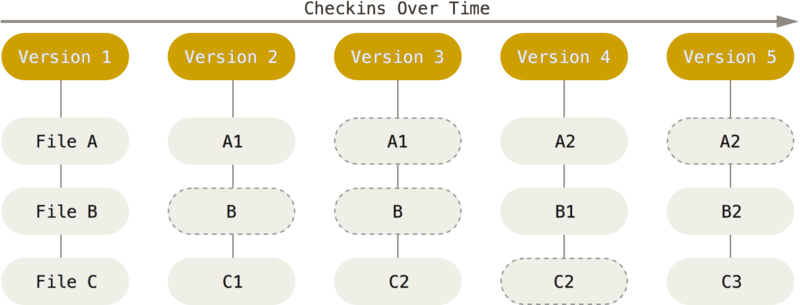
# Git

Материал основан на официальной документации <https://git-scm.com/book/ru/v2/>

Что такое «система контроля версий» и почему это важно? Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий (далее СКВ) — как раз то, что нужно. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.  
  
Очень удобно думать о версиях, как о фотографиях и машине времени. Фотографии хранят информацию о том, как все было в момент, когда фотография была сделана, и позволяют вам вернуться в прошлое.

Подход Git похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит (снимок), 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. Git представляет свои данные как, скажем, поток снимков.



Почти все операции выполняются локально

Для работы большинства операций в Git достаточно локальных файлов и ресурсов — в основном, системе не нужна никакая информация с других компьютеров в вашей сети. Так как вся история проекта хранится прямо на вашем локальном диске, большинство операций кажутся чуть ли не мгновенными.

Для примера, чтобы посмотреть историю проекта, Git не нужно соединяться с сервером для её получения и отображения — система просто считывает данные напрямую из локальной базы данных. Это означает, что вы увидите историю проекта практически моментально. Если вам необходимо посмотреть изменения, сделанные между текущей версией файла и версией, созданной месяц назад, Git может найти файл месячной давности и локально вычислить изменения, вместо того, чтобы запрашивать удалённый сервер.

Это также означает, что есть лишь небольшое количество действий, которые вы не сможете выполнить, если вы находитесь оффлайн или не имеете доступа к VPN в данный момент. Если вы в самолёте или в поезде и хотите немного поработать, вы сможете создавать коммиты без каких-либо проблем (в вашу локальную копию, помните?): когда будет возможность подключиться к сети, все изменения можно будет синхронизировать. Если вы ушли домой и не можете подключиться через VPN, вы всё равно сможете работать.

### Целостность Git

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 директории так, чтобы Git не узнал об этом.

Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длинной в 40 шестнадцатеричных символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Вы будете постоянно встречать хеши в Git, потому что он использует их повсеместно. На самом деле, Git сохраняет все объекты в свою базу данных не по имени, а по хеш-сумме содержимого объекта.

### Git обычно только добавляет данные

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не зафиксированы, но после того, как вы зафиксируете снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Всё это превращает использование Git в одно удовольствие, потому что мы знаем, что можем экспериментировать, не боясь серьёзных проблем. Для более глубокого понимания того, как Git хранит свои данные и как вы можете восстановить данные, которые кажутся утерянными, см. [Операции отмены](https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8-%D0%BE%D1%82%D0%BC%D0%B5%D0%BD%D1%8B#r_undoing).

### Три состояния

Теперь слушайте внимательно. Это самая важная вещь, которую нужно запомнить о Git, если вы хотите, чтобы остаток процесса обучения прошёл гладко. У Git есть три основных состояния, в которых могут находиться ваши файлы: изменённое (modified), подготовленное (staged) и зафиксированное (committed), .

К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

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

Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Если продолжить аналогию с фотографиями. Представим, что у вас есть маленькие котята, и вы хотите иметь возможность вернуться в этот момент времени из будущего

Вы пройдете три стадии:  
  
Ваши котята постоянно куда-то разбегаются - это изменения (modified)  
  
Вы находите всех котят и красиво усаживаете в корзинку - подготавливаете для снимка (staged)  
  
Вы щелкаете камерой, делаете снимок и навсегда фиксируете изменения (commited)  
  
--------------  
  
Вы носите изменения в программу, ваши файлы становятся измененными (modified)

Вы указываете, какие изменения нужно будет зафиксировать в снимке (staged)

Вы делаете снимок (commit)

--------------

Как работать с гит

Для начала нужно установить его на локальную машину, можете сделать это по видео:

<https://youtu.be/LjK7qF-MXoQ>   
  
Мы будем работать с гит из командной строки, поэтому для начала давайте немного разберемся с командной строкой.   
  
Командная строка - это текстовый интерфейс работы с системой. Если вы привыкли работать с Проводником (графичесиким интерфейсом), то работать с командной строкой будет немного непривычно поначалу. Но все сервера, с которыми вы будете работать в будущем, скорее всего будут иметь только консоль, так что хорошо бы привыкнуть к работе с командной строкой.

### Наиболее часто используемые команды командной строки

|  |  |  |
| --- | --- | --- |
| Windows | Linux/Unix | Описание |
| CD | cd | Вывод имени либо смена текущей папки.   * По абсолютному пути: cd C:\Users * На одну директорию назад: cd .. * В директорию внутри текущей: cd directory\_name |
| COPY | cp | Копирование одного или нескольких файлов в указанное место |
| DEL | rm | Удаление одного или нескольких файлов |
| DIR | ls | Показывает список файлов и папок в текущей или указанной папке |
| EXIT | exit | Выход из командной строки |
| MKDIR | mkdir | Создает пустой каталог с заданным именем |
| MOVE | mv | Перемещает файлы из одного места в другое |
| RMDIR | rmdir | Удаляет каталог |

В окне git bash можете использовать команды linux, даже если у вас windows или mac

Настройка гит

### Первичная настройка Git

Первое, что вам следует сделать после установки Git — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена. Это нужно сделать только один раз — при обновлении версии Git настройки сохранятся. Но, при необходимости, вы можете поменять их в любой момент, выполнив те же команды снова.

git config --global user.name "Имя автора коммитов"

git config --global user.email адрес.электронной@почты

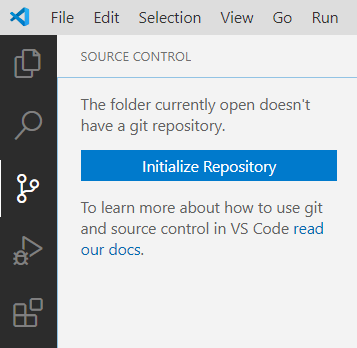
Выполнение этих команд установит глобальную конфигурацию для Git - таким образом, во всех будущих коммитах, неважно в каком проекте, Git будет автоматически добавлять эту информацию в свойство "автора" коммита. Таким образом, вы и другие разработчики всегда будут в курсе, кто отвечает за какие изменения.

Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра --global в каталоге с нужным проектом.

git config user.name "Имя автора коммитов"

git config user.email адрес.электронной@почты

### Создание Git репозитория через IDEA



### Создание Git репозитория из командной строки

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в директорию проекта и в командной строке ввести:

cd my-project-folder

git init   
  
В консоли увидим:  
  
$ git init

Initialized empty Git repository in D:/attractor/month3/19/prep/.git/

Чтобы посмотреть, что изменилось, можем ввести команду git status, ‘та команда ничего не добавляет и не изменяет, просто показывает нам текущее состояние системы:  
  
$ git status

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

• On branch master - говорит нам, что мы сейчас находимся на "ветке" master. Наконец, команда сообщает вам на какой ветке вы находитесь и сообщает вам, что она не синхронизирована с веткой на удаленном сервере (если он есть). Пока что это всегда ветка master, ветка по умолчанию; Ветки - это способ git создавать так называемые "разветвления" проекта, когда у ваc будет храниться несколько разных версий проекта в одном репозитории.

• nothing to commit, working directory clean - Это означает, что у нас чистый рабочий каталог, другими словами – в нем нет отслеживаемых измененных файлов.

Теперь мы можем создавать и изменять и фиксировать изменения в проекте. Создадим файл index.html и добавим в него стартовую разметку (любым способом, из консоли или из IDE). После этого посмотрим статус

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

index.html

nothing added to commit but untracked files present (use "git add" to track)

Мы все еще в ветке master, но уже есть файл, который мы не добавили поду управление гита. Когда мы добавим его, гит начнет сделать за изменениями в файле, и можно будет сделать его снимок.   
  
$ git add index.html

warning: CRLF will be replaced by LF in index.html.

The file will have its original line endings in your working directory

Сообщение предупреждает, что окончания строк изменятся, это нормальное сообщение.

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: index.html

Посмотрите сейчас - гит готов к коммиту, к первому снимку. Простейший способ зафиксировать изменения — это набрать:

$ git commit -m "сообщение к коммиту"

Выполним команду, и всё, вы создали свой первый коммит! Вы можете видеть, что коммит вывел вам немного информации о себе: на какую ветку вы выполнили коммит (master), какая контрольная сумма SHA-1 у этого коммита, сколько файлов было изменено, а также статистику по добавленным/удалённым строкам в этом коммите.

Посмотреть историю коммитов можно командой git log

$ git log