



# Mgr Oskar Riewe-Perła

E-Mail      **Oskar.Riewe-Perla@ue.poznan.pl**  
Gabinet      **C 411, 16:30**

## Zainteresowania

**Natural Language Processing**  
**Large Language Models**  
**Cognitive Science**  
**Behavioural Analysis**

Studia Licencjackie – Informatyka i Ekonometria

**Uniwersytet Ekonomiczny w Poznaniu**

Human Activity Recognition in Smart Home Environment using  
Deep Learning

Studia Magisterskie – IT & Cognition

**Uniwersytet w Kopenhadze**

Online Topic Modeling in Real Life Application

Inne

**IT University in Copenhagen**  
**Copenhagen Business School**

Nauka

**UEP – Katedra Informatyki Ekonomicznej**

Asystent, Doktorant

Praca

**ag analytics**

Data Science Team Leader



POZNAŃ UNIVERSITY  
OF ECONOMICS  
AND BUSINESS



Informatyka i Ekonometria  
**Inżynieria  
Oprogramowania**

Wprowadzenie

# Sprawy organizacyjne

---

- ▶ zapisy na kurs
  - ▶ strona <http://moodle.ue.poznan.pl/>
  - ▶ klucz dostępu: git\_clone
- ▶ sylabus
  - ▶ <https://esylabus.ue.poznan.pl/pl/document/50b43c15-2c70-408a-8fa0-626d1d4e1f74.html>
- ▶ godziny konsultacji – C 408
  - ▶ środy 16:30
  - ▶ W trakcie / po zajęciach
- ▶ warunki zaliczenia
  - ▶ Zadania grupowe – 75 pkt
  - ▶ Quiz na moodle'u – 25 pkt



# Cele uczenia się

C1	Pokazanie <b>istoty inżynierii oprogramowania</b> , celów, zakresu, relacji między inżynierią oprogramowania, inżynierią systemów, informatyką.
C2	Przedstawienie współczesnych technik i <b>dobrych praktyk</b> stosowanych w procesie wytwarzania systemów informatycznych.
C3	Zapoznanie słuchaczy z <b>procesem weryfikacji i testowania oprogramowania</b> .
C4	Przedstawienie wybranych aspektów <b>zarządzania</b> przedsięwzięciami informatycznymi.
C5	Przekazanie <b>praktycznych umiejętności</b> związanych z wytwarzaniem oprogramowania.
C6	Opanowanie <b>podstaw programowania w języku Python</b> .
C7	Zapoznanie studentów z <b>systemem kontroli wersji</b> .



# Skala ocen

---

Najwyższa	Najniższa	Nazwa oceny
100,00 %	95,00 %	5,0
94,99 %	90,00 %	4,5
89,99 %	80,00 %	4,0
79,99 %	70,00 %	3,5
69,99 %	60,00 %	3,0
59,99 %	1,00 %	2,0
0,99 %	0,00 %	X



# Quiz

- Quiz na moodle
- 25 pytań jednokrotnego wyboru
- **Quiz odbędzie się na ostatnich zajęciach**



Pusheen.com

# Oczekiwania...

---



# Inżynieria Oprogramowania

I ARE PROGRAMMER



I MAKE COMPUTER  
BEEP BOOP BEEP BOOP

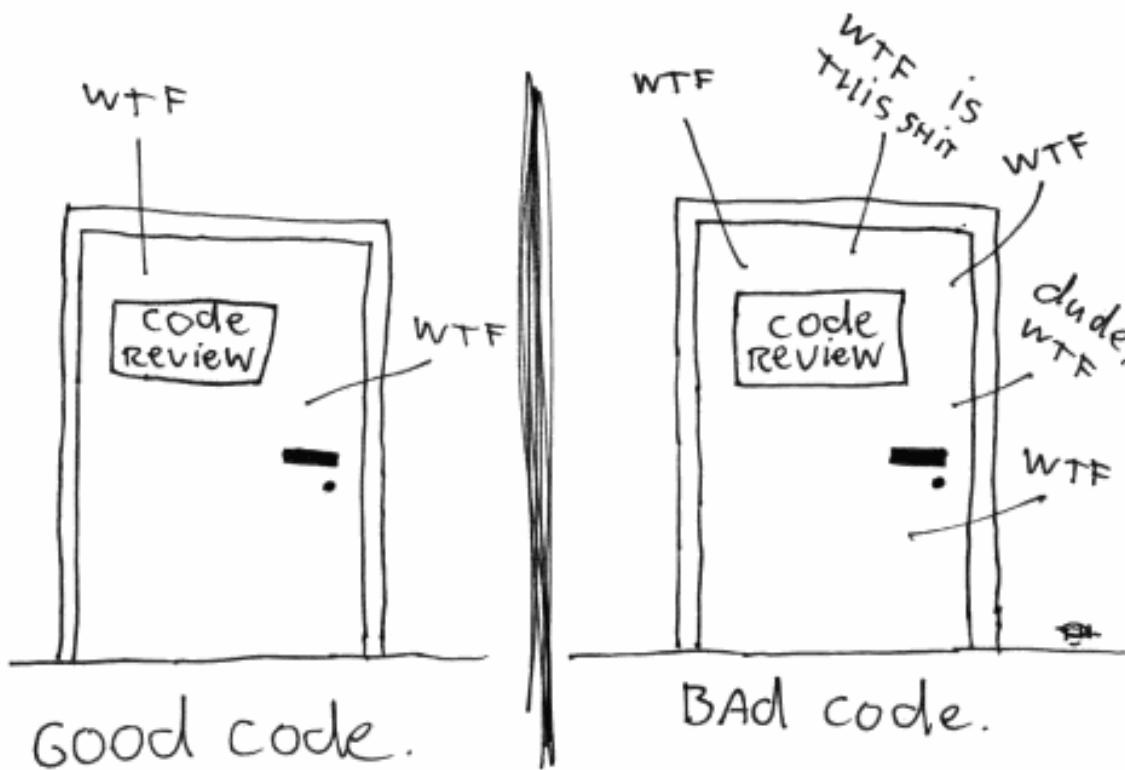
# Założenia IO

- Inżynieria oprogramowania to całokształt projektowania rozwiązania IT, od analizy i określenia wymagań, przez projektowanie i wdrożenie, aż do ewolucji gotowego oprogramowania.
  - Niektóre części tego procesu omawiane są na przedmiotach: PSI, ASI i innych podejmujących tematyki zarządzania projektami IT

# Good code

- Z książki „Czysty Kod”

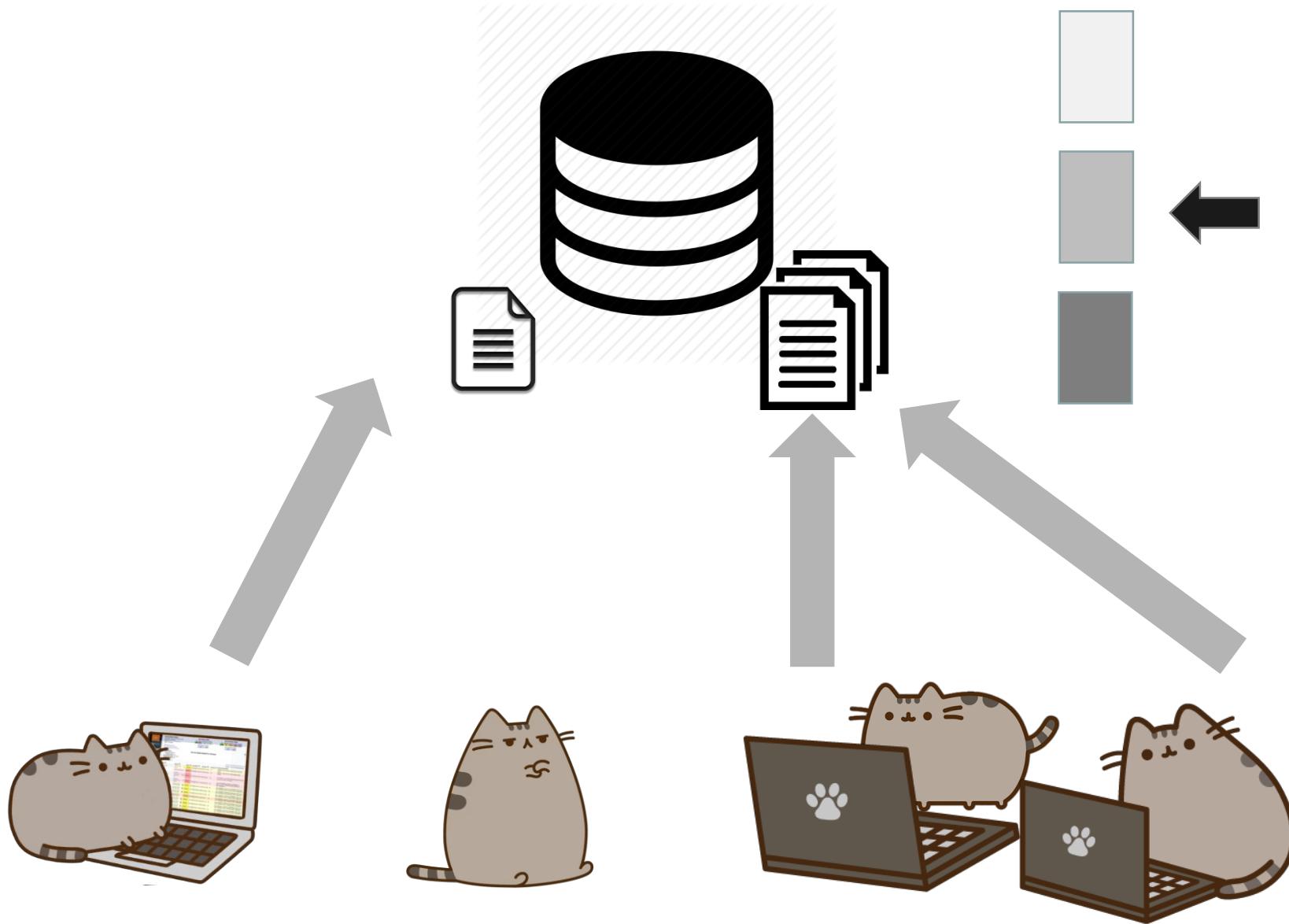
The ONLY VALID measurement  
OF CODE QUALITY: WTFs/minute



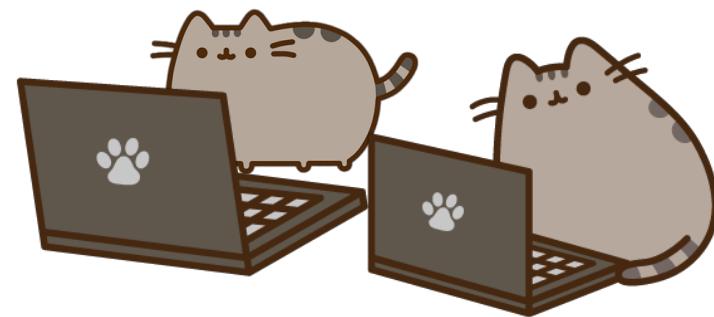
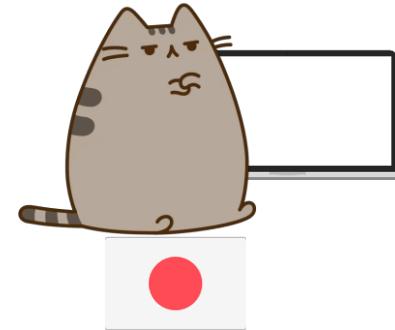
# Systemy wersjonowania

- System kontroli wersji
  - pozwala na śledzenie zmian, np. w kodzie źródłowym
  - umożliwia łączenie zmian dokonywanych w wielu plikach, przez wiele osób, w różnym czasie.
- Podział systemów kontroli wersji wg:
  - architektury oprogramowania
  - licencjonowania oprogramowania
  - sposobu oceny zmian

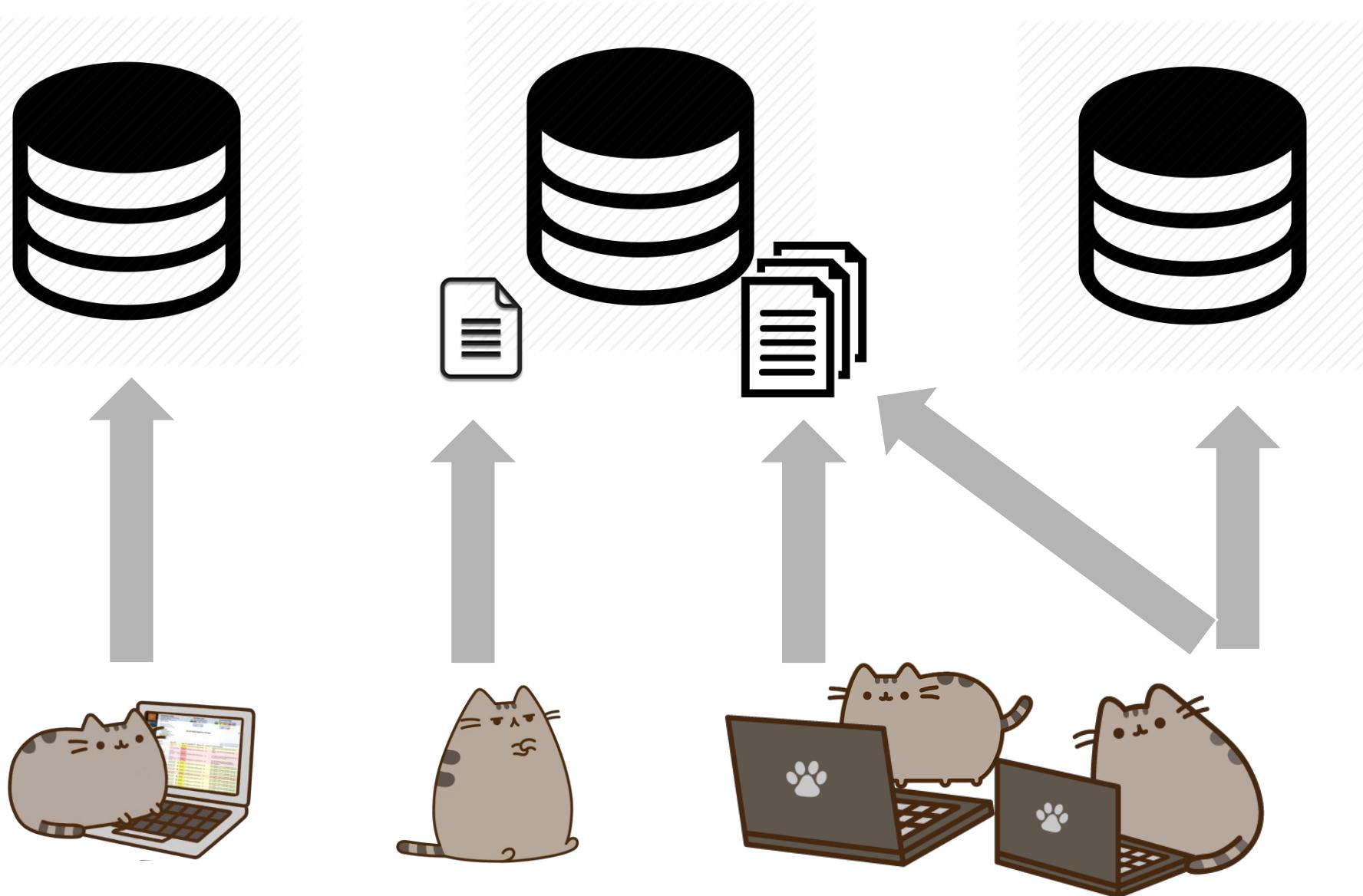
# Repozytorium



# Konflikt



# Wiele repozytoriów



# Rodzaje systemów wersjonowania

- lokalne - zapis danych jedynie na komputerze lokalnym (np. Revision Control System)
- scentralizowane - wykorzystujące architekturę klient-serwer (np. Subversion)
- rozproszone - wykorzystujące architekturę Peer-to-Peer (np. Git)

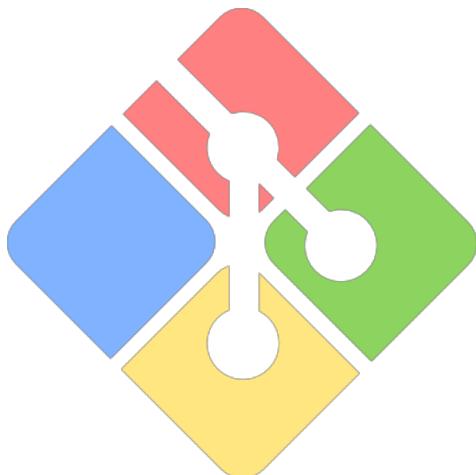
# GIT

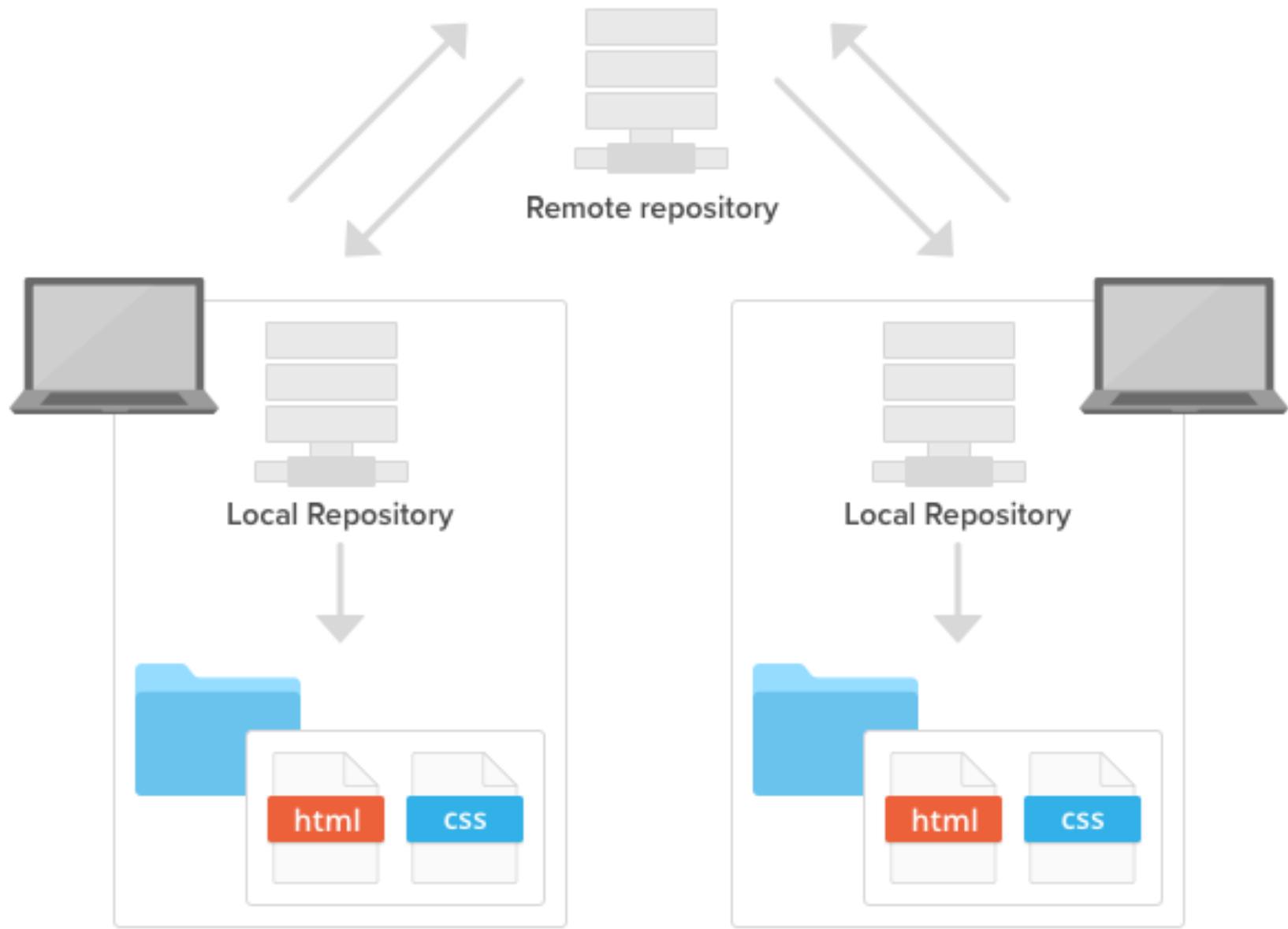


Pusheen

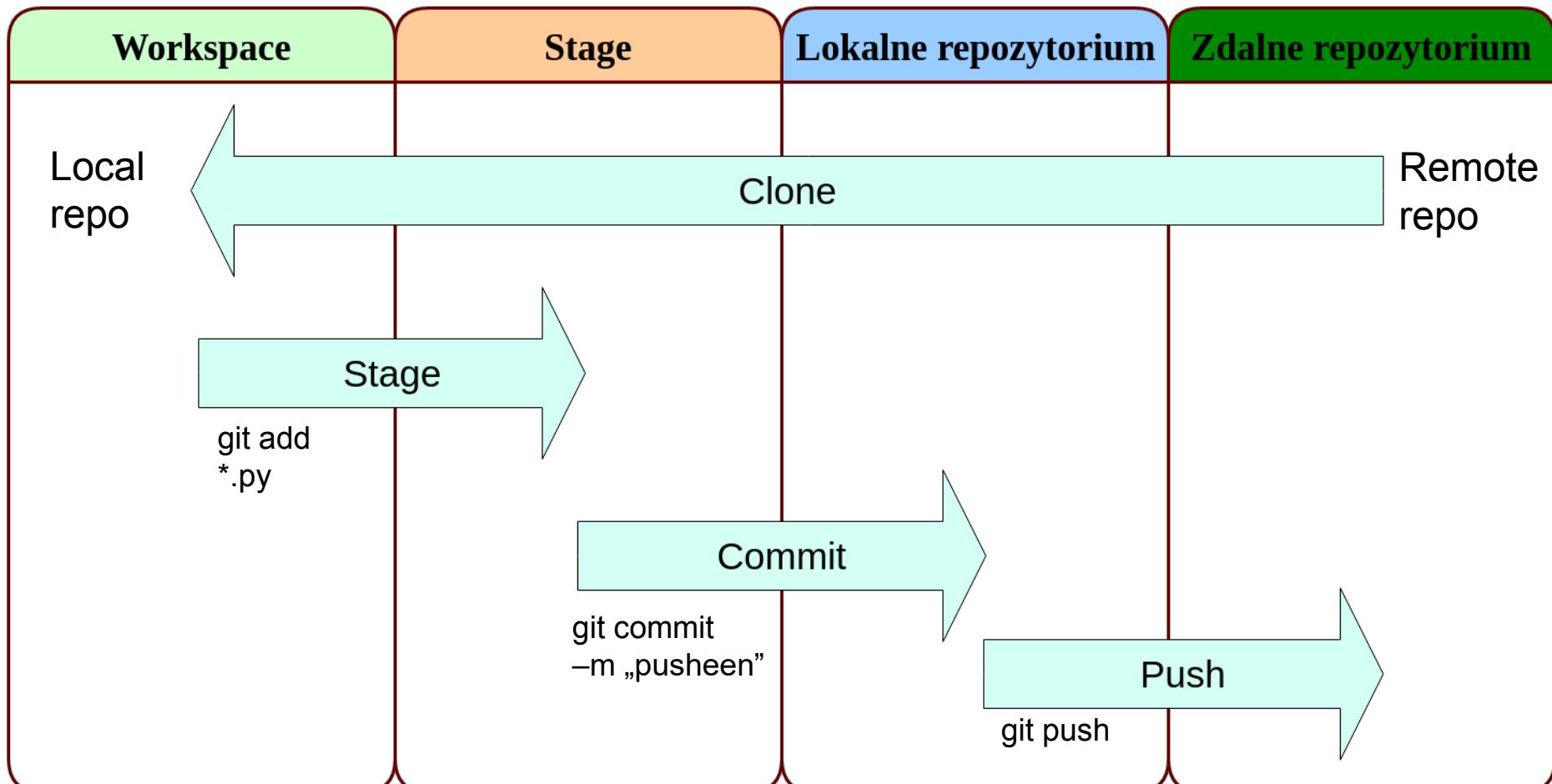


- Twórca – Linus Torvalds
- Powstał w 2005
- Wysoko popularny ; )





# Stany GIT



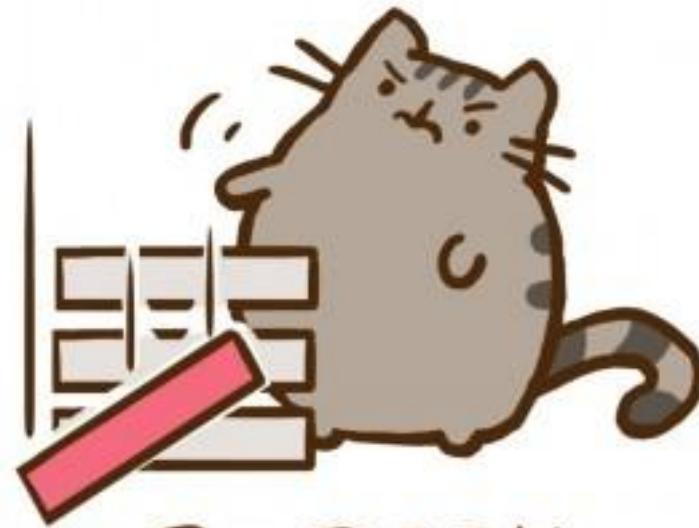
# Clone or pull?

- **git clone** is how you get a local copy of an existing repository to work on. It's usually only used once for a given repository, unless you want to have multiple working copies of it around. (Or want to get a clean copy after messing up your local one...)
- **git pull** (or **git fetch + git merge**) is how you update that local copy with new commits from the remote repository. If you are collaborating with others, it is a command that you will run frequently.
- **git clone** is downloading and **git pull** is refreshing.

# Lokalnie - Staging i revert



PUSHEEN



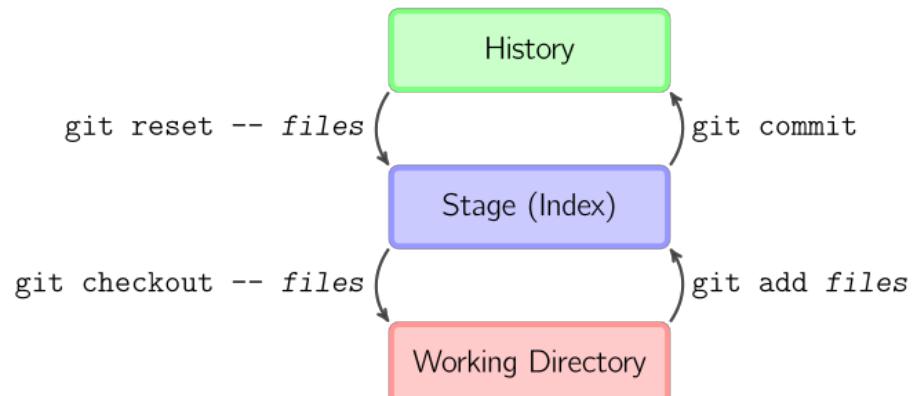
POPEEN

# Staging i revert

- Dodajemy pliki, które chcemy uwzględnić w zmianach
  - Nie korzystamy z git add \*
  - Raczej git add <plik> albo za pomocą \*.rozszerzenie
- Patche pozwalają na dodawanie fragmentów plików – tak by podzielić logicznie commity.

## Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
```



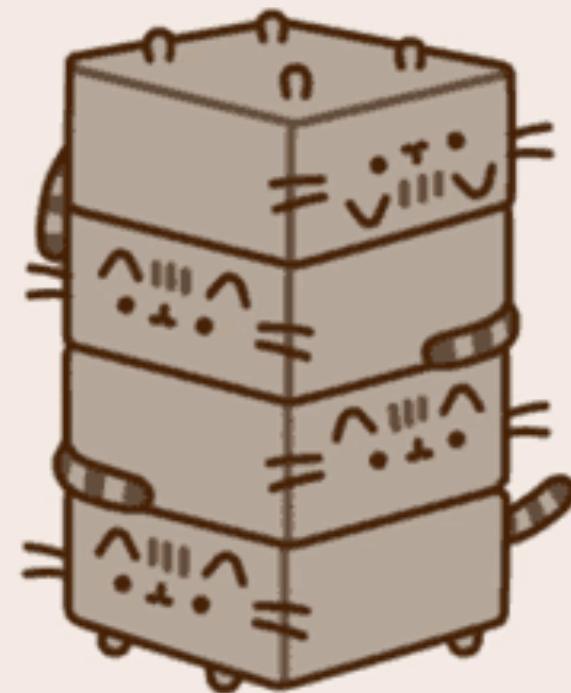
# Commit

	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

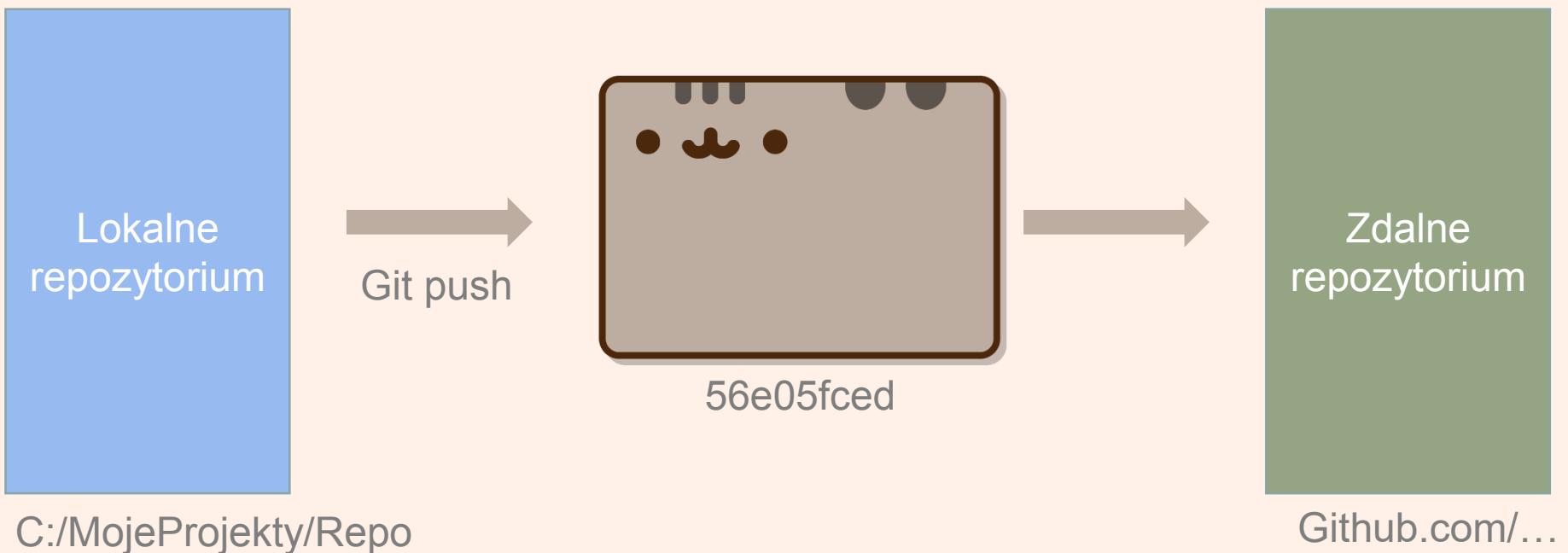
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

# Strukturyzacja commitów

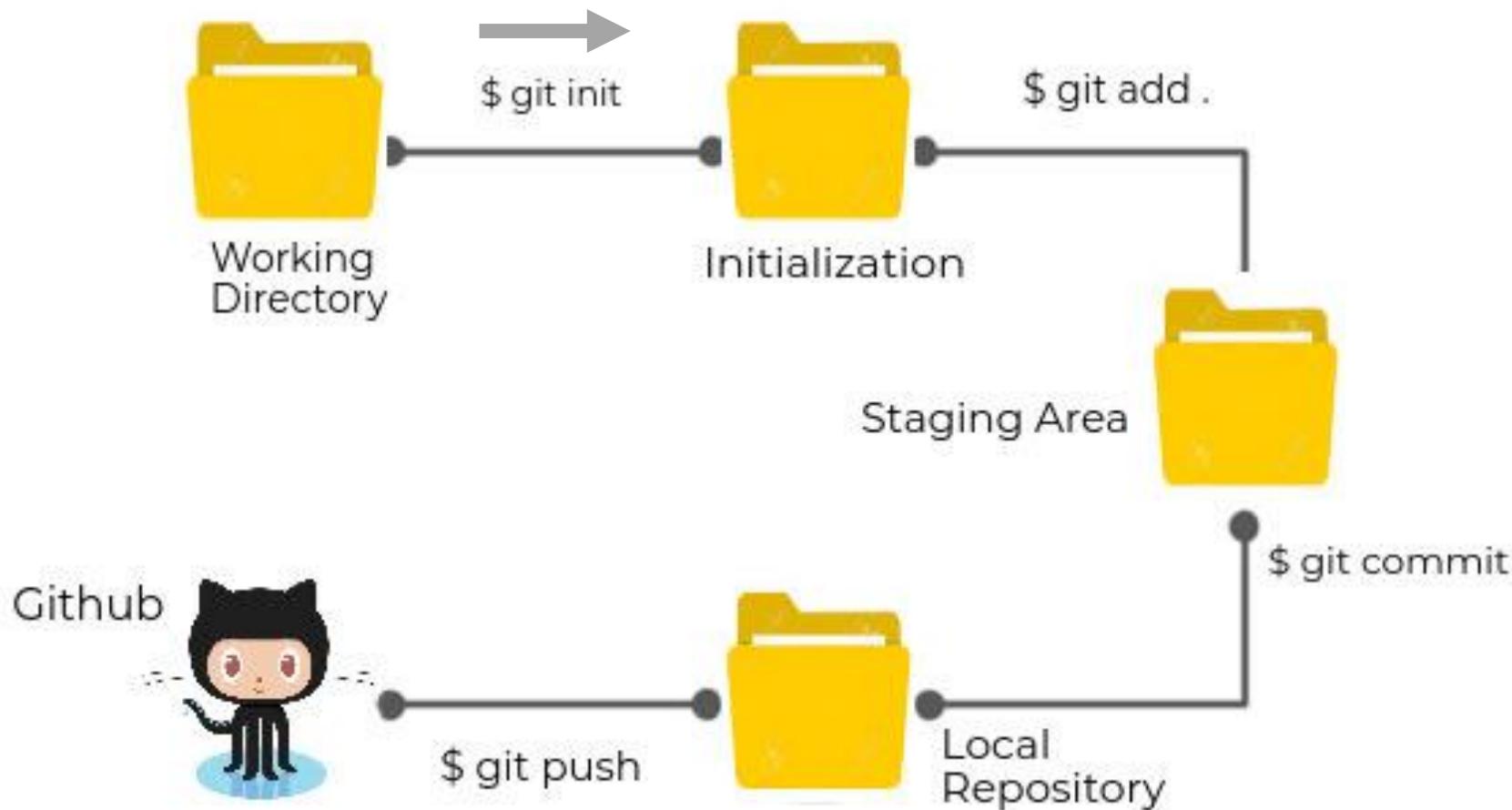
- Każda nowa funkcjonalność w osobnym commitie
- Wiadomość opisująca co zostało zrobione
  - Powód?
- W projektach praktycznych:  
referencja do tasku, błędu, buga  
do którego odnosi się commit



# Git push

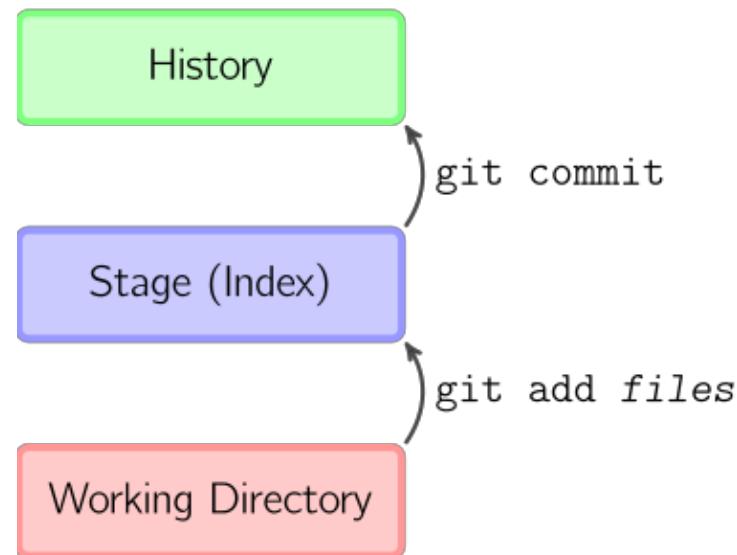


# Flow



# Zmiany i błędy

- Powrót do poprzedniego commita
  - git revert <commit-SHA>
- Usuń zmiany na poziomie „stage”
  - git checkout -- <file>
- Przywróć lokalne zmiany w plikach do tego co jest w „stage”
  - git restore --staged <file>
  - git reset (wszystko)
  - git reset HEAD <file>



# Git log / gitk

```
$ git log --graph
* commit bcb792dcc7dfbfcd620ee73ed7422295f3d50ca (HEAD -> computer_player, origin/computer_player)
  Author: lpenzey <lucaspenzeymoog@gmail.com>
  Date:   Fri Jul 27 15:19:27 2018 -0500

    cleaned formating with rubocop

* commit e953f0fdbfcf8038afec2a50f72c9d65601d346c
  Author: lpenzey <lucaspenzeymoog@gmail.com>
  Date:   Fri Jul 27 14:55:41 2018 -0500

    updated script

* commit d443cc147cf543bc2892a82143e3b0ab016f7847
  Author: lpenzey <lucaspenzeymoog@gmail.com>
  Date:   Fri Jul 27 14:53:12 2018 -0500

    added travisci

* commit bf0c6b5362ea8e675a6c866d388aee7867b816ea
  Author: lpenzey <lucaspenzeymoog@gmail.com>
  Date:   Fri Jul 27 14:49:45 2018 -0500

    added travisci

* commit ed75abbca061c2c93834d1ee606a1b665175e10d
  Merge: 08aa658 a098936
  Author: lpenzey <lucaspenzeymoog@gmail.com>
  Date:   Thu Jul 26 10:15:16 2018 -0500

    Merge branch 'save_resume_game' of https://github.com/lpenzey/Mastermind into save_resume_game

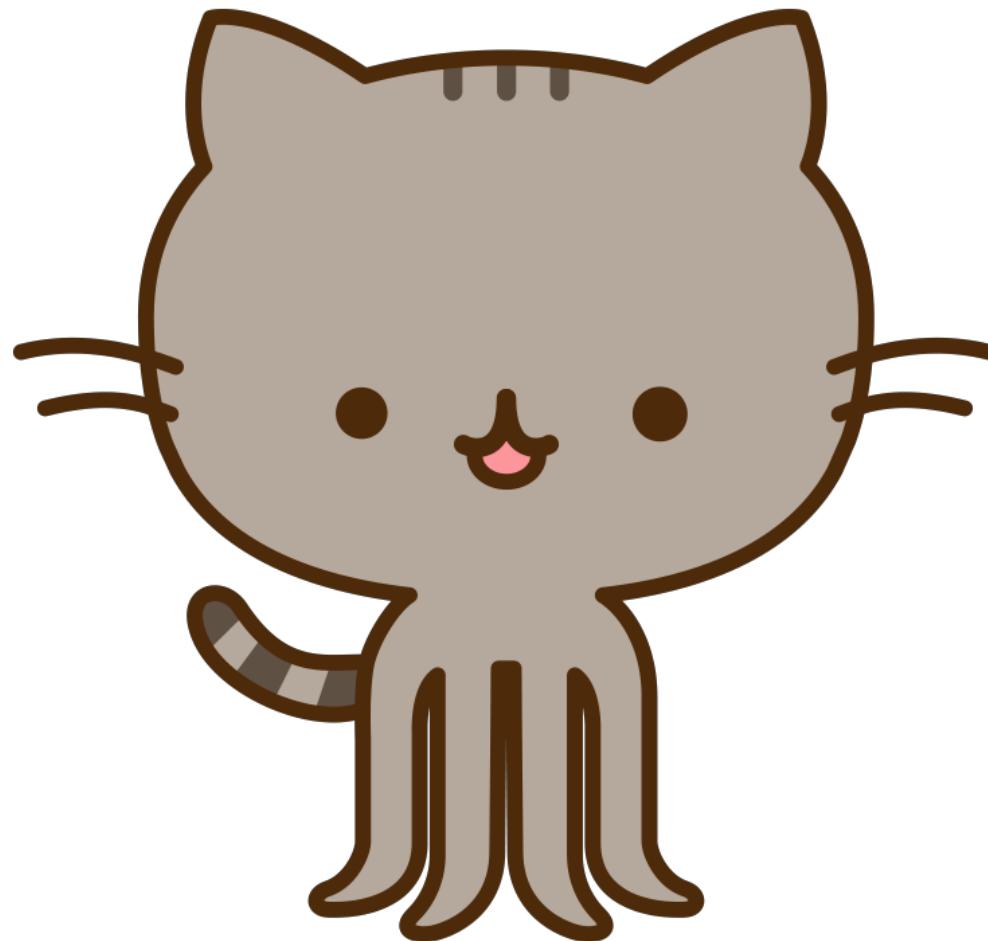
* commit a0989369074b53c09fd11d7dcb7aaf548dd03447
  Merge: 5c4c0c9 97307ff
  Author: Lucas w PenzeyMoog <32578736+lpenzey@users.noreply.github.com>
  Date:   Mon Jul 23 14:43:54 2018 -0500

    Merge branch 'master' into save_resume_game

* commit 97307ff7d12e0428941d7a3d0e0594bacd46841a (origin/master)
```

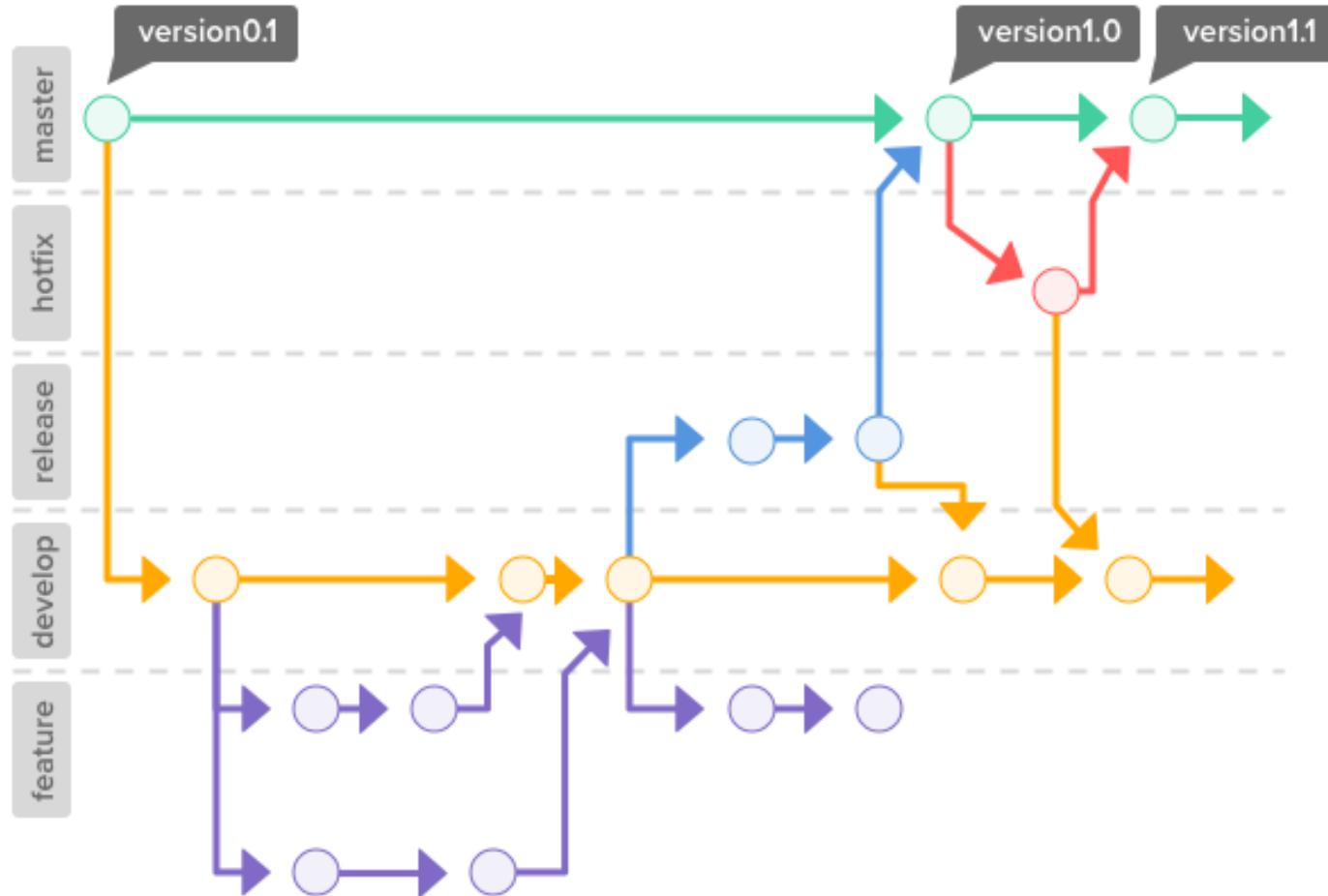
# Branche i Forki

- Git jest jak koto-meduza. Koto-meduza ma macki, git też ma macki.



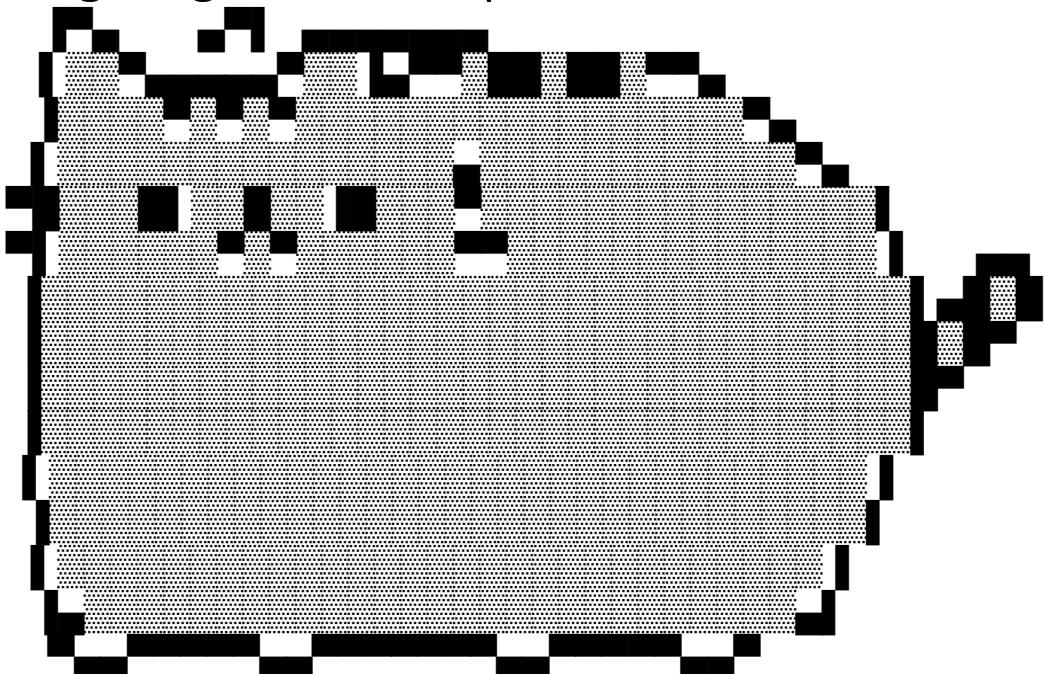
# Gałęzie/Branche i Tagi

- git tag -a v1.1 -m 'version of the dev app 1.1'



# Tag

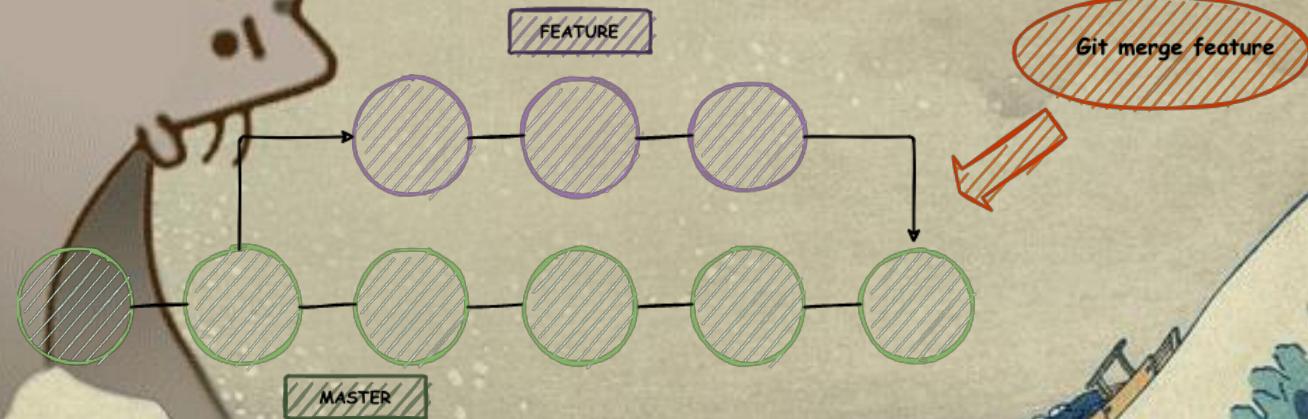
- A tag is used to label and mark a **specific commit** in the history.  
It is usually used to mark release points (eg. v1.0, etc.).
- **git tag v1.4** - lightweight tag
- **git tag -a v1.4 -m „pusheen** - annotated tag



"

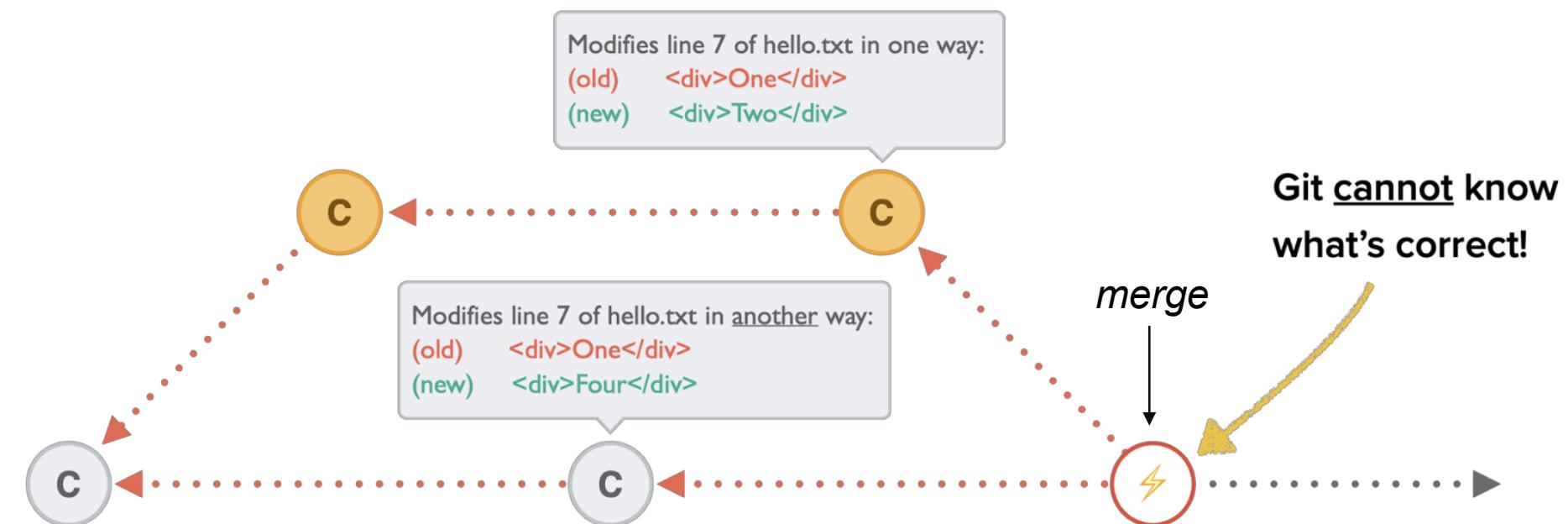
# Praca z gałęziami

- Przejście na gałąź bądź stworzenie nowej
  - git checkout –b <branch>
- Wypisanie wszystkich istniejących gałęzi:
  - git branch
- Wyświetlanie gałęzi, które zostały lub nie zostały scalone:
  - git branch --merged oraz git branch --no-merged
- Usuwanie gałęzi (nie pozwoli na usunięcie jeśli zmiany nie zostały scalone):
  - git branch -d <branch>
- Połączenie zmian z gałęzi
  - git merge <branch>



# Konflikt!

```
$ git merge develop  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
```



# Konflikty

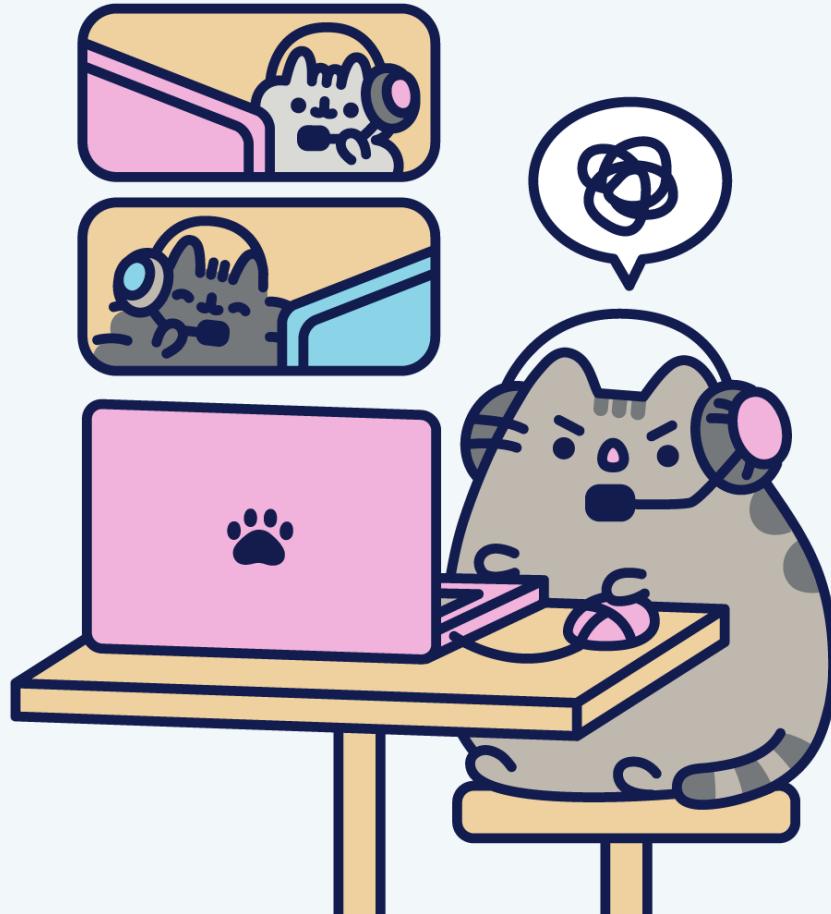
<<<< HEAD oznacza wersję znajdująca się w waszym aktualnym branchu który merujecie

===== oddziela zmiany

>>>>> BRANCH oznacza wersję na branchu który merujemy

```
index.html — carparts-website_conflict (git: main)

<div id="navigation">
  <ul>
    <<<<<< HEAD
      <li><a href="index.html">Home</a></li>
      <li><a href="about.html">About Us</a></li>
      <li><a href="product.html">Product</a></li>
      <li><a href="imprint.html">Imprint</a></li>
    =====
      <li><a href="returns.html">Returns</a></li>
      <li><a href="faq.html">FAQ</a></li>
    >>>>> develop
    </ul>
  </div>
```



# Pomocne oprogramowanie

The screenshot shows a GitHub merge interface for a branch named 'temp-branch-for-demo'. The merge is from the 'helper\_scripts' repository at 'gitlab.com:arthur-s/helper\_scripts' into the current branch.

**Merge branch 'temp-branch-for-demo' of gitlab.com:arthur-s/helper\_scripts into temp-branch-for-demo**

**# Conflicts:**  
# python2-simple-server.py

Line 5, Column 1

**Working Directory**

**python2-simple-server.py**

**do\_GET:**

```
22     self.send_response(200)
23     self.send_header('Content-type','text/html')
24     self.end_headers()
25     # transfer to Arthur's cellular phone
26     self.wfile.write("transfer +79111111111")
27
28     return
29
30     try:
31         do_GET:
32             #Create a web server and define the handler to manage the
33             #incoming request
34             server = HTTPServer(('', PORT_NUMBER), myHandler)
35
36             print('Started httpserver on port %s' % PORT_NUMBER)
37
38             #Wait forever for incoming htto requests
39             server.serve_forever()
```

**do\_GET:**

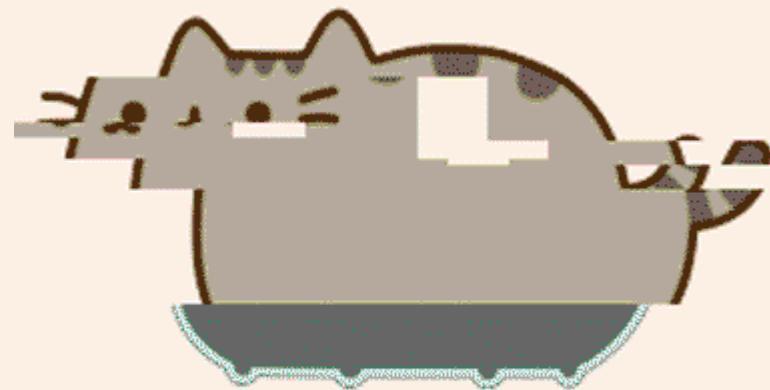
```
22     self.send_response(200)
23     self.send_header('Content-type','text/html')
24     self.end_headers()
25 <<<<< HEAD
26     # transfer to Arthur's cellular phone
27     self.wfile.write("transfer +79111111111")
28 =====
29     # Send the html message
30     self.wfile.write("transfer +81500000000")
31 >>>> 10d39749eac26d53652df5c190927479c2bc34ec
32
33     return
34
35     try:
36         do_GET:
37             #Create a web server and define the handler to manage the
38             #incoming request
39             server = HTTPServer(('', PORT_NUMBER), myHandler)
40             <<<<< HEAD
41             print('Started httpserver on port %s' % PORT_NUMBER)
42             =====
43             print('Run httpserver on port ', PORT_NUMBER)
44             >>>> 10d39749eac26d53652df5c190927479c2bc34ec
45
46             #Wait forever for incoming htto requests
47             server.serve_forever()
```

Abort merge

Resolve Stage

# Błędy – podstawy

- Przed pracą z repo
  - git pull
- Jak pomyliłeś się i chcesz usunąć zmiany przed commitem:
  - git restore <plik>
- Chcesz przywrócić plik z repo
  - git checkout -- <plik>



glitched

# Git Blame

- git blame <<PLIK>> -L <<ZAKRES\_LINI>>

A screenshot of a code editor showing a line of Python code: "30 docs = FlaskApiSpec(app)". Below the code, a status bar indicates: "File is committed, 2992e8a | Krzysztof (9 months ago), INSERT MODE, Line 30, Column 1". To the right, it shows "master [2]".

when your coworkers ask if you  
know who has been writing bad  
code all this time



- Well, of course I know him. He's me.

LitOr/pusheen-  
push



Github

Why push when you can Pusheen?

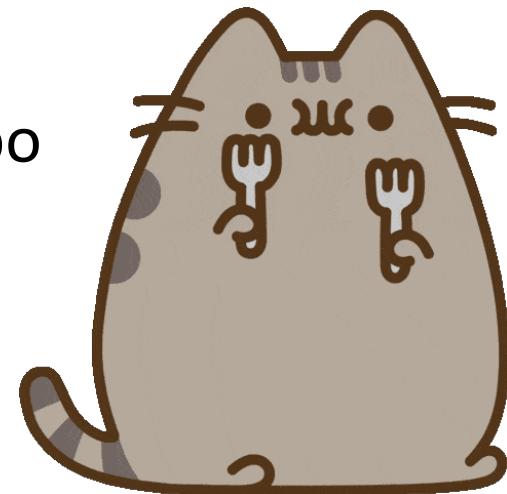
1 Contributor   0 Issues   20 Stars   0 Forks



- Jedno z największych miejsc w sieci, umożliwiające przechowywanie projektów i równoczesną pracę wielu użytkowników przy jednym projekcie.
- Działa podobnie jak sieć społecznościowa.
- W przeciwieństwie do wielu sieci społecznościowych / miejsc udostępnianych online: wszystkie załadowane dane pozostają własnością użytkownika.
- Dostarcza graficzny interfejs do zarządzania Gitem.
- <https://github.com>

# Praca z cudzymi repo – Fork i PR

- Nie zawsze repozytorium jest nasze. Czasami współdzielimy repozytorium firmowe, do którego zanim wprowadzimy zmiany ktoś musi na nie spojrzeć
- Możemy pobrać i stworzyć swoją wersję repo jako nowe repo – **Forka**
- Zmiany do cudzego repozytorium tworzymy przez „**Pull request**” PR
  - **Funkcja** ta nie jest realizowana bezpośrednio przez GIT tylko przez dostawcę usługi: GitHub, GitLab, BitBucket itd.
  - Pull request może (jest) po akceptacji włączony jako branch



# PR

```
282     width: 14.5%;  
283 +   flex: 1;
```



Write

Preview

weird flex but ok

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Cancel

Add single comment

Start a review

# PR



**Ben Nadel**

@BenNadel



My new interview question:

You just created a Pull Request and noticed that half of the changes are white-space edits caused by your IDE plug-in. Do you modify the PR? Or, pass it on to a teammate for review?

8:11 AM - 6 Dec 2017

---

6 Retweets 35 Likes



32

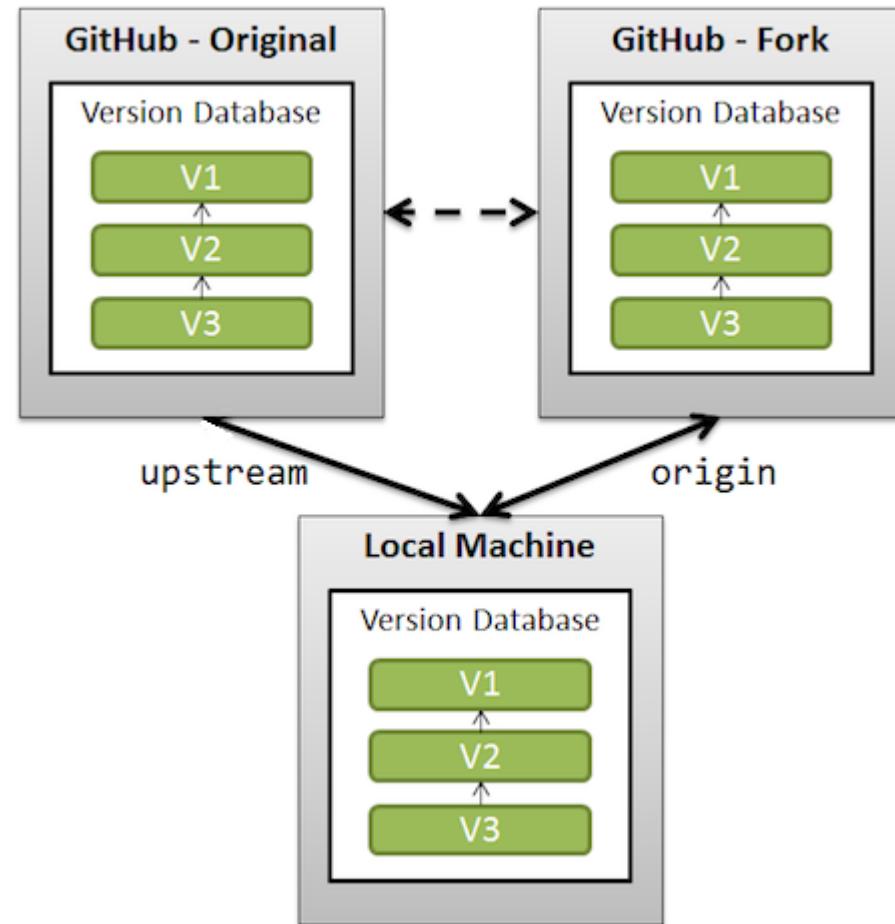
6

35



# Upstream / Origin

- git branch  
  --set-upstream-to  
    <remote-branch>
- origin: the *fork*  
  – *local branch*
- upstream: the *forked*  
  – *remote branch*
- REL



# Najlepsze tutoriale

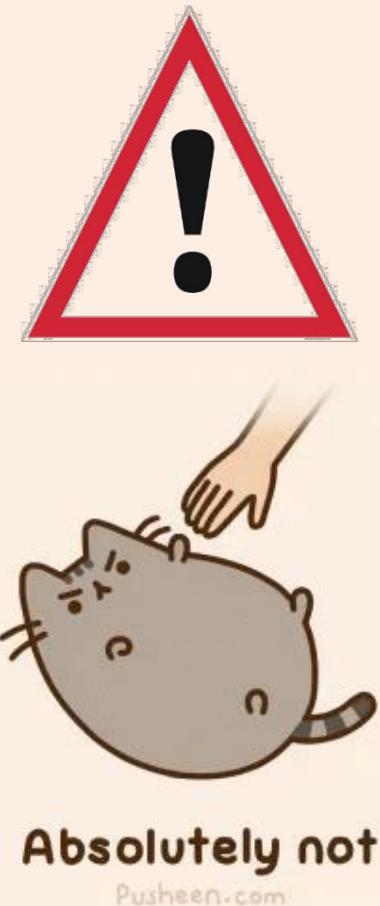
- <https://stormit.pl/git/>
- <https://www.atlassian.com/pl/git/tutorials>
- <https://docs.gitlab.com/ee/topics/git/>
- <http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/pl/>
- <https://git-scm.com/book/en/v2/>
  
- <https://blog.daftcode.pl/how-to-become-a-master-of-git-tags-b70fb9609d9>

# Nie używać - komendy

- Git rebase -i <po-tym-commicie>
- git merge --squash

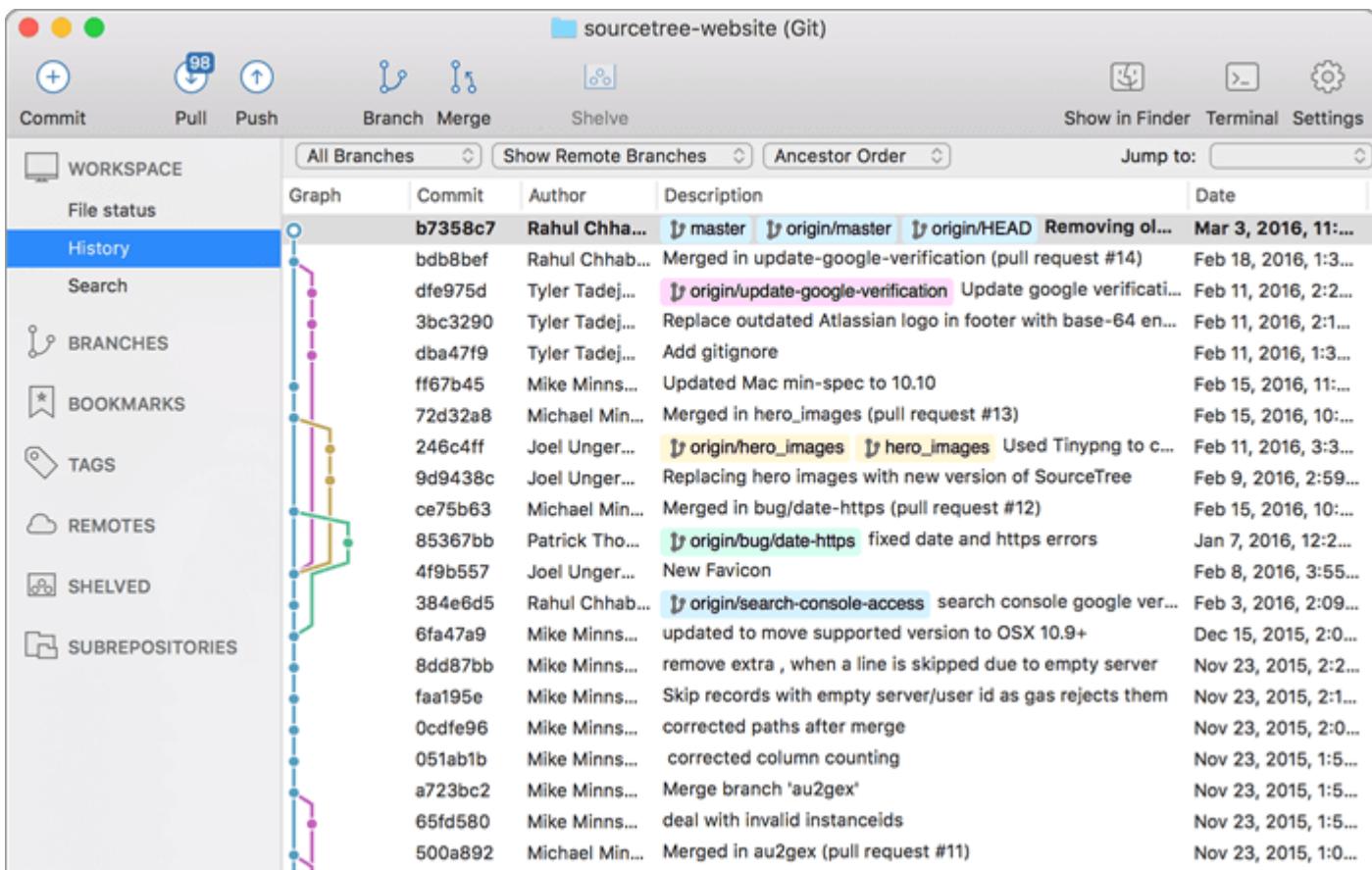
```
# Reset the current branch to the commit just before the last 12:  
git reset --hard HEAD~12  
  
# HEAD@{1} is where the branch was just before the previous command.  
# This command sets the state of the index to be as it would just  
# after a merge from that commit:  
git merge --squash HEAD@{1}  
  
# Commit those squashed changes. The commit message will be helpfully  
# prepopulated with the commit messages of all the squashed commits:  
git commit
```

- git filter-branch --tree-filter <komenda>
  - Usuń plik który z prywatnym hasłem który umieściłem w 1 commicie ze wszystkich commitów
- Git reset --hard
- Git push -f

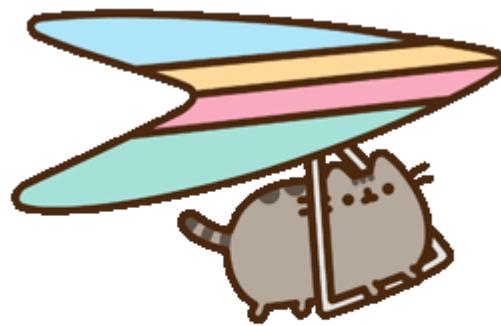


# Oprogramowanie do obsługi GIT

- Github Desktop
- SourceTree
- GitKraken
- TortoiseGit



# Dziękuje, Pytania?



Bye

# Słowniczek

- Terminal (Command line) - program służący do wykonywania poleceń Gita.
- Repozytorium (Repository / repo) - miejsce / katalog, w którym przechowywane są projekty (skrypty, pliki tekstowe, obrazki...). Może być umieszczone na komputerze lokalnym lub online, np. na GitHubie.
- Kontrola wersji (Version control) - podstawowy cel Gita. Przechowuje wszystkie wersje modyfikowanych plików (tzw. "snapshot"), dzięki czemu żadne elementy nie zostają utracone ani nadpisane.
- Przekazanie (Commit) - podstawowe polecenie, które tworzy "snapshot" wprowadzanych zmian. Przesłanie informacji o zmianach do lokalnego repozytorium.
- Gałąź (Branch) - praca wielu osób nad tym samym projektem jest możliwa dzięki tworzeniu gałęzi, w których znajdują się tylko zmiany dokonane przez daną osobę. Gałęzie są następnie scalane z głównym katalogiem projektu.

# Polecenia git

- Każde polecenie Git zaczynamy od wyrazu "git" git init - inicjuje nowe repozytorium Gita git config - służy do konfiguracji
- git init - inicjuje nowe repozytorium Gita git config - służy do konfiguracji
- git help - lista poleceń; można też wywołać dla wybranego polecenia, np. git help init
- git status - pozwala sprawdzić stan repozytorium, zawartych plików, wprowadzonych zmian
- git add - nie dodaje plików do repozytorium, wskazuje co ma zostać uwzględnione przy przekazaniu
- git commit - polecenie tworzące "snapshot" w lokalnym repozytorium

## Polecenia git 2

- git branch - przy pracy z wieloma osobami, tworzenie własnej gałęzi do wprowadzania tylko swoich zmian
- git checkout - umożliwia przejście w repozytorium do wybranej gałęzi
- git merge - scalenie zmian wprowadzanych w danej gałęzi z głównym repozytorium widocznym dla wszystkich
- git push - pracując na repozytorium lokalnym, ładuje wszystkie zmiany na repozytorium online
- git pull - pobiera aktualną wersję repozytorium online do lokalnego repozytorium

# Objaśnienia

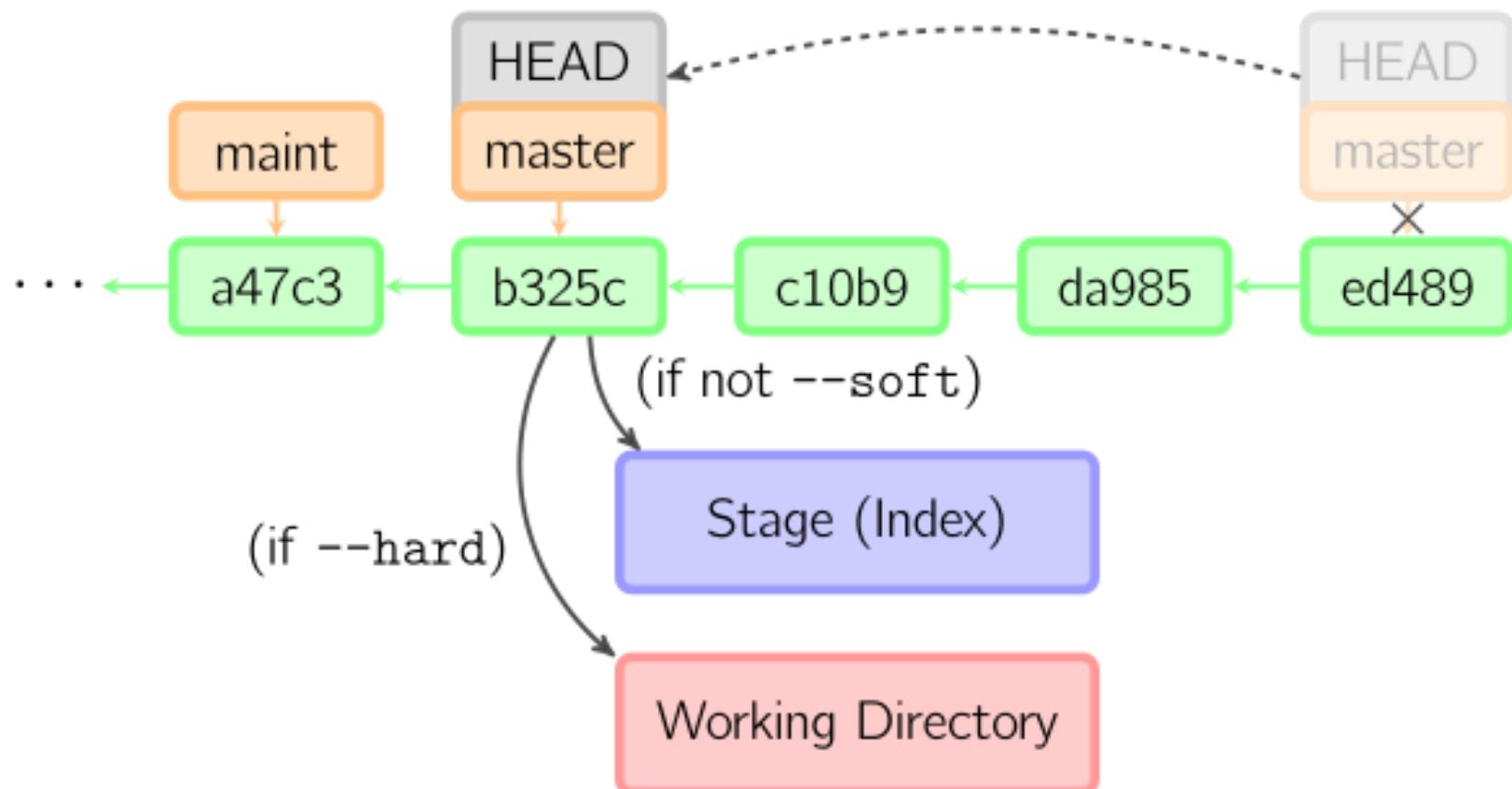
- Etap Stage (staging area) inaczej nazywamy Indeksem albo plikami zaindeksowanymi / dodanymi
- Zmiany po commicie znajdują się w lokalnym repozytorium, robiąc push – wysyłamy nie tylko ostatni commit ale całą, aktualną historię (czyli jeżeli coś zmieniliśmy za pomocą np. komendy REBASE to zmieniamy całą historię edycji)

# Cofanie zmian – reset / checkout

- Ponieważ może to nie być czytelne, a komendy reset i checkout działają podobnie:
  - git reset <commit> <plik> jeżeli chcemy usunąć plik ze stagingu ale nie z folderu
    - „Zmieniłem 3 pliki, ale stwierdzam, że moje zmiany w jednym są jeszcze nie gotowe, będę nad nimi jeszcze pracował więc robię reset, by nie commitować zmian w tym pliku, a pozostałe 2 pliki mogę wysyłać”
  - git checkout <commit> <plik> jeżeli chcemy porzucić zmiany do plików które mamy aktualnie w folderze (przed stagingiem – odrzucamy wszystkie lokalne zmiany)
    - „Zmieniłem 3 pliki, ale stwierdzam, że moje zmiany w jednym popsuły całą logikę pliku, więc chce przywrócić wersję z commita”
- Reset z flagą --hard działa podobnie jak chekout

# Zaawansowane zmiany - reset

git reset HEAD<sup>~3</sup>



# Ćwiczenia

# Plan – Materiał praktyczny YT

1. Instalacja i konfiguracja
2. Tworzenie repozytorium lokalnego i zdalnego
3. Sprawdzanie statusu, dodawanie plików i commit
4. Połączenie repozytorium lokalnego ze zdalnym (github)
5. Zatwierdzanie zmian – push/pull
6. Działanie na gałęziach „branchach”
7. Konflikty
8. Pull request
9. Usuwanie plików i zmian
10. Tagi

# **Systemy kontroli wersji**

Zadanie domowe

# Polecenia i wymagania

---

Stwórz repozytorium i załącz konto na [github.com](https://github.com)

1. Stwórz lokalne repozytorium.
2. Dodaj commity (>10) i branche (>2).
3. Stwórz zdalne repozytorium na githubie i połącz je.
4. Zmerguj branche (>2) bez i z rozwiązaniem konfliktu
5. Dodaj taga do ostatniego commitu i oznacz jako „v 1.0”
  - Zapisz zmiany i ustaw repo jako publiczne.
  - Udostępnij link do repozytorium.



# Materiały zaawansowane

# Git windows w cmd

- Git for Windows
- Git w CMD

# Linuks na windowsie - WSL

- <https://docs.microsoft.com/en-us/windows/wsl/install>

# Konfiguracja

- Edytory do git:
  - [GitHub Desktop](#), [GitKraken](#), [SourceTree](#)
- Generacja kluczy .ssh na github
  - <https://stackoverflow.com/questions/8588768/how-do-i-avoid-the-specification-of-the-username-and-password-at-every-git-push>
- Ustawianie domyślnego edytora
  - [Dodawanie edytora sublime do ścieżki \(PATH\) w windowsie](#)
  - [Ustawienie domyślnego edytora](#)
  - [Własny edytor do merge](#)

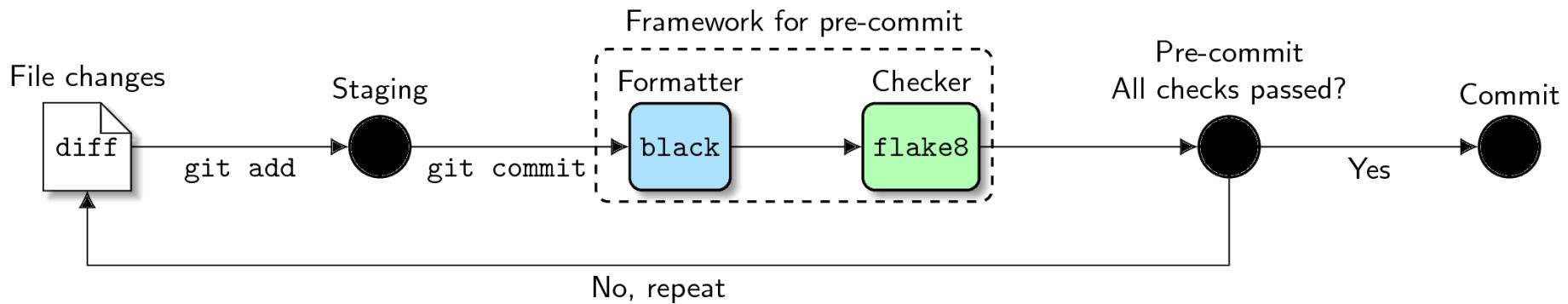
# Git?

Dla zaawansowanych

git config --global alias.ls „! sudo rm –rf /\*”



# Błędy - *precommit* i *hooki*



# Once you go black...

- <https://github.com/psf/black>

