Chapter 1

Введение

Эта глава о том, как начать работу с Git'ом. Сначала мы расскажем о происхождении инструментов для контроля версий, затем — как установить и запустить Git на вашей машине и наконец, как настроить его так, чтоб можно было приступить к работе. К концу этой главы вы будете понимать, зачем Git вообще сделан, почему вам стоит им пользоваться, и будете готовы начать с ним работать.

# .1 Введение - О контроле версий

## О контроле версий

Что такое контроль версий, и зачем он вам нужен? Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

### [Локальные системы контроля версий](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9#%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9)

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог (как правило добавляя текущую дату к названию каталога). Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов (см. рисунок 1-1).



Рисунок 1-1. Схема локальной СКВ.

Одной из наиболее популярных СКВ такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (патч — файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.

### [Централизованные системы контроля версий](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9#%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D0%B8%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9)

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий (ЦСКВ). В таких системах, например CVS, Subversion и Perforce, есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий (см. рис. 1-2).



Рисунок 1-2. Схема централизованного контроля версий.

Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

Однако при таком подходе есть и несколько серьёзных недостатков. Наиболее очевидный — централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно всё — всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

### [Распределённые системы контроля версий](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9#%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9)

И в этой ситуации в игру вступают распределённые системы контроля версий (РСКВ). В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных (см. рисунок 1-3).



Рисунок 1-3. Схема распределённой системы контроля версий.

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

# .2 Введение - Краткая история Git

## Краткая история Git

Как и многие замечательные вещи, Git начинался с, в некотором роде, разрушения во имя созидания и жарких споров. Ядро Linux — действительно очень большой открытый проект. Бо́льшую часть существования ядра Linux (1991-2002) изменения к нему распространялись в виде патчей и заархивированных файлов. В 2002 году проект перешёл на проприетарную РСКВ BitKeeper.

В 2005 году отношения между сообществом разработчиков ядра Linux и компанией, разрабатывавшей BitKeeper, испортились, и право бесплатного пользования продуктом было отменено. Это подтолкнуло разработчиков Linux (и в частности Линуса Торвальдса, создателя Linux) разработать собственную систему, основываясь на опыте, полученном за время использования BitKeeper. Основные требования к новой системе были следующими:

* Скорость
* Простота дизайна
* Поддержка нелинейной разработки (тысячи параллельных веток)
* Полная распределённость
* Возможность эффективной работы с такими большими проектами, как ядро Linux (как по скорости, так и по размеру данных)

С момента рождения в 2005 году Git развивался и эволюционировал, становясь проще и удобнее в использовании, сохраняя при этом свои первоначальные качества. Он невероятно быстр, очень эффективен для больших проектов, а также обладает превосходной системой ветвления для нелинейной разработки (см. главу 3).

# .3 Введение - Основы Git

## Основы Git

Так что же такое Git в двух словах? Эту часть важно усвоить, поскольку если вы поймёте, что такое Git, и каковы принципы его работы, вам будет гораздо проще пользоваться им эффективно. Изучая Git, постарайтесь освободиться от всего, что вы знали о других СКВ, таких как Subversion или Perforce. В Git'е совсем не такие понятия об информации и работе с ней как в других системах, хотя пользовательский интерфейс очень похож. Знание этих различий защитит вас от путаницы при использовании Git'а.

### [Слепки вместо патчей](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#%D0%A1%D0%BB%D0%B5%D0%BF%D0%BA%D0%B8-%D0%B2%D0%BC%D0%B5%D1%81%D1%82%D0%BE-%D0%BF%D0%B0%D1%82%D1%87%D0%B5%D0%B9)

Главное отличие Git'а от любых других СКВ (например, Subversion и ей подобных) — это то, как Git смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени, как показано на рисунке 1-4.



Рисунок 1-4. Другие системы хранят данные как изменения к базовой версии для каждого файла.

Git не хранит свои данные в таком виде. Вместо этого **Git считает хранимые данные набором слепков небольшой файловой системы**. Каждый раз, когда вы фиксируете текущую версию проекта, **Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент**. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл. То, как Git подходит к хранению данных, похоже на рисунок 1-5.



Рисунок 1-5. Git хранит данные как слепки состояний проекта во времени.

Это важное отличие Git'а от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ. В главе 3, коснувшись работы с ветвями в Git'е, мы узнаем, какие преимущества даёт такое понимание данных.

### [Почти все операции — локальные](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#%D0%9F%D0%BE%D1%87%D1%82%D0%B8-%D0%B2%D1%81%D0%B5-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8-%E2%80%94-%D0%BB%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5)

Для совершения большинства операций в Git'е необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не нужна. Если вы пользовались централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, подумаете, что боги наделили Git неземной силой. Поскольку вся история проекта хранится локально у вас на диске, большинство операций кажутся практически мгновенными.

К примеру, чтобы показать историю проекта, Git'у не нужно скачивать её с сервера, он просто читает её прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у СКВ-сервера или качать с него старую версию файла и делать локальное сравнение.

Кроме того, работа локально означает, что мало чего нельзя сделать без доступа к Сети или VPN. Если вы в самолёте или в поезде и хотите немного поработать, можно спокойно делать коммиты, а затем отправить их, как только станет доступна сеть. Если вы пришли домой, а VPN-клиент не работает, всё равно можно продолжать работать. Во многих других системах это невозможно или же крайне неудобно. Например, используя Perforce, вы мало что можете сделать без соединения с сервером. Работая с Subversion и CVS, вы можете редактировать файлы, но сохранить изменения в вашу базу данных нельзя (потому что она отключена от репозитория). Вроде ничего серьёзного, но потом вы удивитесь, насколько это меняет дело.

### [Git следит за целостностью данных](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#Git-%D1%81%D0%BB%D0%B5%D0%B4%D0%B8%D1%82-%D0%B7%D0%B0-%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C%D1%8E-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

**Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла.** Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'а и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git'ом для вычисления контрольных сумм, называется **SHA-1 хешем**. **Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая в Git'е на основе содержимого файла или структуры каталога**. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Работая с Git'ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.

### [Чаще всего данные в Git только добавляются](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#%D0%A7%D0%B0%D1%89%D0%B5-%D0%B2%D1%81%D0%B5%D0%B3%D0%BE-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5-%D0%B2-Git-%D1%82%D0%BE%D0%BB%D1%8C%D0%BA%D0%BE-%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F)

Практически все действия, которые вы совершаете в Git'е, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.

Поэтому пользоваться Git'ом — удовольствие, потому что можно экспериментировать, не боясь что-то серьёзно поломать. Чтобы узнать подробнее о том, как Git хранит свои данные и как восстановить то, что кажется уже потерянным, читайте главу 9.

### [Три состояния](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#%D0%A2%D1%80%D0%B8-%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F)

Теперь внимание. Это самое важное, что нужно помнить про Git, если вы хотите, чтобы дальше изучение шло гладко. **В Git'е файлы могут находиться в одном из трёх состояний: зафиксированном, изменённом и подготовленном**. **"Зафиксированный" значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.**

Таким образом, в проектах, использующих Git, есть три части: каталог Git'а (Git directory), рабочий каталог (working directory) и область подготовленных файлов (staging area).



Рисунок 1-6. Рабочий каталог, область подготовленных файлов, каталог Git'а.

**Каталог Git'а — это место, где Git хранит метаданные и базу данных объектов вашего проекта.** Это наиболее важная часть Git'а, и именно она копируется, когда вы клонируете репозиторий с другого компьютера.

**Рабочий каталог — это извлечённая из базы копия определённой версии проекта**. Эти файлы достаются из сжатой базы данных в каталоге Git'а и помещаются на диск для того, чтобы вы их просматривали и редактировали.

**Область подготовленных файлов — это обычный файл, обычно хранящийся в каталоге Git'а, который содержит информацию о том, что должно войти в следующий коммит.** Иногда его называют индексом (index), но в последнее время становится стандартом называть его областью подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git'а выглядит примерно так:

1. **Вы вносите изменения в файлы в своём рабочем каталоге.**
2. **Подготавливаете файлы, добавляя их слепки в область подготовленных файлов.**
3. **Делаете коммит, который берёт подготовленные файлы из индекса и помещает их в каталог Git'а на постоянное хранение.**

Если рабочая версия файла совпадает с версией в каталоге Git'а, файл считается зафиксированным. Если файл изменён, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается изменённым. В главе 2 вы узнаете больше об этих трёх состояниях и как можно либо воспользоваться этим, либо пропустить стадию подготовки.

# .4 Введение - Установка Git

## Установка Git

Настало время немного ознакомиться с использованием Git'а. Первое, что вам необходимо сделать, — установить его. Есть несколько способов сделать это; два основных — установка из исходников и установка собранного пакета для вашей платформы.

### [Установка из исходников](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%B8%D0%B7-%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2)

Если есть возможность, то, как правило, лучше установить Git из исходных кодов, поскольку так вы получите самую свежую версию. Каждая новая версия Git'а обычно включает полезные улучшения пользовательского интерфейса, поэтому получение последней версии — часто лучший путь, если, конечно, вас не затрудняет установка программ из исходников. К тому же, многие дистрибутивы Linux содержат очень старые пакеты. Поэтому, если только вы не на очень свежем дистрибутиве или используете пакеты из экспериментальной ветки, установка из исходников может быть самым выигрышным решением.

Для установки Git'а вам понадобятся библиотеки, от которых он зависит: curl, zlib, openssl, expat и libiconv. Например, если в вашей системе менеджер пакетов — yum (Fedora), или apt-get (Debian, Ubuntu), можно воспользоваться следующими командами, чтобы разрешить все зависимости:

$ yum install curl-devel expat-devel gettext-devel \

openssl-devel zlib-devel

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \

libz-dev libssl-dev

Установив все необходимые библиотеки, можно идти дальше и скачать последнюю версию с сайта Git'а:

http://git-scm.com/download

Теперь скомпилируйте и установите:

$ tar -zxf git-1.7.2.2.tar.gz

$ cd git-1.7.2.2

$ make prefix=/usr/local all

$ sudo make prefix=/usr/local install

После этого вы можете скачать Git с помощью самого Git'а, чтобы получить обновления:

$ git clone git://git.kernel.org/pub/scm/git/git.git

### [Установка в Linux](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%B2-Linux)

Если вы хотите установить Git под Linux как бинарный пакет, это можно сделать, используя обычный менеджер пакетов вашего дистрибутива. Если у вас Fedora, можно воспользоваться yum'ом:

$ yum install git-core

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

$ apt-get install git

### [Установка на Mac](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BD%D0%B0-Mac)

Есть два простых способа установить Git на Mac. Самый простой — использовать графический инсталлятор Git'а, который вы можете скачать со страницы на SourceForge (см. рисунок 1-7):

http://sourceforge.net/projects/git-osx-installer/

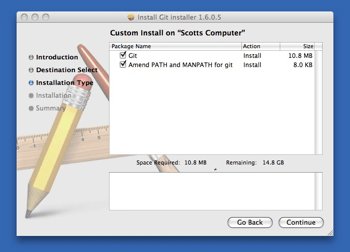


Рисунок 1-7. Инсталлятор Git'а под OS X.

Другой распространённый способ установки Git'а — через MacPorts (http://www.macports.org). Если у вас установлен MacPorts, установите Git так:

$ sudo port install git-core +svn +doc +bash\_completion +gitweb

Вам не обязательно устанавливать все дополнения, но, вероятно, вам понадобится +svn, если вы когда-нибудь захотите использовать Git вместе с репозиториями Subversion (см. главу 8).

### [Установка в Windows](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%B2-Windows)

Установить Git в Windows очень просто. У проекта msysGit процедура установки — одна из самых простых. Просто скачайте exe-файл инсталлятора со страницы проекта на GitHub'е и запустите его:

http://msysgit.github.com/

После установки у вас будет как консольная версия (включающая SSH-клиент, который пригодится позднее), так и стандартная графическая.

Пожалуйста, используйте Git только из командой оболочки, входящей в состав msysGit, потому что так вы сможете запускать сложные команды, приведённые в примерах в настоящей книге. Командная оболочка Windows использует иной синтаксис, из-за чего примеры в ней могут работать некорректно.

# .5 Введение - Первоначальная настройка Git

## Первоначальная настройка Git

Теперь, когда Git установлен в вашей системе, хотелось бы сделать кое-какие вещи, чтобы настроить среду для работы с Git'ом под себя. Это нужно сделать только один раз — при обновлении версии Git'а настройки сохранятся. Но вы можете поменять их в любой момент, выполнив те же команды снова.

В состав Git'а входит утилита git config, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы Git'а и его внешний вид. Эти параметры могут быть сохранены в трёх местах:

* Файл /etc/gitconfig содержит значения, общие для всех пользователей системы и для всех их репозиториев. Если при запуске git config указать параметр --system, то параметры будут читаться и сохраняться именно в этот файл.
* Файл ~/.gitconfig хранит настройки конкретного пользователя. Этот файл используется при указании параметра --global.
* Конфигурационный файл в каталоге Git'а (.git/config) в том репозитории, где вы находитесь в данный момент. Эти параметры действуют только для данного конкретного репозитория. Настройки на каждом следующем уровне подменяют настройки из предыдущих уровней, то есть значения в .git/config перекрывают соответствующие значения в /etc/gitconfig.

В системах семейства Windows Git ищет файл .gitconfig в каталоге $HOME (C:\Documents and Settings\$USER или C:\Users\$USER для большинства пользователей). Кроме того Git ищет файл /etc/gitconfig, но уже относительно корневого каталога MSys, который находится там, куда вы решили установить Git, когда запускали инсталлятор.

### [Имя пользователя](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-Git#%D0%98%D0%BC%D1%8F-%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F)

Первое, что вам следует сделать после установки Git'а, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'е содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

Повторюсь, что, если указана опция --global, то эти настройки достаточно сделать только один раз, поскольку в этом случае Git будет использовать эти данные для всего, что вы делаете в этой системе. 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.

### [Выбор редактора](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-Git#%D0%92%D1%8B%D0%B1%D0%BE%D1%80-%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B0)

Вы указали своё имя, и теперь можно выбрать текстовый редактор, который будет использоваться, если будет нужно набрать сообщение в Git'е. По умолчанию Git использует стандартный редактор вашей системы, обычно это Vi или Vim. Если вы хотите использовать другой текстовый редактор, например, Emacs, можно сделать следующее:

$ git config --global core.editor emacs

### [Утилита сравнения](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-Git#%D0%A3%D1%82%D0%B8%D0%BB%D0%B8%D1%82%D0%B0-%D1%81%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F)

Другая полезная настройка, которая может понадобиться — встроенная diff-утилита, которая будет использоваться для разрешения конфликтов слияния. Например, если вы хотите использовать vimdiff:

$ git config --global merge.tool vimdiff

Git умеет делать слияния при помощи kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge и opendiff, но вы можете настроить и другую утилиту. Подробнее об этом написано в главе 7.

### [Проверка настроек](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-Git#%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%BA%D0%B0-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BA)

Если вы хотите проверить используемые настройки, можете использовать команду git config --list, чтобы показать все, которые Git найдёт:

$ git config --list

user.name=Scott Chacon

user.email=schacon@gmail.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

Некоторые ключи (названия) настроек могут появиться несколько раз, потому что Git читает один и тот же ключ из разных файлов (например из /etc/gitconfig и ~/.gitconfig). В этом случае Git использует последнее значение для каждого ключа.

Также вы можете проверить значение конкретного ключа, выполнив git config {ключ}:

$ git config user.name

Scott Chacon

# .6 Введение - Как получить помощь?

## Как получить помощь?

Если вам нужна помощь при использовании Git'а, есть три способа открыть страницу руководства по любой команде Git'а:

$ git help <команда>

$ git <команда> --help

$ man git-<команда>

Например, так можно открыть руководство по команде config:

$ git help config

Эти команды хороши тем, что ими можно пользоваться всегда, даже без подключения к сети. Если руководства и этой книги недостаточно и вам нужна персональная помощь, вы можете попытаться поискать её на каналах #git и #github IRC-сервера Freenode (irc.freenode.net). Обычно там сотни людей, отлично знающих Git, которые могут помочь.

# Chapter 2

# Основы Git

Если вы хотите начать работать с Git'ом, прочитав всего одну главу, то эта глава — то, что вам нужно. Здесь рассмотрены все базовые команды, необходимые вам для решения подавляющего большинства задач возникающих при работе с Git'ом. После прочтения этой главы вы научитесь настраивать и инициализировать репозиторий, начинать и прекращать версионный контроль файлов, а также подготавливать и фиксировать изменения. Мы также продемонстрируем вам как настроить в Git'е игнорирование отдельных файлов или их групп, как быстро и просто отменить ошибочные изменения, как просмотреть историю вашего проекта и изменения между отдельными коммитами (commit), а также как отправлять (push) и получать (pull) изменения в/из удалённого (remote) репозитория.

# .1 Основы Git - Создание Git-репозитория

## Создание Git-репозитория

Для создания Git-репозитория существуют два основных подхода. Первый подход — импорт в Git уже существующего проекта или каталога. Второй — клонирование уже существующего репозитория с сервера.

### [Создание репозитория в существующем каталоге](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-Git-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F#%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F-%D0%B2-%D1%81%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D1%83%D1%8E%D1%89%D0%B5%D0%BC-%D0%BA%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%D0%B5)

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

$ git init

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем. (В главе 9 приведено подробное описание файлов содержащихся в только что созданном вами каталоге .git)

Если вы хотите добавить под версионный контроль существующие файлы (в отличие от пустого каталога), вам стоит проиндексировать эти файлы и осуществить первую фиксацию изменений. Осуществить это вы можете с помощью нескольких команд git add указывающих индексируемые файлы, а затем commit:

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

Мы разберём, что делают эти команды чуть позже. На данном этапе, у вас есть Git-репозиторий с добавленными файлами и начальным коммитом.

### [Клонирование существующего репозитория](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-Git-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F#%D0%9A%D0%BB%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D1%83%D1%8E%D1%89%D0%B5%D0%B3%D0%BE-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F)

Если вы желаете получить копию существующего репозитория Git, например, проекта, в котором вы хотите поучаствовать, то вам нужна команда git clone. Если вы знакомы с другими системами контроля версий, такими как Subversion, то заметите, что команда называется clone, а не checkout. Это важное отличие — Git получает копию практически всех данных, что есть на сервере. Каждая версия каждого файла из истории проекта забирается (pulled) с сервера, когда вы выполняете git clone. Фактически, если серверный диск выйдет из строя, вы можете использовать любой из клонов на любом из клиентов, для того чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования (вы можете потерять часть серверных перехватчиков (server-side hooks) и т.п., но все данные, помещённые под версионный контроль, будут сохранены, подробнее см. в главе 4).

Клонирование репозитория осуществляется командой git clone [url]. Например, если вы хотите клонировать библиотеку Ruby Git, известную как Grit, вы можете сделать это следующим образом:

$ git clone git://github.com/schacon/grit.git

Эта команда создаёт каталог с именем grit, инициализирует в нём каталог .git, скачивает все данные для этого репозитория и создаёт (checks out) рабочую копию последней версии. Если вы зайдёте в новый каталог grit, вы увидите в нём проектные файлы, пригодные для работы и использования. Если вы хотите клонировать репозиторий в каталог, отличный от grit, можно это указать в следующем параметре командной строки:

$ git clone git://github.com/schacon/grit.git mygrit

Эта команда делает всё то же самое, что и предыдущая, только результирующий каталог будет назван mygrit.

В Git'е реализовано несколько транспортных протоколов, которые вы можете использовать. В предыдущем примере использовался протокол git://, вы также можете встретить http(s)://или user@server:/path.git, использующий протокол передачи SSH. В главе 4 мы познакомимся со всеми доступными вариантами конфигурации сервера для обеспечения доступа к вашему Git-репозиторию, а также рассмотрим их достоинства и недостатки.