Kontrola wersji

Posiadanie wielu backupów swojego oprogramowania stało się normalną praktyką dla ratowania się przed sytuacją gdzie nasz kod przestał działać przez wprowadzenie drobnej zmiany. Rozwiązanie takiego problemu może zabrać programiście wiele godzin i wygenerować wiele frustracji. Taka sytuacja jest niedopuszczalna gdyż czas i mentalna energia to najważniejsze surowce pracy programisty.
Tym co odróżnia doświadczonych programistów od początkujących to stosowanie praktyk które oszczędzają czas i energię, dzięki nie używaniu ich na walkę z problemami których dałoby się uniknąć.
Git jest systemem kontroli wersji oprogramowania. Zapisując różnice między plikami różnych wersji projektu jesteśmy w stanie zminimalizować wielkość posiadanych backupów. Git wprowadza też porządek w przechowywaniu wersji nie skazując nas na posiadanie tysiąca folderów ze starym oprogramowaniem.

Git - podstawy

Nasz projekt w Gicie nosi nazwę **repozytorium**. Każde repo możemy przechowywać lokalnie na swojej maszynie lub zdalnie na serwerze korzystając z takich serwisów jak GitHub czy GitLab.

Z gita możemy korzystać za pomocą GUI lub konsoli. Tu użyjemy konsolowej wersji gita by poznać wszystko od kuchni. Niezależnie od systemu będziemy korzystać z Linuxowej powłoki konsolowej Bash.

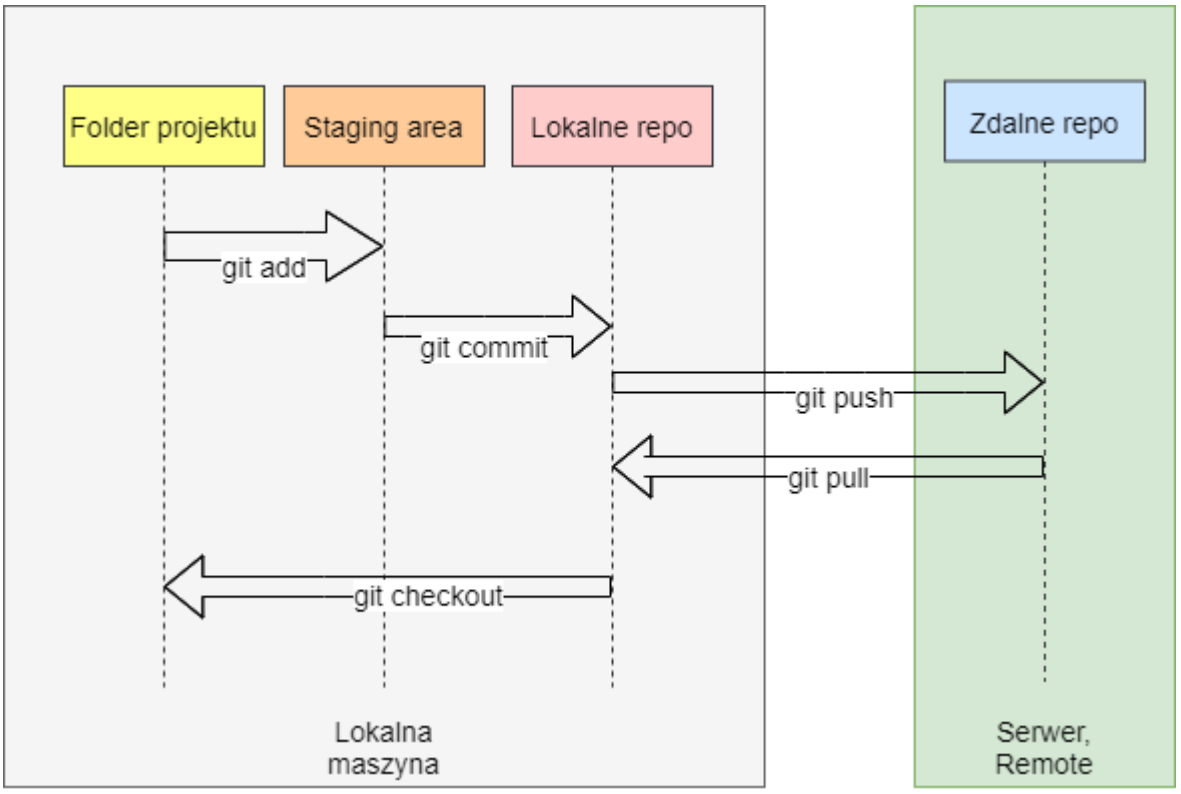
“Sfery” Gita

Możemy przestrzeń gita podzielić na 2 sfery:
- lokalna
- zdalna (remote)

Z czego w sferze lokalnej możemy wydzielić:
- folder projektu
- staging area
- repo

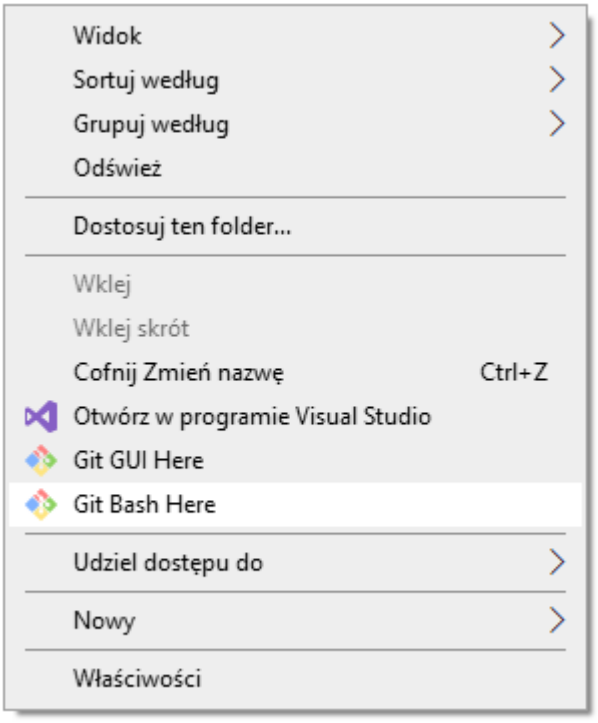
Lokalne repozytorium gita jest zazwyczaj stworzone po to by być klonem repozytorium zdalnego. Taki klon jest naszą kopią repozytorium którą możemy do woli edytować. Zmiany te też możemy do woli przesyłać na zdalne repo.
Lokalnie użytkownik pracuje nad plikami w swoim folderze. Wszelkie zmiany i nowe pliki są początkowo przenoszone do staging area. Zmiany w staging area mogą być zacommitowane, czyli zapisane w repo.
Commit zawiera aktualne zawartości wszystkich plików w repo w momencie stworzenia commitu.Czyli commit jest wersją naszego projektu stworzoną w danym czasie. Każdy

commit ma swój opis, mówiący w skrócie co dana wersja zmieniła względem poprzedniej. Dobre opisywanie commitów to podstawa porządku w repozytorium.



Branching

Git pozwala jeszcze na rozgałęzianie naszego projektu i wprowadzanie zmian wielotorowo w tym samym czasie. Rozgałęzienie projektu osiąga się poprzez stworzenie nowego brancha(gałęzi). Nowo stworzone repo ma domyślnie jeden, główny, branch o nazwie master (w nowszych wersjach main). Wszelkie pozostałe branche można powrotnie scalić (merge). Przy scalaniu mogą pojawić się jednak konflikty jeśli scalimy dwie zmiany tego samego pliku.



Git - obsługa

Ściągamy narzędzia do gita z oficjalnej strony. i instalujemy. W przypadku Linuxa komendy Gita będą dostępne w naszej konsoli od razu.

W przypadku Windowsa w menu rozwijanym dostaniemy nowe opcje gita, będziemy korzystać z “Git Bash here”. Otworzy nam to konsolę Bash.

Inicjalizacja repo

Wybieramy folder w którym będzie nasz projekt. Otwieramy ten folder w konsoli.

OPCJA 1

Jeśli chcemy stworzyć nowe repo od zera wpisujemy:

```
git init
```

Stworzyliśmy tak wyłącznie lokalne repozytorium. Jeśli chcemy połączyć je ze zdalnym musimy dodać komendę:

```
git remote add origin LINK_DO_ZDALNEGO_REPO
```

OPCJA 2

Jeśli mamy zdalne repo które chcemy ściągnąć na naszą maszyne możemy wpisać:

```
git clone LINK_DO_ZDALNEGO_REPO
```

Dodawanie plików do lokalnego repo

Pliki których wersje chcemy przetrzymywać dodajemy za pomocą

```
git add NAZWA_PLIKU
```

By dodać wszystkie pliki w aktualnym folderze piszemy gwiazdkę zamiast nazwy

```
git add *
```

Dodane pliki trafiają do staging area.

Commity

By stworzyć nowy commit i zapisać zmiany dodane do staging area piszemy:

```
git commit -m “OPIS COMMITA”
```

Jeśli chcemy tymczasowo przejść do innego commita, możemy użyć:

git checkout HASH_COMMITA

Wymiana danych z remotem

Jeśli chcemy wysłać nasz nowy commit na serwer, musimy zrobić **push**.

git push

Jeśli ktoś zrobił jakieś zmiany w projekcie na serwerze i chcemy je pobrać, robimy pull.

git pull

Branching

Rozgałęzienie projektu na nową gałąź:

git branch NAZWA_BRANCHA

Powrotne scalenie zmian z dwóch gałęzi:

git merge HASH_COMMITA

Merge wykonujemy będąc na starszym branchu, a w argumencie wpisujemy nowszy branch lub commit.

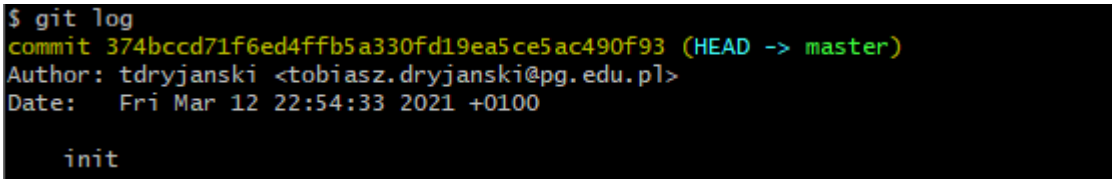
Cykl pracy

W czasie pracy z gitem będziecie korzystać z kilku dodatkowych komend. Typowy cykl pracy z gitem może wyglądać w ten sposób:

1. Robimy ***git pull*** aby pobrać najnowszy kod z remote (gdyż chcemy aktualizować najnowszy kod).
2. Sprawdzamy historię naszych commitów za pomocą:

git log

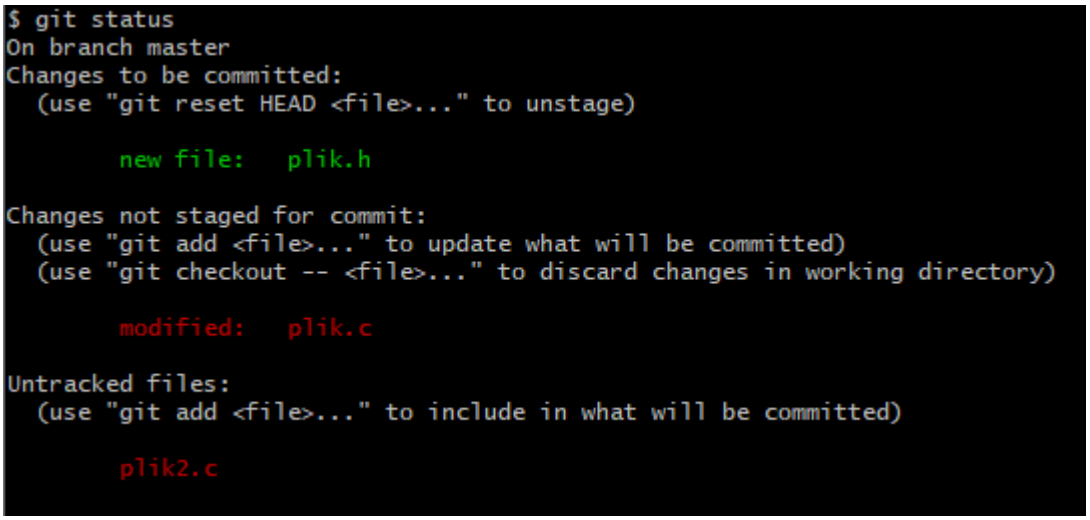
Wskaźnik HEAD pokazuje na jakim commicie jest nasze lokalne repozytorium. Commity są identyfikowane za pomocą unikatowego hasha.



3. Robimy zmiany w naszym lokalnym kodzie
4. By sprawdzić jakie pliki zostały zmienione wpisujemy:

git status

Wszystkie zmienione pliki oraz pliki które nie są w naszym repo, są wypisane na czerwono. Wszystkie nowo dodane zmiany oraz nowo dodane pliki na zielono.
Git status pokazuje nam też jakie nowe pliki zostały stworzone w folderze. Możemy je dodać do repo za pomocą ***git add***.



5. Commitujemy zmiany

By nasz commit dodał automatycznie wszystkie zmiany w dodanych już plikach do staging area, używamy opcji ***-a***. To najszybszy sposób na commitowanie, należy tylko pamiętać by sprawdzać w git status czy nie ma nowych plików które trzeba dodać ręcznie!

git commit -a -m “opis”

6. Push

Zapisujemy pracę na serwerze za pomocą **git push**.

Zakończenie

Git ma wiele dodatkowych opcji, ale te tutaj wymienione są wystarczające by sprawnie posługiwać się gitem. W razie możliwości można korzystać też z GUI gita. Niektóre edytory kodu zawierają domyślnie wsparcie dla gita - wyświetlają zmiany w waszym interfejsie i pozwalają robić commity z GUI.
By utrwalić sobie te informacje należałoby to przećwiczyć:

- spróbować stworzyć swoje lokalne repozytorium z prostymi plikami,
- spróbować te pliki zmienić i zacommitować (kilka razy)
- spróbować wrócić do starszych commitów
- spróbować zrobić nowy branch i zrobić na nim commit
- spróbować zrobić merge nowego brancha z masterem

Jeśli ten wstęp nie rozwiał waszych wątpliwości, możecie poszukać też innych tutoriali do gita w internecie.