Реферат на тему:  
 «Система контроля версий – Git»

Выполнил:

Студент группы ИВТЗ-21

НГТУ им. Алексеева

Зинкин Дмитрий Александрович.

2022

**Оглавление**

[1. Что такое Git? Для чего он нужен? 3](#_Toc106110023)

[2. Целостность Git и добавление данных. 4](#_Toc106110024)

[2.1. Три состояния. 4](#_Toc106110025)

[2.2. Командная строка 6](#_Toc106110026)

[3. Установка Git 6](#_Toc106110027)

[3.1. Установка в Linux 6](#_Toc106110028)

[3.2. Установка в Windows 7](#_Toc106110029)

[4. Ключ к пониманию 7](#_Toc106110030)

[5. Указатели 8](#_Toc106110031)

[6. Основные команды для использования GitHub. 8](#_Toc106110032)

[6.1. Как задать имя пользователя и адрес электронной почты? 8](#_Toc106110033)

[6.2. Инициализация репозитория 9](#_Toc106110034)

[6.3. Добавление отдельных файлов или всех файлов в область подготовленных файлов 9](#_Toc106110035)

[6.4. Просмотр истории коммитов с изменениями 9](#_Toc106110036)

[6.5. Удаление отслеживаемых файлов из текущего рабочего дерева 10](#_Toc106110037)

[6.6. Слияние двух веток 10](#_Toc106110038)

[6.7. Отображение журнала фиксации в виде графика для текущей или всех веток 11](#_Toc106110039)

[6.8. Прекращение слияния при конфликте 11](#_Toc106110040)

[6.9. Добавление удалённого репозитория 12](#_Toc106110041)

[6.10. Просмотр удалённых URL-адресов 12](#_Toc106110042)

[6.11. Получение дополнительных сведений об удалённом репозитории 12](#_Toc106110043)

[6.12. Отправка изменений в удалённый репозиторий 13](#_Toc106110044)

[6.13. Получение изменений из удалённого репозитория 13](#_Toc106110045)

[7. Окончание 13](#_Toc106110046)

# **Что такое Git? Для чего он нужен?**

Сейчас мы поговорим что же такое «Система контроля версий».

Что же такое Git? Git хранит и использует информацию совсем по-другому по сравнению с другими СКВ. (Subversion, Perforce и т.д.) Главное отличие Git – это подход к работе со своими данными. Большинство другим систем хранят информацию в виду списка изменений в файлах, эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле по времени. Обычно это называют контролем версий, основанным на различиях.

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. 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. Можно сказать, что Git хранит в себе каждую версию файла/проекта, если он был, когда и изменен.

Помимо отката проектов к предыдущей версии, он удобен для работы в команде. Ведь каждое изменение отслеживается, и в нужный момент возвращается. Изменение же происходит под определенным аккаунтом, а Root уже выбирает принимать эти изменения или отбросить в сторону.

Все операции выполняются локально – значит, чтобы посмотреть историю проекта, Git не надо соединяться с сервером для её получения и отображения. Из чего следует что вы можете увидеть ее почти моментально. Так же это означает что, если вы находите без доступа к интернету, есть лишь небольшое количество действий, которые вы не сможете выполнить.

# **Целостность Git и добавление данных.**

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его философии. Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

Git пользуется SHA-1 hash для вычисления хеш сумм. Это строка длинной в 40 шестнадцатеричных символов (0-9 и a-f). Она вычисляется на основе содержимого файла или структуры каталога.

Выглядит это примерно так.

24b9da6552252987aa493b52f8696cd6d3b00373

Git обычно только добавляет данные.

Когда вы производите какие-то действия в Git, практически все из них только ДОБАВЛЯЕТ новые элементы в базу Git. Очень сложно заставить систему удалить данные. Как и в любой другой СКВ , вы можете потерять свои записанные данные , пока они не зафиксированы , как только вы зафиксируете снимок в Git , будет очень сложно что-либо потерять, особенно , если вы регулярно синхронизируете свою базу с другим репозиторием.

# **Три состояния.**

У Git есть 3 основных состояния, в которых могут находиться файлы:

\* 1. Изменен - К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

\* 2. Индексированный — это изменённый файл в его текущей версии, отмеченный для включения в следующий коммит.

\* 3. Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

Мы подошли к трём основным секциям проекта Git: рабочая копия (working tree), область индексирования (staging area) и каталог Git (Git directory).

Рабочая копия 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 индексирования — 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», но фраза «область индексирования» также работает.

Каталог 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.

Базовый подход в работе с Git выглядит так:

1. Изменяете файлы вашей рабочей копии.
2. Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в индекс.
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш каталог Git.

Если определённая версия файла есть в каталоге Git, эта версия считается зафиксированной (committed). Если файл был изменён и добавлен в индекс, значит, он индексирован (staged). И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается изменённым (modified). В главе [Основы Git](https://git-scm.com/book/ru/v2/ch00/ch02-git-basics-chapter) вы узнаете больше об этих состояниях и какую пользу вы можете извлечь из них или как полностью пропустить часть с индексом.

# **Командная строка**

Есть много различных способов использования Git. Помимо оригинального клиента, имеющего интерфейс командной строки, существует множество клиентов с графическим пользовательским интерфейсом, в той или иной степени реализующих функциональность Git. В рамках данной книги мы будем использовать Git в командной строке. С одной стороны, командная строка — это единственное место, где вы можете запустить **все** команды Git, так как большинство клиентов с графическим интерфейсом реализуют для простоты только некоторую часть функциональности Git. Если вы знаете, как выполнить какое-либо действие в командной строке, вы, вероятно, сможете выяснить, как тоже самое сделать и в GUI-версии, а вот обратное не всегда верно. Кроме того, в то время как выбор графического клиента — это дело личного вкуса, инструменты командной строки доступны всем пользователям сразу после установки Git.

Поэтому мы предполагаем, что вы знаете, как открыть терминал в Mac или командную строку, или PowerShell в Windows. Если вам не понятно, о чем мы здесь говорим, то вам, возможно, придётся ненадолго прерваться и изучить эти вопросы, чтобы вы могли понимать примеры и пояснения из этой книги.

# **Установка Git**

# **Установка в Linux**

Если вы хотите установить Git под Linux как бинарный пакет, это можно сделать, используя обычный менеджер пакетов вашего дистрибутива. Если у вас Fedora (или другой похожий дистрибутив, такой как RHEL или CentOS), можно воспользоваться dnf:

$ sudo dnf install git-all

Если же у вас дистрибутив, основанный на Debian, например, Ubuntu, попробуйте apt:

$ sudo apt install git

Чтобы воспользоваться дополнительными возможностями, посмотрите инструкцию по установке для нескольких различных разновидностей Unix на сайте Git <https://git-scm.com/download/linux>.

# **Установка в Windows**

Для установки Git в Windows также имеется несколько способов. Официальная сборка доступна для скачивания на официальном сайте Git. Просто перейдите на страницу <https://git-scm.com/download/win>, и загрузка запустится автоматически. Обратите внимание, что это отдельный проект, называемый Git для Windows; для получения дополнительной информации о нём перейдите на [https://gitforwindows.org](https://gitforwindows.org/).

# **Ключ к пониманию**

Ключ к пониманию концепции git — знание о «трех деревьях»:

Рабочая директория — файловая система проекта (те файлы, с которыми вы работаете).

Индекс — список отслеживаемых git-ом файлов и директорий, промежуточное хранилище изменений (редактирование, удаление отслеживаемых файлов).

Директория .git/ — все данные контроля версий этого проекта (вся история разработки: коммиты, ветки, теги и пр.).

Коммит — «сохранение» (хранит набор изменений, сделанный в рабочей директории с момента предыдущего коммита). Коммит неизменен, его нельзя отредактировать.

У всех коммитов (кроме самого первого) есть один или более родительских коммитов, поскольку коммиты хранят изменения от предыдущих состояний.

# **Указатели**

HEAD — указатель на текущий коммит или на текущую ветку (то есть, в любом случае, на коммит). Указывает на родителя коммита, который будет создан следующим.

ORIG\_HEAD — указатель на коммит, с которого вы только что переместили HEAD (командой git reset ..., например).

Ветка (master) — указатель на коммит. При добавлении коммита, указатель ветки перемещается с родительского коммита на новый.

Теги — простые указатели на коммиты. Не перемещаются.

# **Основные команды для использования GitHub.**

# **Как задать имя пользователя и адрес электронной почты?**

Имя пользователя нужно, чтобы привязывать коммиты к вашему имени. Это не то же самое, что имя пользователя учётной записи GitHub, с помощью которого выполняется вход в профиль на GitHub. Задать или изменить имя пользователя можно с помощью команды git config. Новое имя будет автоматически отображаться в последующих коммитах, отправленных на GitHub через командную строку. Если хотите скрыть своё реальное имя, можно использовать в качестве имени пользователя Git произвольный набор символов.

/\*\* Глобальная область видимости \*\*/

git config --global user.name "\*\*Zinkin Dmitriy\*\*"

Кроме того, командой git config можно изменять адрес электронной почты, привязанный к вашим коммитам Git. Новый адрес электронной почты будет автоматически отображаться во всех дальнейших коммитах, поданных на GitHub через командную строку.

git config --global user.email [Zinkin.Dmitry@bk.ru](mailto:Zinkin.Dmitry@bk.ru)

# **Инициализация репозитория**

Создать пустой репозиторий Git или вновь инициализировать существующий можно параметром init. При инициализации он создаст скрытую папку. В ней содержатся все объекты и ссылки, которые Git использует и создаёт в истории работы над проектом.

git init

# **Добавление отдельных файлов или всех файлов в область подготовленных файлов**

Добавить отдельный файл в область подготовленных файлов можно параметром add с указанием имени файла. Просто замените somefile.js на актуальное имя.

git add somefile.java// в нашем случае по лабе html и jpeg

Кроме того, можно добавить все файлы и папки в эту область, предоставив wildcard . вместо имени файла:

git add .

# **Просмотр истории коммитов с изменениями**

Просматривать изменения, внесённые в репозиторий, можно с помощью параметра log. Он отображает список последних коммитов в порядке выполнения. Кроме того, добавив флаг -p, вы можете подробно изучить изменения, внесённые в каждый файл.

git log -p

Чтобы просмотреть изменения до commit’a , можно использовать параметр diff и добавить флаг –staged.

git diff

Так же можно просмотреть изменения только в одном файле.

Для этого место указания флага используется имя файла.

git diff index.html

# **Удаление отслеживаемых файлов из текущего рабочего дерева**

Удалять файлы из текущего рабочего дерева можно с помощью параметра rm. При этом файлы удаляются и из индекса.  
  
git rm dirname/somefile.html

Можно также использовать маски файлов (например \*.js) для удаления всех файлов, соответствующих критерию.  
  
git rm dirname/\*.html

# **Слияние двух веток**

Объединить две ветки можно параметром merge с указанием имени ветки. Команда объединит указанную ветку с основной.  
  
git merge existing\_branch\_name

Если надо выполнить коммит слияния, выполните команду git merge с флагом --no-ff.  
  
git merge --no-ff existing\_branch\_name

Указанная команда объединит заданную ветку с основной и произведёт коммит слияния. Это необходимо для фиксации всех слияний в вашем репозитории.

# **Отображение журнала фиксации в виде графика для текущей или всех веток**

Просмотреть историю коммитов в виде графика для текущей ветки можно с помощью параметра log и флагов --graph --oneline --decorate. Опция --graph выведет график в формате ASCII, отражающий структуру ветвления истории коммитов. В связке с флагами --oneline и --decorate, этот флаг упрощает понимание того, к какой ветке относится каждый коммит.  
  
git log --graph --oneline --decorate

Для просмотра истории коммитов по всем веткам используется флаг --all.  
  
git log --all --graph --oneline –decorate

# **Прекращение слияния при конфликте**

Прервать слияние в случае конфликта можно параметром merge с флагом --abort. Он позволяет остановить процесс слияния и вернуть состояние, с которого этот процесс был начат.  
  
git merge --abort

Также при конфликте слияния можно использовать параметр reset, чтобы восстановить конфликтующие файлы до стабильного состояния.  
  
git reset

# **Добавление удалённого репозитория**

Добавить удалённый репозиторий можно параметром remote add, указав shortname и url требуемого репозитория.  
  
git remote add awesomeapp https://github.com/someurl..

# **Просмотр удалённых URL-адресов**

Просматривать удалённые URL-адреса можно параметром remote с флагом -v. Этот параметр отображает удалённые подключения к другим репозиториям.  
  
git remote -v

Такая команда открывает доступ к интерфейсу управления удалёнными записями, которые хранятся в файле .git/config репозитория.

# **Получение дополнительных сведений об удалённом репозитории**

Получить подробные сведения об удалённом репозитории можно с помощью параметра remote show с указанием имени репозитория — например, origin.  
  
git remote show origin

Эта команда отображает список веток, связанных с удалённым репозиторием, а также рабочих станций, подключённых для получения и отправки файлов.

# **Отправка изменений в удалённый репозиторий**

Отправлять изменения в удалённый репозиторий можно параметром push с указанием имени репозитория и ветки.  
  
git push origin main

Эта команда передаёт локальные изменения в центральный репозиторий, где с ними могут ознакомиться другие участники проекта.

# **Получение изменений из удалённого репозитория**

Для загрузки изменений из удалённого репозитория используется параметр pull. Он скачивает копию текущей ветки с указанного удалённого репозитория и объединяет её с локальной копией. git pull

Также можно просмотреть подробные сведения о загруженных файлах с помощью флага --verbose. git pull --verbose

Слияние удалённого репозитория с локальным выполняется параметром merge с указанием имени удалённого репозитория.

git merge origin

Фунций в GitHub очень много. Выше мы рассмотрели на мой взгляд одни из важнейших функций для работы с Git’ом.

# **Окончание**

Git – это система контроля версий и изменений в программе.

Представим, что нам в каком-то крайнем случае, надо откатить программу на несколько версий назад, в случае с Git’ом нам легко это сделать. Git позволяет выборочно принимать или отклонять изменения в программе, сливать ветки в едино, удалять файлы – тем самым позволяет удобно работать в команде даже УДАЛЕННО.

Современная разработка веб-приложений, да и не только не обходится без git’a. Git - неотъемлемая часть жизненного цикла разработки программ.