



GIT

System kontroli wersji GIT

Trener: Michał Michalczuk

Gdańsk, 29 października 2016 roku

Plan na dzisiaj

- Czym są systemy kontroli wersji i co nam dają
- Jak działa GIT?
- [Live] Podstawowe operacje w GIT
- [Live] Kiedy pracujemy razem, czyli konflikty
- [Tool] GIT w WebStorm
- Jak wydajnie pracować w wiele osób?
- Gdzie trzymać moje repozytorium? Czyli czym jest ten GitHub

Problem

- Jak pisać oprogramowanie w wiele osób?
- Jak wrócić się "w czasie" do poprzedniej wersji kodu?
- Jak modyfikować te same pliki przez parę osób?



Problem jest dosyć powszechny

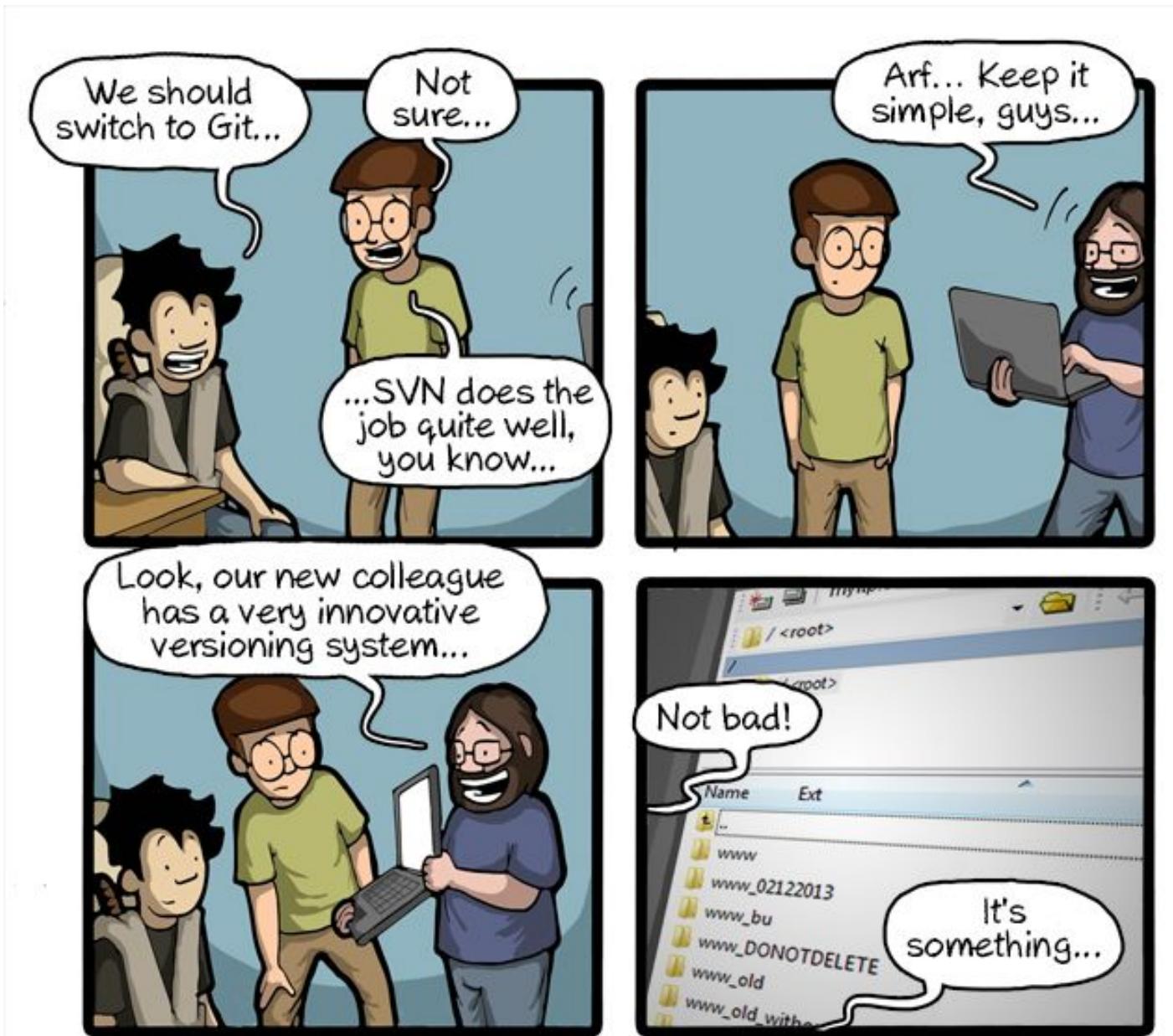


Wg wikipedii istnieje/istniało 45 różnych systemów kontroli wersji.
Znanych.

https://en.wikipedia.org/wiki/List_of_version_control_software

Jak wy sobie radziliście z tym problemem?



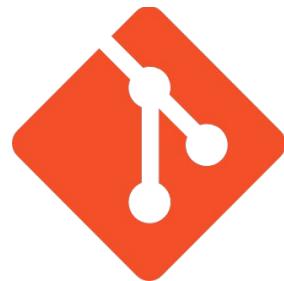


Systemy kontroli wersji



Systemy kontroli wersji

Distributed (rozproszone)



BITKEEPER
Scalable Version Control



mercurial

Client-Server



PERFORCE
Version everything.



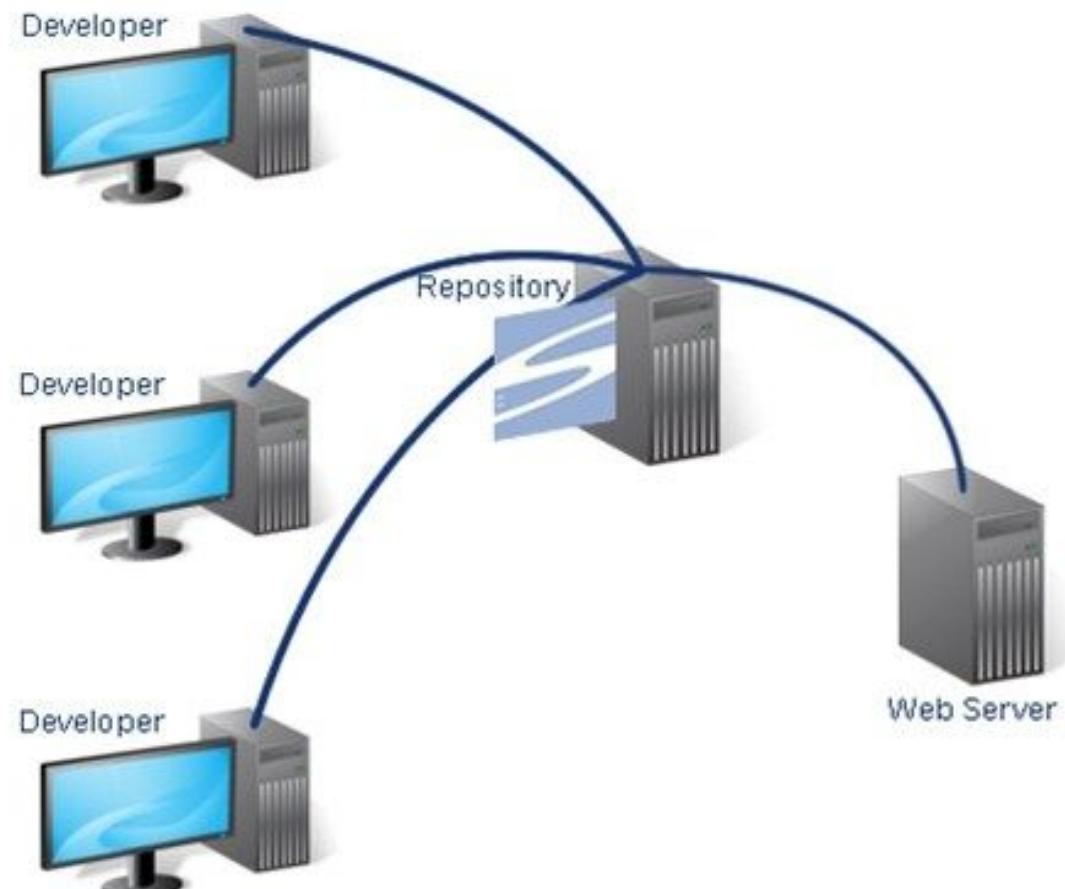
 **Visual Studio**

Team Foundation Server

Systemy kontroli wersji

Client-Server: Centralne

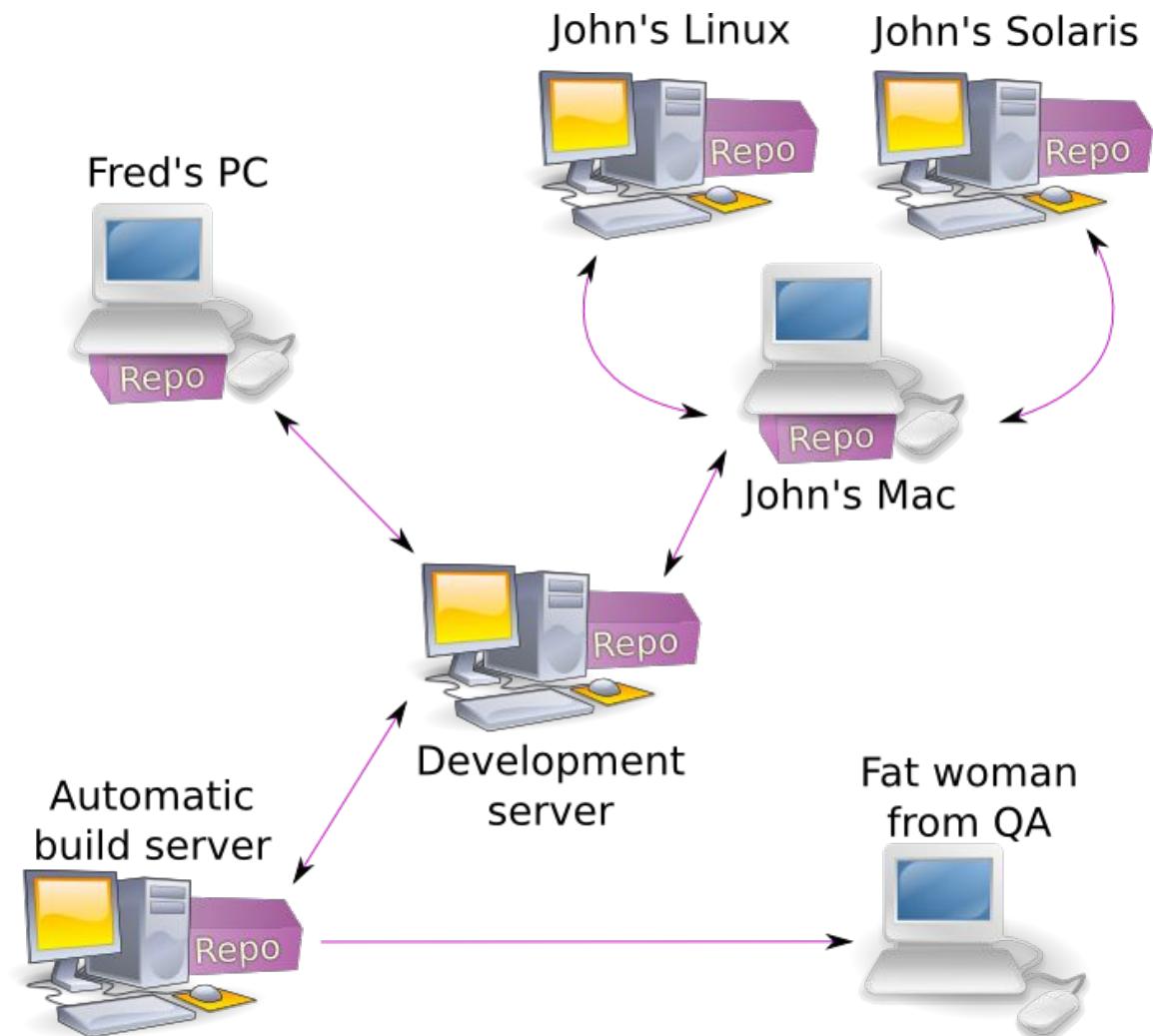
- Mamy tylko jedno repozytorium na serwerze
- Synchronizujemy wersje zawsze na serwerze
- Jeśli coś się ze psuje/ktoś to mamy poważny problem



Systemy kontroli wersji

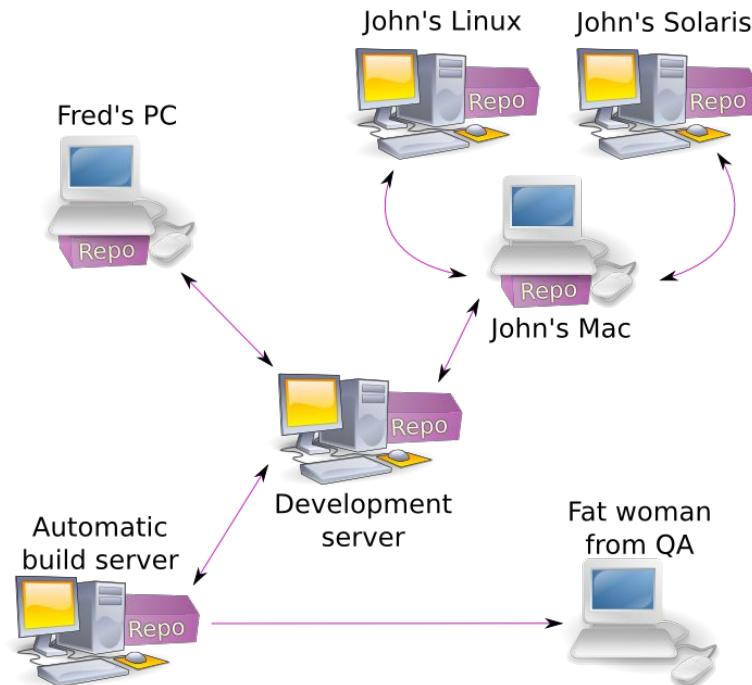
Distributed (rozproszone)

- Każdy ma swoje lokalne repozytorium
- Jeśli coś się zepsuje – mamy tyle repozytoriów ~ ilu developerów



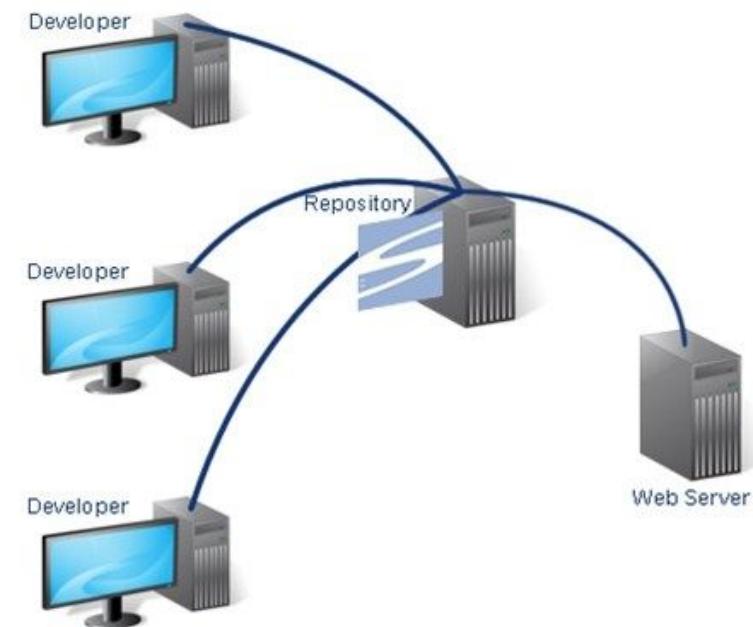
Systemy kontroli wersji

Distributed (rozproszone)



Każdy ma swoje repozytorium.
Serwer też ma swoje repozytorium.

Client-Server



Istnieje tylko jedno centralne repozytorium.
Tylko serwer ma repozytorium.
Reszta ma lokalne kopie plików.

Git – od czego się zaczęło, cechy



GIT – jak to się zaczęło

2005 r.

Jako narzędzie do wersjonowania kodu
przy rozwijaniu Linuxa.



[Linus Torvalds](#)

GIT – jak to się zaczęło

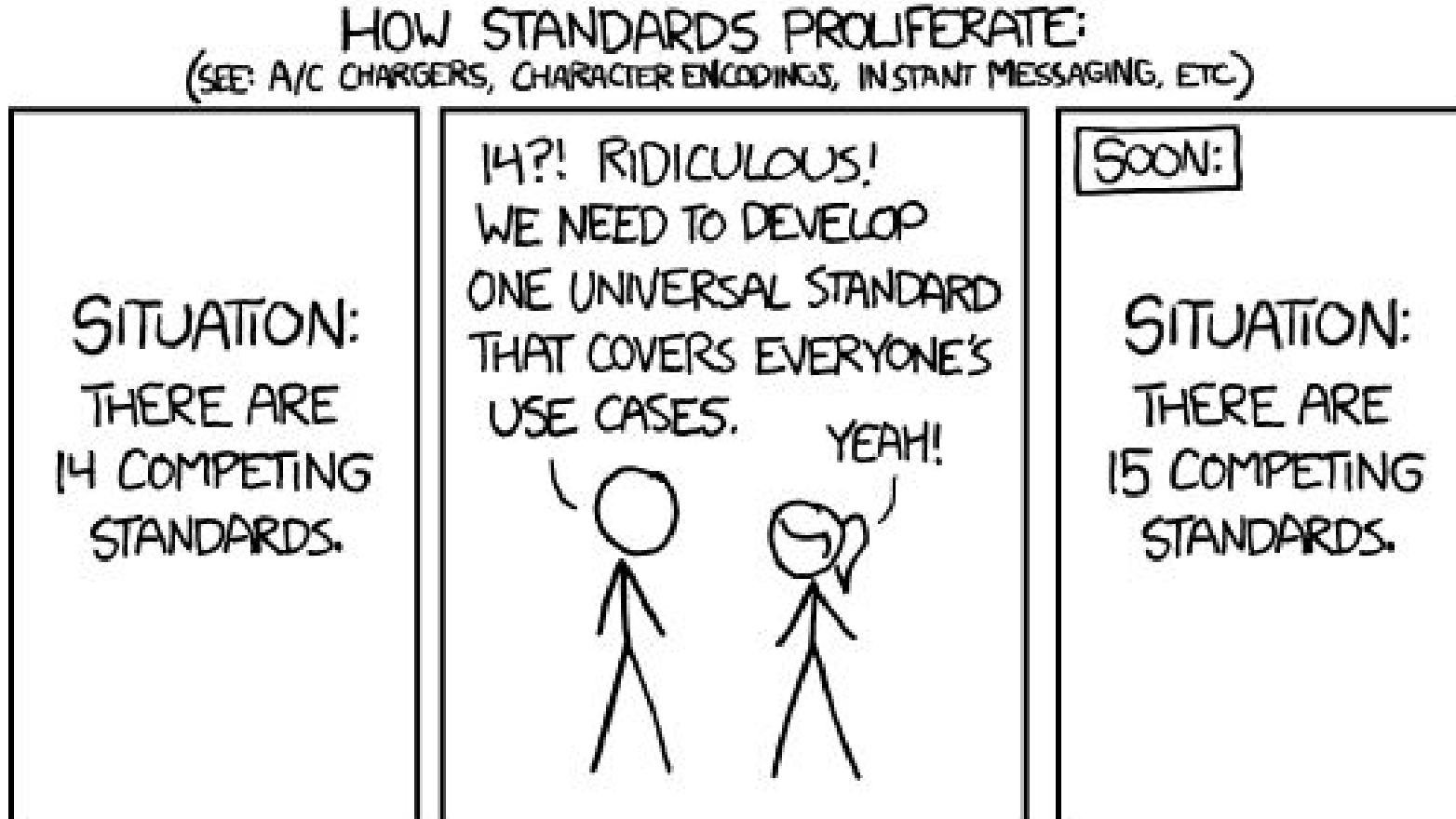
Dlaczego?

- Wydajność i predkość narzędzia
- Śledzenie zawartości a nie plików
- Podejście od strony systemu plików



[Linus Torvalds](#)

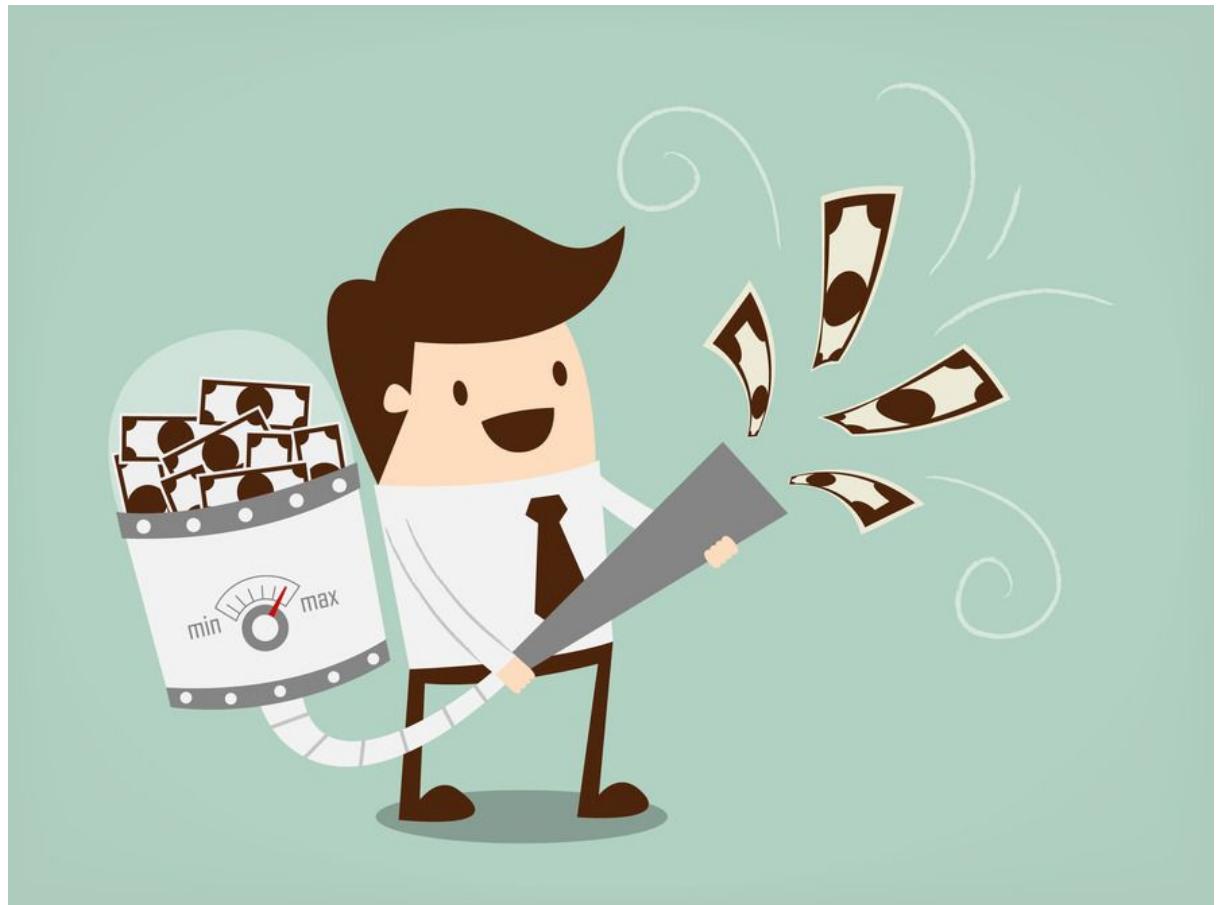
GIT – jak mogło być



[xkcd comics: standards](#)

GIT -zalety

- Rozproszony
- Zoptymalizowany pod kątem wydajności
- Bezpieczny
- Elastyczny – wspiera wiele nieliniowych flow pracy
- Darmowy i rozwijany jako OSS
- Aktualnie to standard



GIT – wady // Co mówi internet

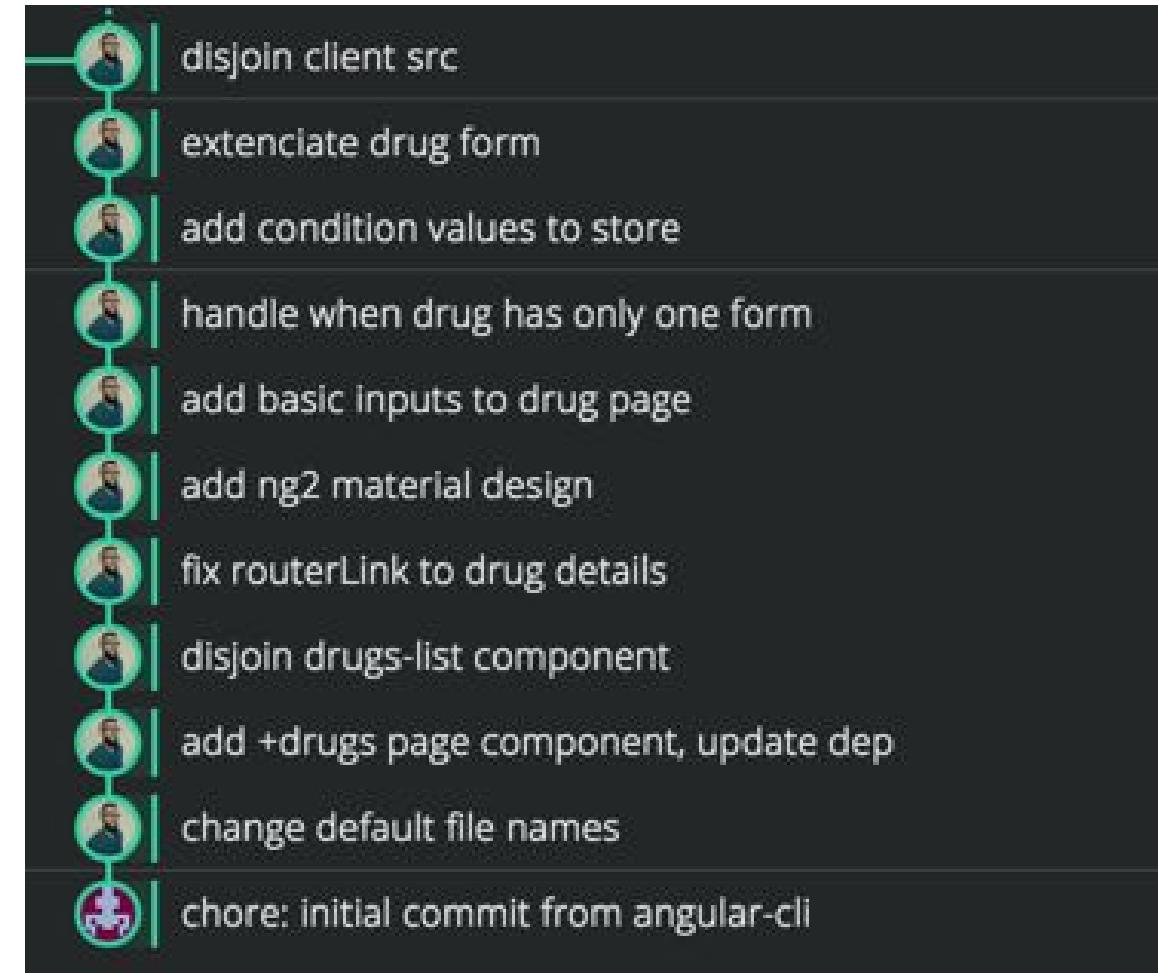
- Skomplikowany model informacji
 - "Crazy command line syntax"
 - Słaba dokumentacja
 - Można zgubić pracę



Rly?

Wersjonowanie i historia zmian

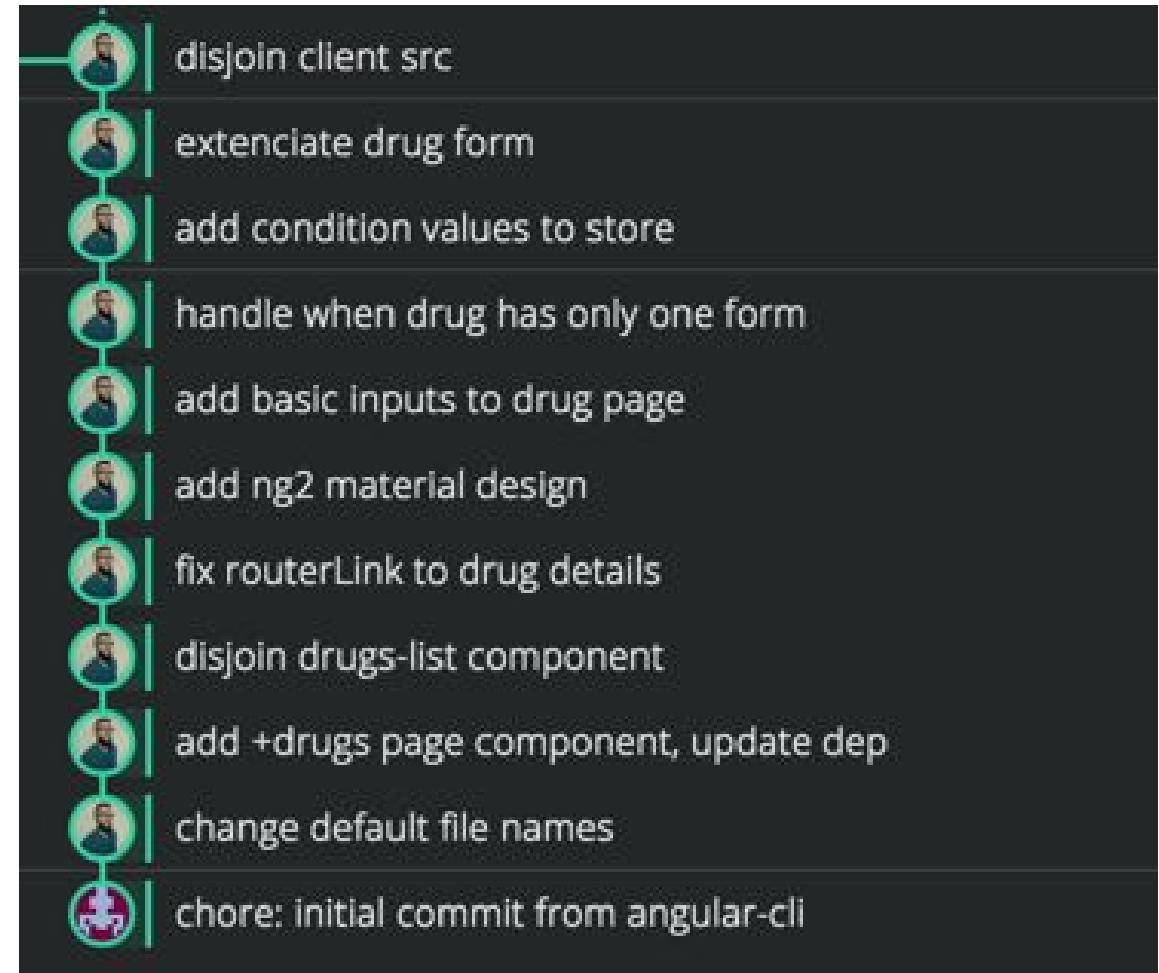
W Git (w SVN też) każdy commit jest różnicą (delta).



Jak wygląda taki zapis historii?

Gdy pracuję sam:

Krok po kroku, kolejne zmiany.



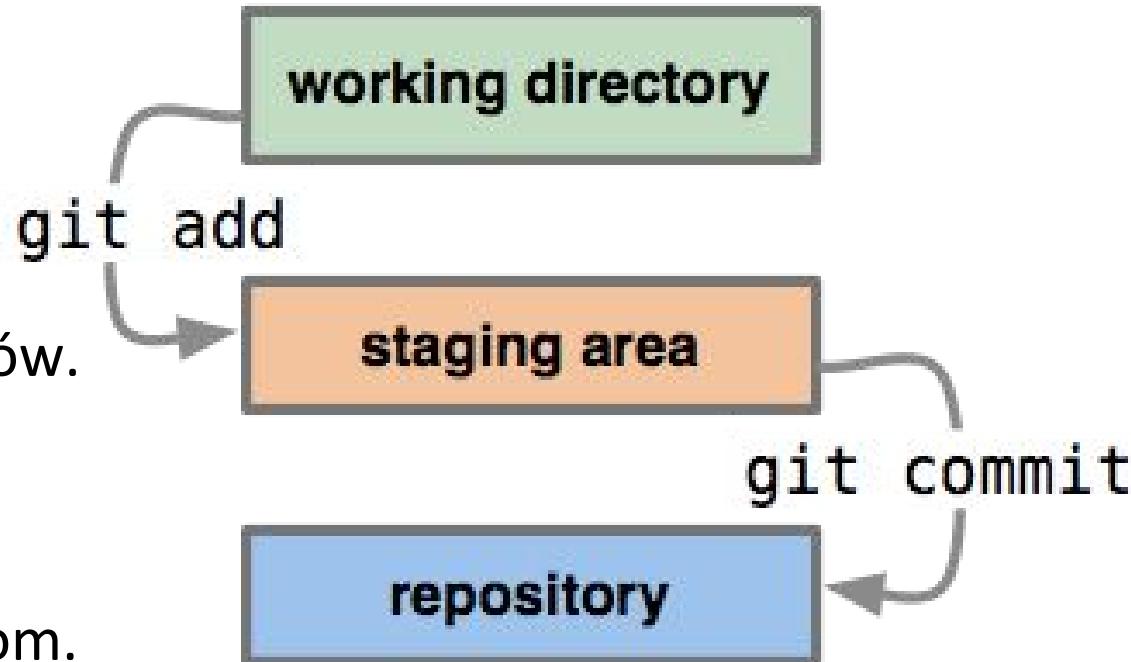
Commit oraz podstawowe operacje



Commit

git commit

- Zapisanie różnicy zawartości plików.
- Commity są stałe i niezmienne.
- Odpowiadają konkretnym zmianom.
- Mają wiadomości które je opisują



Dobry commit

- Ma odpowiednią wiadomość
- Ma zmiany odpowiadające jednej rzeczy
- Z wiadomości da się jednoznacznie określić czego dotyczył
- Ale wiadomość nie jest zbyt długa
- Wiadomość nie jest zbyt szczegółowa

Dobry commit cd.

Odnośnie wiadomości commita:

Use the body to explain what and why vs. how

Dobry commit np.:

Removed formatting from Angular2 Template

Removed placeholder styles that were helping me track my selector during testing. 'template' should be formatted like the rest of the decorator attributes.

 Jonathan Barket commit: 3fa919
authored 12/30/2015 @ 6:18 AM parent: a65d0f

1 modified

Name	Full Path
	grammars/typescript.cson

Dobry commit np.:



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

Zły commit

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

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

[Code] Commitowanie



Operacje które właśnie wykonaliśmy



commit – co go jeszcze charakteryzuje

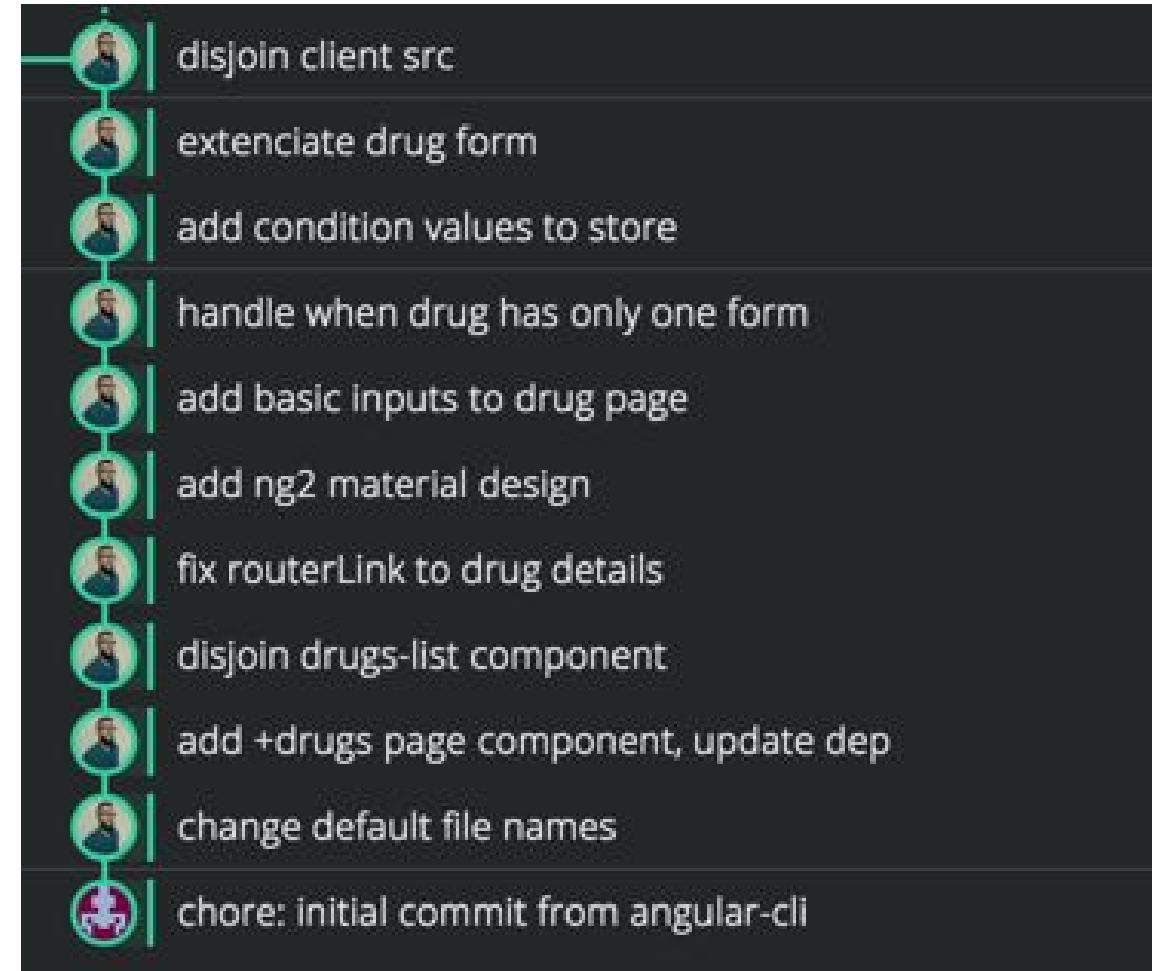
- Każdy commit ma swojego rodzica

- Każdy commit ma swój klucz

Hash SHA1

- Tak naprawdę git trzyma pary:

klucz: wartość



Zrobiliśmy już x commitów, ale co nam to dało?

Możemy teraz podejrzeć krok po kroku jak się zmieniał nasz kod.

Możemy nawet podejrzeć jak się zmieniały poszczególne pliki.

	disjoin client src
	extenciate drug form
	add condition values to store
	handle when drug has only one form
	add basic inputs to drug page
	add ng2 material design
	fix routerLink to drug details
	disjoin drugs-list component
	add +drugs page component, update dep
	change default file names
	chore: initial commit from angular-cli

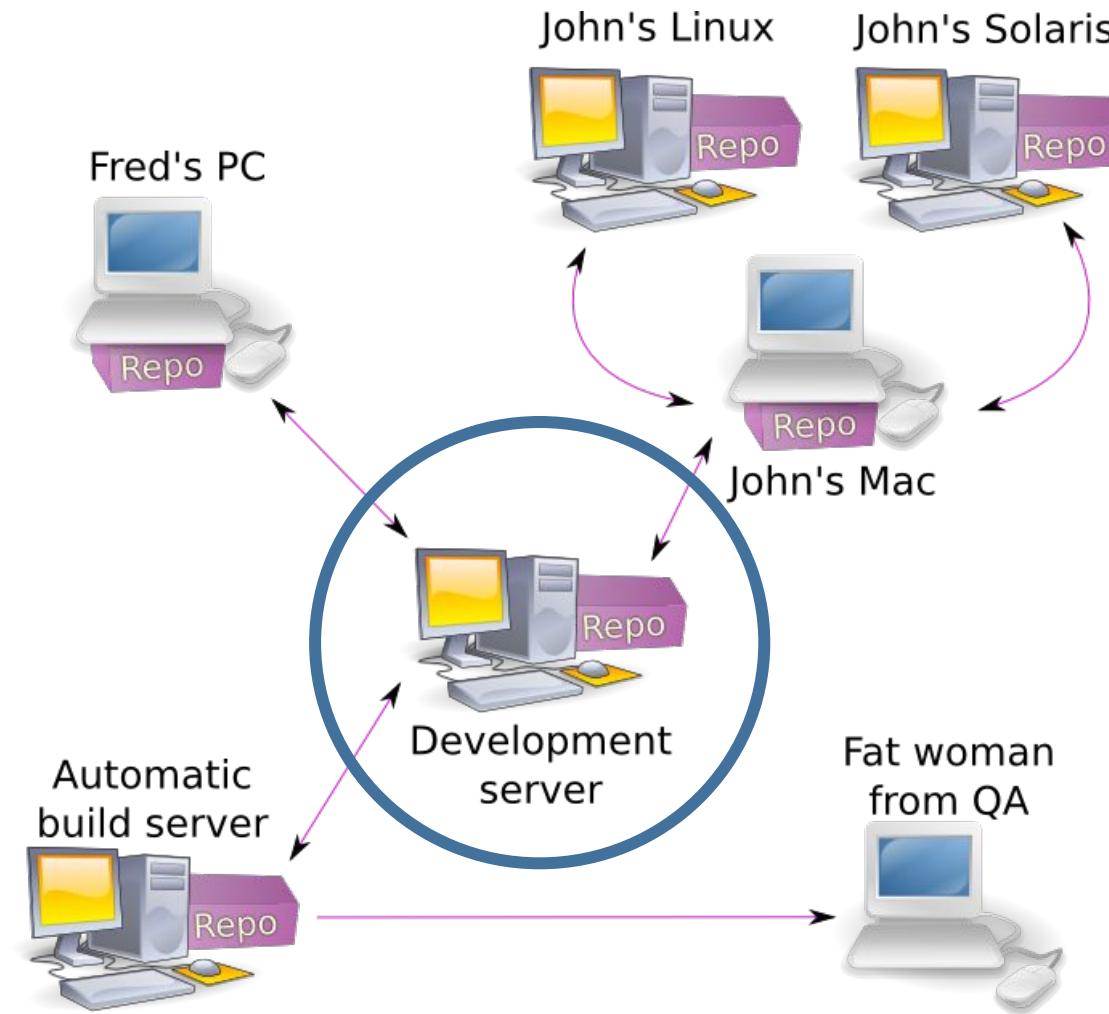
[Code] Zajrzyjmy w historię zmian



Zdalne repozytorium



Zdalne repozytorium



Zdalne repozytorium

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



Popularne usługi które przechowują repozytoria



GitHub to usługa



git



Jak pracować z zdalnym repozytorium?

1. Nasze lokalne repozytorium musi znać adres zdalnego repozytorium
2. Możemy mieć nawet wiele repozytoriów podpiętych



origin

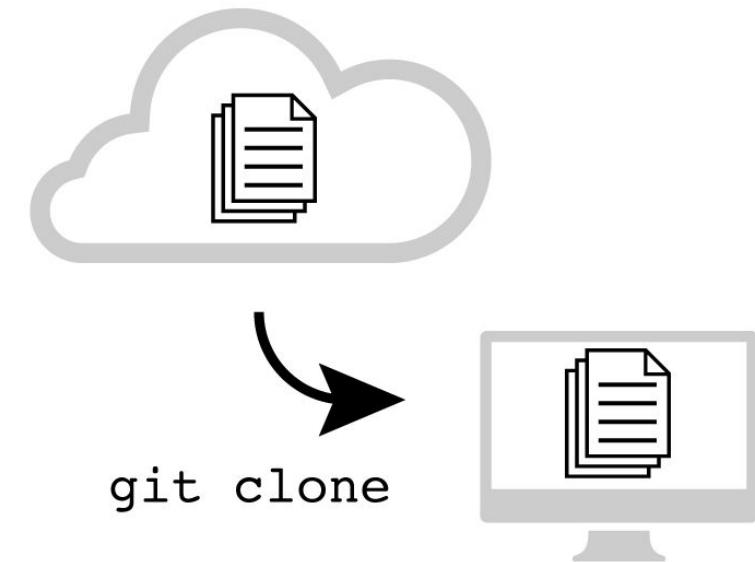
Domyślna nazwa zdalnego
repozytorium.

Takie centralne repozytorium.



git clone

- Kopiuje istniejące repozytorium, np.
z danego adresu <https://foo.git>
- Ustawia to repozytorium jako origin

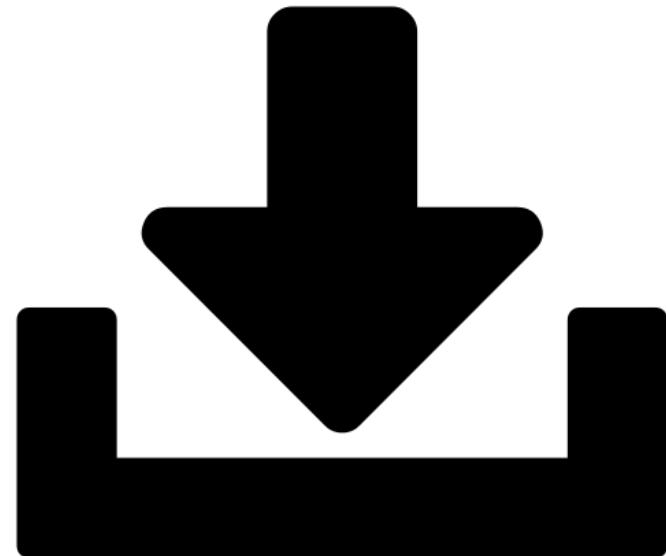


[Code] git clone – sklonujcie moje repozytorium



git fetch

Pobiera commity z zdalnego
repozytorium do naszego lokalnego,
ale jako "remote".



Nie włącza tych zmian. Do naszego
lokalnego repozytorium.

git pull

Włącza commity z zdalnego
repozytorium do naszego lokalnego
repo.

Od razu robi fetch pod spodem.



[Code] fetch& pull



git pull pod spodem

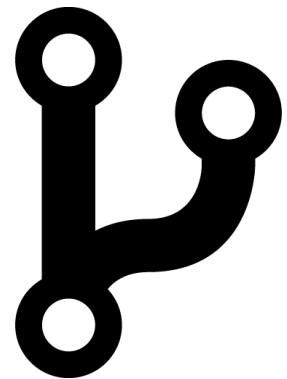
git fetch <remote>

git merge origin/<current-branch>

[Code] Wrzuć swoje repozytorium na GitHub



Branchowanie

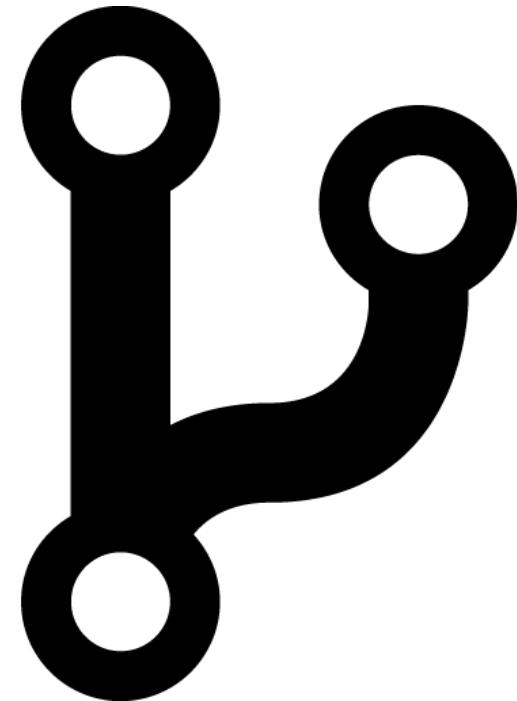


Branch (gałąź)

`git branch`

Branch to **wskaźnik** na commit.

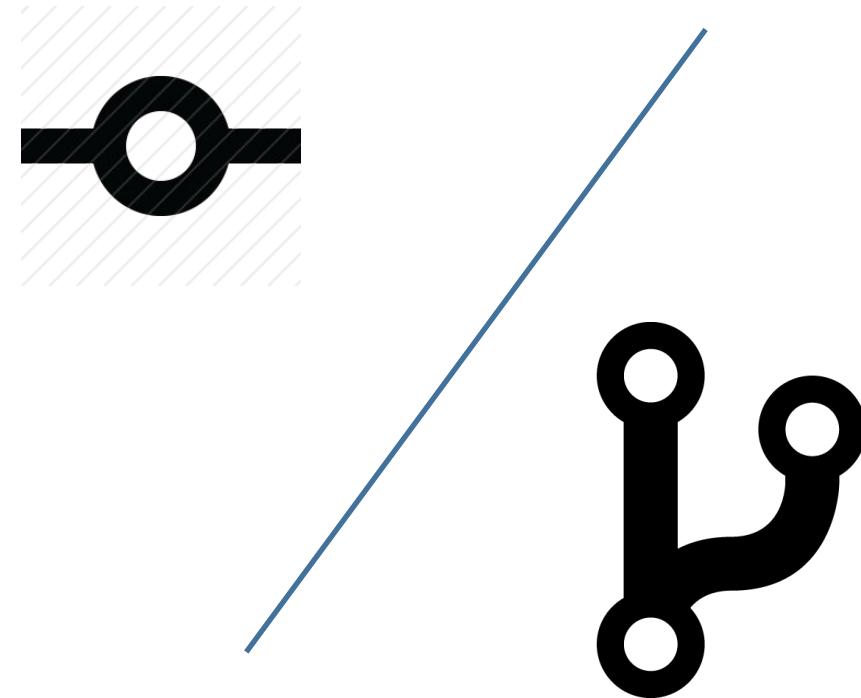
Można go zawsze
dodać/usunąć/zmienić.



Community a branche

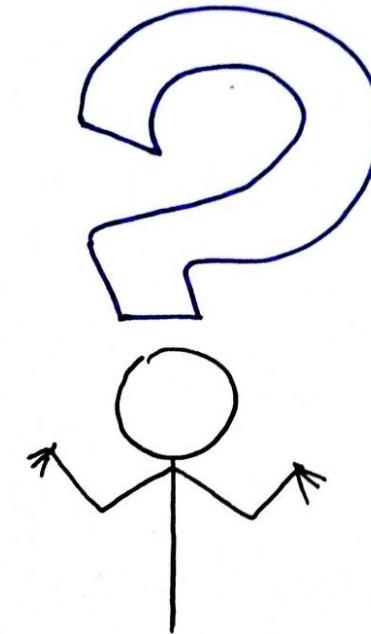
Community są przechowywane
(persistent)

Branche są płynne, to wskaźniki na
community.

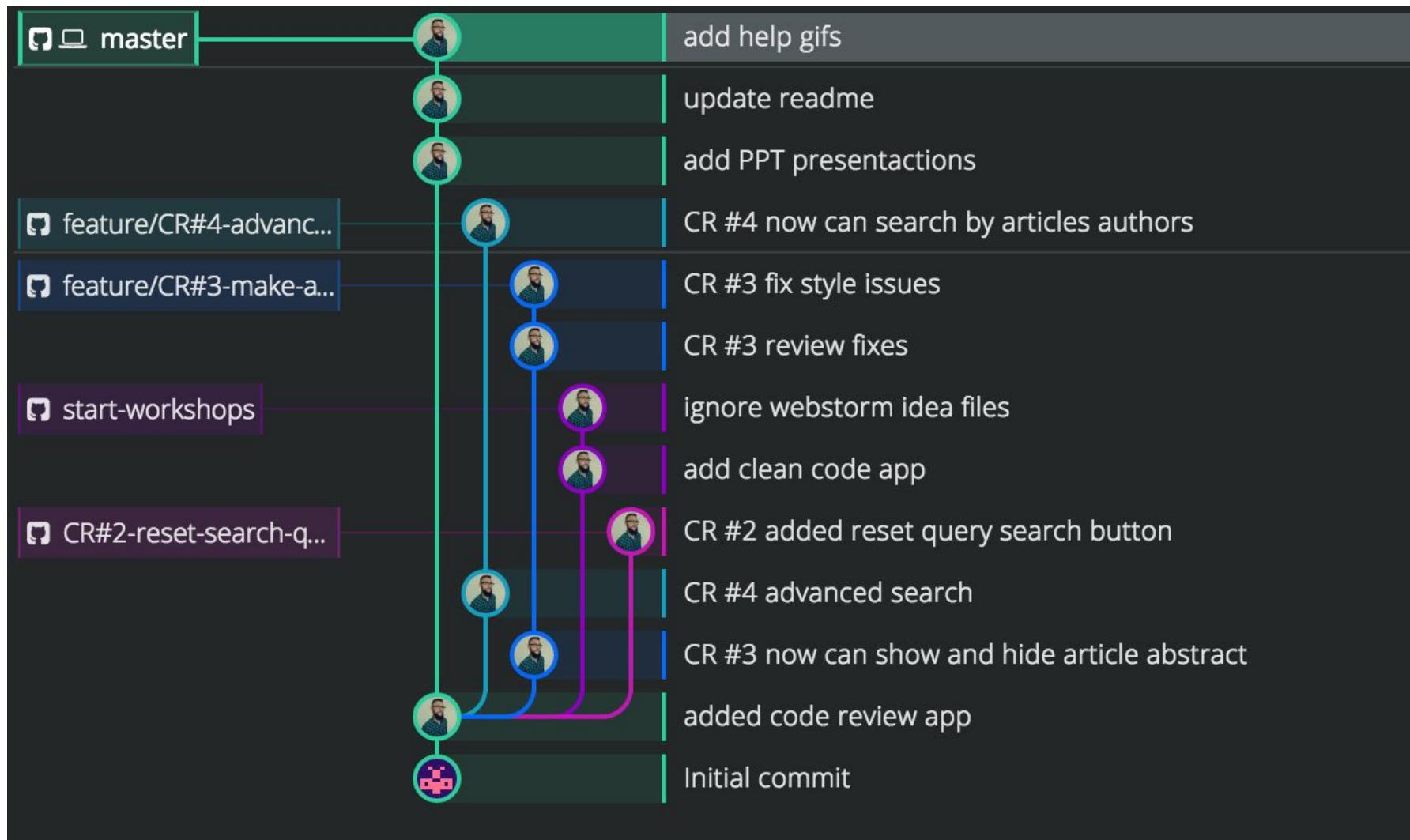


Po co nam branche?

- Problem pracy równoległej
- Problem nadpisywania sobie zmian
- Problem nie ukończonych rzeczy

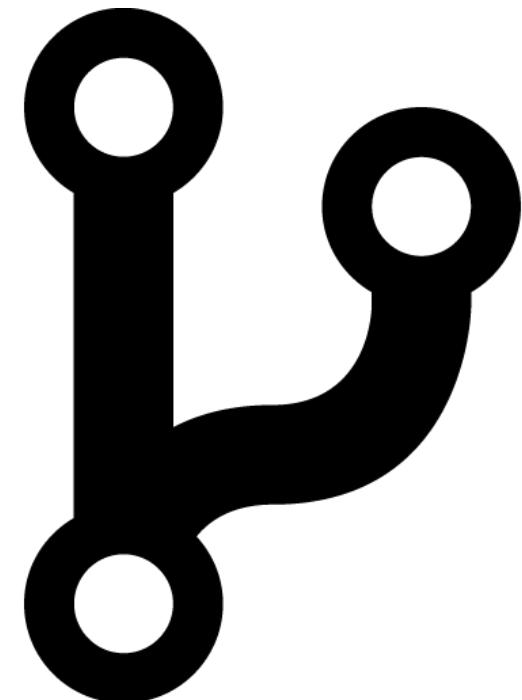


Otwarta praca nad wieloma zadaniami naraz



Co oznacza, że jestem na branchu?

- Aktualnie stan twojego kodu "odpowiada" commitowi na którym jest branch
- Możesz się przełączać pomiędzy branchami



[Code] Tworzymy branche

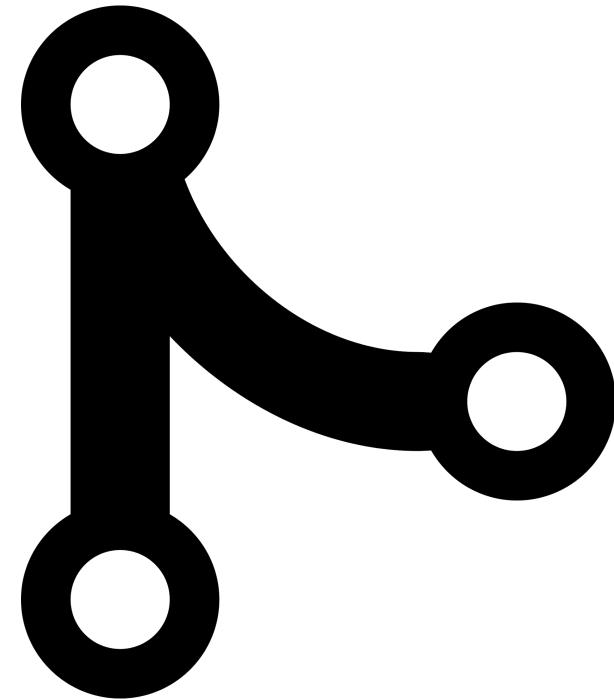


[Code] Zajrzyjmy ponownie w historię zmian

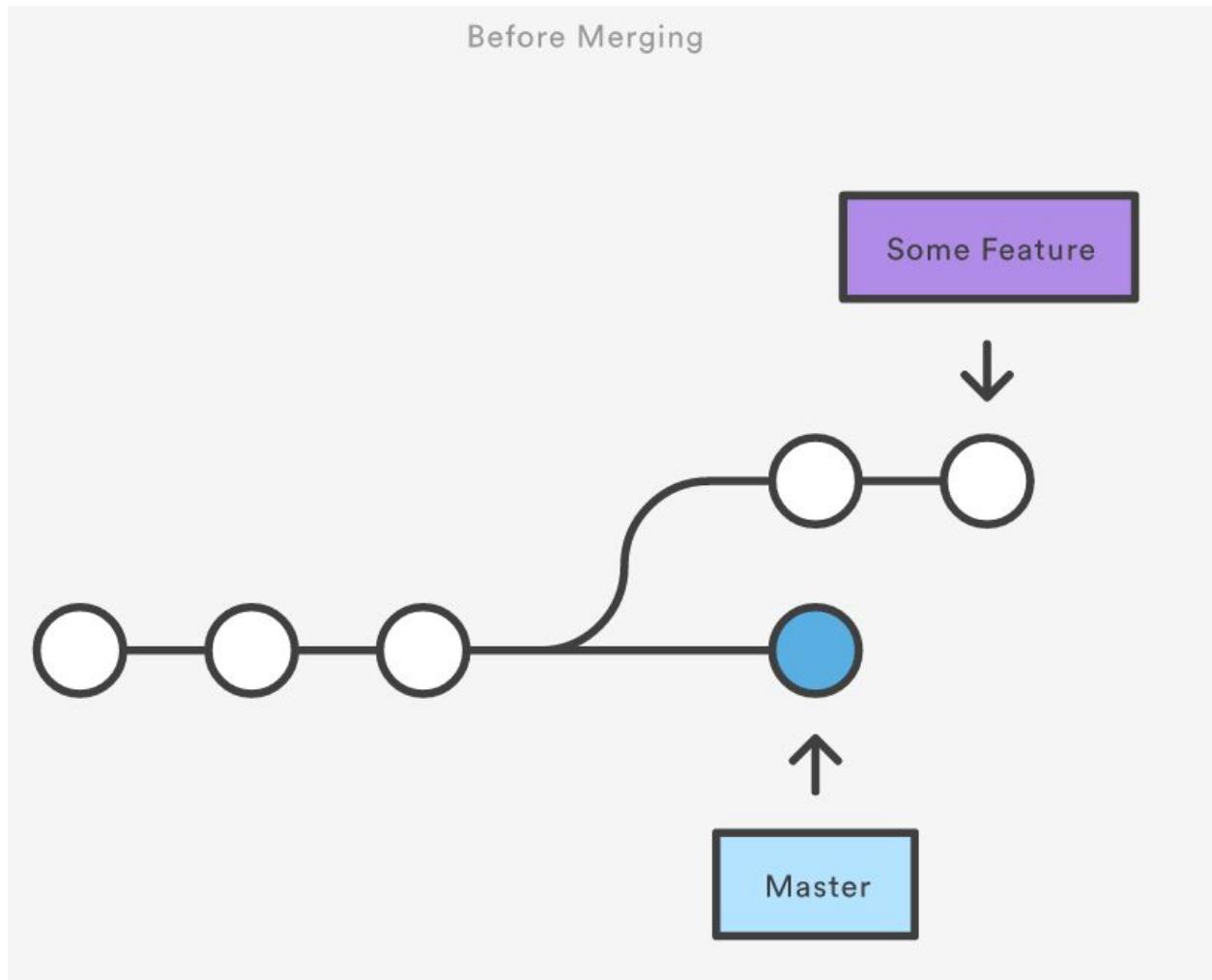


Merge

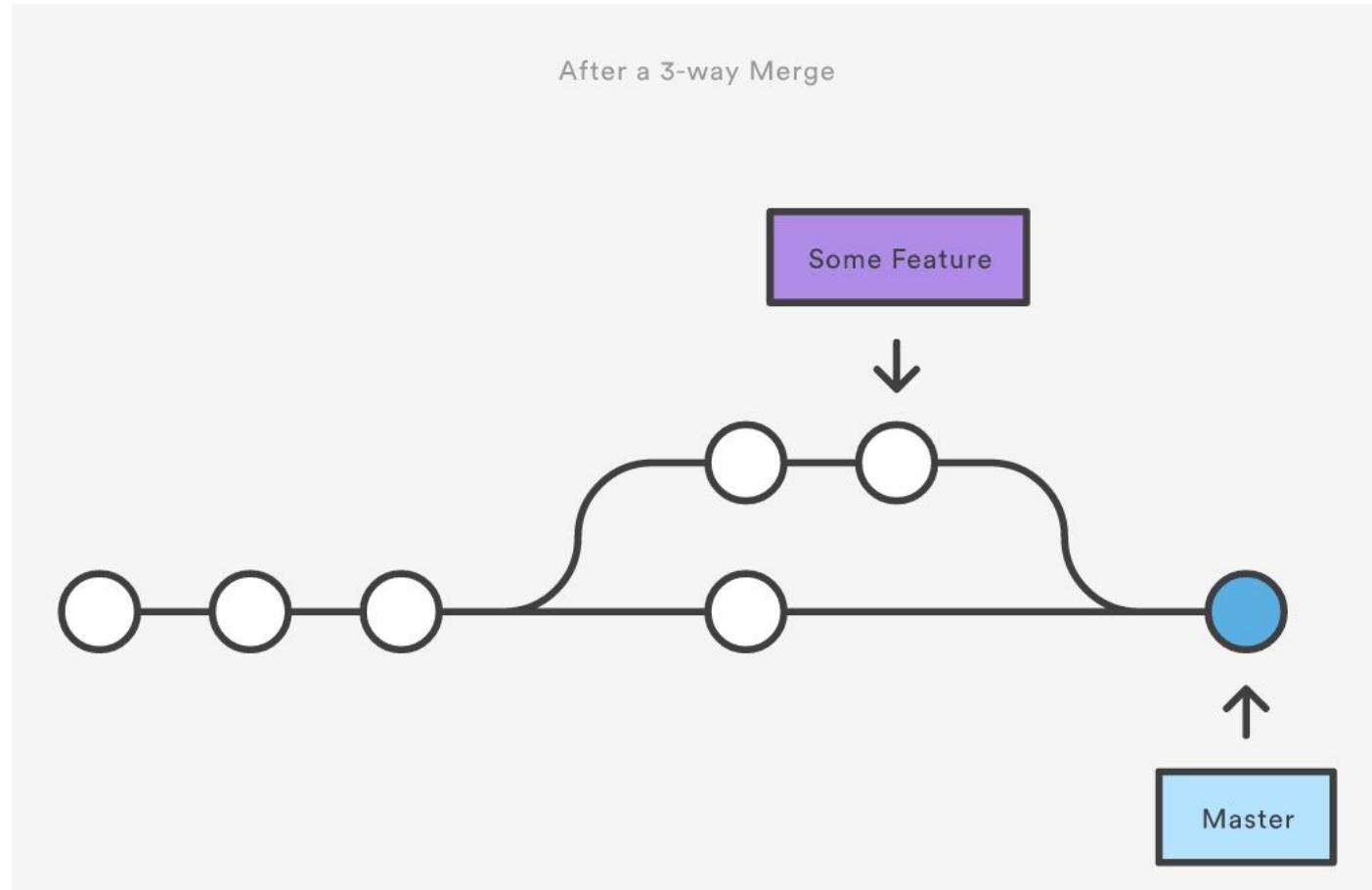
- Połączenie/zmieszanie ze sobą branchy
- Łączenie naszego kodu w całość
- Ile rodziców ma merge?



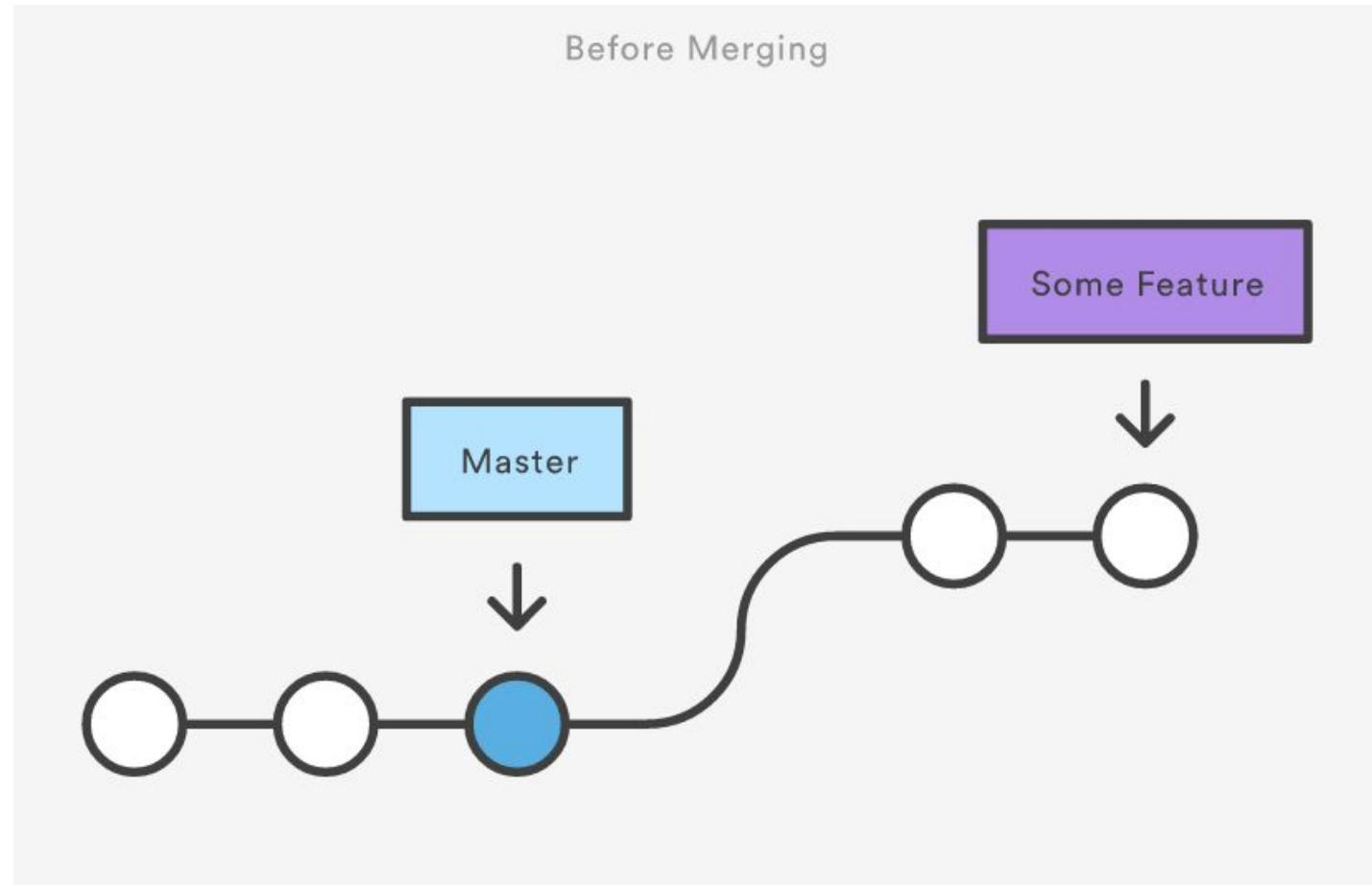
Merge – 3 way merge 1/2



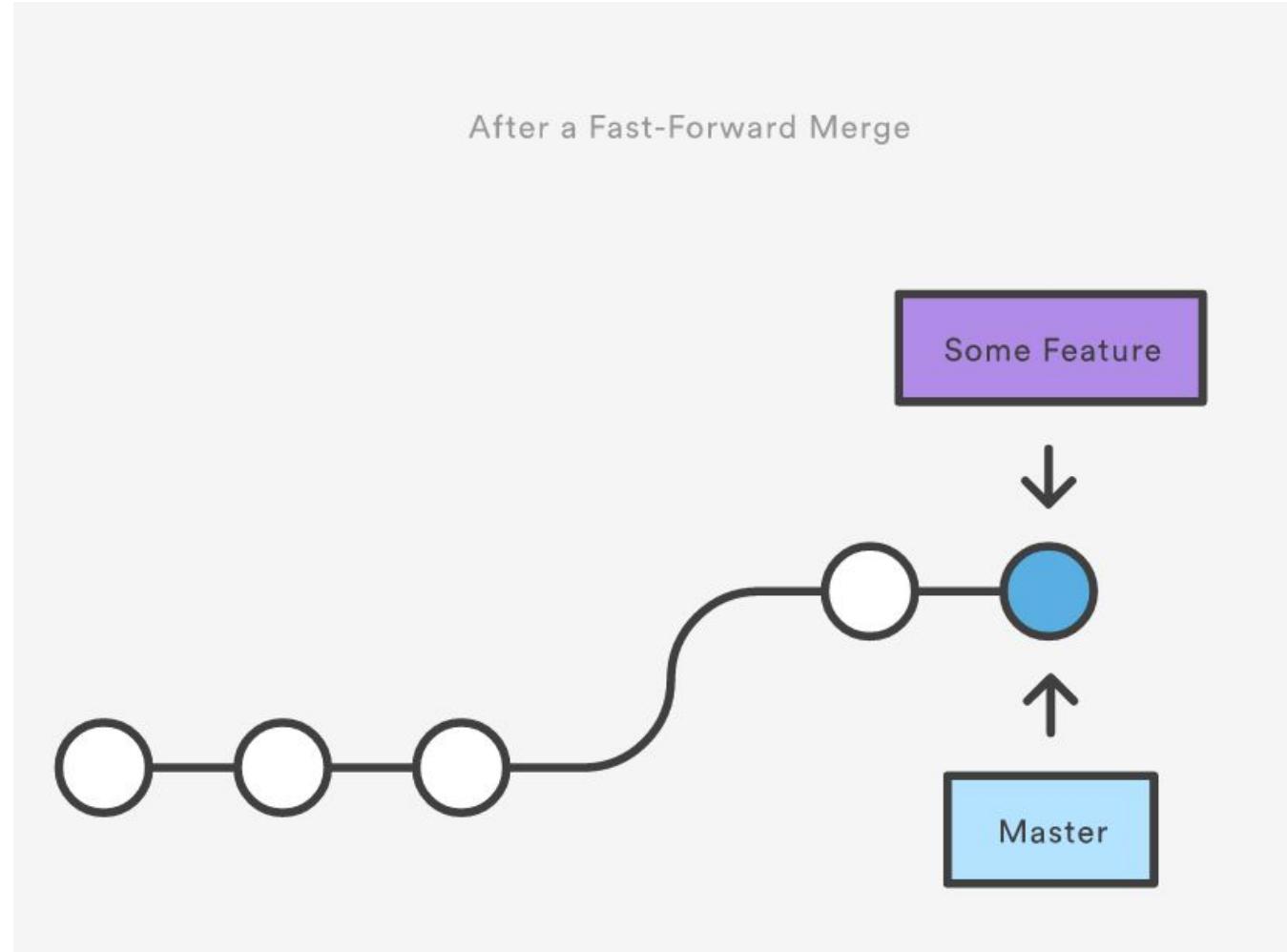
Merge – 3 way merge 2/2



Merge – Fast forward 1/2

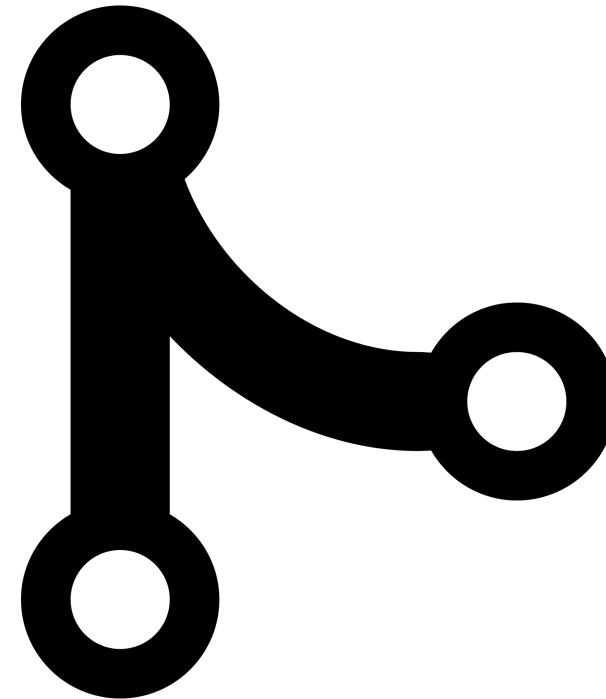


Merge – Fast forward 2/2



Merge – czy zawsze się uda?

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



Merge – konflikty

Są naturalną rzeczą przy pracy git-em.

Nie należy się ich bać :)

Ani lekcewarzyć.

Czasem wymagają konsultacji z

autorem innych zmian.



[Code] Mergujemy nasze features-y.
Rozwiążujemy konflikty.



git pull powraca. Dlaczego?

Pod spodem jest merge.

```
git fetch <remote>
```

```
git merge origin/<current-branch>
```

git pull ... conflict

Pod spodem jest poprostu merge.

Jeśli mergujemy kod -

Może pojawić się konflikt.



git merge – dobre praktyki

- Zrób pull brancha do którego mergujesz przed mergem
- Merguj się często.

Częste merge = mniej konfliktów & prostsze konflikty

Częste merge

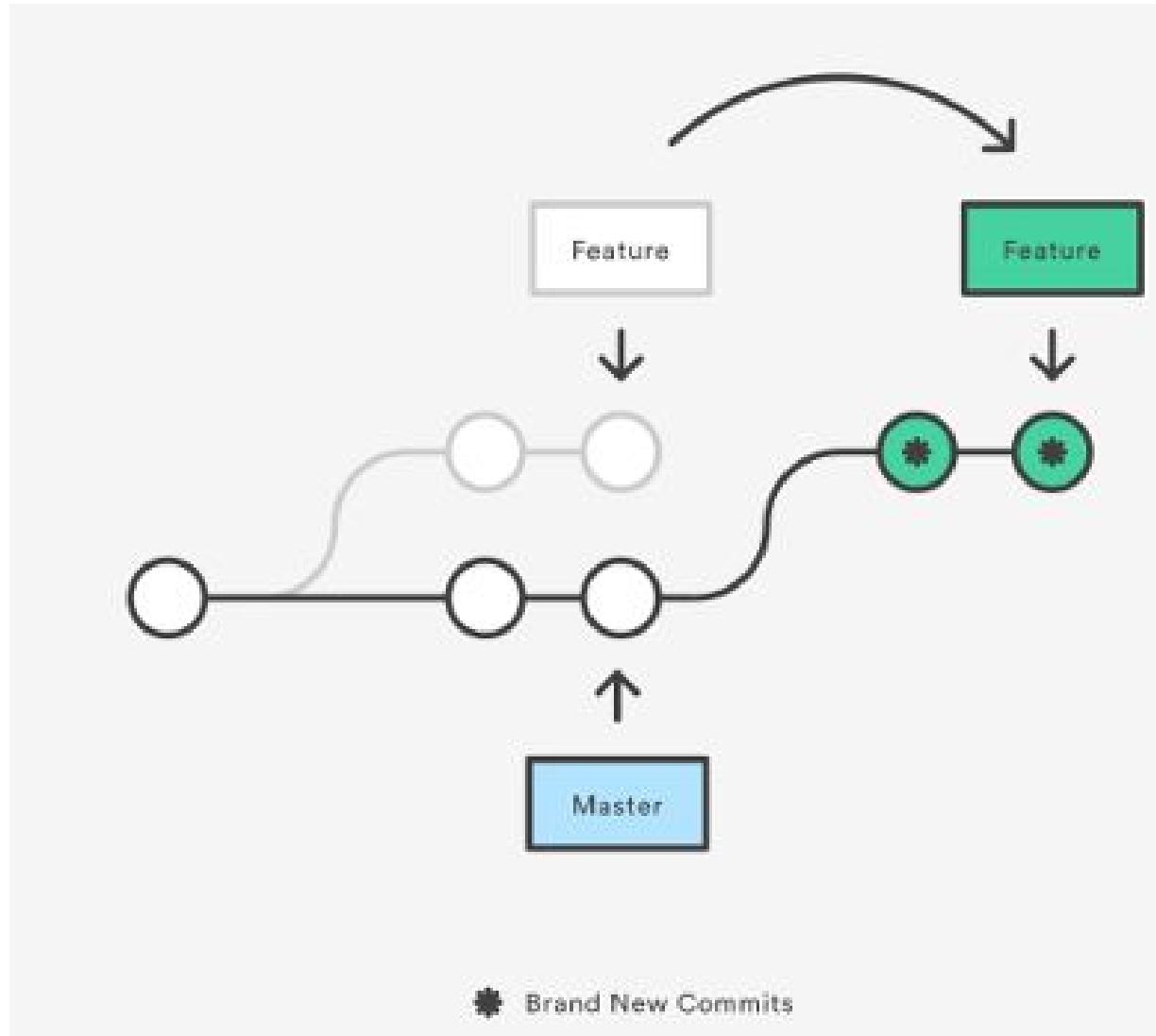
=

**mniej konfliktów & prostsze
konflikty**

git conflicts ... everywhere



git rebase – kolega git merge



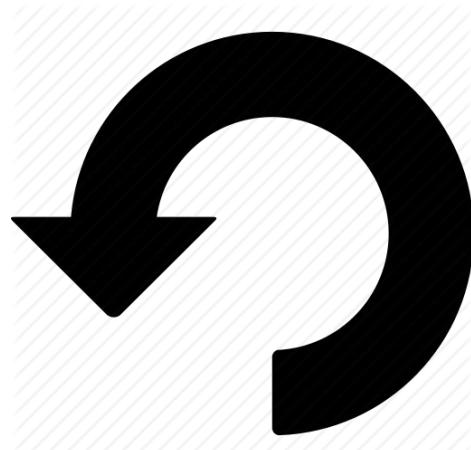
git rebase – kolega git merge

- Zmieniamy bazę naszego brancha
- Tak naprawdę przekładamy po 1 commicie w góre
- Może być tak, że będziemy mieć konflikty co commit

[Code] git rebase

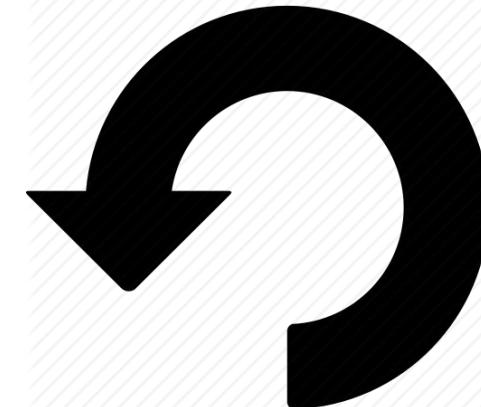


Wycowanie zmian



Gdy się pomylimy

Mechanizmy które umożliwiają zmianę / wycofanie zmian.



- git revert
- git commit --amend
- git rebase – interactive
- git reset

revert - Pomyliłem się, co zrobić?

git revert

Aby bezpiecznie wycować zmiany
robimy ... commit, który je wycofuje.



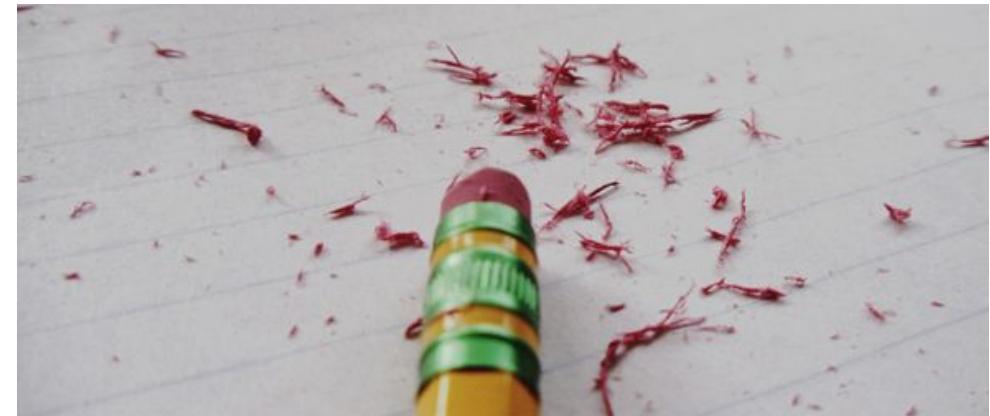
[Code] Git revert



Amend – Znowu się pomyliłem

git commit --amend

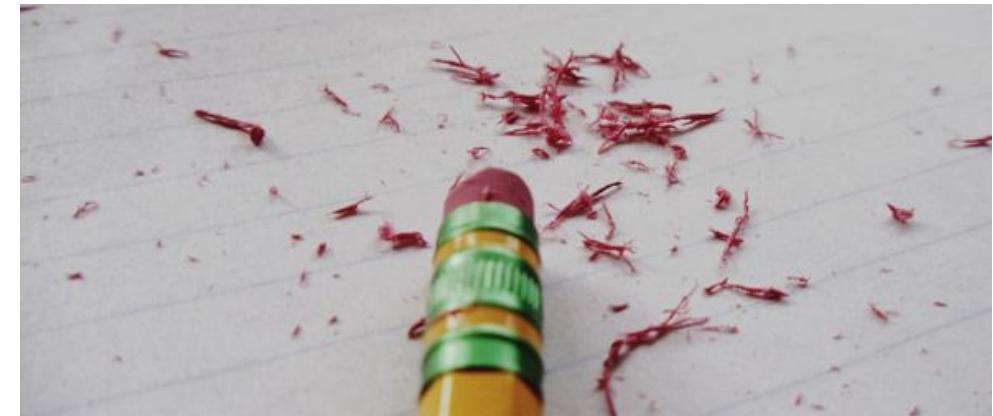
Gdy chcemy coś dołożyć do ostatniego commita albo zmienić mu nazwę.



Amend, ale jak to. Przecież commity sa niezmienne

git commit --amend

Commity są niezmienne więc tak naprawdę tworzymy nowy commit.



[Code] Git commit --amend

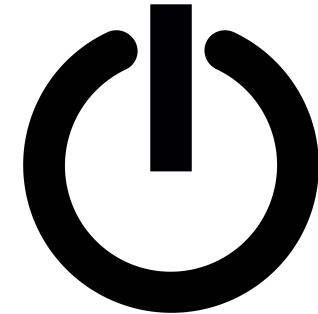


reset – przywracanie stanu

git reset

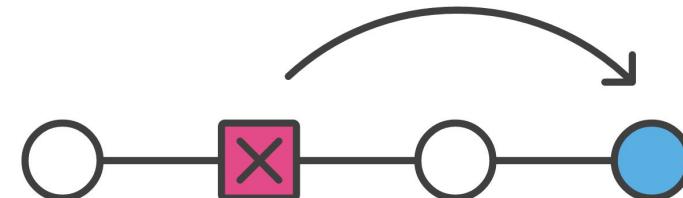
Rozróżnię jej używanie w 2ch kontekstach:

- Wycowanie lokalnych zmian (z Working Directory)
- Ustawianie brancha (wskaźnika) na konkretny commit

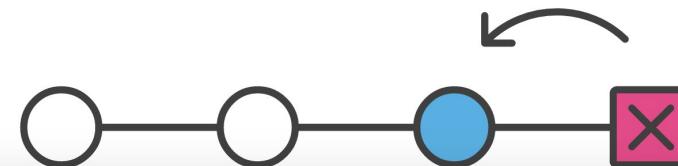


revert vs reset

Reverting



Resetting



<https://www.atlassian.com/git/tutorials/undoing-changes/git-reset>

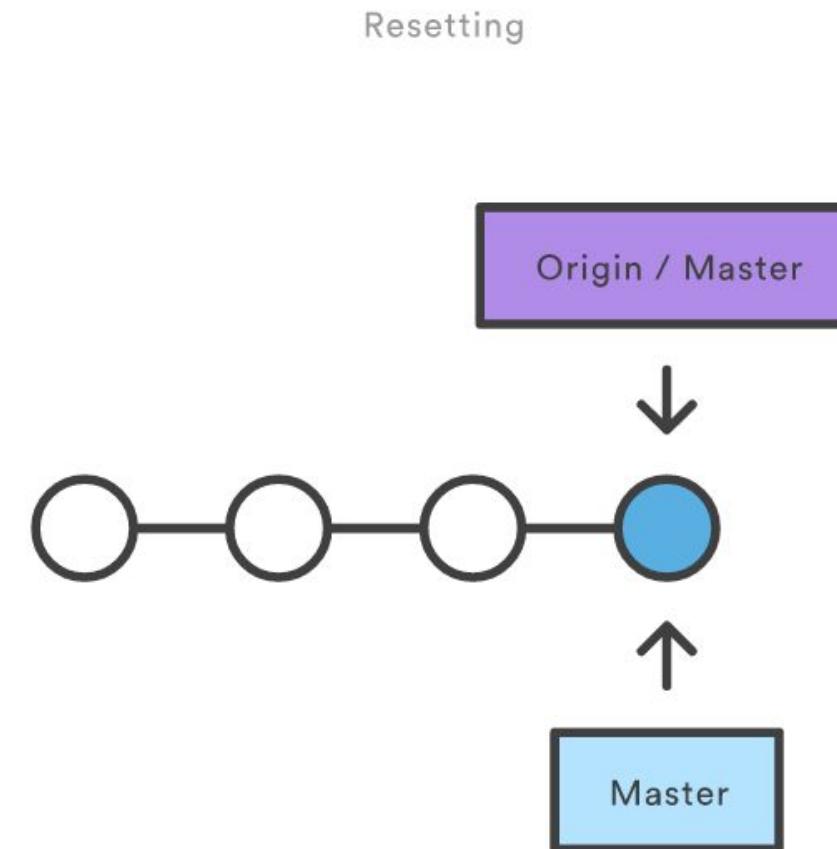
[Code] Git reset – lokalne zmiany



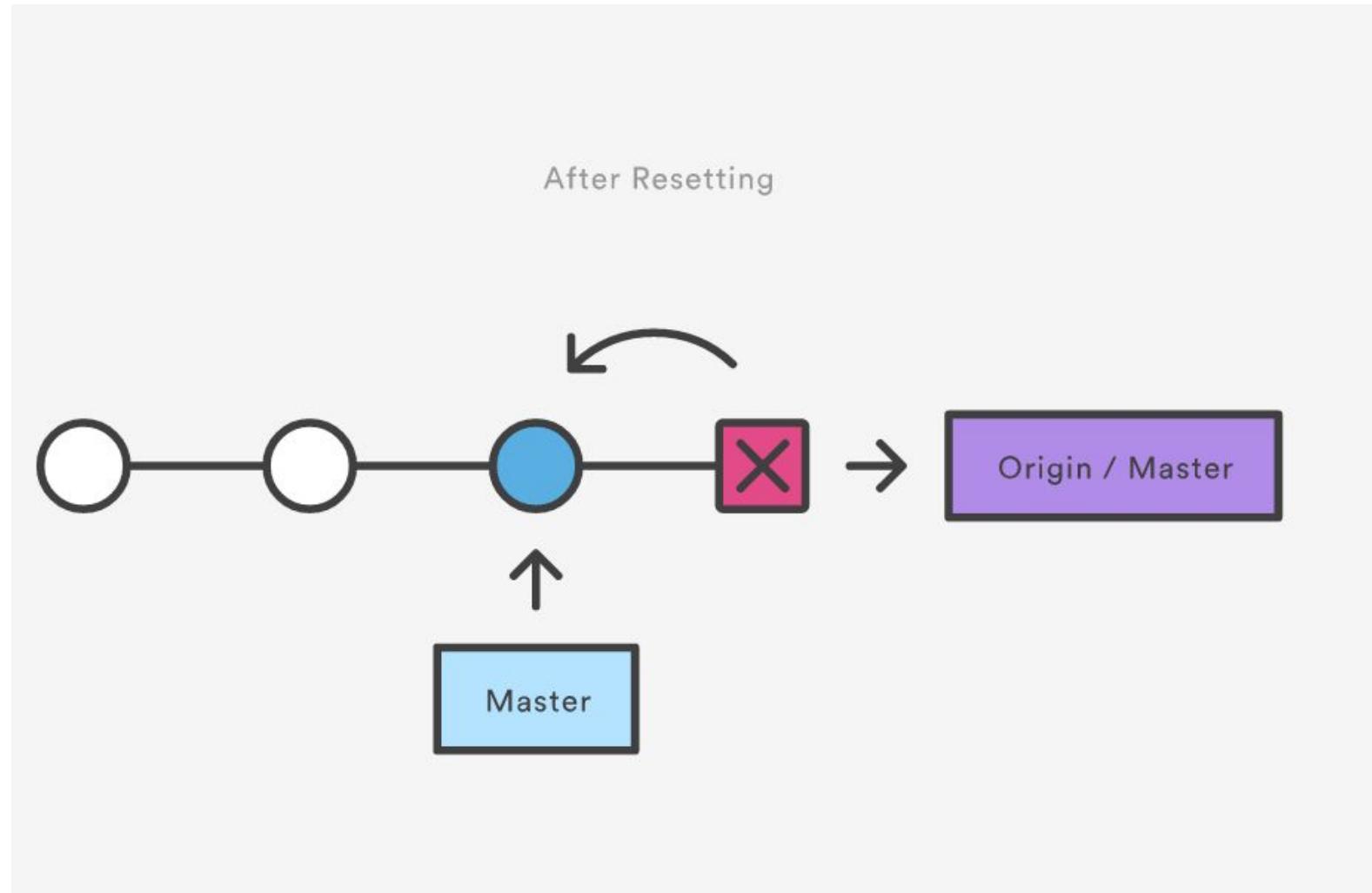
[Code] Git reset – odczepiamy commity od brancha



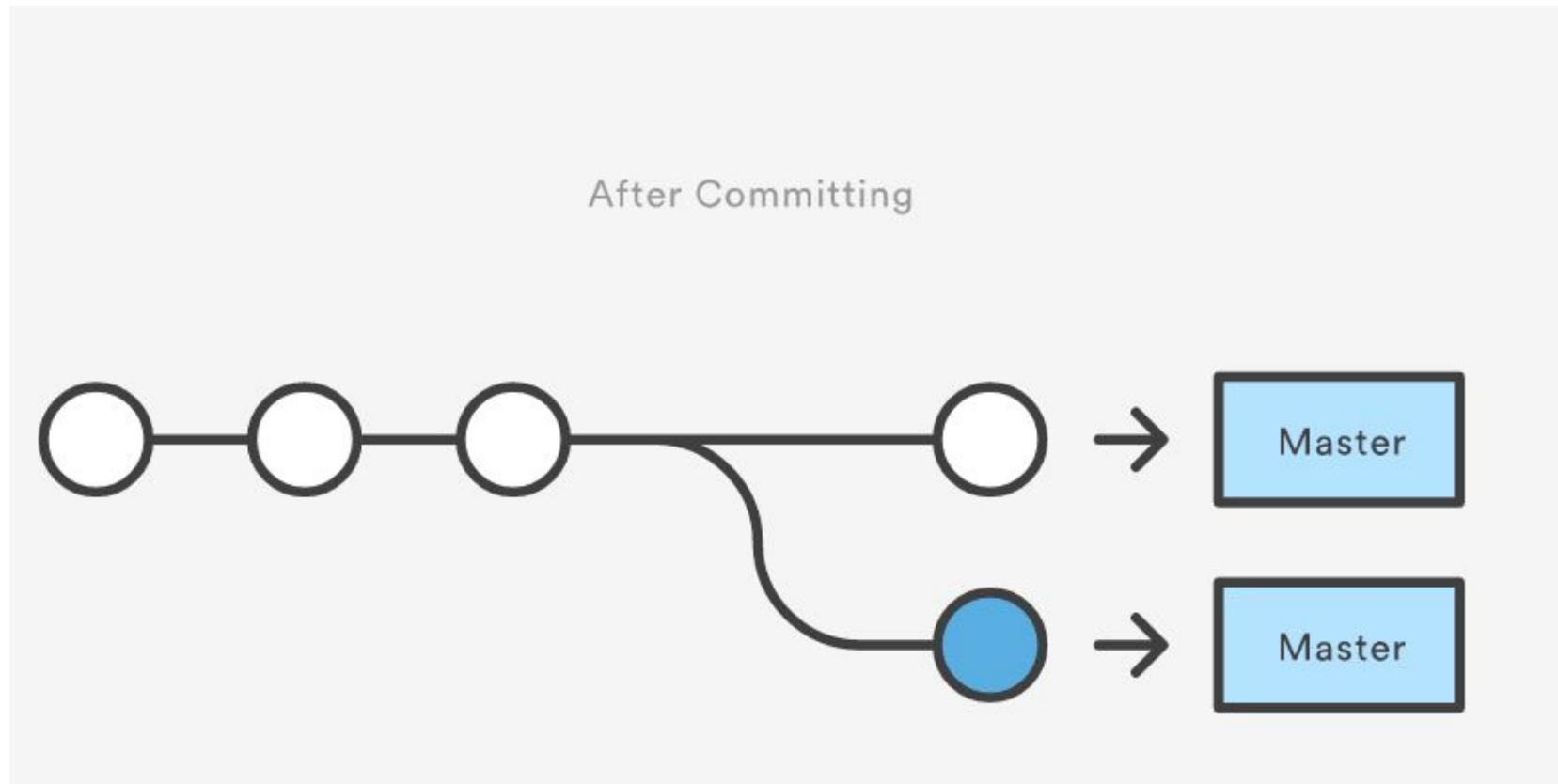
Git reset – jak wygląda nasza sytuacja? 1/3



Git reset – jak wygląda nasza sytuacja? 2/3



Git reset – jak wygląda nasza sytuacja? 3/3

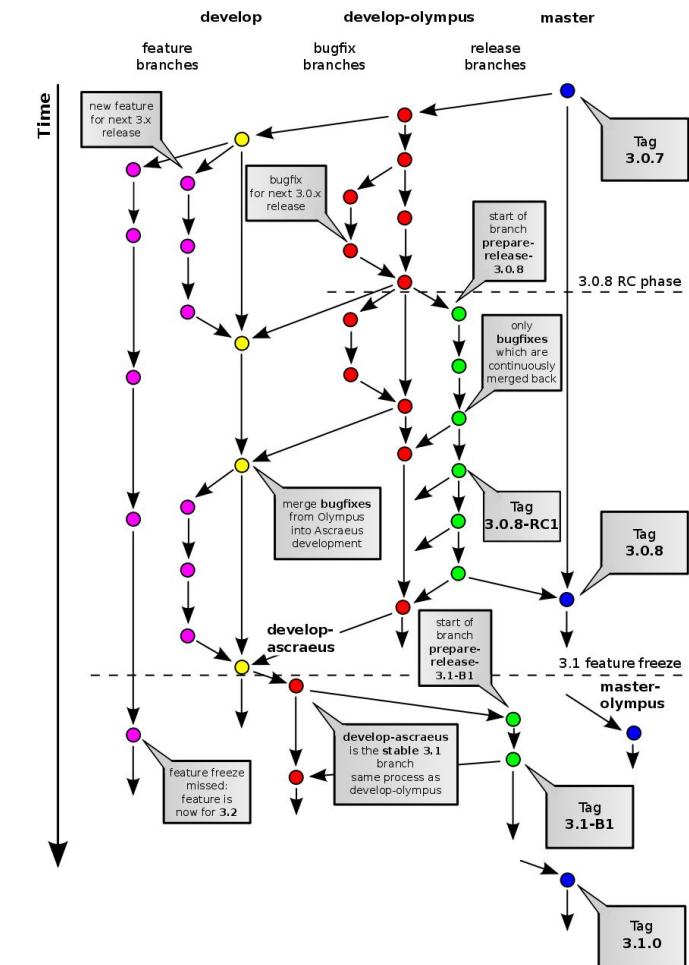


git flow



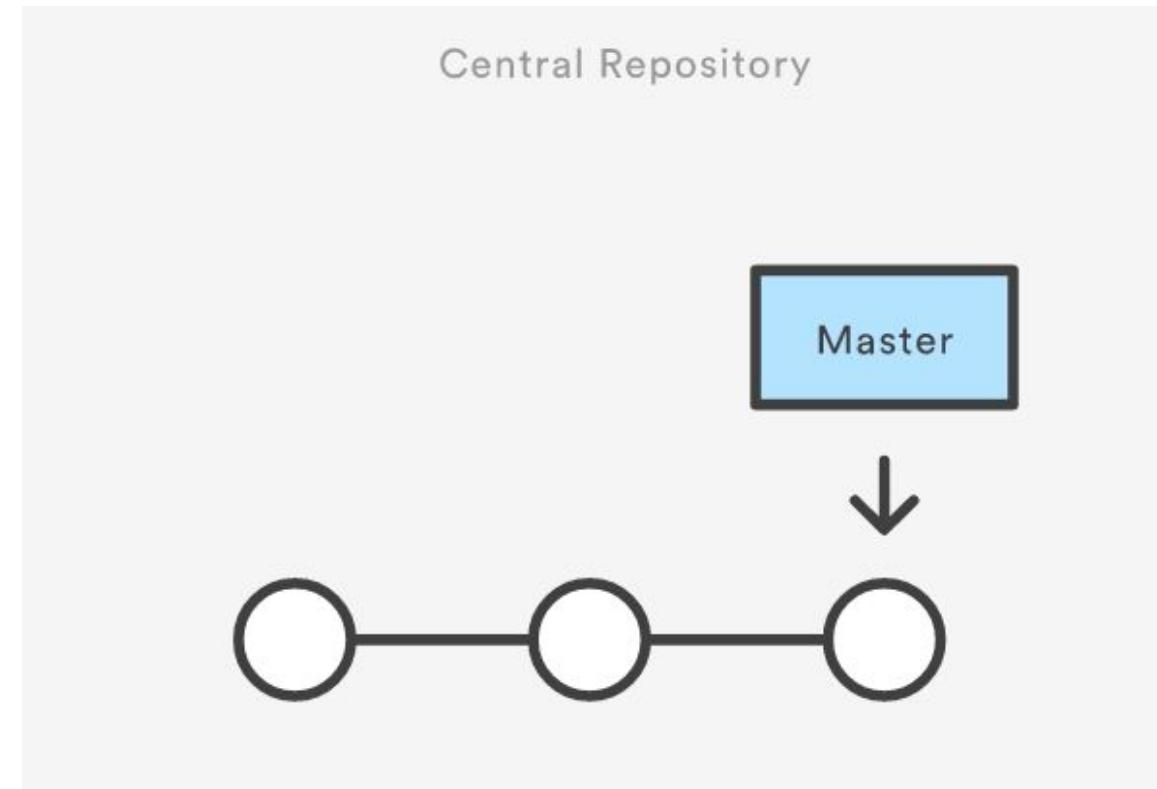
Sposoby pracy z git-em

- git sam w sobie nie narzuca systemu pracy
- istnieje parę popularnych workflow



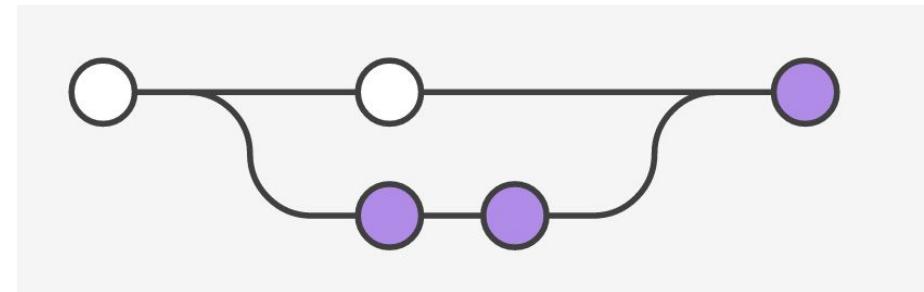
Centralized workflow

- Zbliżony do pracy z SVN
- Praca tylko na jednym branchu
- Dużo konfliktów
- Brak rozróżnienia na kod rozwojowy i kod produkcyjny
- Brak stabilności kodu (efekt)
- Ciężkie w koordynacji przy wielu osobach



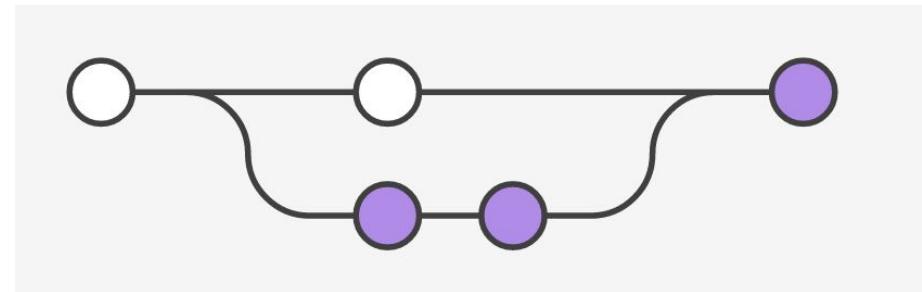
Feature branch workflow

- Naturalny podział zadań
- Każdy feature / bug ma swojego brancha
- Rozdzielenie kodu rozwojowego dla feature-ów
- Wiele osób = Wiele branchów
- Możliwość weryfikacji kodu przed wejściem do master-a
- Dalej brak rozróżnienia na kod rozwojowy i kod produkcyjny



Gitflow workflow

- Rozwinięcie Feature branch workflow
- Oddzielne branche na release-y
- Oddzielnny branch tylko na kod produkcyjny
- Rozróżnienie na kod rozwojowy i kod produkcyjny
- Pełna informacja i historia jaki kod jest w jakiej fazie



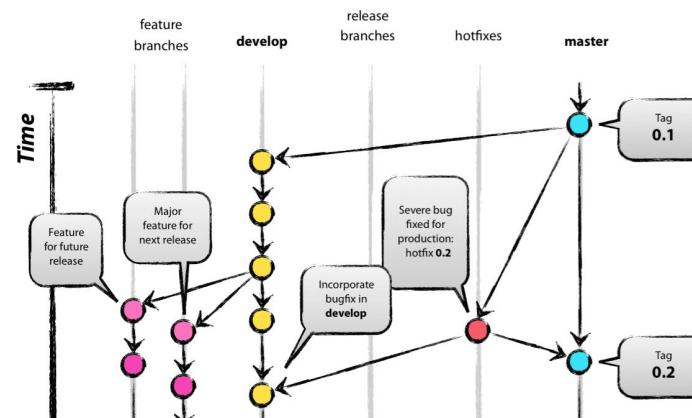
Gitflow workflow – origin

A successful Git branching model



By [Vincent Driessen](#)
on Tuesday, January 05, 2010

In this post I present the development model that I've introduced for some of my projects (both at work and private) about a year ago, and which has turned out to be very successful. I've been meaning to write about it for a while now, but I've never really found the time to do so thoroughly, until now. I won't talk about any of the projects' details, merely about the branching strategy and release management.



Oryginalny post który opisuje git flow znajdzicie: <http://nvie.com/posts/a-successful-git-branching-model/>

Gitflow workflow – jak to działa?

Podział na brache wg funkcji.

Wszystkie nazwy są konwencją.

Więc - to tylko kontrakt pomiędzy developerami.

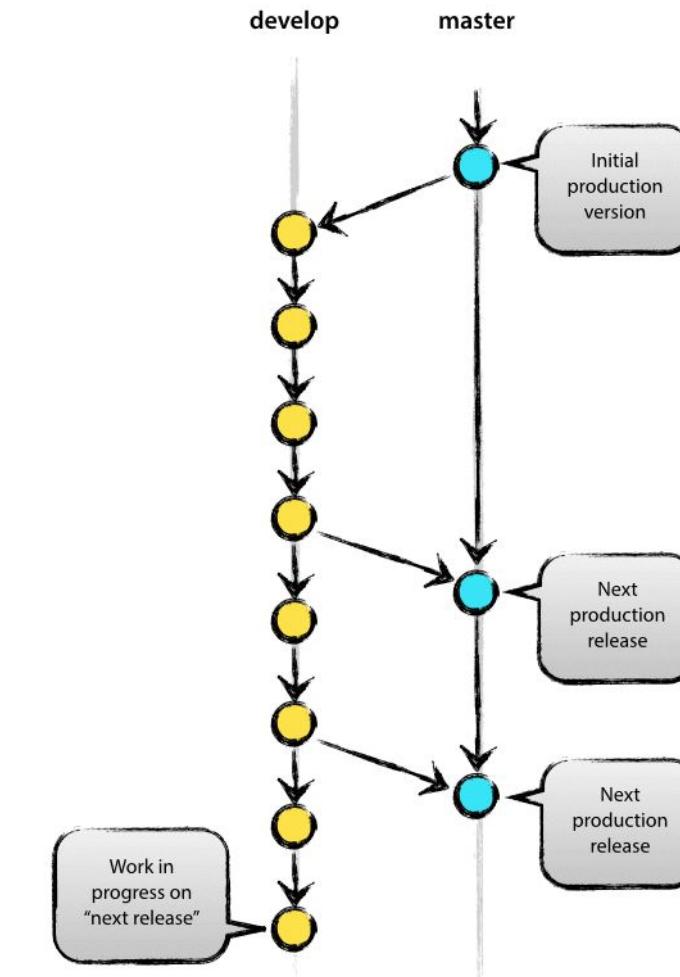
Tak naprawdę branche się technicznie niczym nie różnią.



Gitflow workflow – podstawowe branche

develop: główna gałąź rozwojowa. Tutaj przygotowywujemy kod do kolejnych wydań.

master: główna gałąź produkcyjna. Kod z tej gałęzi działa na serwerze i to z niego korzystają klienci.



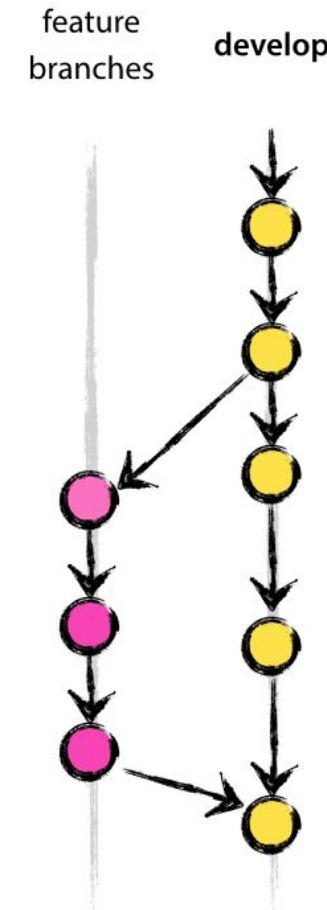
Źródło: <http://nvie.com/posts/a-successful-git-branching-model/>

Gitflow workflow – branche wspierające

feature/XY-feature-name: gałąź na której rozwijamy jakiś feature, np. przeszukiwanie użytkowników po imieniu.

Dlaczego **feature/*?**

Aby łatwo dało się odróżnić te branche



Źródło: <http://nvie.com/posts/a-successful-git-branching-model/>

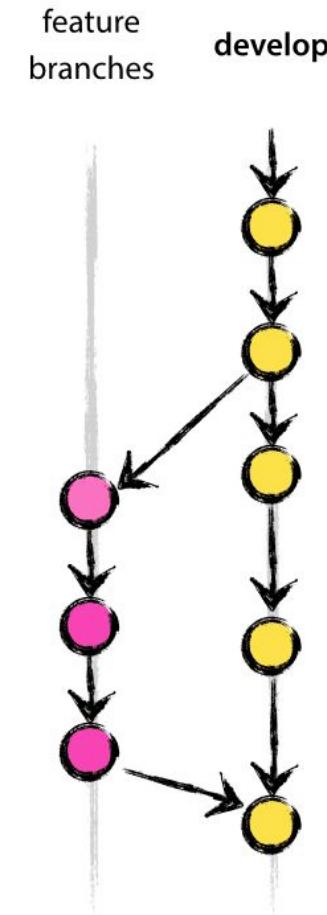
Gitflow workflow – branche wspierające

A dlaczego **feature/XY-feature-name?**

Gdy korzystamy z issue trackerów (np JIRA). Każde zadanie ma jakiś numer.

Np.:**MP-100**

Użyjmy go. Łatwiej będzie potem powiązać branche/commity z wymaganiem.



Źródło: <http://nvie.com/posts/a-successful-git-branching-model/>

[Code] git flow – init & feature branches



Gitflow workflow – branche przygotowujące do wydania

Przygotowywujemy się do wdrożenia na serwer produkcyjny.

Ale najpierw upewnijmy się czy wszystko działa.

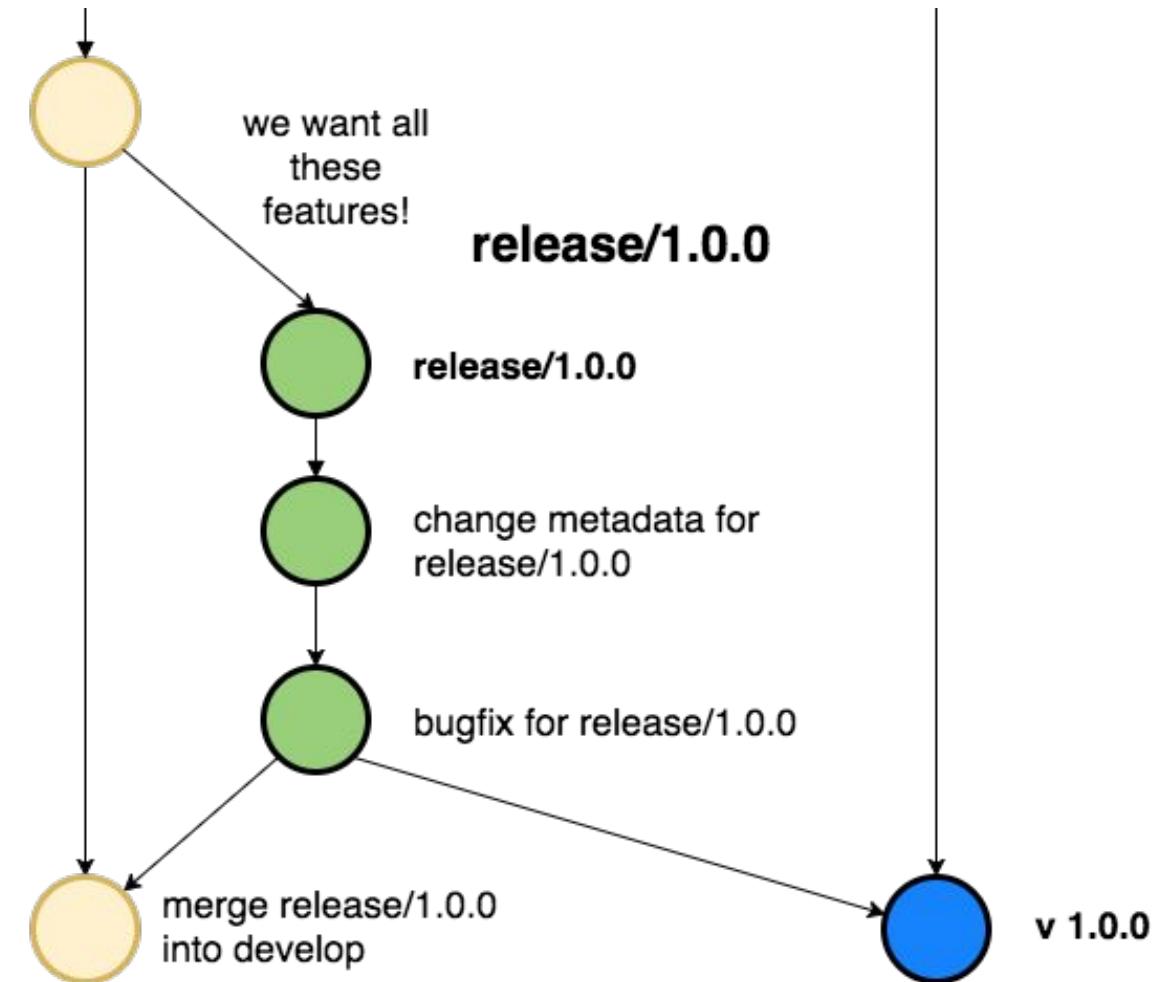
Branche **release/*** są również wdrażane, ale na testowe serwery.



Gitflow workflow – po wydaniu

Te bugfixy które robiliśmy na release branchu.

Przydadzą się też na develop-ie.



Gitflow workflow – hotfix. Kiedy płonie

Czasem zdarzy się **błąd**.

Czasem dosyć **krytyczny**.

Czasem wprowadzić poprawkę
od razu.

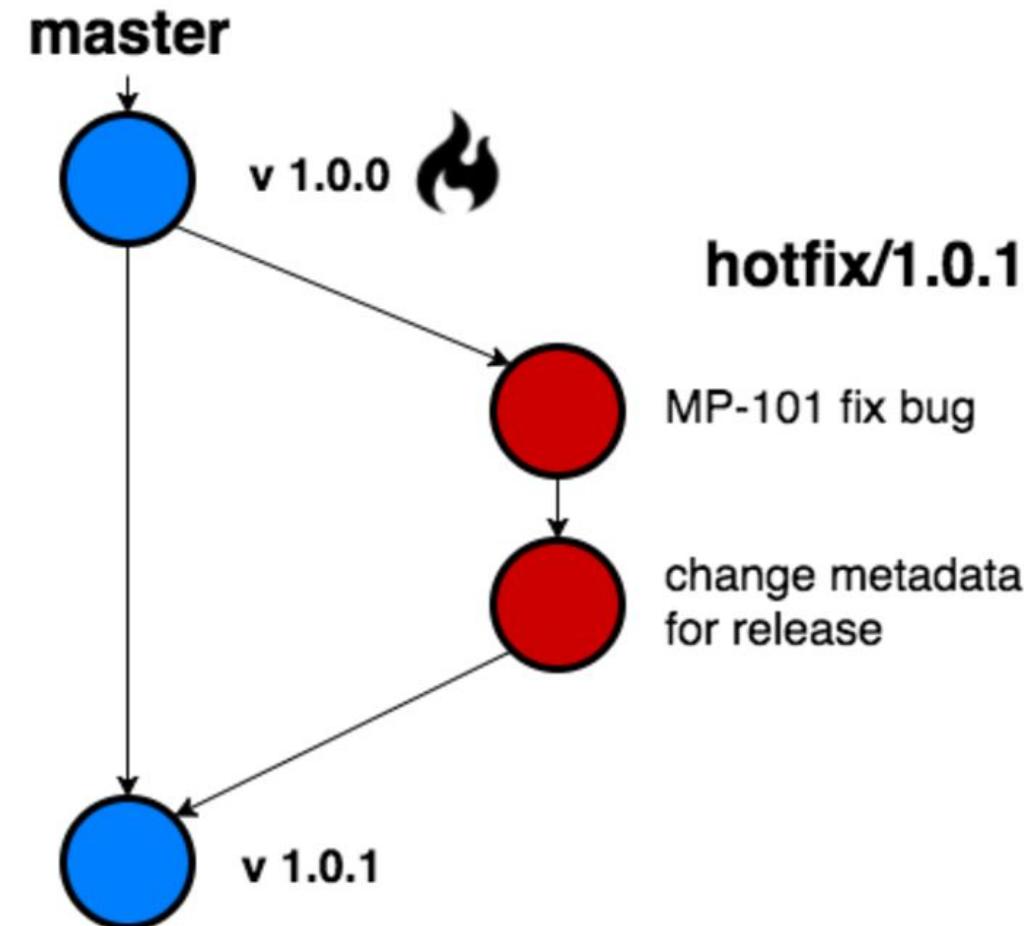
Od tego mamy **hotfixy**.



Gitflow workflow – hotfix. Kiedy płonie

Chcemy wprowadzić poprawkę bez wrzucania nowych feature-ów.

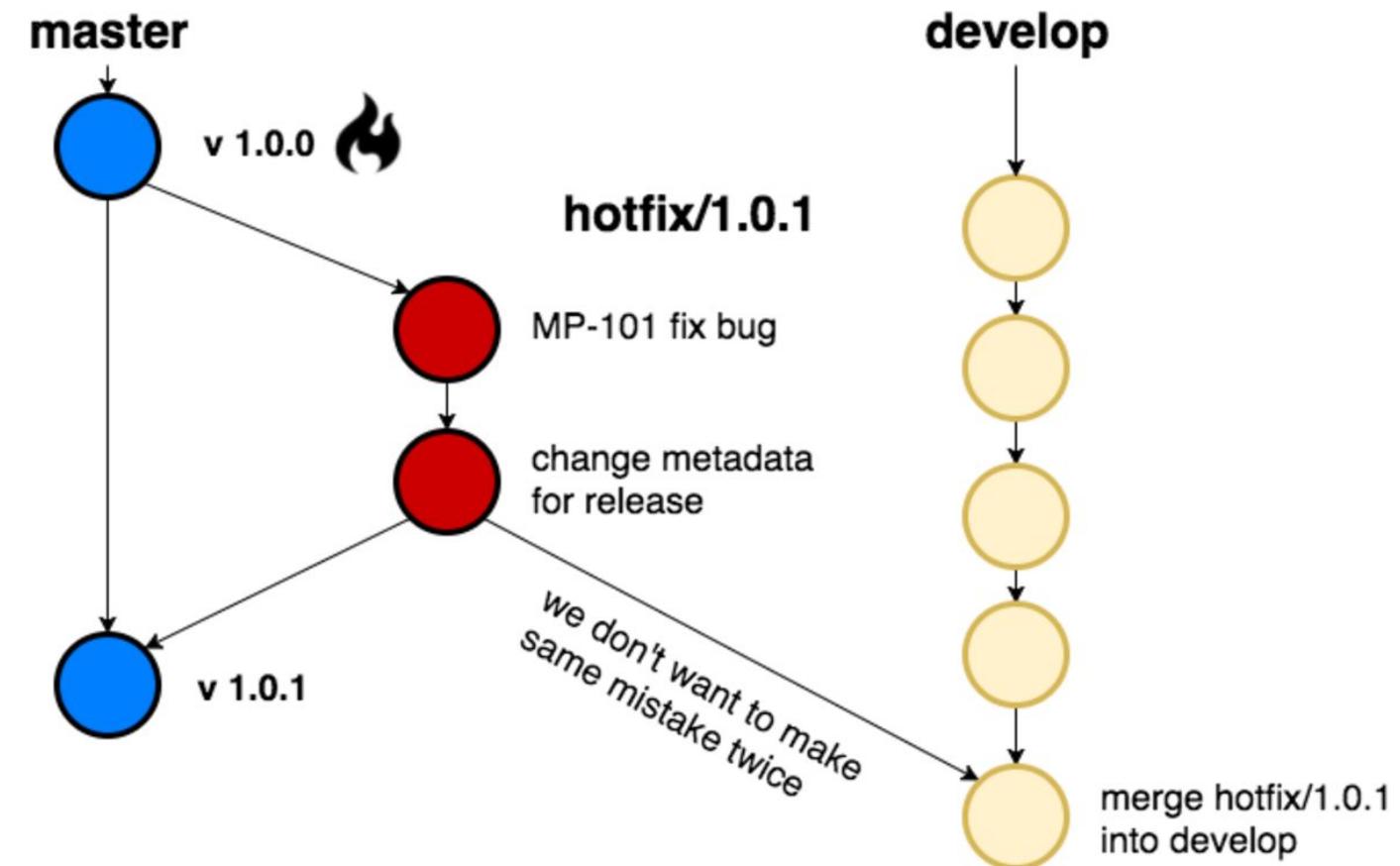
Zwłaszcza tych niewytestowanych.



Gitflow workflow – hotfix. Po wydaniu

Warto nie naprawiać tego samego buga 2 razy.

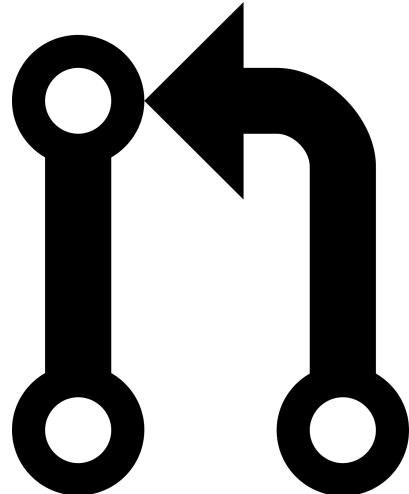
Więc mergujemy hotfix-a do developa.



[Code] git flow – release, hotfix



Pull requests

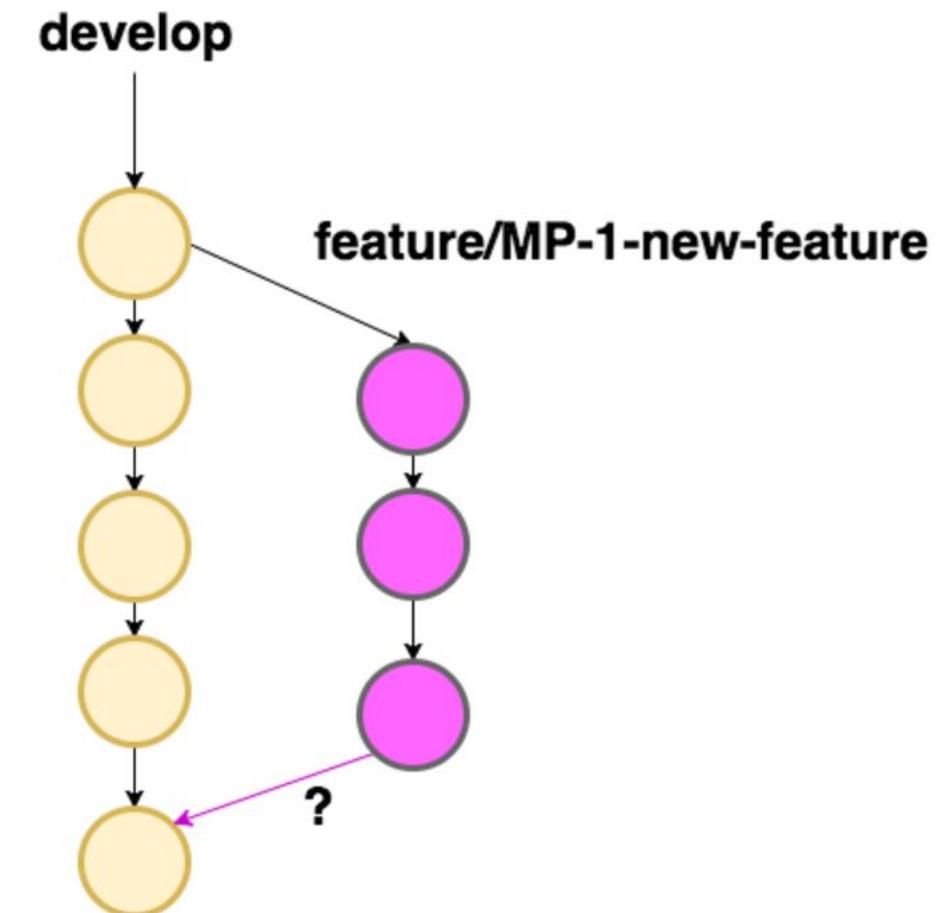


Pull request - żądanie wcielenia kodu

Mamy bogaty model branchowania.

Jest ok.

Ale kiedy stwierdzić że nasz feature jest gotowy do wrzucenia do developa?



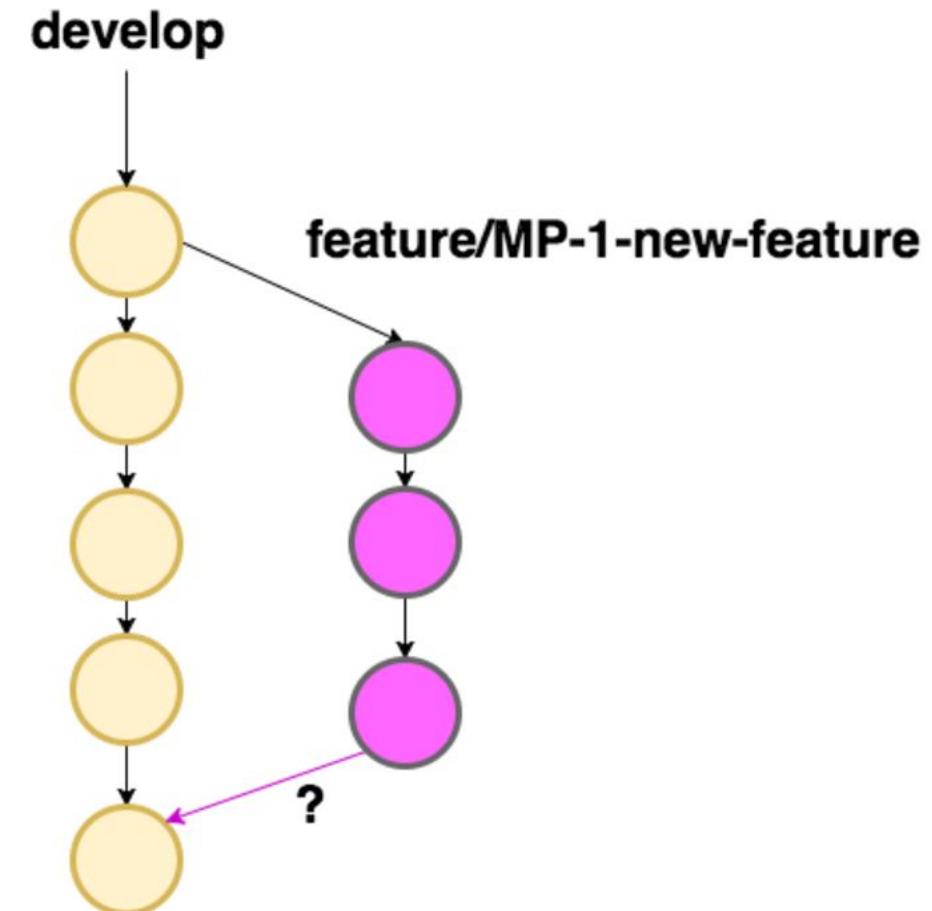
Pull request – code review

Dokładnie w tym momencie kod powinien zostać zweryfikowany.

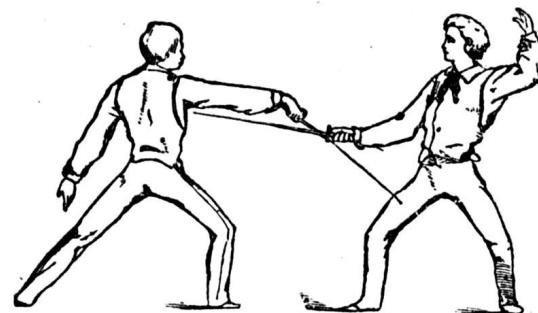
Przez innego developera.

Jeśli oboje zgadzają się na ten stan kodu ..
można mergować :)

Nazywamy to code review.



Gdzie hostować? GH vs BB





vs



GitHub

*GitHub is a **code hosting platform** for version control and collaboration.
It lets you and others work together on projects from anywhere.*



[GitHub guides](#)

BitBucket

Built for professional teams

Distributed version control system that makes it easy for you to collaborate with your team.

The only collaborative Git solution that massively scales.

Code collaboration on steroids

Approve code review more efficiently with pull requests. Hold discussions right in the source code with inline comments.

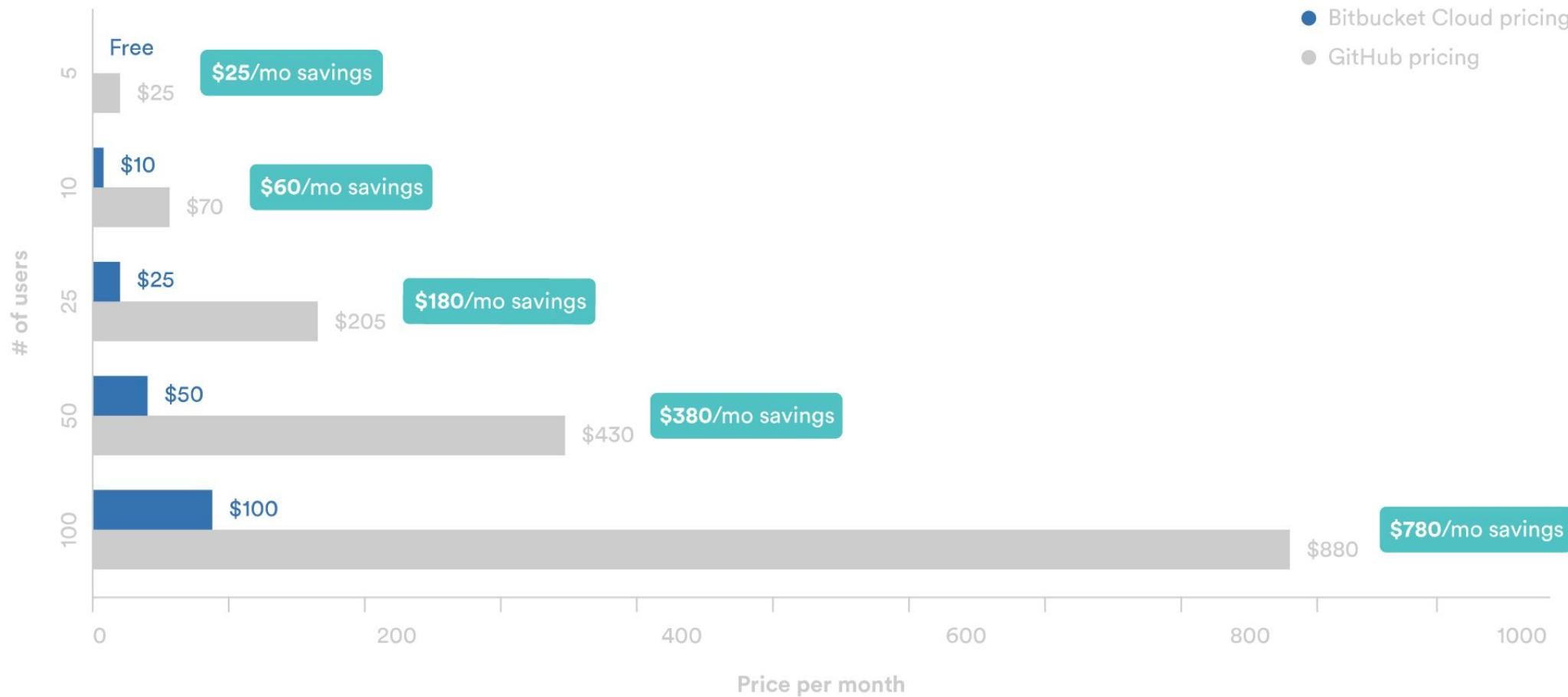


[BitBucket documentation](#)

Małe porównanie

Co?	GitHub	BitBucket
Darmowe publiczne repozytoria	tak	tak
Darmowe prywatne repozytoria	nie	tak (do 5 osób)
Integracje z IDE/git GUI	tak	tak
Pull requesty	tak	tak
Git Large File Storage (LFS)	tak	tak
Integracje z narzędziami CI / Issue Trackerami/ Wiki itd	tak	tak

Pricing wg Atlassian



Źródło: <https://www.atlassian.com/software/bitbucket/comparison/bitbucket-vs-github>

IMHO

Bitbucket

Dashboard

Overview Repositories Pull requests Issues Snippets



Everything's awesome!

All your pull requests and reviews are done and dusted.

[View all pull requests](#)

Repositories

Repository

C#

</>

</>

</>

C#

</>

</>

C#

</>

My private repositories

[View all repositories](#)

A screenshot of a GitHub user profile for 'michalcukm'. The profile features a photo of a man with a beard and glasses, wearing a green and blue plaid shirt. The top navigation bar includes links for 'Pull requests', 'Issues', and 'Gist'. Below the photo, the user's name 'Michał Michalczuk' and handle 'michalcukm' are displayed, along with a 'Add a bio' button and an 'Edit profile' button. A sidebar on the left shows '175 contributions in the last year' and a heatmap of activity. The main content area displays five pinned repositories: 'typescript-workshops' (3 stars, TypeScript), 'good-looking-personal' (0 stars, HTML), 'michalklim/gameJamSuperDuper' (0 stars, JavaScript), 'search-tweets.win8.1.phone' (0 stars, C#), and 'DmitryEfimenko/FlowJs-MVC' (23 stars, C#). Each repository card includes a brief description and a star rating.

IMHO

Co?	GitHub	BitBucket
Darmowe publiczne repozytoria	tak	tak
Darmowe prywatne repozytoria	nie	tak (do 5 osób)
Integracje z IDE/git GUI	tak	tak
Pull requesty	tak	tak
Git Large File Storage (LFS)	tak	tak
Integracje z narzędziami CI / Issue Trackerami/ Wiki itd	tak	tak

Inne usługi warte przejrzenia

GitLab

Download Features Pricing Community Explore Blog Sign In

GitLab Master Plan: Check out the highlights, recording, and slides!

Tools for modern developers

GitLab unifies chat, issues, code review, CI and CD into a single UI

[Sign Up](#) [Learn More](#)

IDEA → ISSUE → PLAN → CODE → COMMIT → TEST → REVIEW → STAGING → PRODUCTION →

370 Sync offline db
Johnnie McLeod Database

323 Exception thrown when entering a bad airport
Jamal Hartnett Severity 2 - High

326 As a customer I should be able to customize overnight kit bags as necessary
Francis Totten

328 As a customer I should be able to book a one-way flight.
Johnnie McLeod

375 Flight Manager failure

308 As a customer, I want to be able to know where my bags are at all times

365 Build Failure in Build:

GitLab: <https://gitlab.com>

Visual Studio

Products Features Downloads News Support Marketplace Documentation Free Visual Studio

MSDN Subscriptions Michal Michalcuk Sign out

Visual Studio Online is now Visual Studio Team Services

Visual Studio Team Services

Services for teams to share code, track work, and ship software – for any language, all in a single package. It's the perfect complement to your IDE.

[Get started for FREE](#)

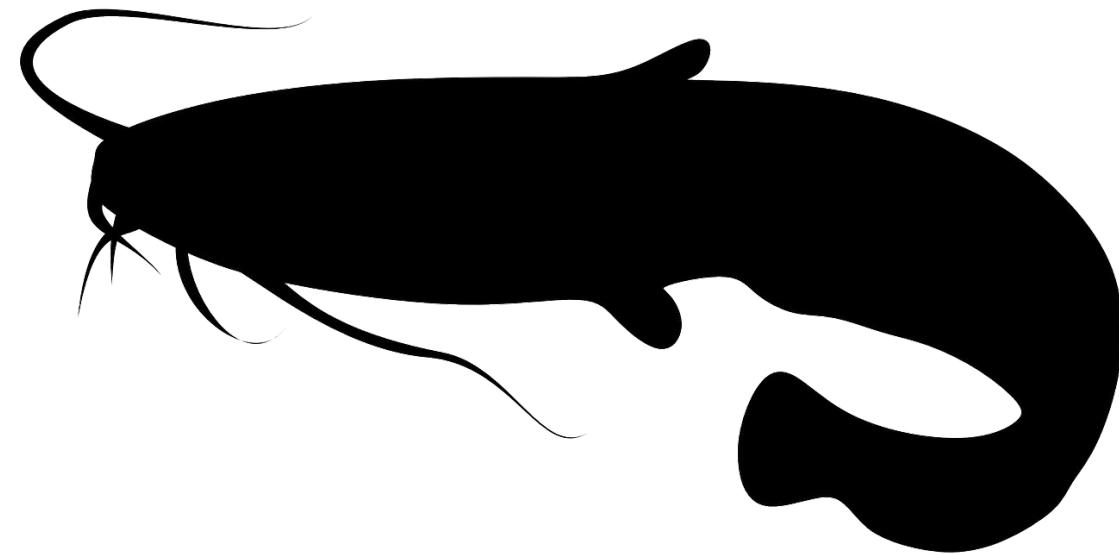
Pricing >

New	Under Analysis	In Development	Release	Done
370 Sync offline db Johnnie McLeod Database	326 Exception when adding infant Francis Totten	312 As a customer, I want to see cheaper flights if they are available Jia-hao Tseng UX	311 As a customer, I want to see cheaper flights if they are available Jia-hao Tseng UX	
323 Exception thrown when entering a bad airport Jamal Hartnett Severity 2 - High	328 As a customer I should be able to book a one-way flight. Johnnie McLeod	308 As a customer, I want to be able to know where my bags are at all times	322 Exception when triple-clicking on a flight Nicole Zamora Severity 3 - Medium	
326 As a customer I should be able to customize overnight kit bags as necessary Francis Totten	375 Flight Manager failure	365 Build Failure in Build:	320 Exception thrown when double clicking on flight details Johnnie McLeod	

Any team, any project

VS Team Services: [https://www.visualstudio.com/...](https://www.visualstudio.com/)

Podsumowanie



Podsumowanie 1/2

- git jest aktualnie najpopularniejszym system kontroli wersji, ale nie jedynym
- git jest **szybki**, to leży u jego podstaw
- git jest **rozproszony**. Repozytorium jest zarówno u ciebie jak i reszty zespoły, oraz na zdalnym serwerze (np GitHub-ie)
- commity to delty (różnice) w zawartości tekstu
- branche to tylko wskaźniki. Pokazuję na dany commit
- **commity są persistent (stałe)**, branche nie
- commita nie da się zmienić - zawsze jest tworzony nowy

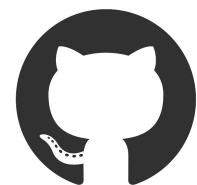
Podsumowanie 2/2

- **revert** robi commita będącego odwrotnością tego co chemy wycofać
- **fetch** pobiera commity z zdalnego repozytorium (np.: na GitHubie), ale nie włącza ich do naszego lokalnego repo
- **pull** włącza commity do naszego lokalnego repo i ustawia branche (wskaźniki)
- nie ma narzuconego workflow dla pracy z git. Ale doświadczenie wypracowało parę dobrych. My promujemy git flow.
- jest wiele usług hostingowych dla repozytorów gita. Najpopularniejsze to Github i Bitbucket

Parę przydatnych linków

- Atlassian git resources, getting started: <https://www.atlassian.com/git/tutorials/what-is-version-control>
- Atlassian git resources, collaborating: <https://www.atlassian.com/git/tutorials syncing>
- Git flow cheatsheet: <http://danielkummer.github.io/git-flow-cheatsheet/>
- Atlassian git resources, workflows: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- The 7 great commit rules <http://chris.beams.io/posts/git-commit/>
- Git console in 15 minutes: <https://try.github.io/>

Dziękuję za uwagę



michalczukm

michalczukm@gmail.com