GiftHub - це система управління проектами і версіями коду, а також платформа соціальних мереж, створена для розробників. Але для чого використовується GitHub? Ну, серед іншого, це дозволяє вам працювати спільно з іншими людьми по всьому світу, планувати свої проекти і відстежувати свою роботу.

GitHub також є одним з найбільших онлайн-сховищ (англ) спільної роботи по всьому світу.Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проектів пропонуються різні платні тарифні плани.

Що таке GIT?

Git (вимовляється «гіт») - розподілена система керування версіями. Проект був створений Лінус Торвальдс для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року. На сьогодні його підтримує Джун Хама.



Що таке система контролю версій?

Коли розробники створюють новий проект, вони завжди продовжують вносити оновлення в код. Навіть після запуску проектів їм все одно потрібно оновлювати версії, виправляти помилки, додавати нові функції і т. Д.

Система контролю версій допомагає відслідковувати зміни, внесені в базу коду. Більш того, він записує, хто вніс зміни і може відновити стертий або змінений код.

Перезаписаних кодів не існує, оскільки Git зберігає кілька копій в сховище. Якщо ви зацікавлені в Git, ви можете дізнатися більше про Git тут.

Що таке?

Якщо Git - це серце GitHub, то Hub - це його душа. Концентратор в GitHub - це те, що перетворює командний рядок, таку як Git, в найбільшу соціальну мережу для розробників.

Крім участі в певному проекті, GitHub дозволяє користувачам спілкуватися з однодумцями. Ви можете слідкувати за людьми і дивитися, що вони роблять або з ким вони спілкуються.

Репозиторій

Репозиторій або сховище - це каталог, в якому зберігаються файли вашого проекту. Він може бути розташований в сховище GitHub або в локальному сховищі на вашому комп'ютері. Ви можете зберігати файли кодів, зображення, аудіо або все, що пов'язано з проектом, в сховище.

Гілка

Гілка це копія вашого сховища. Ви можете використовувати гілку, коли хочете зробити розробку ізольовано.

Робота з гілкою не вплине на центральне сховище або інші гілки. Якщо ви зробили цю роботу, ви можете об'єднати свою гілку з іншими гілками і центральним репозиторієм, використовуючи запит на витяг.

Результат регестрації GitHub



Основні команди Git

**git add**

Команда **git add** додає вміст робочої директорії в індекс (staging area) для подальшого коммітов. За замовчуванням git commit використовує лише цей індекс, так що ви можете використовувати git add для збірки зліпка вашого наступного коммітов.

Це одна з ключових команд Git, ми згадували про неї десятки разів на сторінках книги. Нижче перераховані найбільш цікаві варіанти використання цієї команди.

Знайомство з цією командою відбувається в голові Відстеження нових файлів.

Про те як використовувати git add для вирішення конфліктів злиття написано в розділі Основні конфлікти злиття.

У розділі Інтерактивне індексування показано як використовувати git add для додавання в індекс лише окремих частин зміненого файлу.

У розділі Дерева показано як ця команда працює на низькому рівні, щоб ви розуміли, що відбувається за лаштунками.

**git status**

Команда **git status** показує стану файлів в робочій директорії і індексі: які файли змінені, але не додано в індекс; які очікують коммітов в індексі. Додатково до цього виводяться підказки про те, як змінити стан файлів.

Ми познайомили вас з цією командою в розділі Визначення стану файлів, розібрали стандартний і спрощений формат виведення. І хоча ми використовували **git status** повсюдно в цій книзі, практично всі варіанти використання покриті в зазначеній главі.

**git diff**

Команда **git diff** використовується для обчислення різниці між будь-якими двома Git деревами. Це може бути різниця між вашою робочою директорією і індексом (власне **git diff**), різниця між індексом і останнім коммітов **(git diff** --staged), або між будь-якими двома коммітов **(git diff** master branchB).

Ми познайомили вас з основами цієї команди в розділі Перегляд індексованих і неіндексованих змін, де показали як подивитися які зміни вже додані в індекс, а які - ще ні.

Про те як використовувати цю команду для перевірки на проблеми з пробілами за допомогою аргументу --check можна почитати в розділі Правила створення коммітов.

Ми показали вам як ефективно порівнювати гілки використовуючи синтаксис **git diff** A ... B в розділі Визначення застосовуваних змін.

У розділі Просунуте злиття показано використання опції -w для приховування відмінностей в пробільних символах, а також розказано як порівнювати конфліктуючі зміни з опціями --theirs, --ours і --base.

Використання цієї команди з опцією --submodule для порівняння змін в субмодуля показано в главі Початок роботи з подмодулей.

**git difftool**

Команда **git difftool** просто запускає зовнішню утиліту порівняння для показу відмінностей в двох деревах, на випадок якщо ви хочете використовувати що-небудь відмінне від вбудованого переглядача **git diff**.

Ми лише коротко згадали про неї в голові Перегляд індексованих і неіндексованих змін.

**git commit**

Команда **git commit** бере всі дані, додані в індекс за допомогою **git add**, і зберігає їх зліпок у внутрішній базі даних, а потім зрушує покажчик поточної гілки на цей зліпок.

Ви познайомилися з основами моделі коммітов в розділі Комміт змін. Там же ми продемонстрували використання опцій -a для додавання всіх змін в індекс без використання git add, що може бути зручним в повсякденному використанні, і -m для передачі повідомлення коммітов без запуску повноцінного редактора.

У розділі Операції скасування ми розповіли про опції --amend, використовуваної для зміни останнього скоєного коммітов.

У розділі Про розгалуження в двох словах ми більш детально познайомилися з тим, що робить команда **git commit** і чому вона робить це саме так.

Ми показали вам як підписувати ваші коммітов, використовуючи опцію -S в розділі Підпис коммітов.

І нарешті ми заглянули всередину команди **git commit** в розділі Об'єкти коммітов і дізналися що вона робить за лаштунками.

**git reset**

Команда **git reset**, як можна здогадатися з назви, використовується в основному для скасування змін. Вона змінює покажчик HEAD і, опціонально, стан індексу. Також ця команда може змінити файли в робочій директорії при використанні параметра --hard, що може привести до втрати напрацювань при неправильному використанні, так що переконаєтеся в серйозності своїх намірів перш ніж використовувати його.

Ми розповіли про основи використання **git reset** в розділі Скасування індексації файлу, де ця команда використовувалася для видалення файлу з індексу, доданого туди за допомогою **git add**.

У розділі Розкриття таємниць reset, повністю присвяченій цій команді, ми розібралися в деталях її використання.

Ми використовували git reset --hard щоб скасувати злиття в розділі Переривання злиття, там же було продемонстровано використання команди git merge --abort для цих цілей, яка працює як обгортка над **git reset**.

**git rm**

Команда **git rm** використовується в Git для видалення файлів з індексу і робочої директорії. Вона схожа на **git add** з тим лише винятком, що вона видаляє, а не додає файли для наступного коммітов.

Ми трохи розібралися з цією командою в розділі Видалення файлів, показали як видаляти файли з робочої директорії і індексу і тільки з індексу, використовуючи прапор --cached.

Ще один варіант використання git rm наведено в розділі Видалення об'єктів, де ми коротко пояснили як використовувати опцію --ignore-unmatch при виконанні git filter-branch, яка пригнічує помилки видалення неіснуючих файлів. Це може бути корисно для автоматично виконуваних скриптів.

**git mv**

Команда **git mv** - це всього лише зручний спосіб перемістити файл, а потім виконати **git add** для нового файлу і git rm для старого.

Ми лише коротко згадали цю команду в розділі Переміщення файлів.

**git clean**

Команда **git clean** використовується для видалення сміття з робочою директорії. Це можуть бути результати збирання проекту або файли конфліктів злиттів.

