



# System kontroli wersji GIT

Tomasz lisowski

tomasz.lisowski@protonmail.ch

Gdańsk, 06.06.2017

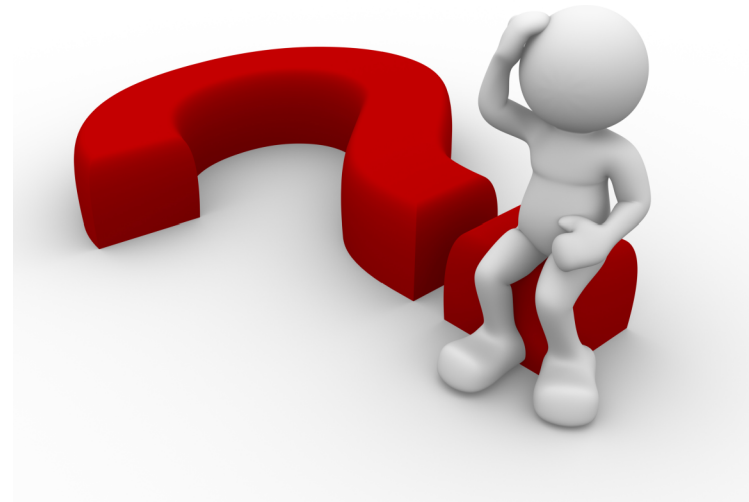
[www.infoshareacademy.com](http://www.infoshareacademy.com)

# Agenda

- Czym są systemy kontroli wersji
- Historia GIT
- Jak to działa?
- Konfiguracja
- Podstawowe operacje

# Problemy we wspólnej pracy

- Jak pisać oprogramowanie w wiele osób?
- Jak wrócić się „w czasie” do poprzedniej wersji kodu?
- Jak modyfikować te same pliki przez kilka osób?
- Ten sam projekt na kilku różnych maszynach



# Systemy kontroli wersji



## Distributed (rozproszone)

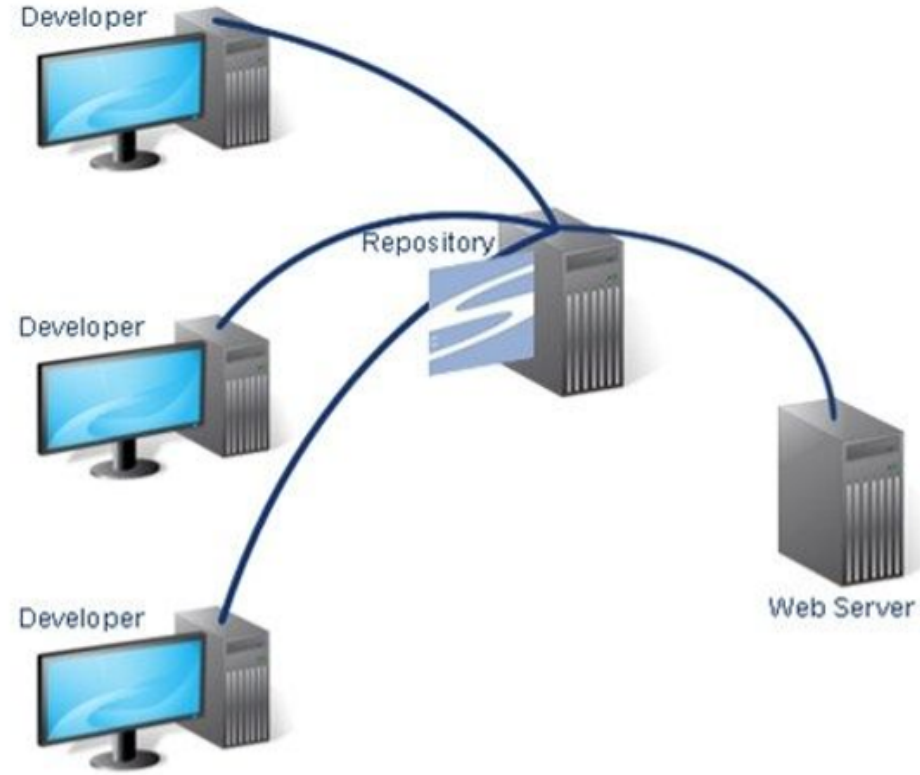


## Client-Server



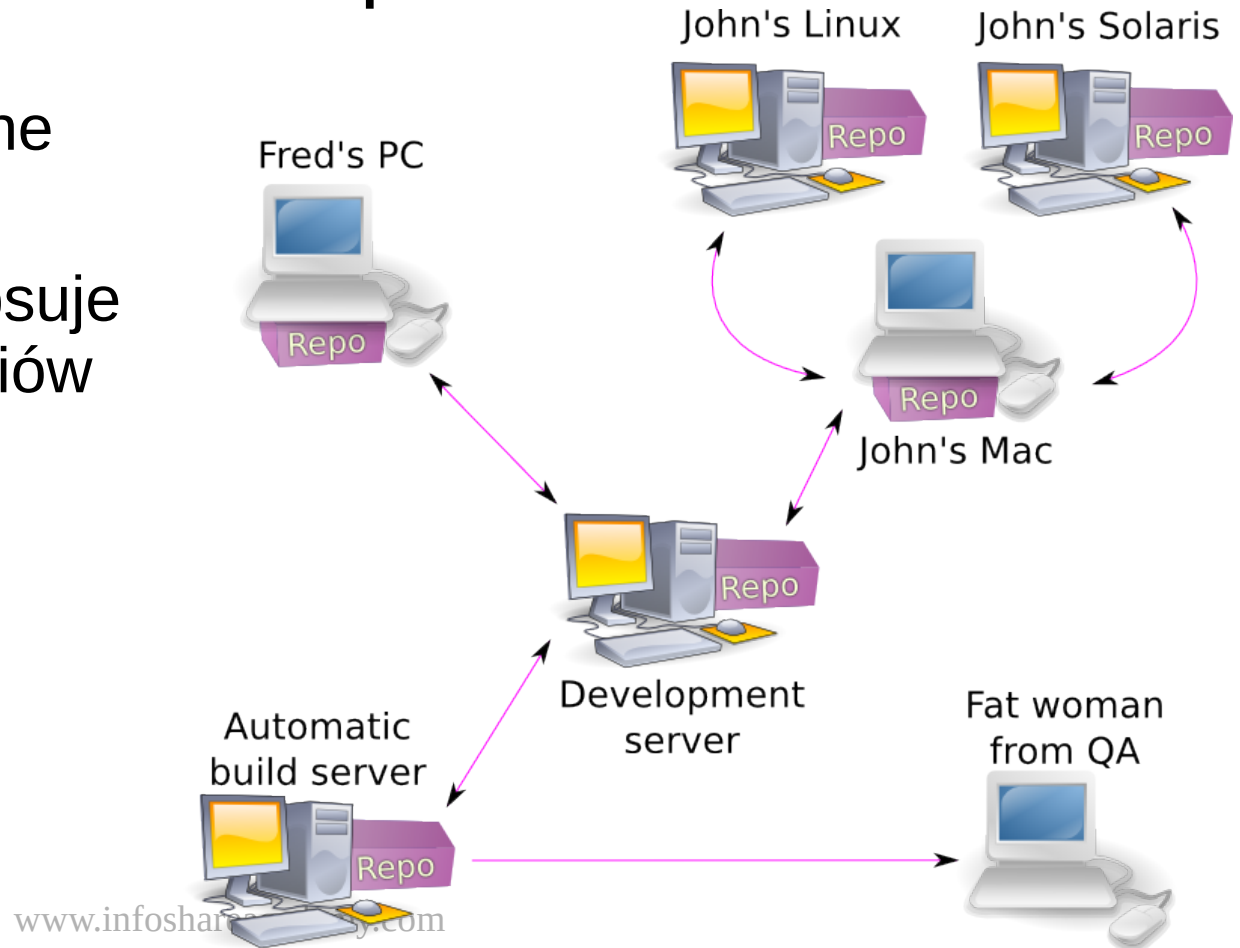
# Client-server: Centralne

- Tylko jedno repozytorium na serwerze
- Wersja synchronizowana zawsze na serwerze
- Jeżeli ktoś/coś się zepsuje na serwerze, to jesteśmy w... trudnym położeniu



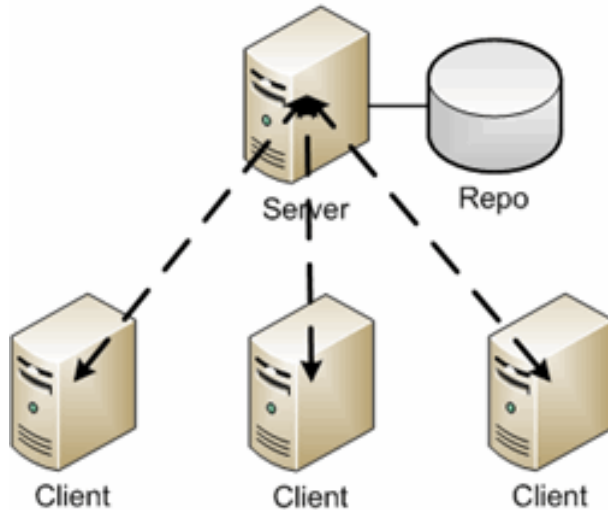
# Distributed: Rozproszone

- Każdy ma swoje lokalne repozytorium
- Jeżeli ktoś/coś się zepsuje – mamy tyle repozytoriów ilu developerów



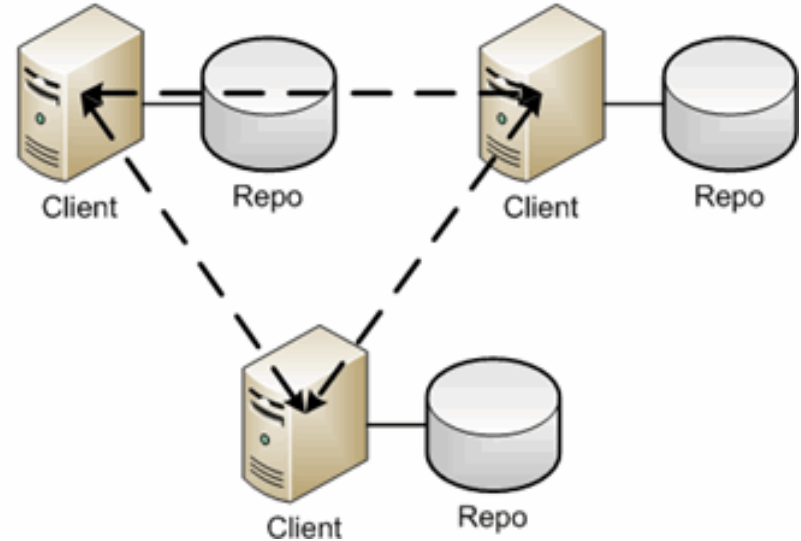
# Systemy kontroli wersji

## Traditional



Istnieje tylko jedno  
centralne repozytorium

## Distributed



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



# Dawno dawno temu..

Od czego zaczęła się historia GIT?  
Jakie są jego najważniejsze cechy?  
Kto za to odpowiada?



# Jak to się zaczęło?

Początki w 2005 roku

narzędzie do wersjonowania  
kodu podczas prac nad  
Linuxem



# Jak to się zaczęło?

Dlaczego?

- Wydajność i prędkość narzędzia
- Śledzenie zmian, a nie plików
- Podejście od strony systemu plików

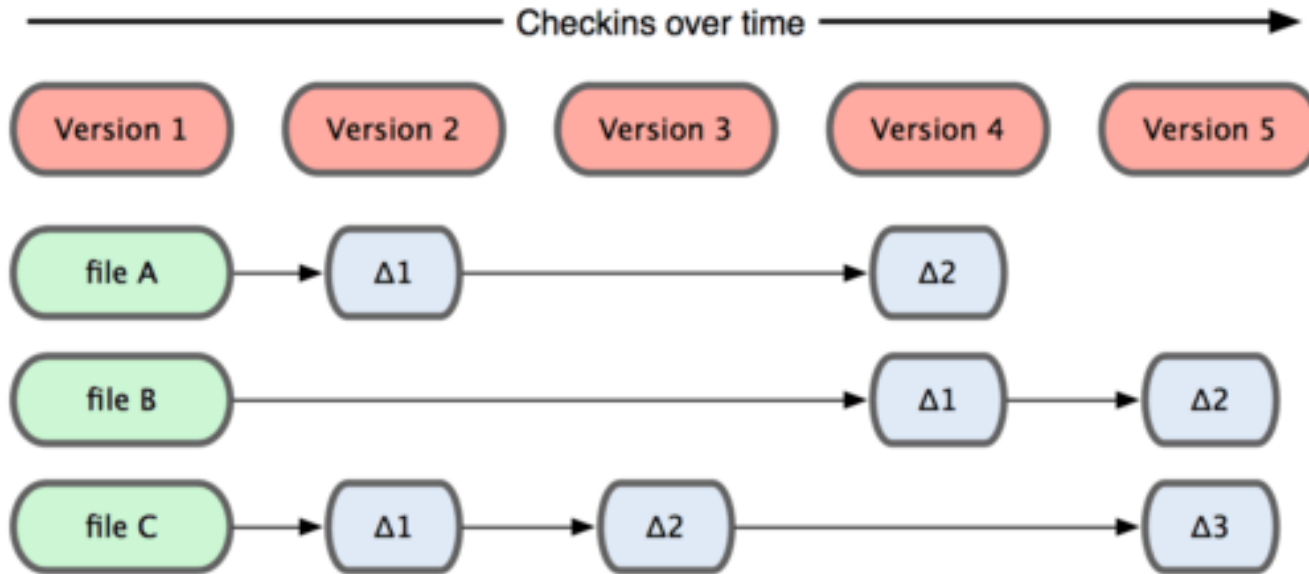


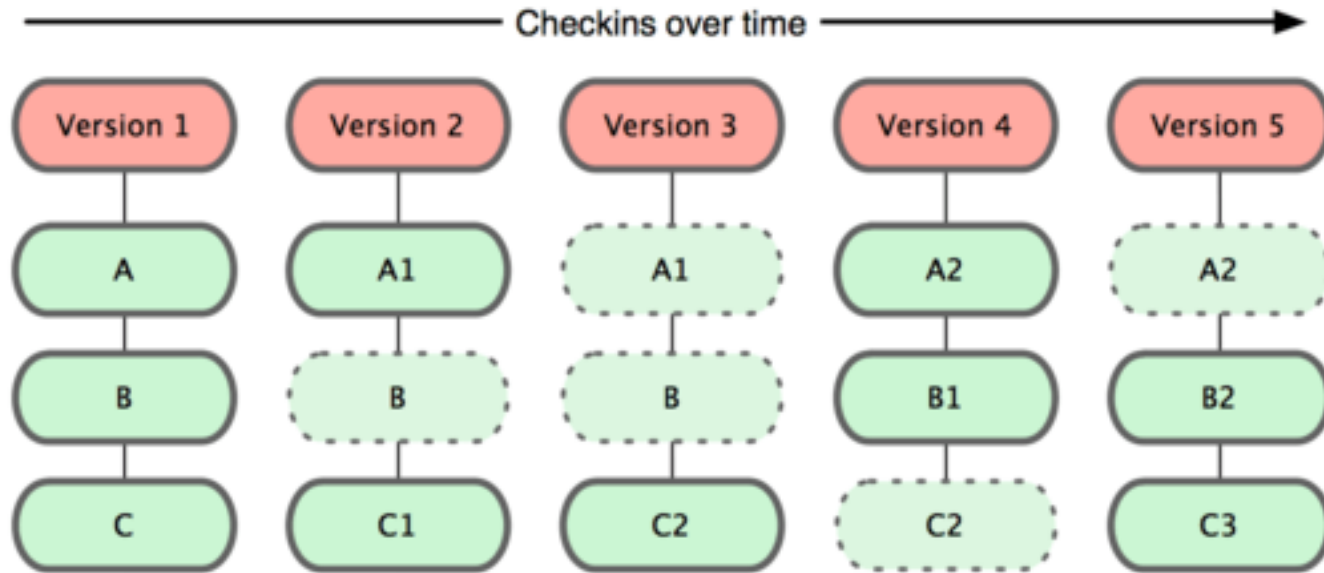
# Cechy

- Dobre wsparcie łączenia zmian (rozgałęziony proces)
- Praca offline – praca na lokalnym repozytorium
- Wsparcie protokołów sieciowych (HTTP, FTP, SSH)
- Szybki – efektywna praca przy dużych projektach
- Nie zapamiętuje zmian, tylko obrazy (migawki)

# 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





# Same zalety

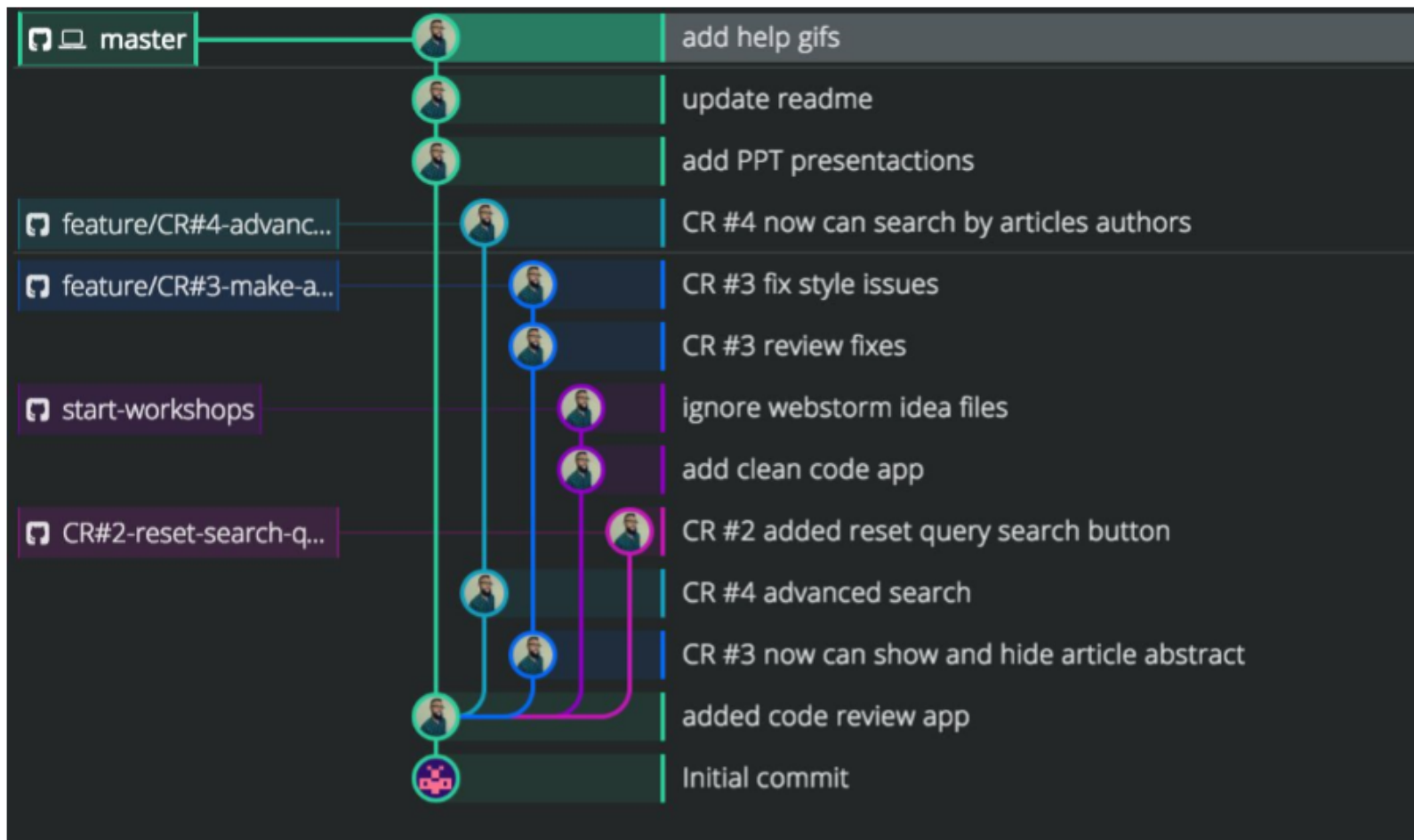
- Rozproszony
- Zoptymalizowany pod kątem wydajności
- Bezpieczny
- Elastyczny (nieliniowe flow pracy)
- Darmowy i rozwijany jako OSS
- Aktualny standard



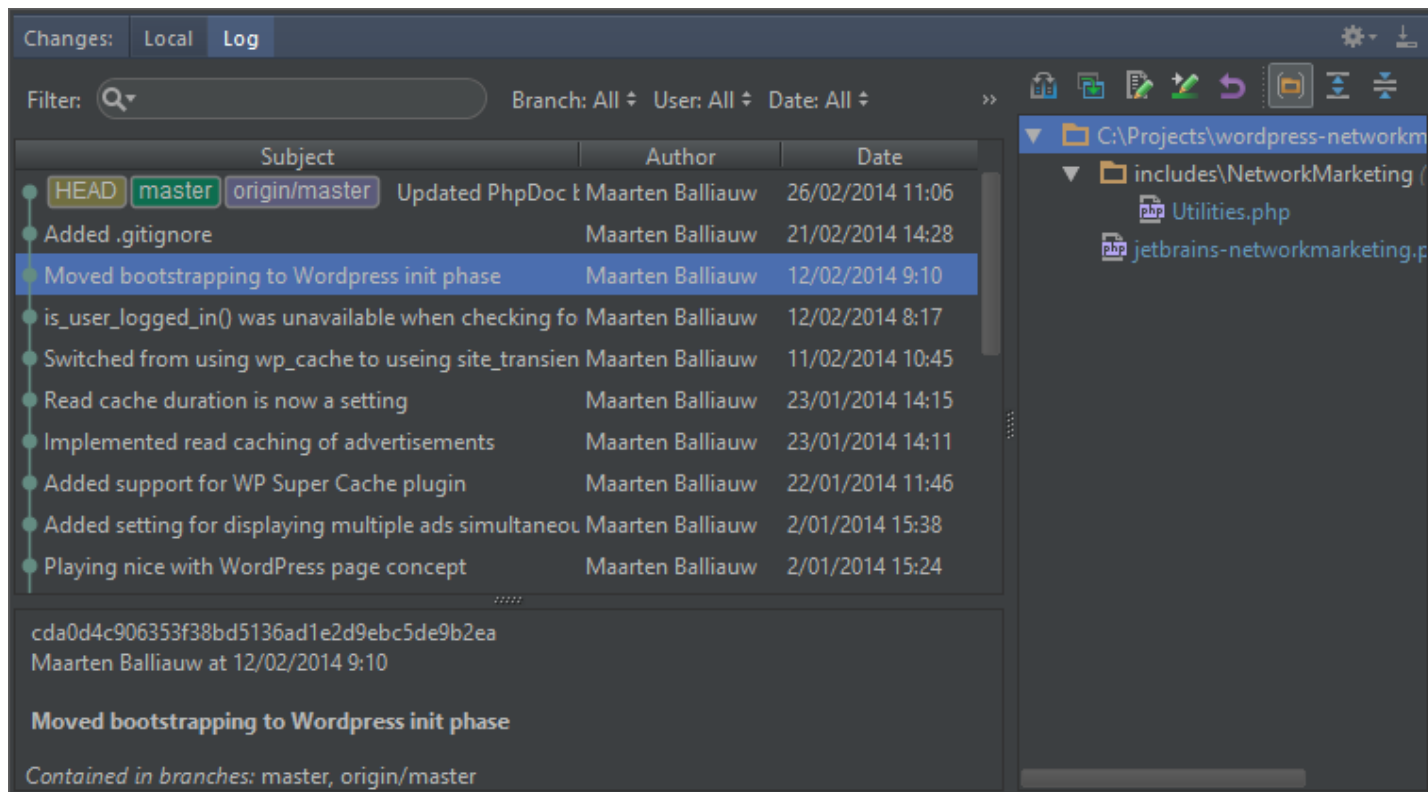
# A może też wady?

- Skomplikowany model informacji
- Słaba dokumentacja
- Można zgubić efekty swojej pracy

# Wersjonowanie i historia zmian



# Samodzielna praca



The screenshot displays a Git log interface within a code editor. The top bar shows 'Changes: Local Log'. Below it, a search filter and branch selection tools are visible. The main log table lists commits with columns for Subject, Author, and Date. The commit 'Moved bootstrapping to Wordpress init phase' is highlighted in blue. To the right, a file explorer shows the project structure, including 'includes\NetworkMarketing' and 'Utilities.php'. The bottom panel shows the commit hash 'cda0d4c906353f38bd5136ad1e2d9ebc5de9b2ea' and the message 'Moved bootstrapping to Wordpress init phase'.

Subject	Author	Date
HEAD master origin/master Updated PhpDoc t	Maarten Balliauw	26/02/2014 11:06
Added .gitignore	Maarten Balliauw	21/02/2014 14:28
Moved bootstrapping to Wordpress init phase	Maarten Balliauw	12/02/2014 9:10
is_user_logged_in() was unavailable when checking fo	Maarten Balliauw	12/02/2014 8:17
Switched from using wp_cache to using site_transien	Maarten Balliauw	11/02/2014 10:45
Read cache duration is now a setting	Maarten Balliauw	23/01/2014 14:15
Implemented read caching of advertisements	Maarten Balliauw	23/01/2014 14:11
Added support for WP Super Cache plugin	Maarten Balliauw	22/01/2014 11:46
Added setting for displaying multiple ads simultaneou	Maarten Balliauw	2/01/2014 15:38
Playing nice with WordPress page concept	Maarten Balliauw	2/01/2014 15:24

cda0d4c906353f38bd5136ad1e2d9ebc5de9b2ea  
Maarten Balliauw at 12/02/2014 9:10

**Moved bootstrapping to Wordpress init phase**

Contained in branches: master, origin/master

# Narzędzia

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

# Konfiguracja

Ćwiczenie:

- `git config --global user.name "Your Name"`
- `git config --global user.email your.name@domain`
- `git config --global --list`

# Pomoc

- Instrukcja dla wybranego polecenia
- `git help <polecenie>`
- `git <polecenie> --help`
- `man git-<polecenie>`

Ćwiczenie:

np. `git help config`

# Podstawowe operacje

- Clone – kopiuje istniejące repozytorium
- Commit – zatwierdza zmiany
- Pull – pobranie i włączenie zmian
- Add – dodaje pliki do śledzenia
- Init – rozpoczyna śledzenie zmian
- Status – sprawdza stan plików

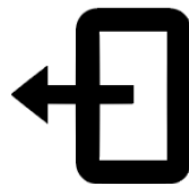
In case of fire



1. `git commit`



2. `git push`

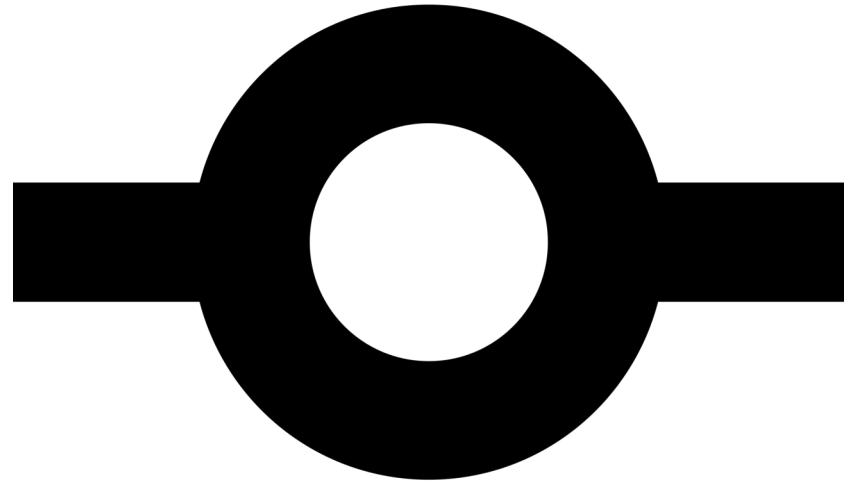


3. leave building



# Commit

- Zapisanie różnicy zawartości plików
- Committed stałe i niezmiennie
- Odpowiadają konkretnym zmianom
- Mają wiadomości, które je opisują



# Dobry commit

- Odpowiednia wiadomość
- Zmiana odpowiadającej jednej rzeczy
- Z wiadomości można jednoznacznie określić czego dotyczy
- .. ale wiadomość nie może być zbyt długa
- ani zbyt szczegółowa

# Dobry commit



**Michał Michalczuk** committed `e4ec315415c` 24 Aug 2016

`SAL-1409 now hide terms and conditions step for limited sellers`



**Michał Michalczuk** committed `897c196dfd4` Yesterday

`SAL-1524 now comments authors names are localized`



**Michał Michalczuk** committed `92c60d88173` 31 Aug 2016

`SAL-1184 now bid confirmation popup registers for events`

# Zł commit

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

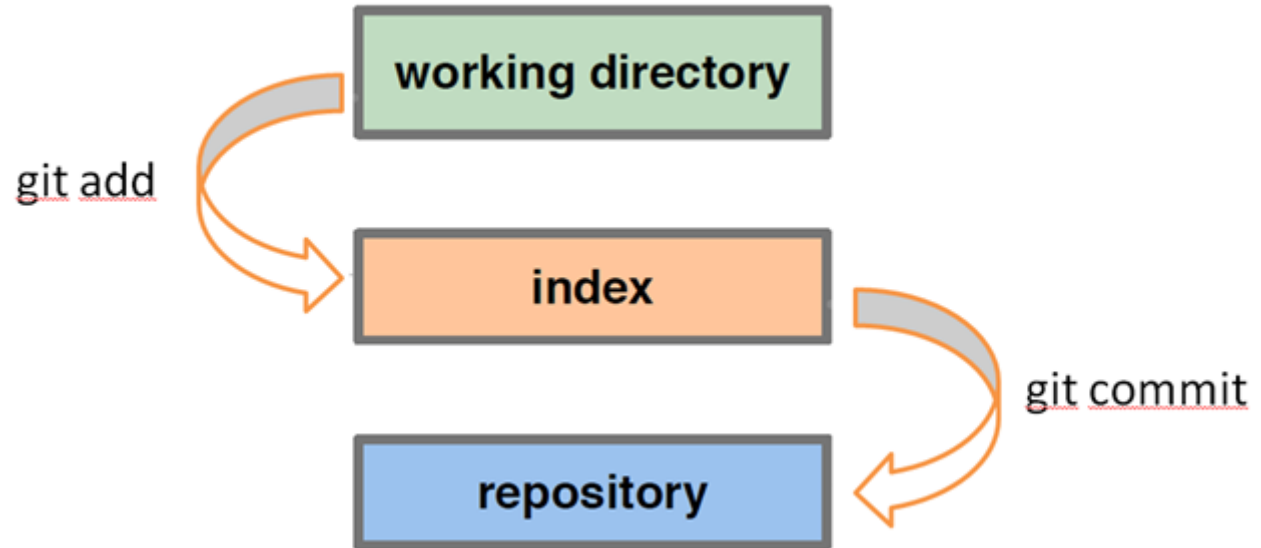
AS A PROJECT DRAGS ON, MY GIT COMMIT  
 MESSAGES GET LESS AND LESS INFORMATIVE.

# Ćwiczenie

- VCS > checkout from Version Control > GitHub
- Logowanie swoimi danymi
- wybieramy repozytorium jfdd7-materialy-git
- jeśli nie istnieje Parent Directory, to go tworzymy, np. jako /home/<nazwa\_użytkownika>/workspace

# Ćwiczenie

- Wprowadź dowolną zmianę (nowy katalog/plik, modyfikacja)
- `git status`
- `git add`
- `git status`
- `git commit`



# Efekt?

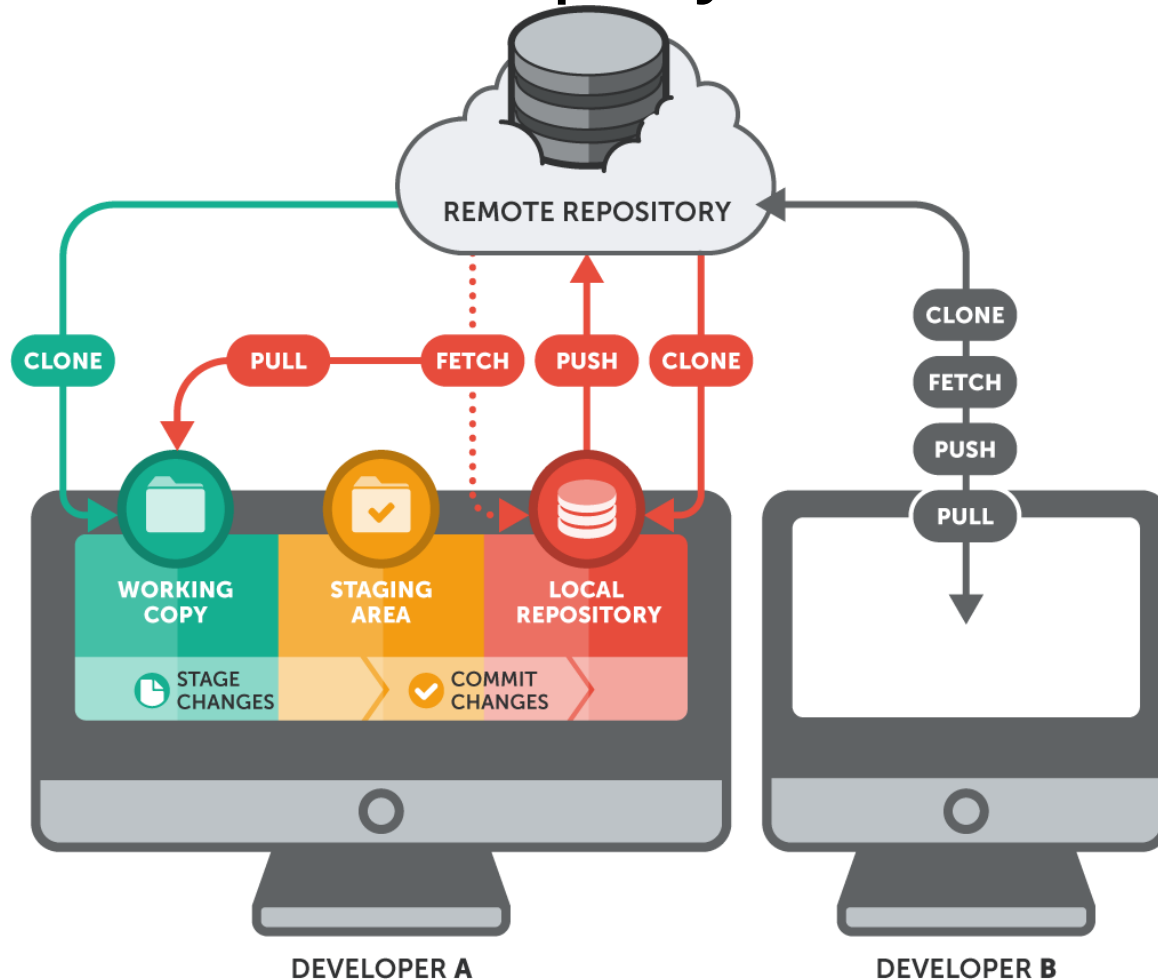


# Commit

- Każdy commit ma swojego rodzica
- Ma też swój UNIKALNY klucz SHA1
- Można przejrzeć krok po kroku jak zmieniał się kod
- Podgląd zmian poszczególnych plików



# Zdalne repozytorium



# Zdalne repozytorium

Repozytorium, z którym chcemy synchronizować/wymieniać się z naszym lokalnym repozytorium.



# Usługi hostujące



# Git czy github?



**git**



**github**  
SOCIAL CODING

# Zdalne repozytorium

- Lokalne repo musi znać adres zdalnego
- Możemy mieć podpięte wiele repozytoriów
- origin – domyślna nazwa zdalnego repo

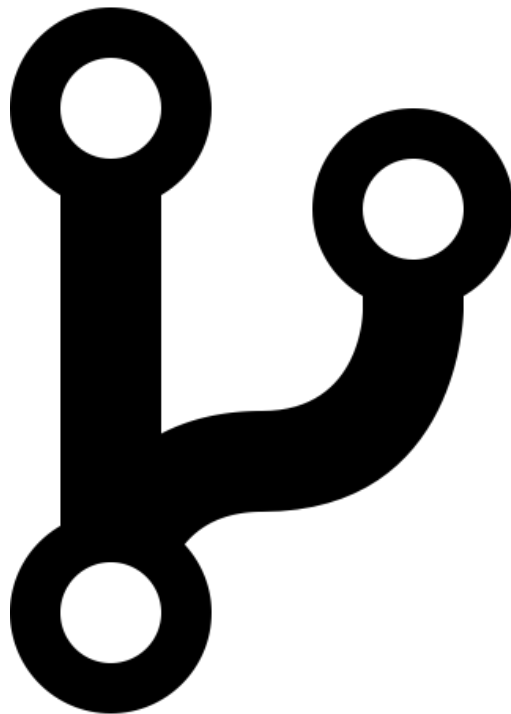


# Pobieranie

- 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

# Branchowanie



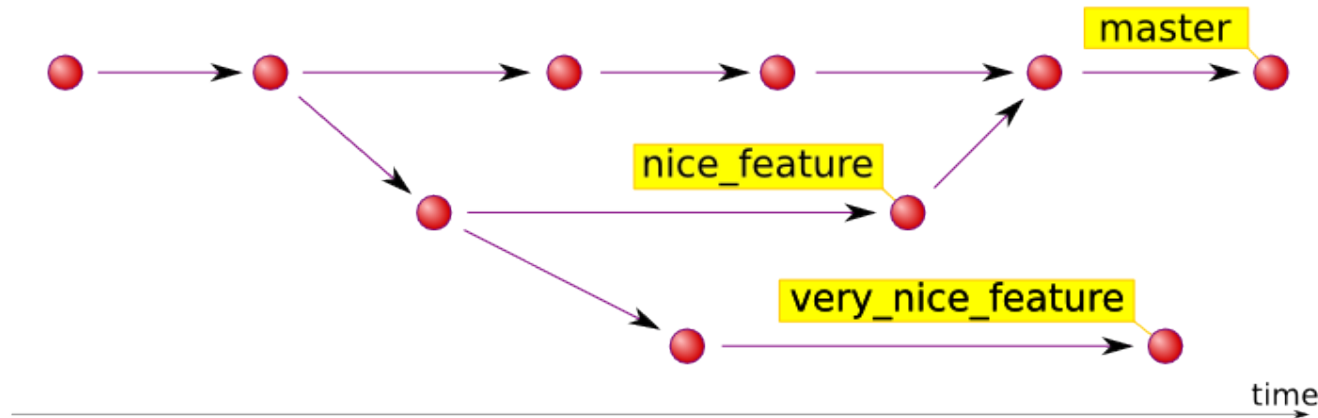
# Branch (gałąź)

- git branch
- Wskaźnik na commit
- Można go zawsze dodać/usunąć/zmienić



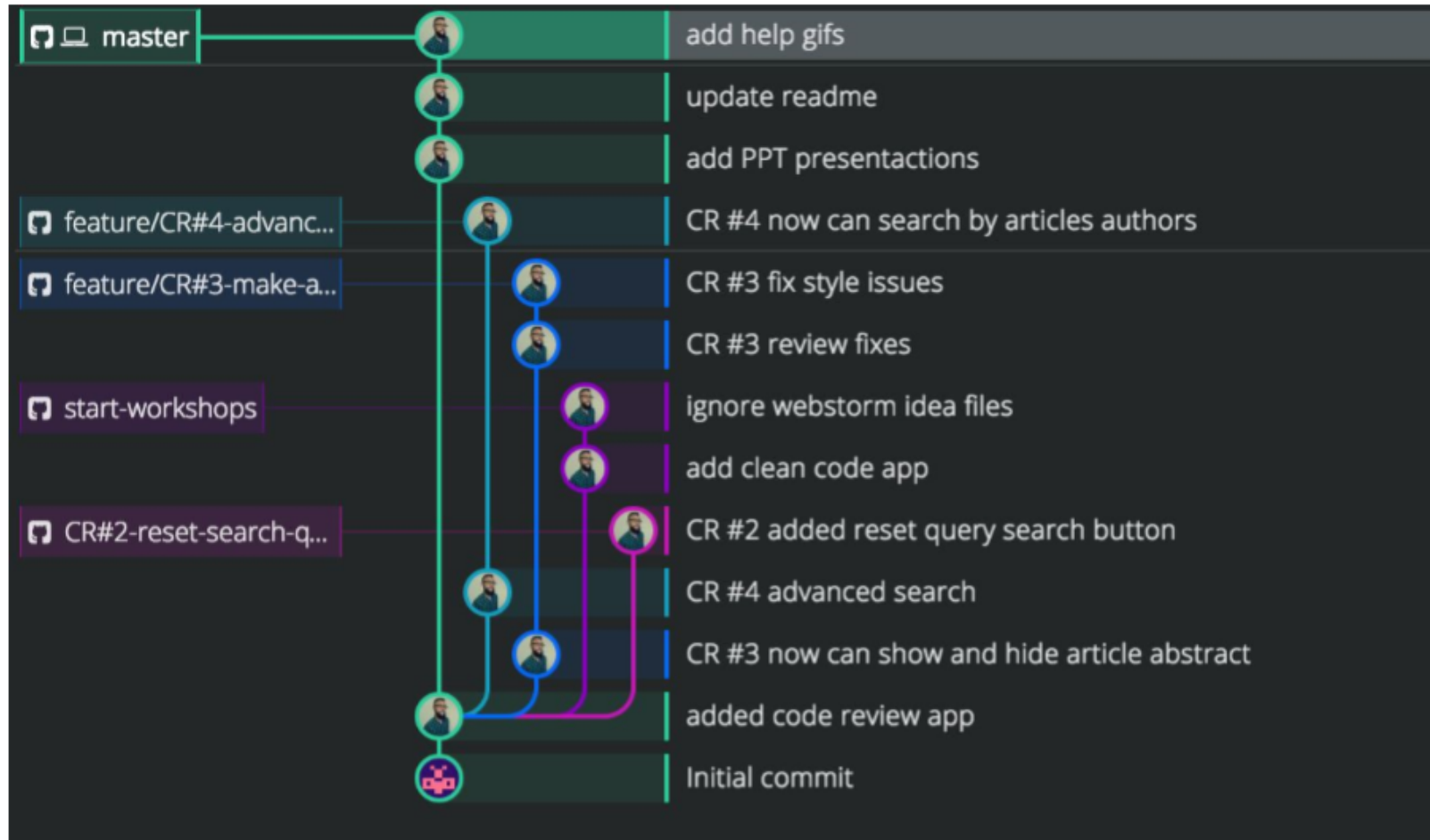
# Commit vs branch

- 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

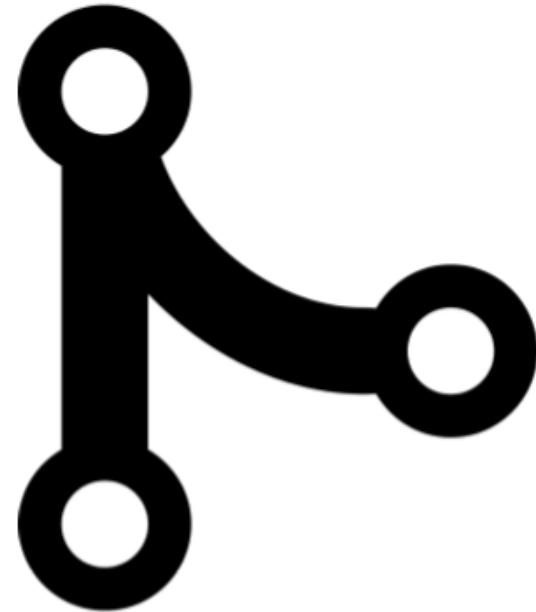


# „jestem na branchu”

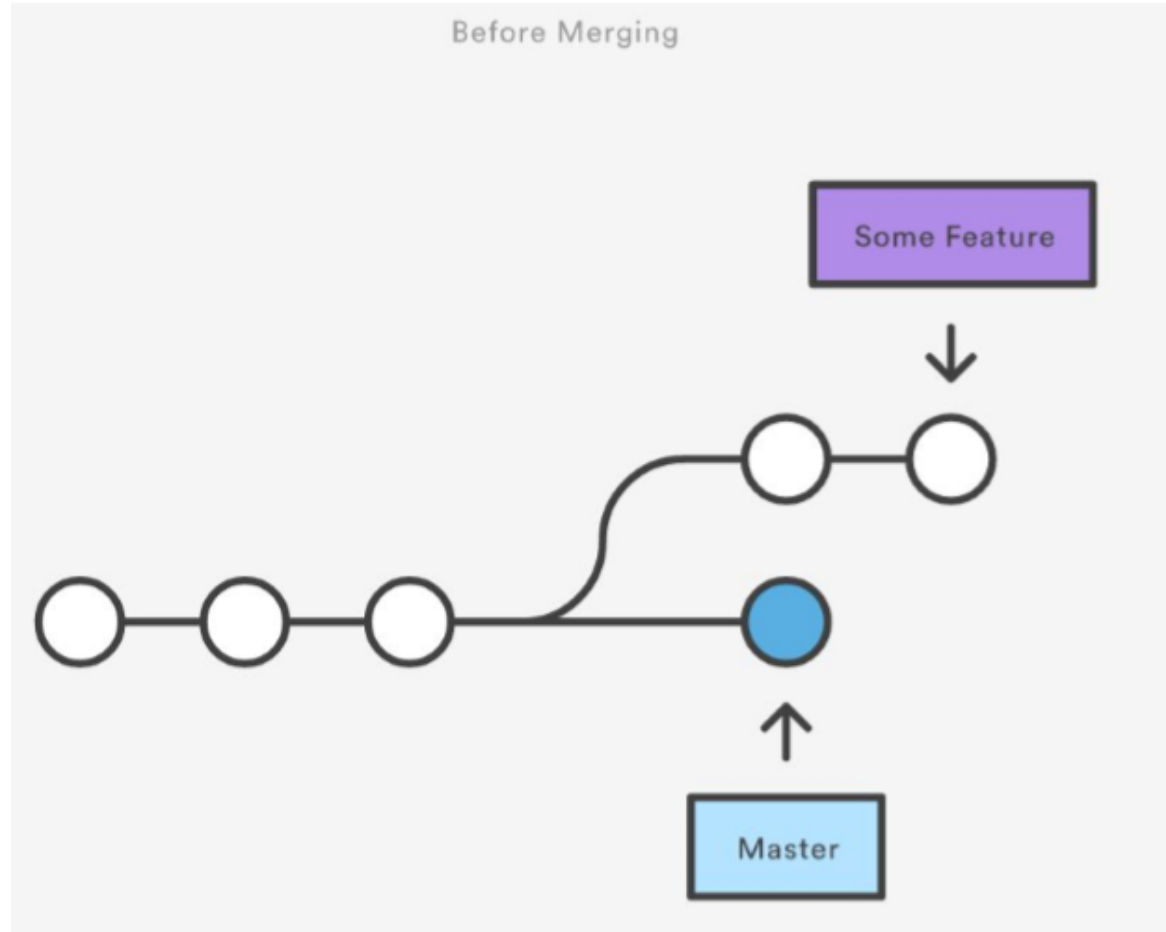
- Stan kodu odpowiada commitowi, na którym jest branch
- Można się przełączać między branchami
- Ćwiczenie:
  - Stwórz nowy branch
  - Wprowadź zmiany
  - Przełącz się na poprzedni branch

# Merge

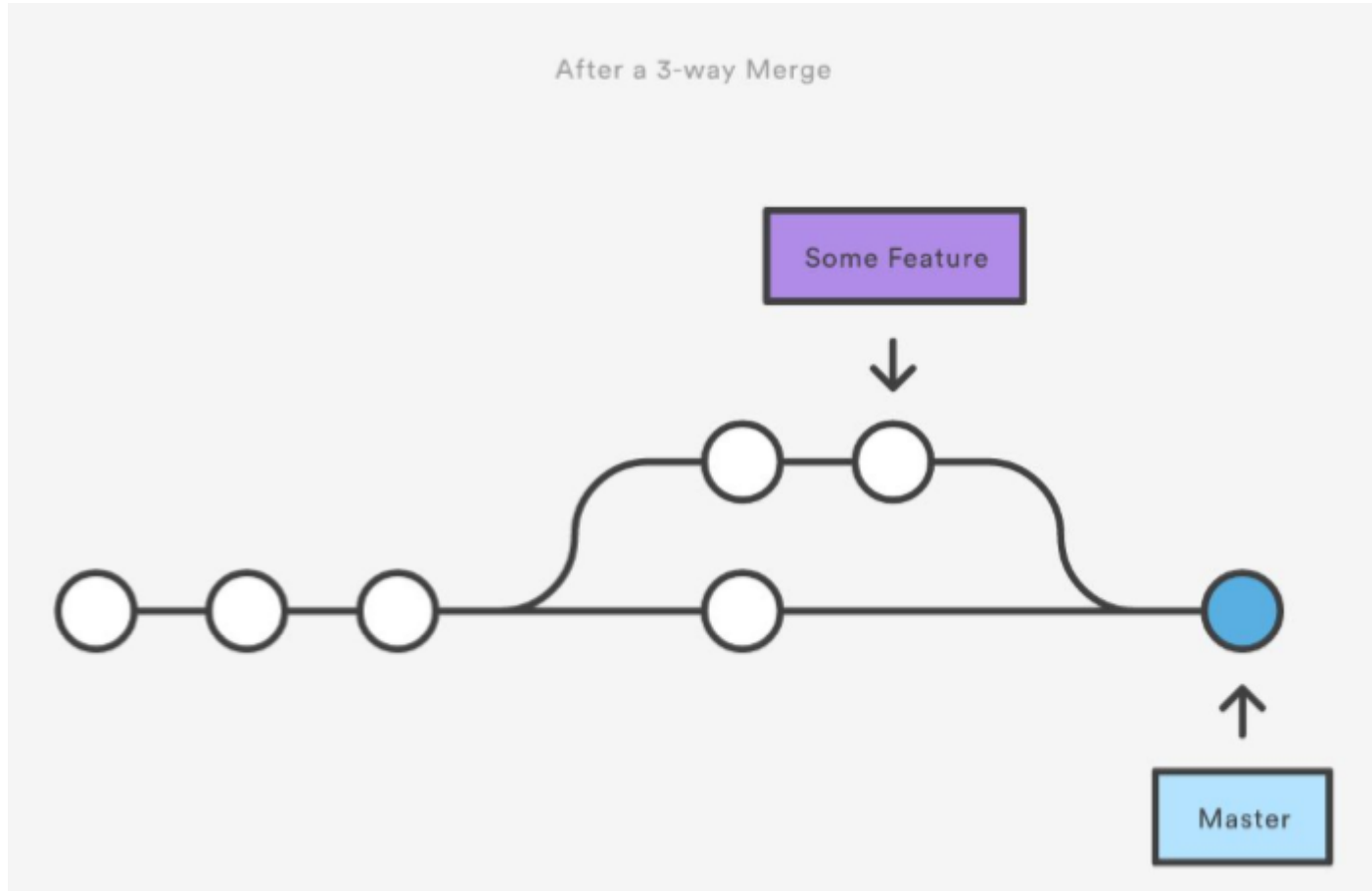
- Połączenie/mieszanie ze sobą różnych branchy
- Łączenie kodu w całość
- Potencjalne konflikty



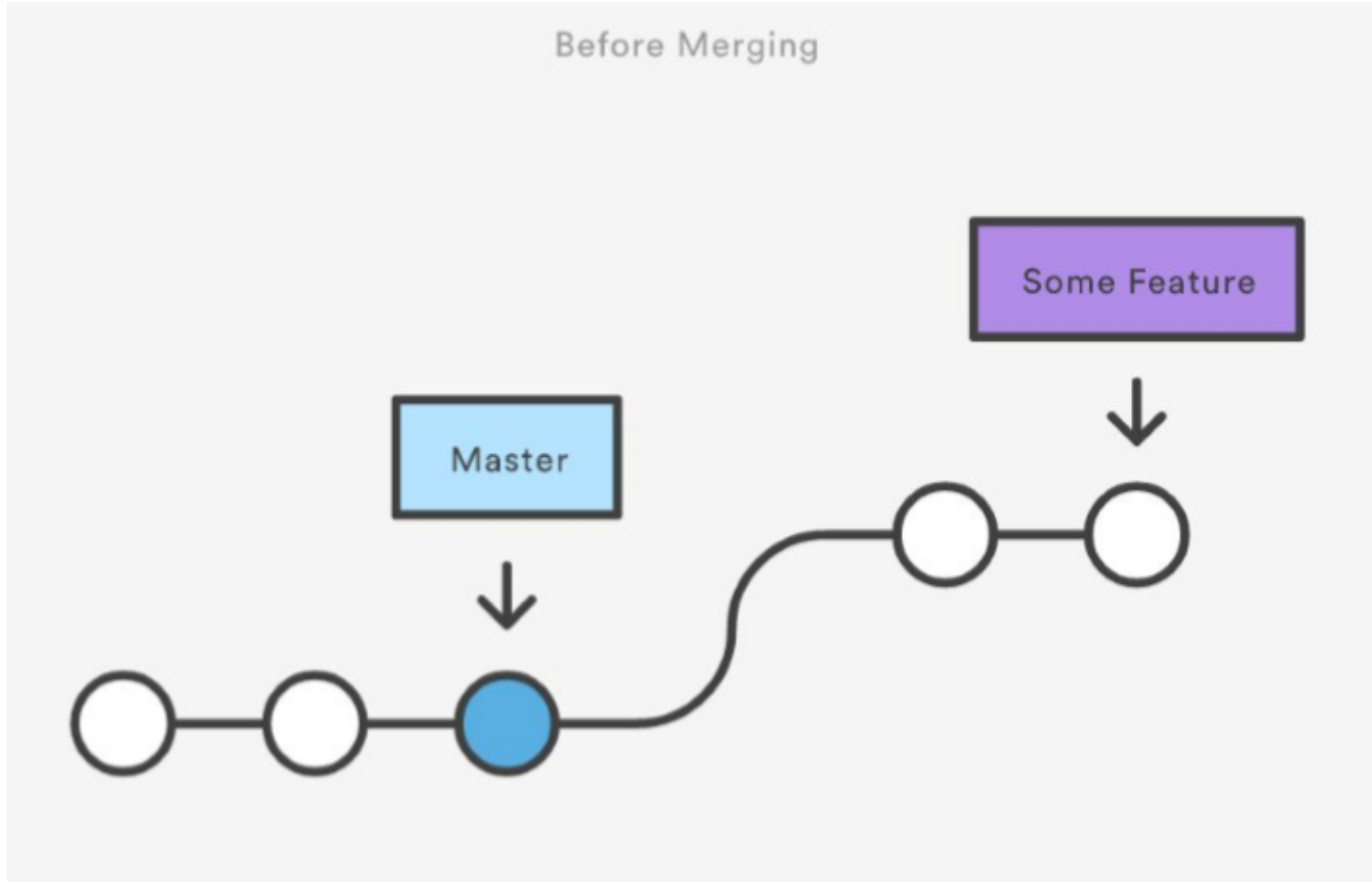
# 3 way merge



# 3 way merge

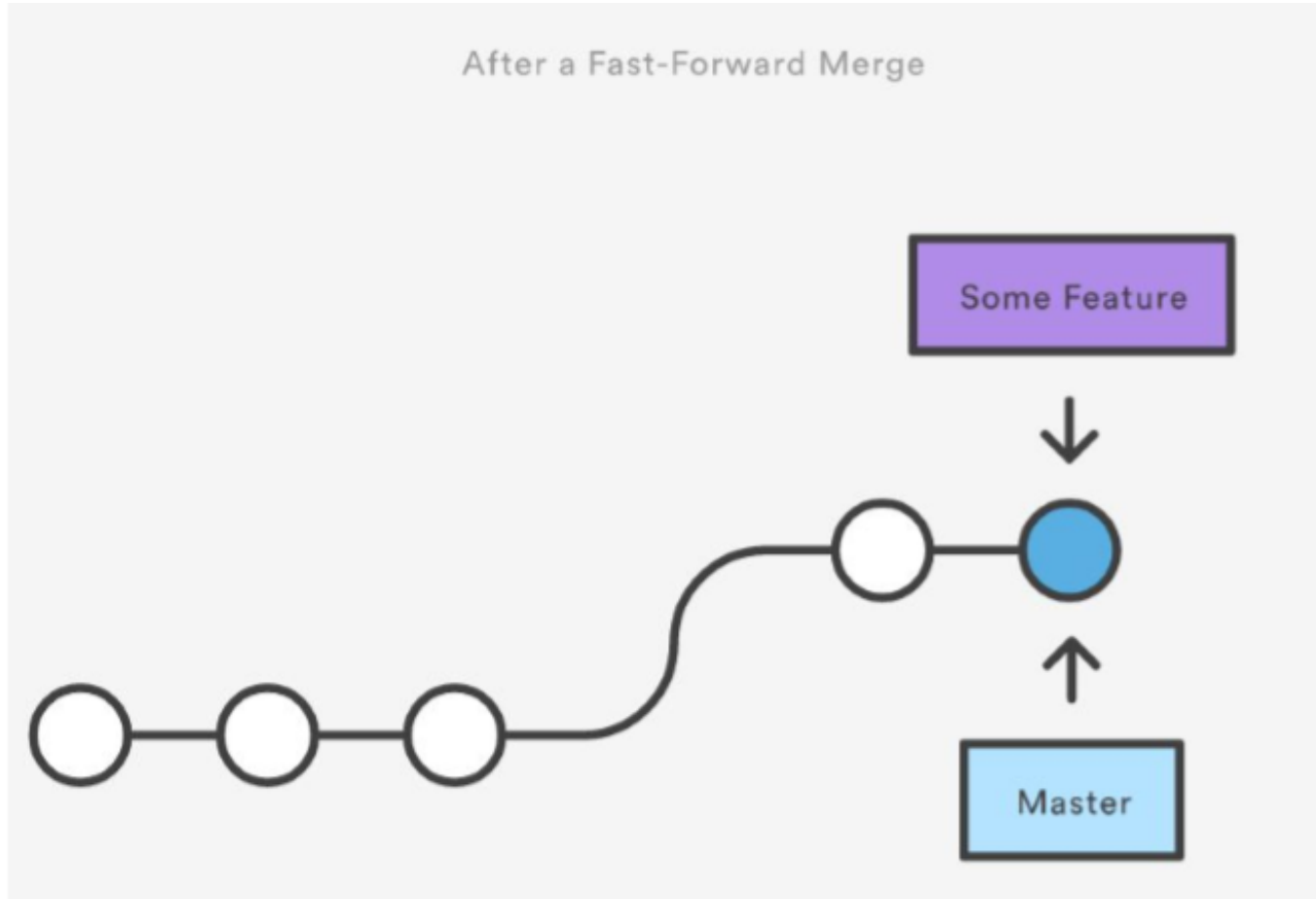


# Fast forward



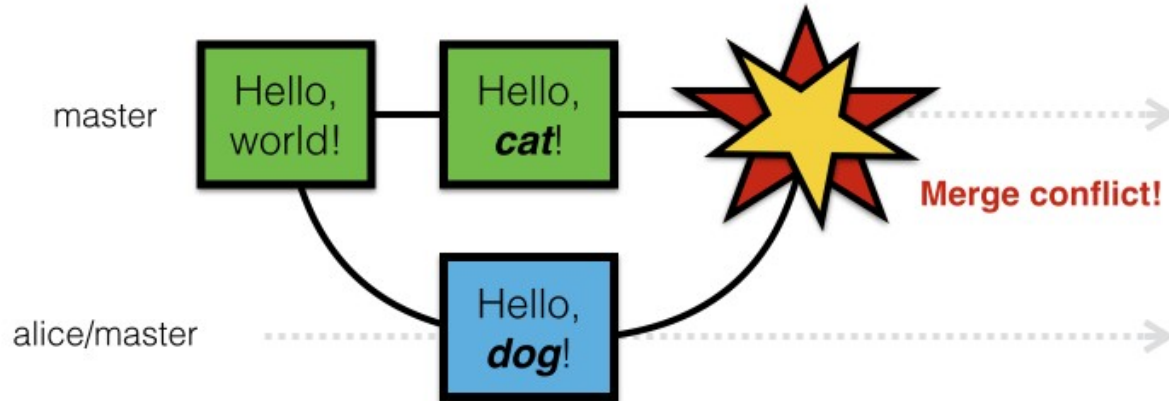


# Fast forward



# Merge

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



# 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

# 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

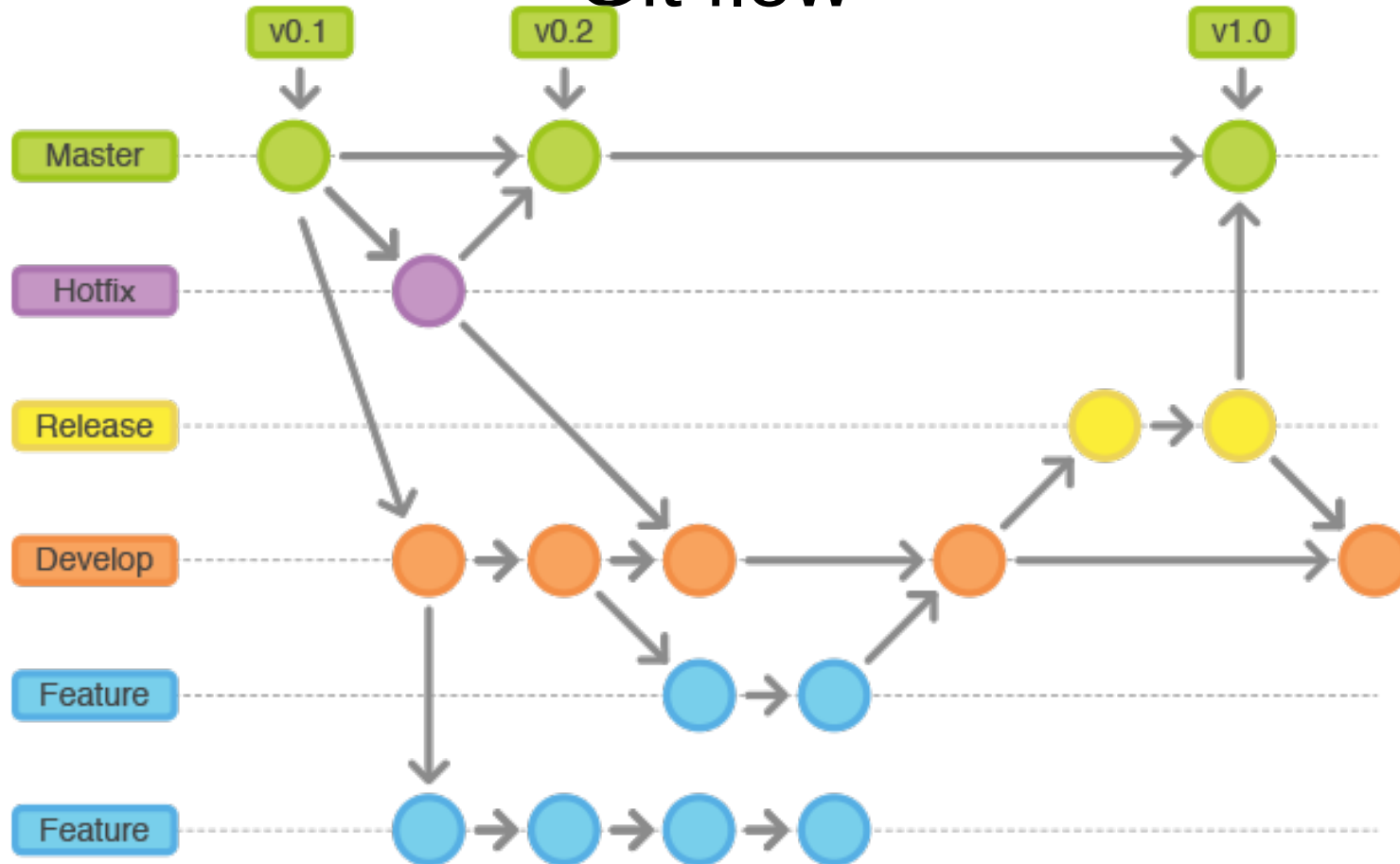
# Git flow

- 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



# Workflow

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



Pytania?

