

| GIT 

# System kontroli wersji



# Kto mówi

Michał Michalczuk

Full-Stack Software Developer

Senior JavaScript Developer @ Ciklum

# 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 / IntelliJ
- Jak wydajnie pracować w wiele osób?
- Gdzie trzymać moje repozytorium? Czyli czym jest ten GitHub
- Klienci git

# Problem

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

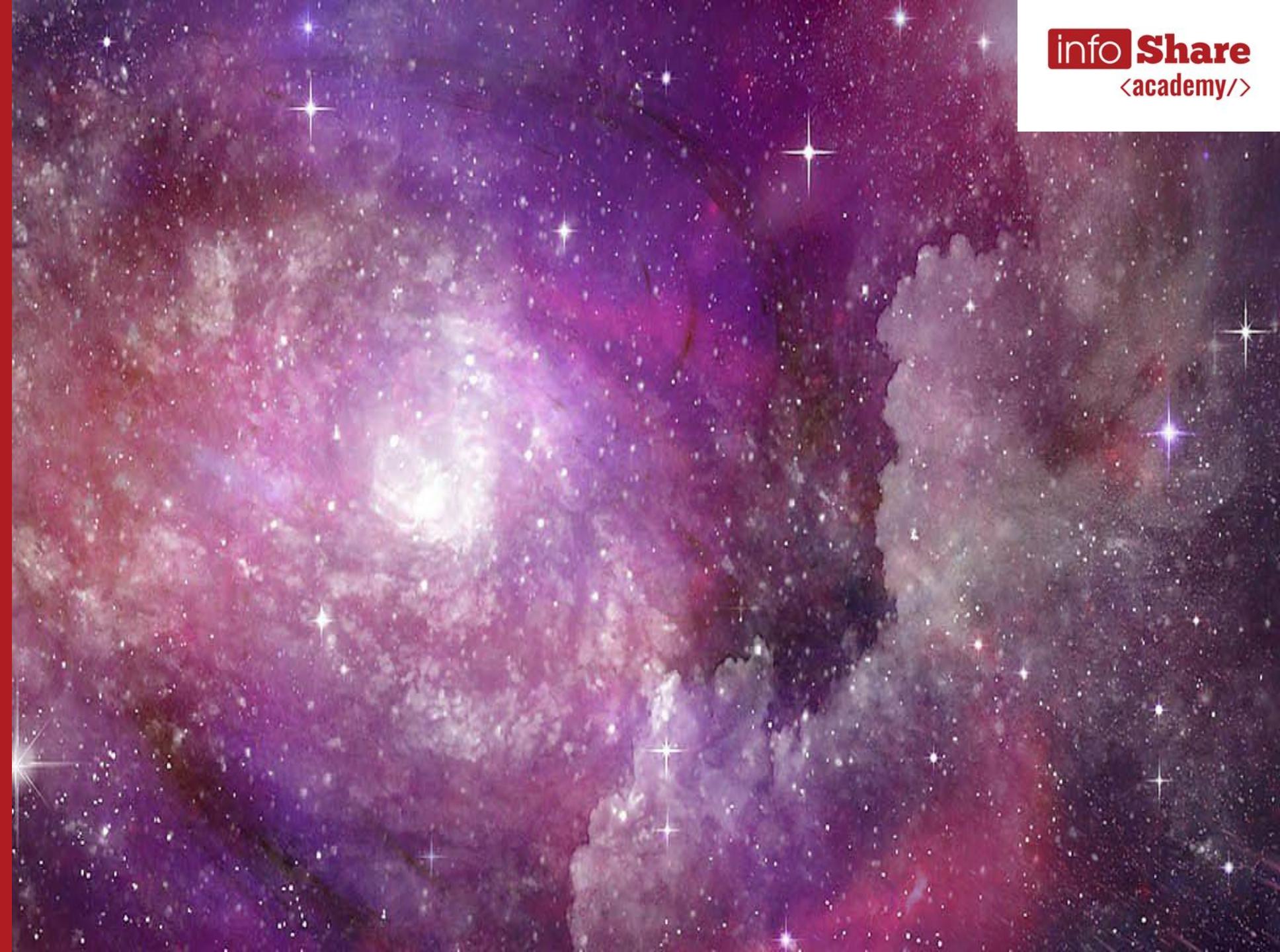


# Problem jest powszechny

Wg wikipedi  
istnieje/istniało  
45 systemów  
kontroli wersji.

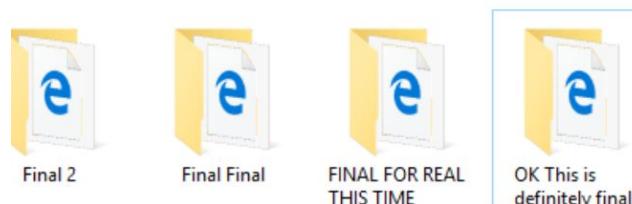
Znanych.

[https://en.wikipedia.org/wiki/List\\_of\\_version\\_control\\_software](https://en.wikipedia.org/wiki/List_of_version_control_software)

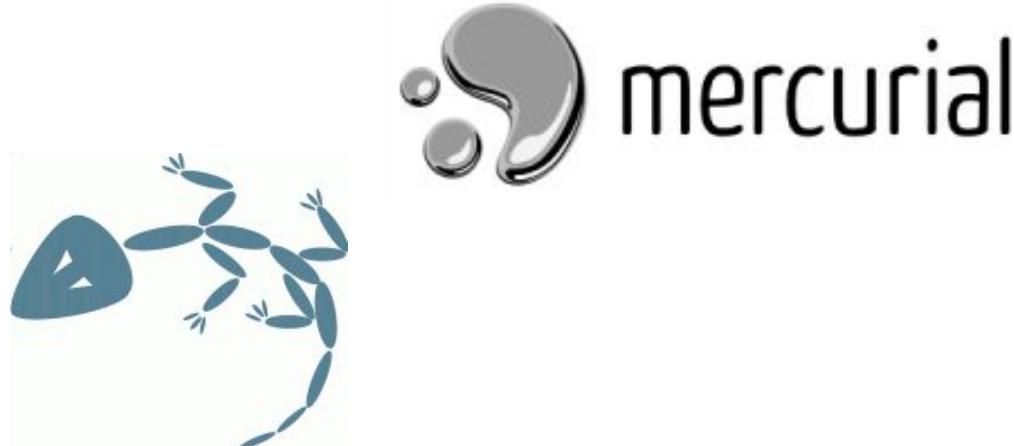
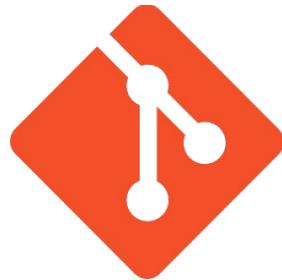


# Jak radziliście sobie z tymi problemami?





# Podział systemów kontroli wersji



# Podział systemów kontroli wersji

## Distributed (rozproszone)

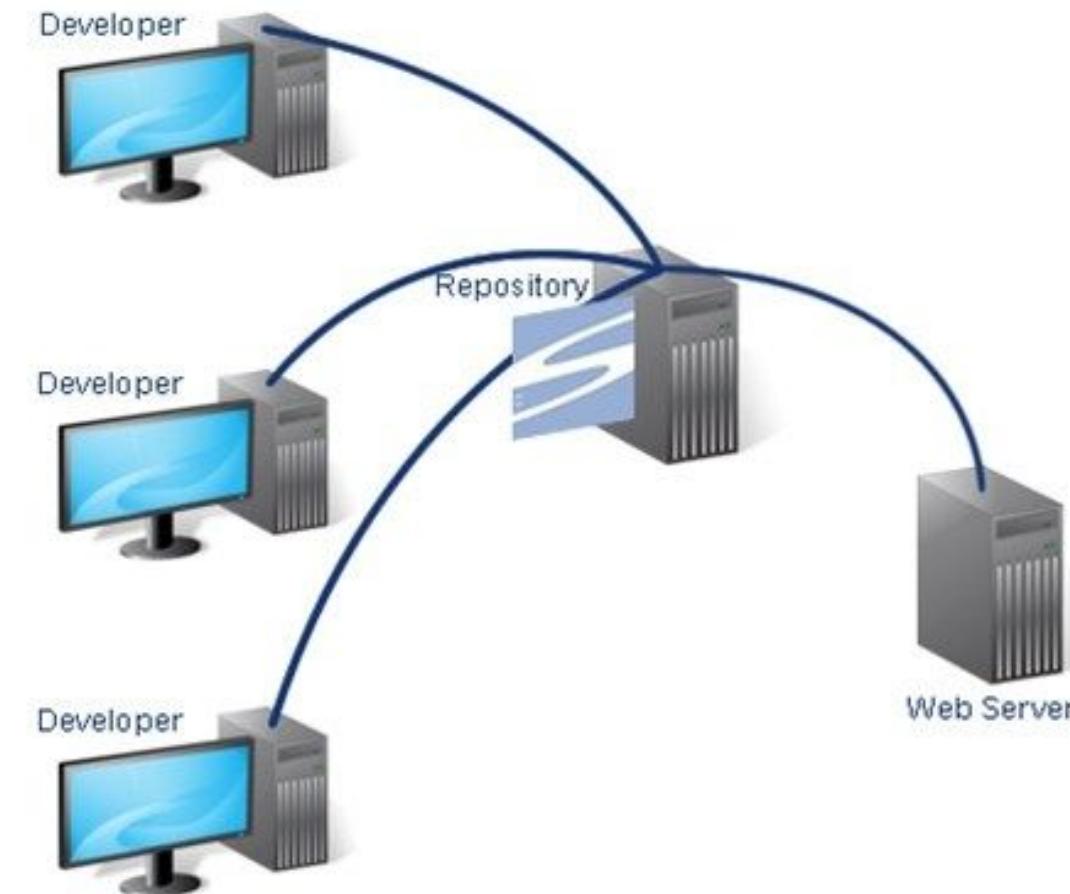


## Client-Server



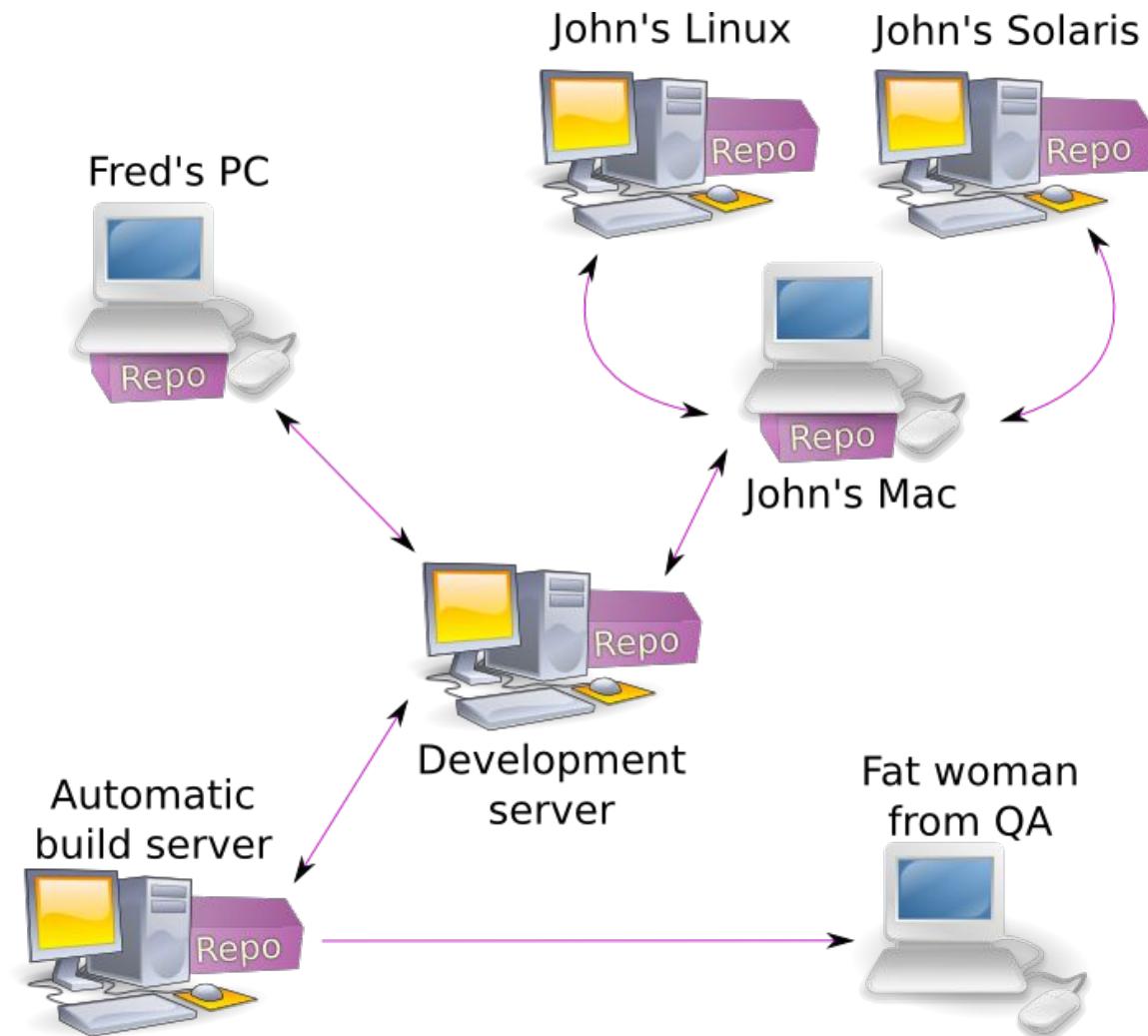
# Client-server (Centralne)

- Mamy tylko jedno repozytorium na serwerze
- Synchronizujemy wersje zawsze na serwerze
- Jeśli coś się zepsuje to mamy poważny problem



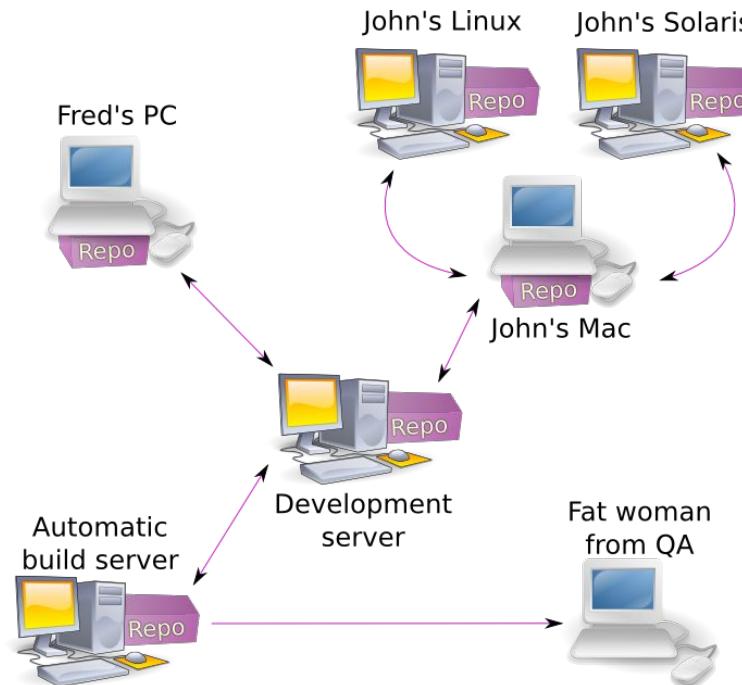
# Distributed (Rozproszone)

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



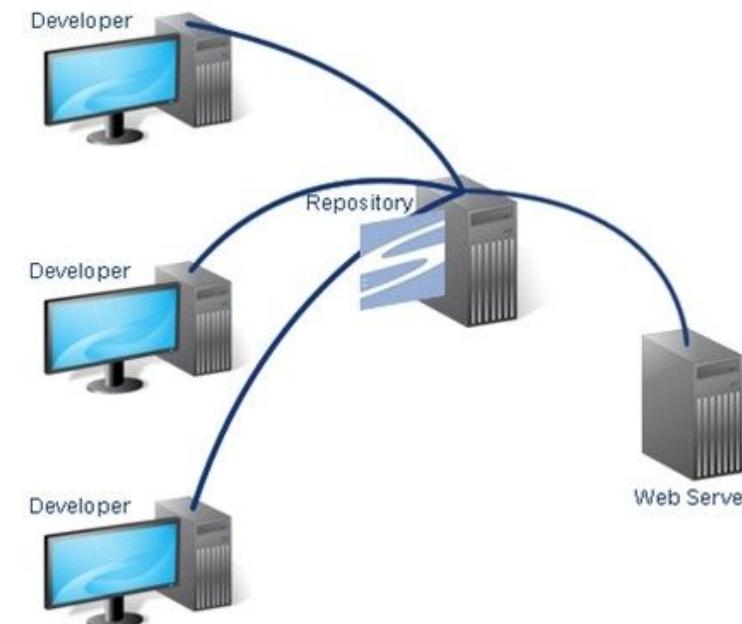
# Podział systemów kontroli wersji

## 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

## System kontroli wersji

Od czego się zaczęło? Czym się cechuje?



# Od czego się zaczęło

2005 r.

Jako narzędzie do  
wersjonowania kodu przy  
rozwijaniu Linuxa.



[Linus Torvalds](#)

# Od czego się zaczęło

Dlaczego?

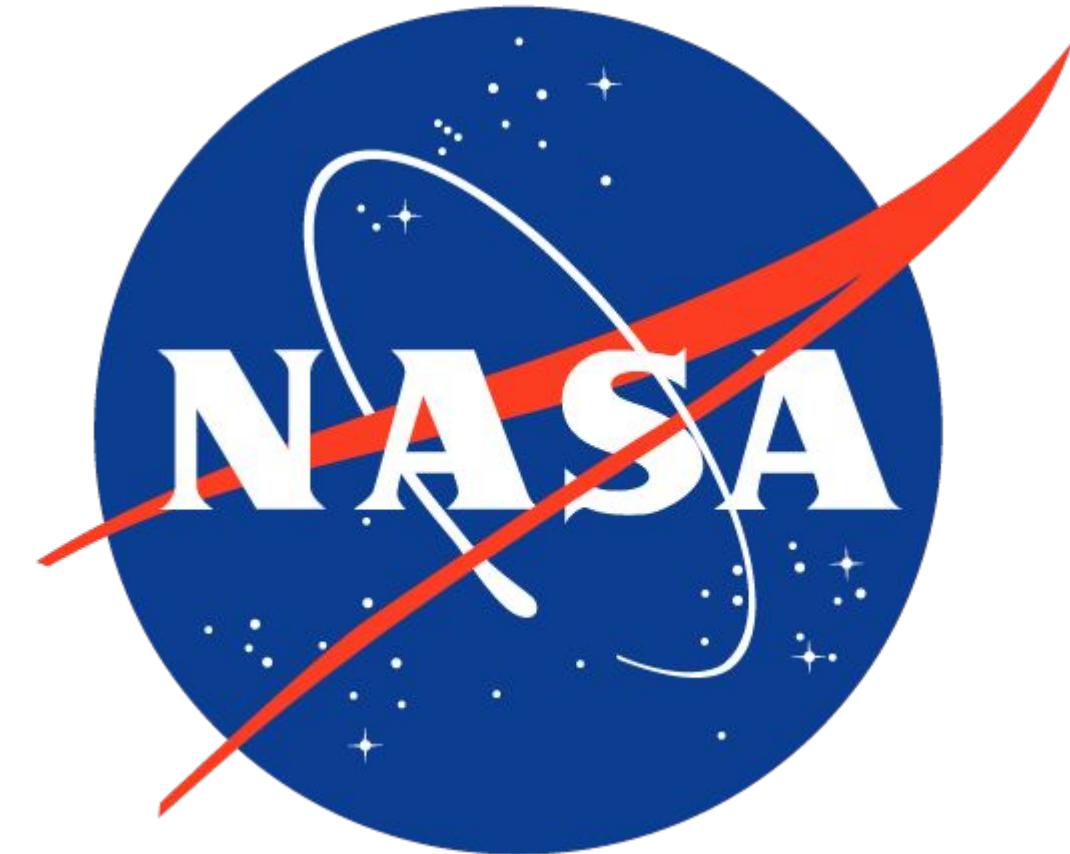
- Wydajność i prędkość narzędzia
- Śledzenie zawartości a nie plików
- Podejście od strony systemu plików



[Linus Torvalds](#)

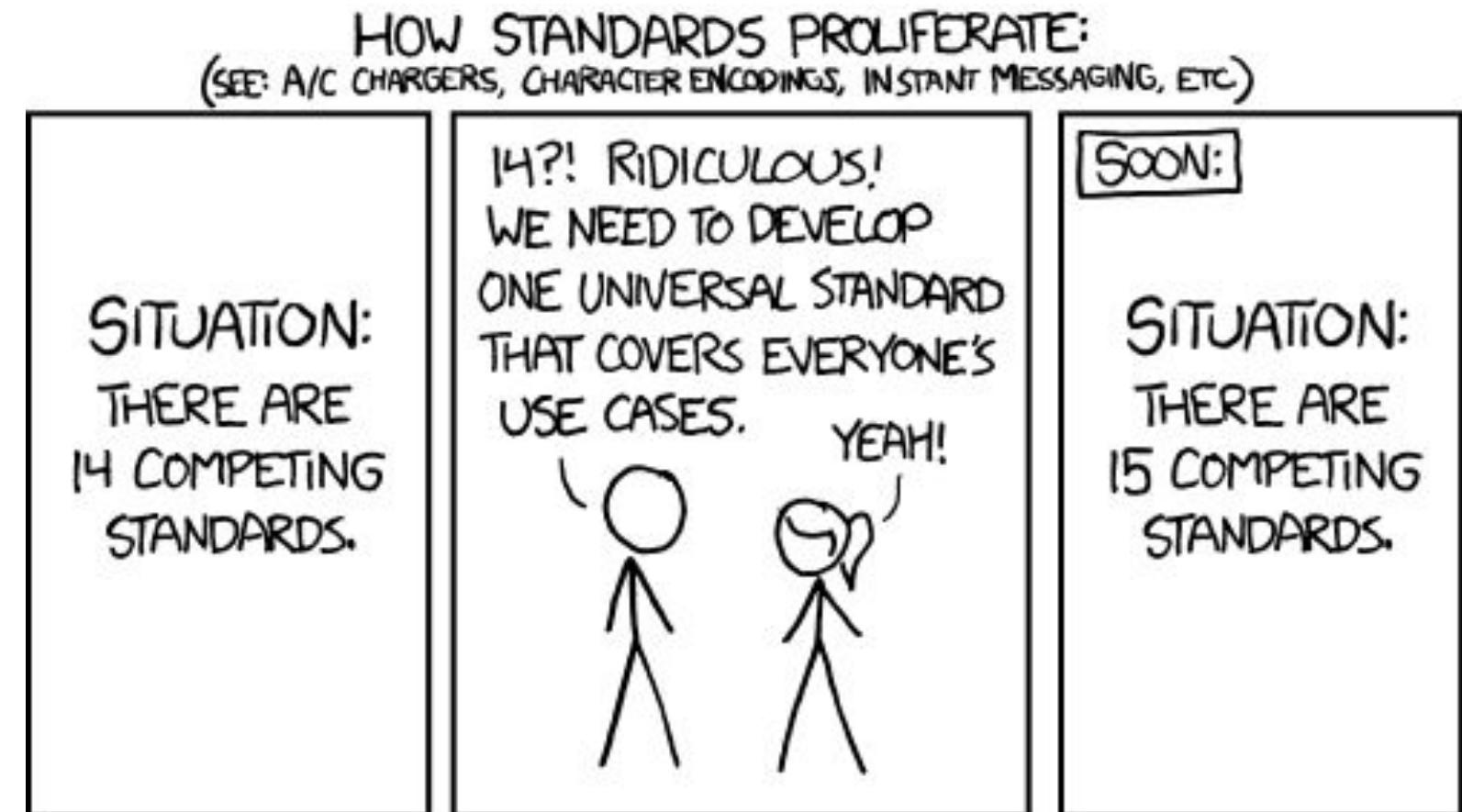
## Ciekawostka na dzisiaj

NASA może używać  
GIT'a od 2015r



# Git jest teraz nr 1.

Ale mogło być tak



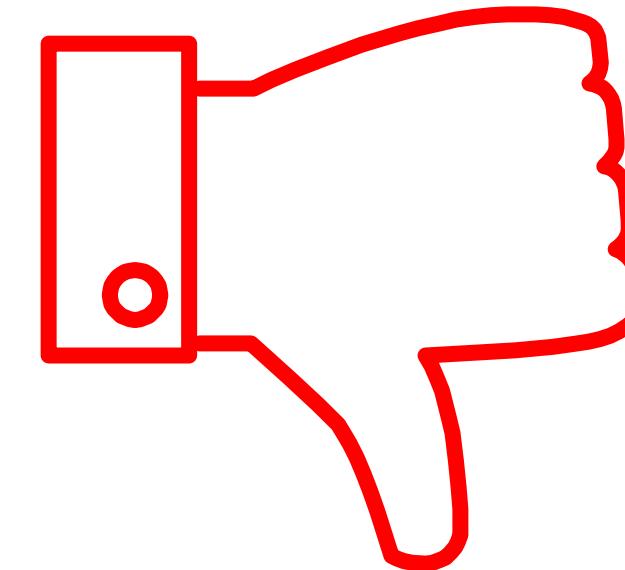
# GIT - zalety

- Rozproszony
- Szybki
- Bezpieczny
- Elastyczny – wiele flow pracy
- Darmowy i rozwijany jako OSS
- Standard



# GIT - wady wg internetu

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



Rly?

# GIT - zapis pracy

- zbiór różnic (delty)
- różnice w zawartości plików

	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

# GIT - zapis pracy

## Gdy pracuję sam

Zbiór zmian.

Krok po kroku

-  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

# Git zapis pracy różnica zawartości plików

```
1 - <!doctype html>
2 <html>
3 <head>
4 -   <meta charset="utf-8">
5   <link rel="stylesheet" type="text/css" href="styles.css">
6 </head>
7
```

```
1 <html>
2 <head>
3 +   <meta charset="UTF-8">
4   <link rel="stylesheet" type="text/css" href="styles.css">
5 +
6 +   <meta name="description" content="Git diff">
7 +   <meta name="keywords" content="git,diff,svc">
8   </head>
9
```

# Czym będziemy się posługiwać? Konsola, czasem GIT w IDE

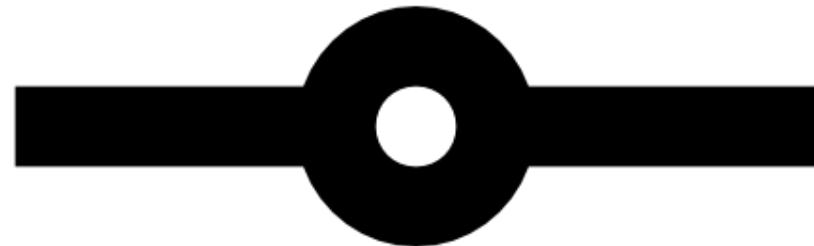
Część operacji nazywa się  
inaczej w WS/IJ niż w GIT.

Ponieważ mają ujednoliconą  
obsługę wielu VCS.

Zostaliście ostrzeżeni 😊



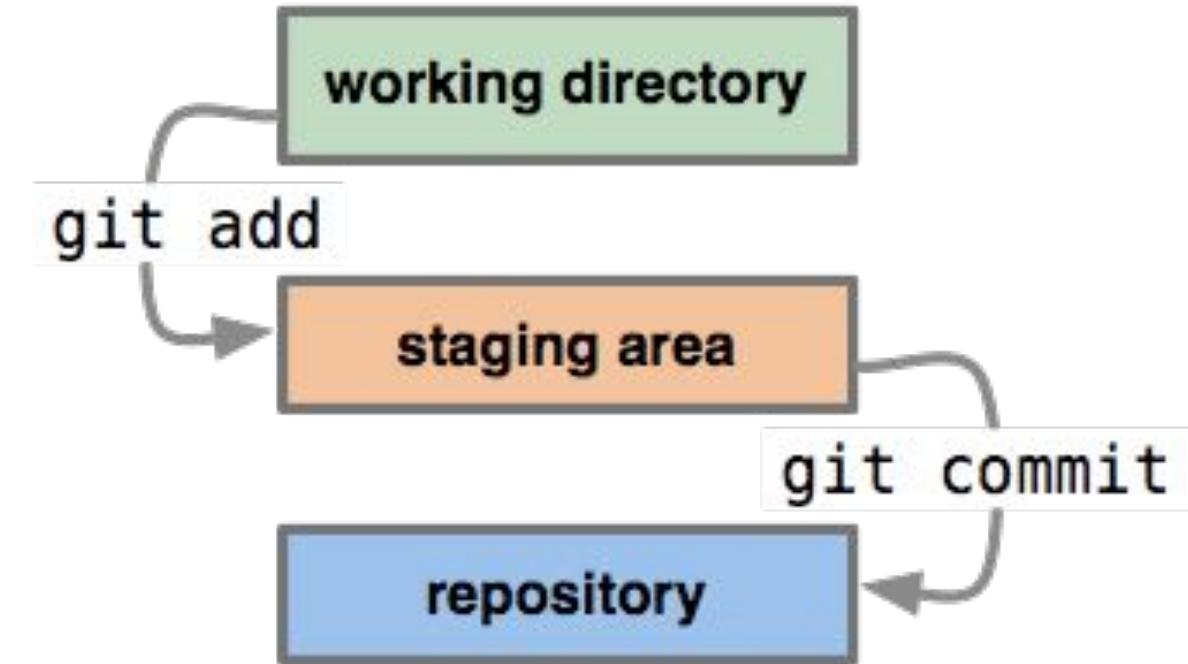
# Commit oraz podstawowe operacje



# Commit

git commit

- Zapisanie różnicy zawartości plików.
- Commity są stałe i niezmienne.
- Commit odpowiada konkretnym zmianom.
- Commit ma wiadomość



# Zanim zaczniemy commitować Nie chcemy wszystkiego commitować

Prawda?

Czego nie warto trzymać na repozytorium?

- Wynik kompilacji kodu
- Personalne ustawienia IDE
- Pliki generowane (nie zawsze)



# .gitignore

Ignoruj, nie śledź

## .gitignore

Specjalny plik w **bazowym katalogu**  
repozytorium.

Zawiera spis rzeczy których nie chcemy śledzić/wersjonować.

```
◆ .gitignore ✘
1 # Created by https://www.gitignore.io/api/webstorm
2
3 # OSX specific files
4 **/.DS_Store
5
6 ### WebStorm ###
7 # Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm,
8 # Reference: https://intellij-support.jetbrains.com/hc/
9
10 # User-specific stuff:
11 .idea/workspace.xml
12 .idea/tasks.xml
13 .idea/dictionaries
14 .idea/vcs.xml
15 .idea/jsLibraryMappings.xml
16
17 # Sensitive or high-churn files:
18 .idea/dataSources.ids
19 .idea/dataSources.xml
```

# .gitignore

Ignoruj, nie śledź

## .gitignore

Musi być w **bazowym katalogu**

repozytorium.

Wszystkie ścieżki muszą być **relatywne**  
do jego położenia.

▼ **jjdz4-materiały-git** ~/Desktop/jjdz4-materiały-git

► .idea

▼ application

App

Doctor

DoctorService

Patient

PatientService

Visit

► out

► presentations

.gitignore

jjdz4-materiały-git.iml

README.md

```
drwxr-xr-x 11 michalczukm staff 352B Oct 15 13:41 .
drwx-----+ 6 michalczukm staff 192B Oct 15 14:08 ..
-rw-r--r--@ 1 michalczukm staff 6.0K Oct 15 13:39 .DS_Store
drwxr-xr-x 14 michalczukm staff 448B Oct 15 14:08 .git
-rw-r--r-- 1 michalczukm staff 1.4K Oct 15 13:41 .gitignore
drwxr-xr-x 7 michalczukm staff 224B Oct 15 14:08 .idea
-rw-r--r-- 1 michalczukm staff 2.8K Oct 15 13:33 README.md
drwxr-xr-x 9 michalczukm staff 288B Oct 15 14:03 application
-rw-r--r-- 1 michalczukm staff 431B Oct 15 13:36 jjdz4-materiały-git.iml
drwxr-xr-x 3 michalczukm staff 96B Oct 15 13:40 out
drwxr-xr-x 3 michalczukm staff 96B Oct 15 13:33 presentations
```

# .gitignore

## Jak go napisać?

Polecam zacząć od  
wygenerowania pliku przez

<https://www.gitignore.io/>

gitignore.io

Create useful .gitignore files for your project

Java IntelliJ WebStorm

Create

[Source Code](#) | [Command Line Docs](#) | [Watch Video Tutorial](#)

# Ćwiczenie commitowanie

Najpierw sprawdź czy masz poprawne ustawienia git-a:

```
cat ~/.gitconfig
```

powinno dać (oczywiście z waszymi danymi):

```
[user]
    name = Michal Michalczuk
    email = michalczukm@gmail.com
[core]
    autocrlf = true
```

```
# ustawienie autocrlf na true
git config --global core.autocrlf true
```

---

Zaraz dostaniecie ode mnie .zip-a.

Wypakuj go do folderu “**projekt-michala**”

Otwórz go w WebStorm / IntelliJ. Powiem co dalej zrobić :)

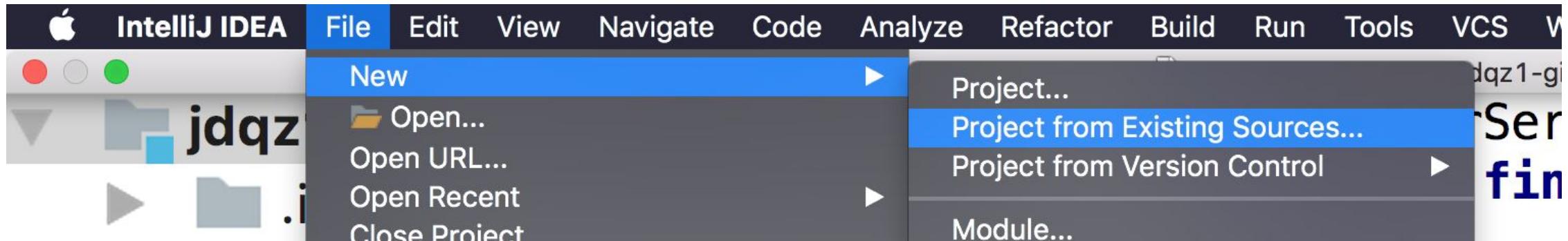


# Ćwiczenie

## założmy lokalne repozytorium

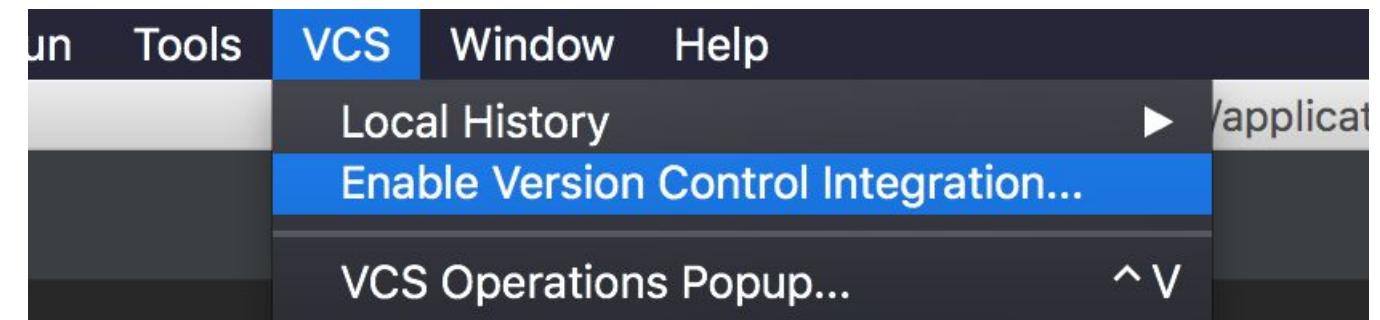


Otwórzcie katalog z projektem (to co było zzipowane)



Stwórz lokalne repozytorium

git init lub



# Operacje które właśnie wykonaliśmy



# Status

```
git status
```

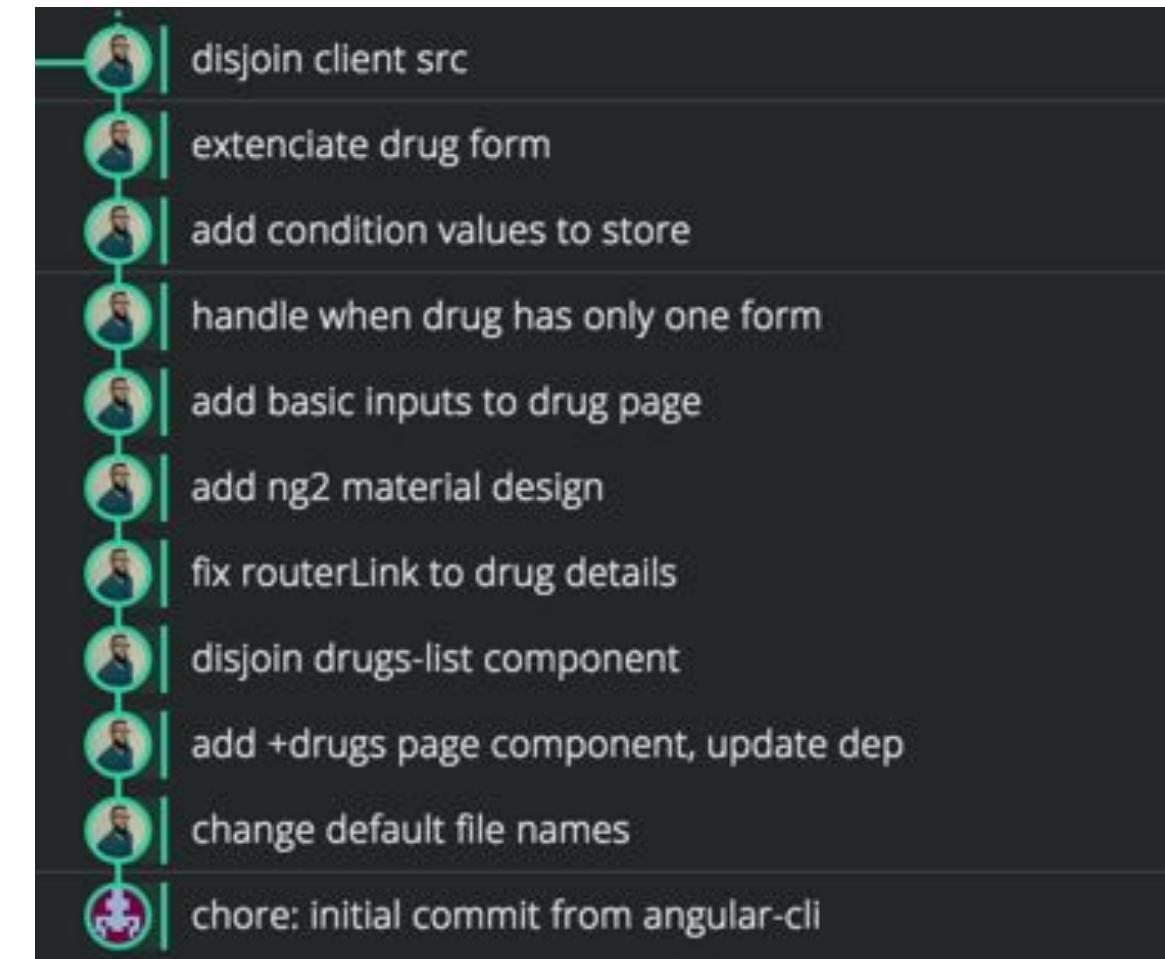
Sprawdź jaki jest aktualny stan repozytorium.

**Oraz** - w jakim “kubelku”, są twoje pliki



# Commit jeszcze zawiera

- Ma autora
- Każdy commit ma swojego rodzica
- Każdy commit ma swój klucz *Hash SHA1 (suma kontrolna)*
- Ten klucz jest unikalny !
- Tak naprawdę git trzyma pary: klucz: wartość



# Dobry Commit

## Zawartość i wiadomość

- Z wiadomości da się jednoznacznie określić czego dotyczył
- Wiadomość nie jest zbyt długa
- Wiadomość nie jest zbyt szczegółowa
- Można go powiązać z zadaniem/wymaganiem

task-5 Now users list contains user email

“ *Use the body to explain what and why vs how*

Czyli - nikogo nie interesuje w wiadomości commita że dodaliście klasę `UserService` w Javie lub `user-list` w CSS.

Interesuje ich **po co** to zrobiliście.

# Zły commit

Don't do this at home

	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.

## Sprawdzamy historię plików

git blame <path>

git annotate <path>

- Spójrzmy jak zmieniał się nasz kod
- Spójrzmy kto i kiedy nanosił zmiany



# Ćwiczenie

## Historia zmian

1. Log
2. git blame
3. Co oferuje nam WebStorm/IntelliJ



# WIADOMOŚCI COMMITÓW POWRACAJĄ

!

Zaglądasz do historii dużego projektu.  
Dużo commitów.

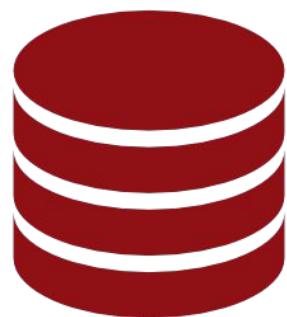
Co chciałbyś zobaczyć?

“ *Use the body to explain what and why vs how*

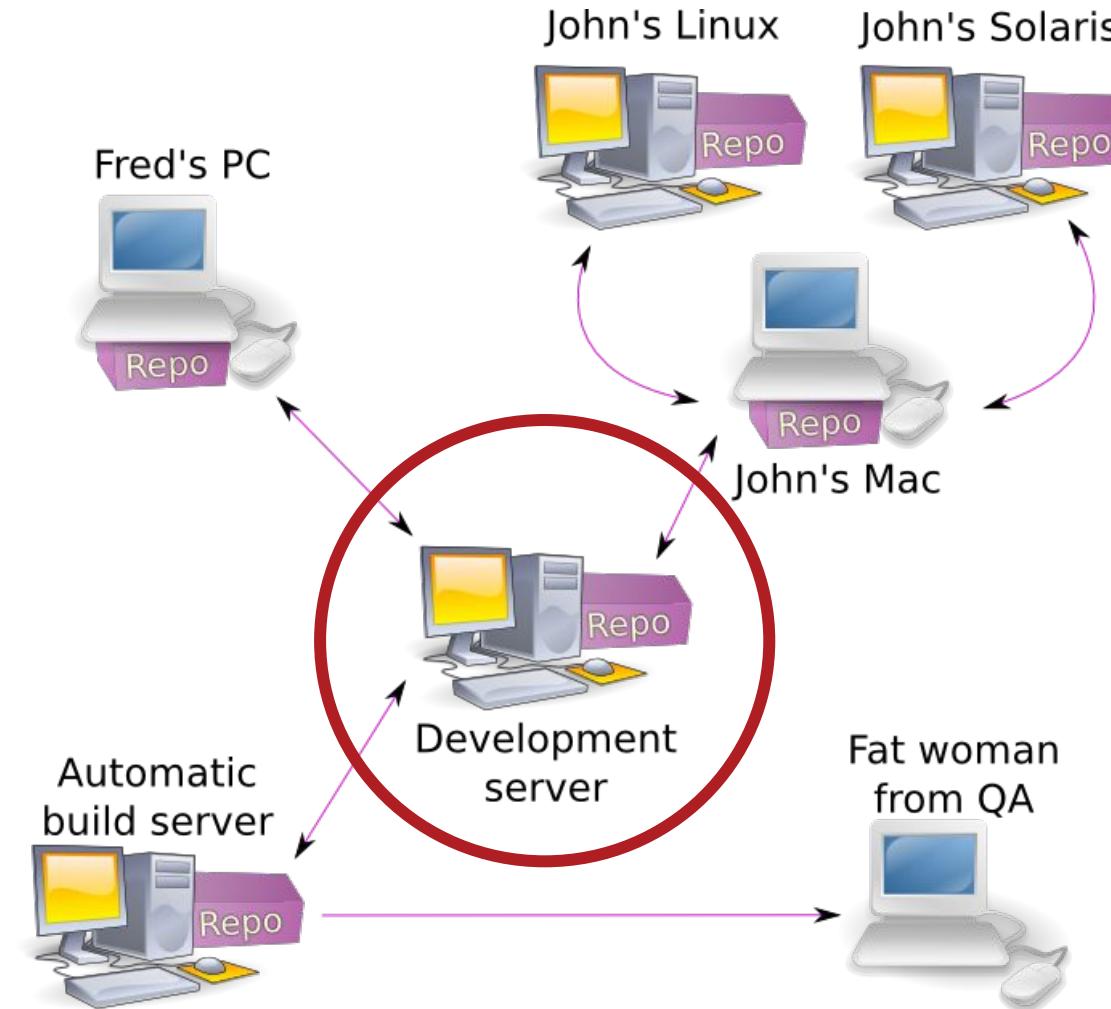
Czyli - nikogo nie interesuje w wiadomości commita że dodaliście klasę `UserService` w Javie lub `user-list` w CSS.

Interesuje ich **po co** to zrobiliście.

# Zdalne repozytorium



# Zdalne repozytorium



# Zdalne repozytorium

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



# Popularne usługi hostujące repozytoria



# GitHub to usługa



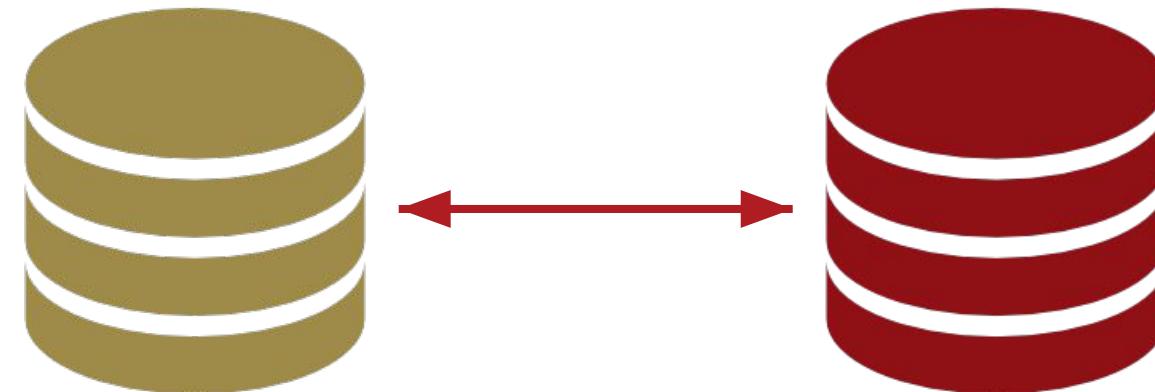
git



github  
SOCIAL CODING

# Jak pracować z zdalnym repozytorium?

- Nasze lokalne repozytorium musi znać adres zdalnego repozytorium
- Możemy mieć nawet wiele repozytoriów podpiętych



# origin

Domyślna nazwa  
zdalnego repozytorium.

Takie centralne  
repozytorium.



| **git push**

| Wyślijmy kod na zdalne repo

```
git push origin <branch>
```

Wysyłamy zmiany na remote'a

*origin.*

Synchronizacja. Wysyłamy całe  
**commity**



# Ćwiczenie

## Wrzuć swoje repo na GitHub

Po tym ćwiczeniu nie będziemy już używać tego projektu.

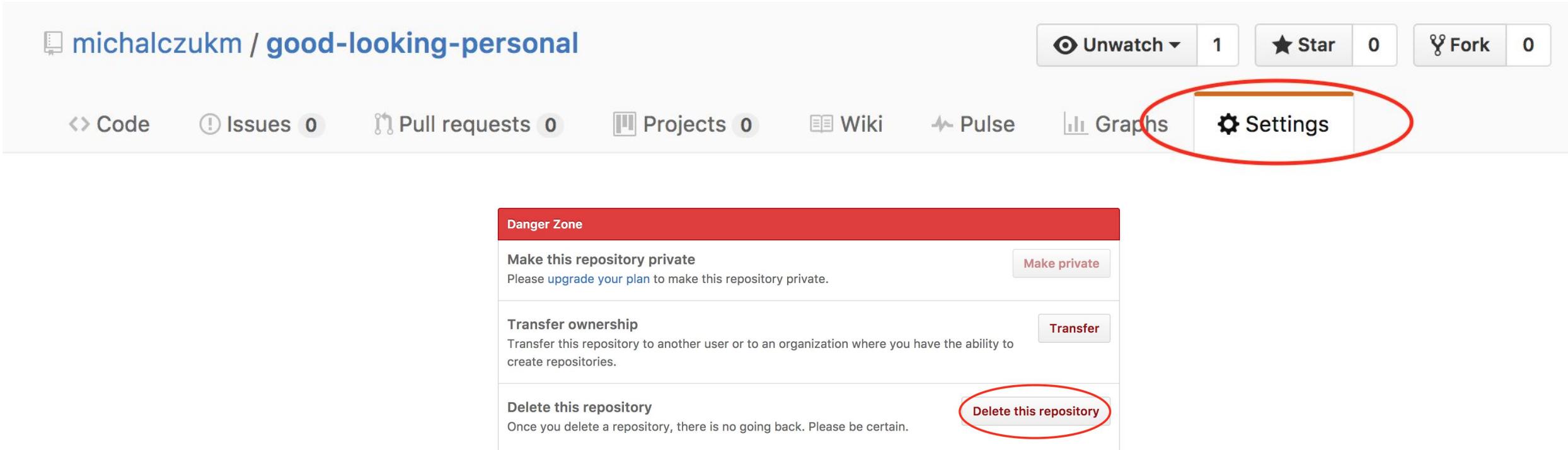
Możecie go usunąć z dysku.



# Ćwiczenie

## Wrzuć swoje repo na GitHub

Po tym ćwiczeniu nie będziemy już używać tego repozytorium. Możecie je usunąć z GitHub'a.



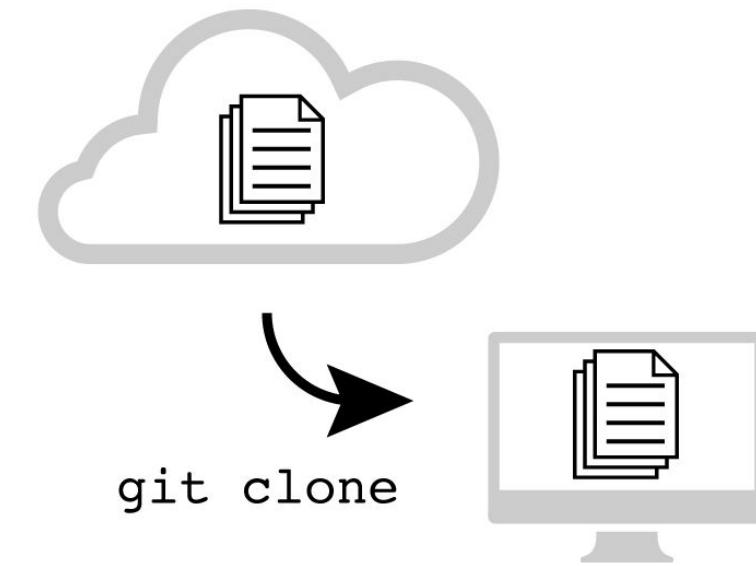
# git clone

```
git clone <url>
```

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

np.:

```
git clone https://foo.git
```



# Ćwiczenie

## Zrób clone mojego repozytorium

Zacznijmy pracować nad jedną bazą kodu.

URL repozytorium - dla przypomnienia na slacku



# git fetch

## pobierz zmiany

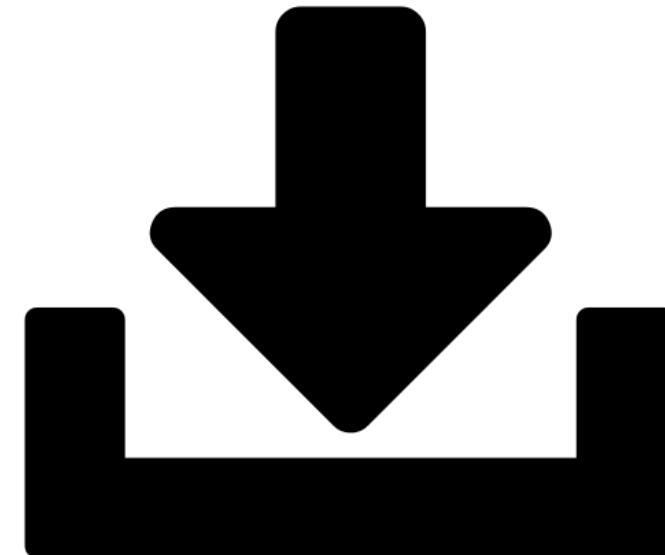
git fetch

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

Domyślnie pobiera z **origin**.

**Nie włącza tych zmian** do naszego lokalnego repozytorium.

Tylko pobiera je na dysk.



**git pull**  
pobierz i włącz zmiany

git pull

Włącza commity z zdalnego  
repozytorium do naszego lokalnego  
repo.

**Na aktualnym branchu.**

Od razu robi fetch pod spodem.



# Ćwiczenie fetch & pull



**git pull  
pod spodem**

git fetch <remote>

git merge origin/<current-branch>

// a konkretnie: git merge FETCH\_HEAD

# Gdzie mieszka git?

Gdzie są pliki GIT naszego lokalnego repo



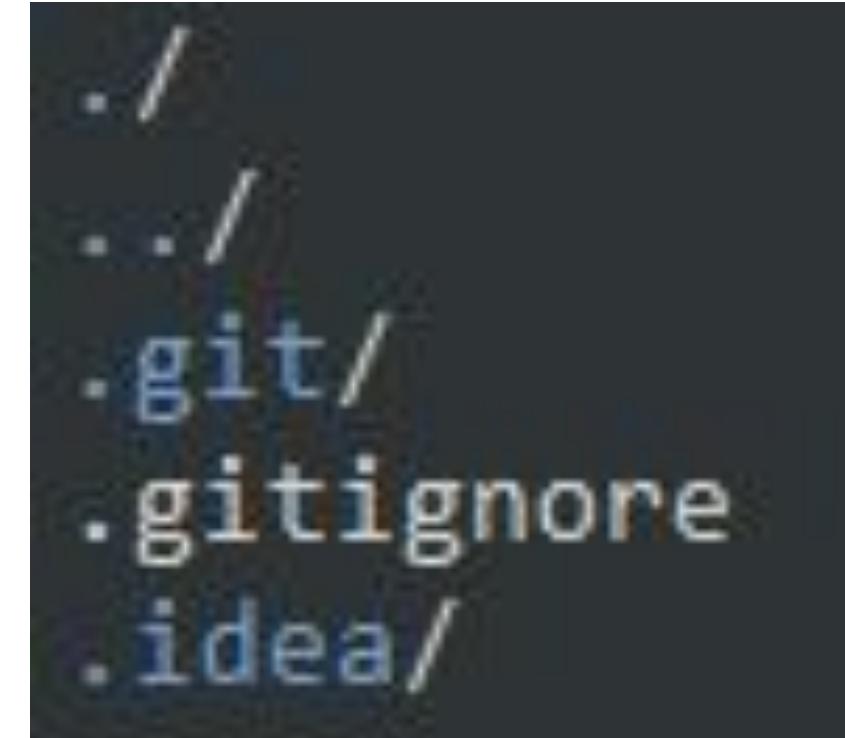
## Pliki repozytorium Ukryty folder .git

Wszystkie informacje o twoim  
repozytorium to pliki znajdujące  
się w **ukrytym folderze**

.git

W root'cie projektu.

Nie usuwaj tego folderu.



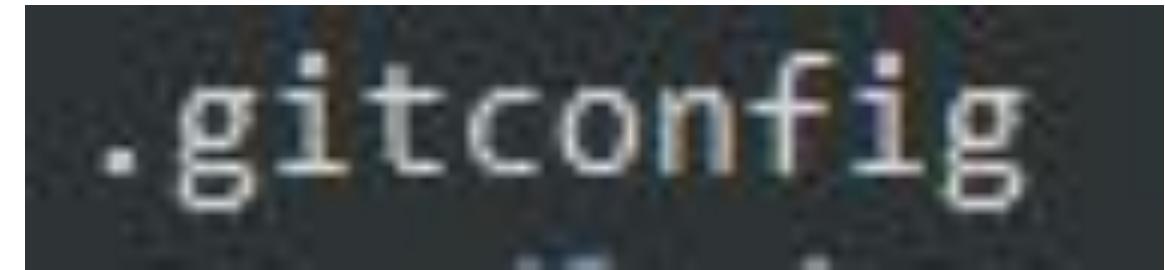
# Ustawienia użytkownika

Wszystko znajdziesz w pliku

~ / .gitconfig

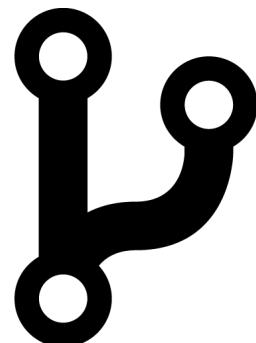
~ - katalog użytkownika

Niezależnie od systemu  
operacyjnego.



# Branche

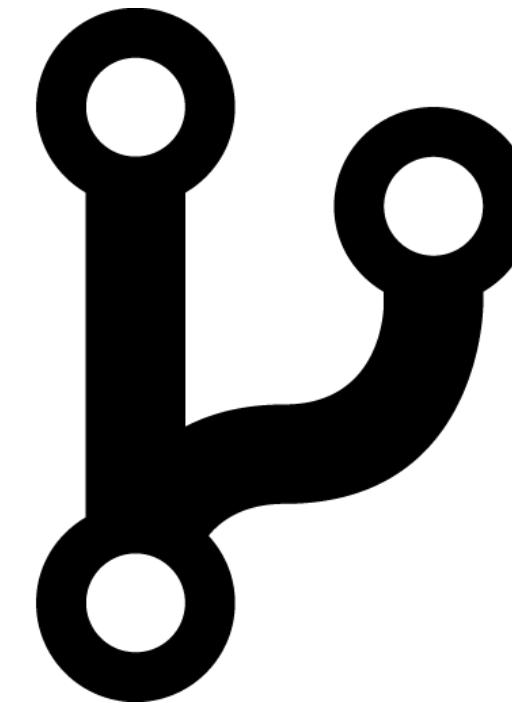
Czym jest branch, po co nam branche?



# Branch (gałąź)

Branch to **wskaźnik** na commit.

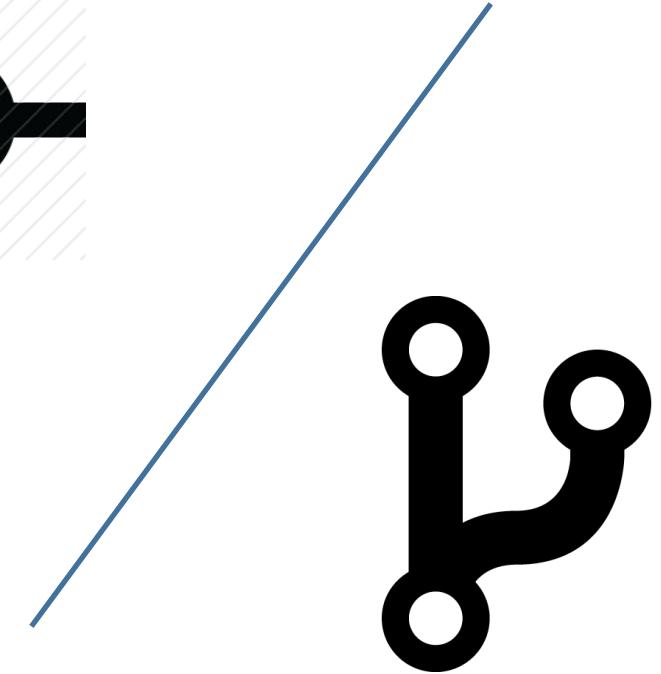
Można go zawsze  
dodać/usunąć/zmienić.



# Community a branche

Community są przechowywane  
(persistent)

Branche są płynne, to wskaźniki na  
community.



# Community a branche



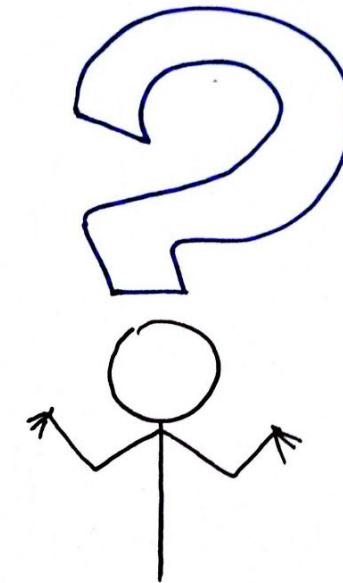
**Repozytorium to książka.**

**Community to kartki,** przedstawiają po kolei historię.

**Branche to zakładki.** Oznaczamy nimi miejsce w historii.

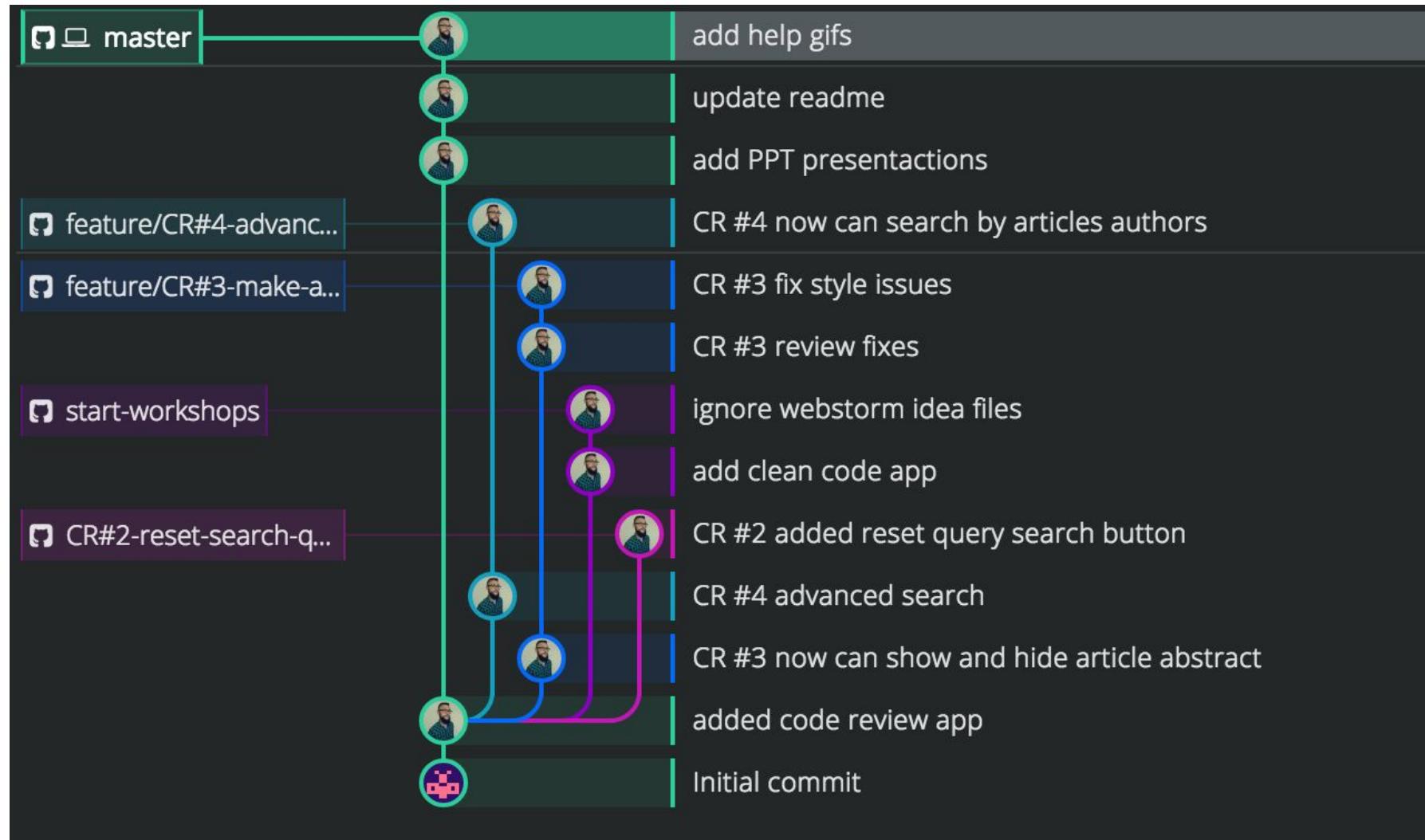
# Po co nam branche?

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



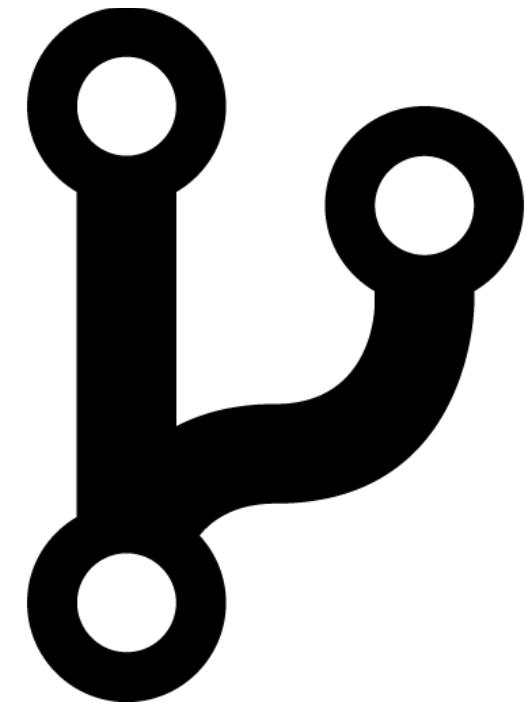
# Rozpoczęta praca nad wieloma zadaniami

## Przecież nie pracujesz sam/sama



# Co oznacza, że “jestem na branchu”

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



# Checkout

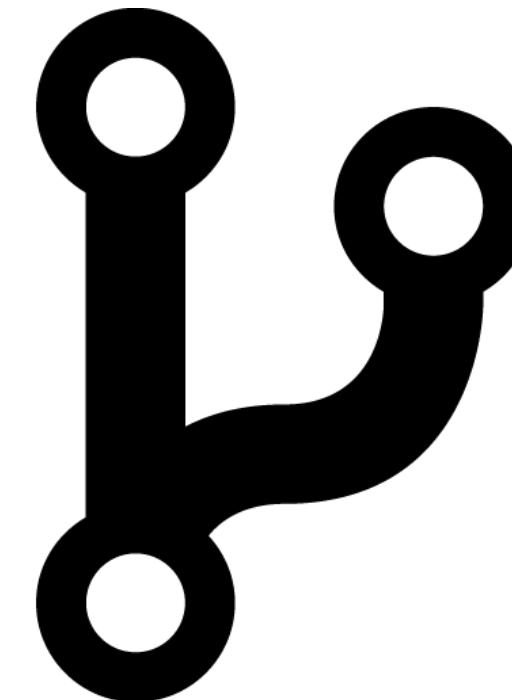
```
git checkout <branch>
```

Przejście/Przełączenie  
na brancha.

**Wpisz**

```
git status
```

i sprawdź gdzie jesteś



# A konkretne obiekt HEAD wskazuje na brancha

The screenshot shows a GitHub repository interface. On the left, a list of commits is displayed:

- update README.md (origin & master, Michal Mi, 27/05/2017 03:43)
- 11 fetch stickers with effects (origin/11, Michal Mi, 27/05/2017 03:15)
- 10-stickers effect (origin/10, Michal Mi, 27/05/2017 03:02)
- 9-blank stickers effect (origin/9, Michal Mi, 27/05/2017 02:54)
- 8-cart.component using isEmpty quer (origin/8, Michal Mi, 27/05/2017 02:50)
- 7-tshirts adds to cart (origin/7, Michal Mi, 27/05/2017 02:38)
- 6-blank tshirts module (origin/6, Michal Mi, 27/05/2017 02:22)
- 5 app.component counter by store-ql (origin/5, Michal Mi, 27/05/2017 02:14)
- 4-app.component counter from store (origin/4, Michal Mi, 27/05/2017 02:12)

On the right, a detailed view of the commit "update README.md" is shown:

C:\Workspaces\github\ngsummit\_angular-meets-redux 1 file

README.md

update README.md

9bf56b4 Michal Michalczuk <michalczukm@gmail.com> on 27/05/2017 at 03:43

committed on 27/05/2017 at 12:11

HEAD master origin/master

In 3 branches: HEAD, master, origin/master

# Commity a branche



**Repozytorium to książka.**

**Commity to kartki,** przedstawiają po kolei historię.

**Branche to zakładki.** Oznaczamy nimi miejsce w historii.

**HEAD** to miejsce w którym **otworzyłeś(a)ś** książkę.

# Ćwiczenie branching

Pojawiają się nowe wymagania do naszej aplikacji.

Zaimplementujmy je na oddzielnych branchach.

**Twoje nazwy branchy:**

ImieNaz\_master

ImieNaz\_feature/GIT-x

**Uwaga:**

Implementacja każdego z wymagań powinna być w 2-3 commitach.

Aby nasze drzewko ładnie wyglądało.



**Pro tip:**

Przed przełączeniem na innego brancha - upewnij się, że **nie masz żadnych zmian** w local changes/workspace

# Ćwiczenie

## zajrzyjmy w historię zmian

Po naniesieniu zmian - spójrzmy raz jeszcze w historię.

git annotate

git blame



# Widzieliśmy obiek HEAD obiekt HEAD wskazuje na aktualnego brancha

The screenshot shows a GitHub commit history and a file viewer side-by-side.

**Commit History:**

- update README.md (origin & master) Michal Mi 27/05/2017 03:43
- 11 fetch stickers with effects (origin/11) Michal Mi 27/05/2017 03:15
- 10-stickers effect (origin/10) Michal Mi 27/05/2017 03:02
- 9-blank stickers effect (origin/9) Michal Mi 27/05/2017 02:54
- 8-cart.component using isEmpty quer (origin/8) Michal Mi 27/05/2017 02:50
- 7-tshirts adds to cart (origin/7) Michal Mi 27/05/2017 02:38
- 6-blank tshirts module (origin/6) Michal Mi 27/05/2017 02:22
- 5 app.component counter by store-ql (origin/5) Michal Mi 27/05/2017 02:14
- 4-app.component counter from store (origin/4) Michal Mi 27/05/2017 02:12

**File Viewer:**

C:\Workspaces\github\ngsummit\_angular-meets-redux 1 file

README.md

```
update README.md

9bf56b4 Michal Michalczuk <michalczukm@gmail.com> on 27/05/2017
at 03:43
committed on 27/05/2017 at 12:11
```

HEAD master origin/master

In 3 branches: HEAD, master, origin/master

# Ale gdy HEAD nie wskazuje na brancha detached HEAD

The screenshot shows a Git commit history and a file tree. The commit history on the left lists several commits, with the third commit, '10-stickers effect', highlighted in blue and circled in red. This commit has a yellow key icon next to it. To its right, the file tree shows a directory structure under 'C:\Workspaces\github\ngsummit\_angular-meets-redux\src\app'. The 'index.ts' file is visible. Below the file tree, the commit message '10-stickers effect' is shown, followed by the commit hash 'f94326c' and the author's details: 'Michal Michalczuk <michalczukm@gmail.com> on 27/05/2017 at 03:02'. A second red circle highlights the yellow key icon next to the word 'HEAD' in the commit message area.

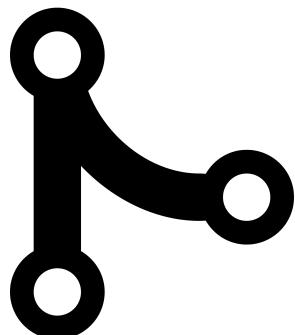
Commit	Branch	Author	Date
update README.md	origin & master	Michal Mi	27/05/2017 03:43
11 fetch stickers with effects	origin/11	Michal Mi	27/05/2017 03:15
10-stickers effect	! origin/10	Michal Mi	27/05/2017 03:02
9-blank stickers effect	origin/9	Michal Mi	27/05/2017 02:54
8-cart.component using isEmpty quer	origin/8	Michal Mi	27/05/2017 02:50
7-tshirts adds to cart	origin/7	Michal Mi	27/05/2017 02:38
6-blank tshirts module	origin/6	Michal Mi	27/05/2017 02:22
5 app.component counter by store-qu	origin/5	Michal Mi	27/05/2017 02:14
4-app.component counter from store	origin/4	Michal Mi	27/05/2017 02:12

GIT limbo.

Nie commituj. Załącz w tym miejscu brancha jeśli musisz.

# Merge

Łączymy wyniki naszej pracy



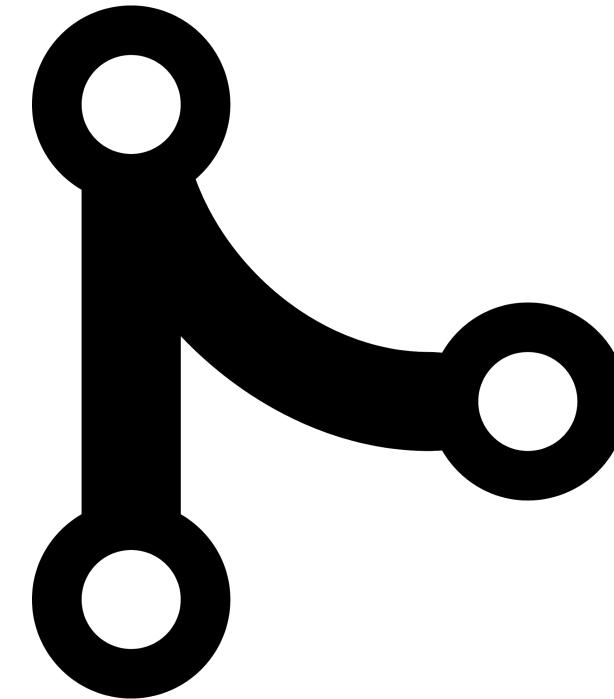
# Merge

```
git merge <branch_name>
```

Połączenie/zmieszanie ze sobą  
branchy.

Łączenie naszego kodu  
w całość

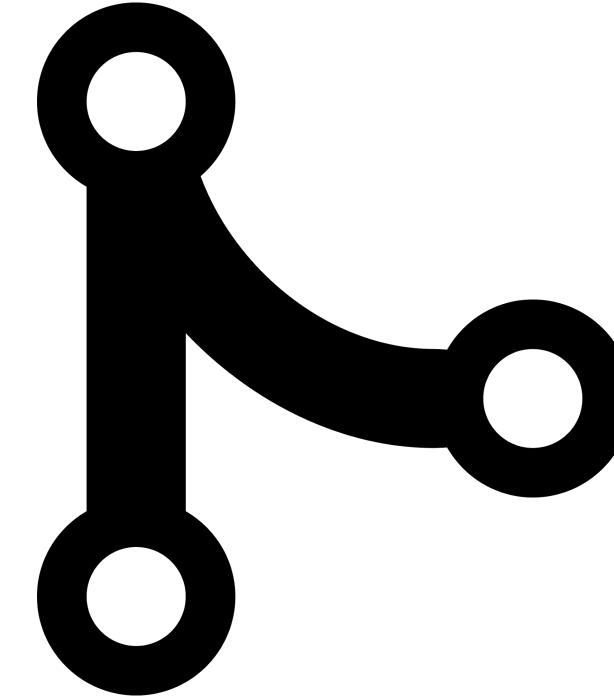
Ilu rodziców ma merge?



# Merge

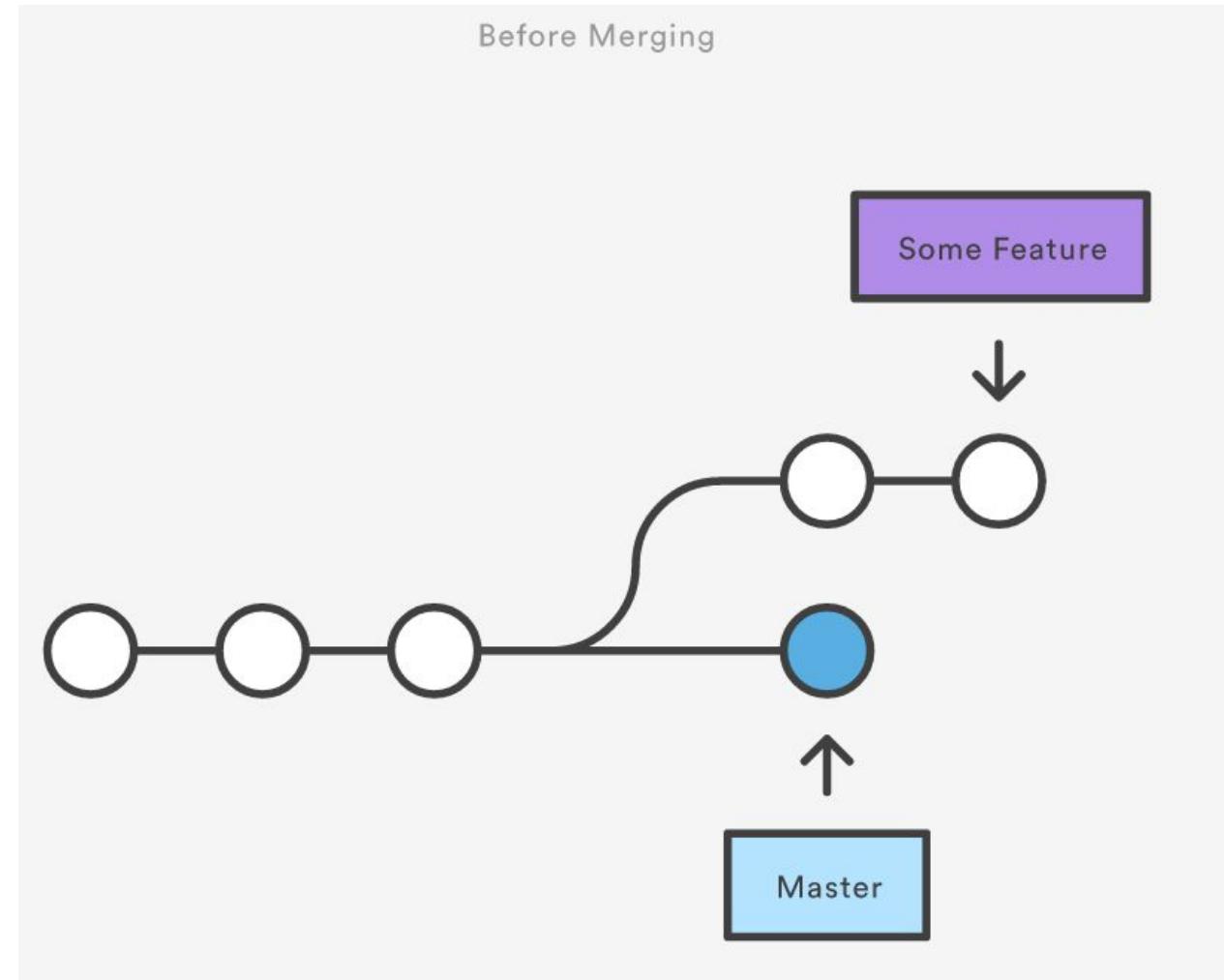
```
git merge <branch_name>
```

Mergujemy *branch\_name* do  
brancha na którym **aktualnie**  
jesteśmy.



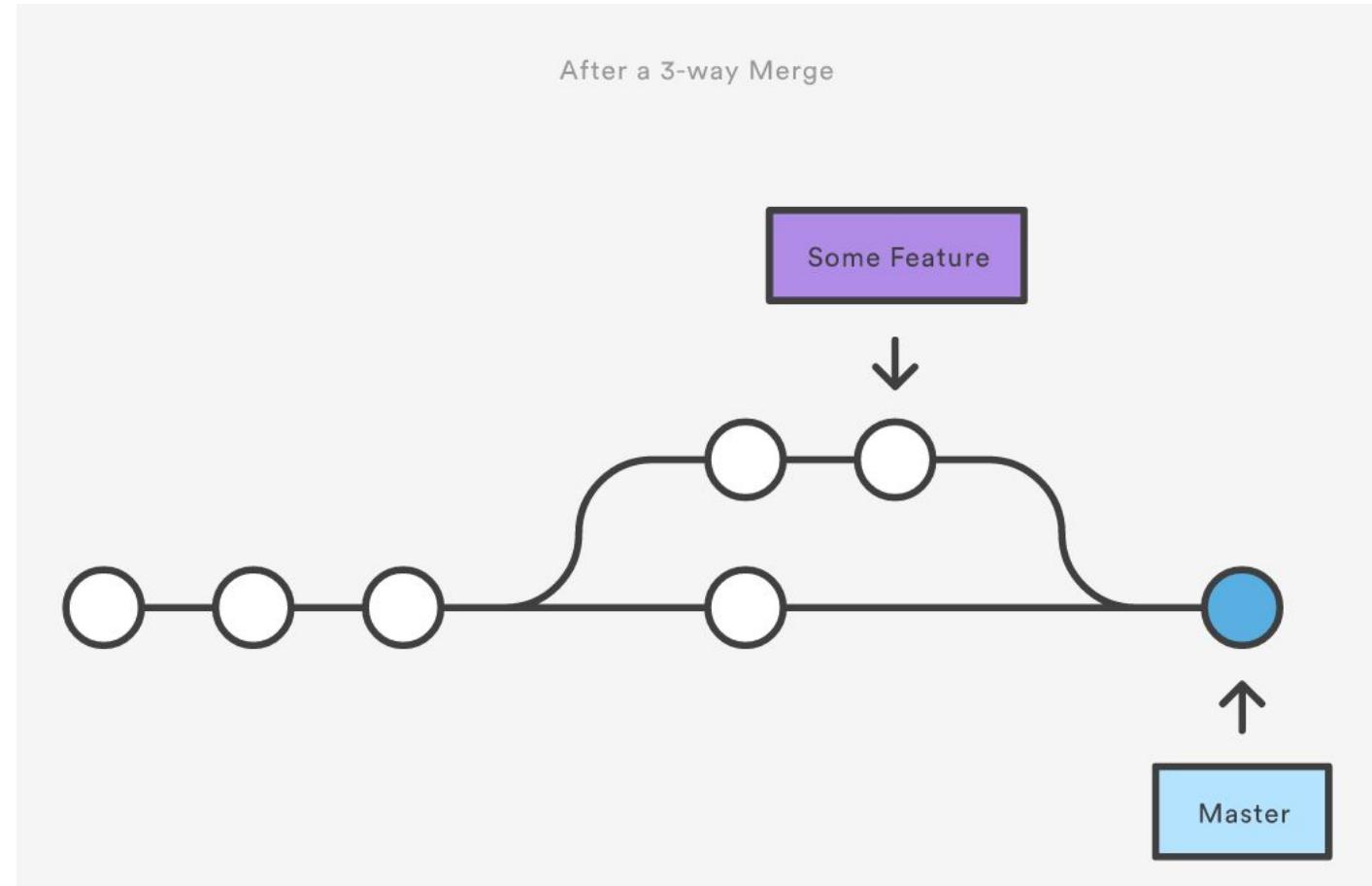
# 3-way merge (brzuszek)

1/2



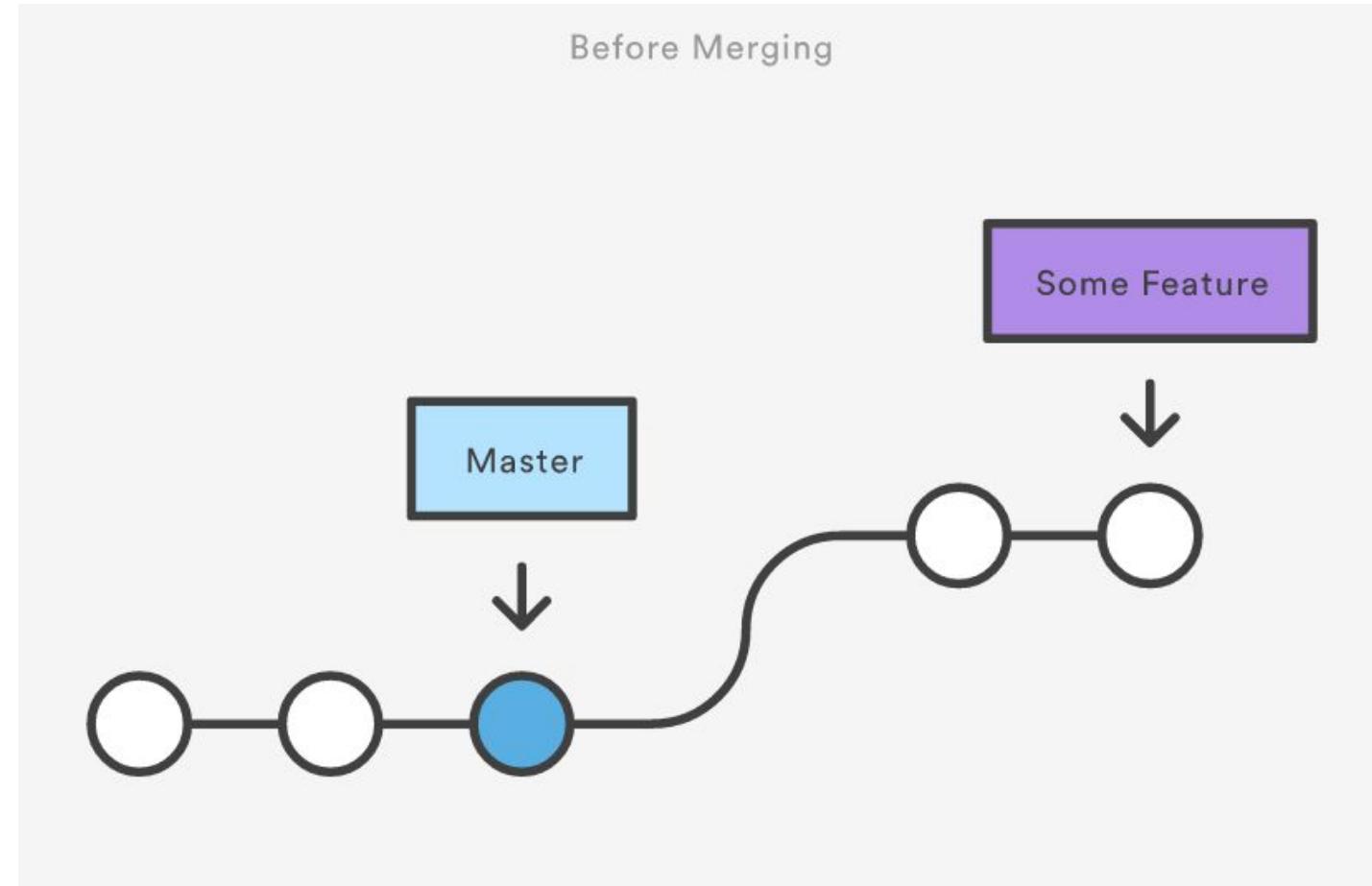
# 3-way merge (brzuszek)

2/2



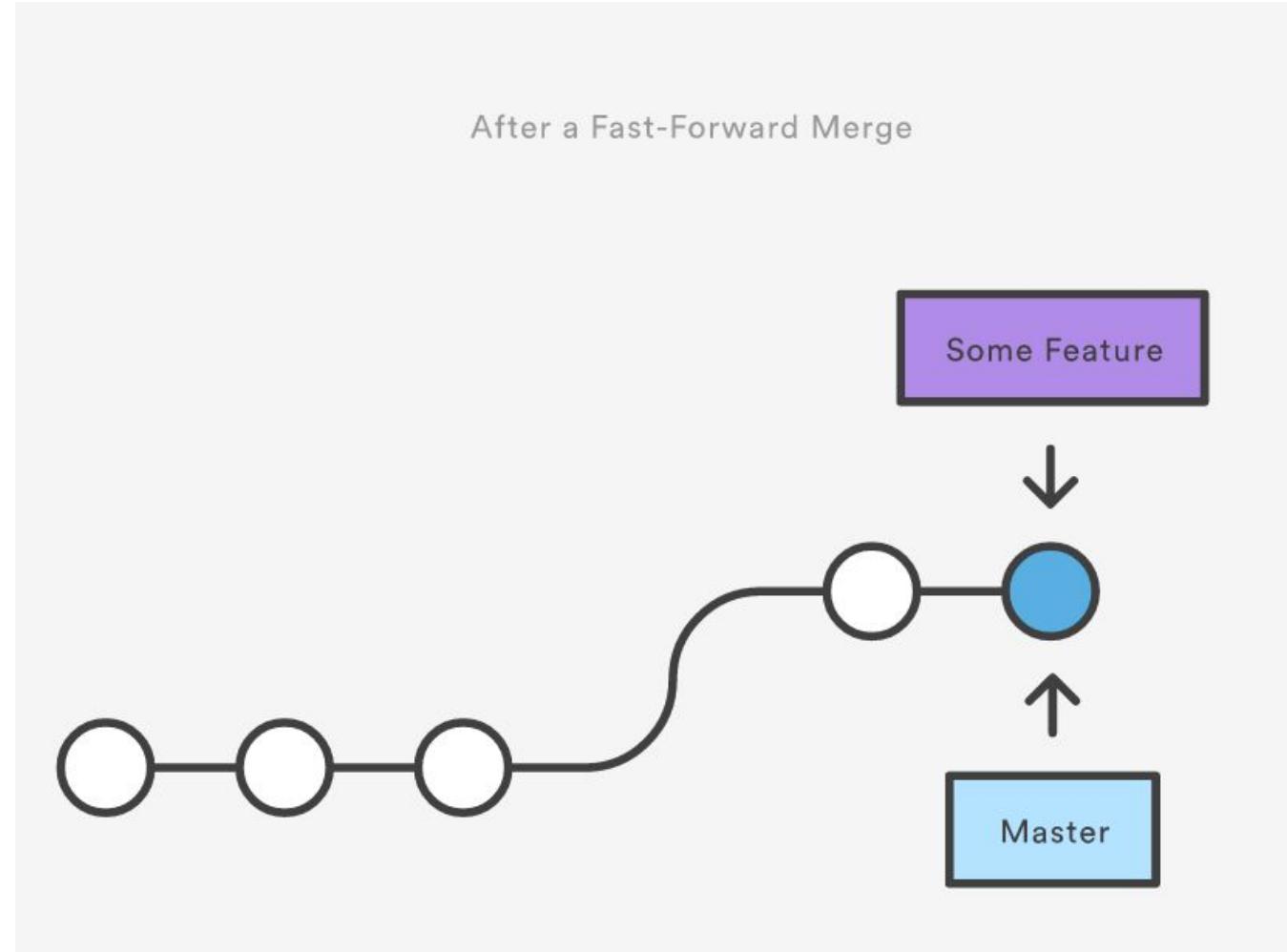
# Fast-forward merge (przeniesienie wskaźnika)

1/2



# Fast-forward merge (przeniesienie wskaźnika)

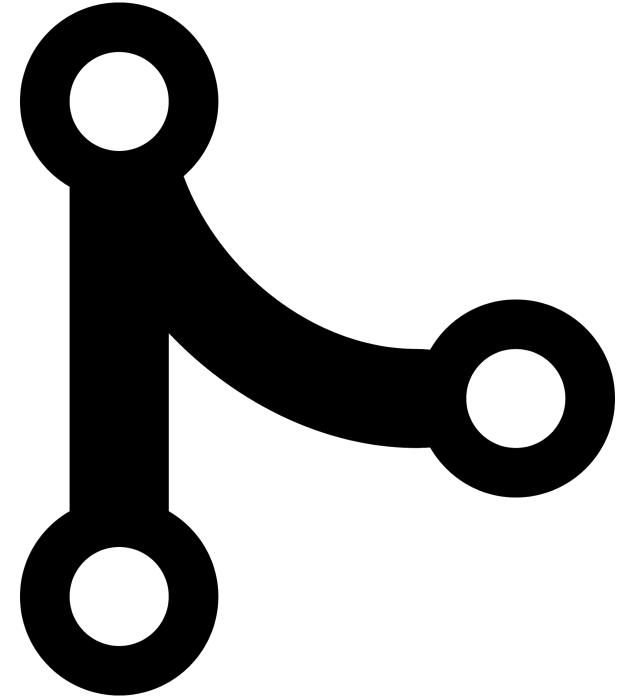
2/2



# 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 w GIT.

Nie należy się ich bać :)

Ani lekceważyć.

Czasem wymagają konsultacji z autorem  
innych zmian.



# Ćwiczenie

## Mergujemy nasze feature'y

Mergujemy po kolejni nasze feature'y z poprzedniego ćwiczenia do brancha master.

1. ImieNaz\_feature/GIT-1 ---> ImieNaz\_master
2. ImieNaz\_feature/GIT-2 ---> ImieNaz\_master
3. ImieNaz\_feature/GIT-3 ---> ImieNaz\_master

Jak się trafi **konflikt** - daj znać!

**Twoje nazwy branchy:**

ImieNaz\_master

ImieNaz\_feature/GIT-x



# Ćwiczenie

## Rozwiążujemy konflikty

Mały testowy konflikt

Zmerguj:  
some-feature ---> master-will-be-conflict



## 'git pull' powraca

### Skąd się biorą konflikty przy pull?

```
git fetch <remote>
```

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

```
// a konkretnie: git merge FETCH_HEAD
```

# 'git pull' powraca

## Skąd się biorą konflikty przy pull?

Pod spodem jest merge.

A jeśli mergujemy kod -

może pojawić się konflikt.



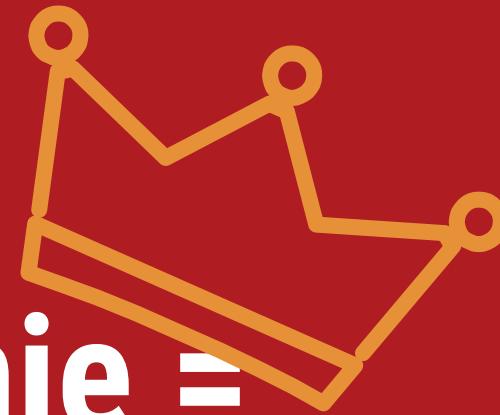
# Merge dobre praktyki

git pull

brancha do którego merujesz

Merguj się często.

Częste mergowanie = mniej konfliktów &  
prostsze konflikty



Częste mergowanie =  
mniej konfliktów & prostsze konflikty

Pro tip.



# Ćwiczenie grupowe

## Przedstaw się!

Plik jest w formacie:

```
<?xml version="1.0" encoding="utf-8" ?>  
<team name="{your team name}">  
    <version>{your file version}</version>  
    <members>  
        <member name="{your name}" surname="{your surname}">{nick}</member>  
    </members>  
</team>
```

Przykładowe uzupełnienie pliku:

```
<?xml version="1.0" encoding="utf-8" ?>  
<team name="ISA-MM-1">  
    <version>3</version>  
    <members>  
        <member name="Michał" surname="Michałczuk">Michał</member>  
        <member name="Agata" surname="Luz">Aga</member>  
        <member name="Maciej" surname="Kowalski">Macias</member>  
    </members>  
</team>
```

Celem ćwiczenia jest zbudowanie całą drużyną pliku .xml który zawiera podstawowe informacje o was.

Każda drużyna uzupełnia jeden plik - odpowiednio `a-team.xml`, `b-team.xml` itd.

Na repozytorium znajduje się plik `example.xml`, który zawiera szablon.  
Ten plik nie może zniknąć z repozytorium!



# Ćwiczenie grupowe

## Przedstaw się!

Celem ćwiczenia jest zbudowanie całą drużyną pliku .xml który zawiera podstawowe informacje o was.

### Zadanie

Waszym zadaniem jest uzupełnić ten plik. Ale jest parę reguł:

- każdy może dodać tylko swoje dane - tj autor commita musi się pokrywać z dodaną osobą
- każda zmiana składu musi być w oddzielnym commicie
- jeśli plik się zmienia (poprzez commit) należy podnieść wersję pliku o 1 - na początku jest 1
- gotowy plik można wrzucić do brancha `master` tylko przez `Pull Request` na GitHub'ie !
- branch zespołu powinien się nazywać `hello-nazwa-zespolu`

### Część A

Wybierzcie czy chcecie pracować na jednym branchu, czy założyć wiele (per user) i wpiszcie członków drużyny wg powyższych reguł.

Które rozwiązanie jest dla was lepsze? Jakie problemy napotkaliście?

### Część B

Wystawcie \*\*jednego\*\* `Pull Requesta` do repozytorium, do brancha master z danymi waszej drużyny. Pull requesty sprawdzimy wspólnie na koniec i je zmergujemy.



# Git stash

Tymczasowe przechowywanie zmian



# git stash

## twój mały schowek

git stash

Chowamy zmiany do schowka.

git stash pop

Wyciągamy zmiany ze schowka.



# git stash

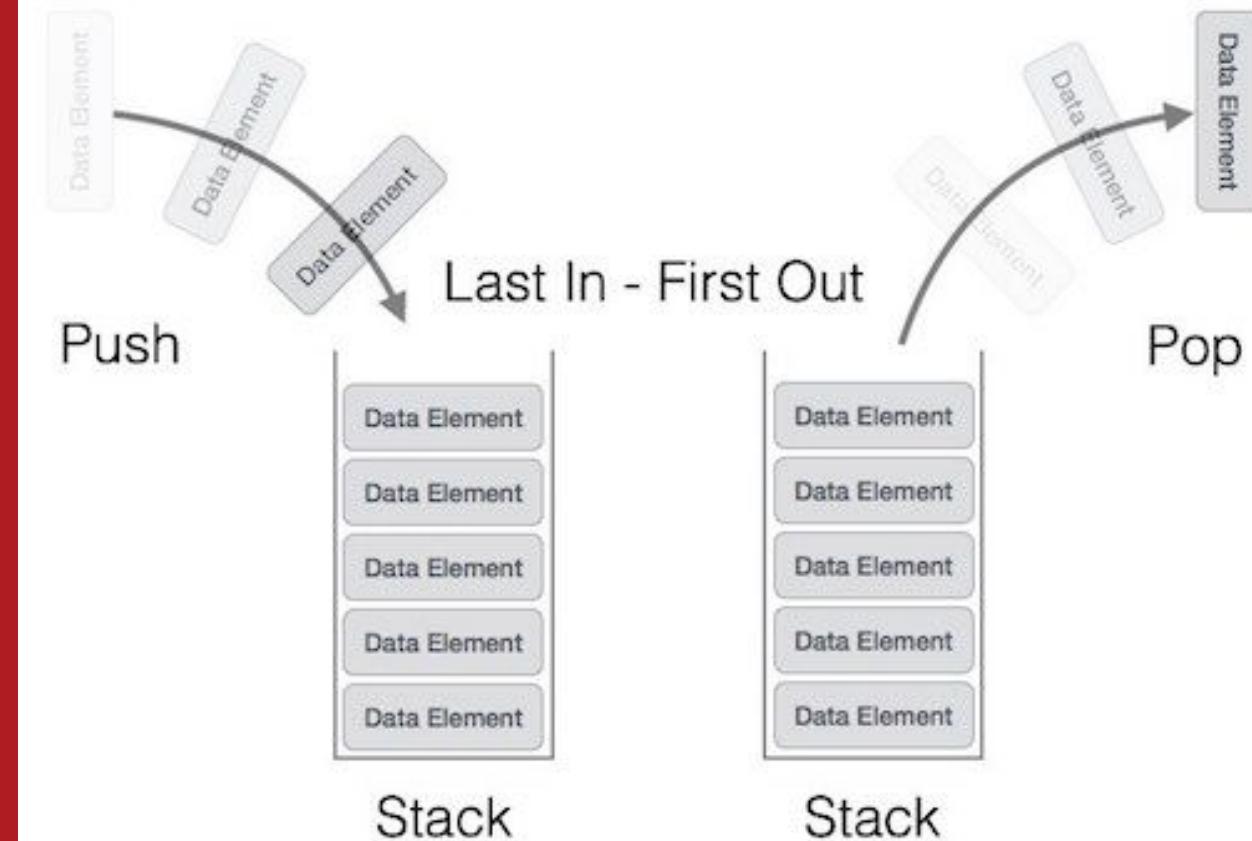
## schowek stos

Stash ma konstrukcję stosu.

Mogemy trzymać wiele zbiorów zmian ...

ale musimy je wkładać i zdejmować.

Jak na stosie.  
push-pop.



# git stash

## twój mały schowek

Kiedy się przydaje?

- mamy zmiany i chcemy przełączyć brancha
- chcemy chwilowo “ukryć nasze zmiany”
- ...



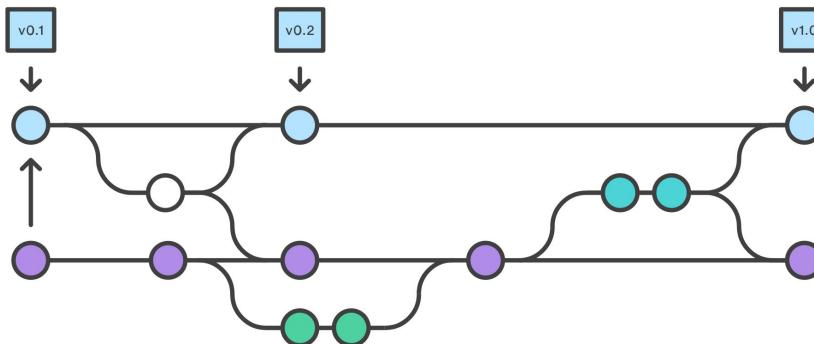
# Ćwiczenie

## Git stash - twój mały schowek



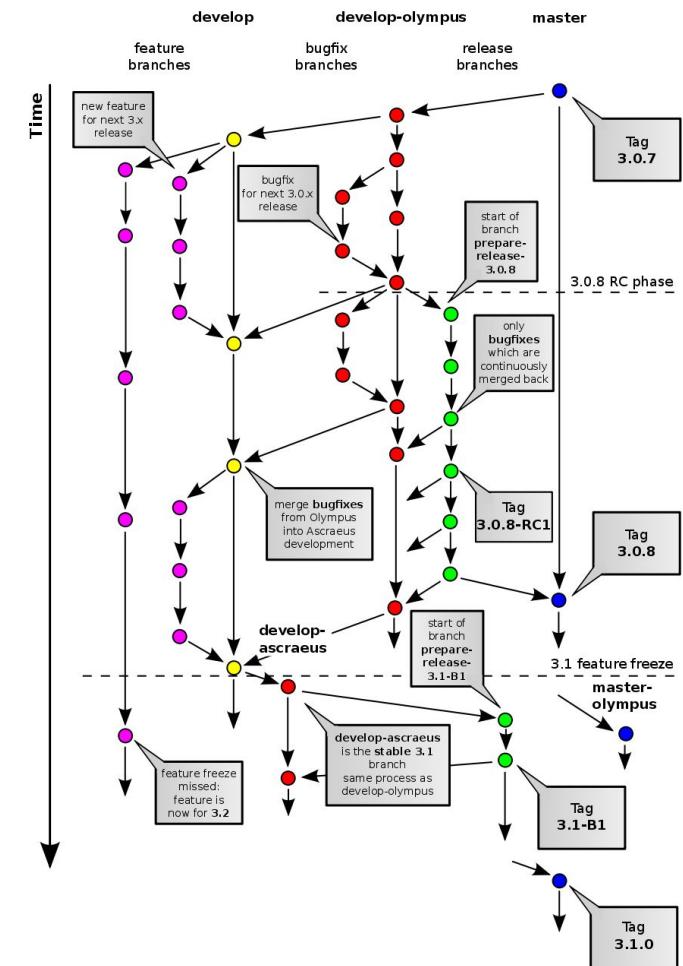
# git flow

Jak usystematyzować pracę z GIT



# Sposoby pracy z GIT

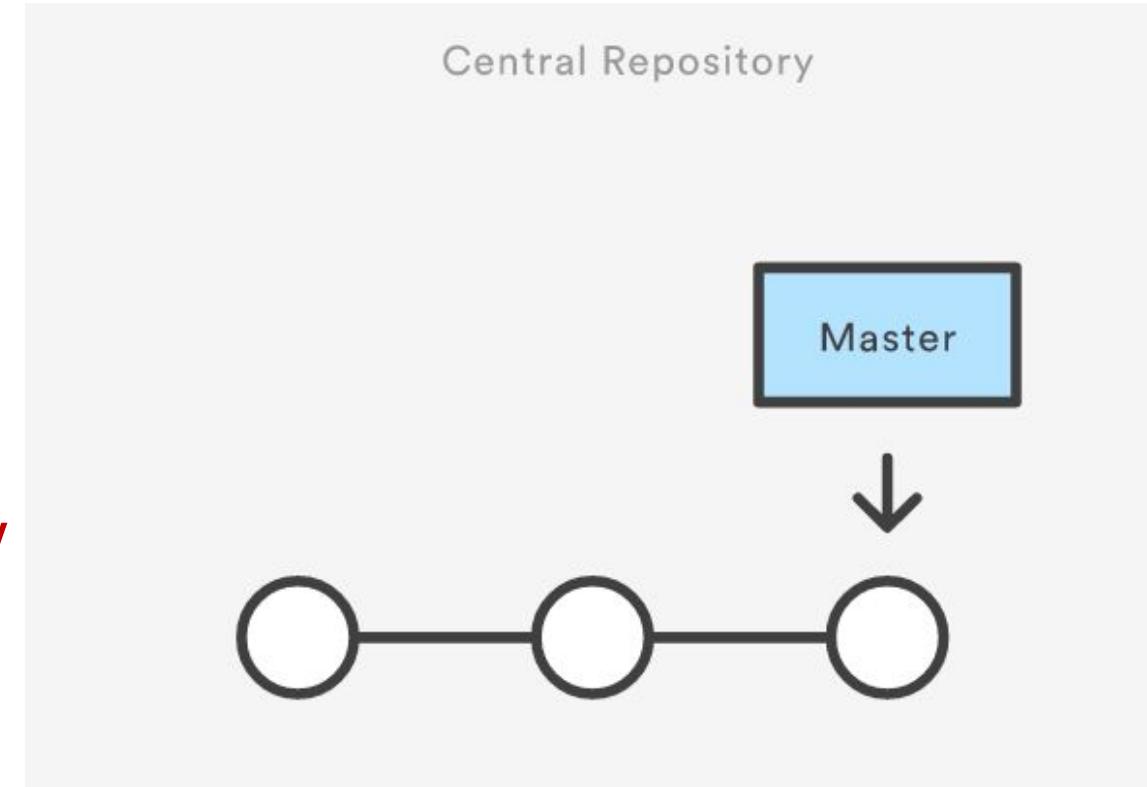
- git sam w sobie nie narzuca systemu pracy
- istnieje parę popularnych workflow



# Sposoby pracy z GIT

## Centralized flow

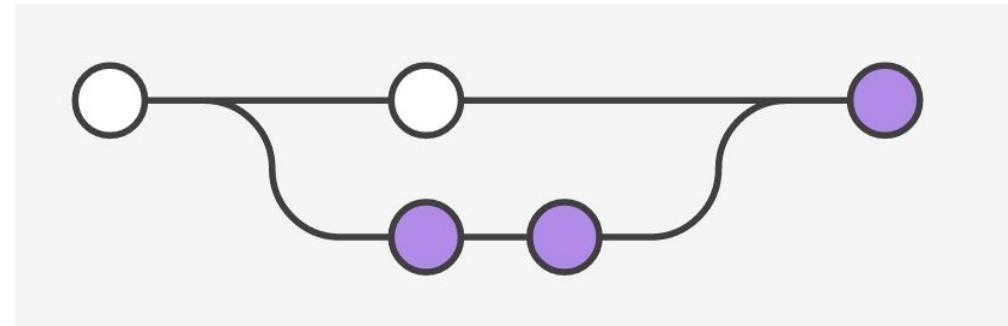
- Zbliżony do pracy z SVN
- Praca tylko na jednym branchu
- Dużo konfliktów
- Brak rozróżnienia na kod rozwojowy i kod produkcyjny
- Brak stabilności kodu (efekt)
- Ciężkie w koordynacji przy wielu osobach



# Sposoby pracy z GIT

## Feature branch flow

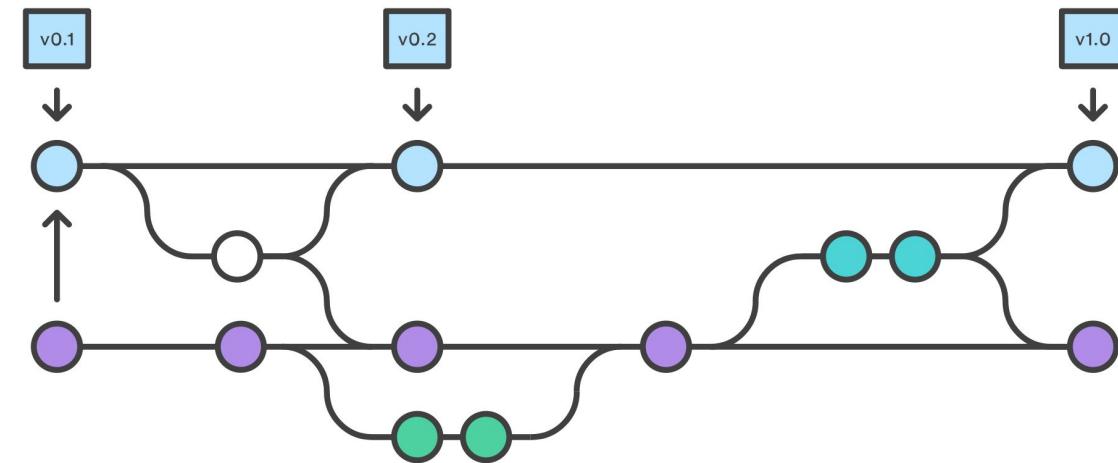
- Naturalny podział zadań
- Każdy feature / bug ma swojego brancha
- Rozdzielenie kodu rozwojowego dla feature-ów
- Wiele osób = Wiele branchów
- Możliwość weryfikacji kodu przed wejściem do master-a
- Dalej brak rozróżnienia na kod rozwojowy I kod produkcyjny



# Sposoby pracy z GIT

## Gitflow workflow

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

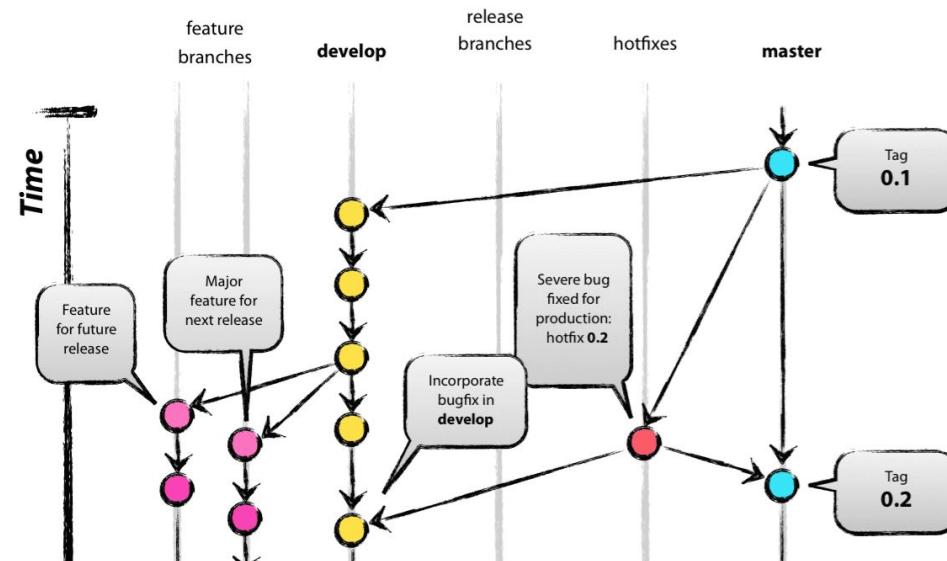


# A successful Git branching model



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.



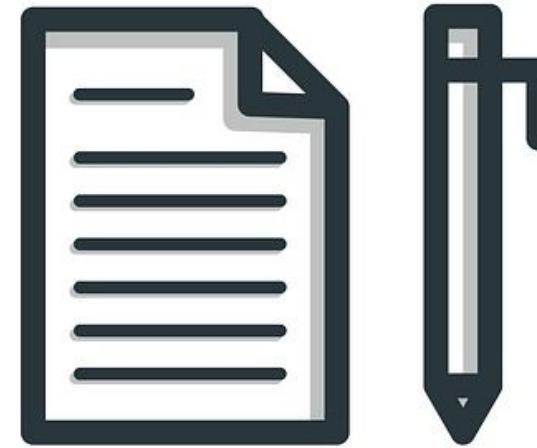
Oryginalny post który opisuje git flow znajdzicie  
<http://nvie.com/posts/a-successful-git-branching-model/>

# Gitflow workflow

## Jak to działa?

Konwencje. Kontrakt.

- 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

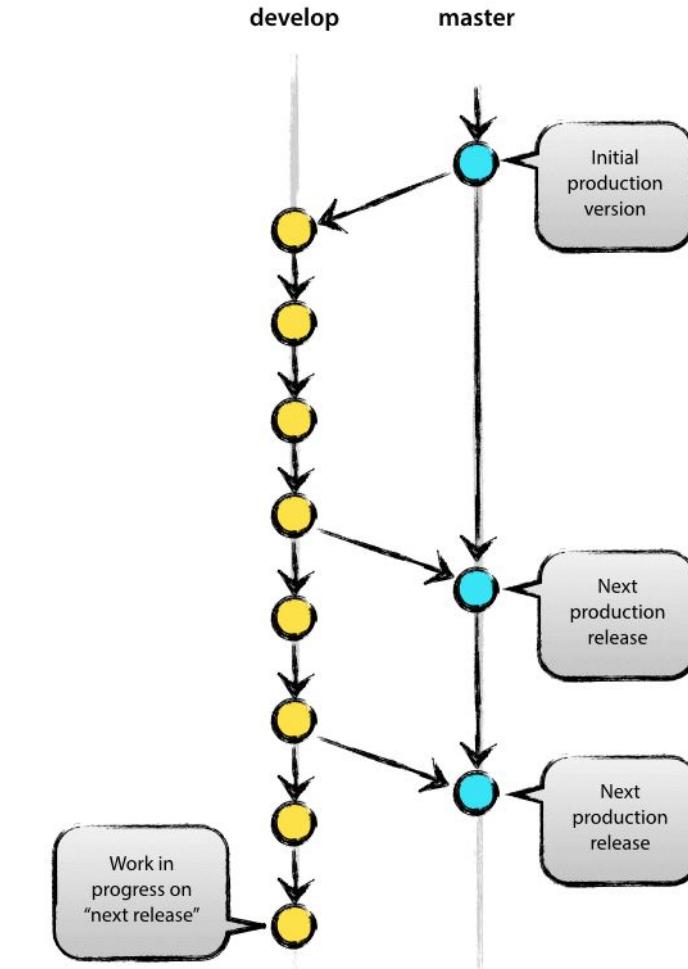
## Podstawowe branche

**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/>

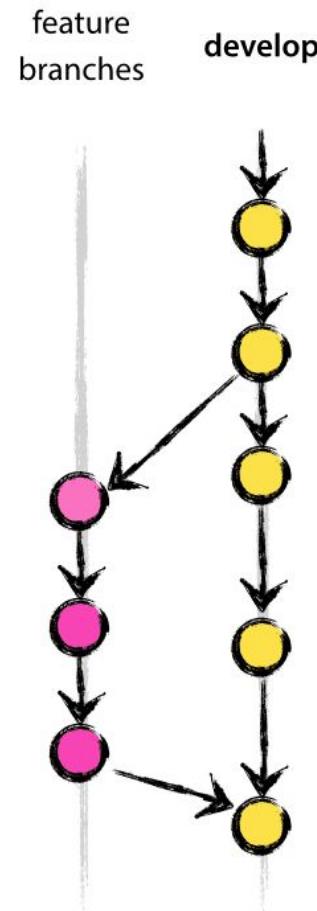
# Gitflow workflow

## Branche wspierające

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

# Dlaczego feature/\*?

Aby je pogrupovać/wyróżnić.



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

# Gitflow workflow

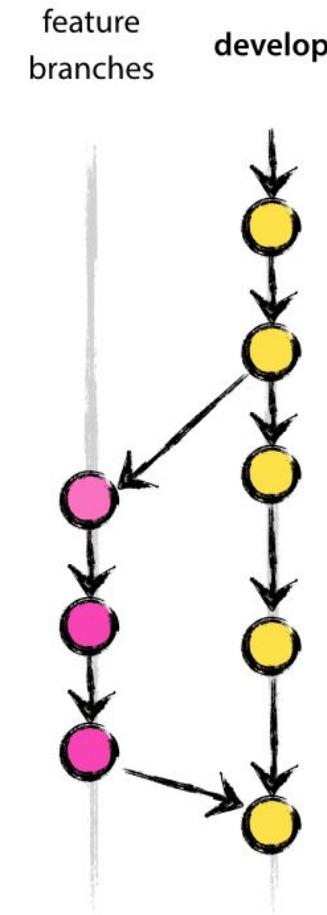
## Branche wspierające

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/>

# 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 wersji

Te bugfixy które robiliśmy na release branchu.

Przydadzą się też na **develop-ie**.

No to merge.



# Gitflow workflow

## Hotfix - kiedy płonie

Czasem zdarzy się **błąd**.

Czasem dosyć **krytyczny**.

Czasem wprowadzić poprawkę  
**od razu**.

Od tego mamy **hotfixy**.

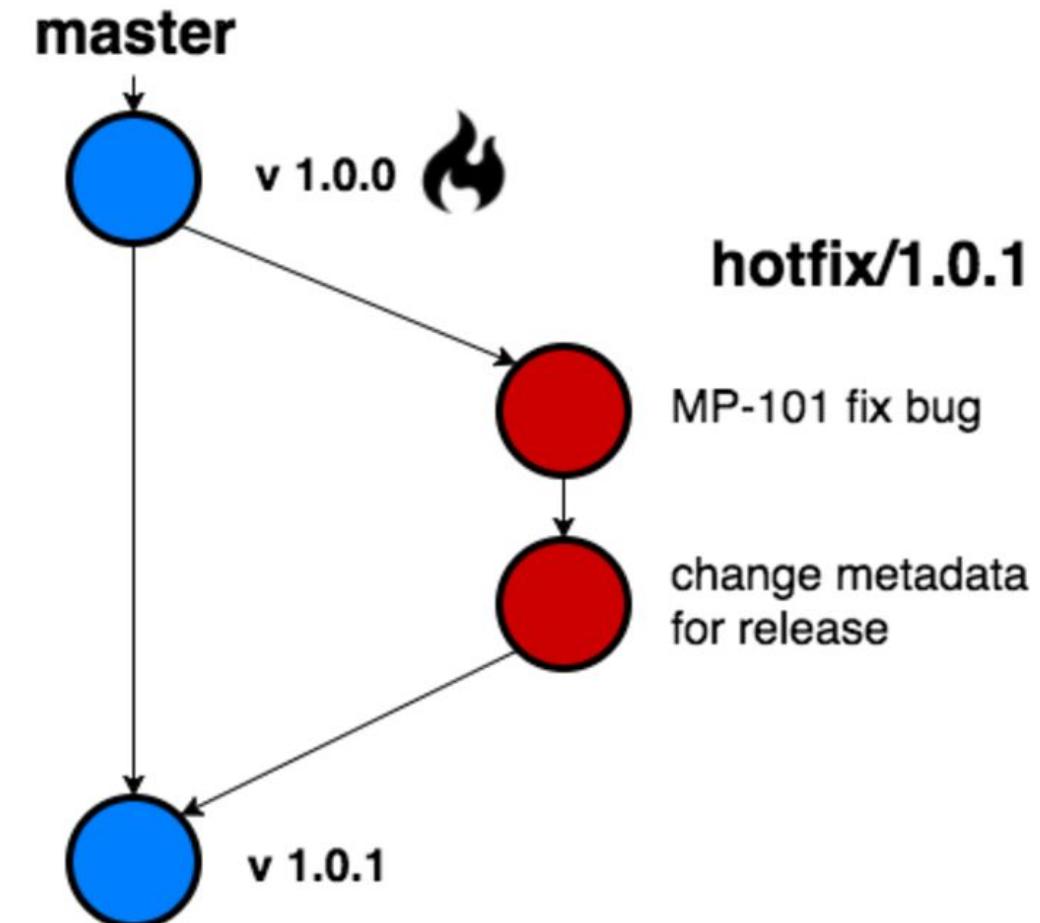


# Gitflow workflow

## Hotfix - kiedy płonie

Chcemy wprowadzić poprawkę bez wrzucania nowych feature-ów.

Zwłaszcza tych nie przetestowanych.

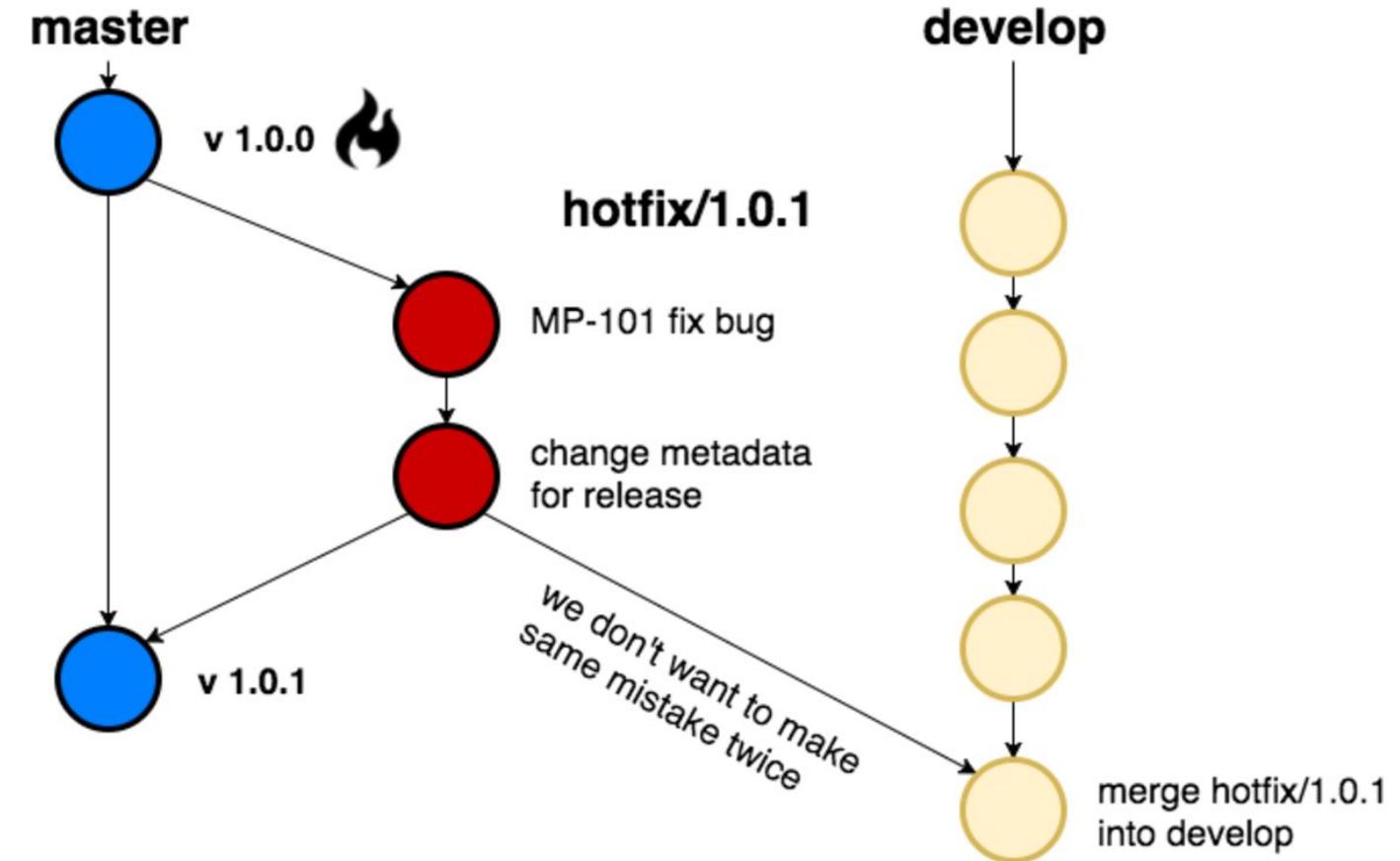


# Gitflow workflow

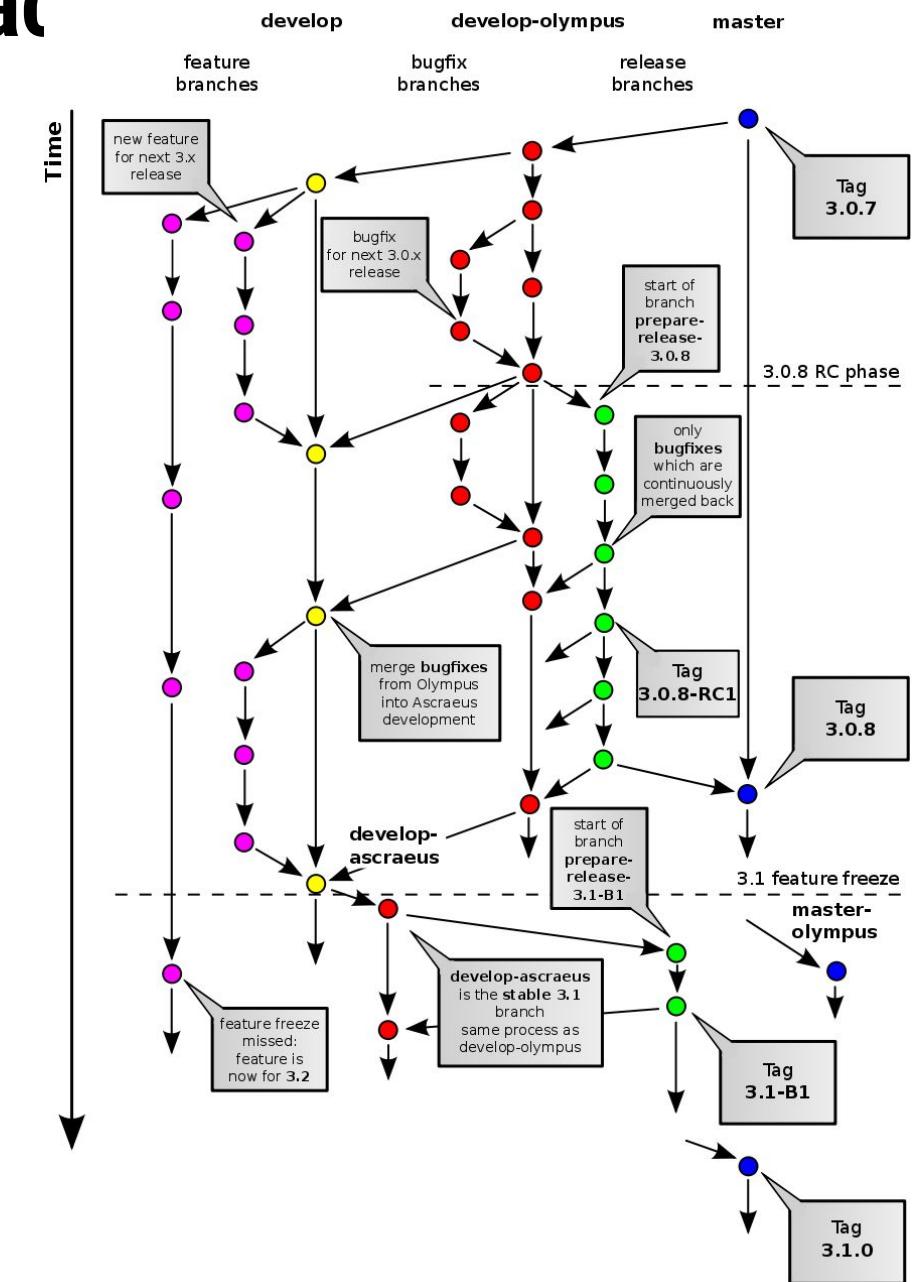
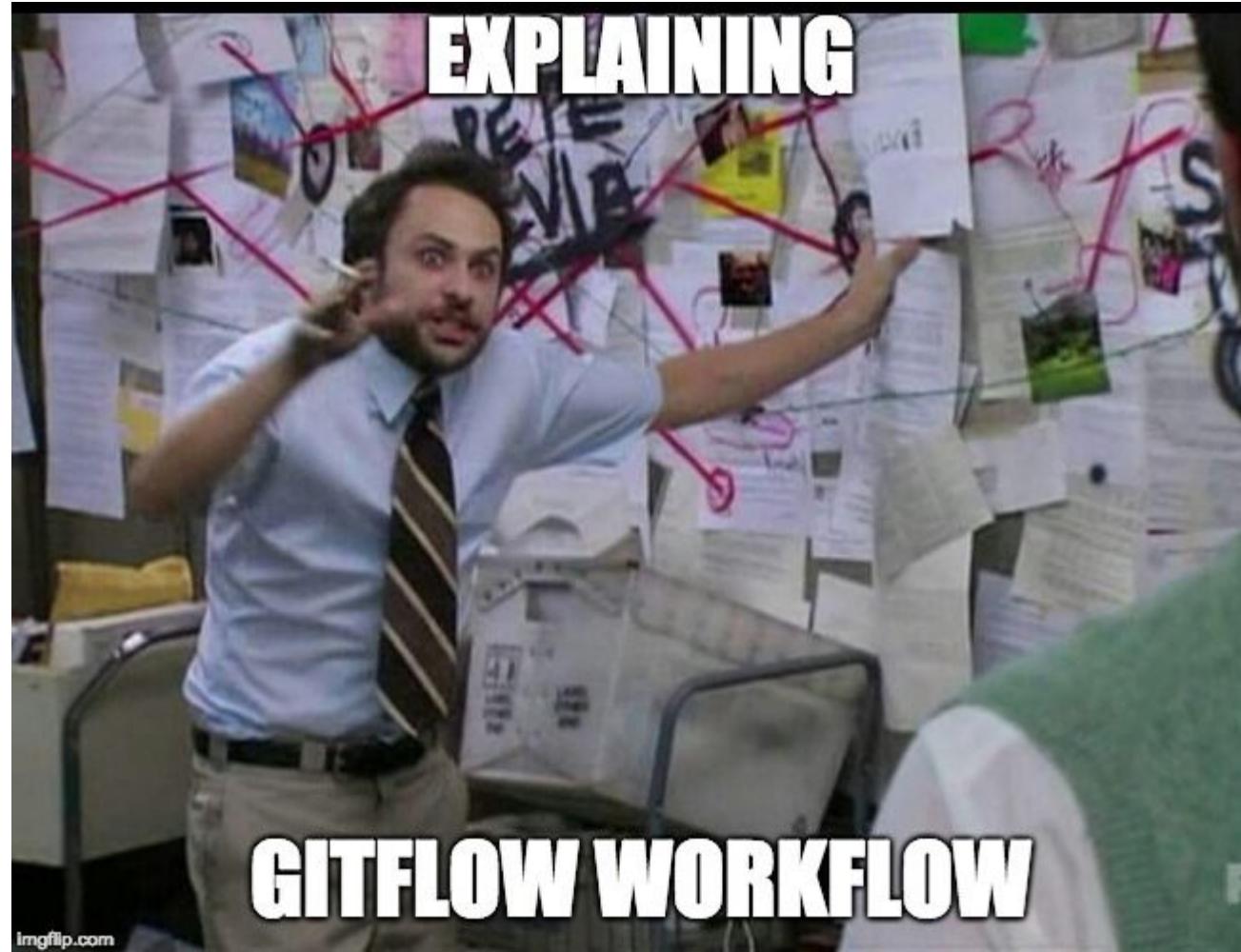
## Hotfix, po wydaniu

Warto nie naprawiać tego samego buga 2 razy.

Więc mergujemy hotfix-a do developa.

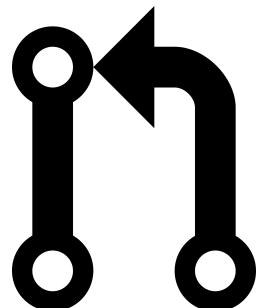


# Gitflow - byle nie przekombinować



# Pull requests

Krótko - co to jest?



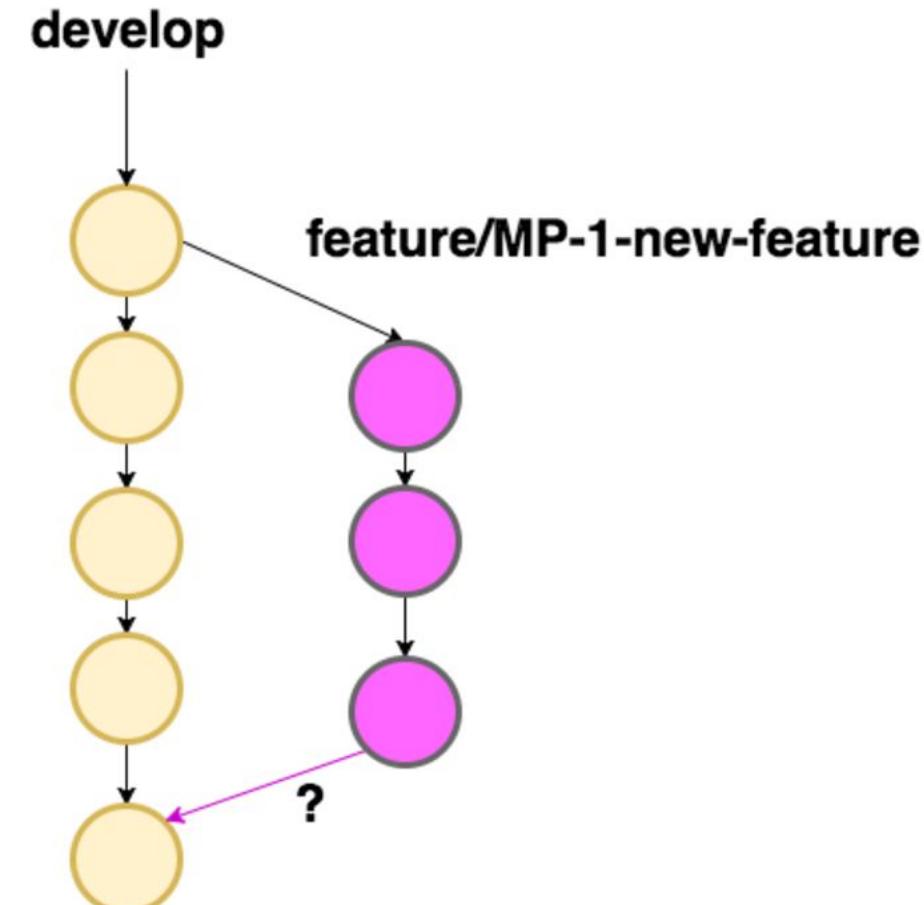
# pull request

## żądanie wcielenia kodu

Mamy bogaty model  
branchowania.

Jest ok.

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



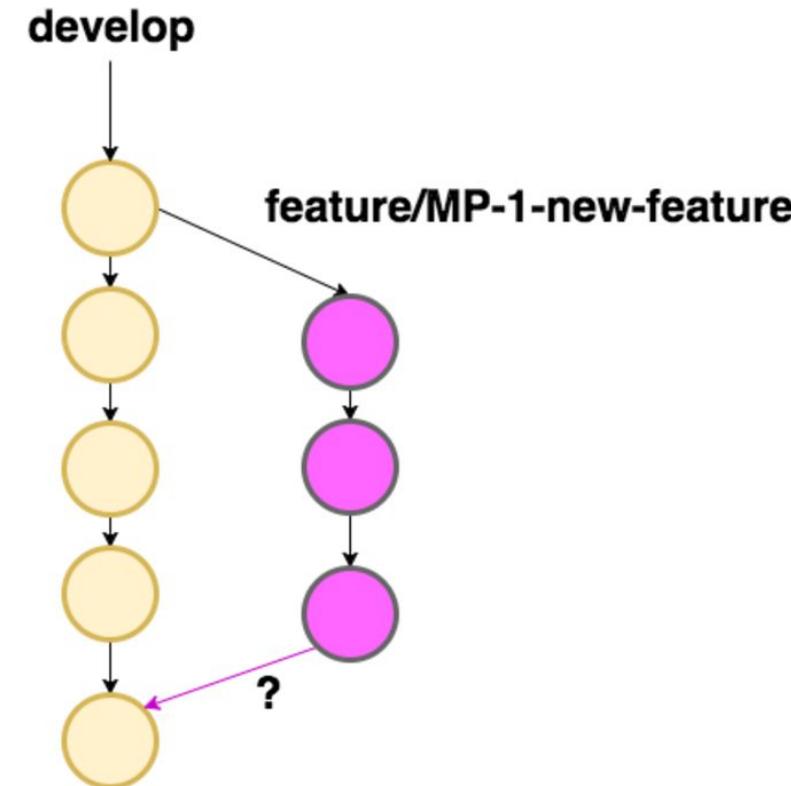
# pull request code review

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 repozytorium?

Jeśli nie chcemy mieć własnego serwera, użyjmy istniejącej usługi!





vs



ATLASSIAN  
Bitbucket

Dlaczego te dwie?  
Bo są najpopularniejsze

# 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.

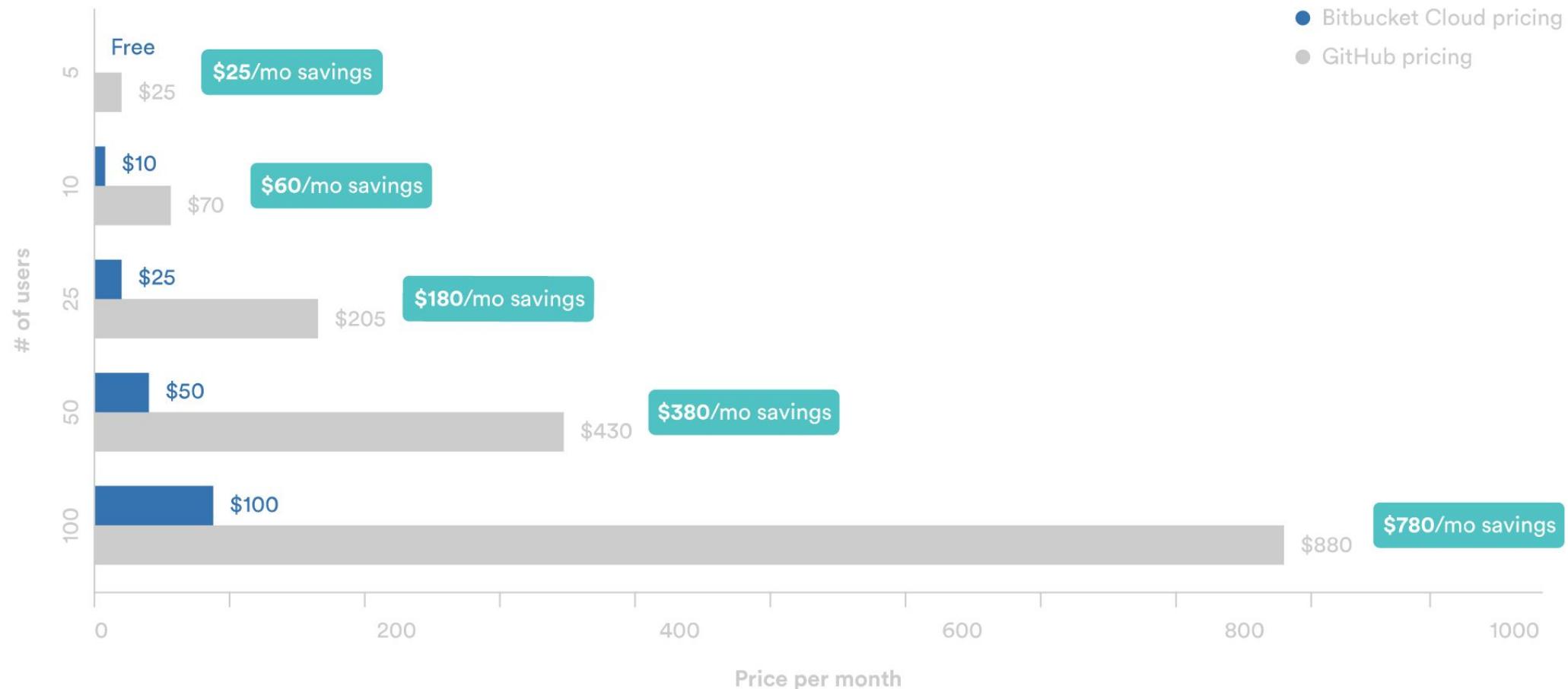


[BitBucket documentation](#)

# Małe porównanie

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

# Pricing wg Atlassian



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

# IMHO gdzie hostować

**Bitbucket**

- Overview
- Repositories**
- Projects
- Pull requests
- Issues
- Snippets

**My private repositories**

Search or jump to... [magnifying glass icon]

Pull requests Issues Marketplace Explore [bell icon] [plus icon] [profile icon]

**Overview**   Repositories 38   Stars 258   Followers 34   Following 8

**Pinned repositories** Customize your pinned repositories

**ngsummit-angular-meets-redux**  
Example usage of redux pattern with Angular. Made for ngSummit conference.  
TypeScript

**simple-hapi-rest-api**  
RESTfull API in hapijs, created to play with it on classes that I teach  
JavaScript

**gy-angular-workshops**  
Angular 4 workshops @ Goyello.  
TypeScript ★ 3 ⚡ 2

**itad-aspnet\_core**  
Repository for IT Academic Days presentation  
C# ★ 1

**typescript-workshops**  
Code and exercises for TypeScript workshops.  
TypeScript ★ 4

**mobile-web-do**  
Server and clients for my presentation about mobile browsers capabilities  
TypeScript ★ 1

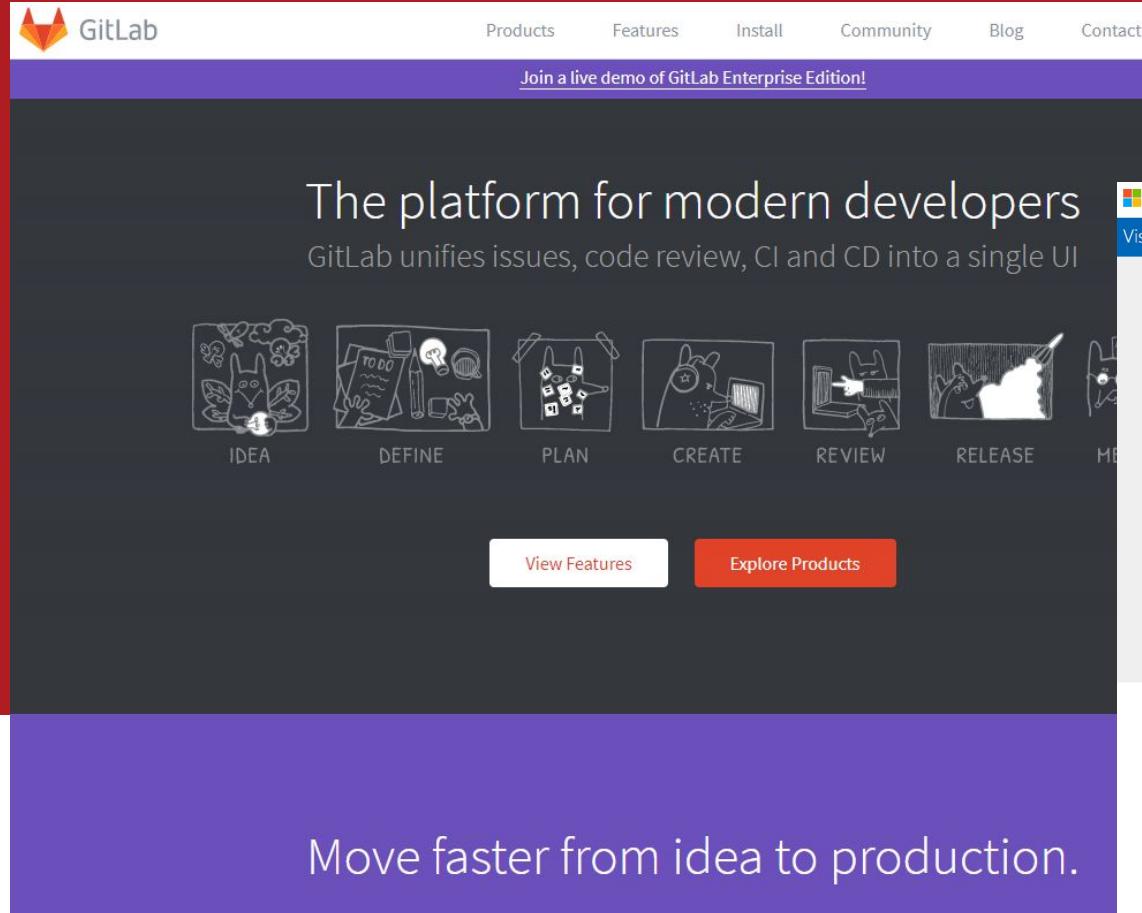
774 contributions in the last year Contribution settings ▾

Learn how we count contributions. Less More

# IMHO gdzie hostować

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

# Inne usługi warte przejrzenia



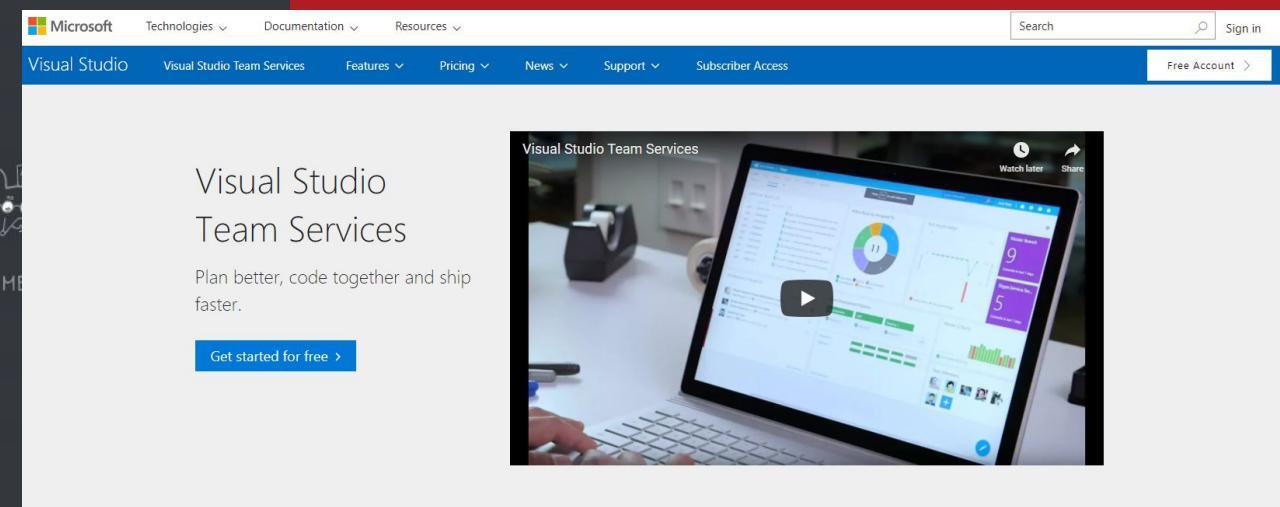
The platform for modern developers  
GitLab unifies issues, code review, CI and CD into a single UI

IDEA    DEFINE    PLAN    CREATE    REVIEW    RELEASE    M

[View Features](#)    [Explore Products](#)

Move faster from idea to production.

GitLab: <https://gitlab.com>



Microsoft Technologies Documentation Resources

Visual Studio Visual Studio Team Services Features Pricing News Support Subscriber Access

Search Sign in

Free Account >

Visual Studio Team Services

Plan better, code together and ship faster.

[Get started for free >](#)

Watch later Share

A laptop screen displaying the Visual Studio Team Services interface, showing a Kanban board, a progress bar, and various team metrics.



Create the perfect dev environment for your team

Code in any IDE/language and build applications for any target platform. Integrate your favorite tools from the marketplace.

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

# Klienci GIT'a

WebStorm/IntelliJ to nie jedyne rozwiązanie



# Sourcetree

<https://www.sourcetreeapp.com/>



<https://git-scm.com/>



<https://desktop.github.com/>



<https://code.visualstudio.com/>  
<http://gitlens.almond.io/>



**axosoft**  
**GitKraken**

<https://www.gitkraken.com/>



<https://www.jetbrains.com/>

Opcji jest masa

# Sourcetree

<https://www.sourcetreeapp.com/>



<https://git-scm.com/>



Opcji jest masa,  
również darmowych



<https://desktop.github.com/>

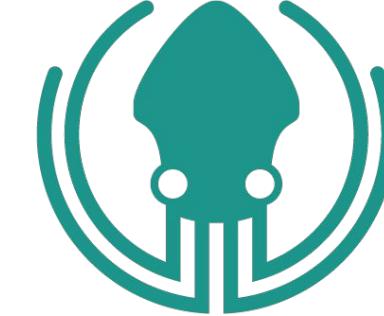


<https://code.visualstudio.com/>  
<http://gitlens.amod.io/>

**Bezpłatne do  
komercyjnego użytku**



<https://www.jetbrains.com/>



**axosoft**  
**GitKraken**

<https://www.gitkraken.com/>

**Bezpłatne dla  
open-source  
oraz do użytku  
domowego**

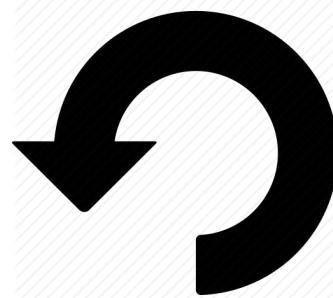
# Git w konsoli

- mój .gitconfig z aliasami do komend:  
<https://gist.github.com/michalczukm/b7dcfceeb68a97bdd916>
- jeśli pracujesz na linux'ie lub OSX polecam shella 'oh my zsh':  
<https://github.com/robbyrussell/oh-my-zsh>



# Wycofywanie zmian

Pomyliłem się. Co robić?



# Wycofywanie zmian

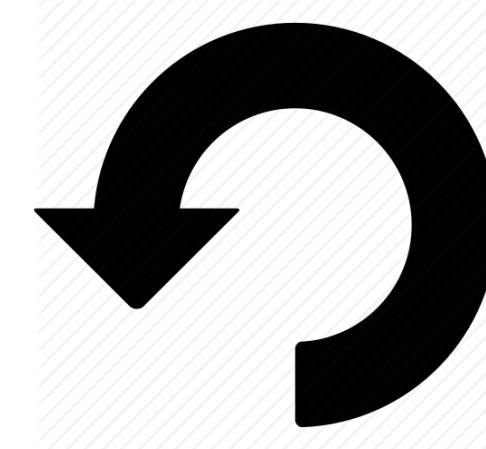
## Jest parę mechanizmów

`git commit --amend`

`git reset`

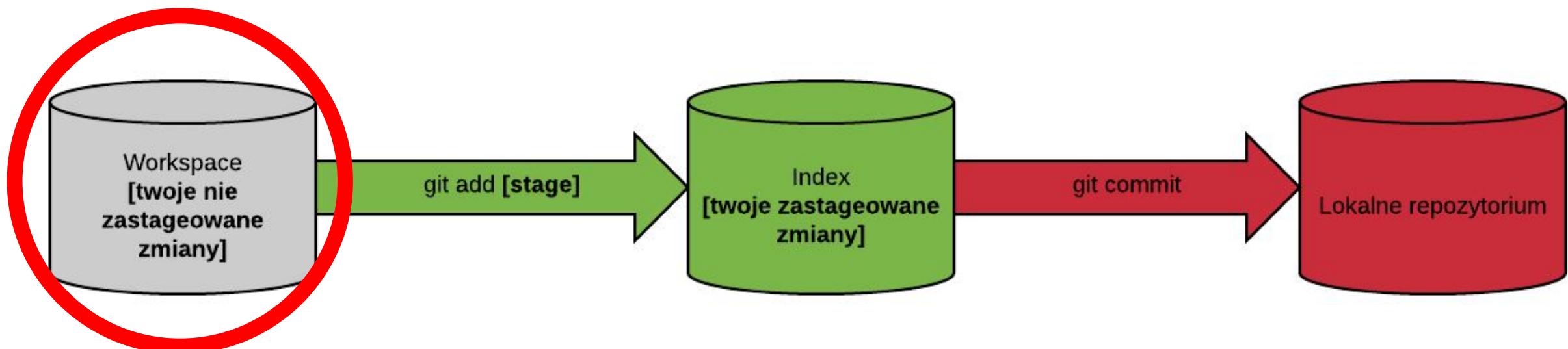
`git revert`

`git rebase -interactive`



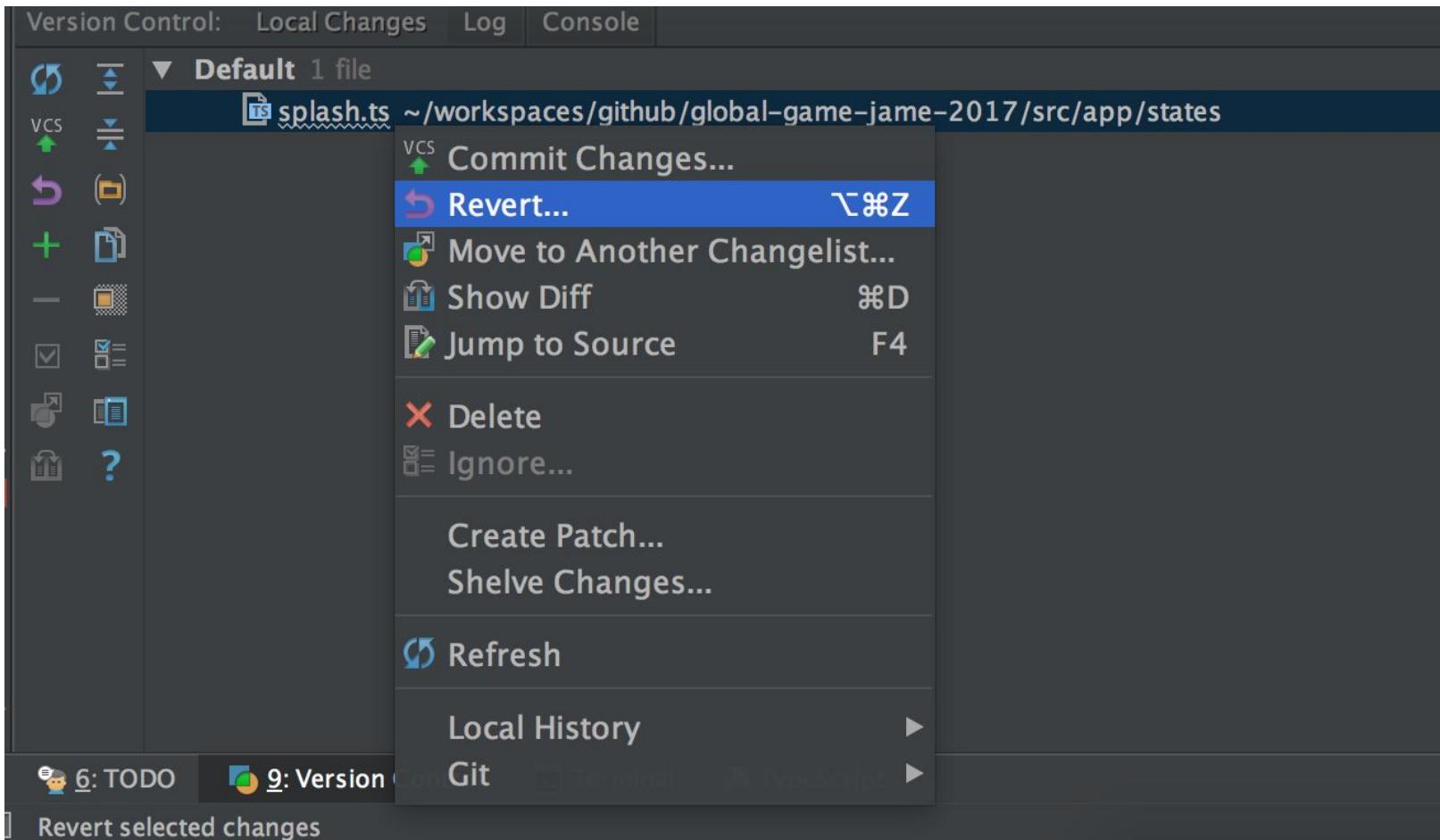
# Wycofanie lokalnych zmian W workspace, przed commitem

git checkout <file-path>



# Ćwiczenie

## Git checkout



Tak.

W WebStorm/IntelliJ ta operacja nazywa się revert.



## git revert

git revert <*sha*>

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

Wskazujemy SHA commita do wycofania.



# Ćwiczenie

## Git revert

**revert** z GIT.

Dodaj commit który posiada:

- 2 dodane linie
- 3 usunięte linie

A teraz go wycofajmy, przez **git revert**.

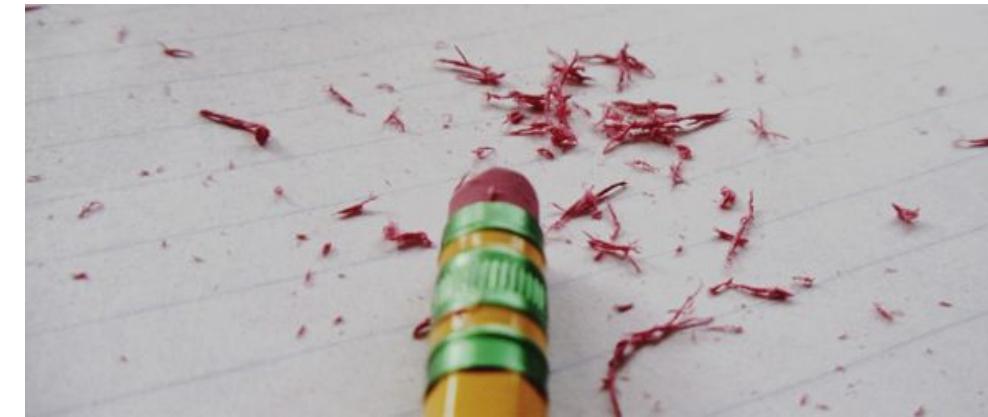


# git commit amend

## znowu się pomyliłem

```
git commit --amend
```

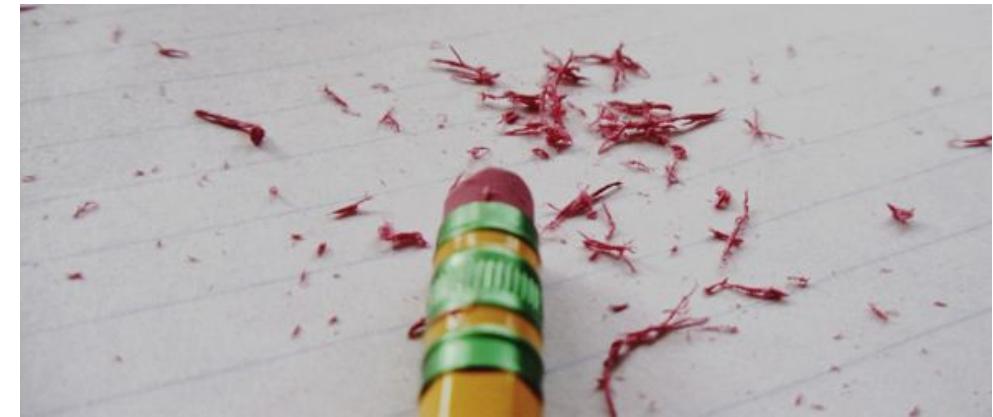
Gdy chcemy coś dołożyć do ostatniego commita albo zmienić mu nazwę.



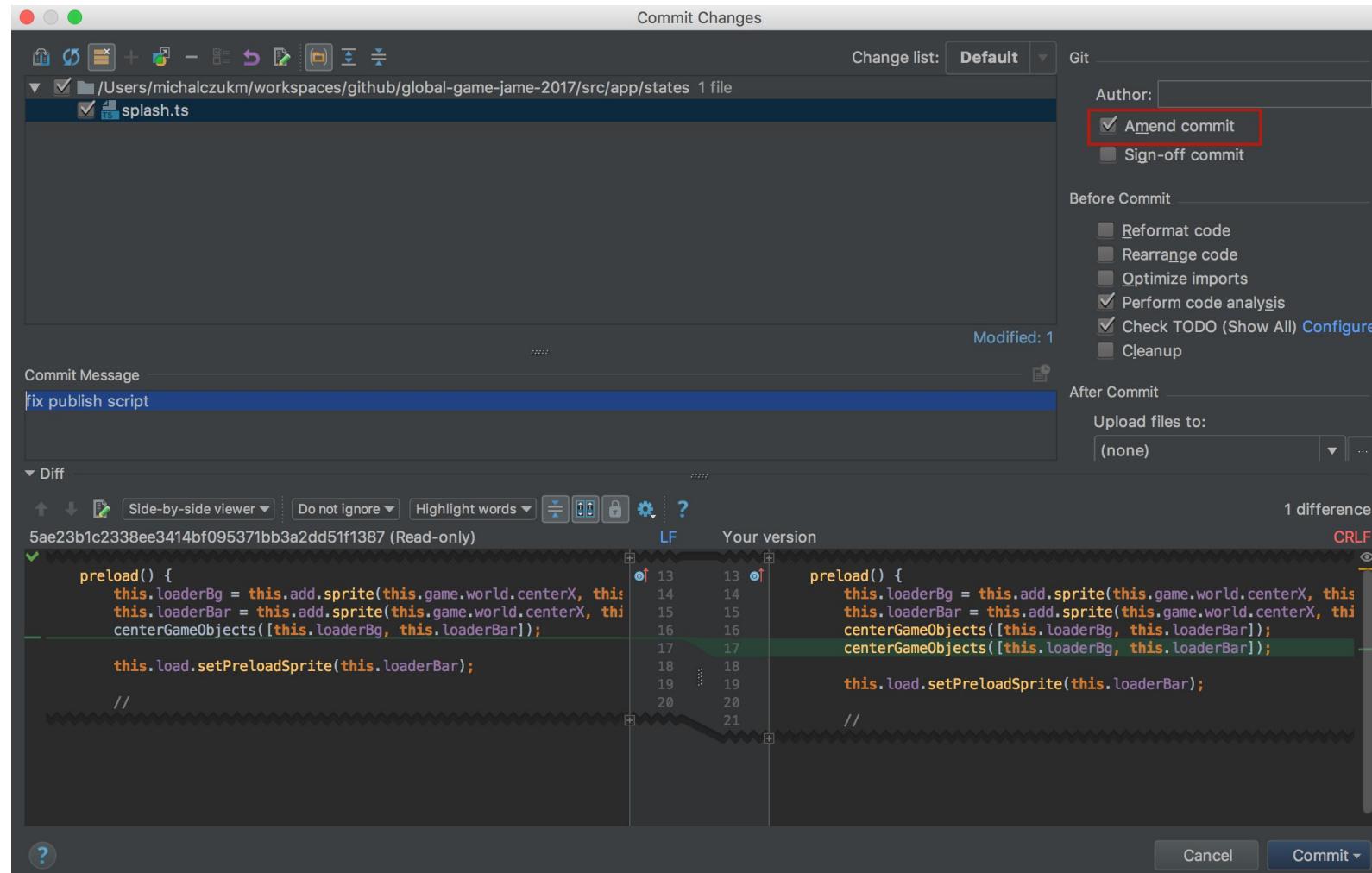
# Zmienić? Przecież commit są niezmienne

```
git commit --amend
```

Commity są **niezmienne** więc tak naprawdę tworzymy **nowy** commit.



# Ćwiczenie git commit --amend



Podczas commitowania.

Checkbox po prawej stronie.

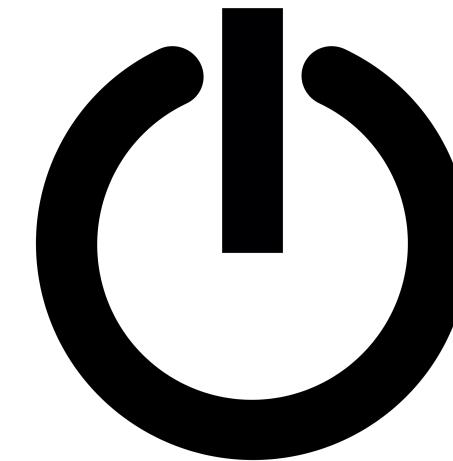
“Amend commit”

# git reset

## przywracanie stanu

git reset

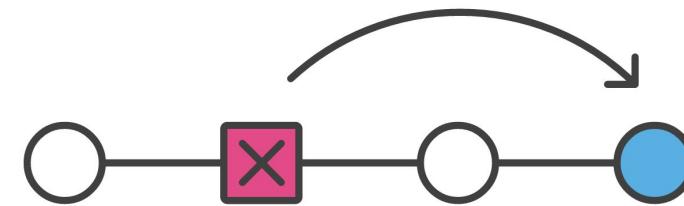
- Ustawianie workspace do stanu commita
- Ustawianie brancha (wskaźnika) na konkretny commit



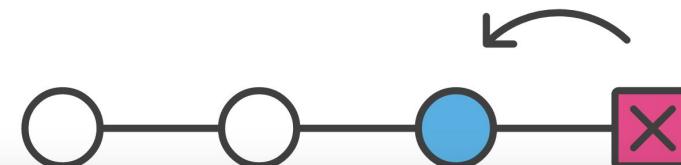
# git revert vs git reset

## Jaka jest różnica

Reverting



Resetting



# Ćwiczenie

## Git reset - wyrzućmy lokalne zmiany



# Ćwiczenie

## Git reset - odczepiamy commita od brancha



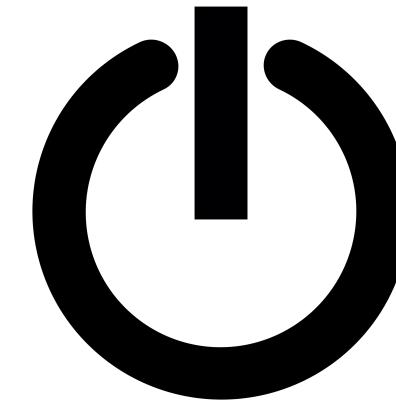
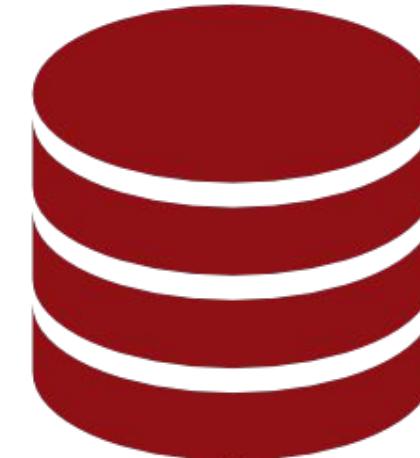
| **git reset**  
| zdalne repo

Jeśli wysłałyśmy już zmiany.

To gorzej.

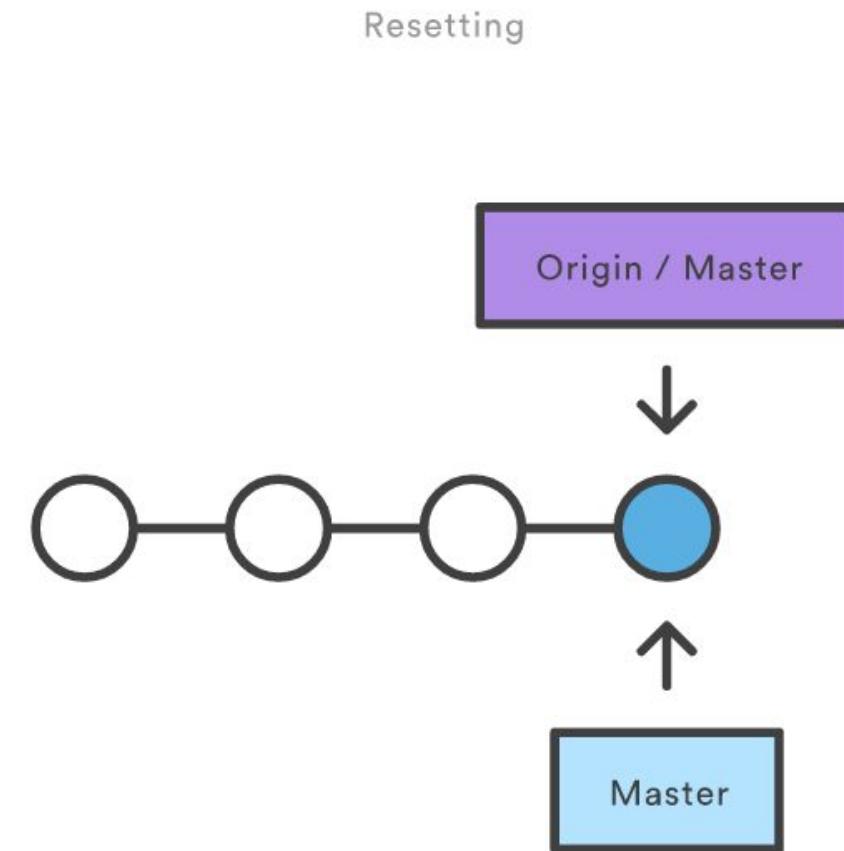
**git reset** nadpisuje historię.

Wskaźniki nie będą się  
zgadzać.



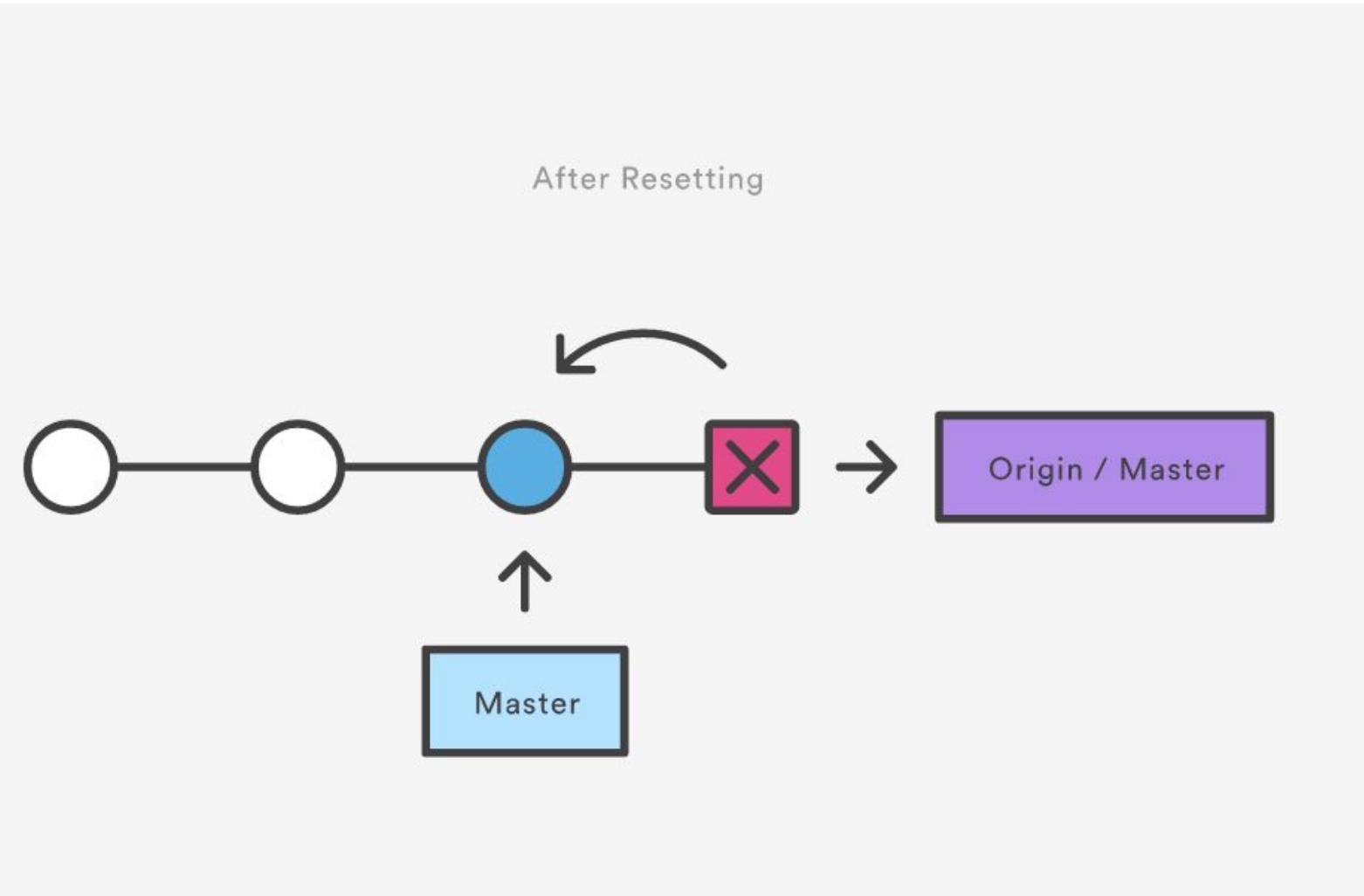
# git reset / zdalne repo - jak to wygląda

1/3



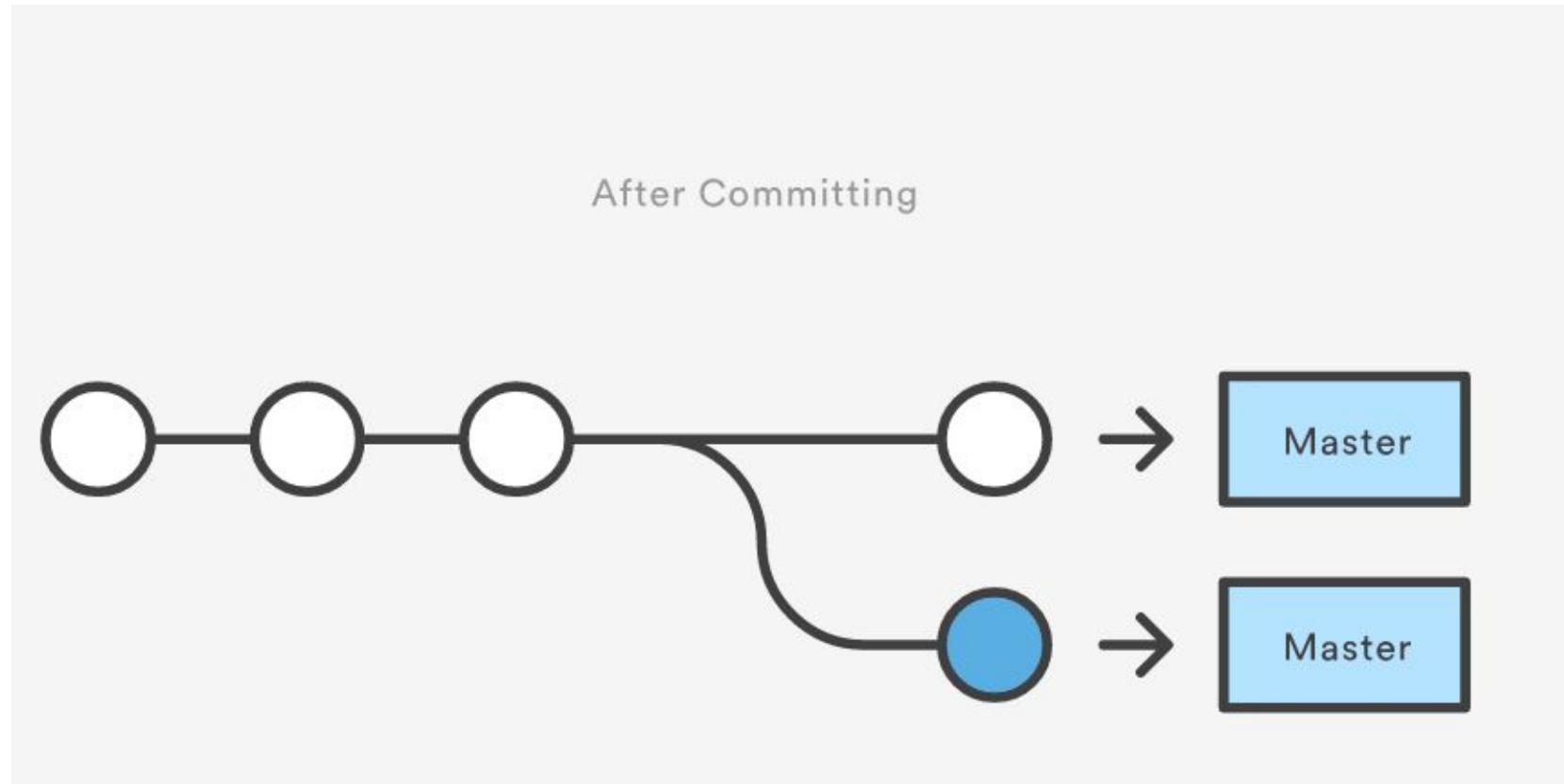
# git reset / zdalne repo - jak to wygląda

2/3



# git reset / zdalne repo - jak to wygląda

3/3



# Git cherry-pick

Wybieraj tylko wisienki



# git cherry-pick

## wybierz commita

```
git cherry-pick <sha>
```

Robimy kopię commita o  
wskazanej sumie kontrolnej  
(SHA) i układamy ją na HEAD.

Czyli na górze naszego  
aktualnego brancha



# git cherry-pick

## wybierz commita

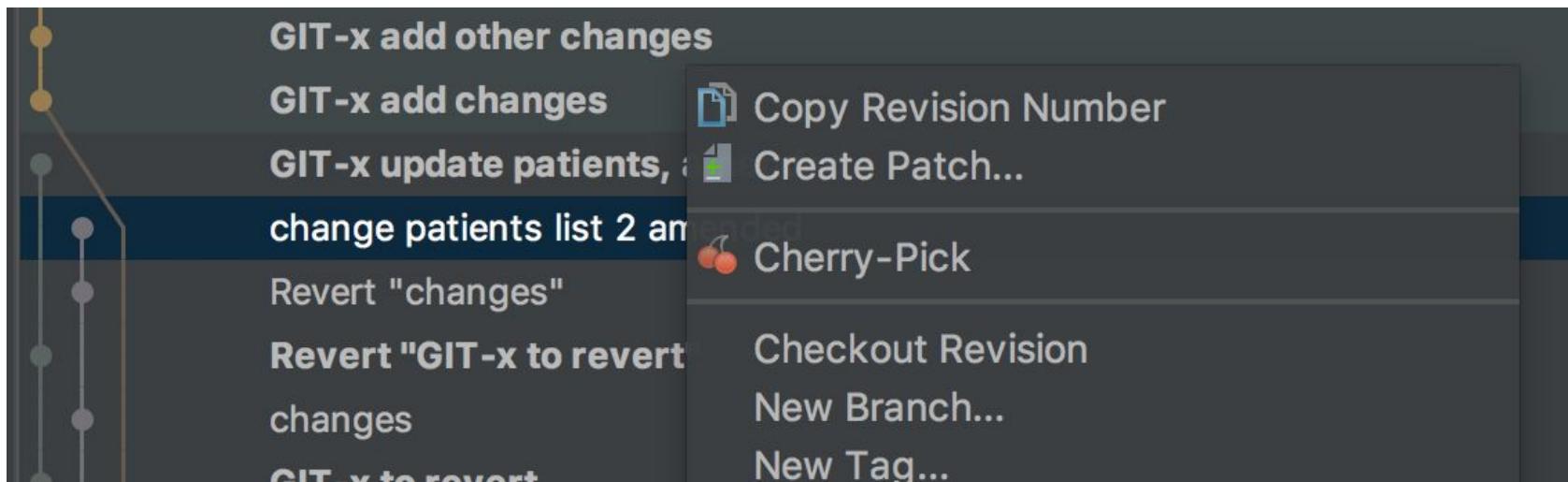
Kiedy to wykorzystać?

- Inna osoba z teamu wciąż robi zadanie a bardzo potrzebujesz jej jednej zmiany
- Na twoim branchu jest fix buga który musi szybko wjechać

*Mając konkretnego commitu się przydaj!*

# Ćwiczenie

## git cherry-pick <sha>



Wybierz commit który chcesz cherry-pickować prawym przyciskiem myszy.

(!) Mogą pojawić się konflikty.  
Dlaczego?

# Ćwiczenie

## Git cherry-pick



# Git connect via SSH

Autoryzuj się kluczem zamiast username:password



ssh

czym jest ssh?

**Protokół sieciowy do  
bezpiecznego łączenia się  
poprzez niezabezpiezioną sieć.**

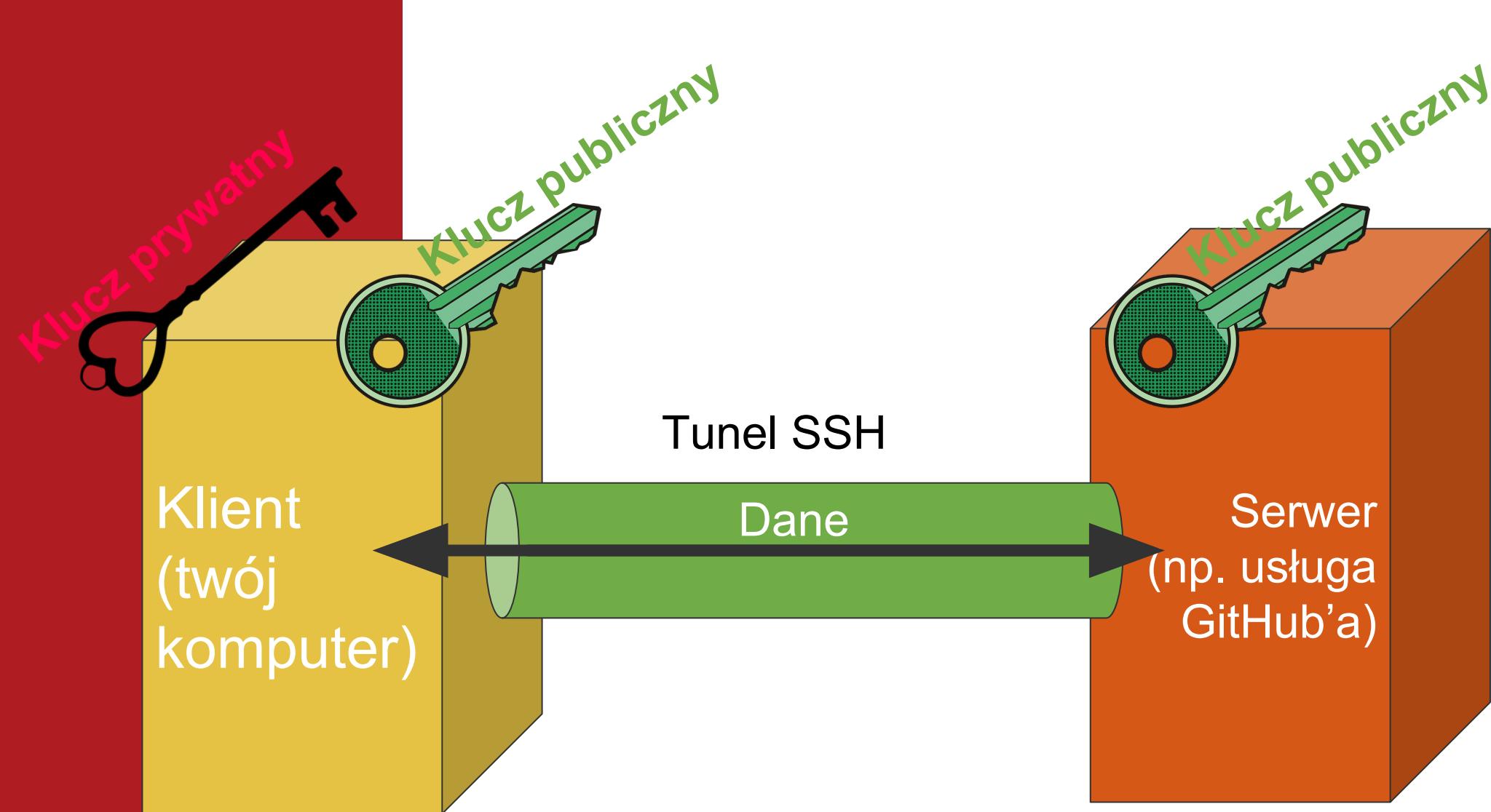
>  
—  
**SSH** 

# ssh + git ? autoryzacja

Zamiast podawania username  
i hasła, wystarczy nam używanie  
klucza.

<https://help.github.com/articles/connecting-to-github-with-ssh/>





# ssh

## czym są klucze?

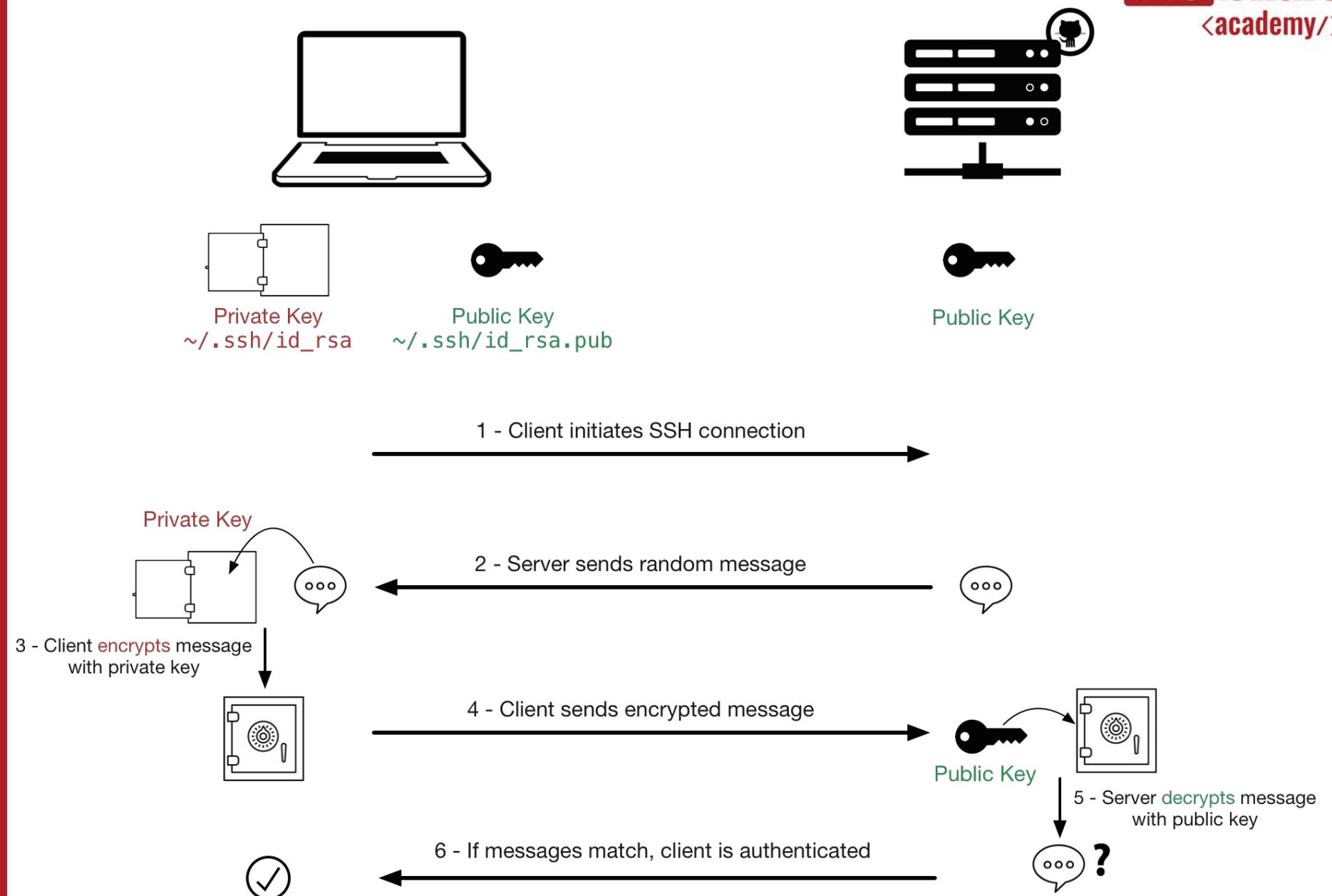
- to **tylko** pliki
- katalog **~/.ssh**
- klucz publiczny - podajemy
- klucz prywatny - **nigdy nie udostępniaj!**



~/.ssh/id\_rsa



~/.ssh/id\_rsa.pub

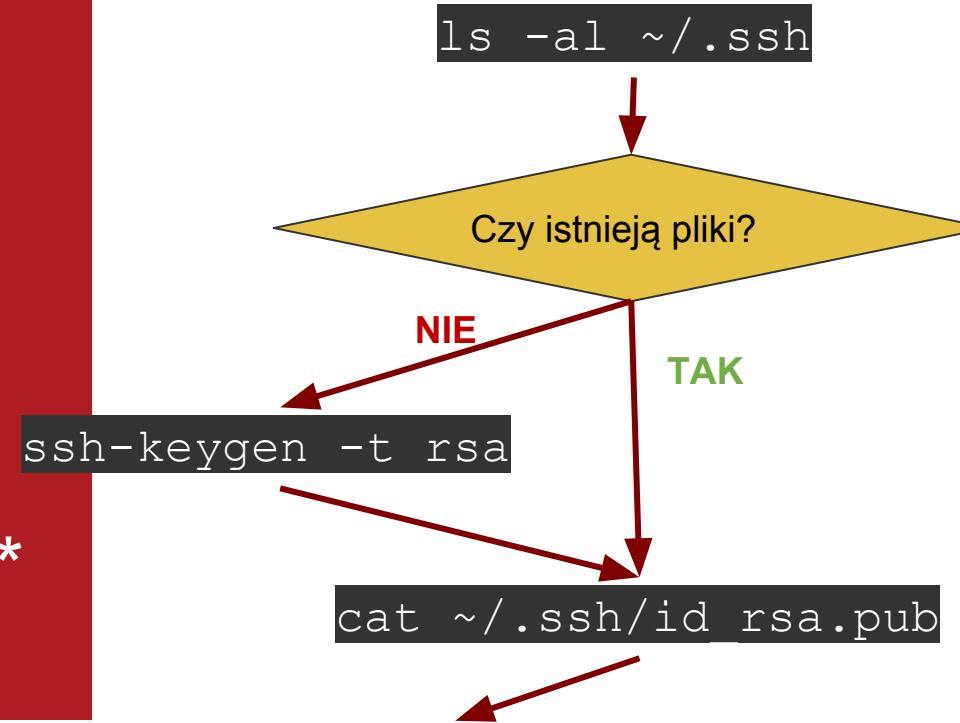


Source:

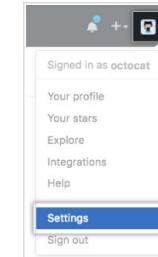
<https://sebastien.saunier.me/blog/2015/05/10/github-public-key-authentication.html>

# ssh generacja kluczy

1. Sprawdź czy już istnieją
  - a. Jeśli nie - wygeneruj klucz\*
2. Wypisz i skopiuj **klucz publiczny**
3. Wrzuć **klucz publiczny** (id\_rsa.pub) na GitHub



2 In the upper-right corner of any page, click your profile photo, then click **Settings**.



3 In the user settings sidebar, click **SSH and GPG keys**.



4 Click **New SSH key** or **Add SSH key**.



5 In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Air".



6 Paste your key into the "Key" field.

# Ćwiczenie klucze SSH



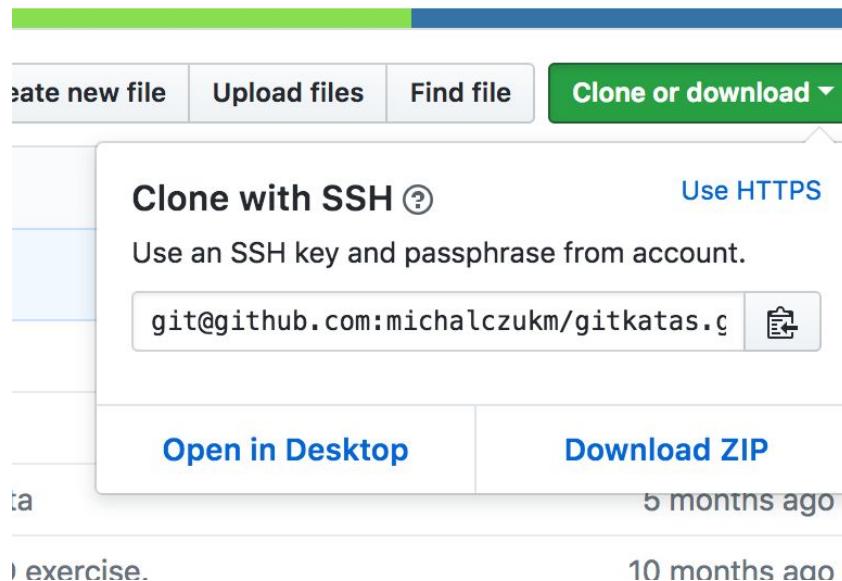
1. Wygeneruj klucze
2. Dodaj klucz do **GitHub'a**
3. Sklonuj repo do tych zajęć przez SSH
4. Zrób pull
5. Załóż brancha **imie-nazwisko** i push'nij tam dowolną zmianę

# Jak ustawić adres repo po SSH dla istniejącego projektu?

Np dla waszego projektu.

```
git remote set-url origin git@github.com/USERNAME/REPOSITORY.git
```

Gdzie adres repozytorium po SSH możesz znaleźć na swoim repo:



# Ćwiczenie git katas



Zróbmy parę przypadków

<https://github.com/michalczukm/gitkatas>

# Podsumowanie

GIT w pigułce



# Podsumowanie 1/2

- git to aktualnie najpopularniejszy system kontroli wersji, ale nie jedyny
- git jest **szybki**, to leży u jego podstaw. Śledzi zawartość plików.
- git jest **rozproszony**. Repozytorium jest zarówno u ciebie jak i reszty zespołu, oraz na zdalnym serwerze (np GitHub-ie)
- **commity** to delty (różnice) w zawartości tekstu
- **commit** ma autora, rodzica i unikatowy hash SHA-1 (sumę kontrolną)
- **branche** to tylko wskaźniki. Pokazują na dany commit  
commity są persistent (stałe), branche nie są
- **commita** nie da się zmienić - zawsze jest tworzony nowy

# Podsumowanie 2/2

- **HEAD** to wskaźnik na aktualny branch (lub commit jeśli jest deattached)
- **revert** robi commita będącego odwrotnością tego co chcemy wycofać
- **fetch** pobiera commity ze zdalnego repozytorium (np.: na GitHubie), ale nie włącza ich do naszego lokalnego repo
- **pull** robi fetch i włącza commity do naszego lokalnego repo
- nie ma narzuconego workflow dla pracy z git. Ale doświadczenie wypracowało parę dobrych. My promujemy git flow.
- git + ssh: <https://help.github.com/articles/connecting-to-github-with-ssh/>
- jest wiele usług hostingowych dla repozytoriów git'a. 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/>

- Git first aid: <http://firstaidgit.io/#/>

# Najważniejsze

In case of fire



1. git commit



2. git push



3. leave building

# Pro tip!

## How to merge feature/\* branch into develop - in the right way

F.e. you have task called *MP-1 add search engine*, so lets create branch for it.

1. Checkout develop branch - `git checkout develop`
2. Pull develop to be up to date - `git pull`
3. Create new branch `feature/MP-1-add-search-engine` here, on the top of develop
  - o Create commits on branch `feature/MP-1-add-search-engine`, put there your changes connected to this feature
  - o Push `feature/MP-1-add-search-engine` to server - do it frequently. Without push you won't have it on GitHub :)
4. When you're ready - take all changes from develop to your branch. Be up to date with your colleagues changes!
  - o checkout develop again. - `git checkout develop`
  - o pull develop To be up to date with others colleagues work - `git pull`
  - o get back to your branch - and merge current develop into it. `git checkout feature/MP-1-add-search-engine` and then `git merge develop`
5. When you're up to date with develop and resolve all merge conflicts. Not its finally time to close your feature :)
  - o checkout develop again. - `git checkout develop`
  - o merge your feature to develop (or create pull request for it) - `git merge feature/MP-1-add-search-engine`
  - o push your changes (so your merge on develop, you should be still on develop branch) - `git push`
6. Congrats - now the whole team can `git fetch` and `git pull develop` to have your changes 😊



# Dziękuję za



**michalczukm**



**michalczukm@gmail.com**



**michalczukm.xyz**