

Знакомая картина? Если вы работаете с кодом (да и вообще с любыми файлами), то наверняка сталкивались с проблемой версий. Как сохранить стабильную версию файла? Что, если вы сохраните изменения, но вам потребуется вернуться к предыдущей? Как разобраться в этих версиях, если их очень много?

Даже если вы работаете в одиночку, то можете столкнуться с этими проблемами. Если же вы — часть команды, то эти вопросы просто необходимо как-то решать.

Поэтому придумали такую вещь, как **системы контроля версий**. Это программы разной степени сложности. Есть такие, которые просто хранят разные версии файлов у вас на компьютере. Но для совместной работы над множеством файлов с необходимостью проверки и исправления кода в огромном количестве постоянно изменяющихся файлов таких приложений недостаточно.

Так возникли более функциональные системы контроля версий. Это сложные программные решения, которые позволяют одновременно работать над файлами множеству людей, сохранять версии, откатывать изменения, делать код-ревью и многое другое.   
  
Существуют различные системы для контроля версий: *SVN*, *Mercurial*, *Perforce*, *CVS*, *Bitkeeper* и другие. Мы же будем работать с самой популярной системой для разработки ПО – *Git* (от англ. — *Global Information Tracker*).

**Git** — распределённая система хранения версий для фиксации изменений в файлах, которая представляет собой набор консольных утилит. Она разработана для координации работы между программистами, но в целом может использоваться и для других целей, которые мы рассмотрим чуть позже.

*Git* поддерживает нелинейное взаимодействие с информацией (*non-linear workflow*).

**Распределённая система — это хорошо?**

Благодаря тому, что *Git* распределённый, можно вносить изменения в код без доступа к интернету, локально, сохраняя все данные в папках на жёстком диске, которые называются **репозиторием**. Чтобы иметь доступ к репозиторию с различных устройств, можно хранить копию репозитория онлайн, что сильно облегчает работу над одним проектом для нескольких людей. Для этой цели существуют специальные сайты, такие как *GitHub*, *Bitbucket* или *GitLab*. Один из таких сервисов мы рассмотрим несколькими юнитами позже.

Виды систем контроля версий

Прежде чем мы перейдём к *Git*, рассмотрим преимущества и возможные недостатки некоторых **видов систем контроля версий**:

* локальной,
* централизованной,
* децентрализованной.

Локальные системы контроля версий

Одна из первых известных систем контроля версий, которая является локальной — это [SCCS (source code control system)](https://habr.com/ru/post/478752/). Другой популярной локальной системой контроля версий была **RCS** (*revision control system*), которая и сегодня поставляется со многими компьютерами.

Локальные системы контроля версий хранят информацию о всех изменениях, внесённых в файлы в базе данных, которая находится там же локально на компьютере. Основываясь на этих данных, система контроля версий воссоздаёт нужную версию файла (актуальную на определённый момент времени). В локальных СКВ обычно хранят список изменений, внесённых в файлы.

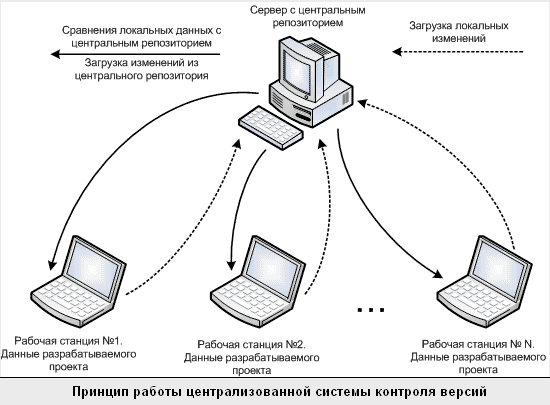
Среди **достоинств** таких систем нужно отметить:

* Возможность восстановления данных из определённой версии (точно определяется по времени записи).
* Высокая скорость восстановления (база данных чётко структурирована, сетевая задержка отсутствует, так как данные хранятся непосредственно на рабочем компьютере).

**Недостатки** таких систем очевидны:

* Возможность потери данных в случае физических поломок оборудования.
* Сложность совместной разработки. Одновременно изменения в определённый файл могут быть внесены только одним пользователем.

Централизованные системы контроля версий

Источник: [all-ht.ru](http://all-ht.ru/inf/prog/p_0_0.html)

В централизованных системах появляется **удалённый репозиторий**, который находится на сервере в Сети. Проекты, над которыми работают разработчики, импортируются в локальные репозитории. Таким образом, появляется возможность работы над одними и теми же файлами одновременно.

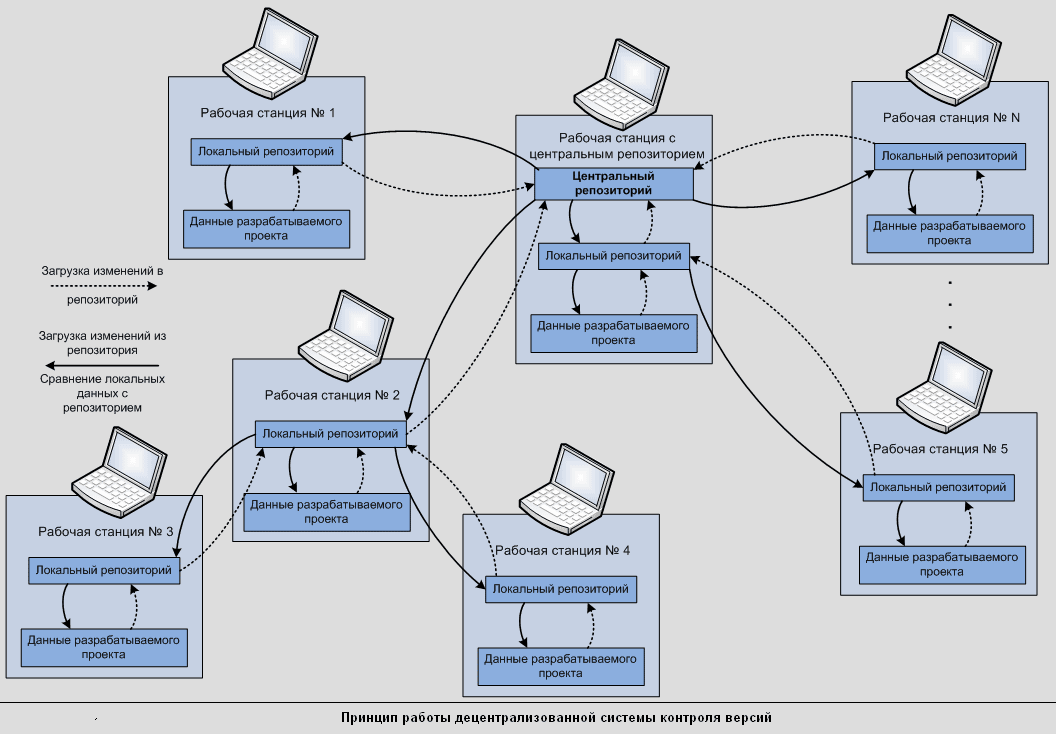
Архитектурно централизованные системы имеют существенные различия. Например, система **CVS** (*Concurrent version system*) для каждого файла хранит свой файл истории, подобно *RCS*, хотя использует при этом сжатие.

В системе **SVN** (*Subversion*) используется база данных, хранящая информацию об изменениях файлов и папок. Также в этой системе можно отслеживать наборы связанных изменений в разных файлах вместе, как в сгруппированном блоке, а не отдельно для каждого файла, как в локальных системах и в *CVS*.

Основное **преимущество централизованных систем** — возможность распределённой разработки и полный контроль администраторов над тем, кто и что может делать. Администрировать централизованную СКВ гораздо проще.

**Недостаток централизованных систем** такой же, как и у локальных — единая точка отказа, которой является центральный сервер. Если этот сервер выйдет из строя, то, пока он недоступен, никто не сможет ни взаимодействовать друг с другом, ни сохранить изменения в файле, с которым он работает. Если жёсткий диск, где хранится центральная база данных, окажется повреждён, а резервных копий не будет, то вся история проекта (за исключением единичных снимков репозитория, которые сохранились на локальных машинах разработчиков) будет потеряна.

Децентрализованные системы контроля версий

Источник: [all-ht.ru](http://all-ht.ru/inf/prog/p_0_0.html)

Герой этого модуля *Git* — это представитель децентрализованной СКВ. То есть центрального репозитория не существует: все копии создаются равными, что резко отличается от централизованных СКВ, где работа основана на добавлении и извлечении файлов из центрального репозитория. Это означает, что разработчики могут обмениваться изменениями друг с другом непосредственно перед объединением своих изменений в официальную ветвь.

Разработчики могут вносить свои изменения в локальную копию репозитория без ведома других репозиториев. Это допускает фиксацию изменений без подключения к сети или интернету. Разработчики могут работать локально в автономном режиме, пока не будут готовы поделиться своей работой с другими. В этот момент изменения отправляются в другие репозитории для проверки, тестирования и развёртывания.

Отличие Git от других СКВ

1

Главное отличие *Git* от централизованных и локальных СКВ: система делает снимок того, как выглядит каждый файл в данный момент, и сохраняет ссылку на этот снимок каждый раз, когда пользователь делает фиксацию изменений в своём проекте. Если файл не менялся, то *Git* не запоминает этот файл вновь, а просто сохраняет ссылку на предыдущую версию этого файла, который был уже сохранён.

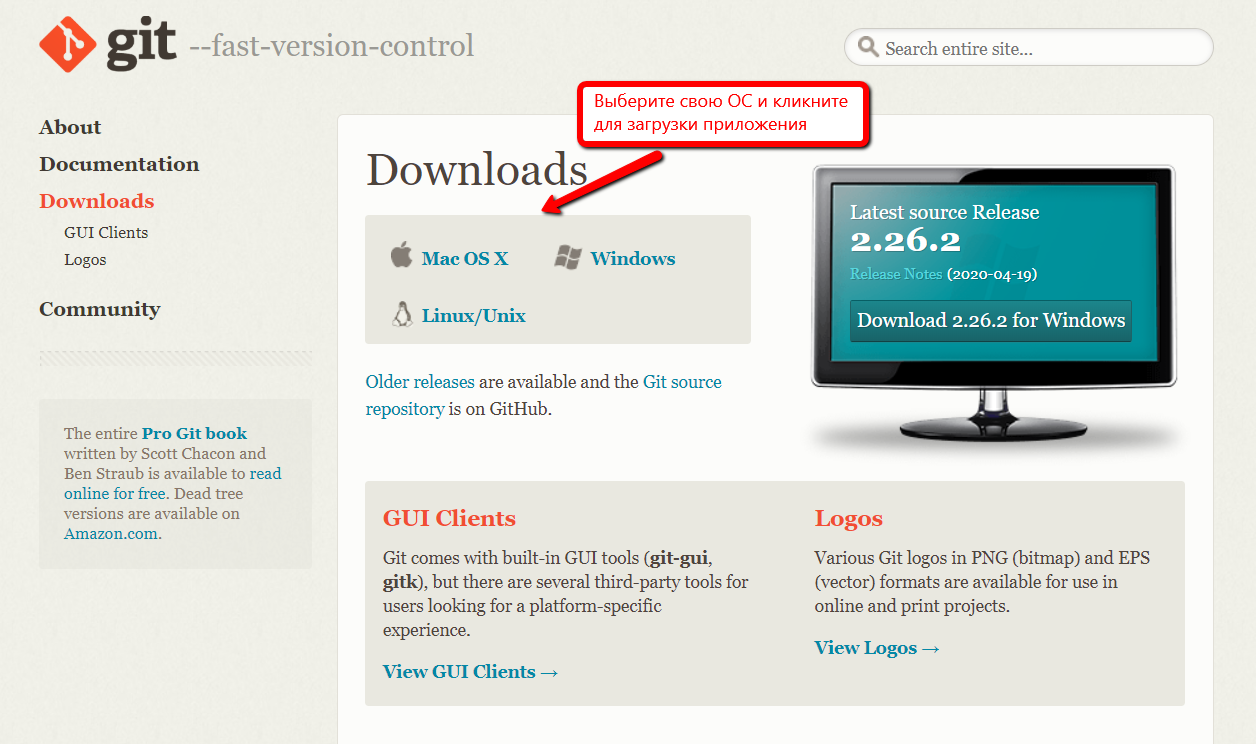
2

Другое очень важное отличие *Git* от других типов СКВ — это **автономность**. Большинство операций выполняется локально, и подключение к удалённому центральному серверу необязательно. Копия репозитория хранится у вас локально, можно очень быстро получить любую версию любого файла. Если изменения зафиксированы и синхронизированы с репозиторием на сервере, то потерять данные достаточно сложно, в особенности если над проектом работает целая команда.

Установка Git

Пришло время установить *Git* на компьютер. В зависимости от вашей операционной системы алгоритм может отличаться.

Перейдите [по ссылке](https://git-scm.com/), откройте раздел *Downloads* (Загрузки), выберите операционную систему вашего компьютера и кликните соответствующую ссылку для загрузки приложения.

Загрузка *Git*

После загрузки приложения установите его. Для пользователя *Windows* установщик последовательно запросит:

* Разрешение изменений на компьютере, согласие с положениями лицензии.
* Каталог для установки приложения, параметры установки, определяемые чекбоксами, название папки приложения в стартовом меню, редактор, который будет использован для редактирования комментариев, использование компонентов приложения (*Git GUI* и *Git Bash* — о них позднее), библиотеки *SSL*/*TLS*, опции преобразования окончания строки, специфичные для операционной системы, и дополнительные опции.

На все эти запросы можно просто выбрать значения по умолчанию.

После установки *git-scm* в каталоге установки появится папка с именем *Git* (если это имя было выбрано при установке). В этой папке будут ссылки для запуска 3-х приложений:

* *Git GUI* — это приложение с графическим интерфейсом для работы с *Git*.
* *Git CMD* — командная строка *Git*, использующая интерпретатор командной строки *Windows*.
* *Git Bash* — командная строка *Git*, использующая эмулятор интерпретатора командной строки *Bash* для *Linux*.

Установка на Windows

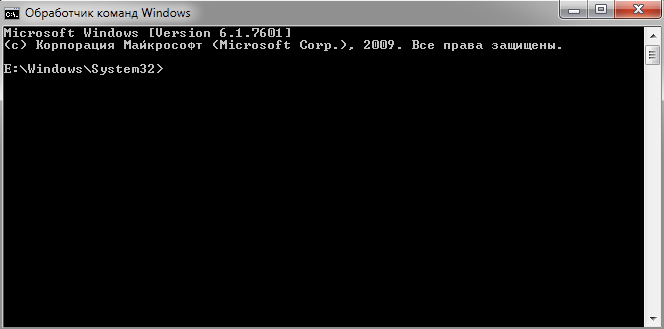
Для начала работы будет полезно установить и клиентское приложение с графическим интерфейсом, например, [Github Desktop](https://desktop.github.com/" \t "_blank) или [SourceTree](https://www.sourcetreeapp.com/).

Базовые настройки пользователя

**У Git нет графической версии.** Поэтому работа с *Git* ведётся через терминал. Также его называют консолью или командной строкой. Обсуждение на тему того, в чём же между ними разница, можете прочитать [здесь](https://toster.ru/q/273105).

**Терминал** — текстовый интерфейс. Чтобы им пользоваться, нужно знать специальные команды, которые надо будет вводить с клавиатуры.

Терминал есть во всех операционных системах. Так как на данный момент самой популярной является *Windows*, все примеры будут проиллюстрированы на ней.



После того как все действия по установке завершены, давайте проверим, что *Git* появился в системе компьютера. Откройте терминал и введите git --version. На экране терминала должна появиться текущая версия программы на вашей машине. Эта проверка подходит для всех операционных систем.

**blahblahs-MacBook-Air:~ blahblah$ git --version  
git version 2.26.2**

**Git** — довольно гибкая система, и в ней есть множество настроек. Но мы начнём с необходимых. Для начала установим имя пользователя и адрес электронной почты. Эти данные будут отображены в каждом изменении, которое вы вносите, и другие пользователи смогут понять, кто автор тех или иных изменений.

В терминале запустите по очереди команды:

git config --global user.name "Name"

git config --global user.email email@example.ru

**Обратите внимание**, что в этих командах есть опция --global. Это значит, что вам достаточно будет один раз ввести эти данные и они будут сохранены для всех проектов. Если вы захотите поменять информацию, то нужно будет ввести ту же команду, только без  --global.

Некоторые полезные команды терминала

Для того чтобы пользоваться *Git*, очень полезно освоить базовые команды терминала [*UNIX*-систем](https://www.dmosk.ru/terminus.php?object=unix):

|  |  |
| --- | --- |
| **Название команды** | **Действие** |
| pwd | показать текущий каталог |
| ls | показать файлы в данной папке, кроме скрытых |
| ls -f | показать файлы в данной папке, включая скрытые |
| cd c: | перейти к диску (в примере диск *C*) |
| cd app | перейти в конкретный каталог (в примере папка *app*) |
| cd - | вернуться назад |
| cd .. | подняться на один уровень вверх |
| cd ../.. | подняться на два уровня вверх |
| mkdir | создать папку |

Если вы разобрались с терминалом, то можете попробовать поочередно вписать команды выше и посмотреть на результат, который будет выведен на экране.

Для ОС *Windows* ситуация несколько сложнее. Чтобы разобраться с терминалом, вы можете установить в качестве второй ОС *Linux*, например, [Ubuntu](https://ubuntu.com). Также вы можете установить терминал для *Windows*.

Несколько полезных ссылок для пользователей Windows:

* [Introducing Windows Terminal](https://devblogs.microsoft.com/commandline/introducing-windows-terminal/),
* [Windows Command Line blog](https://devblogs.microsoft.com/commandline/),
* [Лучшие бесплатные терминалы для Windows](https://techrocks.ru/2019/08/14/best-free-terminals-for-windows/).

Со списком соответствия команд в *UNIX*-системах и OC *Windows* вы можете ознакомиться [здесь](https://white55.ru/cmd-sh.html).

Есть некоторые особенности работы с терминалом, зависящие от операционной системы, но общие принципы одинаковые. Ниже приведён скринкаст, иллюстрирующий работу с терминалом в ОС *Windows*.

Создание локального репозитория

Удобнее всего иметь на компьютере папку в корневой директории и в ней вести все свои проекты. Такую папку можно назвать, например, *projects* или *dev* (от слова *development* — «разработка»).

Как создать репозиторий:

|  |  |
| --- | --- |
| 1 | **mkdir projects**  Создать новую папку в корневом каталоге (терминал открывается в нём по умолчанию). |
| 2 | **cd projects**  Перейти внутрь *projects*. |
| 3 | **mkdir Skillfactory**  Создать новую папку *Skillfactory*. |
| 4 | **cd Skillfactory**  Перейти внутрь *Skillfactory*. |
| 5 | **git init**  Инициализировать папку *Skillfactory* как *Git*-репозиторий. |

В терминале у вас должна появиться примерно такая запись (только с вашими данными вместо *blahblah)*:

Initialized empty Git repository in /Users/blahblah/projects/Skillfactory/.git/  
  
Blahblahs-MacBook-Air:Skillfactory blahblah$

И это значит, что локальный *Git*-репозиторий создан!

Чтобы проверить статус, в котором находится ваш репозиторий, наберите в терминале:

git status

Статус репозитория показывает информацию о текущем состоянии репозитория: есть ли что-то новое в репозитории, изменились ли какие-то файлы, и если да, то какие, и так далее.

Так как пока ваш репозиторий пуст, то при проверке статуса он выдаст сообщение:

On branch master  
  
No commits yet

Другие сообщения, которые вы можете встретить при проверке статуса:

* **On branch master**. Сообщение означает, что в настоящий момент активна ветка *master*. Как правило, это основная ветка в репозитории.
* **Your branch is up to date with origin/master**. Сообщение означает, что файлы в ветке *master* синхронизированы с веткой *master* на сервере. *Origin* — это и есть удалённый сервер.
* **Working tree clean** дословно означает, что рабочий каталог не содержит изменённых файлов. Система контроля версий *Git* использует так называемую архитектуру трёх деревьев. *Working tree* связана с рабочей директорией *Git*, в которой содержатся рабочие версии файлов.
* **Nothing to commit** значит, что все изменения были зафиксированы и нет новых изменений в файлах для фиксации изменений.

**ВНИМАНИЕ:** В новых версиях *Git* ветка *master* может быть по умолчанию переименована в *main.* Посмотреть, какая ветка у вас, можно с помощью команды git branch. Если у вас ветка *main,* то работать с ней следует аналогично ветке *master.*