранит изменения.



Подводя итог этому разделу, можно отметить три важных момента, которые мы узнали о Git:

* *Git хранит моментальный снимок файла в отличие от хранения разницы, что делают другие системы управления версиями.*
* *Git делает только снимок измененных файлов.*
* *Чтобы оптимизировать память, Git сохраняет ссылку на файл, который не изменился вместо того, чтобы делать его копию в новой версии.*

**Git-это распределенная система управления версиями**

Git имеет удаленный репозиторий, который хранится на сервере, и локальный репозиторий, который хранится на компьютере каждого разработчика. Это означает, что код не просто хранится на центральном сервере, но и полная копия кода присутствует на всех компьютерах разработчиков. Поскольку каждый узел имеет локальную копию, почти все операции в Git являются локальными (исключение составляют команды Pull и Push). Это означает, что вам не нужно все время быть подключенным к удаленному хранилищу, чтобы делать свою работу.

С другой стороны, системы управления версиями, такие как CVS и SVN, требуют подключения к серверу для каждой операции. Это дает Git значительное преимущество в скорости. Так как большинство операций выполняются локально, то они проходят очень быстро. Например, если вы хотите внести изменения, это можно сделать прямо в вашей системе. Позже, когда у вас будет несколько коммитов для отправки, вы можете отправить свои изменения на центральный сервер для всех членов команды. Аналогично, если вы хотите видеть историю изменений, внесенных в проект, вам не нужно сетевое подключение. Все изменения можно просмотреть только из локальной копии.

Благодаря локальному хранению репозитория Git может обеспечить значительное преимущество в скорости по сравнению с другими системами управления версиями.

Читайте также: [Как создать сайт на WordPress](https://www.lenakso.top/sozdanie-saita-na-wordpress/).

**Зачем использовать Git?**

С момента разработки и выпуска Git, он приобрел огромную популярность среди разработчиков и, будучи открытым исходным кодом, включил в себя множество функций. Сегодня огромное количество проектов используют Git для контроля версий, как коммерческих, так и персональных. Давайте посмотрим, почему Git стал таким популярным, обсудив его основные особенности

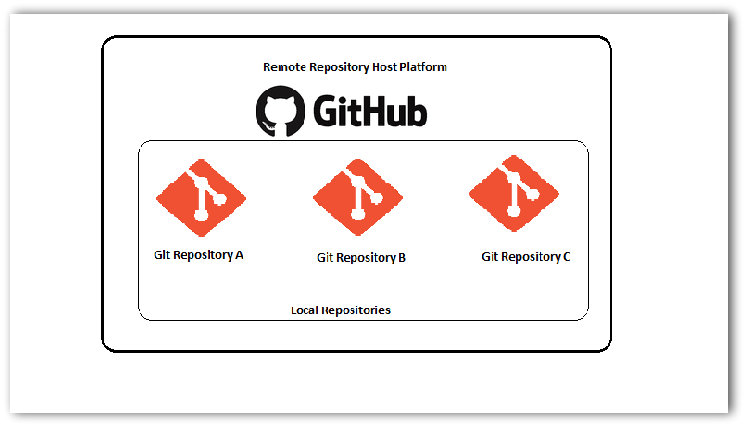
**Производительность:** Git обеспечивает наилучшую производительность, когда речь заходит о системах управления версиями. Committing, branching, merging — все это оптимизировано для лучшей производительности, чем другие системы.  
**Безопасность:** Git обрабатывает вашу безопасность с помощью криптографического метода SHA-1. Алгоритм надежно управляет версиями, файлами и каталогами, чтобы ваша работа не была повреждена.  
**Модель ветвления:** Git имеет другую модель ветвления, чем остальные VCS. Модель ветвления Git позволяет иметь несколько локальных ветвей, независимых друг от друга. Наличие этого также позволяет вам без трений переключать контекст (переключаться назад и вперед к новой фиксации, коду и обратно), код на основе ролей (ветвь, которая всегда идет в производство, другая в тестирование и т. д.) и одноразовые эксперименты (попробуйте что-то, если не работает, удалите его без потери кода).  
**Промежуточная область:** Git имеет промежуточную стадию, называемую «индекс” или «промежуточная область (staging area)», где коммиты могут быть отформатированы и изменены до завершения коммита.  
**Открытый исходный код:** это очень важная особенность любого программного обеспечения, присутствующего сегодня. Будучи открытым исходным кодом, разработчики со всего мира могут внести свой вклад в программное обеспечение и сделать его все более мощным с помощью функций и дополнительных плагинов.

2) [Разница между git и github](https://www.lenakso.top/raznitsa-mezhdu-git-i-github/)

Автор: [wpmaster](https://www.lenakso.top/author/lenakso/" \o "Записи wpmaster) | 20.03.2020

[Нет комментариев](https://www.lenakso.top/raznitsa-mezhdu-git-i-github/#respond)

В ИТ-индустрии часто используются два термина: Git и GitHub. Оба относятся к двум различным сущностям, давайте посмотрим, в чем разница между GIT и GITHUB?  
**Git:** это распределенная система контроля версий для отслеживания версий файлов.  
**Github:** это веб-портал и облачный хостинг для ваших репозиториев Git.



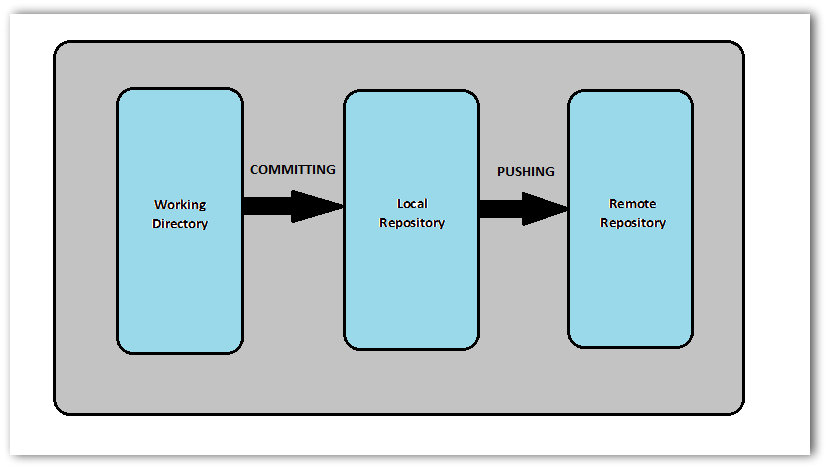
Git позволяет отслеживать и размещать версии файлов на Github. Другими словами, вы используете команды Git для отслеживания версий ваших файлов. А Github — это просто удаленная платформа, на которой размещаются эти файлы.

**В чем разница между GIT и GitHub?**

Git позволяет отслеживать версии кода на локальном компьютере. Однако если вы хотите создать удаленную резервную копию своего кода или опубликовать ее в сообществе, то вам нужно отправить ее на Github. (Подробнее об этом в следующих статьях)

Использование Github не является обязательным. Если вы единственный человек, работающий над проектом, и вам не нужно публиковать свой код, вы можете отказаться от использования Github. Просто отследите версии вашего кода в локальном репозитории вашей машины с помощью Git.

Фиксация кода с помощью Git на вашем локальном компьютере не будет автоматически загружать его на Github. Есть два различных термина для этих действий- Committing и Pushing. Committing (фиксация) — это запись изменений из рабочей копии в локальный репозиторий. С другой стороны, Pushing — это загрузка захваченных изменений из вашего локального репозитория на Github.



Ни один из этих процессов не является автоматическим. Вы должны вручную выбрать файлы, которые вы хотите отслеживать. Таким образом, вы можете разрешить и ограничить отслеживание и публикацию контента. Пример: предполагается, что вы создаете временный класс java, чтобы попробовать определенную функцию. Поскольку это временный файл для написания экспериментального кода, вы не хотите его фиксировать. Таким образом, вы можете исключить файлы из отслеживания, зафиксировав их вручную.

GitHub — это самый популярный на сегодняшний день git хостинг. Главный секрет его успеха-бесплатный план с бесплатными неограниченными публичными хранилищами. Хотя вам нужно платить за частный репозиторий, почти каждый проект с открытым исходным кодом выбирает GitHub. Есть много других хороших функций, таких как отслеживание проблем или обмен фрагментами исходного кода и т. д., они выделяют его среди других.

3) [Этапы жизненного цикла GIT](https://www.lenakso.top/etapy-zhiznennogo-tsikla-git/)

Автор: [wpmaster](https://www.lenakso.top/author/lenakso/" \o "Записи wpmaster) | 27.03.2020

Файлы в проекте Git имеют различные стадии, такие как создание, модификация, рефакторинг, удаление и так далее. Независимо от того, отслеживается ли этот проект Git или нет, эти фазы все еще преобладают. Однако, когда проект находится под управлением системы управления версиями [Git](https://www.lenakso.top/chto-takoe-git/" \t "_blank), они присутствуют в трех основных состояниях Git в дополнение к этим основным. Вот три состояния Git:

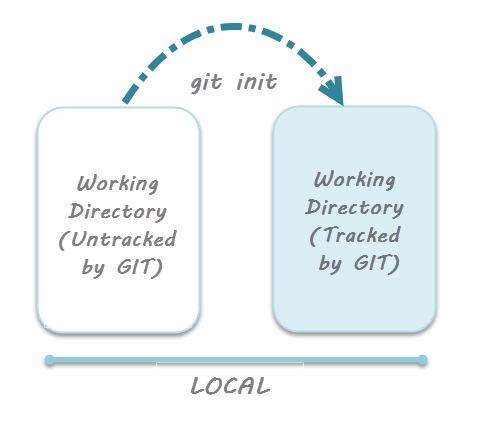
* Рабочий каталог (Working directory)
* Плацдарм (Staging area)
* Каталог Git (Git directory)

Эти стадии и есть суть Git. Вы получаете большую гибкость в отслеживании файлов благодаря этим этапам, на которых файлы могут находиться в Git. Давайте разберемся в каждом из этих состояний по порядку.

**Рабочий каталог**

Рассмотрим проект, находящийся в вашей локальной системе. Этот проект может быть отслежен Git, а может и не отслеживаться. В любом случае этот каталог проекта называется рабочим каталогом.

Рабочий каталог — это каталог, содержащий скрытую папку .git.



примечание: **git init** – команда для инициализации репозитория Git

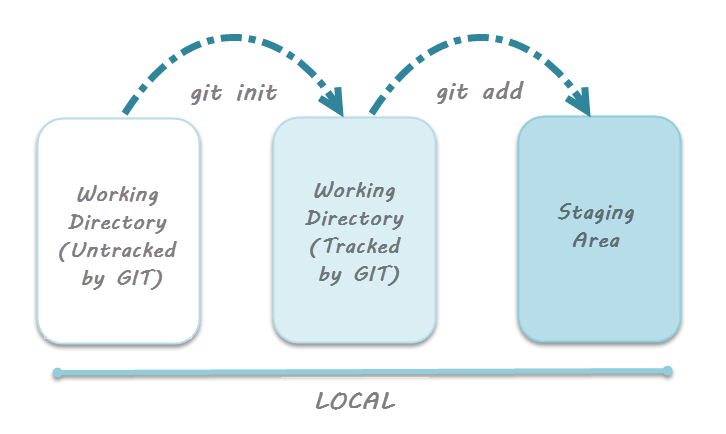
Для дальнейшего обсуждения предположим, что этот каталог теперь отслеживается Git. То есть мы создали репозиторий Git в этом существующем каталоге проекта. В этом состоянии Git просто знает о файлах в проекте. Он еще не отслеживает файлы. Чтобы отслеживать файлы, мы должны зафиксировать эти файлы, предварительно добавив их в промежуточную область. Это приводит нас к следующему состоянию в жизненном цикле Git.

**Этапы жизненного цикла GIT: Плацдарм**

Пока мы находимся в рабочем каталоге, мы выбираем файлы, которые должны быть отслежены Git. Зачем нам это нужно? Почему бы нам не отслеживать все в проекте? Это связано с тем, что некоторые файлы в проекте, такие как файлы классов, файлы журналов, файлы результатов и временные файлы данных, создаются динамически. Нет смысла отслеживать версии этих файлов. В то время как файлы исходного кода, файлы данных, файлы конфигурации и другие артефакты проекта содержат бизнес-логику приложения. Эти файлы должны отслеживаться Git, поэтому их необходимо добавить в промежуточную область.

Другими словами, промежуточная область — это площадка, где вы группируете, добавляете и организуете файлы, которые будут переданы в Git для отслеживания их версий.

Здесь важно быстро отметить термин, называемый индексацией. Индексирование — это процесс добавления файлов в промежуточную область. Другими словами, индекс состоит из файлов, добавленных в промежуточную область.

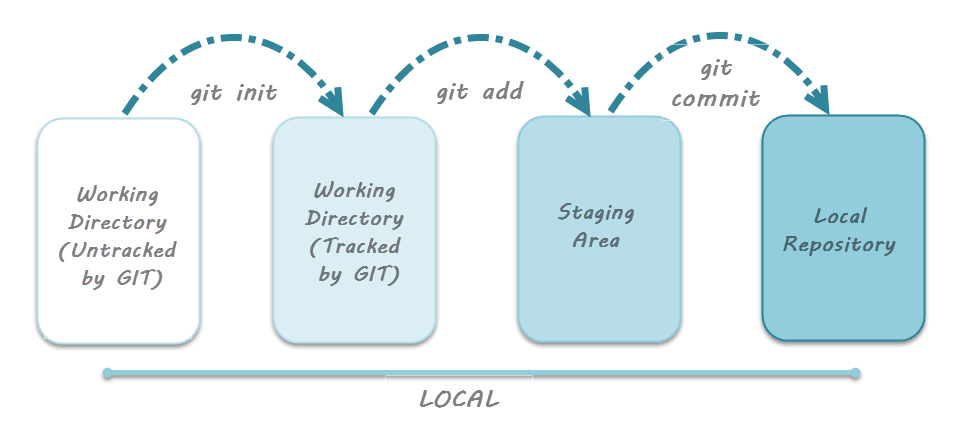


Примечание:**git add** — команда для добавления файлов в промежуточную область.

**Каталог Git**

Теперь, когда файлы, подлежащие фиксации, сгруппированы и готовы в промежуточной области, мы можем зафиксировать эти файлы. Итак, мы фиксируем эту группу файлов вместе с сообщением о фиксации (commit), объясняющим, что мы изменили. Помимо сообщения о фиксации, этот шаг также записывает автора и время коммита. Затем, моментальный снимок файлов фиксируется с помощью Git. Информация, связанная с этим коммитом (имена файлов, дата и время коммита, автор коммита, сообщение о коммите), хранится в каталоге Git.

Таким образом, каталог Git — это база данных, в которой будут отслеживаться метаданные об истории файлов проекта.

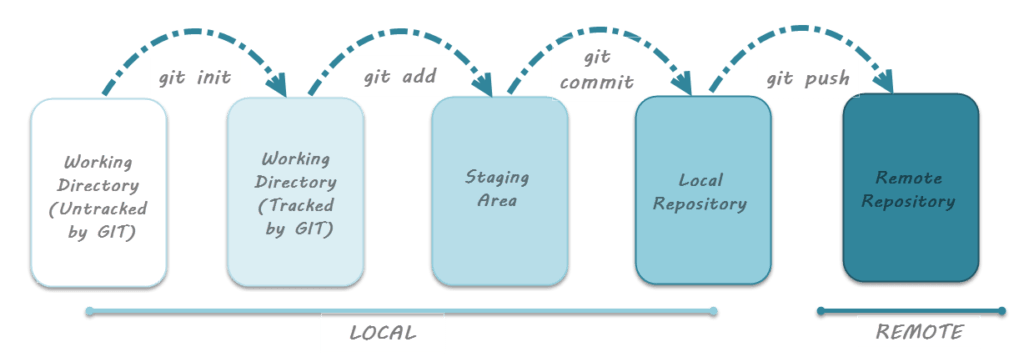


примечание: **git commit-m** «ваше сообщение» — команда для фиксации файлов в репозиторий Git с сообщением.

**Дополнительный этап жизненного цикла с Github**

Имейте в виду, что мы изучаем жизненный цикл исключительно в Git.  То есть важно отметить, что три стадии, рассмотренные выше, предназначены только для Git, а не для Github. Почему? Потому что вы можете отслеживать версии ваших файлов, используя только Git. То есть Github необходим, когда вы хотите сотрудничать и публиковать свой код в команде или сообществе. Таким образом, полезно помнить, что цикл Git обычно не включает Github.

Допустим, что мы работаем в командах и сотрудничаем с несколькими людьми над данным проектом. Это делает необходимым понять дополнительную стадию, связанную с Github. При работе с Github существует понятие удаленного хранилища и связанный с ним процесс, называемый Pushing the files.



**Жизненный Цикл Git**

Как видно на рисунке выше, после фиксации кода в локальном репозитории Git он должен быть перемещен в удаленный репозиторий. Здесь под удаленным репозиторием подразумевается зеркало или клон локального репозитория Git в Github. А push означает загрузку коммитов из локального репозитория Git в удаленный репозиторий, размещенный в Github. Это позволит другим сотрудникам просматривать код. Таким образом, любые изменения, внесенные вами в локальное хранилище Git, будут видны другим сотрудникам, когда вы переместите свой код в удаленное хранилище. Команда для отправки кода в удаленный репозиторий в Github — это git push.

Примечание: **git push**-команда для отправки коммитов из локального репозитория Git в удаленный репозиторий Git, размещенный в Github.

**Необходимость в промежуточной области**

Узнав об этих стадиях, можно, естественно, спросить – зачем нужна промежуточная зона? Почему мы не можем непосредственно зафиксировать код вместо того, чтобы сначала добавить его в промежуточную область? Давайте разберемся в причинах этого с помощью следующих моментов:

**Быстрота.** commit — это, по сути, ресурсоемкое взаимодействие с базой данных Git. Операции фиксации имеют тенденцию быть медленными. Однако в случае git, размещение файлов не требует взаимодействия с базой данных Git. Только когда файлы должны быть зафиксированы, Git проверяет наличие коммитов, сделанных другими пользователями. Таким образом, staging помогает записывать изменения еще до их фиксации в базе данных Git.

**Визуализация** **фиксации перед фактической фиксацией**. Как обсуждалось ранее, промежуточная область — это состояние, в котором находятся файлы до их фиксации. То есть промежуточная область фактически позволяет визуализировать группу изменений, которые будут записаны Git. По сути, это дает вам четкий контроль над тем, что попадает в Git, а что нет.

**Разделение работы на отдельные связанные коммиты**. Кроме первого коммита, когда все файлы проекта фиксируются одновременно, вы должны записывать (делать снимок проекта) через регулярные промежутки времени. То есть предположим, что вы работаете над новой функцией, срок разработки которой составляет несколько дней. Таким образом, пока вы работаете над функцией, вам необходимо выполнить рефакторинг небольшой части кода. В этом случае вы можете быстро внести необходимые изменения, создать необходимый файл и возобновить работу над функцией. Все изменения, т. е. изменения функций и рефакторинг кода могут быть зафиксированы сразу из промежуточной области.