Centrum Mistrzostwa Informatycznego

Zarządzanie zmianami w kodzie programu

lokalne i zdalne repozytoria git

Radosław Roszczyk

18 grudnia 2021

Politechnika Warszawska

O czym będziemy dziś mówić?

1. Wprowadzenie

2. GIT

3. Praca z GIT

Wprowadzenie

System kontroli wersji

Sprawne zarządzanie zmianami w kodzie źródłowym jest jednym z ważniejszych elementów całego procesu tworzenia oprogramowania. Do tego celu wykorzystuje się systemy kontroli wersji (ang. *Version Control System* – VCS). Najważniejszymi zadaniami tego typu systemów są:

- śledzenie zmian wprowadzanych w plikach źródłowych projektu,
- przywoływanie wcześniejszych wersji plików,
- łączenie zmian wprowadzanych przez innych programistów,
- śledzenie wprowadzonych zmian,
- umożliwienie równoczesnej pracy wielu programistów nad wspólnym kodem,
- wersjonowanie plików

Modele systemu kontroli wersji

Systemy scentralizowane (ang. *Centralized Version Control System* – CVCS) wykorzystują centralne serwery do przechowywania całego kodu i wspierają pracę typu klient-serwer. Najczęście do pracy z repozytorium wykorzystywana jest dedykowana aplikacja.

Systemy rozproszone (ang. *Distributed Version Control System* – DVCS) oparte o architekturę P2P, gdzie każdy z użytkowników posiada własną kopię całego repozytorium. Po wprowadzeniu zmian w kodzie, zmiany te są mogą być synchronizowane pomiędzy klientami.

Systemy lokalne umożliwiają tworzenie lokalnych repozytoriów kodu na dysku komputera. Zazwyczaj wykorzystują proste mechanizmy tworzenia i kompresji plików różnicowych zawierających wprowadzone zmiany w połączeniu z ich etykietowaniem.

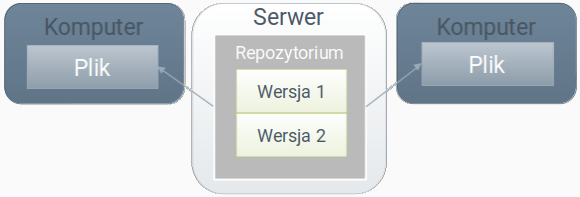
Scentralizowany model pracy

Model scentralizowany pozwala na:

- używanie jako miejsce przechowywania kopii zapasowej (backup),
- nie zapewnia wsparcia równoległej pracy,
- do wykonania większości standardowych operacji wymaga połączenia do serwera,

Systemy scentralizowane:

- SVN
- CVS
- Perforce



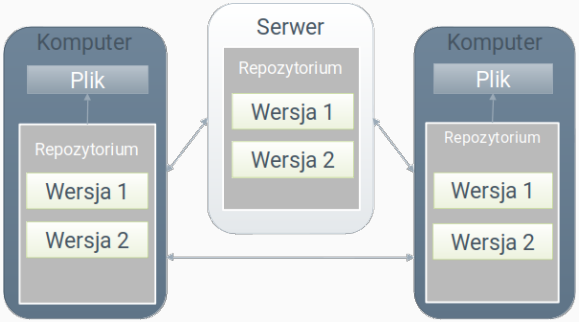
Rozproszony model pracy

Model rozproszony pozwala na:

- możliwość pracy na wielu zdalnych repozytoriach,
- wsparcie dla lokalnych/prywatnych branch'y,
- nie potrzebuje dostępu do serwera dla większości operacji,
- zapewnia pełną historię zmian dostępną lokalnie.

Systemy rozproszone:

- Mercurial
- Bazaar
- GIT



GIT

GIT – uniwersalny system kontroli wersji

Najważniejszymi cechami systemu GIT są:

- możliwość pracy lokalnej, rozproszonej oraz zcentralizowanej,
- możliwość łączenia wielu różnych wersji kodu (gałęzi),
- efektywna praca z projektami zawierającymi dużą liczbę plików,
- praca w trybie off-line, bez połączenia z innymi repozytoriami,
- wsparcie dla nieliniowego programowania (branche),
- wiele możliwości publikacji repozytoriów:
  - git://
  - http(s)://
  - ssh://
- adresowanie przez wartość skrótu – SHA-1
  - #24b9da (24b9da6552252987aa493b52f8696cd6d3b00373)

Co to właściwie ten GIT?

- GIT to system kontroli wersji,
- stworzony jako zastępstwo BitKeepera,
- Linus Torvalds nazwał tak swój kolejny projekt – „*I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git*"
- Projekt miał opierać się na:
  - szybkości,
  - prostocie,
  - wsparcie dla nieliniowego wytwarzania oprogramowania,
  - całkowite rozproszenie,
  - efektywnym przechowywaniu dużych projektów (np. kernel Linuksa).
- open source (GNU General Public License, v2),
- GIT to również zbiór narzędzi do operowania repozytoriami.

Instalacja GIT'a

Oprogramowanie Git jest dostępne dla wielu systemów operacyjnych: Windows, Linux, MacOS. Przesyłanie danych przez sieć w celu synchronizacji repozytoriów oraz zarządzanie plikami w ramach lokalnego repozytorium to zadania, które nie zależą ściśle od wybranego systemu operacyjnego.

- W podstawowej wersji narzędzie to wykorzystuje linię komend,
- Oprócz wersji tekstowej dostępne jest wiele środowisk graficznych wspomagających pracę z GIT,
- Zintegrowane środowiska IDE często posiadają wbudowaną obsługę GIT'a.

Wersję instalacyjną GIT'a można pobrać ze strony: *https://git-scm.com/*

Konfiguracja GIT'a

Git posiada długą listę parametrów konfiguracyjnych pozwalających dostosować do swoich potrzeb środowisko pracy oraz zdefiniować informacje o użytkowniku (autorze) wprowadzanych zmian. Najważniejsze parametry, które warto ustawić to:

- domyślne dane użytkownika,

```
git config --global user.name "Radosław Roszczyk"
git config --global user.email "rroszczyk@noreply.github.com"
```

- ustawienia domyślnego edytora,

```
git config --global core.editor nano
```

- kolorowanie konsoli

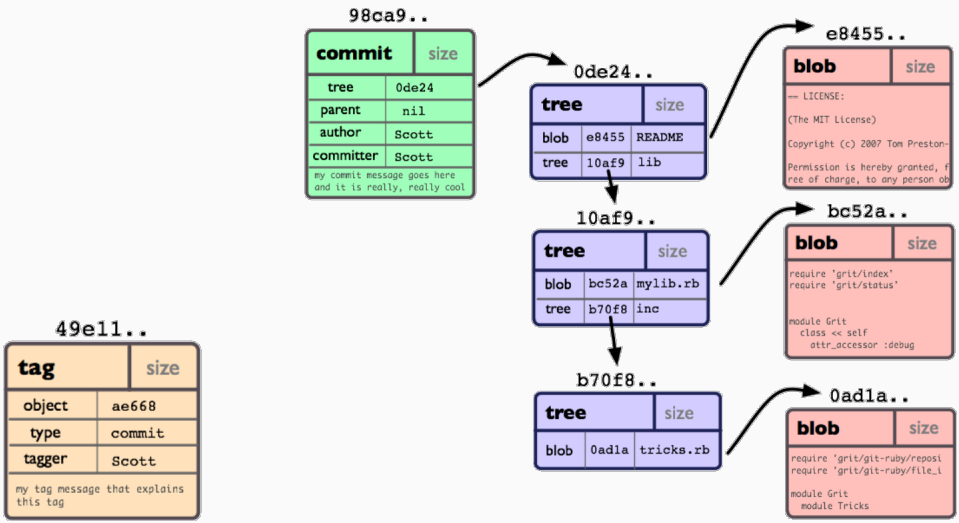
```
git config --global color.ui auto
```

Terminologia GIT'a

Używając GIT'a należy wspomnieć o używanej terminologii:

- **Commit** – stan repozytorium w danym momencie, oznaczony przy pomocy wyliczonego SHA1. Cała historia projektu składa się z commitów
- **Branch** – równoległa gałąź projektu rozwijana oddzielnie od głównej („ruchomy wskaźnik" na commit),
- **Tag** – marker konkretnej wersji („nieruchomy wskaźnik" na commit) projektu,
- **Blob** – zawiera zawartość pliku bez żadnej dodatkowej struktury,
- **Tree** – reprezentuje stan pojedynczego katalogu (lista obiektów blob oraz zagnieżdżonych obiektów tree),
- **Working Dir** – katalog roboczy na którym pracujemy,
- **Index** – rodzaj „cache", czyli miejsca gdzie trzymane są zmiany do commita,
- **Master (Main) Branch** – główny branch z którym łączymy (ang. *merge*) nasze zmiany przed wysłaniem do zdalnego repozytorium.

Obiekty GIT’a



Struktura zmian w GIT'cie

- Przechowywanie zawartości projektu jako „snapshot'ów",
- Kompresowanie zawartości projektu.

Praca z GIT

Tworzenie repozytorium

Repozytorium GIT'a może zostać założone w dowolnym katalogu. Po przejściu do katalogu, możemy stworzyć w nim lokalne repozytorium. Do tego celu należy posłużyć się poleceniem:

```
1  git init
```

Innym sposobem zainicjowania repozytorium jest pobranie jego zawartości z repozytorium zdalnego:

```
1  git clone https://github.com/rroszczyk/Python.git
```

Sprawdzanie repozytorium

Do sprawdzenie statusu repozytorium służą odpowiednio polecenia:

```
1  git status
2  git log
```

Pierwsze z nich podaje nam informacje na temat tego co się w repozytorium zmieniło od ostatniego commit'a.
Drugie polecenie pozwala nam na przejrzenie całej historii zmian w repozytorium.

Dodawanie plików

Aby dodać do repozytorium plik należy wydać jedno z poleceń:

```
1  git add nazwa_pliku
2  git add *
```

W pierwszym przypadku zostanie do repozytorium zostanie dodany plik o określonej nazwie.
Drugie polecenie pozwala na dodanie do repozytorium wszystkich plików o nazwach zgodnych z wpisaną maską. Aby zapobiec dodawaniu niechcianych plików należy w pliku .gitignore zamieścić listę plików które mają być pominięte przy dodawaniu.

Zatwierdzanie zmian

Aby zatwierdzić zmiany w repozytorium należy wykonać commit. Do tego celu wydajemy polecenie:

```
1   git commit -m "komentarz"
```

Komentarz wpisany w poleceniu zostanie dodany do commit'u i będzie widoczny dla innych użytkowników. Jeśli komentarz jest krótki możemy posłużyć się parametrem „-m". Jeśli parametr zostanie pominięty uruchomi się edytor tekstu przy pomocy którego należy wprowadzić komentarz. Możliwe jest wykorzystywanie składni markdown.

Praca z repozytoriami lokalnymi

Do podstawowych operacji GIT'a należą:

- **git init** – stworzenie nowego repozytorium,
- **git add** – dodanie zawartości pliku do Index'u,
- **git rm** – usuwa plik z indexu (plik zniknie z working directory po commit'ie),
- **git mv** – przenosi plik w ramach repozytorium,
- **git commit** – zapisuje zmiany do repozytorium lokalnego
- **git log** – wyświetla logi z commit'ów,
- **git show** – wyświetla obiekt,
- **git status** – pokazuje status katalogu roboczego i poczekalni,
- **git config** – pobiera i ustawia opcje globalne GIT'a lub tylko repozytorium

Praca z repozytoriami zdalnymi

Do operacji na zdalnych repozytoriach mamy dostępne następujące polecenia:

- **git clone** – pobiera zdalne repozytorium do podanego folderu,
- **git fetch** – pobiera obiekty i wskaźniki z innego repozytorium,
- **git pull** – pobiera i integruje obiekty i wskaźniki z innego repozytorium,
- **git push** – aktualizuje zdalne repozytorium o wskaźniki i powiązane obiekty.

GIT nie potrzebuje serwera, możliwe jest sklonowanie lokalnych repozytoriów. Rozwiązanie to jest szczególnie przydatne gdy pracujemy w pojedynkę, a chcemy mieć zachowaną historię zmian i możliwości GIT'a.

Praca z gałęziami

Oprócz operacji na plikach można również wykonywać operacje na gałęziach:

- **git branch** – do zarządzania branch'ami,
- **git checkout** – przełączanie się między branch'ami,
- **git merge** – łączy podane branch'e,
- **git rebase** – zmienia punkt startu dla branch'a,
- **git reset** – przywraca stan katalogu roboczego,
- **git stash** – zapisuje/odczytuje zmiany z przestrzeni tymczasowej (rodzaj schowka),
- **git gc** – porządkowanie i optymalizacja repozytorium

Operacje na znacznikach – tag

Utworzenie znacznika:

```
1   git tag v1.00
```

Wypisanie znaczników:

```
1   git tag
```

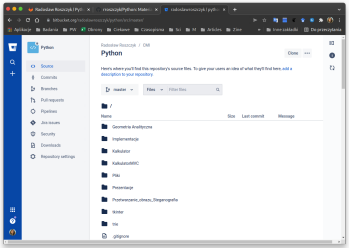
Aktualizacja repozytorium o znaczniki:

```
1   git push -tag
```

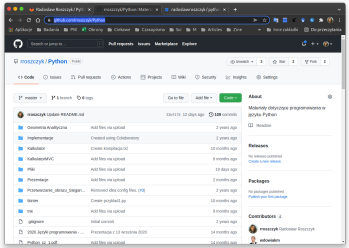
Usunięcie znacznika:

```
1   git tag -d v1.00
```

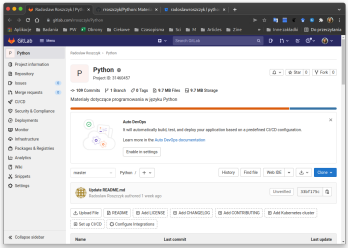
Systemy zarządzania repozytoriami GIT



Rysunek 1: BitBucket



Rysunek 2: GitHub

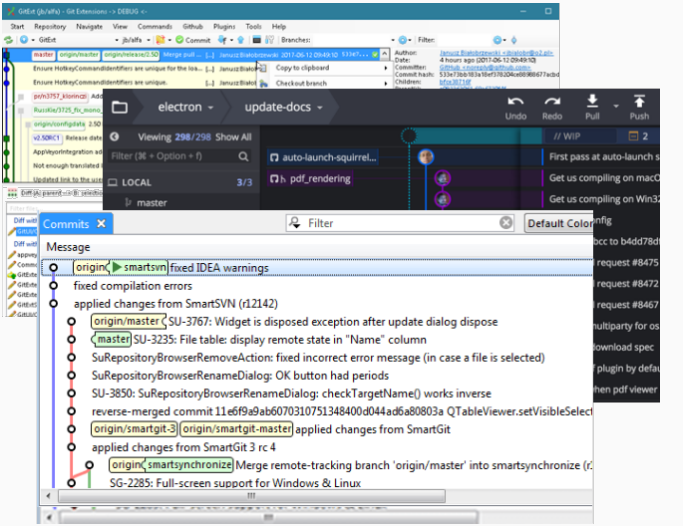


Rysunek 3: GitLab

Lokalna alternatywa w postaci Gogs.

Narzędzia GUI dla GIT

- SmartGit
- GitKraken
- Git-cola
- Giggle
- Gitg
- Git GUI
- GitForce
- SourceTree



SVN vs GIT

- System scentralizowany - SVN
- Globalne repozytorium danych
- Prosty w obsłudze
- Stały dostęp do centralnego repozytorium
- Wsparcie dla wszystkich platform oraz IDE
- Powolny
- Brak restrykcji
- Intuicyjna numeracja rewizji
- Operacje na fragmentach repozytorium (checkout)

- System rozproszony - GIT
- Lokalne oraz globalne repozytorium
- Skomplikowany w obsłudze
- Nie wymaga stałego dostępu do głównego repozytorium
- Słabe wsparcie platformy Windows
- Szybki
- Numeracja rewizji poprzez SHA-1
- Operacje na całym repozytorium (checkout)
- Mniejszy rozmiar repozytorium
- Łatwiejsze łączenie gałęzi (merge)

Podsumowanie

Co nam daje GIT?

- Tworzenie i łączenie (merge) branch'y jest szybkie i łatwe,
- Możliwość wymiany kodu pomiędzy zdalnymi repozytoriami, (Udostępnienie bezpośrednio dla kogoś swojego prywatnego branch'a)
- Łatwe prototypowani – prywatne branche,
- można w łatwy sposób dostosować repozytoria do swojego procesu wytwarzania kodu,
- możliwość automatycznego migrowania repozytorium SVN do GIT'a,
- projekt OpenSource,
- rosnąca popularność dzięki której pojawaia się coraz więczej narzędzi/pluginów do/dla GIT'a.

Pytania?

Piaskownica

Miejsce w którym można poćwiczyć pracę z GIT:

https://learngitbranching.js.org/

Linkowisko

- GIT
  - *https://git-scm.com/*
- Repozytoria kodu
  - *https://gogs.io/*
  - *https://github.com/*
  - *https://gitlab.com/*
  - *https://bitbucket.org/*
- Narzędzia do zarządzania repozytoriami
  - *https://gitkraken.com/*
  - *https://sourcetreeapp.com/*
  - *https://syntevo.com/smartgit/*