Лекция 1. Введение

Лекция 1. Введение

Введение

Эта глава о том, как начать работу с Git. Вначале изучим основы инструментария системы контроля версий, затем перейдём к тому, как запустить Git на вашей ОС и окончательно настроить для работы. В конце главы вы уже будете знать, что такое Git и почему им следует пользоваться, а также получите окончательно настроенную для работы систему.

О системе контроля версий

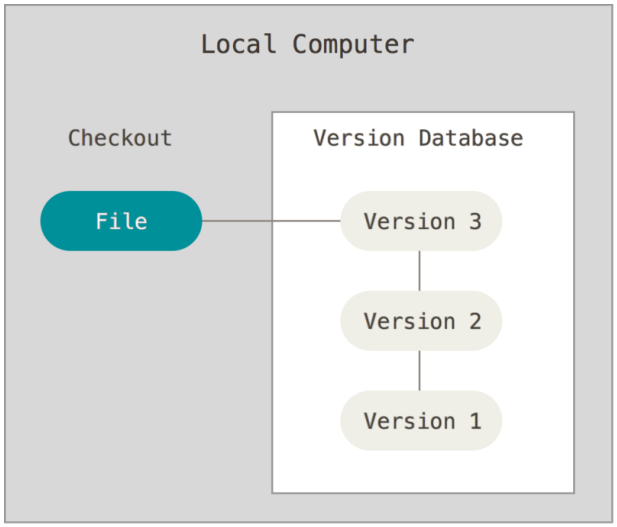
Что такое “система контроля версий”, и почему это важно? Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение большого периода времени, так что вы сможете позже вернуться к определенной версии. Для контроля версий файлов в этой книге, в качестве примера, будет 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 спровоцировал проблему, кто поставил задачу и когда, и многое другое. Использование VCS 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 накладок.

Локальные системы контроля версий

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже директорию с отметкой по времени, если они достаточно умны). Данный подход очень распространён из-за его простоты, однако он, невероятным образом, подвержен появлению ошибок. Можно легко забыть в какой директории вы находитесь и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

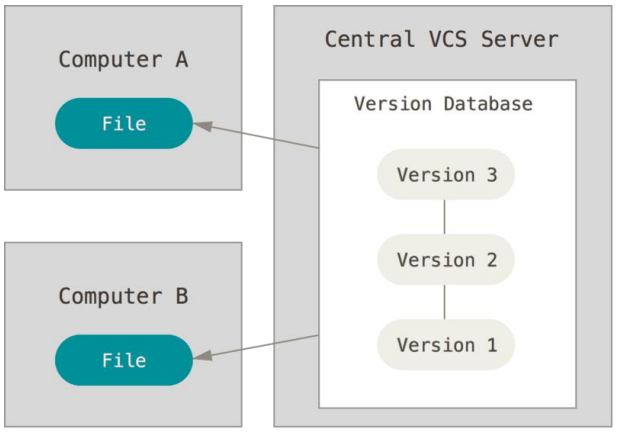


Локальный контроль версий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. Даже популярная операционная система Mac OS X предоставляет команду rcs, после установки Developer Tools. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди - это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как: CVS, Subversion и Perforce, имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.



Централизованный контроль версий.

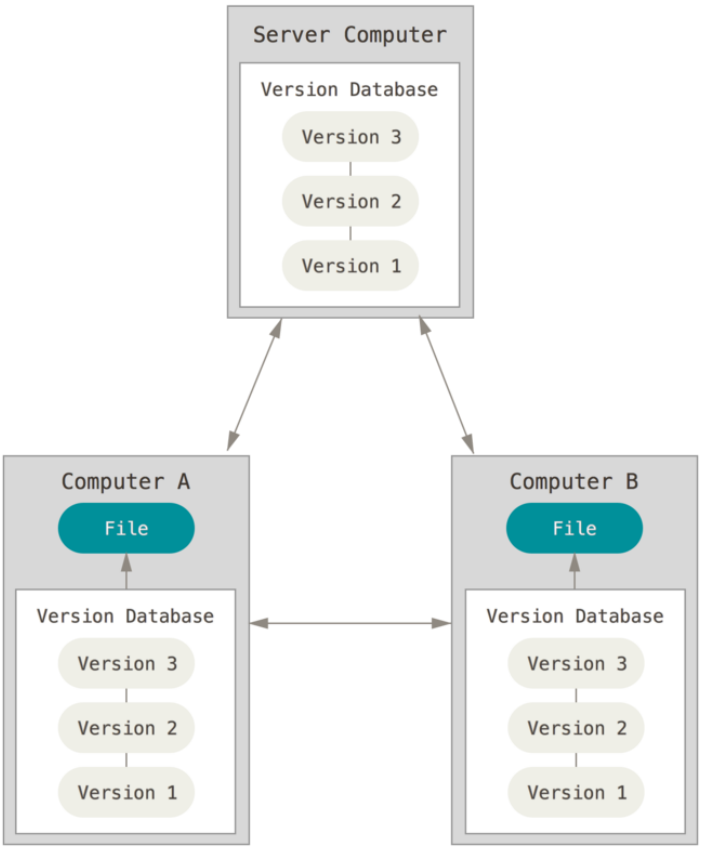
Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта, в определённой степени, знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо

проще, администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус - это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений над которыми он работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё - всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы - когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

Децентрализованные системы контроля версий

Здесь в игру вступают децентрализованные системы контроля версий (ДСКВ). В ДСКВ (таких как Git, Mercurial, Bazaar или Darcs), клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени): они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.



Децентрализованный контроль версий.

Более того, многие ДСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно, в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

Краткая история Git

Как и многие вещи в жизни, Git начинался с капелькой творческого хаоса и бурных споров.

Ядро Linux - это достаточно большой проект с открытым исходным кодом. Большую часть времени разработки ядра Linux (1991-2002 гг.), изменения передавались между разработчиками в виде патчей и архивов. В 2002 году проект ядра Linux начал использовать проприетарную децентрализованную СКВ BitKeeper.

В 2005 году отношения между сообществом разработчиков ядра Linux и коммерческой компанией, которая разрабатывала BitKeeper, прекратились, и бесплатное использование утилиты стало невозможным. Это сподвигло сообщество разработчиков ядра Linux (а в частности Линуса Торвальдса - создателя Linux) разработать свою собственную утилиту, учитывая уроки, полученные при работе с BitKeeper. Некоторыми целями, которые преследовала новая система, были:

- Скорость
- Простая архитектура
- Хорошая поддержка нелинейной разработки (тысячи параллельных веток)
- Полная децентрализация
- Возможность эффективного управления большими проектами, такими как ядро Linux (скорость работы и разумное использование дискового пространства)

С момента своего появления в 2005 году, Git развился в простую в использовании систему, сохранив при этом свои изначальные качества. Он удивительно быстр, эффективен в работе с большими проектами и имеет великолепную систему веток для нелинейной разработки (См. Chapter 3).

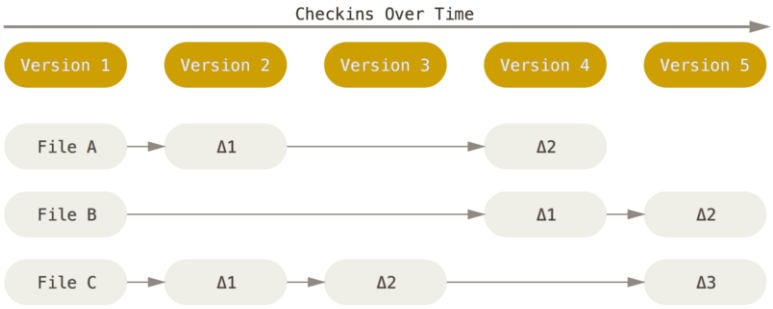
Основы Git

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что если вы поймёте что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать. Пока вы изучаете Git, попробуйте забыть всё что вы знаете о других СКВ, таких как Subversion и Perforce; это позволит вам избежать определенных проблем при использовании утилиты. Git хранит и использует информацию совсем иначе по

сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож, и понимание этих различий поможет вам избежать путаницы во время использования.

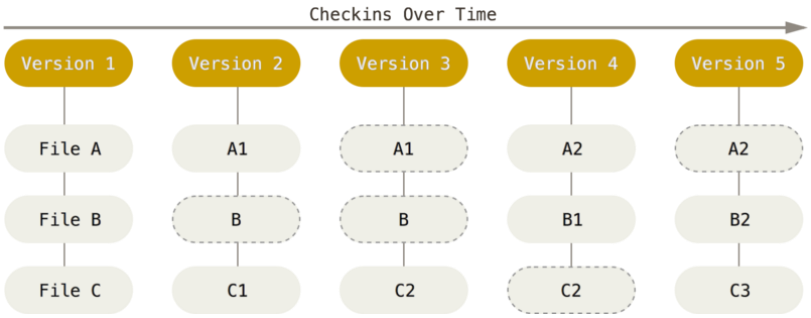
Снимки, а не различия

Основное отличие Git'а от любой другой СКВ (Subversion и друзья включительно), это подход Git'а к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени.



Хранение данных, как набора изменений относительно первоначальной версии каждого из файлов.

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git'а 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'е, 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.**



Хранение данных, как снимков проекта во времени.

Это очень важное отличие между Git и почти любой другой СКВ. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем. Это делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, надстроенными над ней, нежели просто на СКВ. Когда мы будем рассматривать управление ветками в Chapter 3, мы увидим какие преимущества вносит такой подход к работе с данными в Git.

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. Если вы привыкли к ЦСКВ, где большинство операций имеют задержку из-за работы с сетью, то этот аспект Git'а заставит вас думать, что боги скорости наделили Git несказанной мощью. 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'у 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 выполнить эту операцию, либо вместо получения старой версии файла с сервера и выполнения операции локально.

Это также означает, что есть лишь небольшое количество действий, которые вы не сможете выполнить если вы находитесь оффлайн или не имеете доступа к ВПН в данный момент. Если вы в самолёте или в поезде и хотите немного поработать, вы сможете создавать коммиты без каких-либо проблем, когда будет возможность подключиться к сети, все изменения можно будет синхронизировать. Если вы ушли домой и не можете подключиться через виртуальную частную сеть (ВЧС), вы всё равно сможете работать. Добиться такого же поведения во многих других системах либо очень сложно, либо вовсе невозможно. В Perforce, для примера, если вы не подключены к серверу, вам не удастся сделать многого; в Subversion и CVS вы можете редактировать файлы, но вы не сможете сохранить изменения в базу данных (потому что вы не подключены к БД). Всё это может показаться не таким уж и значимым, но вы удивитесь, какое большое значение это может иметь.

Целостность Git

В Git'е 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. 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.

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’е, 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 только добавляет данные

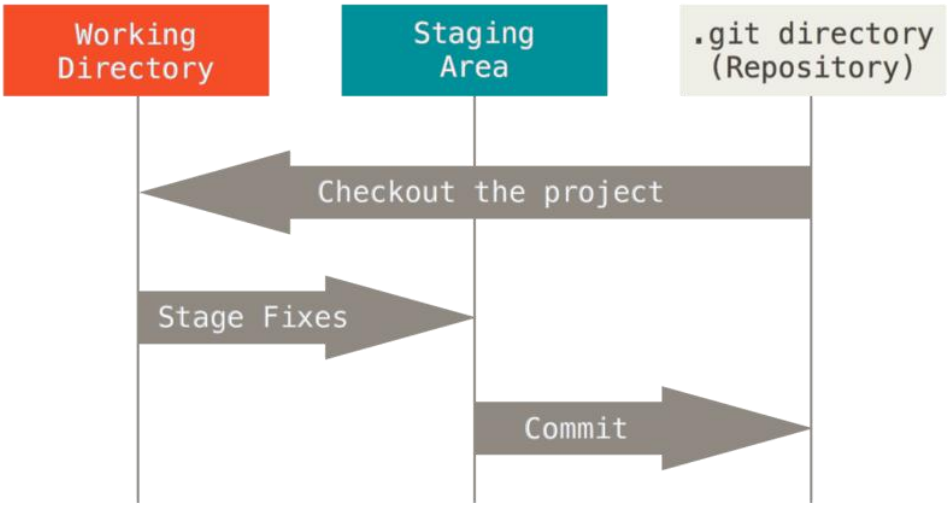
Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные, либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не закоммичены, но после того, как вы закоммитите 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. “Undoing Things”.

Три состояния

Теперь слушайте внимательно. Это самая важная вещь, которую нужно запомнить о Git, если вы хотите, чтобы остаток процесса обучения прошёл гладко. Git имеет три основных состояния, в которых могут находиться ваши файлы: committed, modified и staged. Committed означает, что данные сохранены в локальной базе и находятся в безопасности. Modified значит, что вы внесли изменения в файл, но ещё не закоммитили их в свою базу. Staged указывает, что вы пометили модифицированный файл, в его текущей версии, для следующего коммита.

Мы подошли к трём основным секциям проекта Git: Git директория, рабочая директория и stage область.



Рабочая директория, stage область и директория Git.

Git директория - 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.

Рабочая директория является снимком версии проекта. Файлы распаковываются из сжатой базы данных в Git директории и располагаются на диске, для того, чтобы их можно было изменять и использовать.

Область stage - это файл, располагающийся в вашей Git директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage область также общепринято.

Базовый подход в работе с Git выглядит так:

1. Вы модифицируете файлы в вашей рабочей директории.
2. Вы добавляете файлы в индекс, добавляя тем самым их снимки в stage область.
3. Когда вы делаете коммит, используются файлы из индекса, как есть и этот снимок сохраняется в вашу Git директорию.

Если определённая версия файла есть в Git директории, эта версия закоммичена. Если файл модифицирован и добавлен в область stage, значит он будет добавлен в следующий коммит. И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается модифицированным. В главе Chapter 2, вы узнаете больше об этих состояниях и какую пользу вы можете извлечь из них, либо 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 функционала Git. Если вы знаете, как выполнить какое-либо действие в командной строке, вы, вероятно, сможете выяснить, как то же 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'а.

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, придется ненадолго прерваться и изучить эти вопросы, чтобы вы могли понимать примеры и пояснения из этой книги.

Installing Git

Before you start using Git, you have to make it available on your computer. Even if it's already installed, it's probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself.

This book was written using Git version **2.0.0**. Though most of the commands we use should work even in ancient versions of Git, some of them might not or might act slightly differently if you're using an older version. Since Git is quite excellent at preserving backwards compatibility, any version after 2.0 should work just fine.

Installing on Linux

If you want to install Git on Linux via a binary installer, you can generally do so through the basic package-management tool that comes with your distribution. If you're on Fedora for example, you can use yum:

$ yum install git

If you're on a Debian-based distribution like Ubuntu, try apt-get:

$ apt-get install git

For more options, there are instructions for installing on several different Unix flavors on the Git website, at *http://git-scm.com/download/linux*.

Installing on Mac

There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run *git*from the Terminal the very first time. If you don't have it installed already, it will prompt you to install it.

If you want a more up to date version, you can also install it via a binary installer. An OSX Git installer is maintained and available for download at the Git website, at *http://git-scm.com/download/mac*.

Git OS X Installer.

You can also install it as part of the GitHub for Mac install. Their GUI Git tool has an option to install command line tools as well. You can download that tool from the GitHub for Mac website, at*http://mac.github.com*.

Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to *http://git-scm.com/download/win* and the download will start automatically. Note that this is a project called Git for Windows (also called msysGit), which is separate from Git itself; for more information on it, go to *http://msysgit.github.io/*.

Another easy way to get Git installed is by installing GitHub for Windows. The installer includes a command line version of Git as well as the GUI. It also works well with Powershell, and sets up solid credential caching and sane CRLF settings. We'll learn more about those things a little later, but suffice it to say they're things you want. You can download this from the GitHub for Windows website, at *http://windows.github.com*.

Installing from Source

Some people may instead find it useful to install Git from source, because you'll get the most recent version. The binary installers tend to be a bit behind, though as Git has matured in recent years, this has made less of a difference.

If you do want to install Git from source, you need to have the following libraries that Git depends on: curl, zlib, openssl, expat, and libiconv. For example, if you're on a system that has yum (such as Fedora) or apt-get (such as a Debian based system), you can use one of these commands to install all of the dependencies:

```
$ 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
```

When you have all the necessary dependencies, you can go ahead and grab the latest tagged release tarball from several places. You can get it via the Kernel.org site, at*https://www.kernel.org/pub/software/scm/git*, or the mirror on the GitHub web site, at*https://github.com/git/git/releases*. It's generally a little clearer what the latest version is on the GitHub page, but the kernel.org page also has release signatures if you want to verify your download.

Then, compile and install:

```
$ tar -zxf git-1.9.1.tar.gz
$ cd git-1.9.1
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```

After this is done, you can also get Git via Git itself for updates:

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

Первоначальная настройка Git

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

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

1. 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.
2. Файл ~/.gitconfig или ~/.config/git/config хранит настройки конкретного пользователя. Этот файл используется при указании параметра --global.

3. Файл config в каталоге Git’a (т.е. .git/config) в том репозитории, который вы используете в данный момент, хранит настройки 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:\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

Первое, что вам следует сделать после установки Git’a, — 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
```

Опять же, 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.

Многие GUI-инструменты предлагают сделать это при первом запуске.

Выбор редактора

Теперь, когда вы указали своё имя, самое время выбрать текстовый редактор, который будет использоваться, если будет нужно набрать сообщение в Git’е. По умолчанию Git использует стандартный редактор вашей системы, которым обычно является Vim. Если вы хотите использовать другой текстовый редактор, например, Emacs, можно проделать следующее:

```
$ git config --global core.editor emacs
```

Vim и Emacs — популярные текстовые редакторы часто используется разработчиками в Unix-подобных системах, таких как Linux и Mac. Если Вы не знакомы с каким-либо из этих редакторов или работаете на Windows системе, вам вероятно потребуется инструкция по настройке используемого вами редактора для работы с Git. В случае, если вы не установили свой редактор, и не знакомы с Vim или Emacs, то можете попасть в затруднительное положение, когда они будут запущены.

Проверка настроек

Если вы хотите проверить используемую конфигурацию, можете использовать командуgit config --list, чтобы показать все настройки, которые Git найдёт:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.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 <key>:

```
$ git config user.name
John Doe
```

Как получить помощь?

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

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

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 сервера Freenode IRC (irc.freenode.net). Обычно там сотни людей, отлично знающих Git, которые могут помочь.

Заключение

Вы получили базовые знания о том, что такое Git и чем он отличается от централизованных систем контроля версий, которыми вы, возможно, пользовались. Также вы теперь получили рабочую версию Git в вашей ОС, настроенную и персонализированную. Самое время изучить основы Git.