Git - Intro

Spis treści

Intro

Git jest przykładem systemu **DVCS**, czyli jest rozproszonym systemem kontroli wersji. Spokojnie można powiedzieć, że jest najbardziej popularnym stosowanym rozwiązaniem. Autorem **Git**a jest Linus Torvalds, a sam **Git** został stworzony w roku 2005. Narzędzie to miało pomóc w rozwoju jądra systemu operacyjnego Linux. Swoją drogą to ten sam Pan jest twórcą jądra systemu operacyjnego Linux.

Praca lokalna

Jedną z zalet **Git**a jest to, że można z nim pracować bez stałego podłączenia do Internetu. Wynika to ze sposobu działania tego narzędzia, a konkretnie z tego, że **DVCS** replikuje wszystkie informacje o repozytorium u każdego klienta. Czyli każdy klient, który korzysta z **Git**a ma u siebie na komputerze odtworzoną kopię repozytorium. Synchronizacja lokalnego repozytorium z centralnym serwerem może odbywać się od czasu do czasu. Wszystkie potrzebne informacje są wtedy odtwarzane na komputerze klienta i właśnie dlatego do pracy z tym narzędziem nie jest potrzebny stały dostęp do Internetu. Oznacza to, że możemy przeglądać całą historię zmian w projekcie bez dostępu do Internetu. Możemy również dodawać nowe pliki do repozytorium bez podłączenia do Internetu. Synchronizacja stanu tego repozytorium z serwerem będzie już wymagała takiego podłączenia. Czyli Internet będzie nam potrzebny do pobrania najnowszych informacji lub wysłania naszych najnowszych zmian. Dzięki takiemu podejściu praca z **Git**em jest szybsza niż z innymi rodzajami rozwiązań.

Praca z Serwerem

Korzystając z **Git**a możemy pracować używając tylko i wyłącznie naszego **local repository** (*repozytorium lokalne*), czyli repozytorium u nas na komputerze. Wcale nie musimy synchronizować naszych zmian z innymi serwerami ani innymi komputerami. W takim podejściu należy jednak pamiętać, że jeżeli uszkodzeniu ulegnie dysk naszego komputera - tracimy cały projekt. Bezpieczniejszym podejściem jest synchronizowanie naszych zmian z serwerem, który będzie określany jako **remote repository** (*repozytorium zdalne*). Wspomnieliśmy wcześniej, że klienci mogą umówić się, który komputer jest głównym serwerem, który możemy traktować jak repozytorium zdalne.

Dzięki stosowaniu zdalnego repozytorium możemy współpracować na tym samym projekcie z innymi osobami, gdzie każda osoba jest rozumiana jako klient w rozproszonym systemie kontroli wersji. Należy pamiętać, że **Git** umożliwia jednoczesną współpracę z kilkoma zdalnymi repozytoriami. Dzięki zdalnym

repozytorium możemy współdzielić kod z innymi członkami zespołu.

Jak Git widzi zmiany?

W jaki sposób **Git** rozpoznaje zmiany w plikach? **Git** tworzy **snapshot** (*migawkę*) stanu repozytorium w danej chwili. W takim zrzucie/zdjęciu/migawce zapisywane są informacje o wszystkich plikach w repozytorium. Jeżeli w danym punkcie w czasie, jakiś konkretny plik nie był modyfikowany, **Git** przechowuje referencję do najnowszej wersji takiego pliku. Dzięki temu nie musi za każdym razem aktualizować informacji, które nie są zmieniane.

Git różni się od innych **VCS**ów tym, że inne narzędzia przechowują informacje o modyfikacjach w plikach jako listę zmian, a **Git** przechowuje **snapshot** stanu w danym momencie.

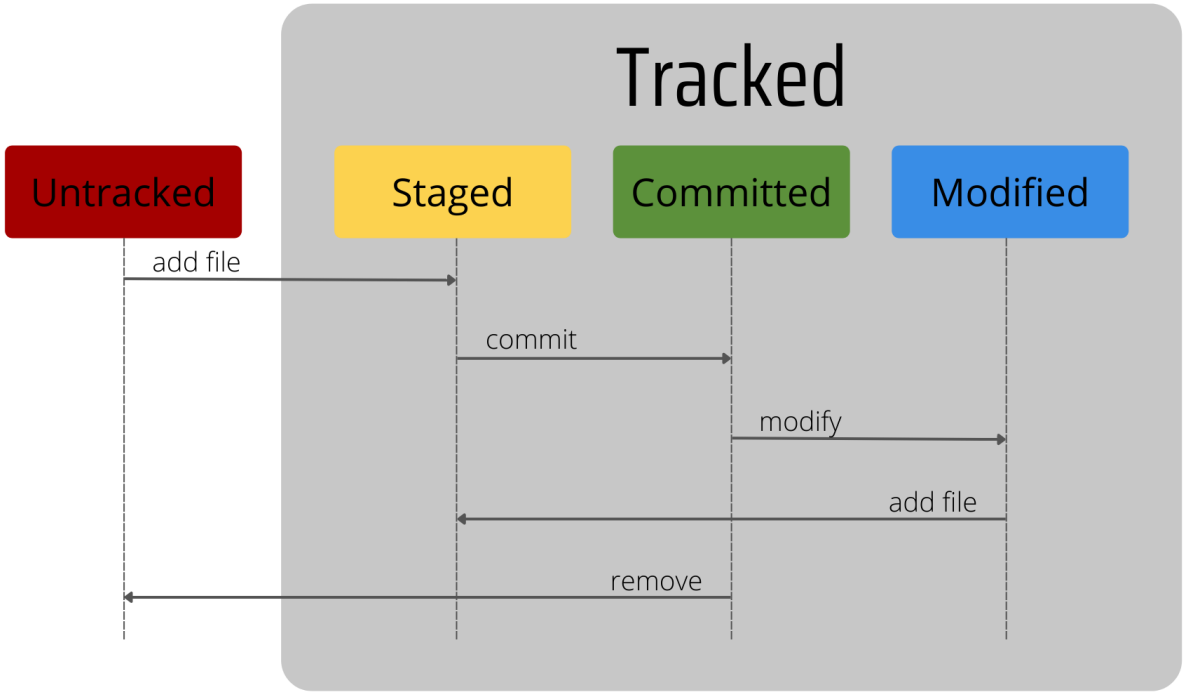
Stany plików

Wybacz moje polskie łamańce językowe, ale w praktyce najczęściej używam w tej tematyce angielskich słów.

Przypomnijmy, że **Git** pomaga w pracy z plikami. W rozumieniu **Git**a, każdy plik może znajdować się w jakimś stanie/statusie. Trzy główne stany plików, jakie wyróżniamy to:

- **modified** (*zmodyfikowany*) - oznacza, że plik został zmodyfikowany, ale informacja o tym nie została zapisana w lokalnej bazie danych,
- **staged** (*wystawiany na scenie* ☺ przetłumaczmy to jako *śledzony*) - oznacza, że plik ma zostać uwzględniony w kolejnej migawce, która będzie zapisywała stan repozytorium,
- **committed** (*zatwierdzony*) - modyfikacja pliku została bezpiecznie zapisana w lokalnej bazie danych.

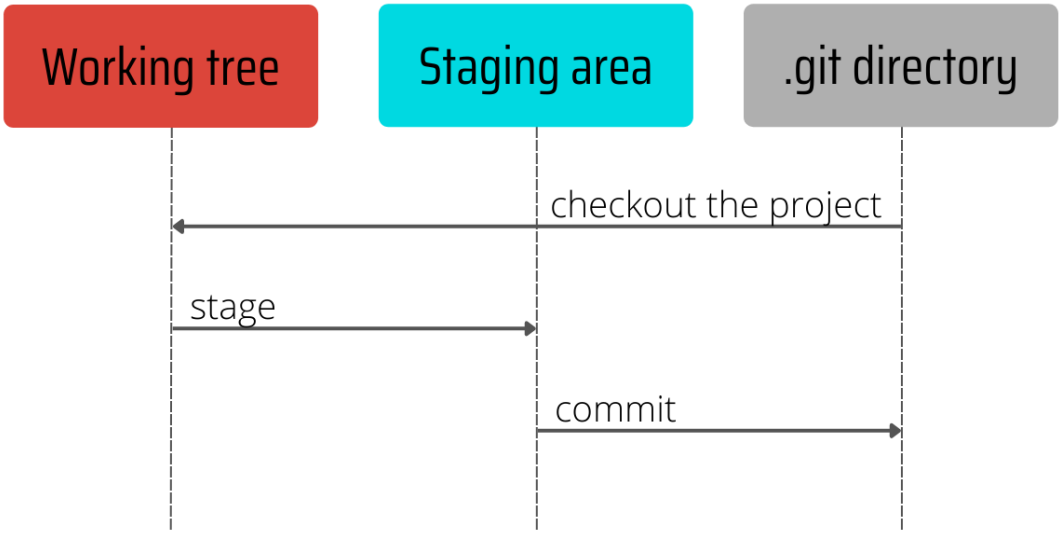
Wszystkie wymienione stany możemy określić jako **tracked**. Czyli są to stany, w których **Git** oznacza pliki, na których pracuje, albo inaczej mówiąc, pliki, które "uznaje", albo o których "wie", że ma na nich pracować. Wspominam o tym, bo jeżeli będziemy odnosili się do plików, o których **Git** nie wie, to nazywamy wtedy takie pliki **untracked**. Wszystko się wyjaśni, jak przejdziemy do przykładów praktycznych. Tymczasem spójrz na diagram poniżej, na którym są pokazane przejścia pomiędzy kolejnymi stanami plików:



Obraz 1. Git stany pliku

Wiedząc już, w jakim stanie mogą znajdować się pliki, możemy przejść do sekcji, jakie możemy wyróżnić w projekcie, który stosuje **Git**:

- **working tree** (*katalog roboczy*) - ten katalog jest reprezentacją projektu w danej wersji. W tym katalogu znajdziemy pliki na dysku, na których możemy pracować, czyli możemy je dodawać, usuwać albo modyfikować. Pliki te są wyciągane (eng. **pulled**) z bazy danych z katalogu `.git`.
- **staging area** (*obszar plików śledzonych*) - jest to oznaczenie dla pliku, który przechowuje informacje o plikach, których stan zostanie zapisany w naszym lokalnym repozytorium. Możemy to rozumieć jako miejsce pośrednie pomiędzy katalogiem roboczym a naszą lokalną bazą danych. Można również spotkać się z określeniem **index**.
- **git directory** (*katalog .git*) - w katalogu `.git` przechowywane są metadane i obiektowa "baza danych" naszego projektu. To w tym katalogu są przechowywane informacje, które są kopiowane, gdy synchronizujemy nasze lokalne repozytorium z repozytorium zdalnym.



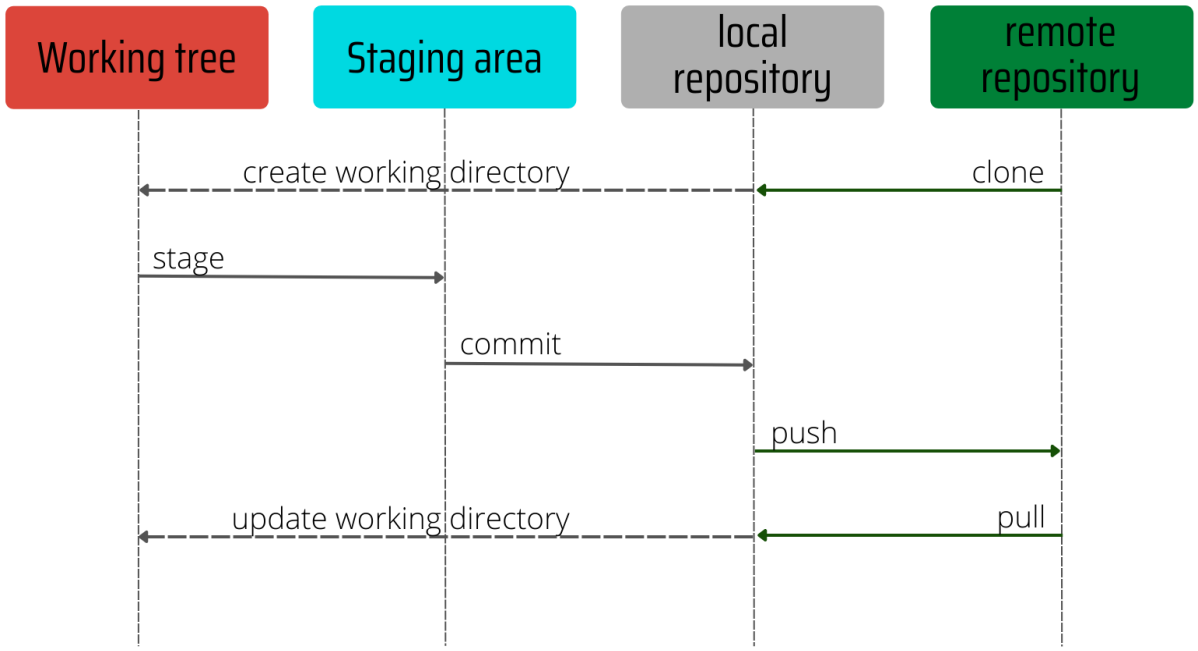
Obraz 2. Git sections

Gdybyśmy mieli wyróżnić, jak możemy przechodzić pomiędzy kolejnymi stanami (możemy to nazwać **Git workflow**), moglibyśmy wyróżnić następujące czynności:

1. Wyciągamy pliki ze skompresowanej bazy danych z **git directory**. Nazywane jest to stwierdzeniem **checkout project**.
2. Modyfikujemy pliki, które znajdują się w **working tree**. Pliki takie znajdą się w stanie **modified**.
3. Wybieramy te, które mają zostać zapisane w kolejnej migawce i dodajemy je do **staging area**. Tylko pliki ze **staging area** będą zapisane w następnej migawce. Pliki takie są w stanie **staged**.
4. Wykonujemy zapis migawki (czyli **commit**), co oznacza, że stan plików, które znajdują się w **staging area**, zostanie zapisany do lokalnej bazy danych. Pliki takie są wtedy w stanie **committed**.

Jeżeli jakaś wersja pliku znajduje się w katalogu `.git`, jest ona wtedy uważana za zatwierdzoną (**committed**). Jeżeli taki plik zostanie zmodyfikowany i dodany do **staging area**, taki plik jest w stanie **staged**. Jeżeli natomiast po zrobieniu **checkout** plik został zmieniony, znajduje się on w stanie **modified**.

Przesyłanie plików do repozytorium zdalnego jest opcjonalne, ale wtedy diagram wyglądałby w ten sposób:



Obraz 3. Git sections with remote repository

Jeżeli chodzi o nazwy wszystkich czynności wymienionych na diagramie - będziemy o nich rozmawiać w kolejnych materiałach.

Na razie to jest sucha teoria, ale w kolejnych materiałach przejdziemy z tym do praktyki. Możliwe, że w tej chwili trochę Ci się to miesza - spokojnie, jak przejdziemy do części praktycznej, wszystko stanie się jaśniejsze.