# GIT

# Systemy kontroli wersji

System kontroli wersji to narzędzie, które pozwala nam zachowywać historię plików, wymieniać je z innymi członkami zespołu oraz dbać o spójność plików projektu.


**GIT** to **rozproszony** system kontroli wersji, świetnie nadający się do zarządzania przede wszystkim plikami źródłowymi programu.

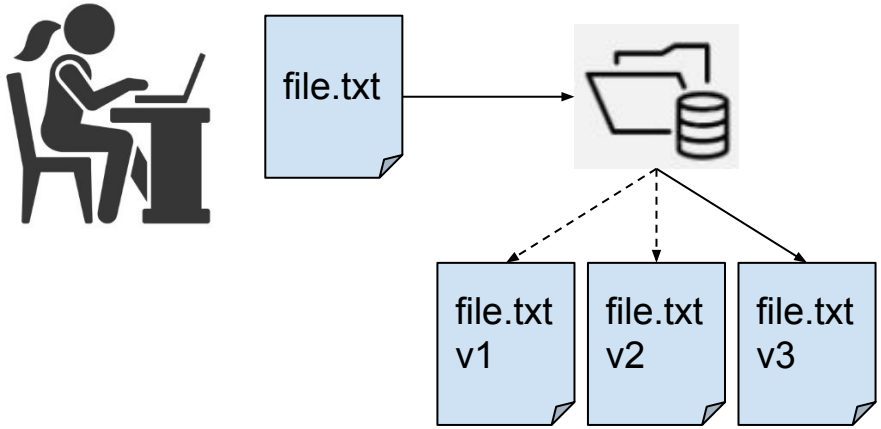
# Repozytorium

Repozytorium to miejsce, gdzie przechowujemy kolejne wersje plików projektu, nad którym pracujemy.

Repozytorium w przypadku systemu Git jest tworzone lokalnie.

W systemie plików jest przechowywane w katalogu .git, natomiast tworzymy je poleceniem

```
git init
```



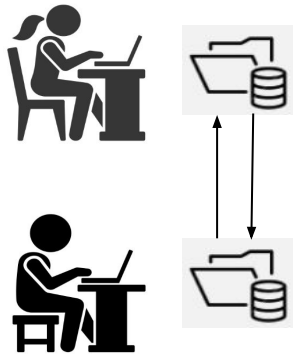
# GIT - budowa i architektura

Git jest tzw. rozproszonym systemem kontroli wersji.

Każdy z członków zespołu pracujących nad projektem ma swoje lokalne repozytorium.

Możemy utworzyć kopię repozytorium innego użytkownika wykonując polecenie

```
git clone
```



Kolejne zmiany ściągamy ze zdalnego repozytorium poleceniami

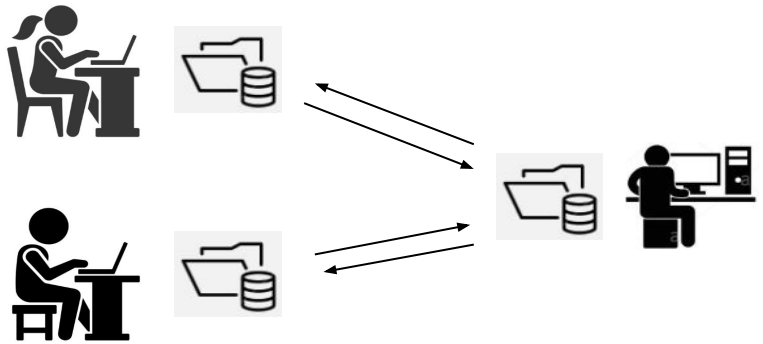
```
git fetch
git merge
```

Nasze zmiany możemy “wypchnąć” do zdalnego repozytorium przy pomocy:

```
git push
```

# Repozytorium główne (?)

Chociaż w systemie Git każde repozytorium może być traktowane równorzędnie, stosuje się również scenariusze, gdzie mamy jedno wydzielone repozytorium główne.



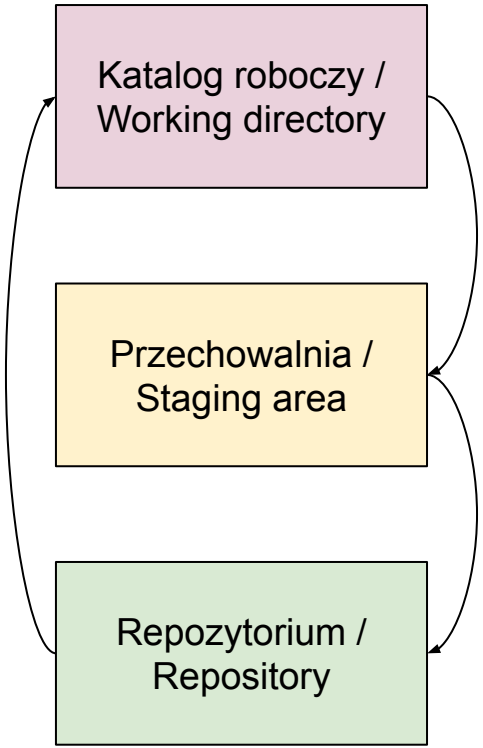
Operacje na nim mogą podlegać ochronie, tj. użytkownicy ze zwykłym poziomem uprawnień nie mogą wykonywać samodzielnie operacji push. Mogą jedynie wystawić tzw. **merge request** (lub **pull request** w GitHubie).

Oznacza to wysłanie prośby do administratora:

*Weź proszę moje zmiany, zrób przegląd (code review), i połącz je z głównym repozytorium*

# Praca lokalna

1. Zmieniając zawartość pliku będącego w repozytorium, lub tworząc nowy plik - tworzymy nową wersję pliku.



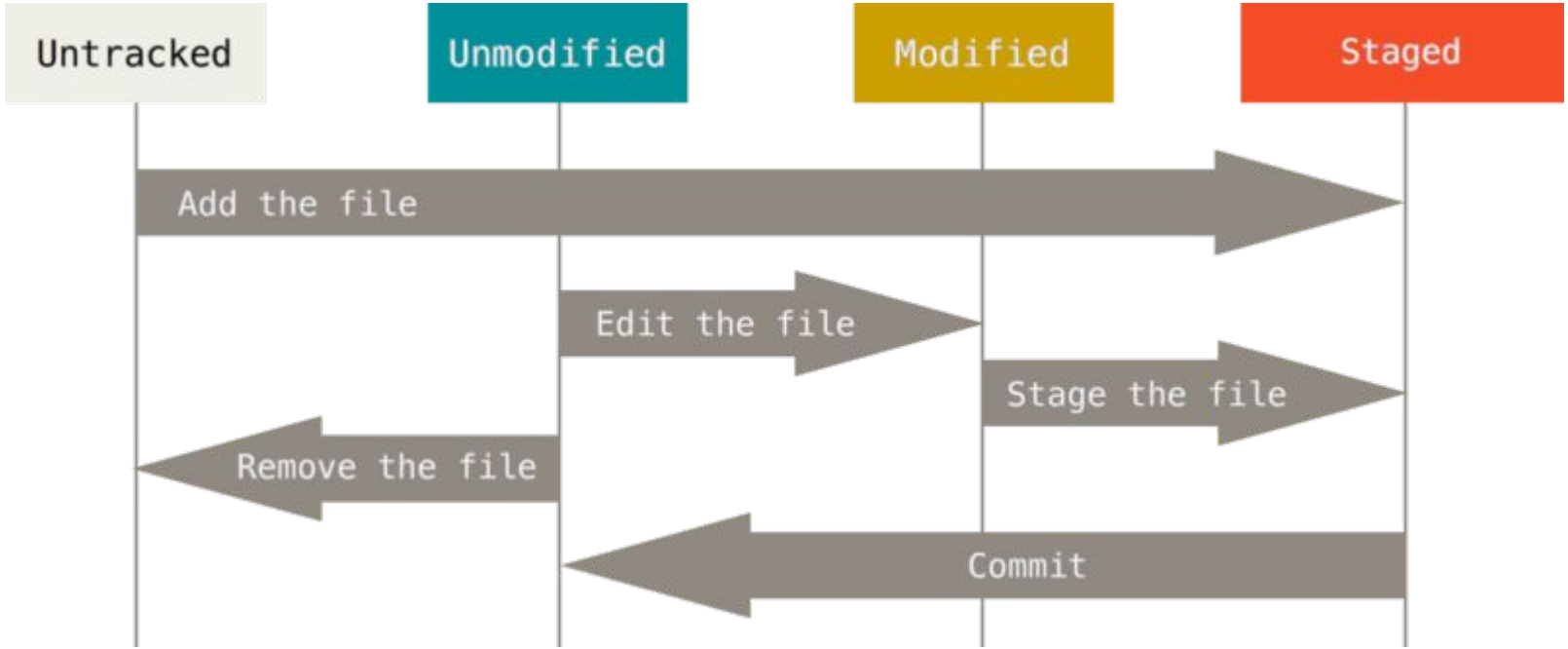

2. Zmieniony plik możemy dołączyć do pakietu zmian, które chcemy dołączyć do repozytorium. Robimy to przy pomocy polecenia:

```
git add .
```

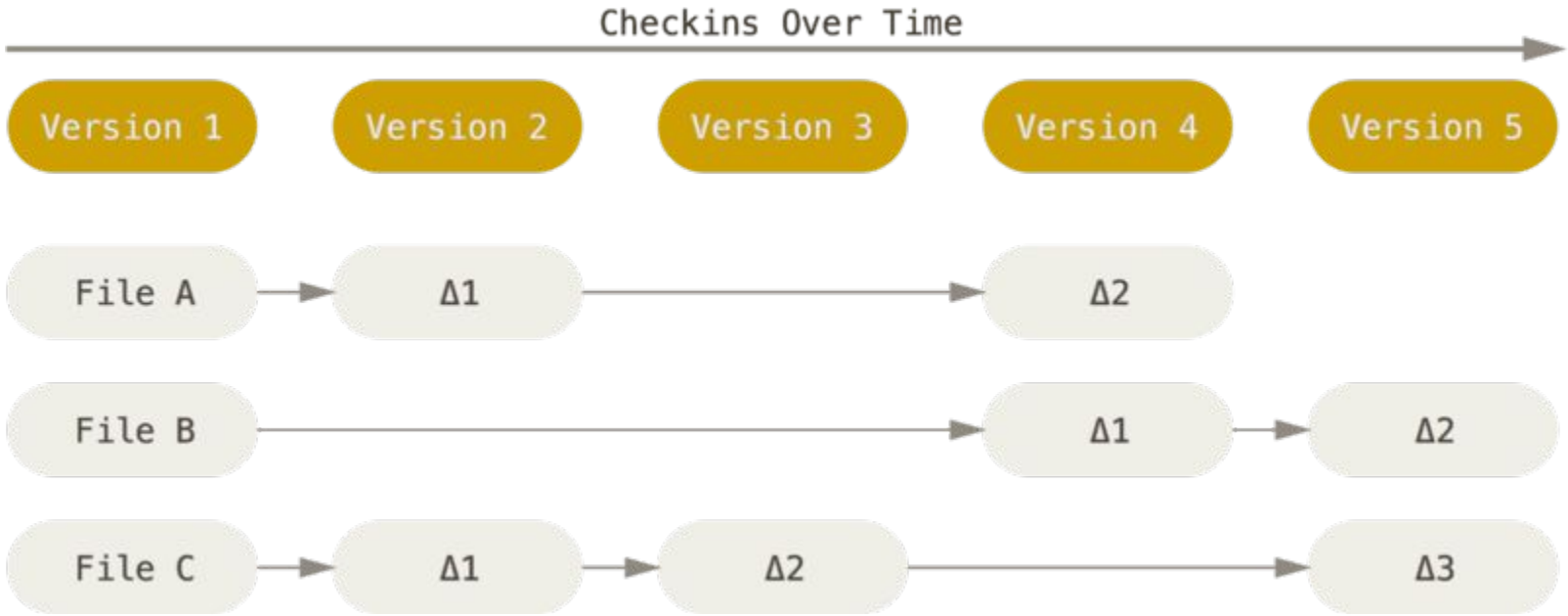
3. Zmiana z przechowalni ostatecznie lądują w repozytorium - za pomocą polecenia:

```
git commit
```

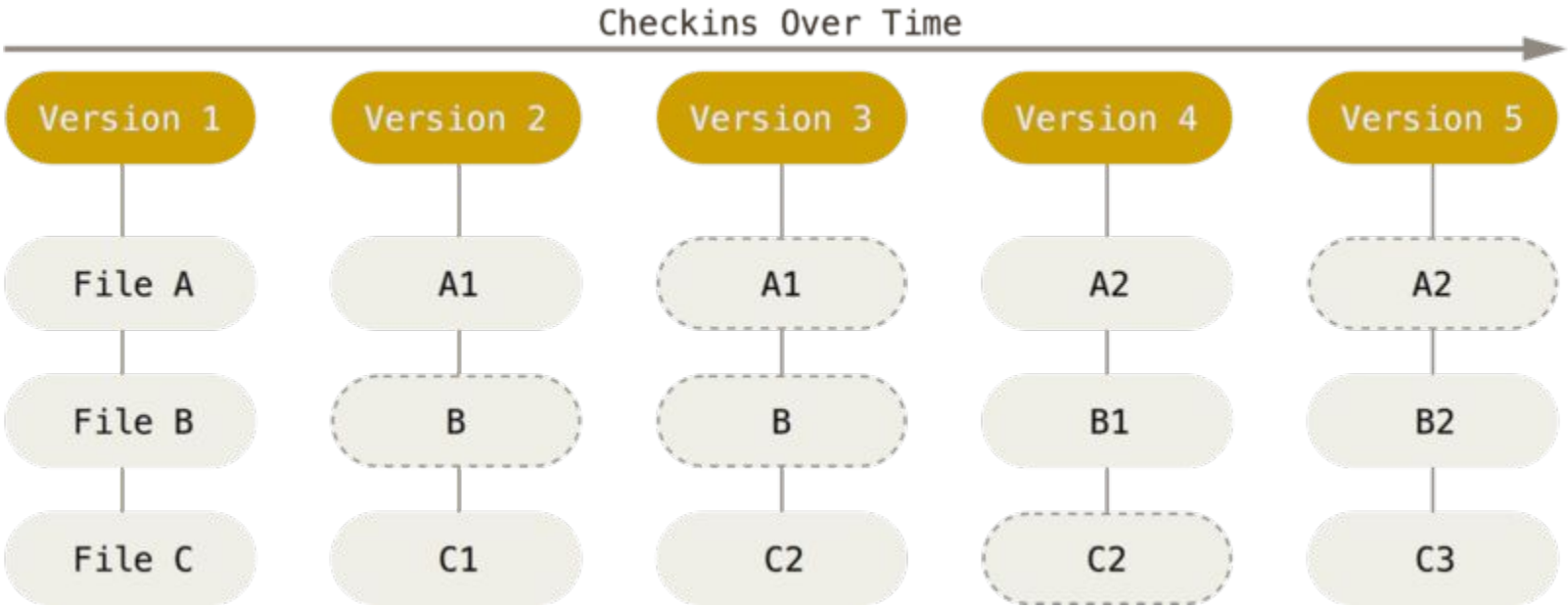
# Cykl życia pliku



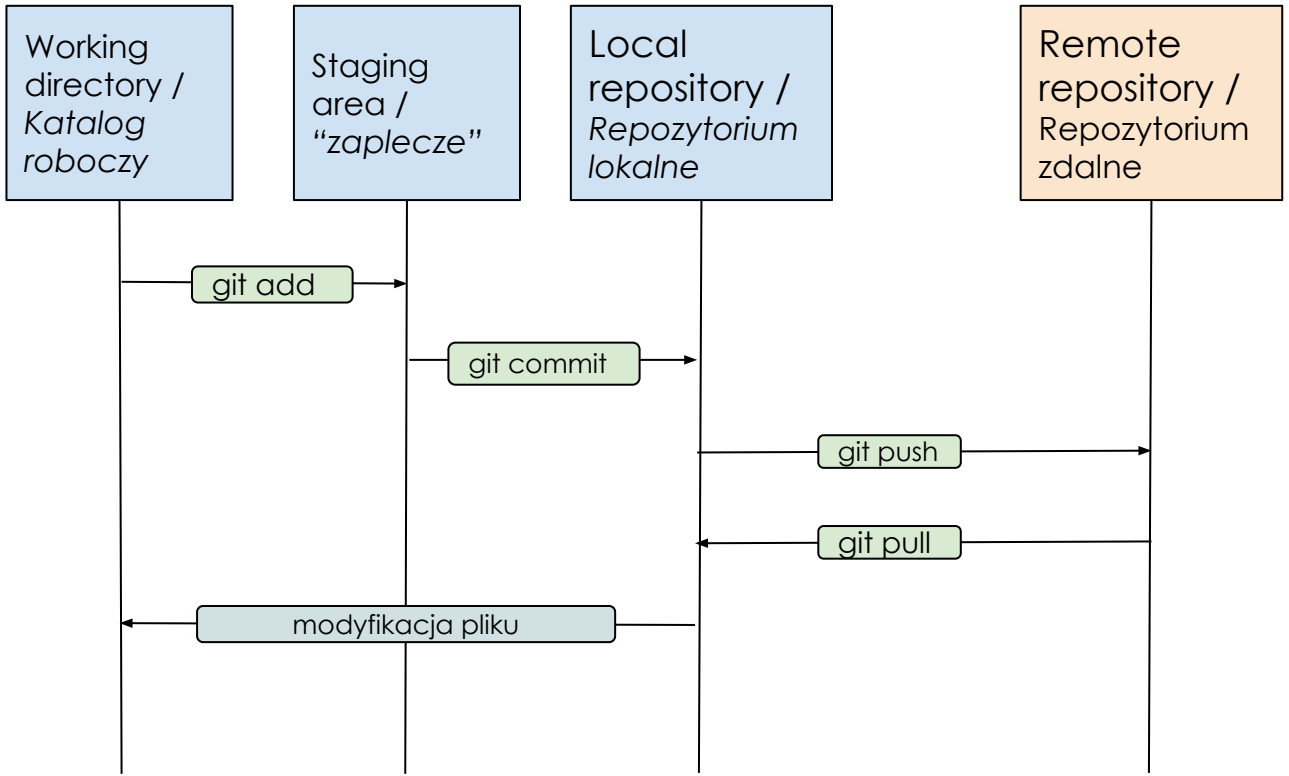
# Sposób różnicowy zapisu wersji (NIE GIT)



# Migawki (stosowane w GIT)



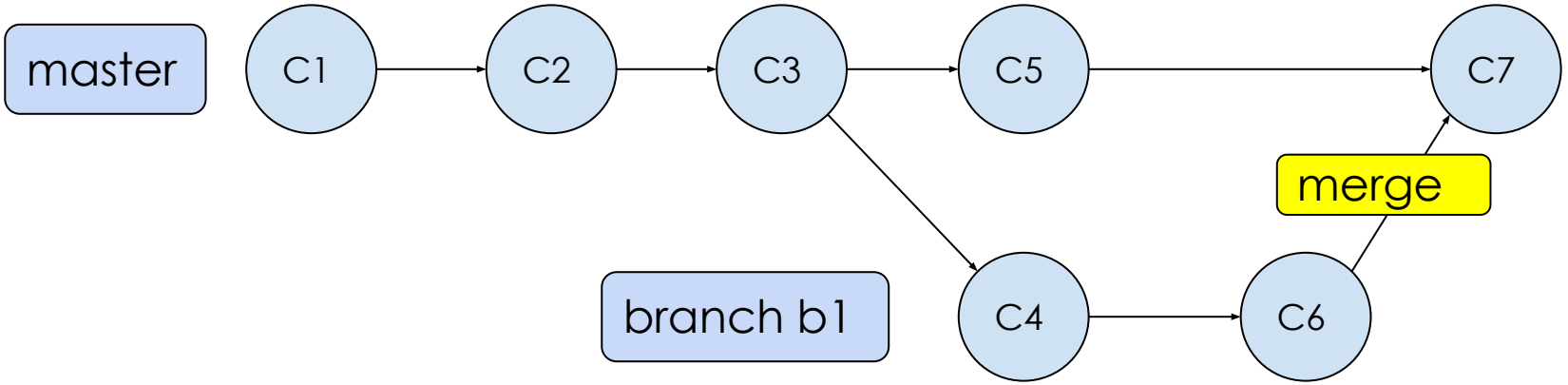
# Praca ze zdalnym repozytorium




Dochodzi kolejny element, czyli "wypchnięcie" naszych zmian do zdalnego repozytorium.

Może to być też operacja odwrotna - ze zdalnego repozytorium ściągamy zmiany, które się pojawiły od momentu rozpoczęcia przez nas pracy nad bieżącą funkcjonalnością.

# Gałęzie (branches)




Gałąź tworzymy (**rozgałęziamy**), gdy chcemy zachować spójność głównego kodu, a w międzyczasie implementujemy inną funkcjonalność.

Po skończonej pracy wykonujemy operację odwrotną - **scalenie** z główną gałęzią.

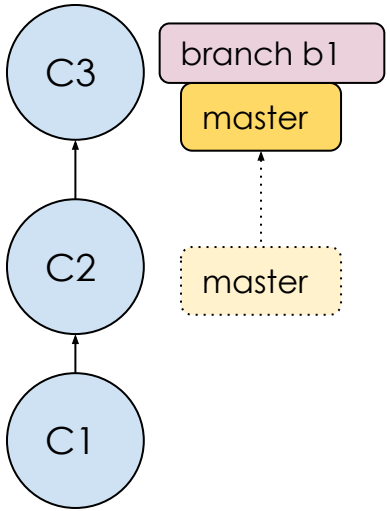
# Scalanie gałęzi - merge

Rozdzielone gałęzie w dowolnym momencie możemy **scalić**. Wykonujemy to poprzez operację **merge**.

Jeżeli w momencie operacji merge w gałęzi master nie było zmian, to wtedy Git wykonuje operację fast-merge.

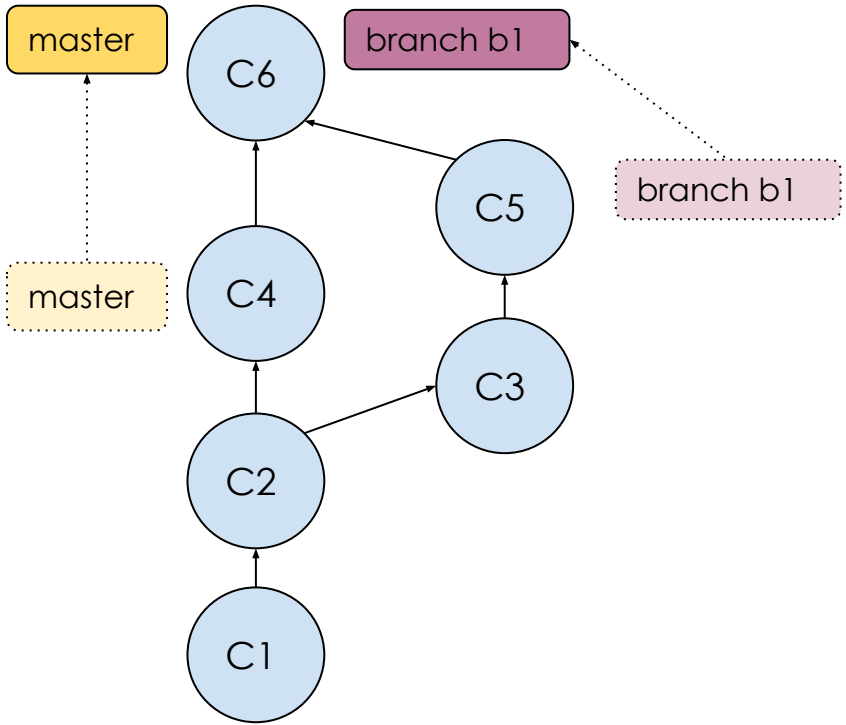
Polega ona na przesunięciu wskaźnika master do commitu oznaczonego wskaźnikiem łączonego brancha.

Nie jest tworzony commit dla operacji merge.



# Scalanie gałęzi - git merge

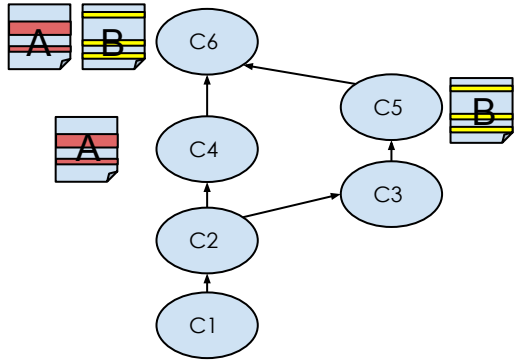
Jeżeli zmiany zaszły w obu gałęziach od momentu rozdzielenia, fast-merge nie jest możliwy. Następuje wtedy sklejenie zmian z obu gałęzi oraz jest tworzony nowy commit.

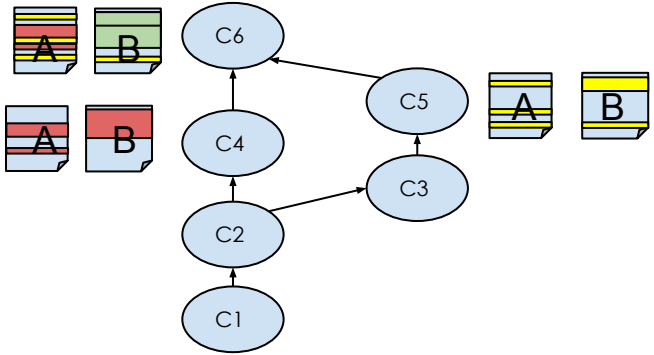


# Scalanie gałęzi - konflikty

Git całkiem nieźle sobie radzi, jeżeli łączymy zmiany nie obejmujące tych samych plików. Jeżeli zarówno w bieżącym jak i łączonym branchu są zmiany w tym samym pliku, to następuje tzw. konflikt.

Częściowo może on zostać rozwiązany automatycznie przez Git, natomiast niektóre zmiany wymagają ręcznej ingerencji.



# Feature branch

Tworząc nową funkcjonalność, tworzymy nowy branch, w którym robimy zmiany, dopóki nie zostanie ona zaimplementowana. Po skończonej pracy robimy operację merge i łączymy zmiany z główną gałęzią produktu.


Feature branch może zostać utworzony:

- na potrzeby nowej funkcjonalności
- dla zmian kodu realizujących konkretne zadanie (np. task w systemie JIRA)

Dzięki temu główna gałąź produktu pozostaje działająca także w trakcie prac nad nową funkcjonalnością.

# GIT Workflow