System kontroli wersji GIT

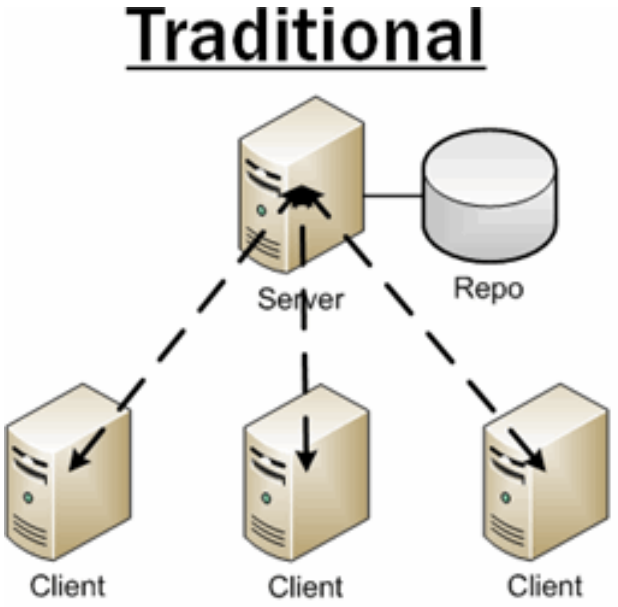
Tomasz lisowski
tomasz.lisowski@protonmail.ch

Gdańsk, 06.06.2017

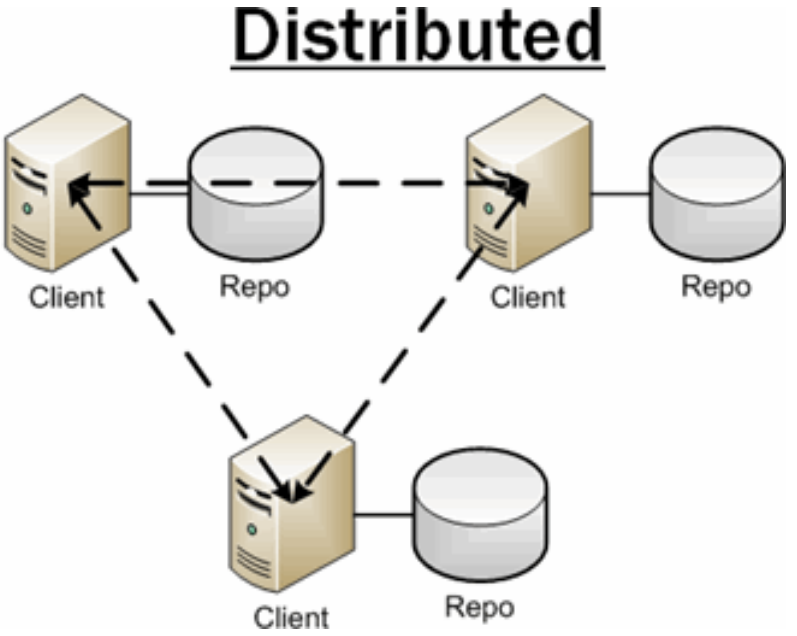
Agenda

- Podstawowe operacje
- Cofanie zmian
- Branche
- Merge vs rebase
- Konflikty
- Git-flow
- Pull-request

Systemy kontroli wersji



Istnieje tylko jedno centralne repozytorium



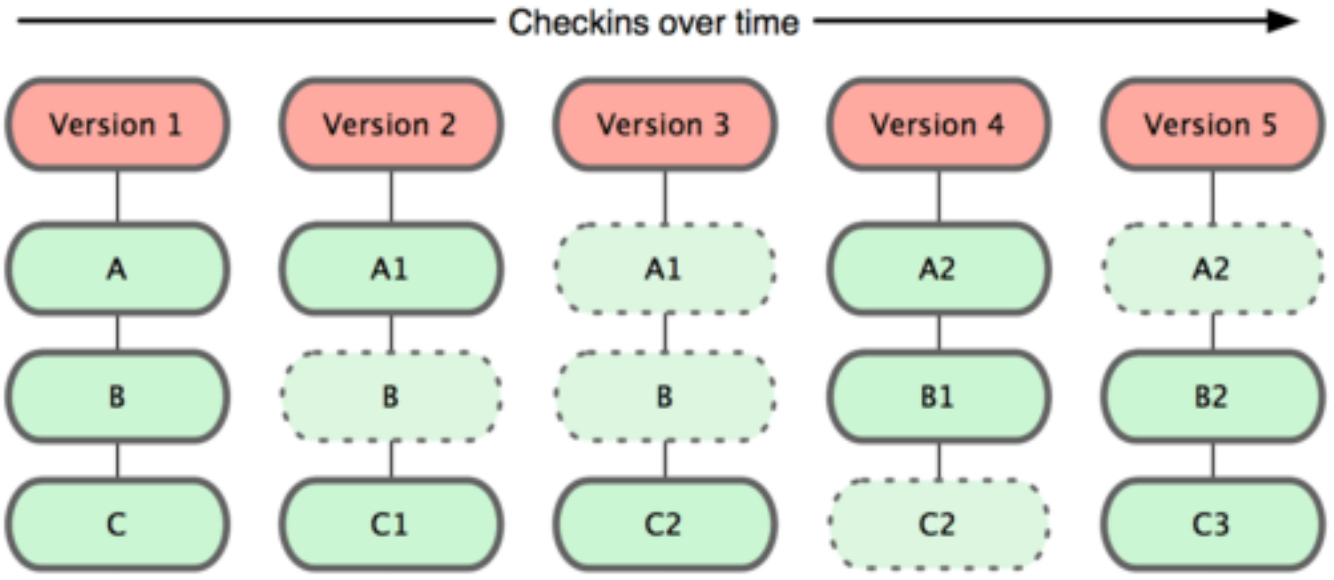
Każdy ma swoje lokalne repozytorium. Serwer też ma swoje repo.

Migawki

- Git traktuje dane jak zestaw migawek (ang. snapshots) małego systemu plików
- Każdy commit tworzy obraz wszystkich plików i przechowuje ich referencje
- Jeśli plik nie zostanie zmieniony, git nie zapisuje go ponownie
- Zapisuje tylko referencję do poprzedniej wersji

Lokalna baza

- Klonowanie pobiera całe repozytorium
- Tworzy lokalną bazę danych projektu
- Para klucz:wartość


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

Same zalety

- Rozproszony
- Zoptymalizowany pod kątem wydajności
- Bezpieczny
- Elastyczny (nieliniowe flow pracy)
- Darmowy i rozwijany jako OSS
- Aktualny standard

Narzędzia

- Polecenia bezpośrednio z konsoli
- Lub z IDE
- Niektóre operacje mogą się inaczej nazywać
  - IDE ma ujednoliconą obsługę wielu VCS

clone

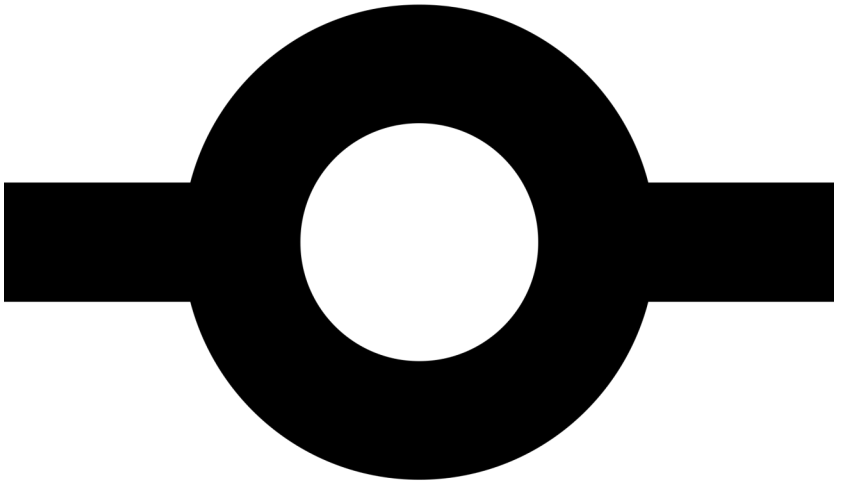
- Kopiuje istniejące repozytorium
- Pobiera prawie wszystkie dane z serwera
- Każda rewizja, każdego pliku
- Dostępna cała historia
- Możliwość odtworzenia repozytorium

add

- Dodaje pliki do śledzenia
- Pliki w poczekalnie - ‘Changes to be commited’
- Zapisana wersja pliku z momentu wykonania add
- Parametrem nazwa pliku/katalogu lub .
- Ćwiczenie ?

commit

- Zatwierdza zmiany
- Zapisanie różnicy zawartości plików
- Stałe i niezmienne
- Odpowiadają konkretnym zmianom
- Mają wiadomości, które je opisują



commit

- Każdy commit ma swojego rodzica
- Ma też swój UNIKALNY klucz SHA1
- Można przejrzeć krok po kroku jak zmieniał się kod
- Podgląd zmian poszczególnych plików
- git commit -a → pod spodem wykonuje ‘git add’

push

- Wypycha zmiany na serwer zewnętrzny
- git push [nazwa-repo] [nazwa-gałęzi]
  np. git push origin master
- Wymagane uprawnienia do zapisu
- Oraz brak konfliktów

pull

- Pobranie i włączenie zmian
- Wszystkie dane, których brak lokalnie
- Próba automatycznego scalenia
- Możliwe problemy

Pobieranie

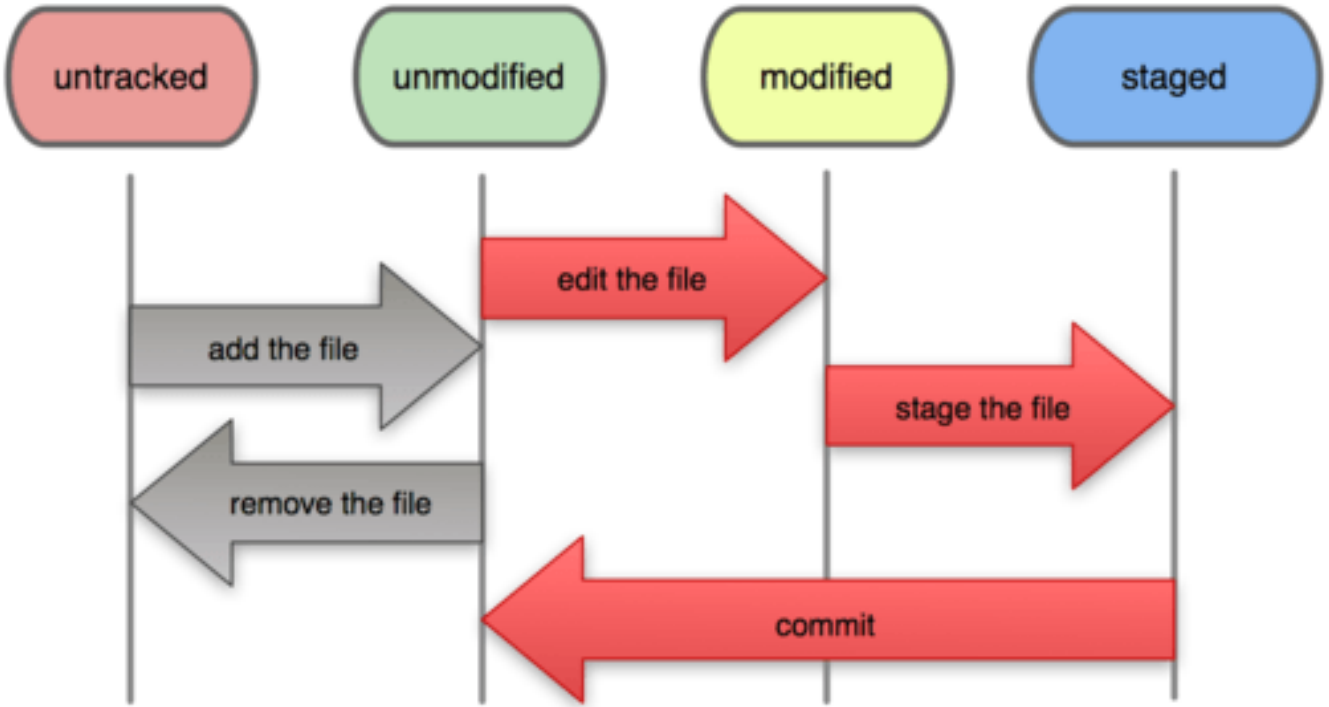
- Git clone – kopiuje istniejące repozytorium, ustawia jako origin
- Git fetch – pobiera commity ze zdalnego repo, ale jako „remote”, nie włącza ich do naszego lokalnego repo
  - Do tego konieczne będzie mergowanie
- Git pull – pobiera commity i włącza je do naszego repo (robi fetch „pod spodem”)

git pull = git fetch + git merge

status

- Sprawdza stan plików
- Pokazuje aktualną gałąź
- *nothing to commit, working directory clean*
- *Untracked files:*
- *Changes to be committed:*

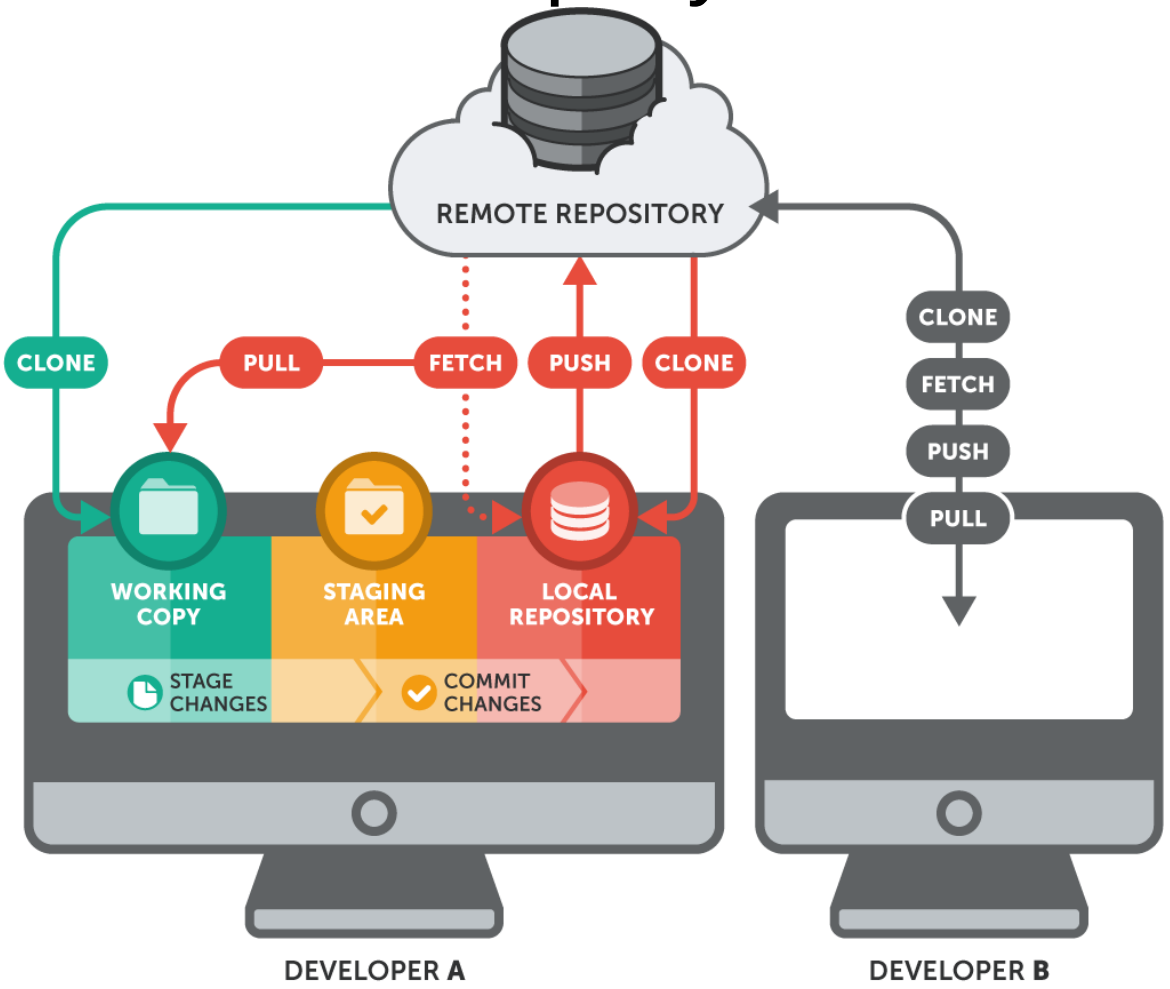
File Status Lifecycle



Ćwiczenie

- Sprawdź status
- Wprowadź zmianę
- Dodaj plik do śledzenia
- Wprowadź zmianę
- Wynik?

Zdalne repozytorium



Gitignore

- Ukrywa pliki/katalogi przed Gitem
- Często dane generowane automatycznie
- .gitignore – lista wzorców
- np. *.[oa] – wszystko co kończy się literą ‘o’ lub ‘a’
- *~ - pliki tymczasowe
- Możliwa negacja wyrażen - !

Gitignore

```
# komentarz – ta linia jest ignorowana
# żadnych plików .a
*.a
# ale uwzględniaj lib.a, pomimo ignorowania .a w linijce powyżej
!lib.a
# ignoruj plik TODO w katalogu głównym, ale nie podkatalog/TODO
/TODO
# ignoruj wszystkie pliki znajdujące się w katalogu build/
build/
# ignoruj doc/notatki.txt, ale nie doc/server/arch.txt
doc/*.txt
```

Ćwiczenie

- Dodaj plik o nazwie ‘prywatnyPlik’
- Sprawdź status
- Dodaj plik do .gitignore

Ćwiczenie

- Usuń poprzedni plik z .gitignore
- Dodaj katalog ‘prywatnyKatalog’
- Ustaw nowe pliki i katalog jako pominięte

diff

- Informuje o dokładnych zmianach
- Zmiany przed wysłaniem do ‘poczekalni’
- git diff --staged (lub --cached)
- Wykorzystywany interfejs graficzny

log

- Podgląd istniejących commitów
- Duża ilość parametrów
- git log -p -2 → 2 ostatnie commity
- git log --stat
- git log --pretty=oneline (pretty oznacza format)

Wycofywanie zmian

- reset
- commit --amend
- checkout

Reset

- Wycofanie lokalnych zmian z poczekalnie
- Nie zmienia stanu pliku

  git reset HEAD [plik]
- Ustawia branch na konkretny commit

  git reset [branch]

--amend

- ‘poprawianie’ commita
- Nie tworzy nowej migawki – aktualizuje obecną
- Możliwość zmiany plików
- Albo samego komentarza

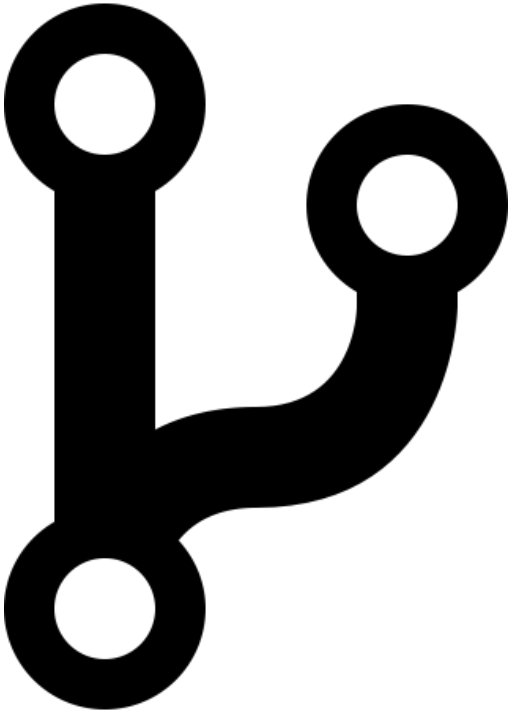
Checkout

- Gdy chcemy wycofać zmiany do poprzedniej wersji migawki
- *(use "git checkout -- <file>..." to discard changes in working directory)*
- Ryzykowne – nie można odzyskać tak usuniętych zmian

Ćwiczenie

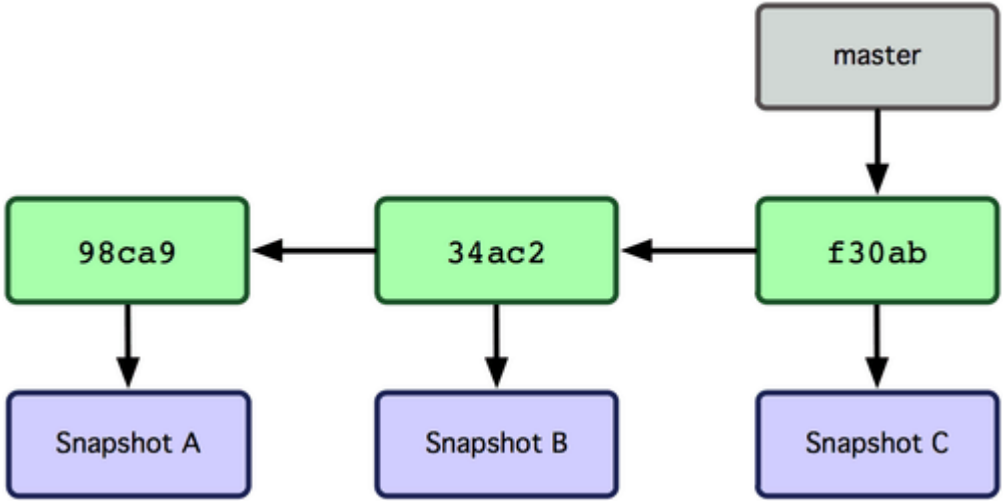
- Wprowadzanie zmian
- reset
- checkout

Branchowanie



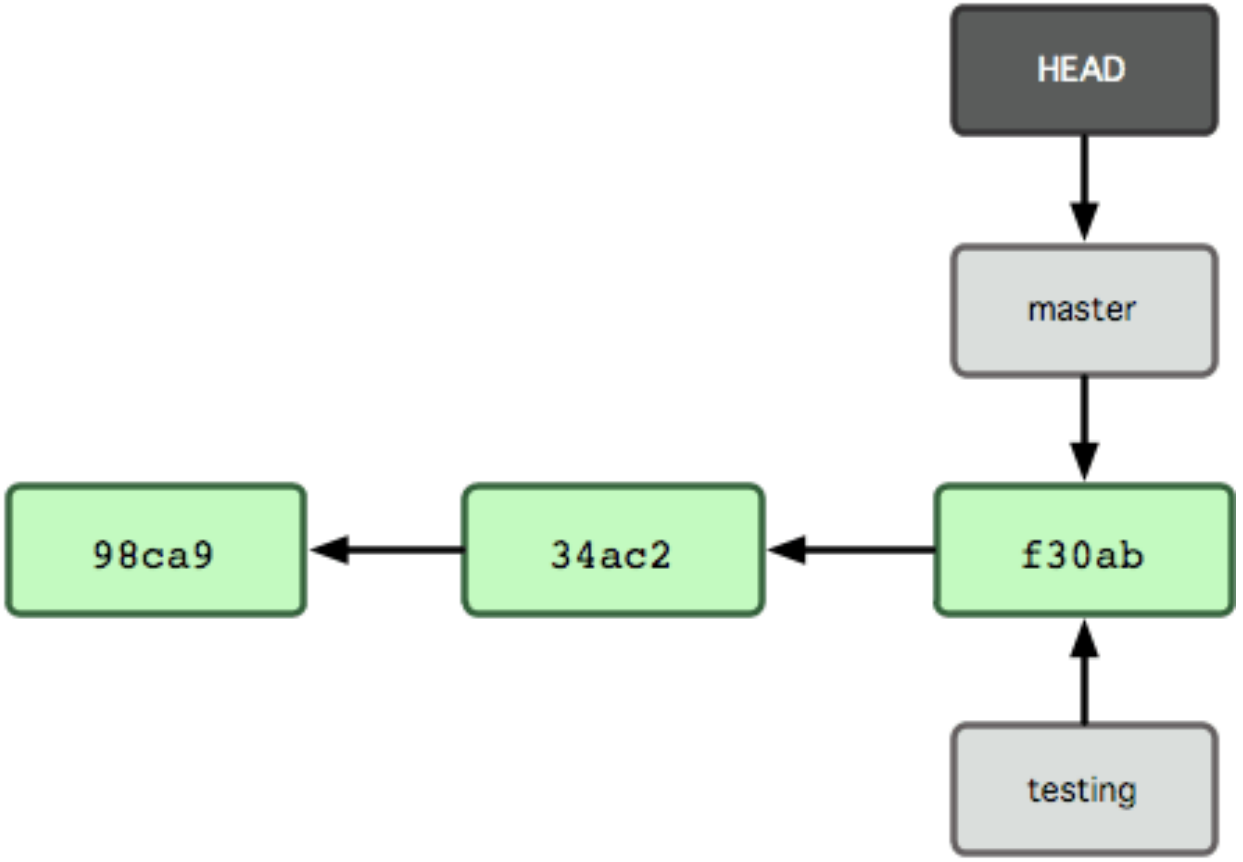
Branch (gałąź)

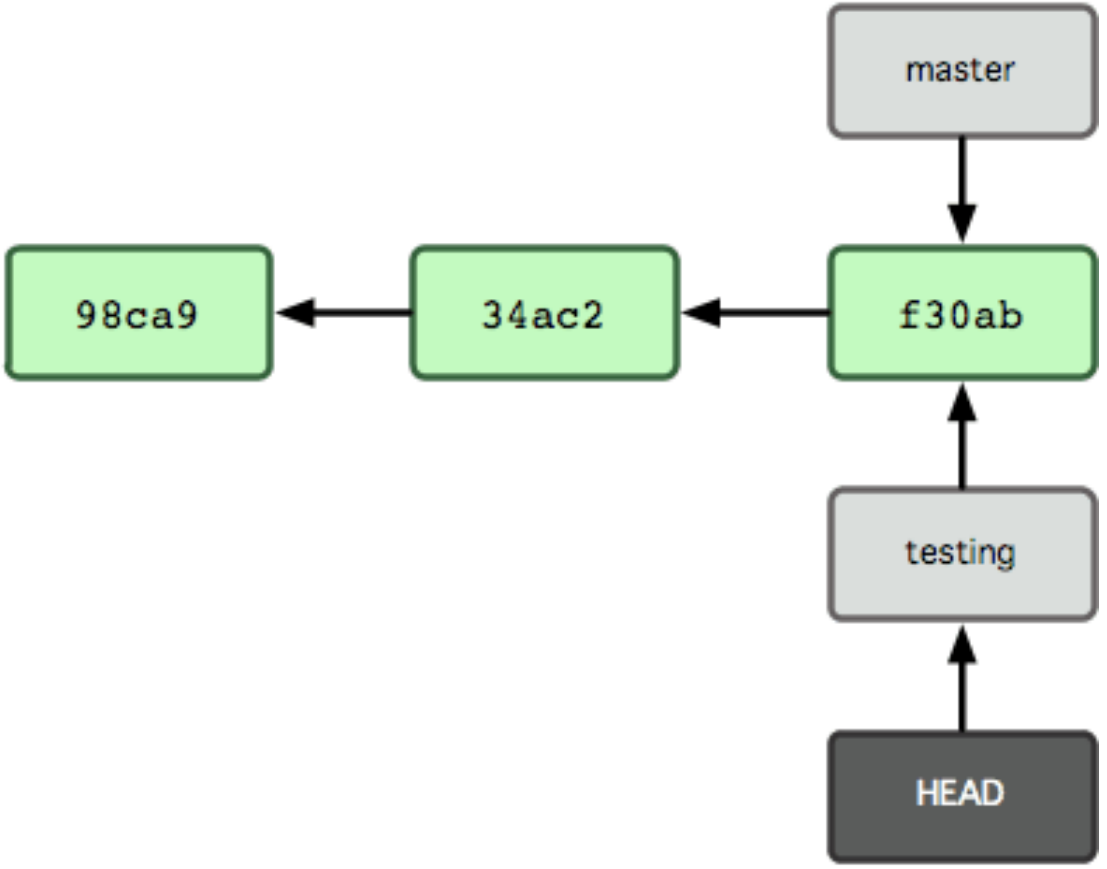
- git branch [nazwa]
- Wskaźnik na commit
- Można go zawsze dodać/usunąć/zmienić
- Pierwsza i domyślna - master

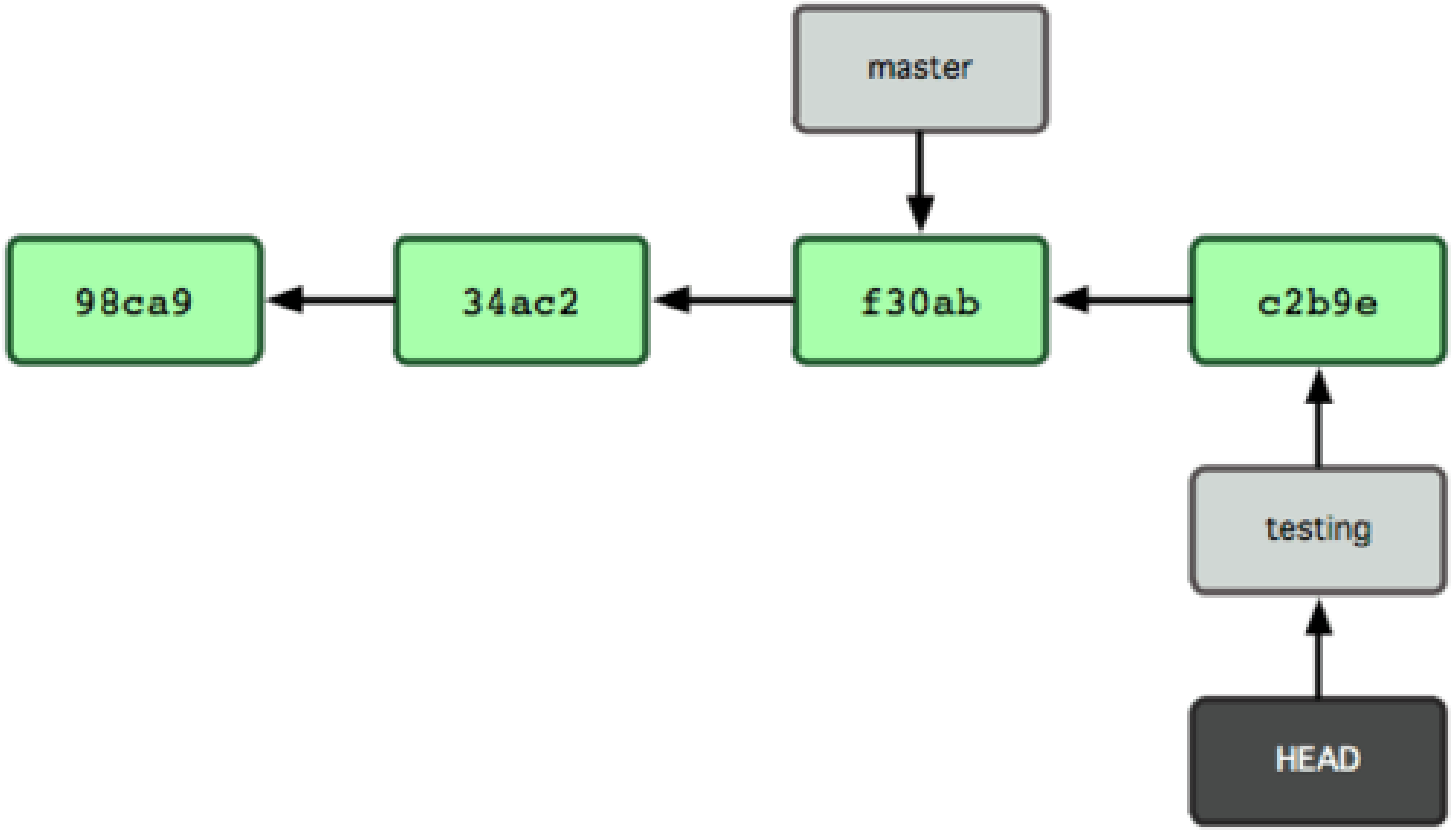


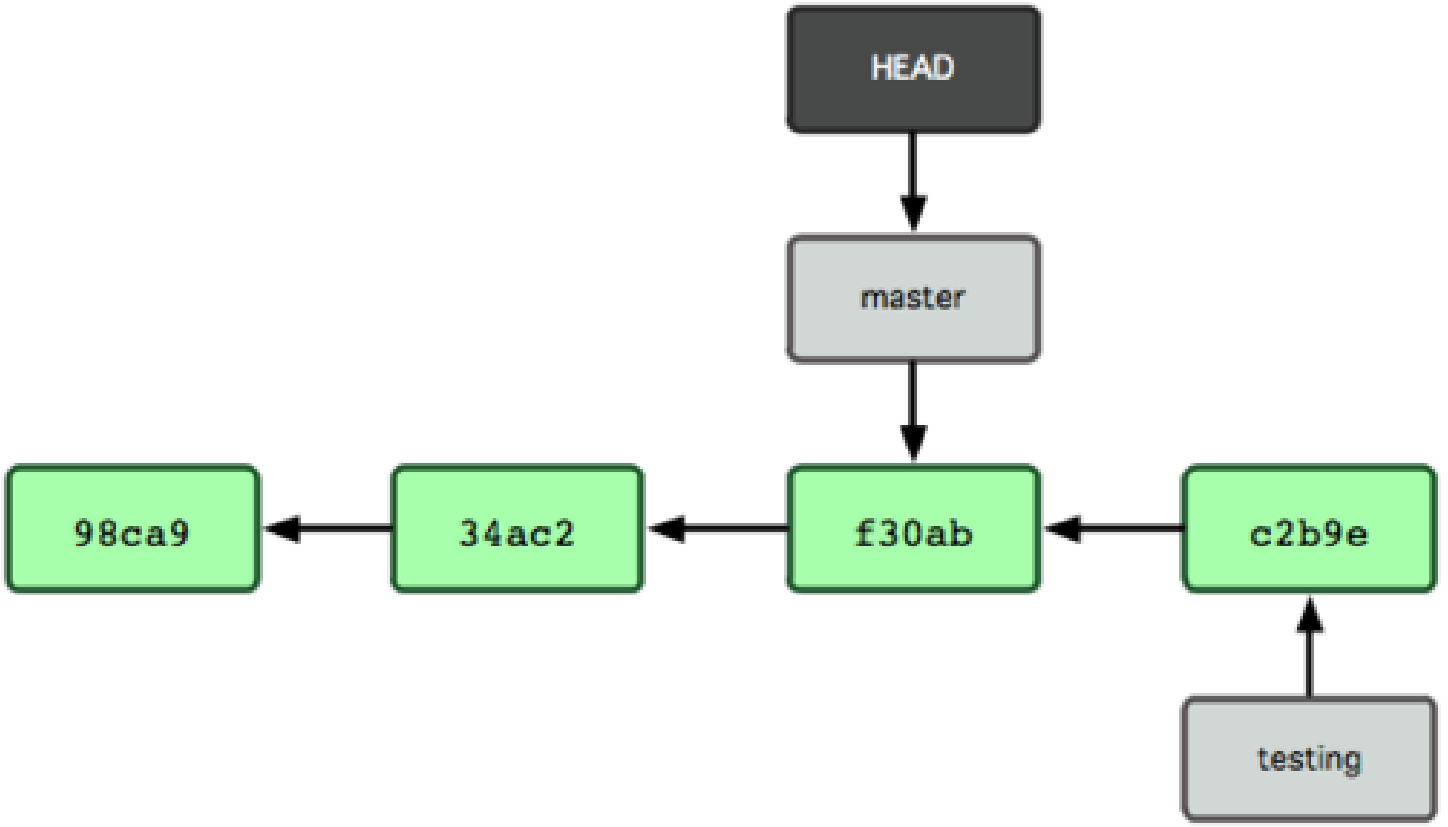
HEAD

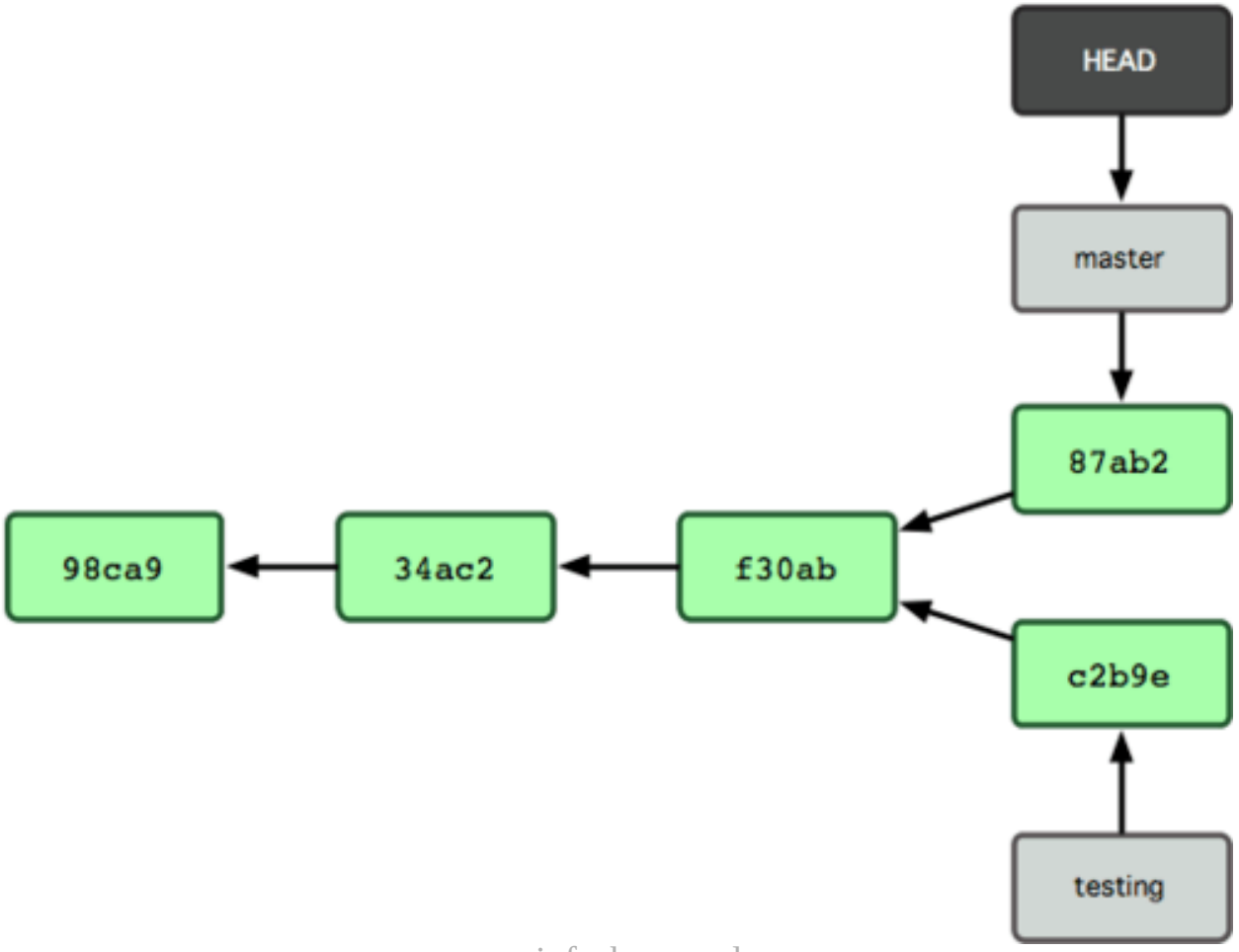
- Specjalny wskaźnik gałęzi
- Wskazuje aktualną, lokalną gałąź
- *git branch* tworzy nową, ale nie zmienia HEADa
- *git checkout*


HEAD
master
98ca9
34ac2
f30ab
testing


master
98ca9
34ac2
f30ab
testing
HEAD


master
98ca9
34ac2
f30ab
c2b9e
testing
HEAD


HEAD
master
98ca9
34ac2
f30ab
c2b9e
testing

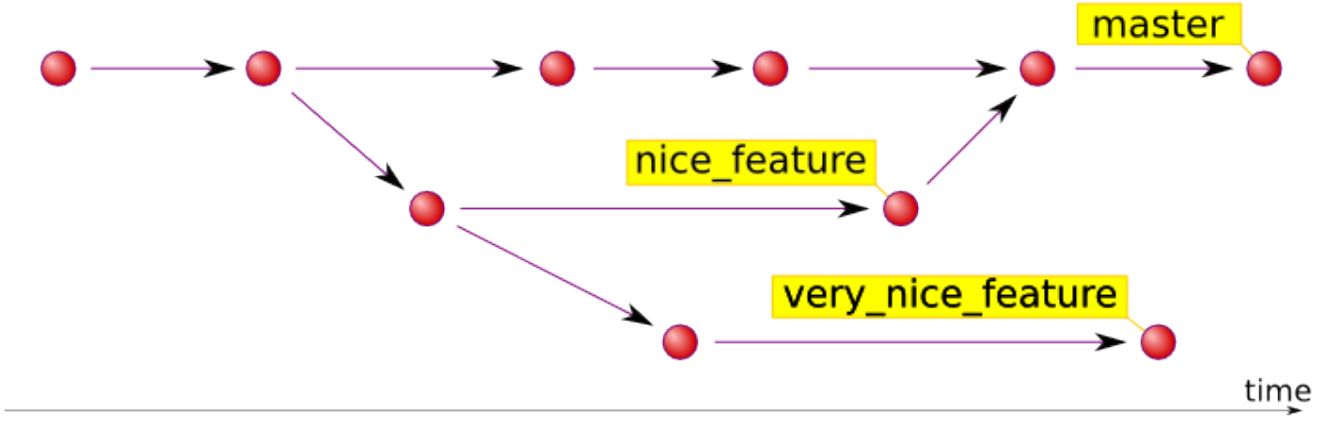

HEAD
master
87ab2
98ca9
34ac2
f30ab
c2b9e
testing

Branch

- Fizycznie branch jest tylko plikiem
- 40 znaków skrótu SHA-1 – wskaźnik na commit
- Tanie w tworzeniu i usuwaniu
- Szybkie
- Zupełnie odmienne od innych systemów

Commit vs branch

- Zupełnie dwie różne rzeczy
- Commity są przechowywane (persistent)
- Branche są płynne, to wskaźniki na commity

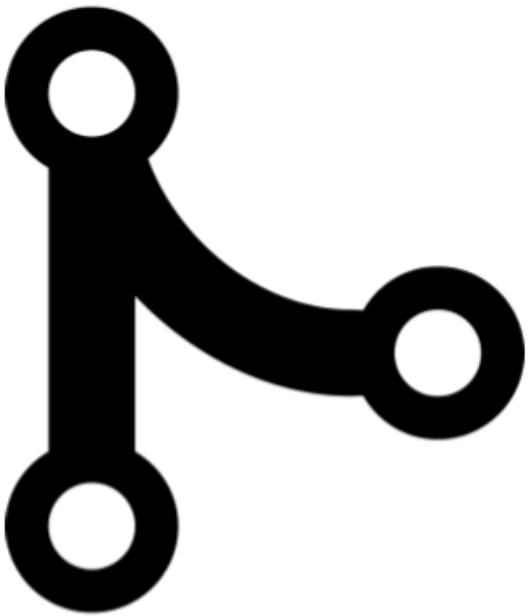


Po co branche?

- Problem pracy równoległej
- Problem nadpisywania sobie zmian
- Problem nie ukończonych rzeczy
- Praca na wieloma zadaniami na raz

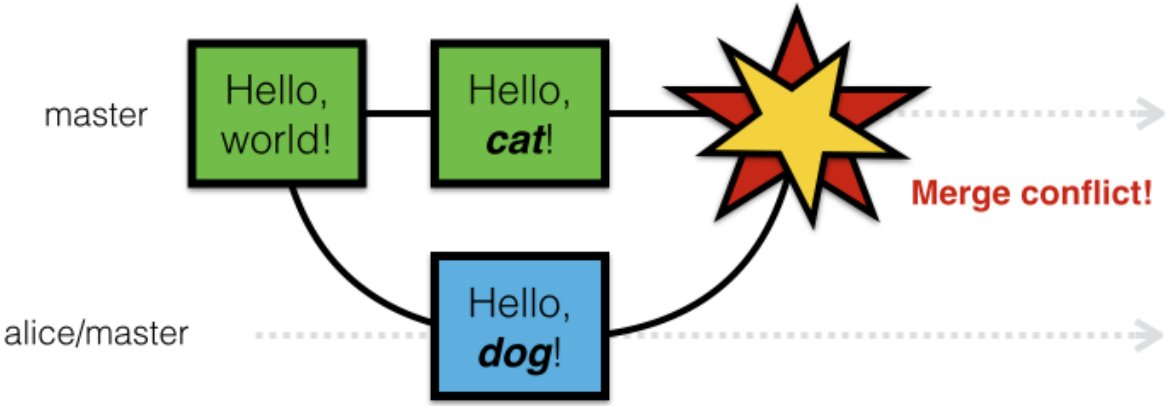
Merge

- Połączenie/mieszanie ze sobą różnych branchy
- Łączenie kodu w całość
- Potencjalne konflikty



Merge

- Czy zawsze się uda?
- Co się stanie gdy pracujemy na dokładnie tym samym kodzie co ktoś inny?
- Które zmiany wybrać?



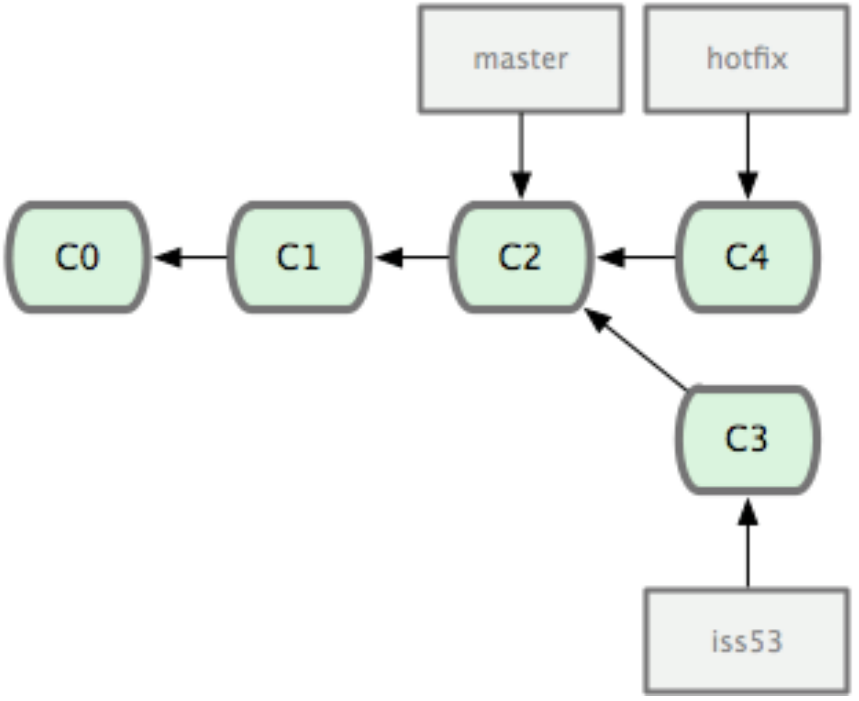
Merge

- Przełączenie na gałąź, DO której chcemy scalać
- Polecenie merge wskazujące na branch do scalenia

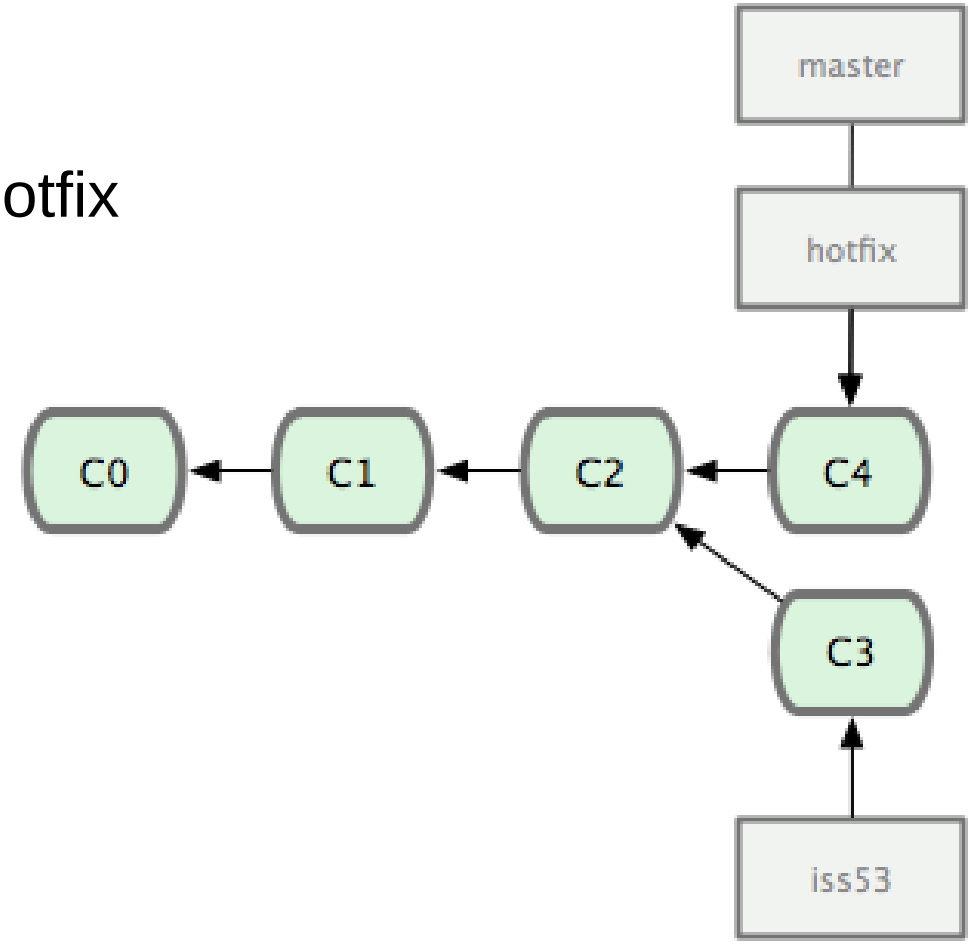
  np.:
  git checkout master
  git merge feature1

Fast forward

- Gdy zmiany są w relacji rodzic-dziecko
- Przesuwa wskaźnik



git merge hotfix



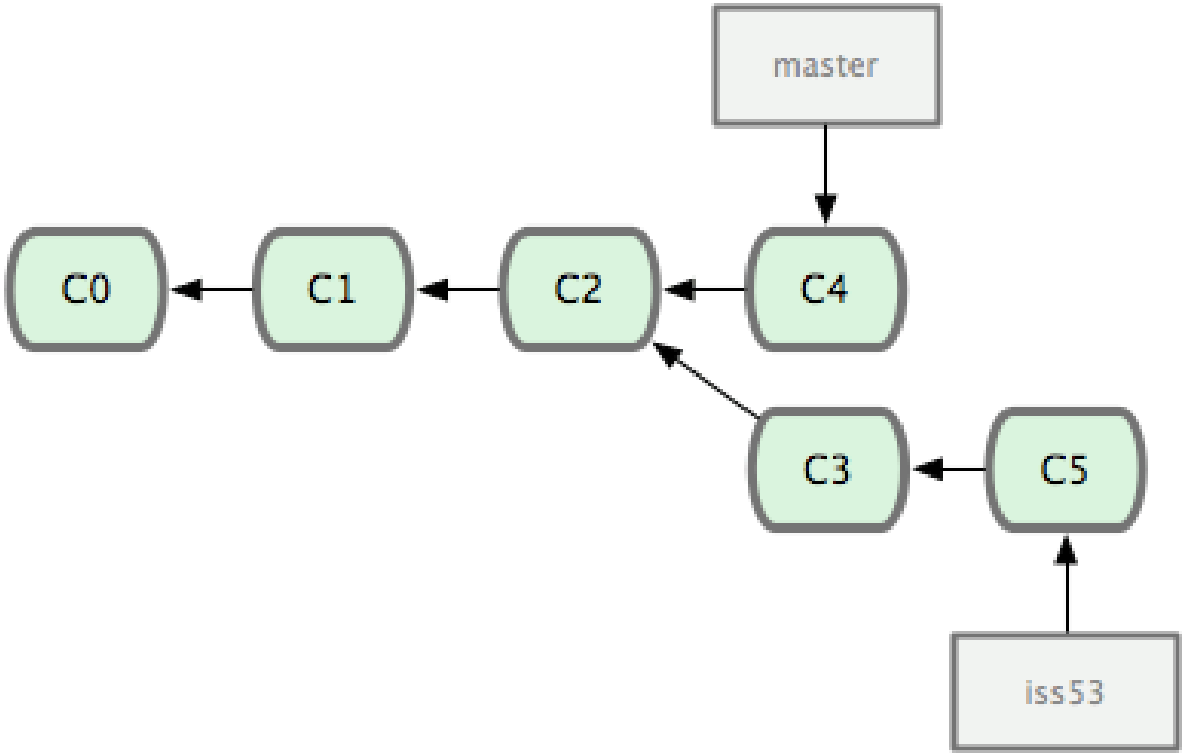
Ćwiczenie

- Stworzyć branch ‘superFeature’
- Wykonać hotfix

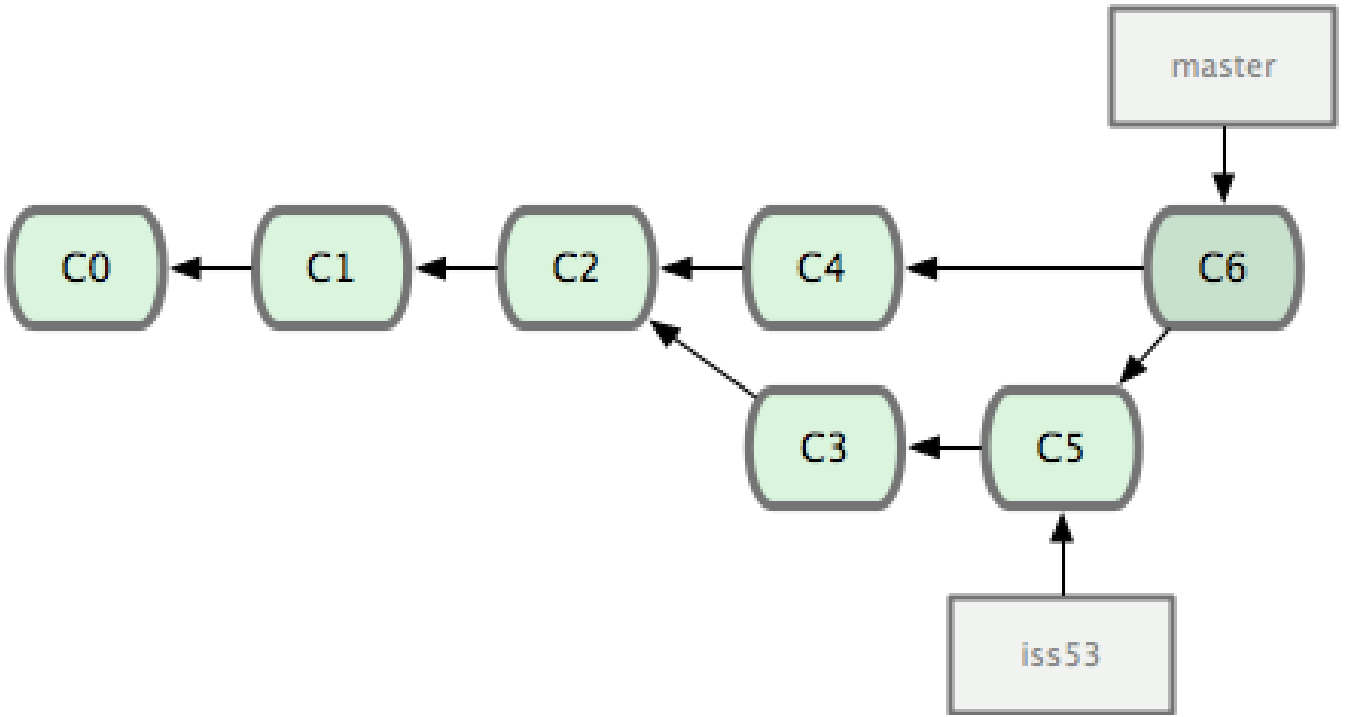
3-way merge

- Gdy zmiany nie są bezpośrednim potomkiem
- Git szuka wspólnego przodka scalanych migawek
- Tworzy nową migawkę
- Można usunąć scalany branch
  git branch -d [nazwa]

3-way merge



3-way merge



Merge conflict

- Naturalna część pracy z kodem
- Nie należy się ich bać
- Czasem wymagają konsultacji z autorem innych zmian
- Konflikty podczas ‘git pull’
  - ponieważ pod spodem jest merge

Merge conflict

- Zmiana tego samego fragmentu
- *Automatic merge failed; fix conflicts and then commit the result.*

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Rozwiązywanie konfliktów

- VCS > GIT > Resolve Conflicts..
- Merge..
- Po rozwiązaniu konfliktów sprawdzenie statusu
- Dodanie śledzenia
- Commit potwierdzający nową migawkę po scaleniu

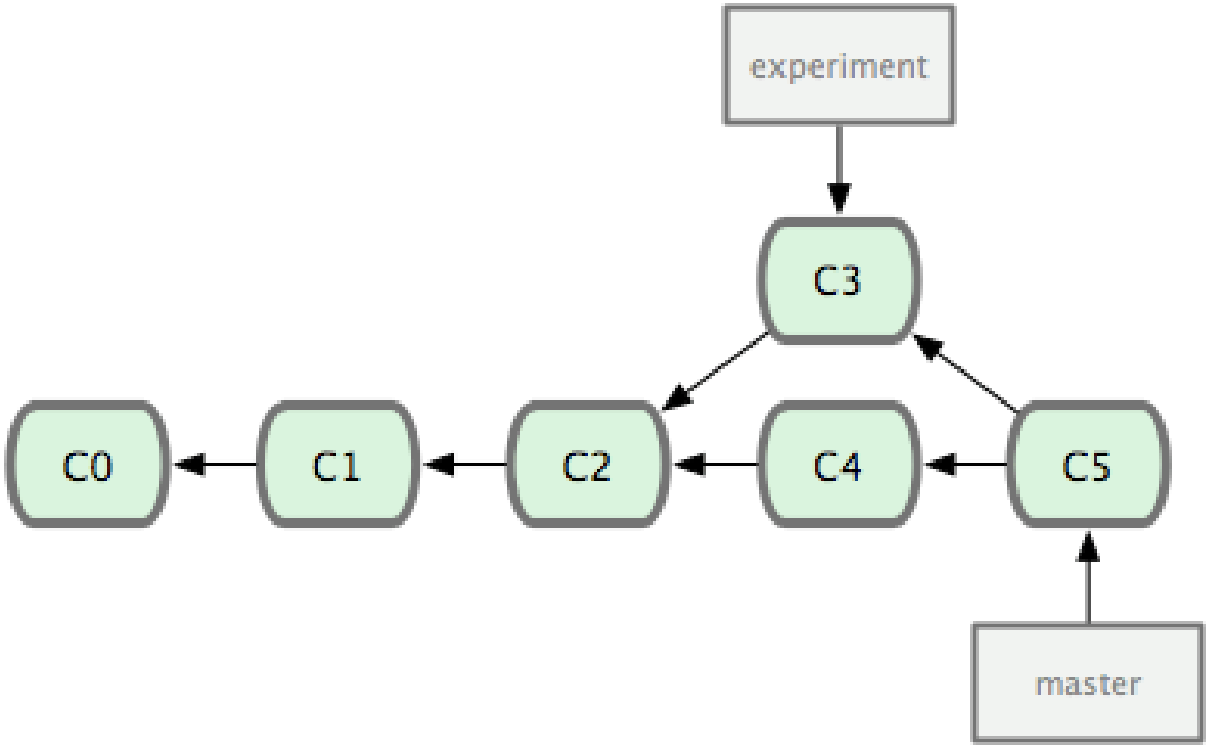
Dobre praktyki

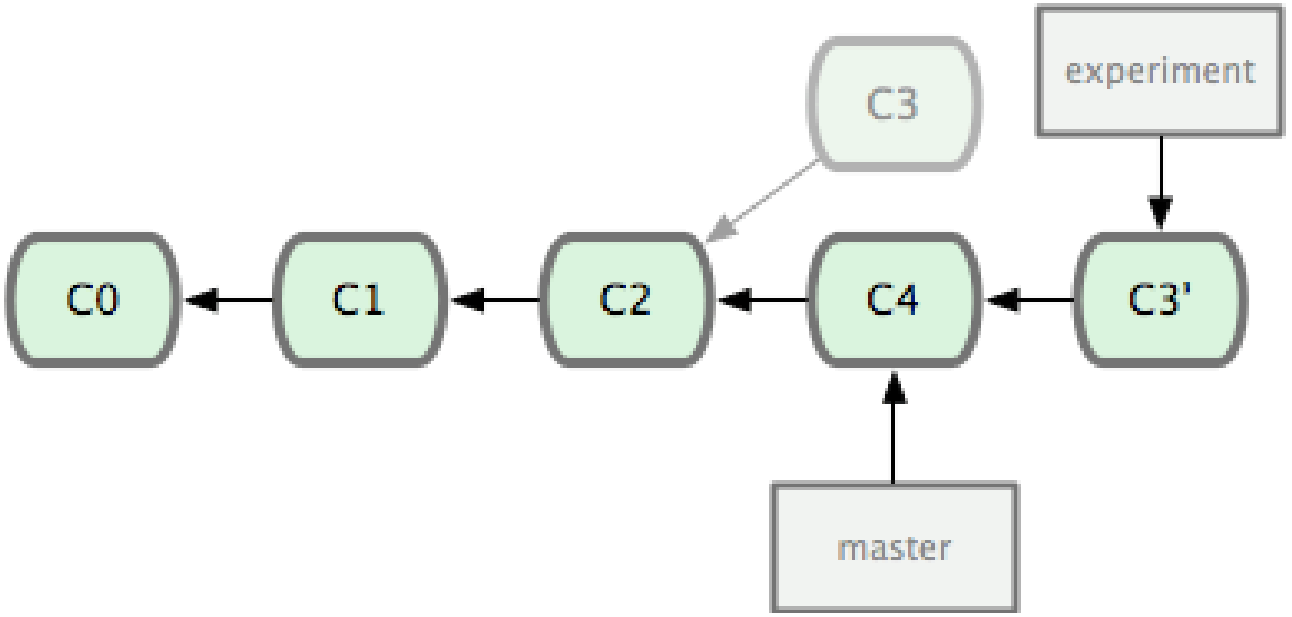
- Zrób pull brancha, do którego chcesz się mergować
- Często się merguj
- Jak najkrócej trzymaj nieaktualną wersję na swoim lokalnym repo

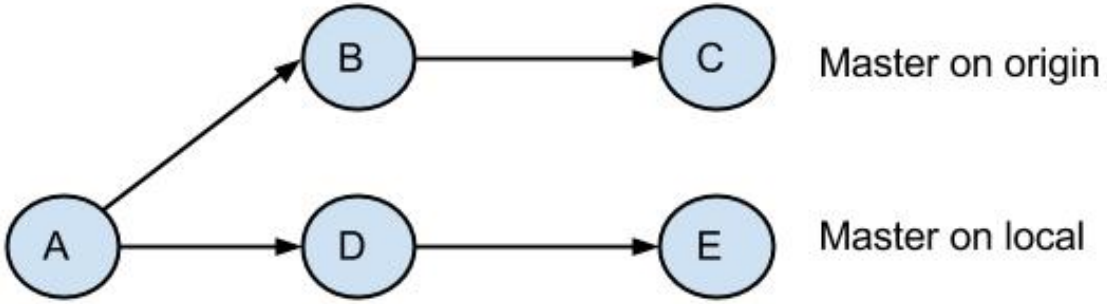
częste merge = mniej konfliktów

merge vs rebase

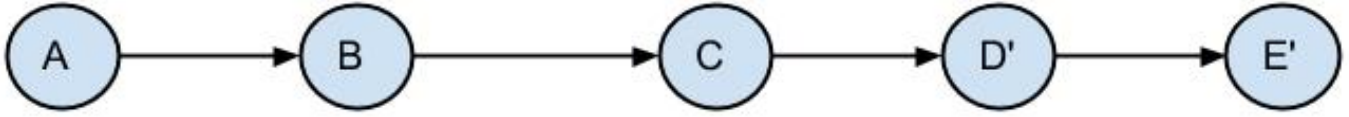
- Rebase zmienia bazę brancha
- Przekładamy po 1 commicie w górę
- Możliwe dużo konfliktów
- Czystsza historia rewizji
- rebase nanosi zmiany, w kolejności, w której były wprowadzane
- merge bierze 2 koncówki i je integruje ze sobą


experiment
C3
C0
C1
C2
C4
C5
master


C3
experiment
C0
C1
C2
C4
C3'
master



Before git pull --rebase



After git pull --rebase

Rebase

- „Nie zmieniaj bazy rewizji, które wypchnąłeś już do publicznego repozytorium.”
- rebase porzuca istniejące rewizje i tworzy nowe, które są podobne, ale inne

Worklow

- Git nie narzuca sposobu pracy
- Istnieje kilka popularnych praktyk
  - Centralized worklow
  - Feature branch workflow
  - Gitflow workflow

Centralized workflow

- Praca na jednym branchu (podobnie jak SVN)
- KONFLIKTY
- Brak podziału na kod rozwojowy i produkcyjny
- Trudna koordynacja przy wielu osobach

Feature branch workflow

- Naturalny podział zadań
- Każdy feature/bug ma swojego brancha
- Rozdzielenie kodu rozwojowego dla feature-ów
- Wiele osób = wiele branchy
- Weryfikacja przed wejściem do mastera
- Brak podziału na kod rozwojowy i produkcyjny

Git workflow

- Rozwinięcie ‘feature branch’
- Oddzielne branche na release’y
- Oddzielny branch na kod produkcyjny (master)
- Rozróżnienie produkcji z kodem rozwojowym (develop)
- Pełna informacja jaki kod w jakiej jest fazie

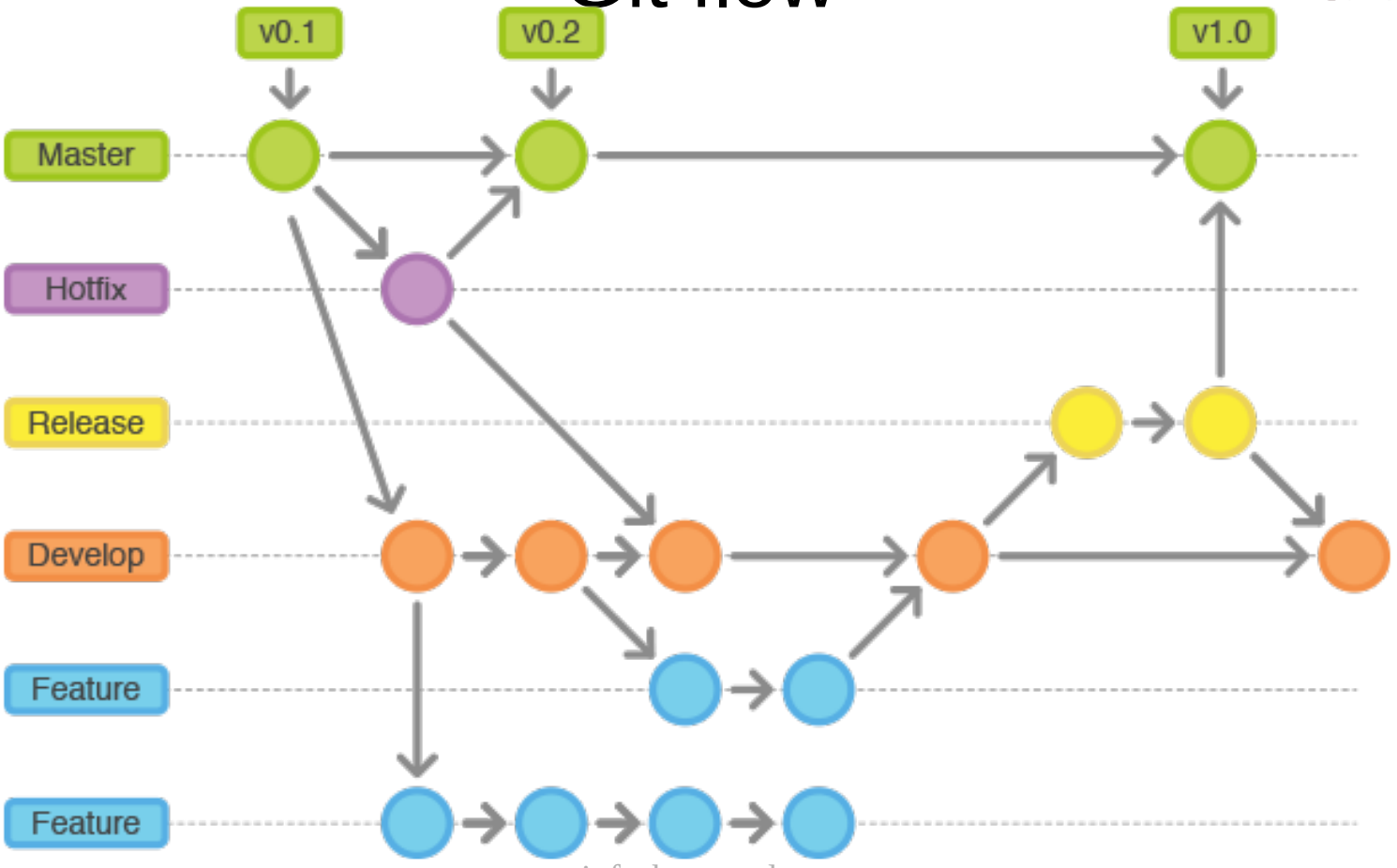
Git flow

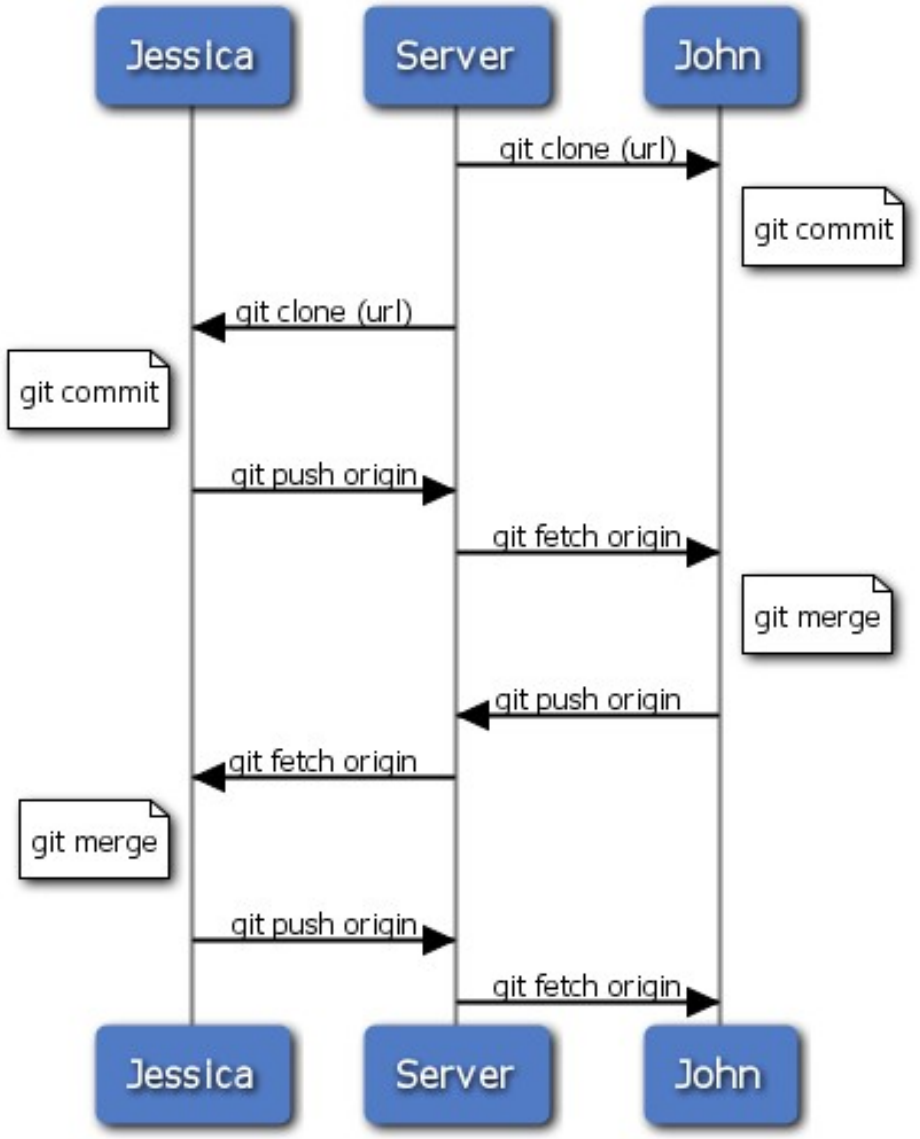
- Podział na branche wg odpowiedzialności
- Wszystkie nazwy są konwencją
- Często taski z JIRy
- Kontrakt między developerami, technicznie branche są takie same

Git flow

- **Develop** – główna gałąź rozwojowa, tutaj przygotowany kod do kolejnych wydań
- **Master –** główna gałąź produkcyjna, kod z tej gałęzi działa na serwerze i z niego korzystają klienci
- **FeatureBranch** – branch dla konkretnego feature’a, nazwa może być np. nazwą taska z jiry

Git flow



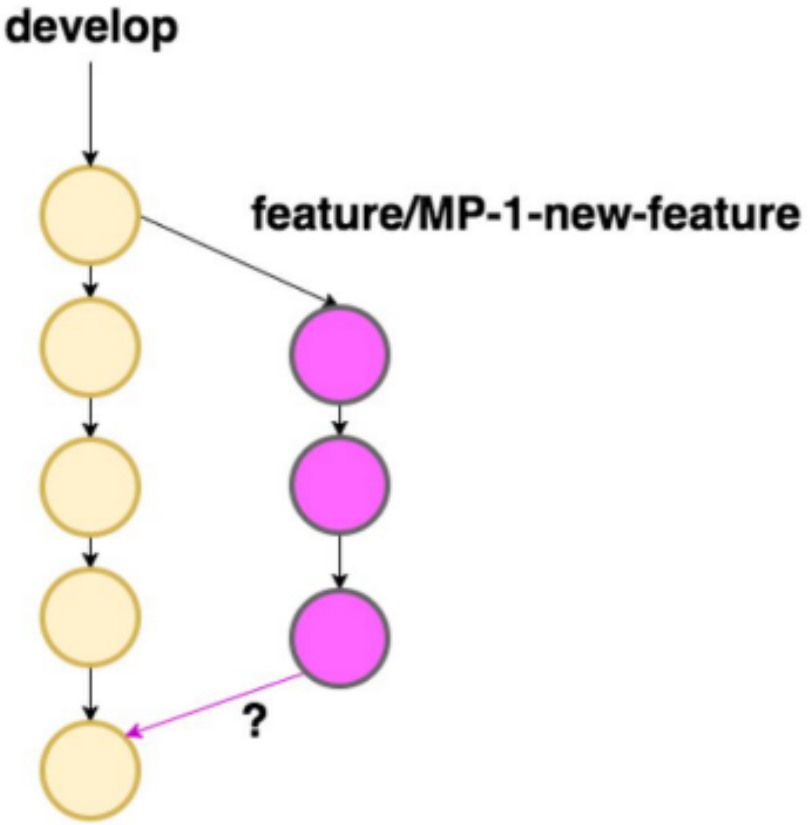

Jessica
Server
John
git clone (url)
git commit
git clone (url)
git commit
git push origin
git fetch origin
git merge
git push origin
git fetch origin
git merge
git push origin
git fetch origin
Jessica
Server
John

Pull request

- Żądanie wcielenia kodu z naszego brancha
- Koniec pracy nad danym feature’em kończy się pushem i stworzeniem pull request’a
- Pull request powinien zostać obsłużony przez innego programistę (lub kilku) z zespołu

Pull request

- W tym momencie kod powinien zostać zweryfikowany zanim zostanie złączony z develop’em

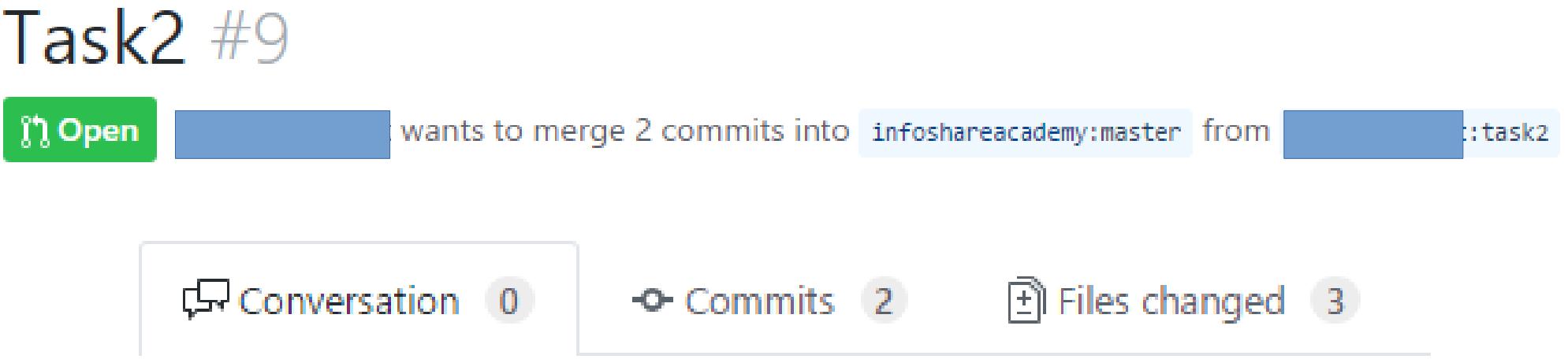


Github pull request

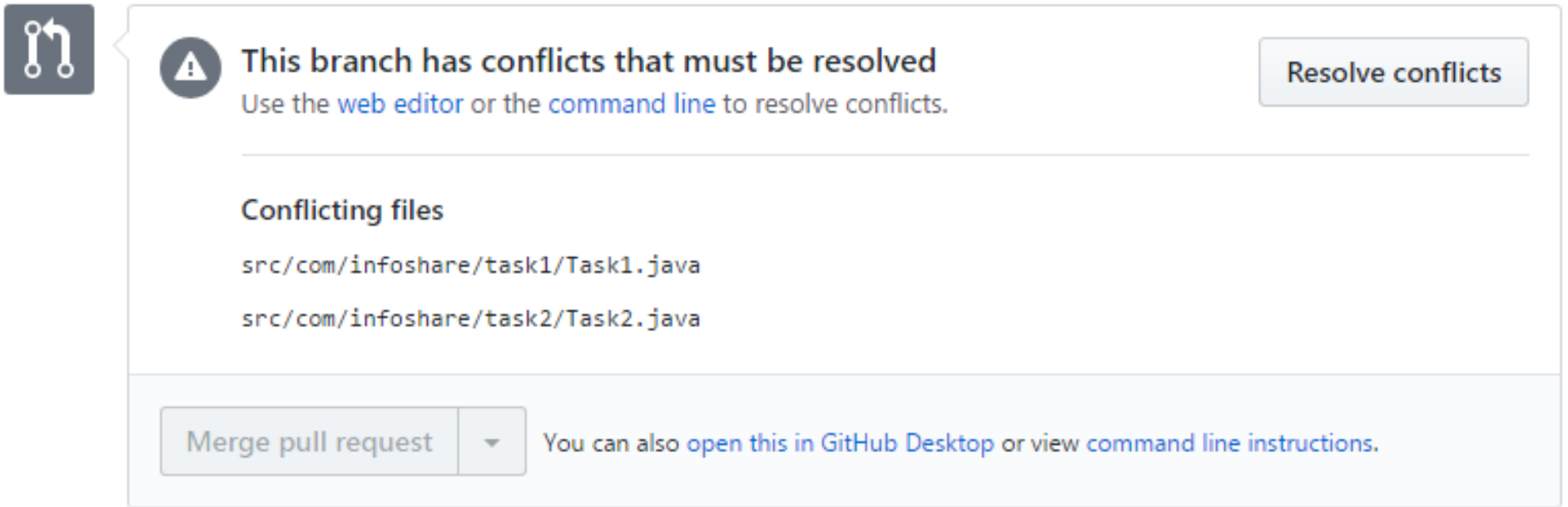
- W naszym repozytorium na githubie
- Nowy pull request dla już zrobionego pusha

Github pull request

- Otwarty pull request



Konflikty


This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts.
Resolve conflicts
Conflicting files
src/com/infoshare/task1/Task1.java
src/com/infoshare/task2/Task2.java
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

Schowek

- Gdy istnieją zmiany, nie można przełączyć brancha
- Schowek – stash
- Schowek zatrzyma wprowadzone zmiany
- Ale ‘wyczyści’ aktualny branch

Schowek

- git stash
- Możemy zachować wiele schowków jednocześnie

```
$ git stash
Saved working directory and index state \
  "WIP on master: 049d078 added the index file"
HEAD is now at 049d078 added the index file
(To restore them type "git stash apply")
```

Przywracanie schowka

- git stash apply
- Lub przywrócenie konkretnej wersji:
  git stash apply [numer]

```
$ git stash list
stash@{0}: WIP on master: 049d078 added the index file
stash@{1}: WIP on master: c264051... Revert "added file_size"
stash@{2}: WIP on master: 21d80a5... added number to log
```

Podsumowanie

- Najpopularniejszy system kontroli wersji
- Jest bardzo szybki
- Rozproszony, każdy developer na repozytorium u siebie
- Commity do delty (zmiany)
- Branche to tylko wskaźniki na commity
- Commity są stałe

Podsumowanie

- Fetch pobiera commity ze zdalnego repo
- Ale ich nie włącza
- Pull = fetch + merge
- Nie ma narzuconego workflow
- Ale istnieje dobry, wypracowany git flow
- Wiele usług hostingowych

Pytania?