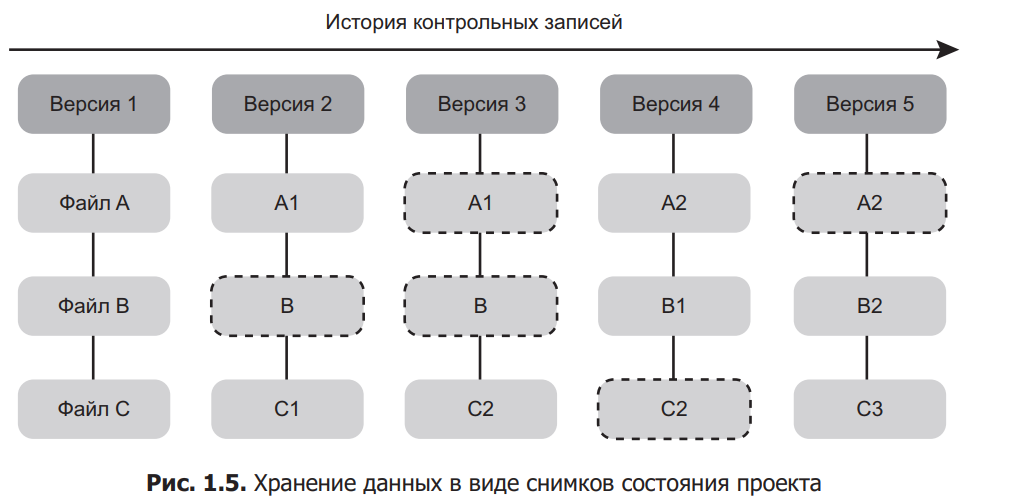
**Конспект по Git.**

Введение в Git.

Git является системой контроля версий, основное отличие Git состоит в том, что Git воспринимает поток изменений как набор снимков состояния. Каждый раз, когда вы делаете снимок или сохраняете текущее состояние проекта, Git делает снимок всех файлов вашего проекта и сохраняет ссылку на них. Для повышения продуктивности вместо файлов, которые не менялись, сохраняется всего лишь ссылка на них.



Локальность операций.

Для осуществления практически всех операций системе Git требуются только локальные файлы и папки. Поэтому, почти все операции выполняются локально. Это в разы повышает скорость работы, потому что обращение происходит только к локальном диску.

Чтобы найти изменения в каком-либо файле, Git берет, к примеру, файл месячной давности и последнюю версию с локального диска, а не отправляет запрос о подобной операции на сервер.

Целостность Git.

Перед сохранением Git вычисляет хэш сумму. Другими словами, содержимое папки или файла не может измениться без того, чтобы об этом узнала Git. Эта функция реализована на самом низком уровне, поэтому невозможно потерять или повредить файл.

Три состояния.

Файлы в Git могут находиться в 3х состояниях: *зафиксированном, модифицированном и индексированном*.

***Зафиксированное*** состояние означает, что данные надежно сохранены в локальной базе данных.

***Модифицированное*** состояние означает, что данные в файле/папке изменены, но изменения ещё не зафиксированы в базе данных.

***Индексированное*** состояние означает, что пометили текущую версию модифицированного файла как предназначенную для след. фиксации.

В результате Git-проект разделяется на 3 области *папка Git, рабочая папка и область индексирования*.



***Папка Git*** – это папка, где Git хранит метаданные и объектную базу данных проекта. Видимо, это наиболее важна часть проекта, поскольку хранит и проект, и вычисленные хэш функции. Данная папка копируется при дублировании репозитория с другого компьютера.

***Рабочая папка*** – это место куда выполняется выгрузка одной из версий проекта. Эти файлы извлекаются из папки Git и копируются на локальный ПК. Дальше все изменения происходят в них, т. е. локально.

**Область индексирования** – это файл, обычно в папке Git, в котором хранится информация о том, что именно войдет в след. операцию фиксации. Иногда её еще называют промежуточной областью.

Базовый рабочий процесс выглядит так:

1. Вы редактируете файлы в рабочей папке.
2. Вы индексируете файлы, добавляя их снимки в папку индексирования.
3. Вы выполняете фиксацию, беря файлы из папки индексирования и добавляя

их снимки в папку Git.

Файл, находящийся в папке Git, называется зафиксированным.

Установка Git для Windows.

Существуют 2 разные версии Git для Windows:

GitHub – это графическая версия Git. <http://windows.github.com>

Git – это терминальная версия с CLI. <http://windows.github.com>

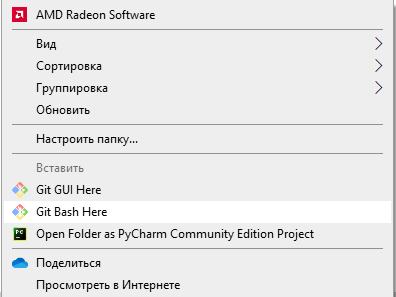
Установка – как обычно. Git встраивается в контекстное меню Windows(нажимаем правую кнопку мыши).

2 Основы Git.

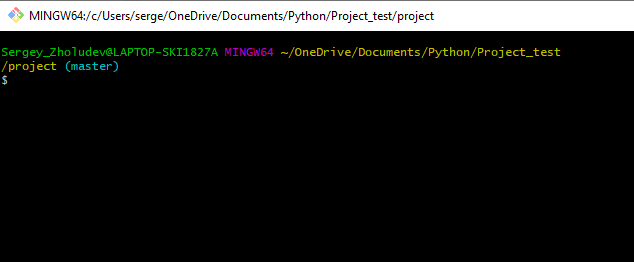
Есть 2 подхода к созданию репозитория в Git. Можно взять текущую папку и импортировать её в Git или клонировать существующий репозиторий с другого сервера.

Инициализация репозитория в существующей папке.

Чтобы начать отслеживать изменения с помощью Git, надо перейти в папку этого проекта в проводнике Windows. Нажать правую кнопку мыши – запустить Git.



Дальше откроется CLI Git Bash, где, собственно, можно вводить команды, перечисленные во всех учебниках:



Чтобы добавить конкретную папку в Git:

$ git init

В результате в папке появится еще одна папка .git. Чтобы начать, добавьте файлы и выполните первую фиксацию изменений:

$ git add \*

$ git commit -m ‘первоначальная версия проекта’

Клонирование локального репозитория на сервер Github.

Теперь вы хотите перенести изменения в свой репозиторий git, размещенный с github. Для этого вам нужно указать git , чтобы **добавить удаленное местоположение**, и вы сделаете это с помощью этой команды:

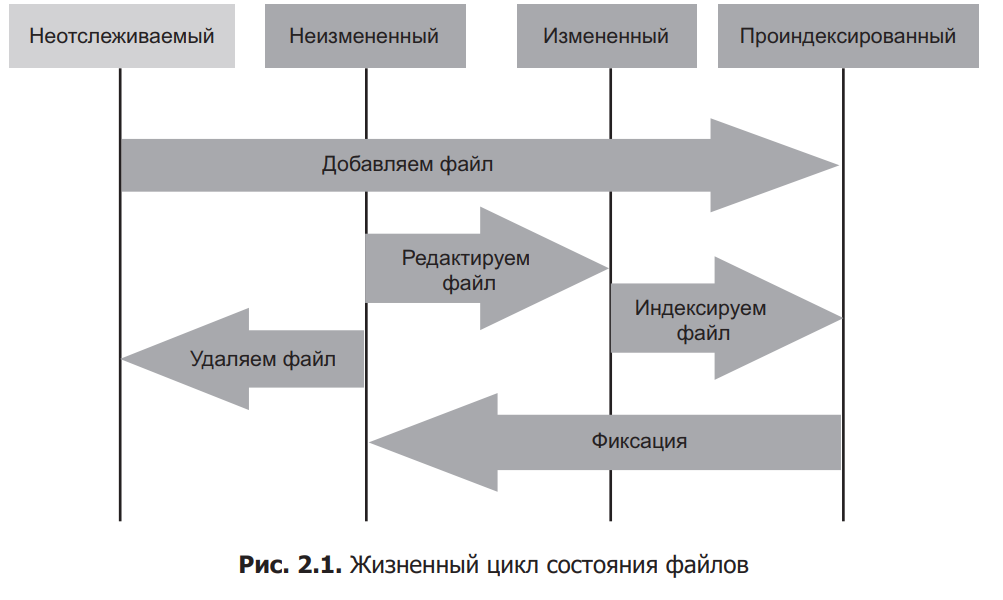
$git remote add origin https://github.com/yourusername/your-repo-name.git

\*Note: your-repo-name должно быть создано в GitHub до того, как вы сделаете git remote add origin ... После того, как вы это сделаете, git теперь знает о вашем удаленном репозитории. Затем вы можете сказать ему, чтобы он передал (что равно "upload") ваши файлы:

$git push -u origin master

Весь жизненный цикл отслеживаемого файла, могут быть и не отслеживаемые, состоит в следующем:

1. вносятся изменения в файл.
2. файл индексируются, перемещаясь в определенную папку.
3. файл фиксируются перемещаясь, опять-таки, в определенную папку.



Проверка состояния файлов.

Основным инструментов проверки состояния файлов является команда git status.

$ git status

On branch master

nothing to commit, working directory clean

Как видно из сообщения, в папке ни находится ни одного измененного файла в проиндексированном состоянии. Т.е. нечего фиксировать. В данной команде так же отражаются неотслеживаемые файлы:

Изображение выглядит как текст

Автоматически созданное описание

Слежение за новыми файлами.

Чтобы начать слежение за новыми файлами нужно использовать команду git add, к примеру для файла Readme она будет выглядеть так:

$ git add README

Все проиндексированные, а следовательно, готовые к фиксации файлы перечислены под заголовком Changes to be committed.

Индексация измененных файлов.

Если отредактировать файл или группу файлов и ввести команду $git status, то результат будет примерно таким:

Изображение выглядит как текст

Автоматически созданное описание

Другими словами, система «видит», что файл изменен, но он не добавлен в commit, т.е. не выполнено индексирование данного файла. Индексирование выполняется командой git add, данная команда многоцелевая:

1. добавляет новые файлы в проект.
2. индексирует текущие файлы.

Можно даже её рассматривать как команду добавления содержимого к след. коммиту, частный случай такого добавления – добавление файла.

Итого:

1. $git add \* - индексация файла.
2. $git commit -m ‘<комментарий>’– фиксация файла.

В конечный commit входит не то состояние файла на момент выполнения команды $git commit, а состояние файла на момент выполнения команды $git add.

Редактирование файла после выполнения команды $git add, требует заново выполнить эту команду, для индексирования изменений.

Краткий отчет о состоянии.

Для получения краткого отчёта о состоянии надо использовать команду $git status -s, рядом с именами неотслеживаемых файлов стоит **??**, новые файлы, добавленные в область предварительной подготовки, помечены буквой **A**, а модифицированные файлы буквой **M**. Так же имеет место быть цветовая индикация атрибутов файлов, т.е. M означает «модифицирован», но не проиндексирован. М – модифицирован и проиндексирован, готов к фиксации.

Игнорирование файлов.

Иногда бывает так, что не все файлы должны быть включены в проект, для этого существует механизм игнорирования файлов. Он заключается в создании файла .gitignore. Пример:

$ cat .gitignore

\*.[oa]

\*~

Первая строка заставляет Git игнорировать любые файлы начинающиеся на  *o* или *a*. Вторая строка предписывает игнорировать файлы начинающиеся на ~.

Правила паттернов, которые можно вставлять в файл .gitignore:

* Игнорируются пустые строки, начинающиеся с символа #.
* Работают стандартные глобальные паттерны.
* Паттерны можно заканчивать слешем (/), указывая папку.
* Можно инвертировать паттерн, начав его с восклицательного знака(!).

Глобальные паттерны представляют собой регулярные выражения.

* (\*) – замещает любое кол-во символов.
* [abc] – запись соответствует любом символу в скобках.
* (?) – замещает любой одиночный символ.
* [0-9] – совпадают с любым символом в данном диапазоне.
* \*\* - применяются для отображения любых вложенных папок.

Вот еще один пример файла .gitignore:  
# комментарий – это игнорируется  
\*.a # пропускать файлы, заканчивающиеся на .a  
!lib.a # но отслеживать файлы lib.a, несмотря на пропуск файлов на .a  
/TODO # игнорировать только корневой файл TODO, а не файлы вида subdir/TODO  
build/ # игнорировать все файлы в папке build/  
doc/\*.txt # игнорировать doc/notes.txt, но не doc/server/arch.txt

Как выяснилось в результате тестирования работы:

* регулярные выражения не работают.
* если добавляешь текст в файл .gitignore, как \*.txt(к примеру).

$echo \*.txt >> .gitignore(то интерпретатор «разворачивает» звёздочку во все имена файлов в одну строку и это не работает. Если файлов с таким расширением в папке ещё нет, то интерпретатор добавит «звёздочку» и будет работать как надо. Поэтому видимо рекомендуют делать файл .gitignore заранее, перед началом работы над проектом.

Правила синтаксиса

* Одна строчка - одно правило,
* Пустые строки игнорируются,
* Комментарии доступны через решётку(#) в начале строки,
* Символ "/" в начале строки указывает, что правило применяется только к файлам и папкам, которые располагаются в той же папке, что и сам файл .gitignore,
* Доступно использовать спецсимволы: звёздочка(\*) заменяет любое количество символов(ноль или больше), вопросик(?) заменяет от нуля до одного символа. Можно размещать в любом месте правила,
* Две звёздочки(\*\*) используются для указания любого количества поддиректорий, подробнее смотри ниже в примерах,
* Восклицательный знак(!) в начале строки означает инвертирование правила, необходим для указания исключений из правил игнорирования,
* Символ "\" используется для экранирования спецсимволов, например, чтобы игнорировать файл с именем "!readme!.txt", нужно написать такое правило: "\!readme!.txt",
* Для игнорирования всей директории, правило должно оканчиваться на слэш(/), в противном случае правило считается именем файла.

Просмотр индексированных и неиндексированных изменений.

Для того чтобы понять, что именно изменилось в файлах надо использовать команды $git diff, она отвечает на 2 вопроса:

1. что вы отредактировали(но пока не проиндексировали).
2. что из проиндексированного готово к фиксации.

Команда $git diff показывает конкретные строки, удаленные и добавленные. Другими словами, команда сравнивает то, что находится в рабочей папке и области индексирования. Чтобы понять, что войдет в след. commit есть команда $git diff –-staged. Она сравнивает индексированные файлы и зафиксированными.

Пропуск области индексирования.

Иногда надо пропустить шаг индексирования в процессе Git. Для этого используется конструкция:

$git commit -a -m ‘<комментарий>’

Если при работе с коммандой $git commit не использовать флаг -m, то откроется текстовый редактора указанный в глобальном файле конфигурации. Там можно написать развернутый или какой угодно комментарий.

Удаление файлов.

Чтобы система перестала отслеживать файл, его нужно удалить из списка отслеживаемых. Делается это командой:

$git rm *<имя файла>*

Если требуется убрать файл из области индексирования, он был помещен туда после команды $git add, необходимо использовать команды $git rm -f <имя файла>.

Если необходимо удалить файл из папки и, как следствие из списка отслеживания – то $rm <имя файла>. Если файл уже отслеживается, то удалить его из списка слежения можно, используя атрибут -f. Если требуется удалить файл из фиксированных, используем команду $git rm –cashed <имя файла>

Перемещение файлов.

Git не отслеживает перемещение, а так же переименование файлов. Если файл добавить в Git, затем переименовать то Git посчитает файл с предыдущим именем удаленным, файл с новым именем отметит как неотслеживаемый.

Изображение выглядит как текст

Автоматически созданное описание

Но если переименовать файл прямо из Git, то всё будет в порядке. К примеру:

Изображение выглядит как текст

Автоматически созданное описание

Просмотр истории версий.

Самый мощный инструмент $git log, можно исследовать на основе тестового проекта. Для этого предлагается его скопировать с текущего репозитория:

$git clone <https://github.com/schacon/simplegit-progit>

Репозиторий скопируется на локальный диск, использовав $git log, можно будет увидеть следующее:

Изображение выглядит как текст

Автоматически созданное описание

По умолчанию git log выводит в обратном хронологическом порядке список сохраненных версий. Т.е. первыми показываются самые свежие коммиты. У данной команды есть множество параметров:

* -p -2 – показывает разницу внесенную каждым коммитом. -2 ограничивает информацию 2мя последними коммитами.

**Таблица 2.2.** Распространенные параметры команды git log

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **-p** | Показывает изменения, внесенные в каждую версию |
| **--stat** | Показывает статистику измененных файлов в каждом коммите |
| **--shortstat** | Показывает только строку с изменениями/вставками/ удалениями от команды --stat |
| **--name-only** | Показывает список измененных файлов после информации о коммите |
| **--name-status** | Показывает список измененных файлов с информацией о добавлении/изменении/удалении |
| **--abbrev-commit** | Показывает только первые несколько символов контрольной суммы SHA-1 вместо всех 40 |
| **--relative-date** | Показывает дату не в полном, а в относительном формате (например, «2 недели назад») |
| **--graph** | Показывает ASCII-граф истории ветвлений и слияний вместе с выводом команды log |
| **--pretty** | Показывает коммиты в альтернативном формате. Возможны параметры oneline, short, full, fuller и format (с указанием вашей версии формата) |

Ограничение вывода команды log

Мы уже встречали параметры ограничения вывода команды log. Также существуют другие, как например **---since** и **--until**. Пример команды, выводящей изменения за последние 2 недели:

$git log –since=2.weeks

Параметр **--author** позволяет увидеть версии одного автора, **--grep** позволяет найти ключевые слова в коммитах. Параметр -S принимает в качестве аргумента строку. С этим параметром будут выведены только те коммиты, которые включают данную строку. Пример:

$git log –Sfunction\_name

Отмена изменений.

Необходимость отмены изменений может возникнуть на любой стадии проекта. Тут надо быть осторожным, поскольку после отмены изменений не всегда можно вернуться в предшествующее состояние, а это значит результат работы за некоторое время будет потерян. Необходимость возникает, когда забыли включить какие-либо файлы или написали неправильный коммит. Для повторного сохранения версии в такой ситуации можно использовать параметр **--amend**.

$git commit --amend

Эта команда берет область индексирования и включает в коммит всю обнаруженную там информацию. Если данные не менялись, то добавится только сообщение фиксации. Откроется тот же самый редактор, но уже с внесенным туда текстом предыдущей фиксации.

Отмена индексирования.

Допустим мы случаем проиндексировали 2 файла file\_1 и file\_2, хотя изначально хотели сделать индексацию раздельно. Чтобы отменить индексирование надо ввести команду:

$git reset HEAD <имя файла>

Отмена внесенных в файл изменений.

Как отменить те изменения, которые были зафиксированы?

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

**ВНИМАНИЕ.**

Важно понимать степень опасности команды git checkout -- [файл]. Все внесенные  
в файл изменения пропадают — вы просто копируете в этот файл содержимое другого  
файла. Никогда не пользуйтесь этой командой, если не уверены, что данный файл вам  
больше не понадобится.

**-------------------------------------------------------------------------------------------------------------**

Удаленные репозитории.

Навыки работы с удалёнными репозиториями нужны для совместной работы надо проектами. Надо научиться извлекать данные из репозитория и складывать их туда, как результат работы.

Отображение удаленных репозиториев.

Просмотр уже настроенных удалённых репозиториев выполняется командой $git remote -v, параметр -v позволяет увидеть источник, т.е. имя которое Git по умолчанию присваивает удалённому серверу. Один репозиторий можно привязать к нескольким удалённым копиям.

$git remote add origin URL1

$git remote add origin URL2

$git remote add origin URL3

Так как репозиторий «привязан» к нескольким удаленным копиям, то мы можем скопировать любые изменения, сделанным любым разработчиком в любой копии, а так же выгрузить свои, имея при этом соответствующие разрешения.

Добавление удаленных репозиториев.