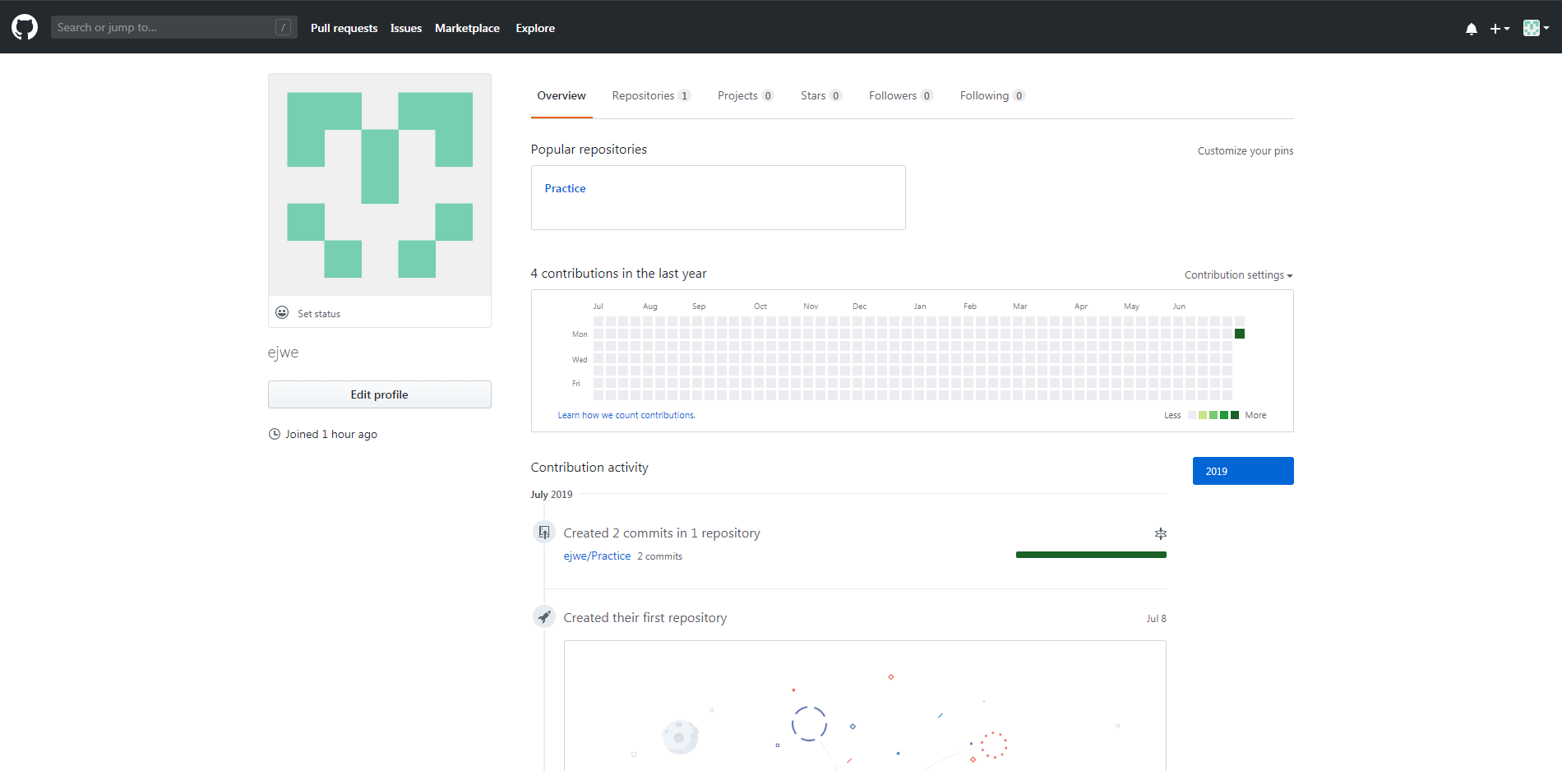
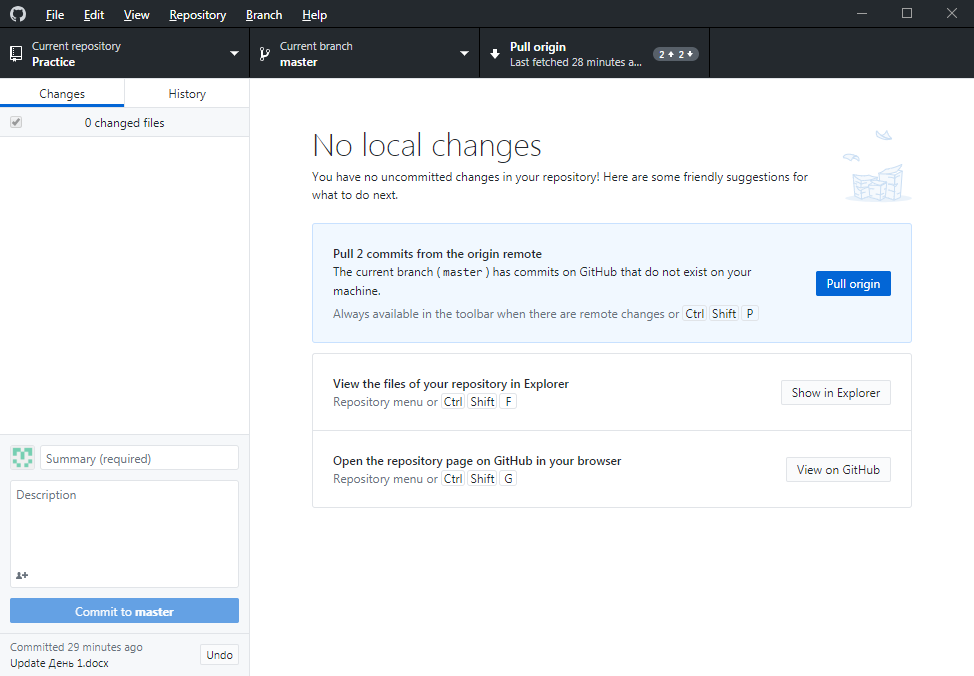
Регистрация на Github:



Клиент:



**Git. Описание и применение**

Git — это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но вы можете использовать его для любых файлов на ваш вкус). С его помощью вы можете откатиться на более старую версию вашего проекта, сравнивать, анализировать, сливать изменения и многое другое. Этот процесс называется контролем версий. Существуют различные системы для контроля версий. Вы, возможно, о них слышали: SVN, Mercurial, Perforce, CVS, Bitkeeper и другие.

Git является распределенным, то есть не зависит от одного центрального сервера, на котором хранятся файлы. Вместо этого он работает полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием. Тем не менее, вы можете хранить копию репозитория онлайн, это сильно облегчает работу над одним проектом для нескольких людей. Для этого используются сайты вроде github и bitbucket.

**Создание нового репозитория**

Как мы отметили ранее, git хранит свои файлы и историю прямо в папке проекта. Чтобы создать новый репозиторий, нам нужно открыть терминал, зайти в папку нашего проекта и выполнить команду init. Это включит приложение в этой конкретной папке и создаст скрытую директорию .git, где будет храниться история репозитория и настройки.  
Создайте на рабочем столе папку под названием git\_exercise. Для этого в окне терминала введите:



|  |  |
| --- | --- |
| 1  2  3 | $ mkdir Desktop/git\_exercise/  $ cd Desktop/git\_exercise/  $ git init |

Командная строка должна вернуть что-то вроде:



|  |  |
| --- | --- |
| 1 | Initialized empty Git repository in /home/user/Desktop/git\_exercise/.git/ |

Это значит, что наш репозиторий был успешно создан, но пока что пуст.

**Подготовка файлов**

В git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, которые нужны в коммите. Сперва он пустой, но затем мы добавляем на него файлы (или части файлов, или даже одиночные строчки) командой add и, наконец, коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой commit.  
В нашем случае у нас только один файл, так что добавим его:



|  |  |
| --- | --- |
| 1 | $ git add hello.txt |

Если нам нужно добавить все, что находится в директории, мы можем использовать



|  |  |
| --- | --- |
| 1 | $ git add -A |

Проверим статус снова, на этот раз мы должны получить другой ответ:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ git status  On branch master  Initial commit  Changes to be committed:  (use "git rm --cached ..." to unstage)  new file: hello.txt |

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

**Коммит(фиксация изменений)**

Коммит представляет собой состояние репозитория в определенный момент времени. Это похоже на снапшот, к которому мы можем вернуться и увидеть состояние объектов на определенный момент времени.  
Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы только что создали его при помощи git add), после которого мы может коммитить:



|  |  |
| --- | --- |
| 1 | $ git commit -m "Initial commit." |

Эта команда создаст новый коммит со всеми изменениями из области подготовки (добавление файла hello.txt). Ключ -m и сообщение «Initial commit.» — это созданное пользователем описание всех изменений, включенных в коммит. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии.

## **Основные команды**

Всего несколько команд нужно для базового варианта использования Git для ведения истории изменений.

**git add**

Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

Это одна из ключевых команд Git, мы упоминали о ней десятки раз на страницах книги. Ниже перечислены наиболее интересные варианты использования этой команды.

Знакомство с этой командой происходит в главе [Отслеживание новых файлов](https://git-scm.com/book/ru/v2/ch00/r_tracking_files).

О том как использовать git add для разрешения конфликтов слияния написано в главе [Основные конфликты слияния](https://git-scm.com/book/ru/v2/ch00/r_basic_merge_conflicts).

В главе [Интерактивное индексирование](https://git-scm.com/book/ru/v2/ch00/r_interactive_staging) показано как использовать git add для добавления в индекс лишь отдельных частей изменённого файла.

В главе [Деревья](https://git-scm.com/book/ru/v2/ch00/r_tree_objects) показано как эта команда работает на низком уровне, чтобы вы понимали, что происходит за кулисами.

**git status**

Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

Мы познакомили вас с этой командой в главе [Определение состояния файлов](https://git-scm.com/book/ru/v2/ch00/r_checking_status), разобрали стандартный и упрощённый формат вывода. И хотя мы использовали git status повсеместно в этой книге, практически все варианты использования покрыты в указанной главе.

**git diff**

Команда git diff используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между вашей рабочей директорией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

Мы познакомили вас с основами этой команды в главе [Просмотр индексированных и неиндексированных изменений](https://git-scm.com/book/ru/v2/ch00/r_git_diff_staged), где показали как посмотреть какие изменения уже добавлены в индекс, а какие — ещё нет.

О том как использовать эту команду для проверки на проблемы с пробелами с помощью аргумента --check можно почитать в главе [Правила создания коммитов](https://git-scm.com/book/ru/v2/ch00/r_commit_guidelines).

Мы показали вам как эффективно сравнивать ветки используя синтаксис git diff A...B в главе [Определение применяемых изменений](https://git-scm.com/book/ru/v2/ch00/r_what_is_introduced).

В главе [Продвинутое слияние](https://git-scm.com/book/ru/v2/ch00/r_advanced_merging) показано использование опции -w для скрытия различий в пробельных символах, а также рассказано как сравнивать конфликтующие изменения с опциями --theirs, --ours и --base.

Использование этой команды с опцией --submodule для сравнения изменений в субмодулях показано в главе [Начало работы с подмодулями](https://git-scm.com/book/ru/v2/ch00/r_starting_submodules).

**git difftool**

Команда git difftool просто запускает внешнюю утилиту сравнения для показа различий в двух деревьях, на случай если вы хотите использовать что-либо отличное от встроенного просмотрщика git diff.

Мы лишь вкратце упомянули о ней в главе [Просмотр индексированных и неиндексированных изменений](https://git-scm.com/book/ru/v2/ch00/r_git_diff_staged).

**git commit**

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

Вы познакомились с основами модели коммитов в главе [Коммит изменений](https://git-scm.com/book/ru/v2/ch00/r_committing_changes). Там же мы продемонстрировали использование опций -a для добавления всех изменений в индекс без использования git add, что может быть удобным в повседневном использовании, и -m для передачи сообщения коммита без запуска полноценного редактора.

В главе [Операции отмены](https://git-scm.com/book/ru/v2/ch00/r_undoing) мы рассказали об опции --amend, используемой для изменения последнего совершённого коммита.

В главе [О ветвлении в двух словах](https://git-scm.com/book/ru/v2/ch00/r_git_branches_overview) мы более подробно познакомились с тем, что делает команда git commit и почему она делает это именно так.

Мы показали вам как подписывать ваши коммиты, используя опцию -S в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

И наконец мы заглянули внутрь команды git commit в главе [Commit Objects](https://git-scm.com/book/ru/v2/ch00/r_git_commit_objects) и узнали что она делает за кулисами.

**git reset**

Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочей директории при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.

Мы рассказали об основах использования git reset в главе [Отмена подготовки файла](https://git-scm.com/book/ru/v2/ch00/r_unstaging), где эта команда использовалась для удаления файла из индекса, добавленного туда с помощью git add.

В главе [Раскрытие тайн reset](https://git-scm.com/book/ru/v2/ch00/r_git_reset), полностью посвящённой этой команде, мы разобрались в деталях её использования.

Мы использовали git reset --hard чтобы отменить слияние в главе [Прерывание слияния](https://git-scm.com/book/ru/v2/ch00/r_abort_merge), там же было продемонстрировано использование команды git merge --abort для этих целей, которая работает как обёртка над git reset.

**git rm**

Команда git rm используется в Git для удаления файлов из индекса и рабочей директории. Она похожа на git add с тем лишь исключением, что она удаляет, а не добавляет файлы для следующего коммита.

Мы немного разобрались с этой командой в главе [Удаление файлов](https://git-scm.com/book/ru/v2/ch00/r_removing_files), показали как удалять файлы из рабочей директории и индекса и только из индекса, используя флаг --cached.

Ещё один вариант использования git rm приведён в главе [Removing Objects](https://git-scm.com/book/ru/v2/ch00/r_removing_objects), где мы вкратце объяснили как использовать опцию --ignore-unmatch при выполнении git filter-branch, которая подавляет ошибки удаления несуществующих файлов. Это может быть полезно для автоматически выполняемых скриптов.

**git mv**

Команда git mv — это всего лишь удобный способ переместить файл, а затем выполнить git addдля нового файла и git rm для старого.

**git clean**

Команда git clean используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.