**Основи**

**Git** - це набір консольних утиліт, які відстежують і фіксують зміни у файлах (найчастіше йдеться про вихідний код програм, але ви можете використовувати його для будь-яких файлів на ваш смак). Спочатку Git був створений Лінусом Торвальдсом під час розробки ядра Linux. Однак інструмент так сподобався розробникам, що згодом, він набув широкого поширення і його стали використовувати в інших проектах. З його допомогою ви можете порівнювати, аналізувати, редагувати, зливати зміни і повертатися назад до останнього збереження. Цей процес називається контролем версій.

**Для чого він потрібен?** Ну, по-перше, щоб відстежити зміни, що відбулися з проектом, з часом. Простіше кажучи, ми можемо подивитися, як змінювалися файли програми, на всіх етапах розроблення, і за потреби повернутися назад і щось відредагувати. Часто бувають ситуації, коли, в цілком собі працюючий код, вам потрібно внести певні правки або поліпшити якийсь функціонал, за бажанням замовника. Однак після впровадження нововведень, ви з жахом розумієте, що все зламалося. У вас починає судорожно смикатися око, а в повітрі повисає німе питання: "Що робити?" Без системи контролю версій, вам треба було б довго напружено переглядати код, щоб зрозуміти як було до того, як все перестало працювати. З Гітом же, все що потрібно зробити - це відкотитися на коміт назад.

По-друге, він надзвичайно корисний під час одночасної роботи кількох фахівців над одним проєктом. Без Гіта трапиться колапс, коли розробники, скопіювавши весь код з головної папки і зробивши з ним задумане, спробують одночасно повернути весь код назад.

Git є розподіленим, тобто не залежить від одного центрального сервера, на якому зберігаються файли. Замість цього він працює повністю локально, зберігаючи дані в директоріях на жорсткому диску, які називаються сховищем. Проте ви можете зберігати копію сховища в Інтернеті, це значно полегшує роботу над одним проектом для кількох людей. Для цього використовуються сайти на кшталт github і bitbucket.

**Установка**

Встановити git на свою машину дуже просто:

* Linux - потрібно просто відкрити термінал і встановити додаток за допомогою пакетного менеджера вашого дистрибутива. Для Ubuntu команда матиме такий вигляд:

sudo apt-get install git

* Windows - ми рекомендуємо git for windows, оскільки він містить і клієнт із графічним інтерфейсом, і емулятор bash.
* OS X - найпростіше скористатися homebrew. Після його встановлення запустіть у терміналі:

brew install git

Якщо ви новачок, клієнт із графічним інтерфейсом (наприклад GitHub Desktop і Sourcetree) буде корисним, але, тим не менш, знати команди дуже важливо.

**Налаштування**

Отже, ми встановили git, тепер потрібно додати трохи налаштувань. Існує досить багато опцій, з якими можна грати, але ми налаштуємо найважливіші: наше ім'я користувача та електронну адресу. Відкрийте термінал і запустіть команди:

git config --global user.name "My Name"

git config --global user.email myEmail@example.com

Тепер кожна наша дія буде позначена ім'ям і поштою. Таким чином, користувачі завжди будуть у курсі, хто відповідає за які зміни - це вносить порядок.

Git зберігає весь пакет конфігурацій у файлі .gitconfig, що знаходиться у вашому локальному каталозі. Щоб зробити ці налаштування глобальними, тобто застосовними до всіх проєктів, необхідно додати прапор -global. Якщо ви цього не зробите, вони поширюватимуться тільки на поточний репозиторій.

Для того, щоб подивитися всі налаштування системи, використовуйте команду:

git config --list

Для удобства и легкости зрительного восприятия, некоторые группы команд в Гит можно выделить цветом, для этого нужно прописать в консоли:

git config --global color.ui true

git config --global color.status auto

git config --global color.branch auto

Якщо ви не до кінця налаштували систему для роботи, на початку свого шляху - не біда. Git завжди підкаже розробнику, якщо той заплутався, наприклад:

1. Команда git --help виводить загальну документацію з git'а
2. Якщо введемо git log --help - він надасть нам документацію по якійсь певній команді (в даному випадку це - log)
3. Якщо ви раптом зробили помилку - система підкаже вам потрібну команду
4. Після виконання будь-якої команди - відзвітує про те, що ви накоїли
5. Також Гіт прогнозує подальші варіанти розвитку подій і завжди направить розробника, який не знає, куди рухатися далі

Тут варто зазначити, що підказувати система буде англійською, але не хвилюйтеся, згодом ви вивчите нескладний алгоритм її роботи і будете розмовляти з нею однією мовою.

**Створення нового сховища**

Як ми зазначили раніше, git зберігає свої файли та історію прямо в папці проєкту. Щоб створити нове сховище, нам потрібно відкрити термінал, зайти в папку нашого проєкту і виконати команду init. Це увімкне додаток у цій конкретній папці та створить приховану директорію .git, де зберігатиметься історія сховища та налаштування.

Створіть на робочому столі папку під назвою git\_exercise. Для цього у вікні терміналу введіть:

$ mkdir Desktop/git\_exercise/

$ cd Desktop/git\_exercise/

$ git init

Командний рядок має повернути щось на кшталт:

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

Це означає, що наш репозиторій був успішно створений, але поки що порожній. Тепер створіть текстовий файл під назвою hello.txt і збережіть його в директорії git\_exercise.

**Визначення стану**

status - це ще одна найважливіша команда, яка показує інформацію про поточний стан сховища: чи актуальна інформація на ньому, чи немає чогось нового, що змінилося тощо. Запуск git status на нашому свіжоствореному репозиторії має видати:

$ git status

On branch master

Initial commit

Untracked files:

(use "git add ..." to include in what will be committed)

hello.txt

Повідомлення говорить про те, що файл hello.txt невідстежуваний. Це означає, що файл новий і система ще не знає, чи потрібно стежити за змінами у файлі, чи його можна просто ігнорувати. Для того, щоб почати відстежувати новий файл, потрібно його спеціальним чином оголосити.

**Підготовка файлів**

У git є концепція області підготовлених файлів. Можна уявити її як полотно, на яке наносять зміни, які потрібні в коміті. Спершу воно порожнє, але потім ми додаємо на нього файли (або частини файлів, або навіть поодинокі рядки) командою add і, зрештою, комітимо все потрібне до репозиторію (створюємо зліпок потрібного нам стану) командою commit.

У нашому випадку у нас тільки один файл, тож додамо його:

$ git add hello.txt

Якщо нам потрібно додати все, що знаходиться в директорії, ми можемо використовувати

$ git add -A

Перевіримо статус знову, цього разу ми повинні отримати іншу відповідь:

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: hello.txt

Файл готовий до коміту. Повідомлення про стан також говорить нам про те, які зміни щодо файлу були проведені в області підготовки - в даному випадку це новий файл, але файли можуть бути модифіковані або видалені.

**Фіксація змін**

**Як зробити коміт**

Уявімо, що нам потрібно додати пару нових блоків у html-розмітку (index.html) і стилізувати їх у файлі style.css. Для збереження змін, їх необхідно закоммітити. Але спочатку, ми маємо позначити ці файли для Гіта, за допомогою команди git add, що додає (або готує) їх до коміту. Додавати їх можна окремо:

git add index.html

git add css/style.css

или вместе - всё сразу:

git add .

Звичайно, додавати все відразу зручніше, ніж прописувати кожну позицію окремо. Однак, тут треба бути уважним, щоб не додати помилково непотрібні елементи. Якщо ж таке сталося, вилучити звідти помилковий файл можна за допомогою команди

git reset:

git reset css/style.css

Тепер створимо безпосередньо сам коміт

git commit -m 'Add some code'

Прапорець -m задасть commit message - коментар розробника. Він необхідний для опису закоммічених змін. І тут працює золоте правило всіх коментарів у коді: "Максимально ясно, просто і змістовно познач написане!"

**Як подивитися коміти**

Для перегляду всіх виконаних фіксацій можна скористатися історією комітів. Вона містить відомості про кожен проведений комітет проєкту. Запросити її можна за допомогою команди:

git log

У ній міститься вся інформація про кожен окремий коміт, із зазначенням його хеша, автора, списку змін і дати, коли вони були зроблені. Відстежити операції, які вас цікавлять, у списку змін, можна за хешем коміту, за допомогою команди git show :

git show hash\_commit

Ну а якщо раптом нам потрібно переробити commit message і внести туди новий коментар, можна написати таку конструкцію:

git commit --amend -m 'Новый комментарий'

У цьому разі повідомлення останнього коміту перезапишеться. Але зловживати цим не варто, оскільки ця операція небезпечна і краще її робити до надсилання коміту на сервер.

**Віддалені репозиторії**

Наразі наш коміт є локальним - існує лише в директорії .git на нашій файловій системі. Незважаючи на те, що сам по собі локальний репозиторій корисний, у більшості випадків ми хочемо поділитися нашою роботою або доставити код на сервер, де він буде виконуватися.

**1. Що таке віддалений репозиторій**

Репозиторій, що зберігається в хмарі, на сторонньому сервісі, спеціально створеному для роботи з git, має низку переваг. По-перше - це свого роду резервна копія вашого проєкту, що надає можливість безболісної роботи в команді. А ще в такому репозиторії можна користуватися додатковими можливостями хостингу. Наприклад - візуалізацією історії або можливістю розробляти вашу програму безпосередньо у веб-інтерфейсі.

**Клонування**

Клонування - це коли ви копіюєте віддалений репозиторій до себе на локальний ПК. Це те, з чого зазвичай починається будь-який проєкт. При цьому ви переносите собі всі файли і папки проєкту, а також всю його історію з моменту його створення. Щоб схилити проєкт, спершу необхідно дізнатися, де він розташований, і скопіювати посилання на нього. У нашому посібнику ми використовуватимемо адресу https://github.com/tutorialzine/awesome-project, але вам порадимо спробувати створити свій репозиторій у GitHub, BitBucket або будь-якому іншому сервісі:

git clone https://github.com/tutorialzine/awesome-project

У разі клонування в поточний каталог, там буде створено папку, в яку помістяться всі проєктні файли і прихована директорія .git, з самим репозиторієм, або з необхідною інформацією про нього. У такій ситуації, для клонованого репозиторію, за замовчуванням, буде створена папка з однойменною назвою, але його можна залити і в іншу директорію, наприклад:

git clone https://github.com/tutorialzine/awesome-project new-folder

**2. Підключення до віддаленого репозиторію**

Щоб завантажити що-небудь у віддалений репозиторій, спочатку потрібно до нього підключитися. Реєстрація та встановлення може зайняти час, але всі подібні сервіси надають хорошу документацію.

Щоб зв'язати наш локальний репозиторій із репозиторієм на GitHub, виконаємо таку команду в терміналі. Зверніть увагу, що потрібно обов'язково змінити URI репозиторію на свій.

# This is only an example. Replace the URI with your own repository address.

$ git remote add origin https://github.com/tutorialzine/awesome-project.git

Проєкт може мати кілька віддалених репозиторіїв одночасно. Щоб їх розрізняти, ми дамо їм різні імена. Зазвичай головний репозиторій називається origin.

**3. Надсилання змін на сервер**

Зараз саме час переслати наш локальний коміт на сервер. Цей процес відбувається щоразу, коли ми хочемо оновити дані у віддаленому репозиторії.

Команда, призначена для цього - push. Вона приймає два параметри: ім'я віддаленого сховища (ми назвали наш origin) і гілку, до якої необхідно внести зміни (master - це гілка за замовчуванням для всіх репозиторіїв).

$ git push origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 212 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/tutorialzine/awesome-project.git

\* [new branch] master -> master

Ця команда трохи схожа на git fetch, з тією лише різницею, що за допомогою fetch ми імпортуємо комміти до локальної гілки, а застосувавши push, ми експортуємо їх з локальної до віддаленої. Якщо вам необхідно налаштувати віддалену гілку, використовуйте git remote. Однак пушити треба обережно, адже розглянута команда перезаписує безповоротно всі зміни. Здебільшого її використовують, щоб опублікувати локальні зміни, що вивантажуються, в центральний репозиторій. А ще її застосовують для того, щоб поділитися, внесеними в локальний репозиторій, нововведеннями, з колегами або іншими віддаленими учасниками розробки проєкту. Підсумувавши сказане, можна назвати git push - командою вивантаження, а git pull і git fetch - командами завантаження або скачування. Після того як ви успішно запустили змінені дані, їх необхідно впровадити або інтегрувати, за допомогою команди злиття git merge.

Залежно від сервісу, який ви використовуєте, вам може знадобитися аутентифікуватись, щоб зміни відправилися. Якщо все зроблено правильно, то коли ви подивитеся у віддалений репозиторій за допомогою браузера, ви побачите файл hello.txt

**4. Запит змін із сервера**

Якщо ви зробили зміни у вашому віддаленому репозиторії, інші користувачі можуть завантажити зміни за допомогою команди pull.

$ git pull origin master

From https://github.com/tutorialzine/awesome-project

\* branch master -> FETCH\_HEAD

Already up-to-date.

Оскільки нових комітів відтоді, як ми склонували собі проєкт, не було, жодних змін доступних для скачування немає.

**Як видалити локальний репозиторій**

Вам не сподобався один із ваших локальних Git-репозиторіїв і ви хочете стерти його зі своєї машини. Для цього вам всього лише треба видалити приховану папку ".git" у кореневому каталозі репозиторію. Зробити це можна 3 способами:

1. Найпростіше вручну видалити цю папку ".git" у кореневому каталозі "Git Local Warehouse".
2. Також видалити репозиторій, який вас не влаштовує, можна на github. Відкриваєте потрібний вам об'єкт і переходите в пункт меню Налаштування. Там, прокрутивши повзунок донизу, ви потрапите в зону небезпеки, де один із пунктів називатиметься "видалення цього сховища".
3. Останній метод видалення локального сховища через командний рядок, для цього в терміналі необхідно ввести таку команду:

cd repository-path/

rm -r .git

**Розгалуження**

Під час розробки нової функціональності вважається гарною практикою працювати з копією оригінального проєкту, яку називають гілкою. Гілки мають свою власну історію та ізольовані одна від одної зміни доти, доки ви не вирішуєте злити зміни разом. Це відбувається з набору причин:

* Уже робоча, стабільна версія коду зберігається.
* Різні нові функції можуть розроблятися паралельно різними програмістами.
* Розробники можуть працювати з власними гілками без ризику, що кодова база зміниться через чужі зміни.
* У разі сумнівів, різні реалізації однієї й тієї самої ідеї можуть бути розроблені в різних гілках і потім порівнюватися.

**1. Створення нової гілки**

Основна гілка в кожному репозиторії називається master. Щоб створити ще одну гілку, використовуємо команду branch <name>

$ git branch amazing\_new\_feature

Це створить нову гілку, поки що точну копію гілки master.

**2. Перемикання між гілками**

Зараз, якщо ми запустимо branch, ми побачимо дві доступні опції:

$ git branch

amazing\_new\_feature

\* master

master - це активна гілка, вона позначена зірочкою. Але ми хочемо працювати з нашою "новою приголомшливою фічею", тож нам знадобиться переключитися на іншу гілку. Для цього скористаємося командою checkout, вона приймає один параметр - ім'я гілки, на яку необхідно переключитися.

$ git checkout amazing\_new\_feature

У Git гілка - це окрема лінія розробки. Git checkout дозволяє нам перемикатися як між віддаленими, так і між локальними гілками. Це один зі способів отримати доступ до роботи колеги або співавтора, що забезпечує більш високу продуктивність спільної роботи. Однак тут треба пам'ятати, що поки ви не закомітили зміни, ви не зможете переключитися на іншу гілку. У такій ситуації потрібно або зробити коміт, або відкласти його, за допомогою команди git stash, що додає поточні незакомічені зміни до стека змін і скидає робочу копію до HEAD'а репозиторію.

**3. Злиття гілок**

Наша “потрясающая новая фича” будет еще одним текстовым файлом под названием feature.txt. Мы создадим его, добавим и закоммитим:

$ git add feature.txt

$ git commit -m "New feature complete.”

Зміни завершено, тепер ми можемо переключитися назад на гілку master.

$ git checkout master

Тепер, якщо ми відкриємо наш проєкт у файловому менеджері, ми не побачимо файлу feature.txt, тому що ми перемкнулися назад на гілку master, у якій такого файлу не існує. Щоб він з'явився, потрібно скористатися merge для об'єднання гілок (застосування змін із гілки amazing\_new\_feature до основної версії проєкту).

$ git merge amazing\_new\_feature

Тепер гілка master актуальна. Гілка amazing\_new\_feature більше не потрібна, і її можна видалити.

$ git branch -d awesome\_new\_feature

Якщо хочете створити копію віддаленого репозиторію - використовуйте git clone. Однак якщо вам потрібна лише певна його гілка, а не все сховище - після git clone виконайте наступну команду у відповідному репозиторії:

git checkout -b <имя ветки> origin/<имя ветки>

Після цього нова гілка створюється на машині автоматично.

**Як видаляти гілки в Git?**

Бувають ситуації, коли після зливу якихось змін із робочої гілки у вихідну версію проєкту, її, за правилами хорошого тону, необхідно видалити, щоб вона більше не заважала у вашому коді. Але як це зробити?

Для локально розташованих гілок існує команда:

git branch -d local\_branch\_name

де прапорець -d є опцією команди git branch - це скорочена версія ключового слова --delete, призначеного для видалення гілки, а local\_branch\_name - назва непотрібної нам гілки.

Однак тут є нюанс: видалити поточну гілку, в яку ви, в даний момент переглядаєте - не можна. Якщо ж ви все-таки спробуєте це зробити, система насварить вас і видасть помилку з таким змістом:

Error: Cannot delete branch local\_branch\_name checked out at название\_директории

Тож під час видалення гілок, обов'язково переключіться на інший branch.