GitHub

GitHub – serwis hostujący repozytoria gita z dodatkowymi możliwościami (w szczególności dla usprawnienia kolaboracji nad projektami). Bardzo popularny i używany przez wiele projektów, zarówno closed source jak i open source.

- Hostuje około 370 milionów repozytoriów.
- Przykłady hostowanych projektów open source: CPython (i wiele jego bibliotek, np. numpy, matplotlib, ...), LaTeX, jądro Linuksa, git (czyli „wszystko" znane do tej pory z wykładu – ilustracja dominacji gita + GitHuba).
- Jeśli projekt jest open source, przypuszczalnie jest na GitHubie.
- Użyteczny również dla prywatnych (jedno- lub kilkuosobowych) projektów.
- Interfejs webowy, pozwala na korzystanie z gita bez użycia konsoli.
- Dużo narzędzi do inspekcji repozytoriów, ich historii, ...
- Podobne alternatywy: np. GitLab, Bitbucket.

Kilka terminów

Wstęp do kolaboracji – kilka terminów.

Commit – „wersja", „snapshot" (=migawka) projektu. Repozytorium jest bazą danych commitów. Każdy commit ma unikalny identyfikator (na slajdach skrócony do dwóch cyfr szesnastkowych, np. $\#0\mathrm{a}$).

Gałąź – formalnie: alias konkretnego commitu (i nic więcej). Konceptualnie: cała „chronologia" commitów.

Każdy commit poza pierwszym ma poprzednika (i „wskazuje" na niego).

Najprostsza sytuacja:

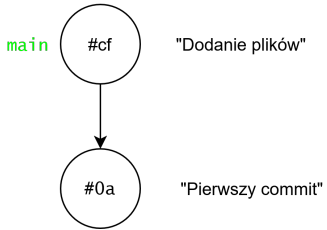
- Nowy commit wskazuje na (dotychczasowy) ostatni commit.
- Istnieje jedna gałąź (typowo main lub master) nazywająca najnowszy commit.

git – rozgałęzianie i scalanie

Diagram commitów (dla „najprostszej sytuacji").

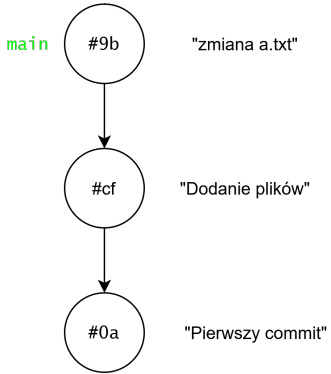
git – rozgałęzianie i scalanie

Diagram commitów (dla „najprostszej sytuacji").



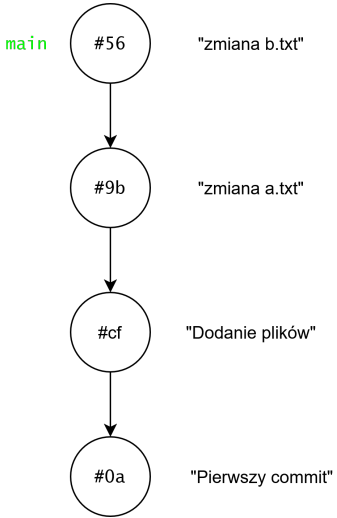
git – rozgałęzianie i scalanie

Diagram commitów (dla „najprostszej sytuacji").



git – rozgałęzianie i scalanie

Diagram commitów (dla „najprostszej sytuacji”).



git – rozgałęzianie i scalanie

Naturalny problem przy kolaboracji – modyfikowanie tych samych plików przez różne osoby. Przykład: duży zespół pracuje nad wersją 2.0 pewnej aplikacji. Podzespoły A i B zajmują się implementacją jej dwóch nowych elementów. W większości A i B zajmują się innymi plikami projektu, ale istnieje plik config.txt zmieniony przez oba zespoły: Oryginał pliku:

```
time=0
message="cat"
```

Wersja, z którą pracuje A:

```
time=5
message="dog"
```

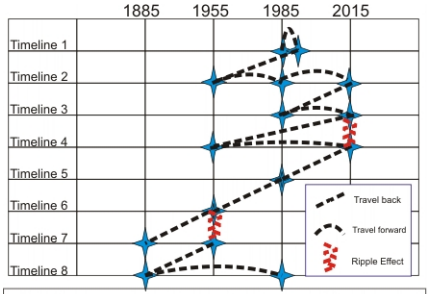
Wersja, z którą pracuje B:

```
time=0
error="unknown"
```

Pracując jak w „najprostszej sytuacji” ryzykujemy, że A i B nadpisują nawzajem swoje zmiany. Takich zmian może być wiele i trudno je ręcznie śledzić.

Co więcej, ostatecznie w repozytorium musi zaistnieć commit odpowiadający wersji 2.0 aplikacji, nad którą pracują podzespoły, a w niej musi znaleźć się konkretna wersja $\mathrm{config.txt}$. Jednak modyfikacje wykonane przez A i B nie są kompatybilne.

Rozwiązanie: rozgałęzienie projektu



TL1: 1985: Einstein moves 1 minute into the future. Marty then travels to 1955 creating timeline 2.
TL2: 1955: Marty changes elements of his parent's history then moves forward to 1985. The Doc then travels to 2015.
TL3: 2015: Doc arrives in 1985 and takes Marty and Jennifer to 2015 to solve a problem with Marty Jr. Whilst there, Old Biff steals the De Lorean and travels to 1955 creating timeline 4.
TL4: 1955 Old Biff gives Young Biff the Almanac and changes history so drastically that he is no longer alive (he would have died in the 1990s. The ripple effect removes him from the Timeline. Doc and Marty head back to 1985 as the ripple effect is transforming TL3 into TL4
TL5: The Doc and Marty arrive in 1985A and realise what has happened. They go to 1955 to solve it.
TL6: Whilst in 1955 the De Lorean is struck by lightening and sends the Doc back to 1885. The ripple effect cleanses the TI6 into TL7.
TL7: A delivery man hands Marty instructions the Doc wrote in 1885 and Marty travels to 1885 to save him.
TL8: Marty helps Doc and travels back to 1985.

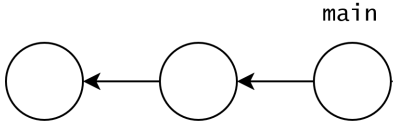
Rozwiązanie: rozgałęzienie projektu

Zespół A pracuje na głównej gałęzi (main), zespół B tworzy nową gałąź (np. feature-new). Od teraz jeden wybrany commit będzie poprzednikiem dla dwóch (lub więcej) commitów i „chronologia" projektu się rozgałęzia.

Przypomnienie: gałąź to w rzeczywistości alias commitu.

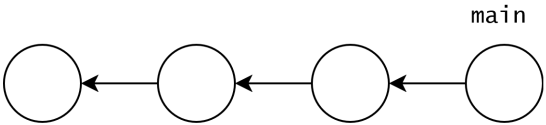
git – rozgałęzianie i scalanie

Zespół A pracuje na gałęzi main.

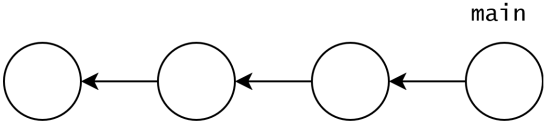


git – rozgałęzianie i scalanie

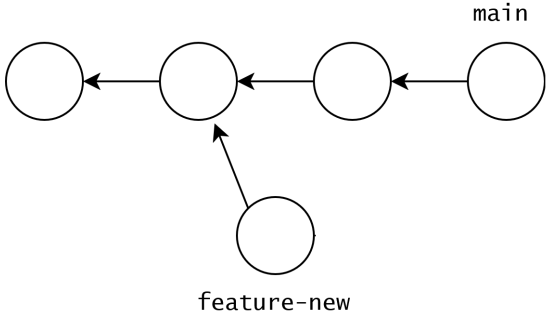
Zespół A pracuje na gałęzi main.



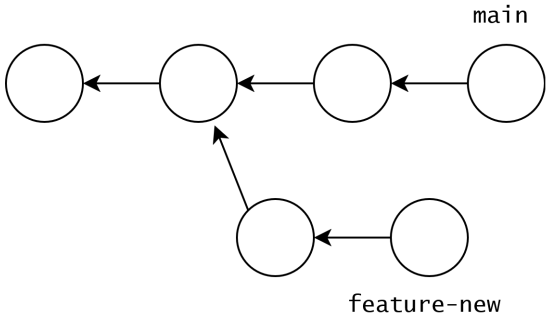
Zespół B zaczyna pracę, przyjmując za bazę jeden z istniejących commitów.



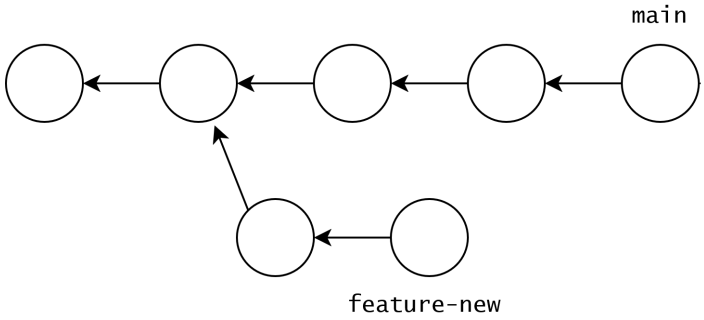
Zespół B zaczyna pracę, przyjmując za bazę jeden z istniejących commitów.



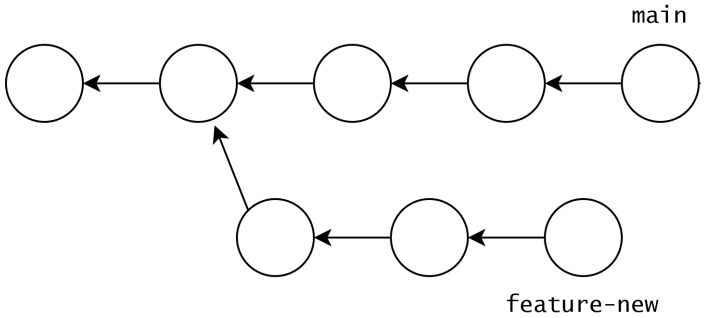
Nowy commit B na feature-new.



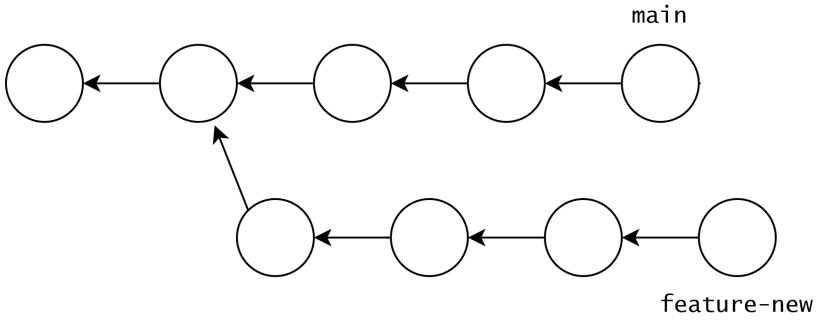
Nowy commit A na main.



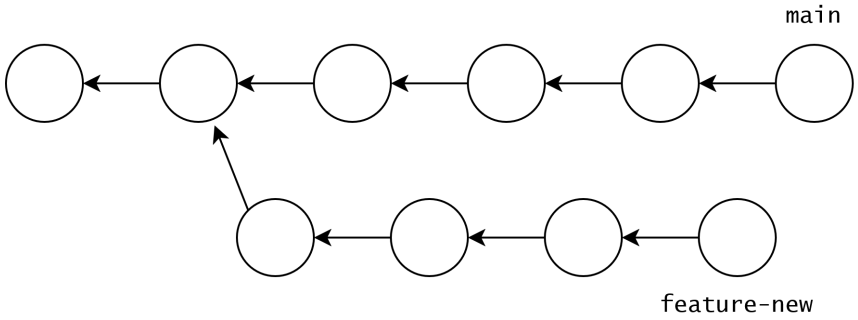
Kolejny commit B na feature-new.



Jeszcze jeden commit B na feature-new.



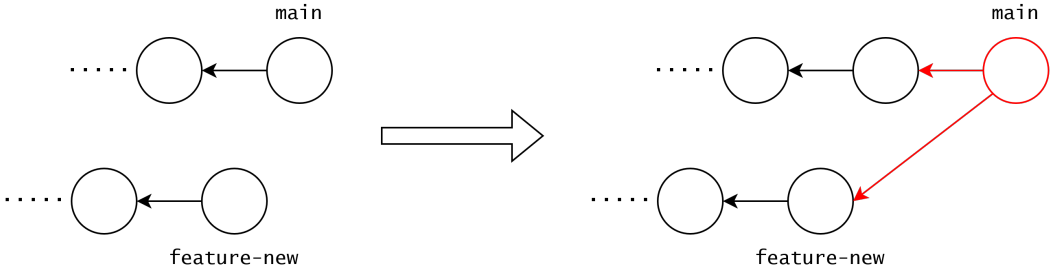
Nowy commit A na main.



git – rozgałęzianie i scalanie

„Drugi koniec” – scalanie (merge) jednej gałęzi z drugą.

Polega na dodaniu commita, którego poprzednikami są obie dotychczasowe gałęzie. Druga gałąź staje się aliasem na ten commit. Scalanie feature-new z main:



Typowo takie scalenie wykonuje się, gdy praca na feature-new została zakończona i efekty można włączyć do głównej gałęzi.

Scalane gałęzie mogą mieć konflikty i scalenie musi je rozwiązać poprzez utworzenie wspólnej wersji każdego pliku - wystarczy to jednak zrobić w samym momencie scalania (a nie na bieżąco z każdym dotychczasowym commitem).

Uwaga: każde dwie gałęzie mają *wspólnego przodka*: commit w którym się rozchodzą (w najgorszym wypadku będzie to pierwszy commit).

git – rozgałęzianie i scalanie

W wielu sytuacjach konflikt scalania gałęzi X i Y nie występuje i scalenie następuje automatycznie. Niech P będzie wspólnym przodkiem X i Y. Przykłady:

- Dany plik nie został zmieniony w X (względem tego, jak wyglądał w P), a został zmieniony w Y – wtedy w scaleniu wzięta jest wersja zmieniona (z Y).
- Plik został zmieniony (względem P) w X i Y, ale w róznych linijkach – w scaleniu pojawią się obie zmiany.
- Plik przeniesiony/ze zmienioną nazwą w X (względem P), bez zmian w X – przeniesienie/zmiana nazwy w scaleniu.

Pozostałe konflikty zazwyczaj wymagają interwencji scalającego.

git – rozgałęzianie i scalanie

Live demo na GitHubie:

- Commit do nowej gałęzi.
- Pull request.
- Ręczne rozwiązywanie konfliktów i akceptowanie pull request.
- (Pogadanka: rola pull request w ekosystemie GitHuba, forking repozytoriów.)