Systemy kontroli wersji Git

Grupa Pomarańczowa

Wprowadzenie do kontroli wersji

Kontrola wersji to proces, który pozwala na śledzenie zmian w plikach, cofanie się do wcześniejszych wersji oraz współpracę nad kodem w sposób uporządkowany. Jest to szczególnie przydatne dla programistów, ale także dla osób pracujących nad dokumentami czy projektami graficznymi.

Funkcje kontroli wersji

- Śledzenie zmian w kodzie lub plikach.
- Przechowywanie historii zmian w projekcie.
- Przywracanie wcześniejszych wersji plików.
- Bezpieczna współpraca zespołowa.
- Zmniejszenie ryzyka utraty danych.

Rodzaje systemów kontroli wersji

Podział ze względu na architekturę:

- SCM (Source Code Management) – zarządzanie kodem źródłowym (np.: IBM Rational ClearCase).
- CVCS (Centralized Version Control System) – systemy scentralizowane (np.: Subversion).
- DVCS (Distributed Version Control System) – systemy rozproszone (np.: Git, Mercurial).

Rodzaje systemów kontroli wersji

Podział ze względu na licencje:

- Open Source – kod źródłowy oprogramowania jest dostępny publicznie (np.: Git, Mercurial, Subversion).
- Komercyjne – oprogramowanie, które wymaga zakupu licencji lub subskrypcji (np.: IBM Rational ClearCase).
- SaaS (Software as a Service) – usługa oparta na chmurze, gdzie oprogramowanie jest udostępniane jako usługa online (np.: GitHub, GitLab, Bitbucket).

SCM (Source Code Management)

SCM to szerokie pojęcie obejmujące metody i narzędzia służące do zarządzania kodem źródłowym w projektach programistycznych. Jego główne funkcje to:

- Przechowywanie kodu w repozytorium.
- Śledzenie zmian w plikach i kodzie.
- Wersjonowanie plików.
- Umożliwienie współpracy wielu programistów.
- Zabezpieczenie danych przed przypadkową utratą.

CVCS (Centralized Version Control System)

CVCS to scentralizowany system kontroli wersji, w których repozytorium jest przechowywane na jednym, centralnym serwerze. Programiści pobierają najnowszą wersję kodu, wprowadzają zmiany i przesyłają je z powrotem na serwer. Przykłady:

- **Subversion** – popularny w korporacjach i projektach open source.
- **Perforce** – stosowany w dużych firmach i projektach, zwłaszcza w branży gier komputerowych.

Zalety CVCS:

- Łatwa administracja i kontrola dostępu.
- Wymuszenie spójności kodu.
- Mniejsza złożoność w porównaniu do systemów rozproszonych.

Wady CVCS:

- Brak dostępu do historii zmian bez połączenia z serwerem.
- Wysokie ryzyko utraty danych przy awarii serwera.
- Możliwość spowolnienia pracy przy dużej ilości użytkowników.

DVCS (Distributed Version Control System)

DVCS to rozproszony system kontroli wersji, w których każdy użytkownik posiada pełną kopię repozytorium, w tym pełną historię zmian. Programiści mogą pracować lokalnie, a zmiany synchronizować z innymi użytkownikami. Przykłady:

- **Git** – najpopularniejszy DVCS, stosowany w projektach open source i w firmach IT.
- **Mercurial** – szybszy w działaniu i prostszy w obsłudze niż Git.

Zalety DVCS:

- Pełna historia zmian dostępna lokalnie.
- Mniejsze ryzyko utraty danych przez kopie zapasowe na komputerach użytkowników.
- Umożliwia pracę offline i późniejszą synchronizację zmian.

Wady DVCS:

- Większa złożoność w porównaniu do systemów scentralizowanych.
- Większe zużycia miejsca na dysku lokalnym.
- Potrzeba synchronizacji z innymi repozytoriami.
- Początkowa nauka może być trudna.

Popularne systemy kontroli wersji

1. RCS (Revision Control System)

Jeden z najstarszych systemów kontroli wersji, wywodzący się ze środowiska Unix. Działa na poziomie pojedynczych, lokalnych plików.

Zalety:

- Prosty w użyciu do zarządzania wersjami pojedynczych plików.
- Idealny do prostych projektów lokalnych.

Wady:

- Brak wsparcia dla projektów zespołowych.
- Nie oferuje funkcji pracy na gałęziach.
- Nie obsługuje historii całego projektu, tylko poszczególnych plików.

2. CVS (Concurrent Versions System)

Scentralizowany system kontroli wersji. Stworzony został na bazie RCS. Wspiera pracę zespołową.

Zalety:

- Pozwala na współpracę wielu programistów nad jednym projektem.
- Umożliwia równoczesną edycję plików przez wielu użytkowników.

Wady:

- Operacje na repozytorium nie są atomowe.
- Stworzenie nowej gałęzi pochłania dużo miejsca na dysku.
- Brak wersjonowania dla zmian nazw i usuwania plików.

3. Subversion (SVN)

Scentralizowany system kontroli wersji. Powstał w celu zastąpienia CVS. Wspiera zarządzanie całymi katalogami i projektami.

Zalety:

- Pełne wsparcie dla historii projektów.
- Możliwość pracy na gałęziach i tagowania wersji.

Wady:

- Wymaga stałego połączenia z serwerem.
- Przy braku połączenia, użytkownik traci dostępu do historii wersji.

4. Mercurial (Hg)

Rozproszony, międzyplatformowy system kontroli wersji napisany głównie w Pythonie. Kładzie nacisk na prostotę i szybkość działania.

Zalety:

- Pełna historia zmian dostępna lokalnie.
- Prostszy w obsłudze niż Git.
- Szybki, nawet przy dużych projektach.

Wady:

- Mniejsza liczba dostępnych narzędzi i zasobów edukacyjnych.
- Mniej elastyczny w bardziej zaawansowanych scenariuszach prac.

5. Git

Rozproszony system kontroli wersji. Używany jest w projektach open source oraz komercyjnych.

Zalety:

- Pełna historia projektu dostępna lokalnie.
- Doskonałe wsparcie dla pracy na gałęziach i scalania.
- Umożliwia pracę offline.
- Duża społeczność i szerokie wsparcie narzędzi (np.: GitHub, GitLab).

Wady:

- Większa złożoność początkowa dla nowych użytkowników.
- Wymaga więcej zasobów na dyskach lokalnych (pełna kopia repozytorium lokalnie).

GIT

Najpopularniejszy system kontroli wersji

Git to rozproszony system kontroli wersji stworzony przez Linusa Torvaldsa w 2005 roku. Jest szeroko stosowany w świecie programowania dzięki swojej wydajności, elastyczności i łatwości użycia.

Kim jest Linus Torvalds?

Linus Torvalds urodził się 28 grudnia 1969 roku w Helsinkach, Finlandia. Jest fińskim programistą, najbardziej znanym jako twórca jądra systemu operacyjnego Linux oraz Gita. Od wielu lat pracuje nad rozwojem jądra Linux w ramach organizacji Linux Foundation.

Torvalds jest znany ze swojego bezpośredniego stylu komunikacji oraz ogromnego wkładu w rozwój oprogramowania open source. Jego praca przyczyniła się do rozwoju nowoczesnych technologii, a Git stał się standardem w zarządzaniu wersjami oprogramowania na całym świecie.

Krótka historia Gita

Prace nad Gitem rozpoczęły się w 2005 roku przez Linusa Torvaldsa, kiedy system kontroli wersji BitKeeper przestał być darmowy dla projektów o otwartym kodzie źródłowym (open source), jakim był Linux. Torvalds szukał zatem alternatywy, która umożliwiałaby rozproszoną kontrolę wersji, szybkie operacje oraz wsparcie dla nieliniowego kodu, ale niestety inne systemy kontroli wersji nie spełniały jego założeń. Dlatego Torvalds postanowił sam napisać własny system, czego efektem był właśnie Git. Pracę rozpoczął na początku kwietnia, a już w połowie czerwca Linux był hostowany przez Gita.

Początkowo Git był bardzo surowym narzędziem, przeznaczonym głównie dla zaawansowanych programistów. W kolejnych latach powstały graficzne interfejsy użytkownika (GUI) oraz platformy hostujące repozytoria, takie jak GitHub (2008), GitLab (2011) czy Bitbucket (2010), które znacznie ułatwiły pracę z Git. Dzięki temu Git w krótkim czasie zdobył dużą popularność wśród społeczności programistów.

Charakterystyka Gita

Git jest rozproszonym systemem kontroli wersji, który umożliwia programistom zarządzanie kodem źródłowym oraz śledzenie historii zmian w projekcie. Git jest szeroko stosowany zarówno w projektach open source, jak i komercyjnych, a także stanowi fundament dla popularnych platform takich jak GitHub, GitLab i Bitbucket.

Zalety Gita

- Rozproszony model repozytoriów:
  - Każdy użytkownik posiada pełną kopię repozytorium z całą historią zmian, co umożliwia pracę offline.
  - Zmniejsza ryzyko utraty danych dzięki lokalnym kopiom.
- Współpraca i gałęzie (branching):
  - Git wspiera nieliniowy rozwój kodu, co umożliwia tworzenie wielu niezależnych gałęzi.
  - Ułatwia pracę zespołową i wdrażanie nowych funkcji bez zakłócania głównej wersji kodu.
  - Proces scalania (merging) pozwala na łatwe integrowanie zmian.
- Szybkość i wydajność:
  - Większość operacji odbywa się lokalnie, co zwiększa prędkość działania.
  - Git jest zoptymalizowany do obsługi dużych projektów.
- Elastyczność:
  - Git umożliwia korzystanie z różnych przepływów pracy (workflow) np.: Feature Branch, Gitflow, Forking Workflow.
- Szerokie wsparcie i społeczność:
  - Git posiada bogatą dokumentację i dużą społeczność użytkowników, co ułatwia naukę i rozwiązywanie problemów.

Wady Gita

- Krzywa uczenia się:
  - Git oferuje zaawansowane funkcje, ale nowi użytkownicy mogą początkowo mieć trudności z bardziej skomplikowanymi operacjami, takimi jak rebase, cherry-pick czy merge conflicts.
- Większe zużycie zasobów:
  - Pełna kopia repozytorium zajmuje więcej miejsca na dysku w porównaniu do systemów scentralizowanych.
- Potencjalne problemy z synchronizacją:
  - W przypadku zespołów pracujących zdalnie, synchronizacja lokalnych repozytoriów może wymagać dodatkowej uwagi i komunikacji między użytkownikami, szczególnie przy rozwiązywaniu konfliktów.
- Ryzyko utraty danych przy nieprawidłowym użyciu:
  - Zaawansowane operacje jak reset czy rebase mogą prowadzić do utraty zmian, jeśli nie są używane prawidłowo.

TERMINOLOGIA GITA

Najważniejsze terminy związane z Git'em

- **Working Directory (Katalog roboczy):**
  - Katalog na lokalnym komputerze, w którym przechowywane są pliki projektu. Jest to miejsce, gdzie programista dokonuje zmian przed zapisaniem ich w repozytorium.
- **Repository (Repozytorium):**
  - Miejsce, w którym Git przechowuje wszystkie pliki projektu oraz pełną historię ich zmian. Może być lokalne (na komputerze) lub zdalne (np.: na GitHubie).
- **Branch (Gałąź):**
  - Niezależna linia rozwoju kodu. Pozwala na pracę nad nowymi funkcjami lub poprawkami bez wpływu na główną wersję kodu (np.: main lub master). Popularne gałęzie to:
    - feature - dla nowych funkcji.
    - hotfix - dla szybkich poprawek błędów.
    - develop - gałąź rozwoju, często używana w metodologii GitFlow.
- **Commit:**
  - Zapisanie zmian z katalogu roboczego do lokalnego repozytorium. Commit zawiera wiadomość opisującą zmiany, co ułatwia śledzenie historii projektu.

Najważniejsze terminy związane z Git'em

- **Merge (Scalanie):**
  - Połączenie zmian z jednej gałęzi do drugiej. Najczęściej używane do wdrażania nowych funkcji do głównej gałęzi projektu.
- **Checkout:**
  - Przełączenie się na inną gałąź lub wersję projektu. Umożliwia powrót do starszych commitów lub rozpoczęcie pracy na nowej gałęzi.
- **Hotfix:**
  - Szybka poprawka błędu w głównej wersji kodu. Gałęzie hotfix są zwykle tworzone w sytuacjach kryzysowych i po rozwiązaniu problemu szybko scalane do main lub master.
- **Feature Branch:**
  - Gałąź dedykowana pracy nad konkretną funkcjonalnością. Po zakończeniu prac i przetestowaniu jest scalana z główną gałęzią projektu.
- **Clone (Klonowanie):**
  - Pobranie zdalnego repozytorium na lokalny komputer. Tworzy pełną kopię projektu razem z całą historią zmian.

Najważniejsze terminy związane z Git'em

- **Push:**
  - Wysłanie commitów z lokalnego repozytorium do repozytorium zdalnego (np.: GitHub, GitLab). Pozwala innym członkom zespołu na dostęp do najnowszych zmian.
- **Pull:**
  - Pobranie najnowszych zmian ze zdalnego repozytorium do lokalnego komputera. Często łączy się z automatycznym scalaniem (merge).
- **Rebase:**
  - Zamiana podstawy gałęzi na inny commit. Pomaga w utrzymaniu czystej historii projektu, ale wymaga ostrożności, aby uniknąć konfliktów.
- **Stash:**
  - Tymczasowe zapisanie zmian z katalogu roboczego bez commitowania ich. Umożliwia przełączenie się na inną gałąź bez utraty pracy.
- **Reset:**
  - Cofnięcie stanu repozytorium do wcześniejszego commita. Może usunąć zmiany lokalne lub jedynie wycofać commity z historii.

Najważniejsze terminy związane z Git'em

- **Conflict (Konflikt):**
  - Sytuacja, w której Git nie może automatycznie scalić zmian z dwóch różnych gałęzi. Wymaga ręcznego rozwiązania konfliktu przez programistę.
- **Fork:**
  - Kopia zdalnego repozytorium utworzona na własnym koncie (np.: na GitHubie). Umożliwia modyfikowanie projektu open source i proponowanie zmian autorowi.
- **Pull Request / Merge Request:**
  - Prośba o scalenie zmian z jednej gałęzi lub repozytorium do drugiego. Popularne na platformach takich jak GitHub czy GitLab jako forma przeglądu kodu przez zespół.
- **Blob (Binary Large Object):**
  - Przechowuje zawartość pojedynczego pliku jako ciąg znaków (bez nazwy pliku, metadanych).
- **Tree (Drzewo):**
  - Reprezentuje katalog, zawiera referencje do blobów (pliki) i innych drzew (podkatalogi).
- **Tag:**
  - Etykieta przypisana do konkretnego commitu, często używana do oznaczania wersji.

PODSTAWOWE KOMENDY GITA

KONFIGURACJA

Ustawienie nazwy użytkownika i e-maila

```
$ git config --global user.name "Twoje Imię"
$ git config --global user.email "twojemail@example.com"
```

INICJALIZACJA REPOZYTORIUM

Tworzenie nowego repozytorium

```
$ git init
```

Pobranie istniejącego repozytorium

```
$ git clone https://github.com/uzytkownik/repo.git
```

ŚLEDZENIE ZMIAN

Sprawdzenie statusu repozytorium

$ git status

Dodanie plików do śledzenia

$ git add nazwa_pliku

$ git add . # Dodanie wszystkich zmian

Zatwierdzenie zmian

$ git commit -m "Opis zmian"

PRACA Z GAŁĘZIAMI (BRANCHING)

Tworzenie nowej gałęzi

$ git branch nowa-galaz

Przełączanie się na nową gałąź

$ git checkout nowa-galaz

Alternatywnie: tworzenie i przełączanie w jednej komendzie

$ git checkout -b nowa-galaz

AKTUALIZACJA I SYNCHRONIZACJA

Pobranie najnowszych zmian z repozytorium zdalnego

$ git pull origin main

Wysłanie lokalnych zmian do repozytorium zdalnego

- $ git push origin main

SCALANIE ZMIAN

Scalanie gałęzi do głównej linii kodu

$ git merge nowa-galaz

WYCOFYWANIE ZMIAN

Cofnięcie zmian przed commitem

```
$ git checkout -- nazwa_pliku
```

Wycofanie ostatniego commita

```
$ git reset --soft HEAD~1  # Zachowuje zmiany w plikach
$ git reset --hard HEAD~1  # Usuwa zmiany
```

OBIEKTY GITA

Czym są obiekty w Git?

Obiekty w Git to podstawowe jednostki przechowywania danych.

Są cztery typy obiektów:

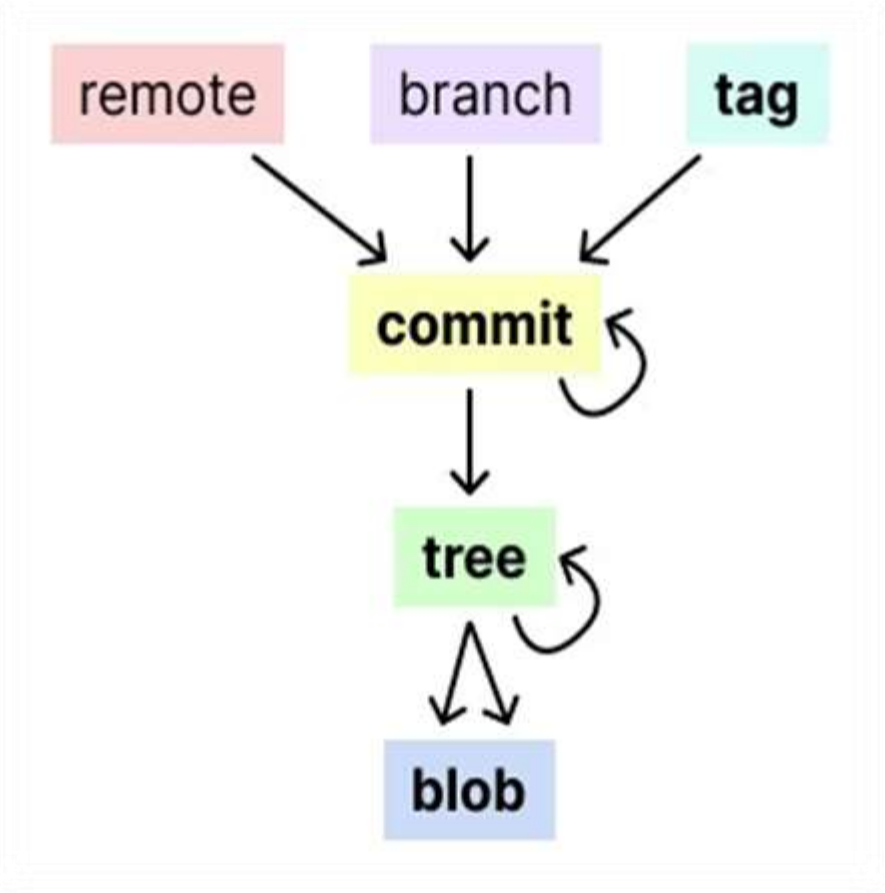
- **Blob (Binary Large Object):** Przechowuje zawartość plików.
- **Tree:** Struktura katalogów, wskazuje na bloby i inne drzewa.
- **Commit:** Reprezentuje zapis zmian, wskazuje na obiekt tree.
- **Tag:** Oznaczanie konkretnego commita przyjazną nazwą.

Git obsługuje cztery typy obiektów: commits, trees, blobs, tags.

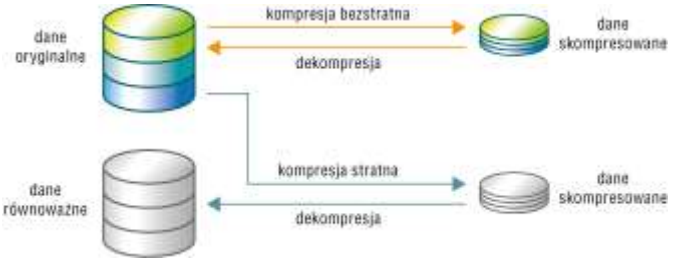
Te obiekty są niezmienne, co oznacza, że nie można ich zmienić po utworzeniu.

Każdy obiekt jest identyfikowany przez 40-znakowy skrót SHA. Jest to prosty sposób tworzenia globalnie unikalnych identyfikatorów w sposób rozproszony.

W .git/objects/, obiekty są zapisywane w katalogu nazwanym zgodnie z pierwszymi dwoma znakami skrótu SHA, a nazwa pliku to pozostałe 38.



Kompresja



Git stosuje zaawansowane techniki kompresji, aby efektywnie zarządzać i przechowywać wersje plików w repozytorium. Główne mechanizmy kompresji w Git to:

- **Kompresja zlib dla obiektów luźnych (loose objects):** Każdy nowo dodany plik jest kompresowany przy użyciu algorytmu zlib i przechowywany jako oddzielny obiekt w katalogu .git/objects.
- **Pliki pakietów (packfiles):** Aby zredukować ilość miejsca zajmowanego przez obiekty, Git grupuje je w tzw. pliki pakietów. Proces ten polega na znalezieniu podobnych obiektów i zapisaniu jednego z nich w całości, a pozostałych jako różnice (deltas) względem niego.
- **Kompresja delta:** Git przechowuje tylko różnice między wersjami plików, co pozwala na znaczne oszczędności miejsca.

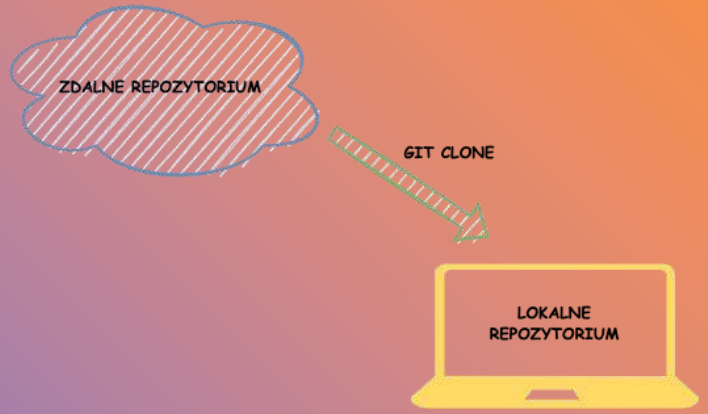
ZDALNE REPOZYTORIUM GIT

Co to jest zdalne repozytorium Git?

Zdalne repozytorium to centralne miejsce przechowywania kodu, które umożliwia współpracę wielu programistów.

Funkcje:

- Przechowywanie kodu w chmurze.
- Śledzenie historii zmian.
- Ułatwienie pracy zespołowej.



GitHub

Krótki opis: Najpopularniejsza platforma do hostowania repozytoriów Git.

Główne funkcje:

- Pull requests, issues, actions (CI/CD).
- Integracja z wieloma narzędziami DevOps.
- Duża społeczność open source.

Zastosowania: Projekty open source, jak i komercyjne.

GitLab

Krótki opis: Platforma do zarządzania kodem i DevOps, oferująca zarówno wersję chmurową, jak i możliwość hostowania lokalnego.

Główne funkcje:

- Kompletny pipeline DevOps (CI/CD, monitoring, zarządzanie projektami).
- Zaawansowane zarządzanie uprawnieniami.
- Wersja self-hosted z pełnym dostępem do kodu źródłowego.

Zastosowania: Firmy potrzebujące prywatnego hostingu i rozbudowanych narzędzi DevOps.

Bitbucket

Krótki opis: Platforma do hostowania repozytoriów Git, szczególnie popularna w firmach korzystających z Jira i Confluence.

Główne funkcje:

- Integracja z ekosystemem Atlassian.
- Obsługa zarówno Git, jak i Mercurial (do 2020 r.).
- Pipeline CI/CD.

Zastosowania: Firmy zintegrowane z Atlassian oraz zespoły deweloperskie.

Porównanie GitHub, GitLab, Bitbucket

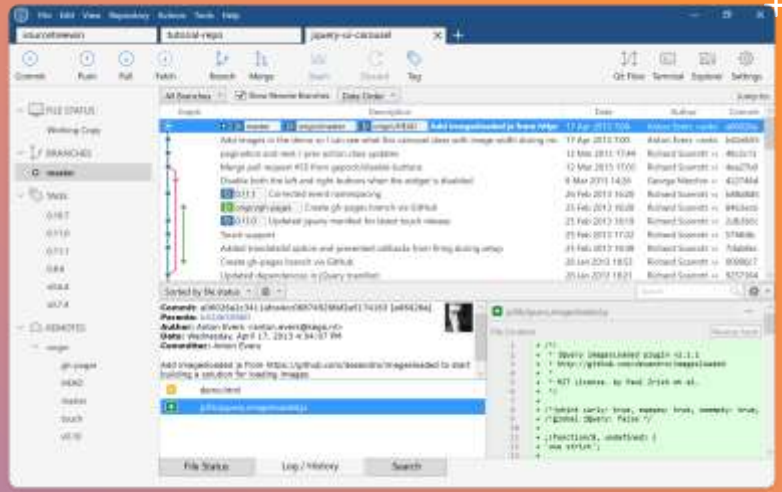
Cechy	GitHub	GitLab	Bitbucket
Wersja darmowa	✓	✓	✓
Hostowanie lokalne	✕	✓	✓
CI/CD	✓ (Actions)	✓ (wbudowane)	✓ (Pipelines)
Integracja z Jira	Możliwa	Możliwa	✓ (natywna)
Społeczność open source	✓	✓	✕

NARZĘDZIA DESKTOPOWE DO GITA

Git GUI

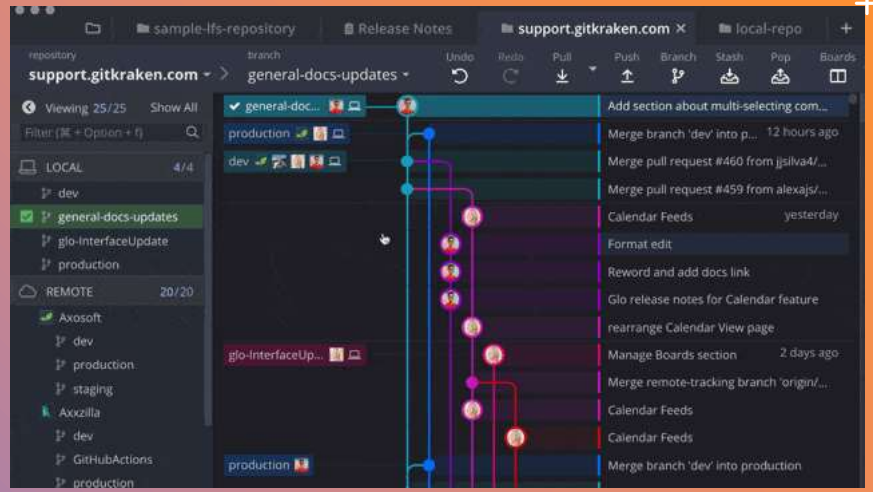
- Proste narzędzie dostarczane z Git.
- Umożliwia wykonywanie podstawowych operacji (commit, push, pull) w interfejsie graficznym.
- Intuicyjny interfejs idealny dla początkujących użytkowników Gita.
- Brak zaawansowanych funkcji, ale doskonałe do małych projektów.

Sourcetree



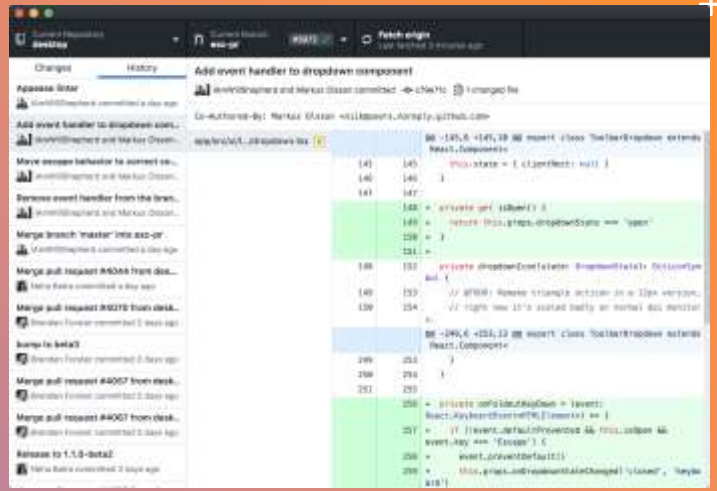
- Narzędzie od Atlassian, dostępne za darmo.
- Wspiera zarówno Git, jak i Mercurial.
- Rozbudowane funkcje, w tym podgląd drzewka commitów i łatwe zarządzanie branchami.
- Integracja z Bitbucket oraz Jira, co ułatwia pracę zespołową.

GitKraken



- Dostępny w wersji darmowej oraz płatnej (wspiera funkcje premium).
- Przyjazny interfejs, wizualizacja historii commitów.
- Wspiera GitFlow oraz Git LFS (Large File Storage).
- Integracje z GitHub, GitLab, Bitbucket oraz innymi zdalnymi repozytoriami.

GitHub Desktop



- Dedykowane narzędzie do zarządzania repozytoriami na GitHubie.
- Intuicyjny interfejs, który ułatwia tworzenie commitów, synchronizację zmian i rozwiązywanie konfliktów.
- Świetne dla początkujących, oferuje pomocne podpowiedzi i instrukcje.
- Możliwość pracy z wieloma repozytoriami jednocześnie.

Podsumowanie

Git to potężne narzędzie do zarządzania wersjami kodu i pracy zespołowej. Opanowanie podstawowych komend i najlepszych praktyk ułatwi organizację pracy i zwiększy efektywność w każdym projekcie.

DZIĘKUJEMY ZA UWAGĘ