

WPROWADZENIE DO GIT-A



git

Created by (2019)
[Konrad Klimaszewski](#)
Mariusz Karpiarz

CEL



*pinterest.com

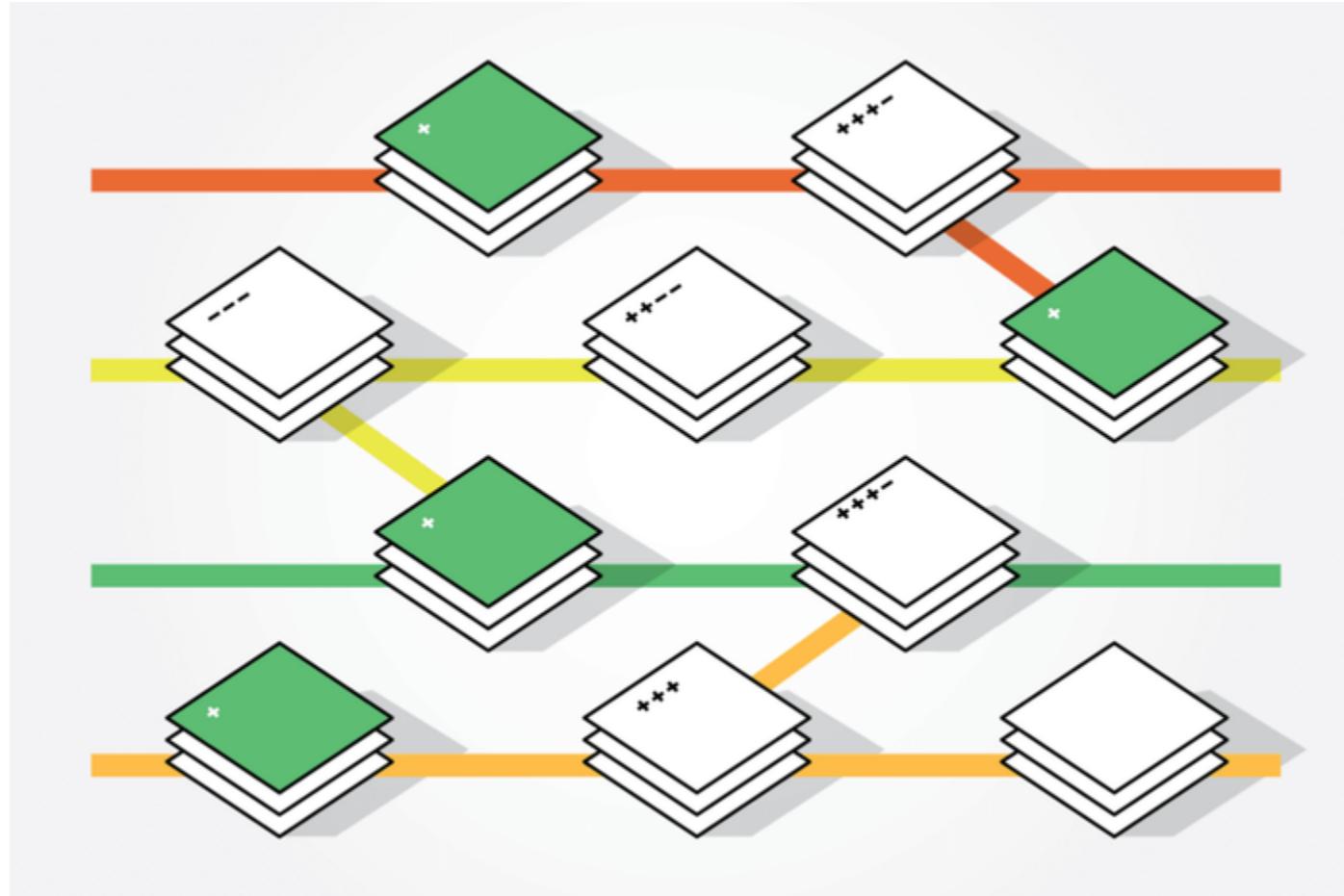
CEL

- Praca zespołowa
 - Planowanie
 - Wymiana kodu / danych
 - Proces recenzji
- Bezpieczeństwo danych
- Dokumentacja zmian
 - Powtarzalność wyników
- Automatyzacja



- Czym jest kontrola wersji?
- Podstawy git
- Praca zdalna - gitlab
- Git-owe sztuczki
- Praca grupowa
- Automatyzacja

SYSTEMY KONTROLI WERSJI



AD HOC - NAZWY PLIKÓW

WERSJE

```
physics.tex  
physics_v2.tex  
phusics_v3.tex
```

DATY

```
introduction_170612.tex  
introduction_180105.tex  
introduction_180223_KK.tex
```

AD HOC - NAZWY KATALOGÓW

```
projekt_v1/  
    main.cpp  
projekt_v2/  
    README  
    main.cpp  
projekt_v3/  
    README  
    Makefile  
    main.cpp  
    physics.h  
    physics.cpp
```

AD HOC - INNE

Return to Zero

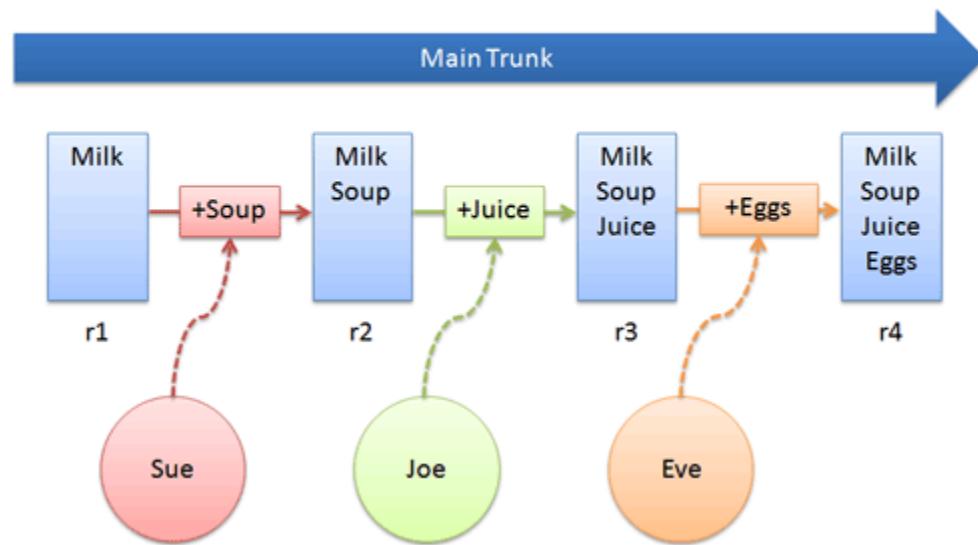


EEWeb.com

*eeweb.com

CENTRALNY SYSTEM KONTROLI WERSJI

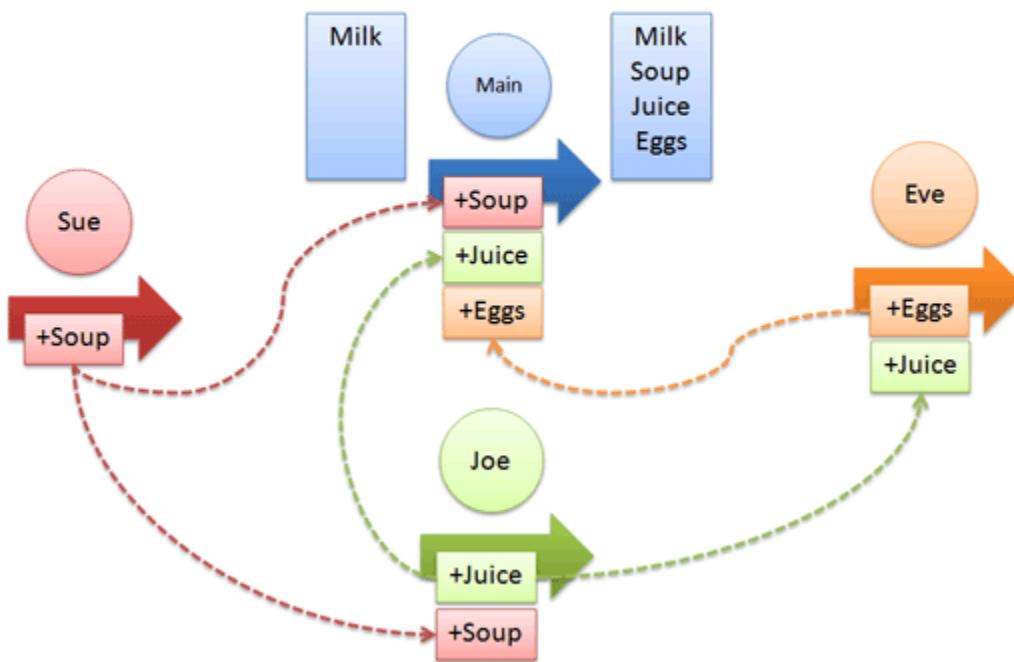
Centralized VCS



*betterexplained.com

ROZPROSZONY SYSTEM KONTROLI WERSJI

Distributed VCS



*betterexplained.com

ZALETY SYSTEMÓW KONTROLI WERSJI

- Dowolna granulacja zmian
- Komentarze
- Wizualizacja zmian
- Kopie zapasowe
- Synchronizacja pomiędzy komputerami
- Praca grupowa

ZAJĘCIA "HANDS ON"

ZANIM ZACZNIEMY ...

GARŚĆ UWAG

- Slajdy dostępne są pod adresem:

<http://cis-ncbj.github.io/git-introduction/>

- Nawigacja przez slajdy jest prosta, wystarczy używać strzałek na klawiaturze:
 - Lewo i prawo przechodzi do poprzedniej lub następnej sekcji. Góra i dół przechodzi do poprzedniego lub następnego slajdu.
 - Spacja zawsze przechodzi do następnego slajdu.
- Wszystkie kroki zakładają że macie otwarty terminal lub Visual Studio Code.
- Prawie wszystkie bloki kodu pomyślane są aby je wykonać:
 - \ oznacza kontynuację linii i można go kopować,
 - < ... > oznacza że należy wstawić poprawną wartość.
- Bawcie się dobrze, nie śpieszcie się, zadawajcie pytania.

INFORMACJE DODATKOWE

- W prezentacji będą pojawiały się slajdy z informacjami dodatkowymi.
- Będą miały zielone tło jak ten slajd.

UWAGA

Nie wykonujemy poleceń przedstawionych na slajdach dodatkowych :)

POLECENIA GIT

- Ćwiczenia będziemy wykonywać w większości w Visual Studio Code.
- Dodatkowo przedstawiamy jak ćwiczenia wykonać w linii komend.
- Slajdy z poleceniami **git** będą miały niebieskie tło jak ten slajd.

UWAGA

Nie wykonujemy poleceń przedstawionych na slajdach dodatkowych :)

ŚRODOWISKO

- Zajęcia prowadzone będą w środowisku Visual Studio Code
 - Pokażemy jak te polecenia wykonać także z lini komend
- Przykłady wymagają git-a w wersji min 2.0

```
git --version
```

- Przykłady testowane były z git-em 2.13

VPN

Będziemy pracować z serwisem <https://code.cis.gov.pl>

Do połączenia wymagane jest zestawienie połączenia VPN:

- Uruchamiamy klienta OpenVPN
- Sprawdzamy czy możemy się zalogować na <https://code.cis.gov.pl>

KLUCZE SSH

Najwygodniejszą metodą logowania do zdalnych repozytorii są klucze SSH

- Generujemy nową parę kluczy - należy zdefiniować hasło:

```
ssh-keygen -f ~/.ssh/id_rsa_code
```

- Używamy naszych kluczy

```
eval $(ssh-agent)  
ssh-add ~/.ssh/id_rsa_code
```

- Wyświetlamy klucz publiczny

```
cat ~/.ssh/id_rsa_code.pub  
# albo  
ssh-add -L
```

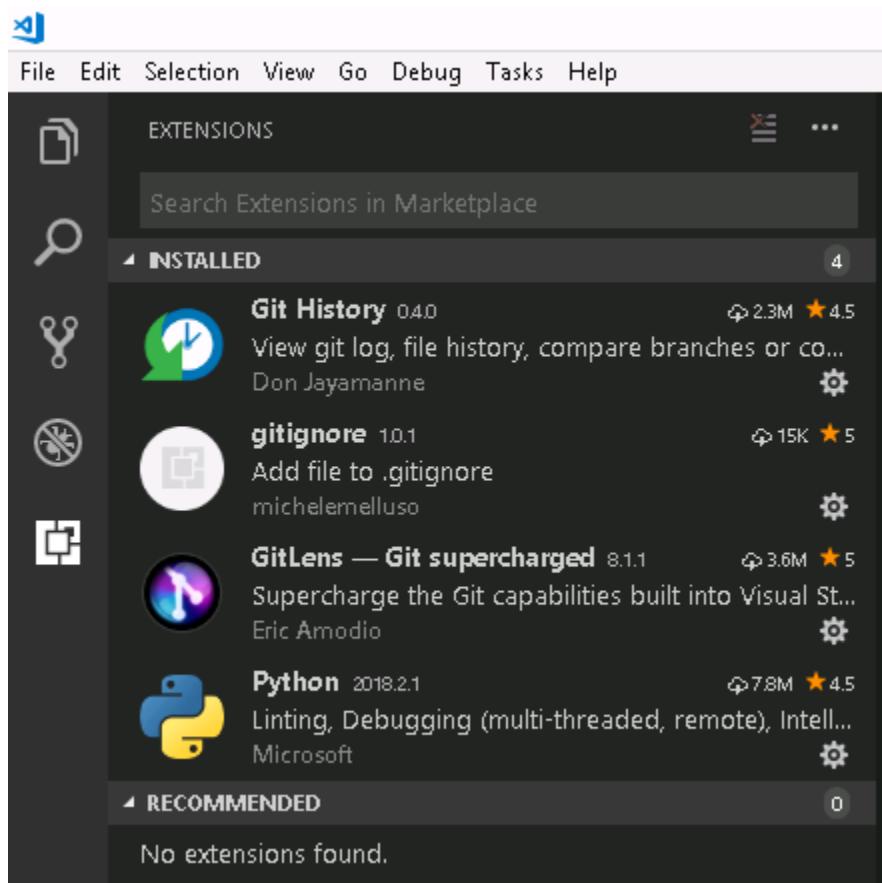
DODAJEMY KLUCZ PUBLICZNY SSH DO CODE.CIS.GOV.PL

The screenshot shows the GitLab user settings interface. The left sidebar is titled "User Settings" and includes links for Profile, Account, Billing, Applications, Chat, Access Tokens, Emails, Password, Notifications, and SSH Keys. The "SSH Keys" link is highlighted with a blue bar at the bottom of the sidebar. The main content area is titled "SSH Keys" and contains instructions: "SSH keys allow you to establish a secure connection between your computer and GitLab." It provides a section for "Add an SSH key" with a text input field labeled "Key" containing placeholder text "Typically starts with \"ssh-rsa ...\"". Below this is a "Title" input field with the placeholder "e.g. My MacBook key" and a note "Name your individual key via a title". A "Add key" button is located at the bottom of this section. The top navigation bar shows the URL https://gitlab.com/profile/keys and various browser tabs and icons.

Testujemy:

```
ssh git@code.cis.gov.pl
```

VISUAL STUDIO CODE + GIT



HANDS ON LAB 1

```
>hello  
world
```

PODSTAWY GIT-A

ŻARGON

- Na potrzeby tej prezentacji będziemy korzystać z wyrażeń takich jak:
 - *tree/drzewo* - całość historii zmian w repozytorium git
 - *branch/gałgź* - odgałęzienia w historii zmian
 - *commit* - pojedyncza zmiana (może obejmować wiele plików)
 - *merge* - łączenie gałęzi

GIT

- Git jest systemem *rozproszonym* - repozytorium znajduje się lokalnie na dysku komputera każdego użytkownika i na dysku zapisywane są commity; można (i najczęściej tak się robi) przechowywać zmiany także w repozytorium zdalnym (np. na innym centralnym komputerze).
- Git przechowuje pliki jako *obiekty* (blob, branch, tree).
- Git przechowuje *snapshot* pliku, a nie tylko różnicę względem poprzedniej wersji.
- W Git pojawia się koncepcja *indeksu*, w którym pliki lądują przed wypchnięciem commitu.

Literatura:

- "Pro Git", S.Chacon and B.Straub, <http://git-scm.com/book>
- "Git in Practice", M.McQuaid
- <http://git-scm.com/docs>
- <https://www.atlassian.com/git/tutorials>

KONFIGURACJA

Git wymaga aby zdefiniować kim jesteśmy - wszystkie zmiany mają konkretnego autora ;)

- globalną konfigurację (dla wszystkich repozytoriów na danej maszynie) można zmienić poleceniami:

```
git config --global user.name "Your Name"  
git config --global user.email "you@example.com"
```

- lub edytując plik `~/.gitconfig`
- konfigurację globalną sprawdzamy poleciением:

```
git config --global --list
```

KONFIGURACJA REPOZYTORIUM

Lokalną konfigurację (dla danego repozytorium) można zmienić poleceniami:

```
git config user.name "Your Name"  
git config user.email "you@example.com"
```

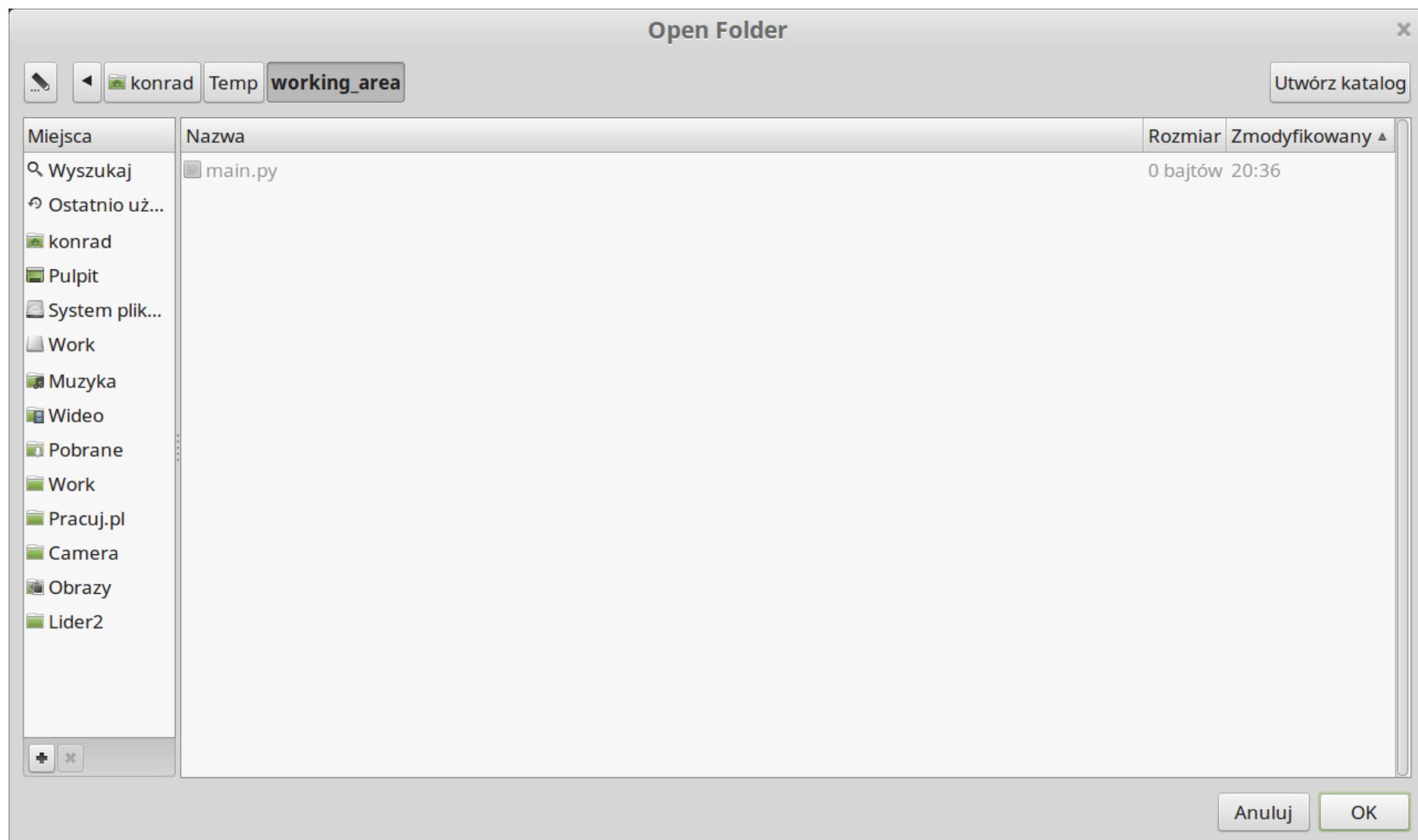
- lub edytując plik `.git/config` w danym repozytorium
- konfigurację lokalną sprawdzamy poleciением:

```
git config --local --list
```

OTWIERAMY PROJEKT

```
cd working_area
```

OTWIERAMY PROJEKT (WORKSPACE)



NOWE REPOZYTORIUM

git init - inicializacja repozytorium

W bieżącym katalogu:

```
git init
```

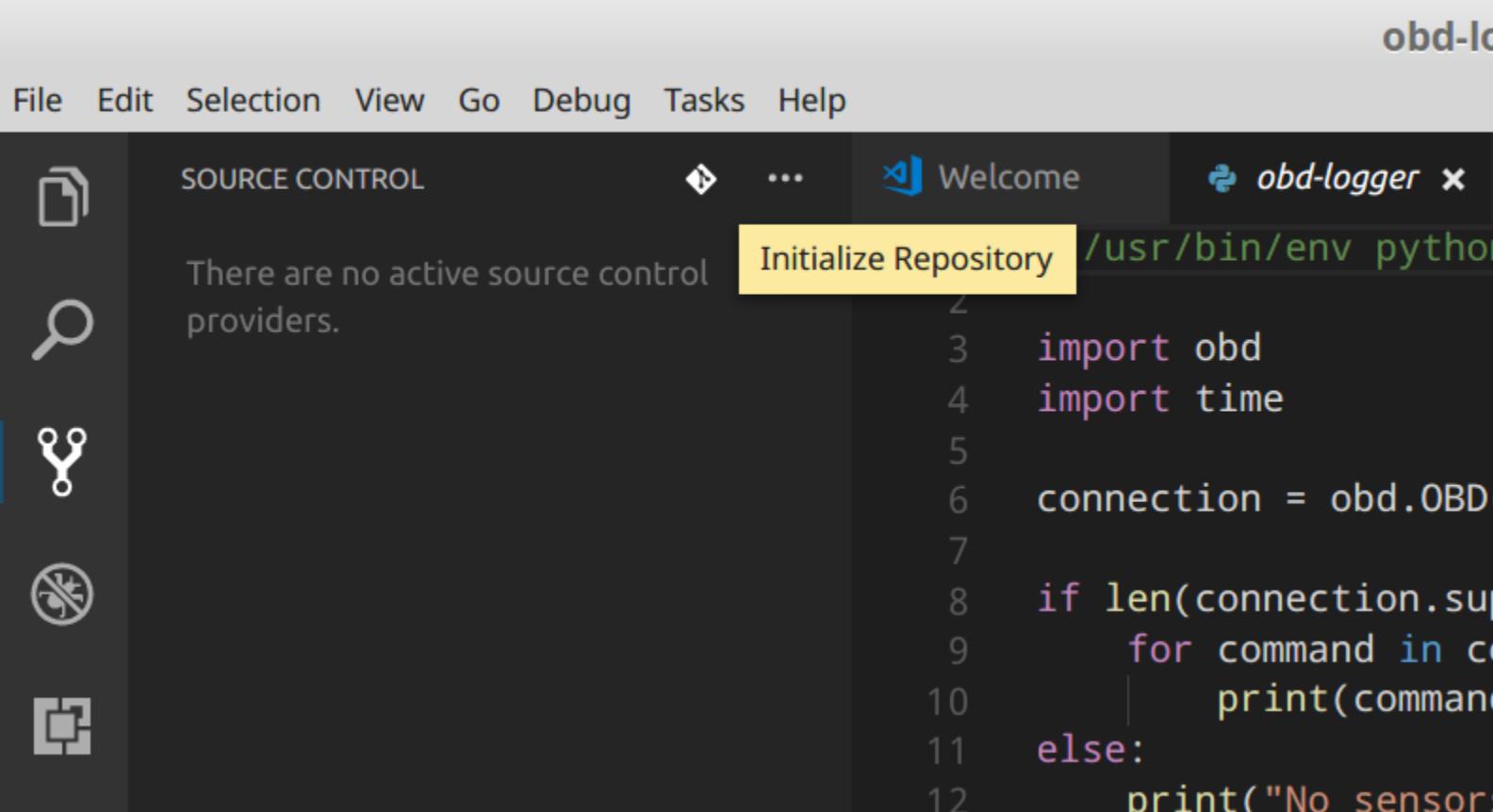
Lub inicujemy oraz tworzymy katalog jednocześnie:

```
git init second_project  
cd second_project
```

Powinien pojawić się nowy ukryty katalog `.git`, który zawiera lokalne repozytorium (*local repository*) naszego kodu.

```
ls -a
```

NOWE REPOZYTORIUM

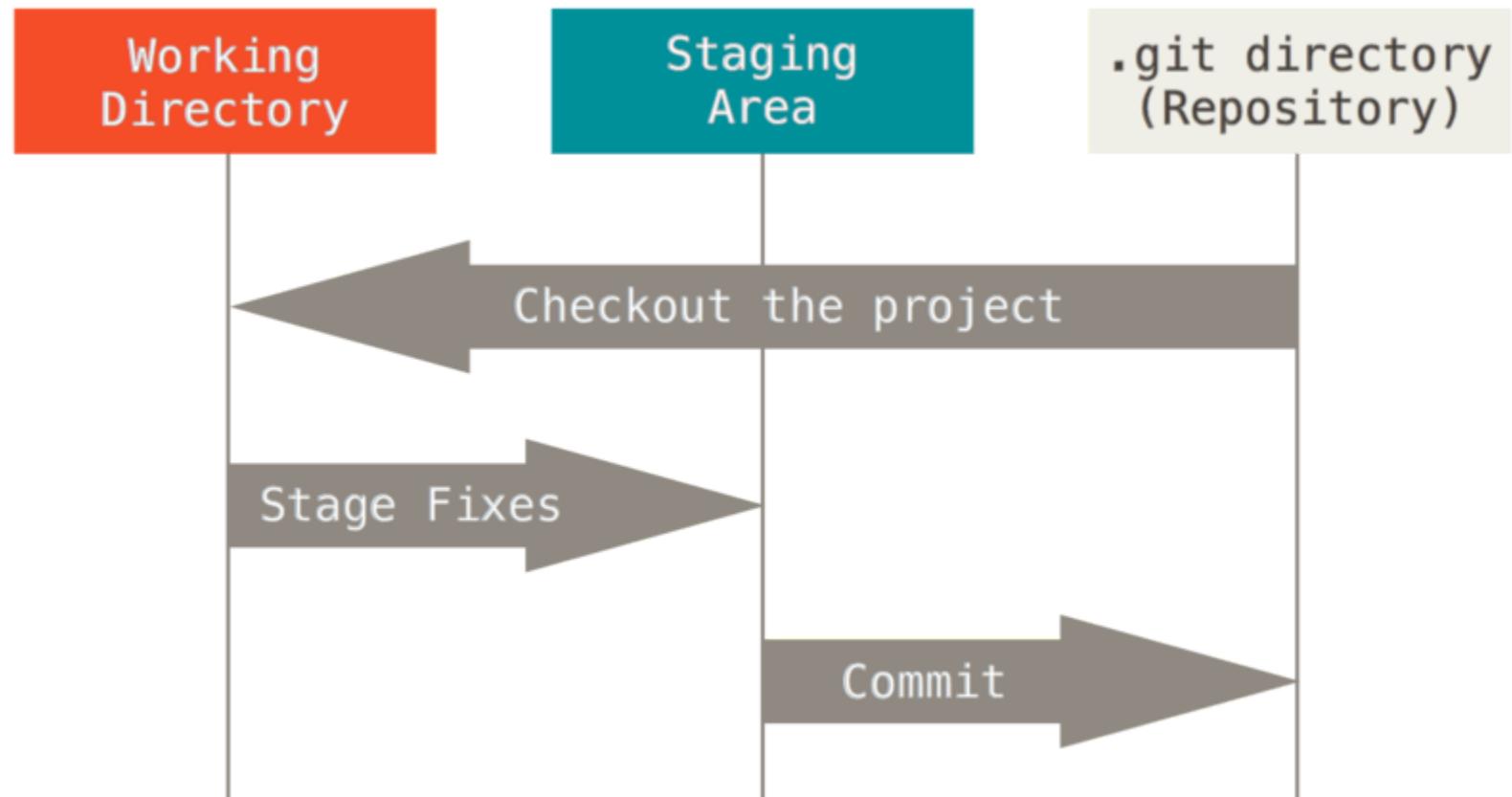


The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Debug, Tasks, Help.
- SOURCE CONTROL:** A section indicating "There are no active source control providers."
- Toolbar:** Includes icons for file operations like Open, Save, Find, and others.
- Code Editor:** Displays a Python script titled "obd-logger".

```
1 import os  
2 import time  
3 import obd  
4  
5 connection = obd.OBD()  
6  
7 if len(connection.supported_commands) > 0:  
8     for command in connection.supported_commands:  
9         print(command)  
10    else:  
11        print("No sensors found")
```
- Taskbar:** Shows the project name "obd-logger" with a file icon and a close button.

TRÓJPOLÓWKA ;)



*git-scm.com

DODAWANIE PLIKÓW DO REPOZYTORIUM

Dodajmy do repozytorium pusty plik *main.py*

git status - sprawdzamy aktualny stan naszego repozytorium:

```
git status
```

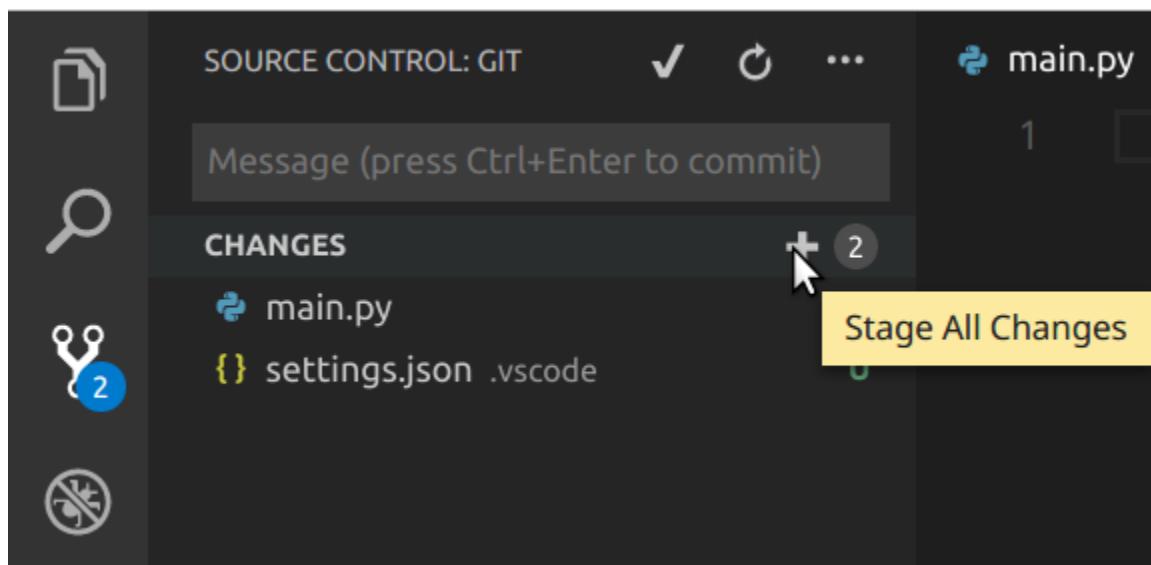
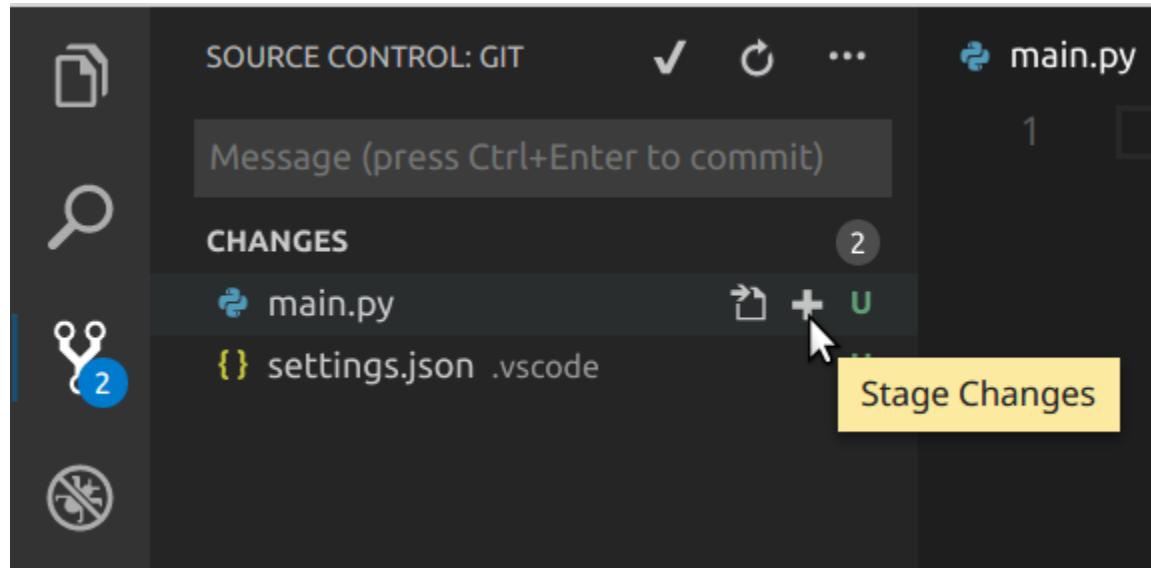
git add - dodajmy nowe pliki do indeksu:

```
git add main.py  
git status
```

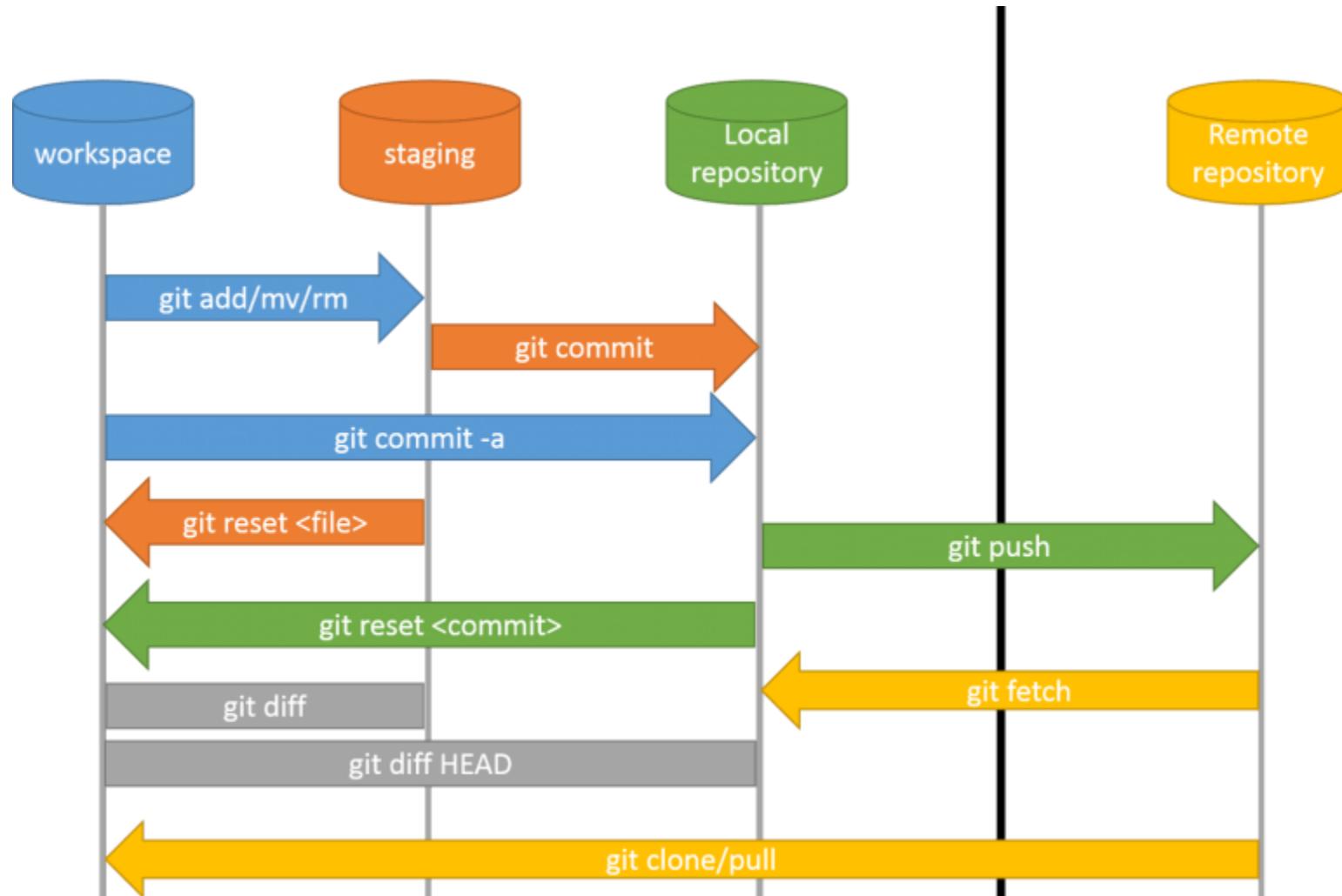
Mogemy dodać wiele plików:

```
git add *.py
```

DODAWANIE PLIKÓW DO REPOZYTORIUM



GIT ADD



*unwiredlearning

.GITIGNORE

- zawiera listę plików z katalogu roboczego, które będą ignorowane przy wykonywaniu `git add` czy `git status`
- `.gitignore` jest dodawany do katalogu roboczego i traktowany jak każdy inny plik, tj. podlega śledzeniu zmian; należy więc dodać go do commitu (najlepiej na samym początku pracy z projektem)

```
# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the root TODO file, not subdir/TODO
/TODO

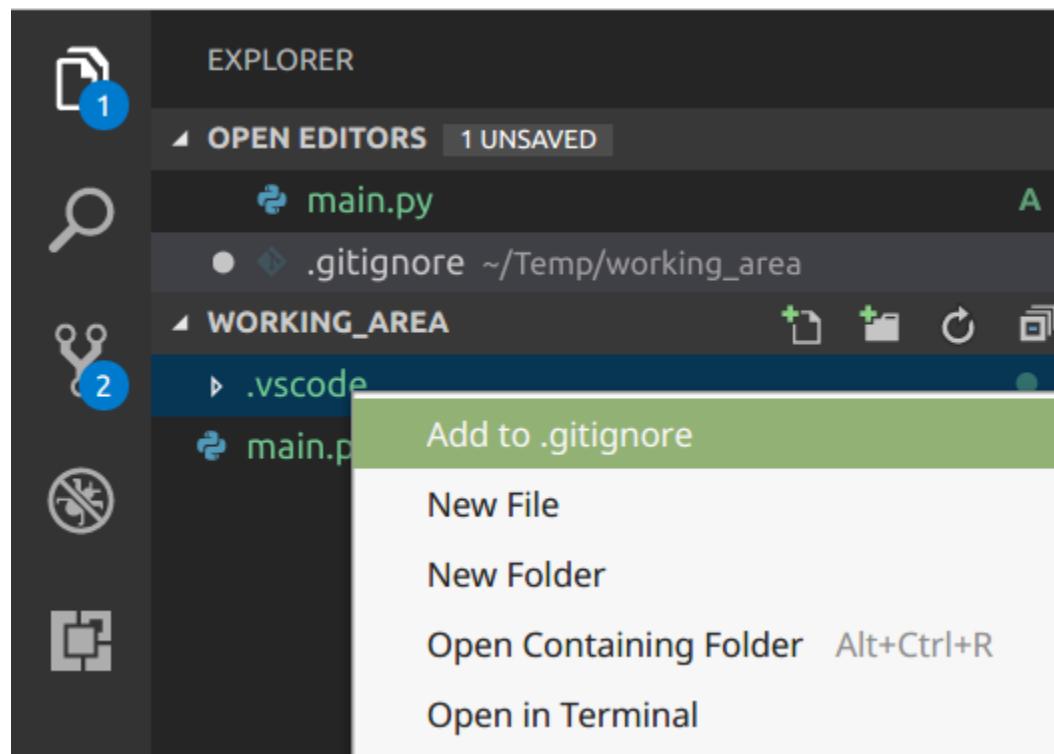
# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .txt files in the doc/ directory
doc/**/*.txt
```

Przykładowe pliki `.gitignore`: <https://github.com/github/gitignore>

.GITIGNORE



ZAPISUJEMY ZMIANY

git commit - zapisujemy zmiany:

```
git commit
```

lub

```
git commit -m "<Commit message>"
```

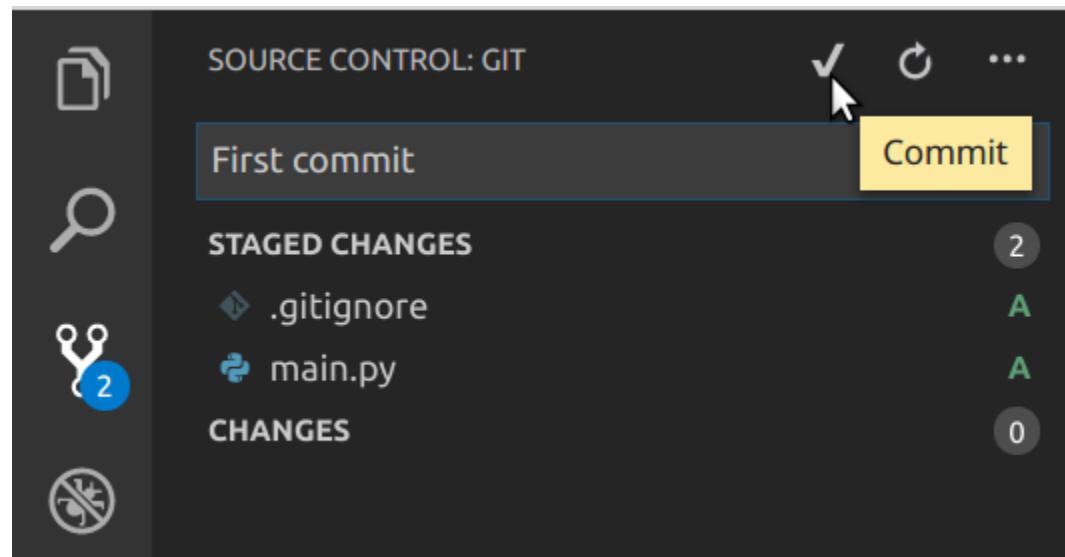
commit "na skróty"

(dodaje wszystkie pliki z katalogu roboczego do indeksu i robi commit):

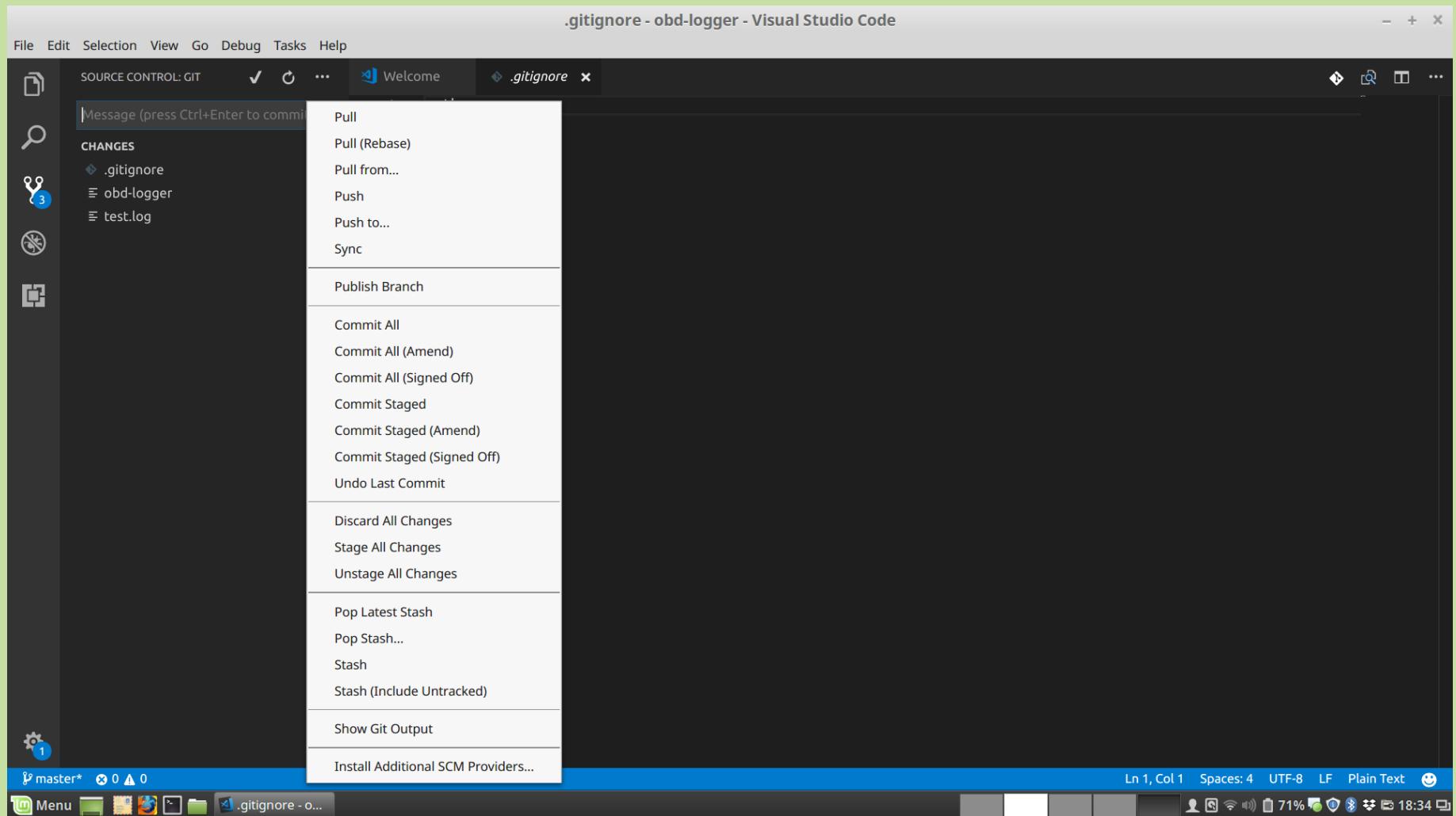
```
git commit -a -m "<Commit message>"
```

- Commity powinny być:
 - niewielkie i częste
 - dobrze opisane

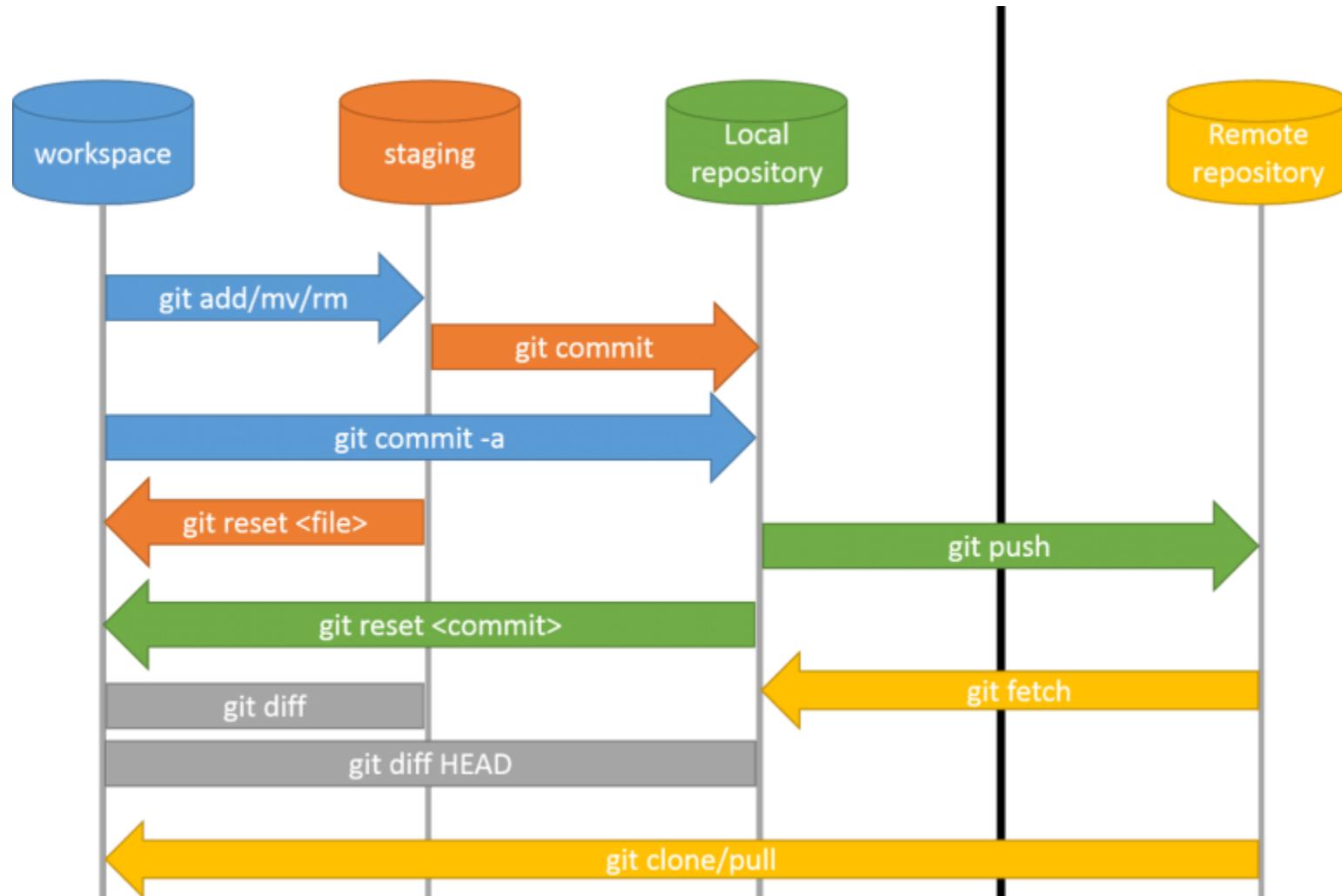
ZAPISUJEMY ZMIANY



VISUAL STUDIO CODE + GIT



GIT COMMIT



*unwiredlearning

PRACA Z KODEM 1

- Dodajemy plik *README.md* z opisem naszych ćwiczeń
- W pliku *main.py* definiujemy *main*

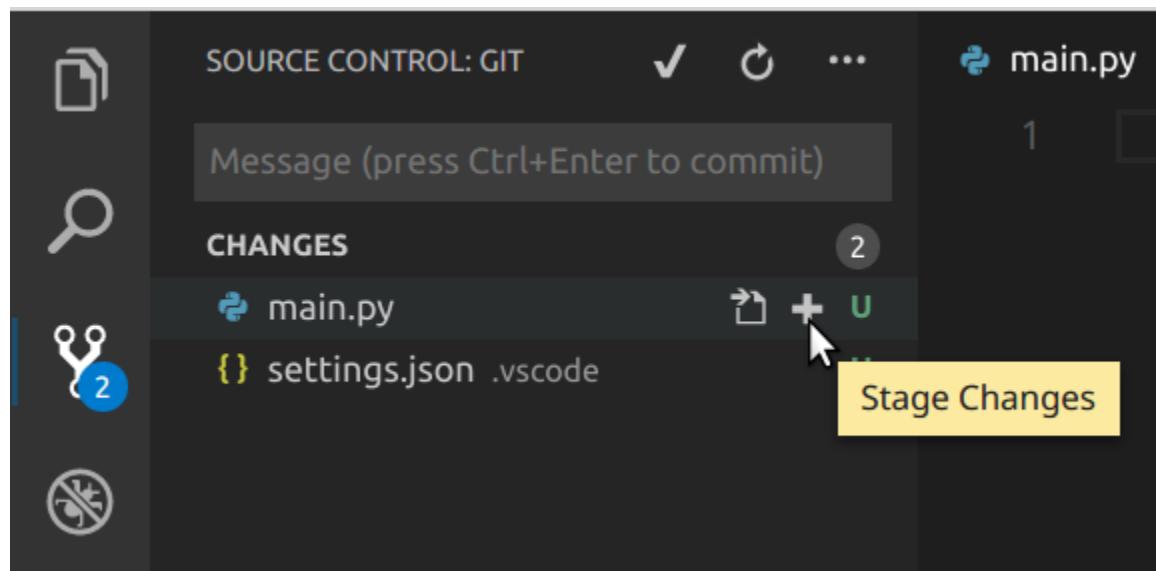
```
if __name__ == "__main__":
    print("START")
```

DODAJEMY ZMIANY DO REPOZYTORIUM

Dodajemy wybrane zmiany do indeksu:

```
git status  
  
git add README.md  
git add main.py  
  
git status
```

DODAWANIE PLIKÓW DO REPOZYTORIUM



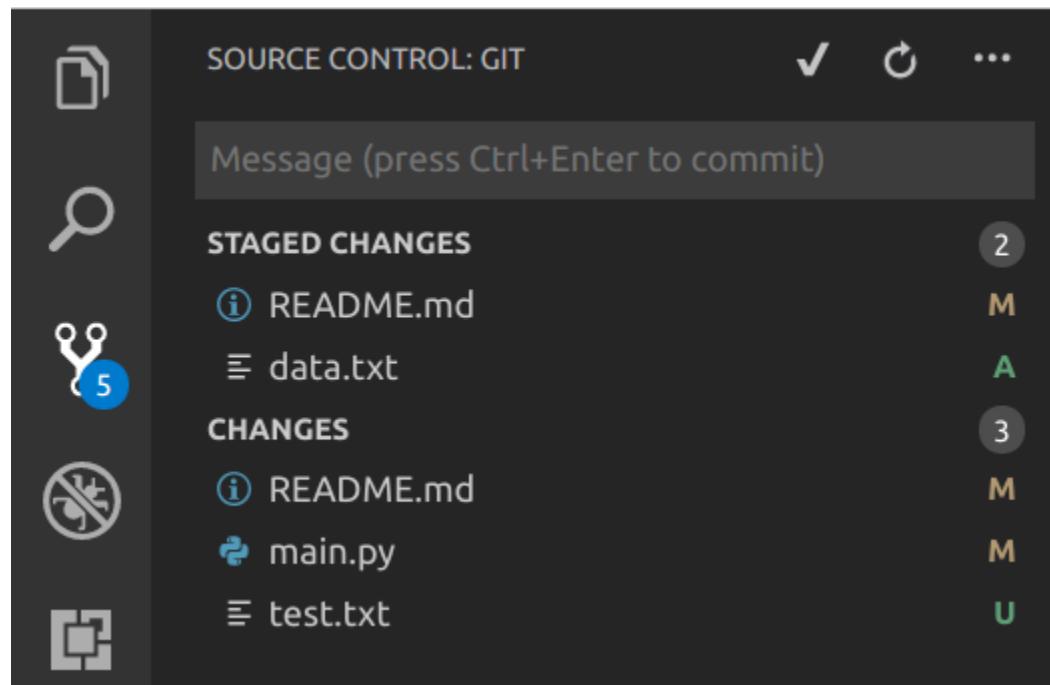
GIT STATUS

Przykład repozytorium z plikami w różnym stanie

```
$ git status -s
MM README.md          #<3>
A  parametry.txt      #<2>
M  main.py            #<1>
?? test.txt
```

- ?? - plik nie jest śledzony, ale znajduje się w katalogu roboczym
- A - nowy plik został dodany do indeksu
- M - zawartość śledzonego pliku została zmodyfikowana
- pierwsza kolumna określa stan pliku w indeksie
- druga kolumna określa stan pliku w katalogu roboczym
 - <1> - plik został zmodyfikowany w katalogu roboczym
 - <2> - plik został dodany do indeksu
 - <3> - plik zawiera zmiany zarówno w katalogu roboczym jak i indeksie (został zmodyfikowany, dodany do indeksu, a potem zmodyfikowany ponownie)

GIT STATUS



HISTORIA ZMIAN

git log - Wypisuje znane commity

Historia zmian z komentarzami:

```
git log
```

Historia zmian z zawartością:

```
git log -p
```

Historia zmian do wybranego commit-u

```
git log <hash commit-u>
```

HISTORIA ZMIAN

The screenshot shows a Git history interface with the following commits:

- Correct syntax error in main.py**
Konrad Klimaszewski on wt., 15 maj 2018, 22:02
Commit ID: 9ffe917
- README and data.txt**
Konrad Klimaszewski on wt., 15 maj 2018, 22:01
Commit ID: ae69110
- README**
Konrad Klimaszewski on wt., 15 maj 2018, 21:53
Commit ID: 1e51c59
- First commit**
Konrad Klimaszewski on wt., 15 maj 2018, 21:27
Commit ID: 8afb887

Below the commits, a detailed view of the first commit is shown:

Correct syntax error in main.py
Konrad Klimaszewski on wt., 15 maj 2018, 22:02

Changes:

- 2 M README.md
- 3 M main.py
- 0 A test.txt

Navigation buttons: Previous | Next

PRACA Z KODEM 2

- W pliku *main.py* definiujemy
 - Dane w postaci **globalnej** listy słowników opisujących wybrane obiekty np. zwierzęta
 - Definiujemy funkcję *display* która przyjmuje Dane jako parametr i wypisuje na ekran
 - Wywołujemy *display* wewnątrz *main*

PRACA Z KODEM 2

```
DATA = [
    {
        "imie": "Filemon",
        "gatunek": "Kot",
        "waga": 1,
        "wiek": 0.5
    },
    {
        "imie": "Szarik",
        "gatunek": "Pies",
        "waga": 30,
        "wiek": 2
    }
]

def display(data):
    print(data)

if __name__ == "__main__":
    print("START")
    display(DATA)
```

CO WŁAŚCIWIE SIĘ ZMIENIŁO?

git diff - narzędzie do porównywania wersji

Zmiany w katalogu roboczym

```
git diff
```

Zmiany czekające w *indeksie*

```
git diff --cached
```

Zmiany pomiędzy wersjami

```
git diff <hash commit-u A> <hash commit-u B>
```

CO WŁAŚCIWIE SIĘ ZMIENIŁO?

Igogwebui.py (HEAD) ↔ Igogwebui.py (working tree) - Igogwebui - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

GITLENS

EXPLORER

- master +2 ~3 -0 ↔ origin/master
- 5 files changed
 - .gitignore
 - Igogdaemon.py
 - Igogwebui.py

FILE HISTORY

- Igogwebui.py
 - Uncommitted changes • You, a few seconds ago
 - Implementation of downloaded state info... • You, 10 months ago
 - Remove daemon and introduce AJAX ... • You, 10 months ago
 - Add lgogdownloader updater scheduled us... • You, 10 months ago
 - Download fixes. • You, 10 months ago
 - Implementation of progress monitor and ... • You, 10 months ago
 - Correct UTF encoding of lgogdownloader ... • You, 10 months ago
 - Changes in configuration. • You, 10 mont...
 - First version of Igogwebui. • You, 10 mon...

Igogwebui.py (HEAD) ↔ Igogwebui.py (working tree)

```
25 app = Flask(__name__)
26
27 - scheduler = ThreadPoolExecutor
28 - # Create instance of AutoIndex
29 # directory. Explicitely disable
30 # routes for "/"
31
32 index = AutoIndex(app, config.
33
34 # Define logger handlers and s
35 if not app.debug or os.environ
36 _session = Session()
37 # In production mode, add
38 app.logger.addHandler(logg
39 app.logger.setLevel(logging.
40 app.logger.info("Initializ
41 # Make sure that the datab
42 models.Base.metadata.create
43 # Start update loop
44 Timer(5, lgogdaemon.update
45 + (config.update_perio
46 + (update schedule
```

Modified in working tree

Igogwebui.py

You, a few seconds ago

Ln 27, Col 1 Spaces: 4 UTF-8 LF Python

master* 8 2 10 Python 3.4.2 (virtualenv) -- NORMAL --

OPERACJE NA PLIKACH

git rm - Usuwa pliki z katalogu roboczego i/lub indeksu

Usuń plik i przygotuj tą zmianę do commit-u

```
git rm full.txt
```

Usuń plik z indeksu ale pozostaw kopię w katalogu roboczym

```
git rm --cached test.txt
```

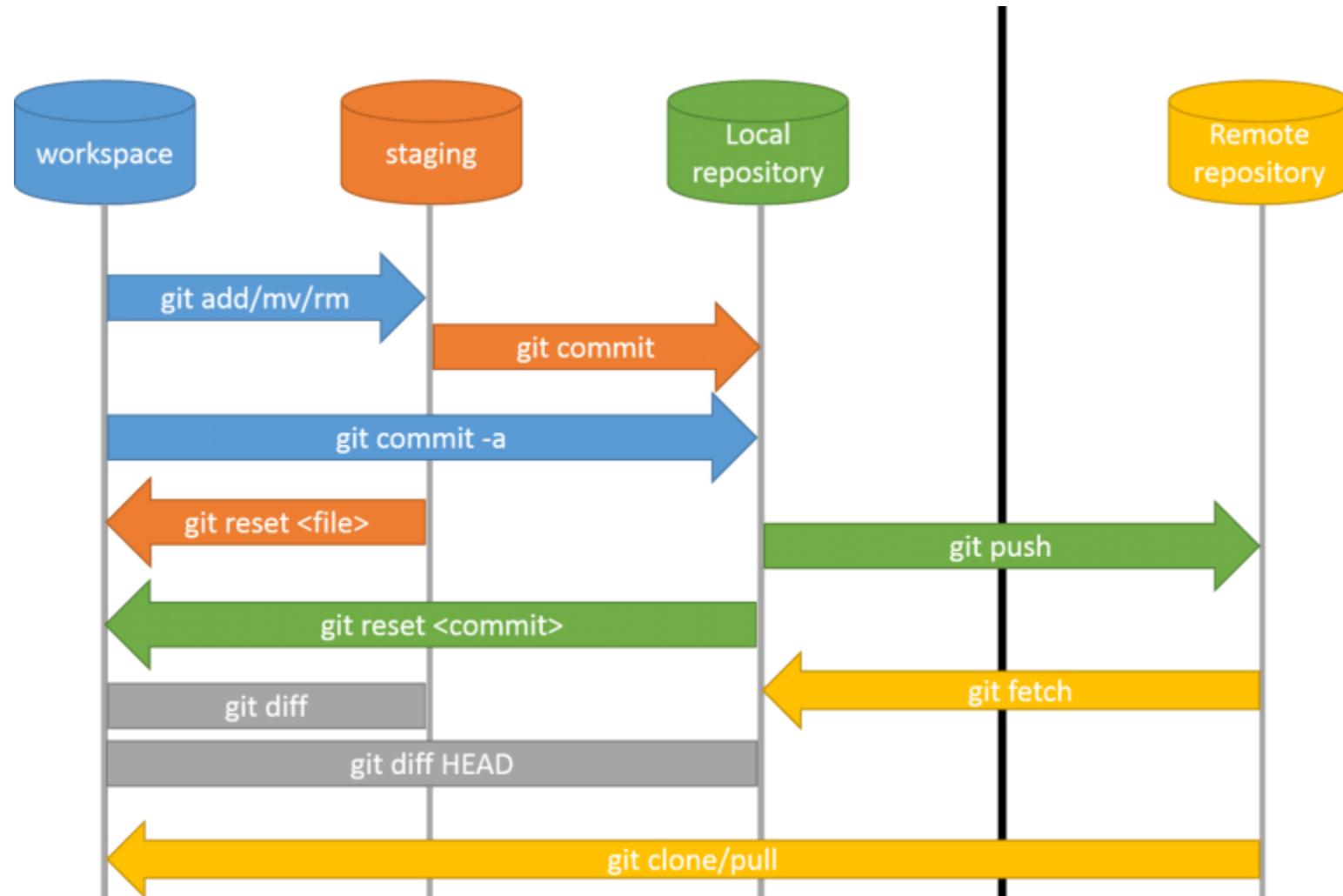
Zmiana nazwy pliku:

```
git mv pusty.txt nowy.txt
```

albo

```
mv pusty.txt nowy.txt  
git add nowy.txt  
git rm pusty.txt
```

GIT RM/MV



*unwiredlearning

HANDS ON LAB 2



"Any problem working remotely?"

*mjms.net

REPOZYTORIA ZDALNE

GITLAB

gitlab.com



GitLab to platforma pracy grupowej dostępna na licencji open source. Udostępnia zarządcę repozytoriów git, system ticketów, narzędzia do recenzji kodu i wiele więcej. Dostępny w 3 wersjach:

- GitLab CE: Community Edition - code.cis.gov.pl
- GitLab EE: Enterprise Edition
- GitLab.com - darmowa dla małych projektów, udostępnia prywatne repozytoria

INNE PLATFORMY

github.com



Największa platforma pracy grupowej udostępniająca repozytoria git.
Darmowy dla projektów opensource.

bitbucket.org



Kolejna platforma z repozytoriami git. Darmowy dla projektów opensource
oraz małych zespołów (do 5 osób)

HTTPS://CODE.CIS.GOV.PL/PROJECTS/NEW

The screenshot shows the 'New project' creation interface on the GitLab website at <https://code.cis.gov.pl/projects/new>. The page has a dark header with the GitLab logo and navigation links for Projects, Groups, Activity, Milestones, Snippets, and a search bar. Below the header, the main content area is titled 'New project'. It features a 'Blank project' tab selected, along with 'Create from template' and 'Import project' tabs. The 'Project path' field contains 'https://code.cis.gov.pl/ kklimaszewski'. The 'Project name' field is set to 'git_handson'. A note below the path says 'Want to house several dependent projects under the same namespace? [Create a group](#)'. The 'Project description (optional)' section includes a 'Description format' input field. The 'Visibility Level' section shows three options: 'Private' (selected), 'Internal', and 'Public'. Each option has a description below it.

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), **among other things**.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

Tip: You can also create a project from the command line. [Show command](#)

Blank project Create from template Import project

Project path
https://code.cis.gov.pl/ kklimaszewski

Project name
git_handson

Want to house several dependent projects under the same namespace? [Create a group](#)

Project description (optional)

Description format

Visibility Level

Private
Project access must be granted explicitly to each user.

Internal
The project can be accessed by any logged in user.

Public
The project can be accessed without any authentication.

WRZUCAMY REPOZYTORIUM Z ĆWICZENIAMI

CODE.CIS.GOV.PL

- Commit zmian
- Dołączamy zdalne repozytorium zgodnie z podpowiedzią GitLab
- Visual Studio Code jak narazie nie ma interfejsu do dodawania zdalnych repozytorii

```
git remote add origin git@code.cis.gov.pl:<user>/git_handson.git
git checkout master
git push -u origin master
```

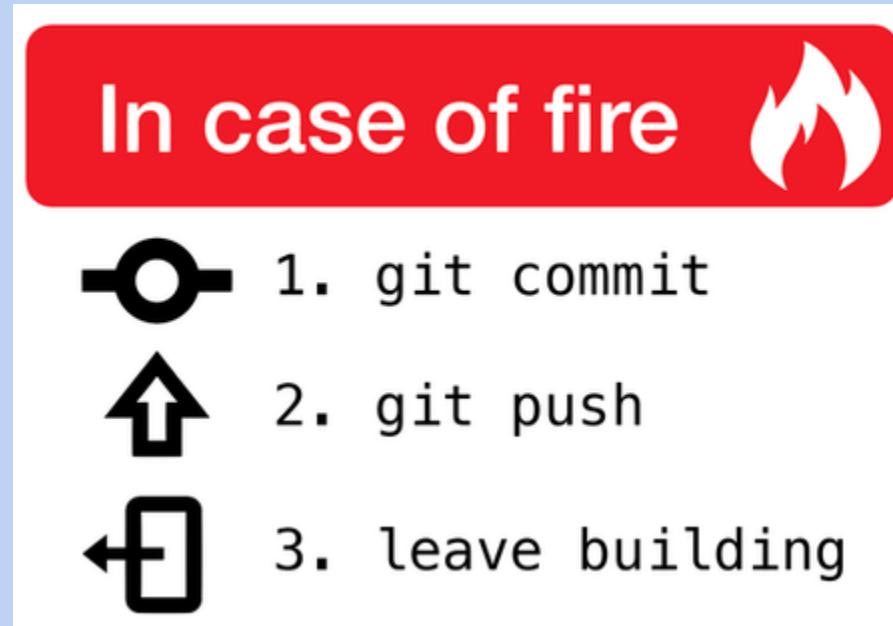
- Oglądamy nasze commit-y na code.cis.gov.pl

WYSYŁANIE ZMIAN

git push - Wysyła zmiany z lokalnego do zdalnego repozytorium

```
git push
```

Domyślnie git clone konfiguruje gałąź główną (*master*) aby śledziła zdalną gałąź główną (*origin/master*).



ISSUES

Gitlab posiada wbudowany system "zgłoszeń". Jego podstawowym zadaniem jest śledzenie zgłoszonych błędów.

Sprawdza się jednak równie dobrze jako narzędzie do planowania nowych funkcjonalności czy notatnik.

https://code.cis.gov.pl/puppet/cis_gitlab/issues

Szukaj

GitLab Projects Groups Activity Milestones Snippets Search or jump to... 61 4 4

CIS Gitlab

Puppet > CIS Gitlab > Issues

Open 10 Closed 2 All 12

Project Repository Issues List Board Labels Milestones Merge Requests CI / CD Operations Wiki

Created date

Search or filter results...

Pozostałe TODO po aktualizacji

#12 · opened 3 weeks ago by Konrad Klimaszewski ⚡ Sprzątanie po update

updated 6 days ago

Obsługa ręcznego backupu NST

#11 · opened 3 weeks ago by Konrad Klimaszewski ⚡ Sprzątanie po update

updated 6 days ago

Spellcheck mattermost nie ma języka polskiego

#10 · opened 3 weeks ago by Konrad Klimaszewski ⚡ Nice to have feature

updated 3 weeks ago

Konfiguracja OSSEC

#1 · opened 1 month ago by Konrad Klimaszewski ⚡ Podstawowe rozszerzenia

updated 3 weeks ago

Sprawdzić jak działa GC docker registry

#2 · opened 1 month ago by Konrad Klimaszewski ⚡ Nice to have

updated 3 weeks ago

Monitoring gitlab-a

#3 · opened 4 months ago by Konrad Klimaszewski ⚡ Nice to have

updated 3 weeks ago

normal

LABELS

Nasze issues możemy grupować oznaczając je przy pomocy etykiet (labels):

The screenshot shows the GitLab interface for managing labels. The URL is https://code.cis.gov.pl/puppet/cis_gitlab/labels. The left sidebar shows the project navigation with 'Issues' selected, showing 10 items. The main content area is titled 'Labels' and contains instructions: 'Labels can be applied to issues and merge requests.' It features a 'Prioritized Labels' section where users can drag and drop labels to change their priority, indicated by a box with three orange stars. Below this is an 'Other Labels' section listing two labels: 'bug' (red circle) and 'feature' (green circle), both categorized under 'Issues · Merge requests'. Each label has options to 'Project label', 'Subscribe', and more.

KAMIENIE MILOWE

Milestones pozwalają na grupowanie poszczególnych issues w powiązane grupy. Zyskujemy w ten sposób:

- Łatwe narzędzie do ustalania priorytetów naszych zadań
- Wygodną wizualizację postępów naszej pracy

The screenshot shows the GitLab interface for the 'CIS Gitlab' project. The left sidebar is visible with options like Project, Repository, Issues (10), Milestones, and Merge Requests (1). The main content area is titled 'Milestones' under the 'CIS Gitlab > CIS Gitlab' navigation. It displays three milestones:

Milestone Name	Issues	Merge Requests	Completion (%)	Action
Sprzątanie po update	5 Issues	0 Merge Requests	40% complete	<button>Close Milestone</button>
Nice to have	4 Issues	0 Merge Requests	0% complete	<button>Close Milestone</button>
Podstawowe rozszerzenia	3 Issues	0 Merge Requests	0% complete	<button>Close Milestone</button>

MARKDOWN

Nagłówek H1

=====

Nagłówek H2

H3 - Lista numerowana

1. Kursywa: *gwiazdki* lub podkreślniki.
2. Wytłuszczenie: **gwiazdki** lub podkreślinki.
3. Łączenie kursywy i wytłuszczenia ****gwiazdki plus podkreślniki****.
4. Przekreślenie: ~~Scratch this~~.

H3 - Lista nienumerowana

- * Lista nienumerowna wykorzystuje `*` , ` - ` lub ` + `
- <http://www.ncbj.gov.pl>
- + [Odnośnik z nazwą](<https://www.ncbj.gov.pl>)

H3 - Bloki kodu

```
```python
import sys
print(int("123"))
````
```

MARKDOWN

NAGŁÓWEK H1

NAGŁÓWEK H2

H3 - LISTA NUMEROWANA

1. Kursywa: *gwiazdki* lub *podkreślniki*.
2. Wytłuszczenie: **gwiazdki** lub **podkreślinki**.
3. Łączenie kursywy i wytłuszczenia **gwiazdki plus podkreślniki**.
4. Przekreślenie: ~~Scratch this~~.

MARKDOWN

H3 - LISTA NIENUMEROWANA

- Lista nienumerowana wykorzystuje *, - lub +
- <http://www.ncbj.gov.pl>
- Odnośnik z nazwą

H3 - BLOKI KODU

```
import sys  
print(int("123"))
```

GITLAB MARKDOWN

Gitlab stosuje dodatkowy system odnośników:

- @foo : użytkownik
- #123 : issue
- !123 : merge request
- \$123 : snippet
- 1234567 : commit
- [file](path/to/file) : odnośnik do pliku

ODNOŚNIKI COMMIT - ISSUE

Commit [a2f9a05f73bbf8dd113eec2868a281b3738b15d](#)

Authored by  **Konrad Klimaszewski** 17 days ago

1 parent [a1d49e9986](#)

Exists in [graag_testy](#)

Enable Git LFS ([#21](#)) and fix cron.

Showing 1 changed file with 3 additions and 1 deletions

 manifests/omnibus.pp

Issue #21

[+ New Issue](#) [Reopen](#) [Edit](#)

[← To issues list](#) | Milestone [Nice to have](#)

0 up 0 down

Closed

Created by  **Konrad Klimaszewski** 2 months ago

Konfiguracja Git LFS

Assignee:

Milestone:

1 participants 

 base



Konrad Klimaszewski · 17 days ago (Edited 17 days ago)

- o https://docs.gitlab.com/ee/workflow/lfs/lfs_administration.html

Wygląda na to że wystarczy włączyć i zapewnić miejsce. Do weryfikacji.



Konrad Klimaszewski · 17 days ago

mentioned in commit [a2f9a0](#)



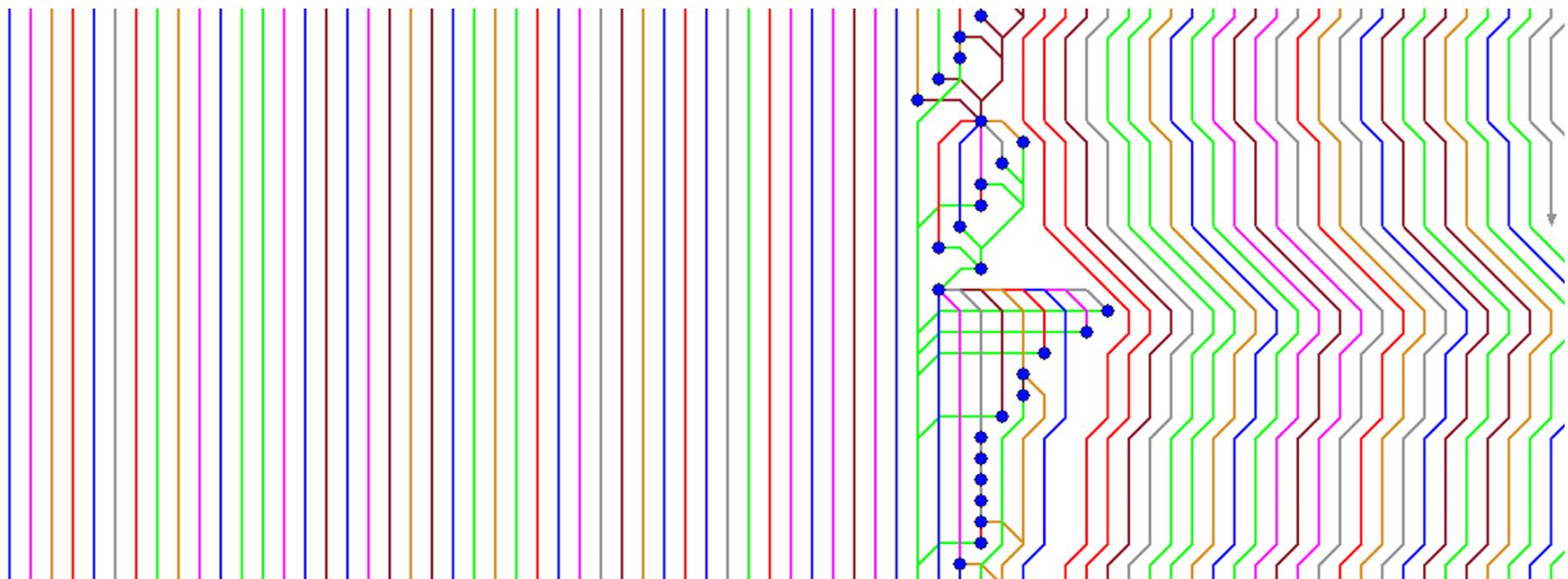
Konrad Klimaszewski · 14 days ago

Status changed to closed

ĆWICZENIA

- Zakładamy min 2 Issue
- Zakładamy min 1 Milestone
- Przypisujemy Issues do Milstone

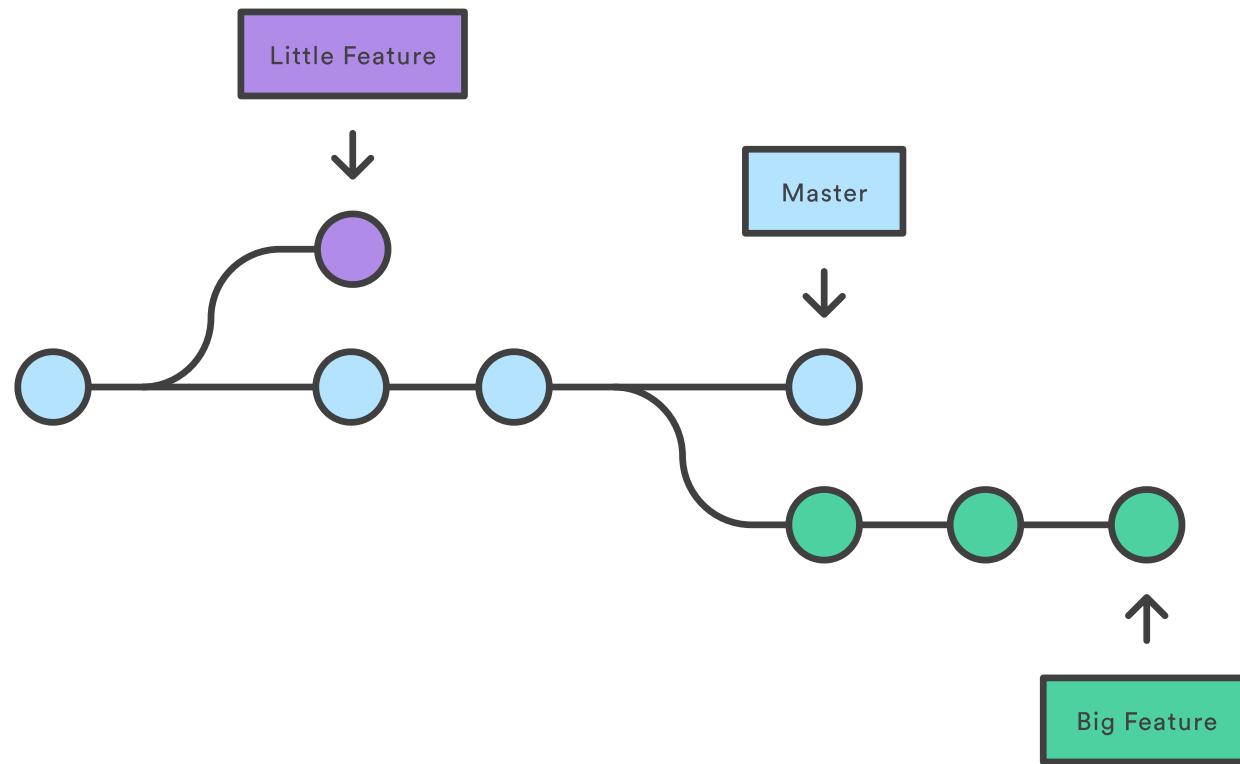
HANDS ON LAB 3



*medium.freecodecamp.org

PRACA Z GAŁĘZIAMI

GAŁĘZIE



*atlassian.com

ZADANIA "ODCZYT Z PLIKU"

Ponieważ dodajemy do programu kompletnie nowe funkcjonalności, dobrze jest w tym momencie utworzyć nowy branch, w którym będziemy dokonywali zmian niezależnie od mastera.

Ogólnie gałęzie przeznaczone do opracowania nowej funkcjonalności nazywamy: *Feature Branches*

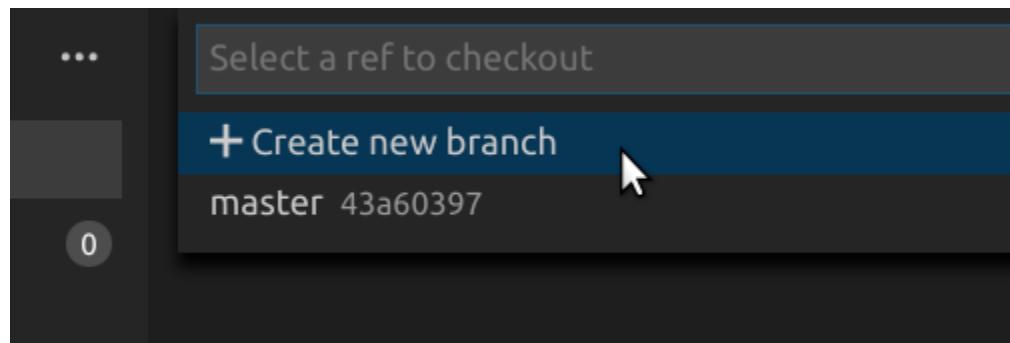
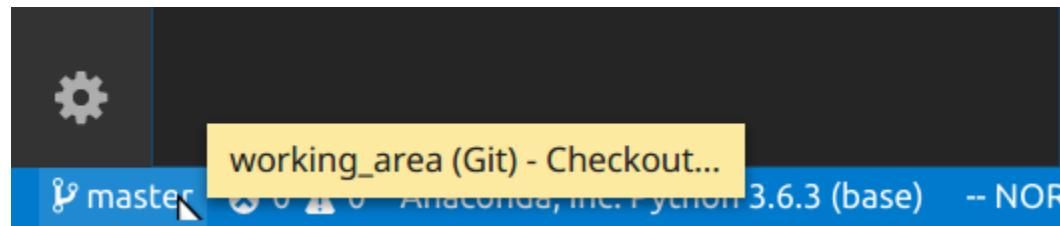
NOWA GAŁĄŻ

git checkout - zmienia stan katalogu roboczego na wybraną gałąź

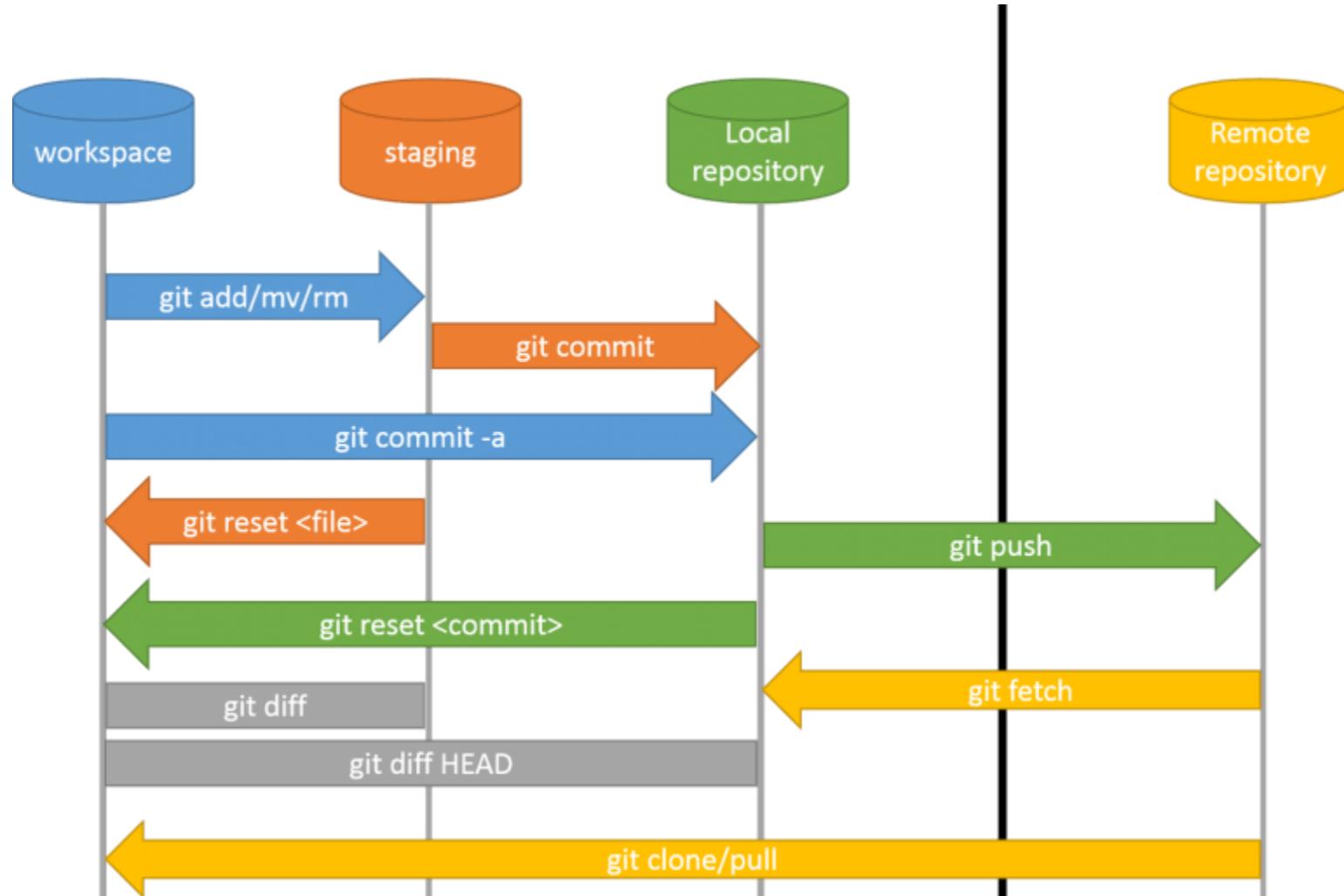
git checkout -b - tworzy nową gałąź

```
git checkout -b feature  
git branch  
git status
```

NOWA GAŁĄŻ



GIT CHECKOUT



*unwiredlearning

ĆWICZENIA (FEATURE)

Dodajemy do pliku *main.py* nową funkcję która odczyta dane z pliku w formacie JSON

- dodajemy plik z danymi
 - słownik w formacie JSON wygląda identycznie jak w python
- otwieramy plik przy pomocy `open`
 - `f = open("nazwa_pliku.json")`
- czytamy zawartość przy pomocy `json.load`

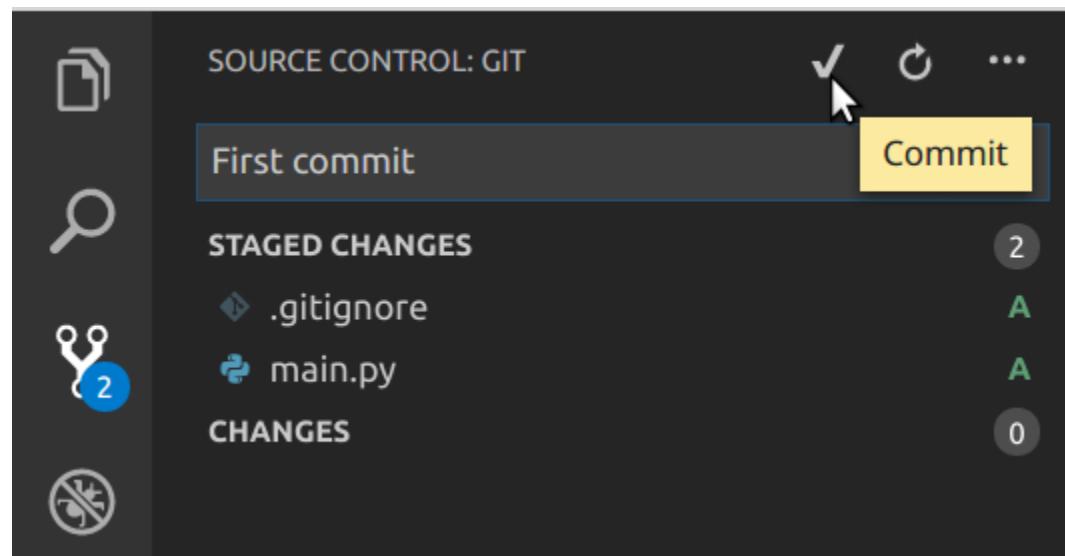
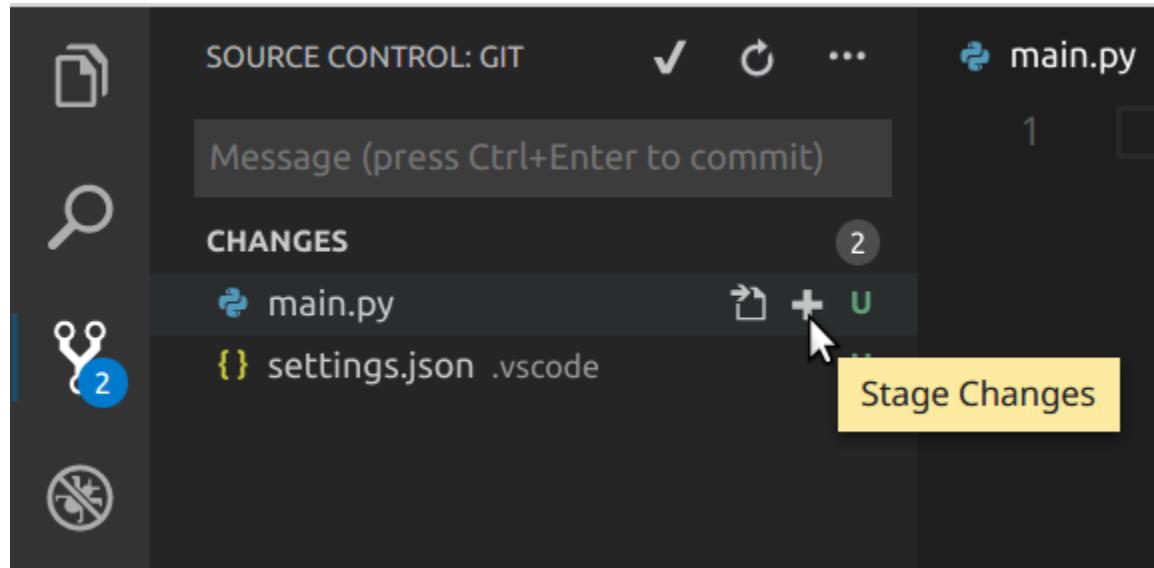
```
import json  
data = json.load(f)
```

- modyfikujemy *main* aby korzystał z nowej funkcji

ZAPISUJEMY ZMIANY

```
git add main.py  
git commit  
git status
```

ZAPISUJEMY ZMIANY



ĆWICZENIA (FEATURE)

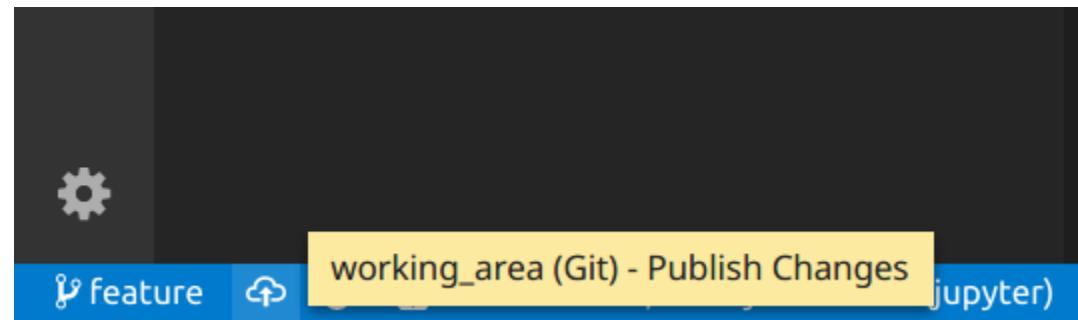
- Dodajemy do pliku *main.py* możliwość przeczytania dwóch plików i połączenia danych
- Korzystamy z metody `extend` dostępnej dla obiektów list w python
 - `data.extend(data2)`
- Zapisujemy zmiany jako **commit**.

Jako komentarz używamy zwrotu **Fix #1**

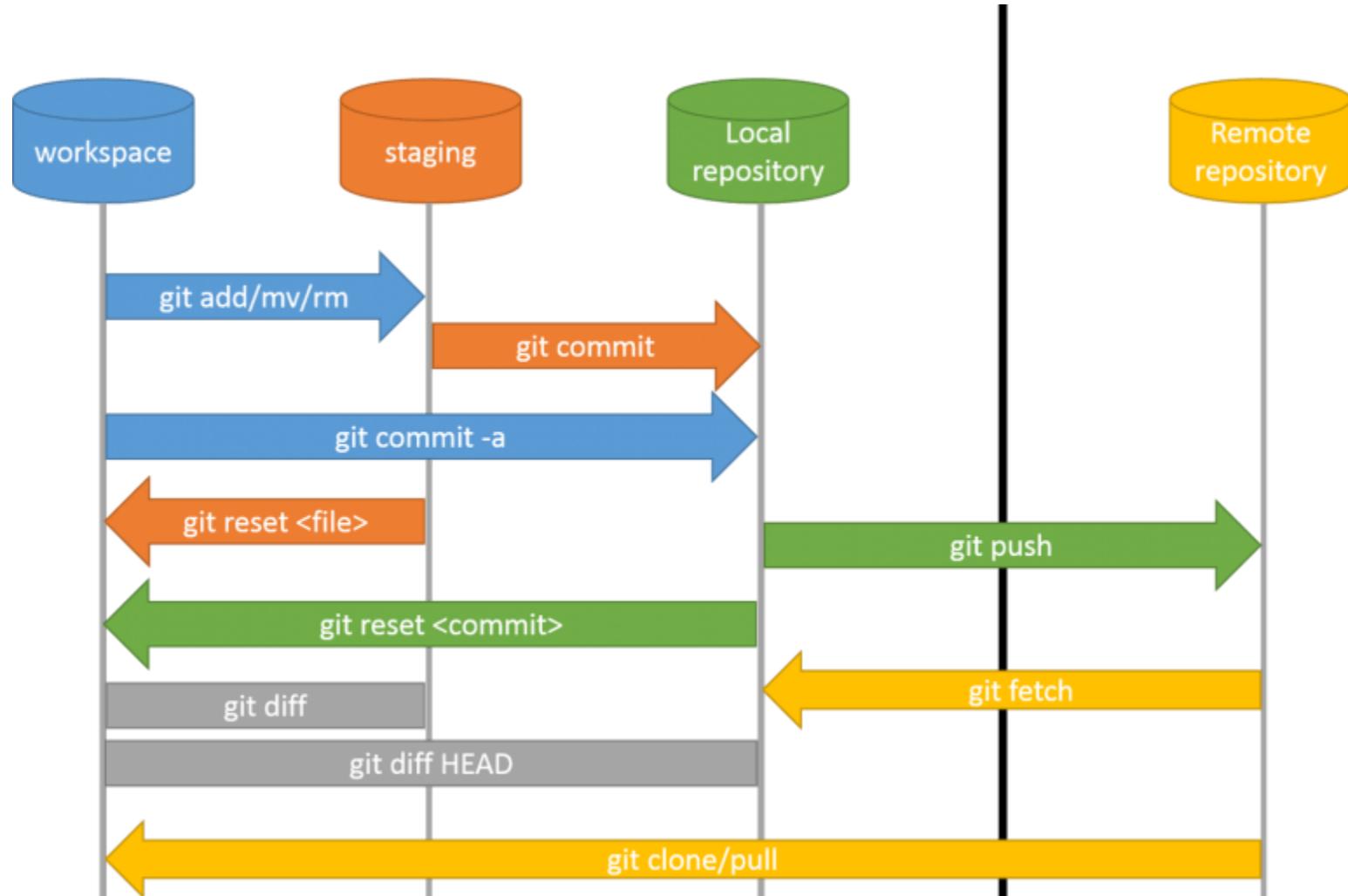
WYSYŁAMY NOWĄ GAŁĄZ

```
git checkout feature  
git push -u origin feature
```

WYSYŁAMY NOWĄ GAŁĄŻ



GIT PUSH



*unwiredlearning

PRACA Z GAŁĘZIAMI

```
git status  
ls  
git checkout master  
git status  
ls  
git diff feature
```

PRACA Z GAŁĘZIAMI

The screenshot shows the Visual Studio Code interface with the title bar "Git History - working_area - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Debug, Tasks, and Help. The left sidebar has icons for File, Search, Repository, and Settings. The main area is titled "Git History" and shows a list of commits for the file "test.txt". The commits are:

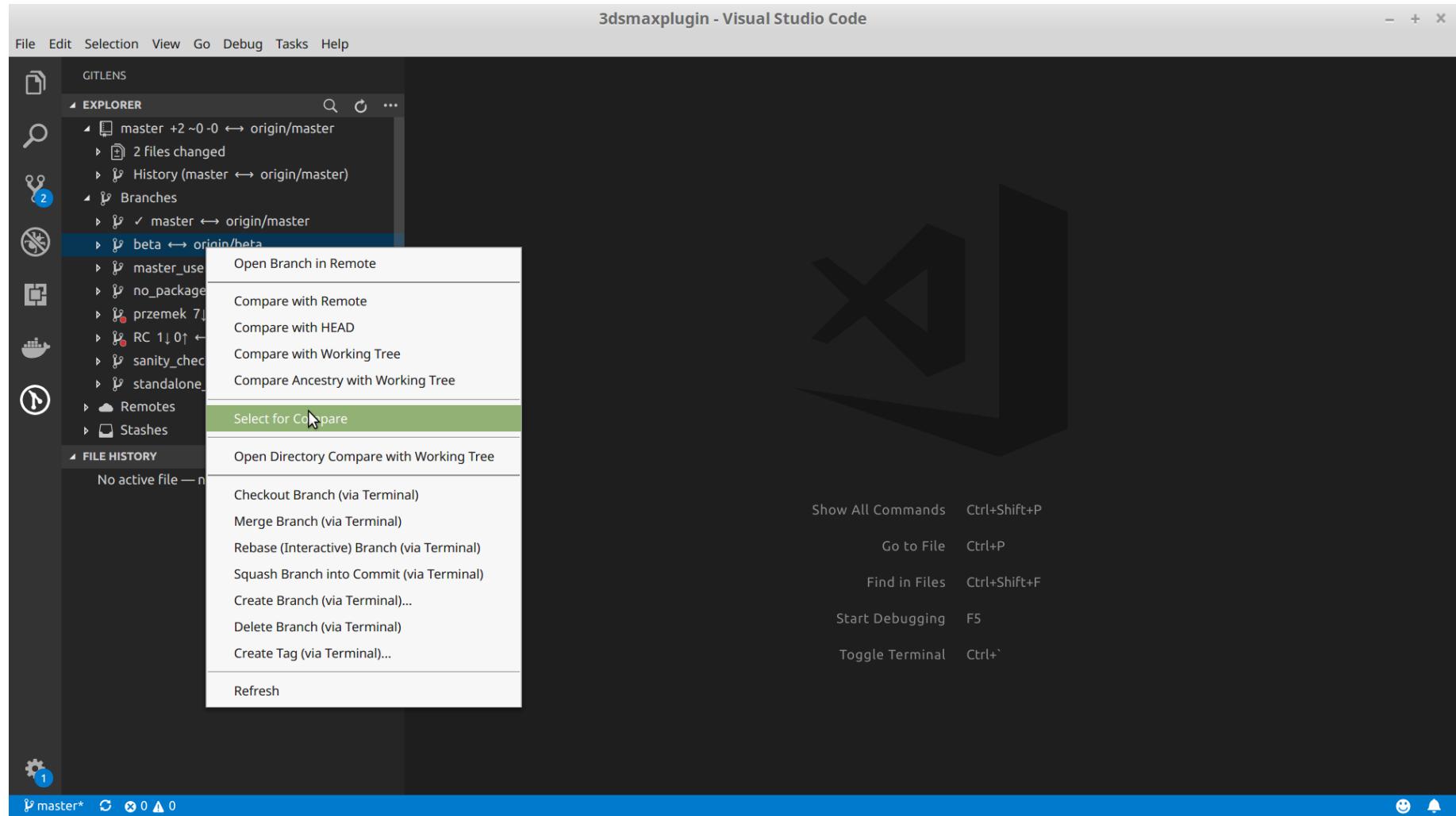
- New data (master, 3749c83, Konrad Klimaszewski, śr., 16 maj 2018, 00:33)
- Add data plotter. (feature, c926571, Konrad Klimaszewski, śr., 16 maj 2018, 00:02)
- Add input data (master, df0c960, Konrad Klimaszewski, śr., 16 maj 2018, 00:01)
- README adjustments (master, 43a6039, Konrad Klimaszewski, wt., 15 maj 2018, 23:47)
- Correct syntax error in main.py (master, 9ffe917, Konrad Klimaszewski, wt., 15 maj 2018, 22:02)
- README and data.txt (master, ae69110, Konrad Klimaszewski, wt., 15 maj 2018, 22:01)
- README (master, 1e51c59, Konrad Klimaszewski, wt., 15 maj 2018, 21:53)

Below the commit list, there is a terminal window showing the command used to generate the history:

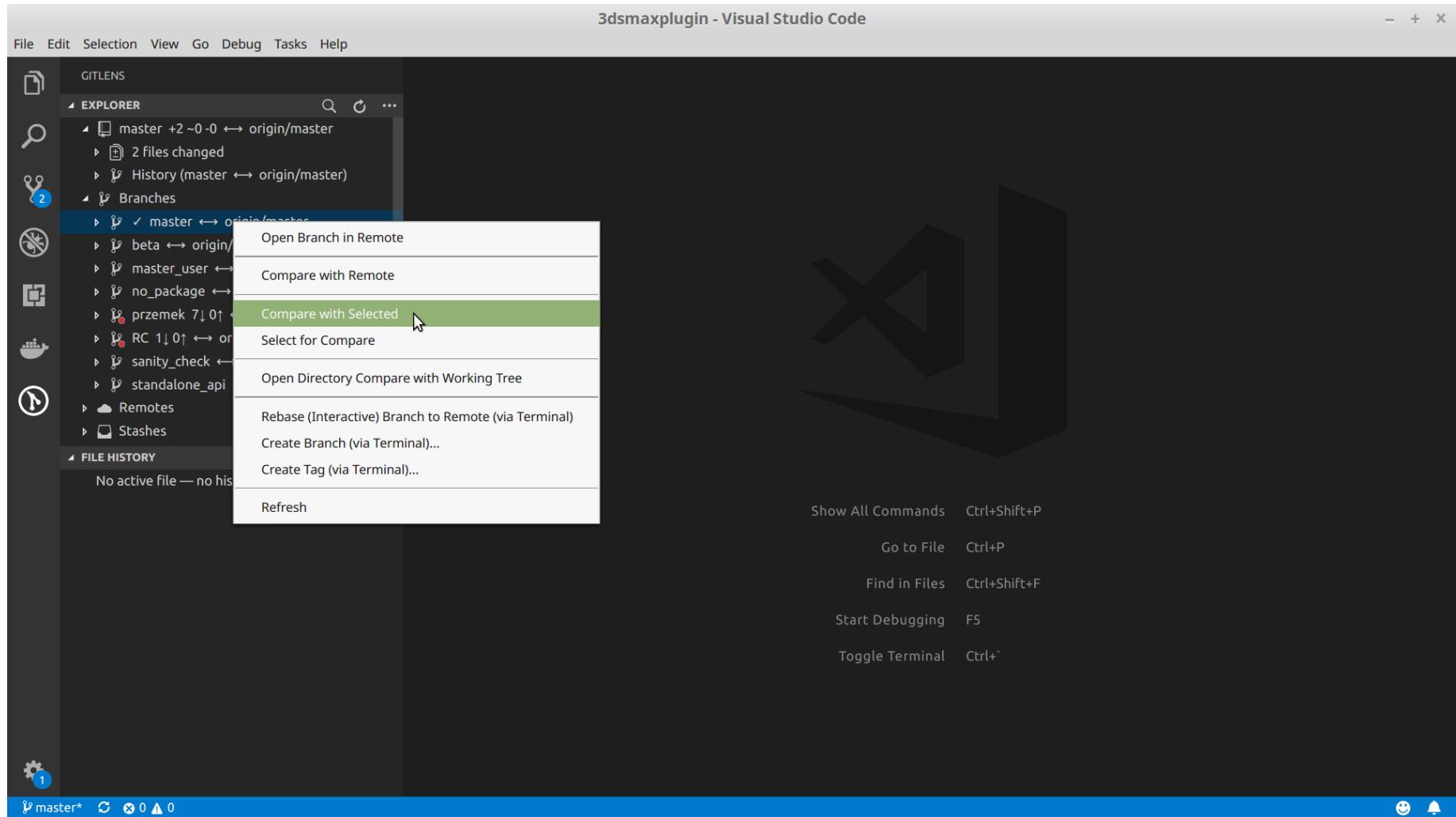
```
git log --full-history --pretty=oneline --date-order --decorate=full --skip=0 --max-count=100 --branches --tags --remotes (completed in 0.013s)
git show-ref (completed in 0.013s)
git branch --all --contains 3749c831bc1f9799fbb42955e7793070d83f49af (completed in 0.028s)
git branch --all --contains c9265713ada3b20ee06d5cd86976154521028cc2 (completed in 0.019s)
git remote get-url origin (completed in 0.019s)
fatal: No such remote 'origin'
```

The status bar at the bottom shows "master" and "Anaconda, Inc. Python 3.6.4 (jupyter) - INSERT -".

PRACA Z GAŁĘZIAМИ



PRACA Z GAŁĘZIAМИ



PRACA Z GAŁĘZIAМИ

The screenshot shows a Visual Studio Code window with the title "3dsmax.py (beta) ↔ 3dsmax.py (master) - 3dsmaxplugin - Visual Studio Code". The left sidebar displays the GitHub interface, showing a pull request from 'beta' to 'master'. The main editor area shows the code for "3dsmax.py" with changes highlighted in red and green. The terminal at the bottom shows the output of a pip upgrade command.

```
File Edit Selection View Go Debug Tasks Help  
GITLENS  
EXPLORER  
no_package ↔ origin/no_package  
przemek 7↓ 0↑ ↔ origin/przemek  
RC 1↓ 0↑ ↔ origin/RC  
sanity_check ↔ origin/sanity_check  
standalone_api ↔ origin/standalone_api  
FILE HISTORY  
RESULTS  
Add example 3ds Max scene. • You, 2 ye...  
Merge branch 'beta' into standalone_ap...  
Cookie getter • You, 2 years ago (4f1c...  
Benchamrk utils • You, 2 years ago (3d...  
Corrections to benchamrk code. • You, ...  
Documentation • You, 2 years ago (2a...  
Standalone class for renderownia.pl API...  
74 files changed  
api  
max_script  
scripts  
setup_data  
test  
tests  
transfer_utils  
ui  
widgets  
.gitignore  
3dsmax.py  
config.py  
dev_requirements.txt  
plugin.txt  
master* 0 0 0 Python 2.7.9 (virtualenv)  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
1: Python  
Found existing installation: six 1.10.0  
Uninstalling six-1.10.0:  
Successfully uninstalled six-1.10.0  
Running setup.py install for configparser ... done  
Running setup.py install for wrapt ... done  
Successfully installed astroid-1.6.5 backports.functools-lru-cache-1.5 configparser-3.5.0 enum34-1.1.6 futures-3.2.0 isort-4.3.4 lazy-object-proxy-1.3.1 mccabe-0.6.1 pylint-1.9.3 singledispatch-3.4.0.3 six-1.11.0 wrapt-1.10.11  
You are using pip version 9.0.1, however version 18.0 is available.  
You should consider upgrading via the 'pip install --upgrade pip' command.  
(python)konrad@nb53cis ~/Soft/Renderownia/3dsmaxplugin $  
You, 10 months ago Ln 20, Col 1 Spaces: 4 UTF-8 LF Python ☺ 🔔
```

ĆWICZENIA (MASTER)

W gałęzi **master**:

- Upiększamy wypisywanie danych: zamieniamy funkcję `print` na `pprint`

```
from pprint import pprint  
pprint(data)
```

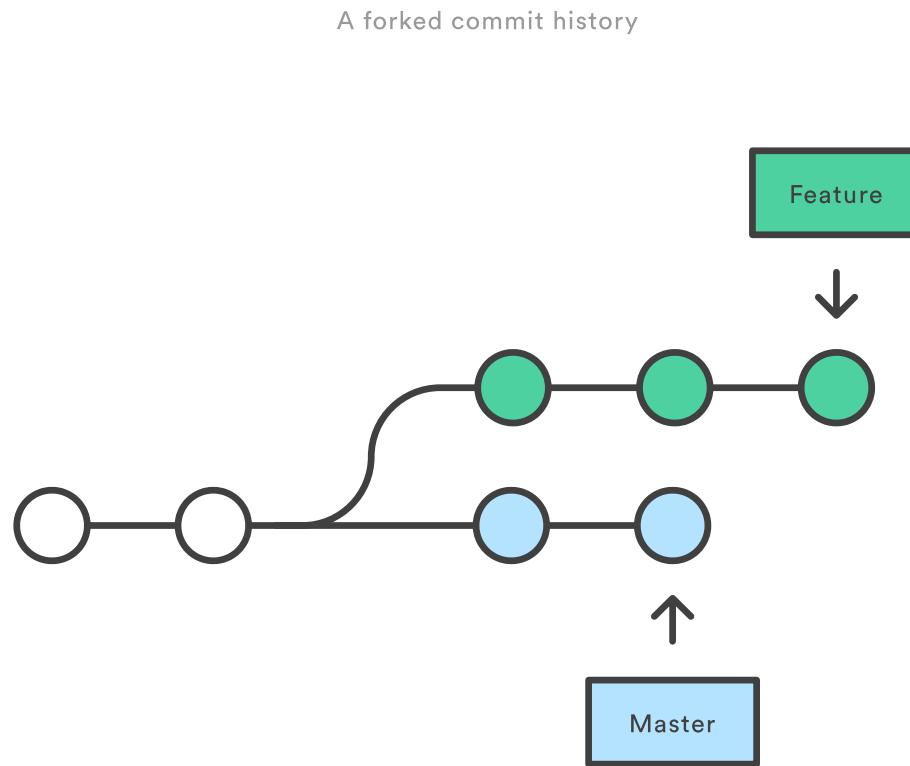
- Zapisujemy jako commit

Jako komentarza używamy zwrotu **Fixes #2**

- Modyfikujemy funkcję `display`
 - przyjmuje dodatkowy parametr `select` z domyślną wartością `None`
 - jeśli parametr został zdefiniowany wyświetla tylko wybrane dane (np. na podstawie wieku)

GIT - ŁĄCZENIE GAŁĘZI

Załóżmy że gałęzie Feature oraz Master rozjechały się

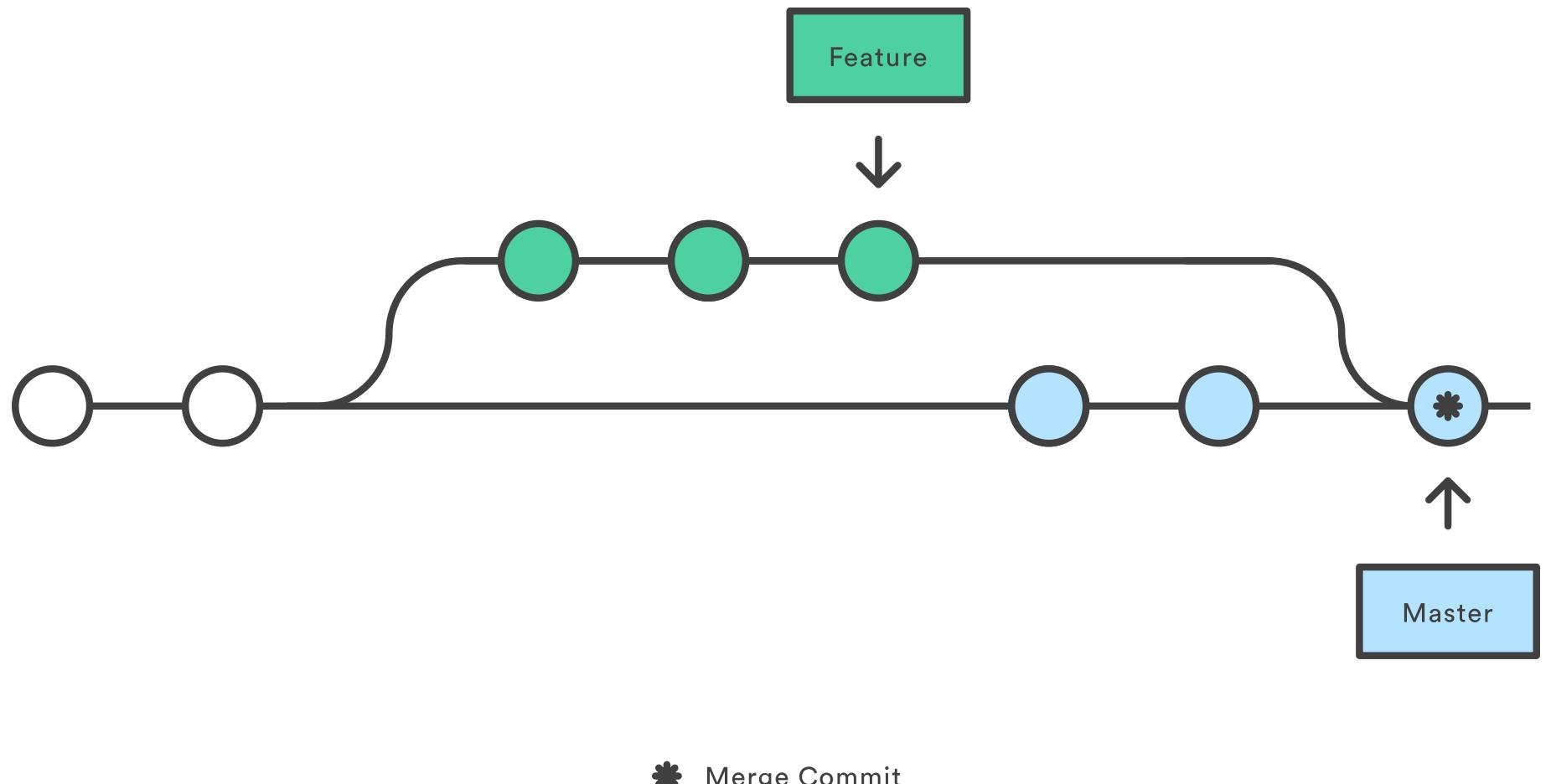


MERGE

- Pozostawia commity bez zmian
- Rozwiązywanie konfliktów wykonujemy dla pełnego merge
- Historia repozytorium staje się nieliniowa
- Jeśli repozytoria nie rozjechały się możliwy jest "fast forward"
 - Nowe commity dopisywane są na końcu gałęzi bez dodatkowego commitu "merge"

MERGE

Merge without rebasing



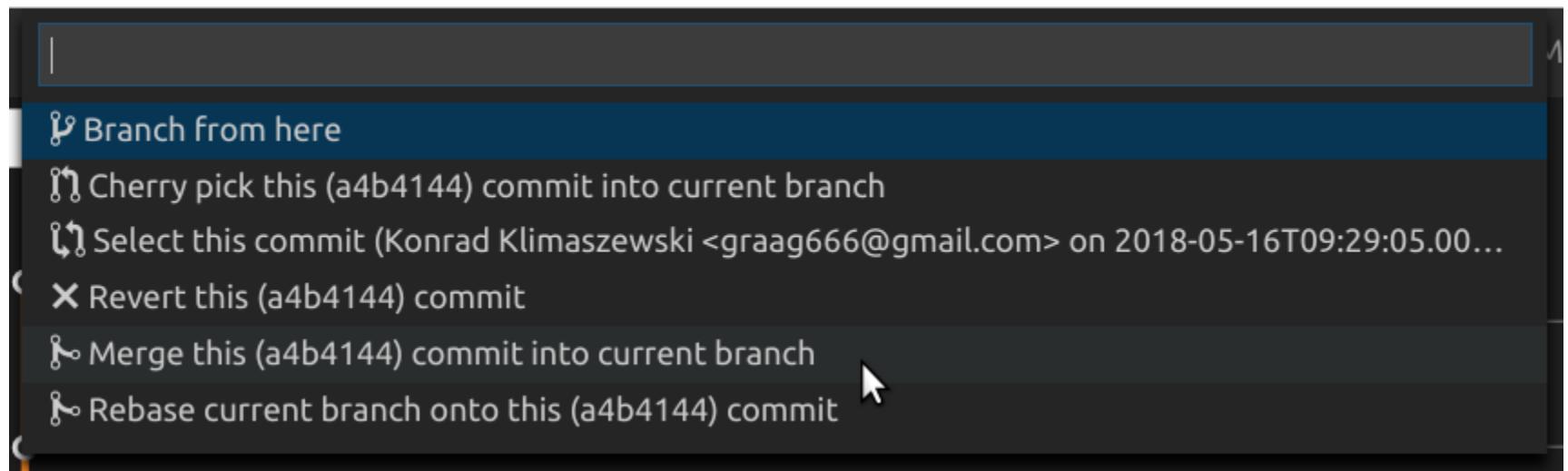
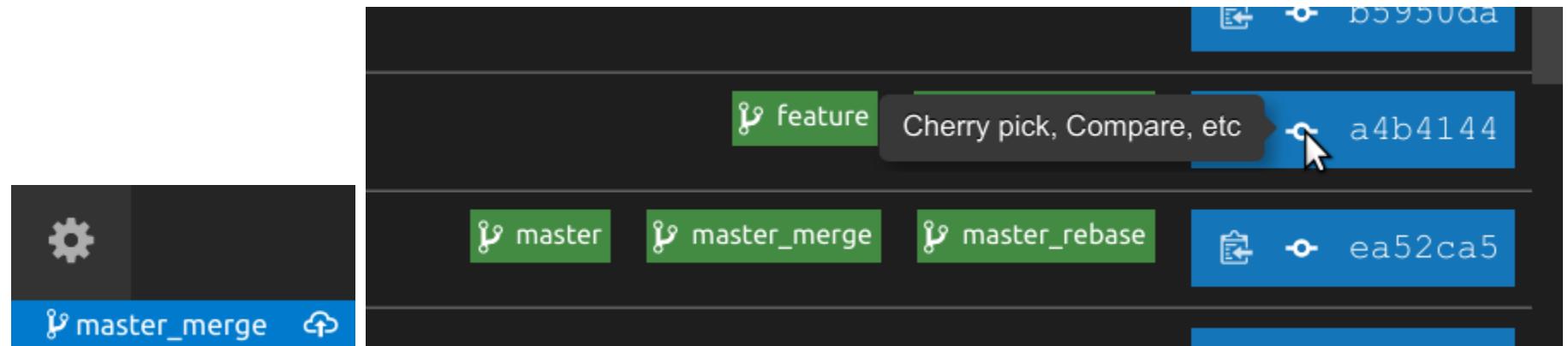
* Merge Commit

MERGE

Będąc w gałęzi **master** wykonujemy *merge* zmian z gałęzi **feature**

```
git checkout master  
git merge feature  
gitk --all
```

MERGE



ROZWIĄZYWANIE KONFLIKTÓW

main.py - git_hands_on - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

SOURCE CONTROL: GIT

Message (press Ctrl+Enter to commit)

MERGE CHANGES

main.py 1, C

CHANGES 0

You, a few seconds ago | 1 author (You) | Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<< HEAD You, a few seconds ago * Uncommitted changes
1 p1, p2, p3 = 1, -2, -7
2 p4, p5, p6 = 1, 5, 7
3 =====
4 >>>> feature (Incoming Change)
5 def solve_poly(a, b, c):
6     delta = b*b - 4*a*c
7     x_1 = (-b - delta**0.5)/2/a
8     x_2 = (-b + delta**0.5)/2/a
9
10    return x_1, x_2
11
12
13
14 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
15 <<<< HEAD (Current Change)
16 print(solve_poly(p1, p2, p3))
17 print(solve_poly(p4, p5, p6))
18 =====
19 def read_file(file_name):
20     with open(file_name, 'r') as f:
21         plik = f.readlines()
22     return plik
23
24 parametry = read_file('parametry.txt')
25
26 for line in parametry:
27     p1, p2, p3 = [float(x) for x in line.split(',')]
28     x1, x2 = solve_poly(p1, p2, p3)
29     print(p1, p2, p3, '\t', x1, x2)
30
31 >>>> feature (Incoming Change)
32
33
```

You, a few seconds ago Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 😊 🔔

FAST FORWARD

Fast Forward występuje wtedy gdy zmiany z gałęzi możemy czysto nałożyć na aktualną.

- Zakładamy gałąź *clean*
- Dodajemy do repozytorium nowy plik *clean.txt*
- Wykonujemy **git commit**
- Przechodzimy do gałęzi *master*
- Wykonujemy **git merge clean**

GIT - PRZEPISYWANIE HISTORII

- `git commit --amend` - umożliwia poprawienie ostatniego wysłanego commitu
- `git rebase | git pull --rebase`
- `git cherry-pick`
- `git filter-branch` - pozwala przepisać historię WSZYSTKICH commitów

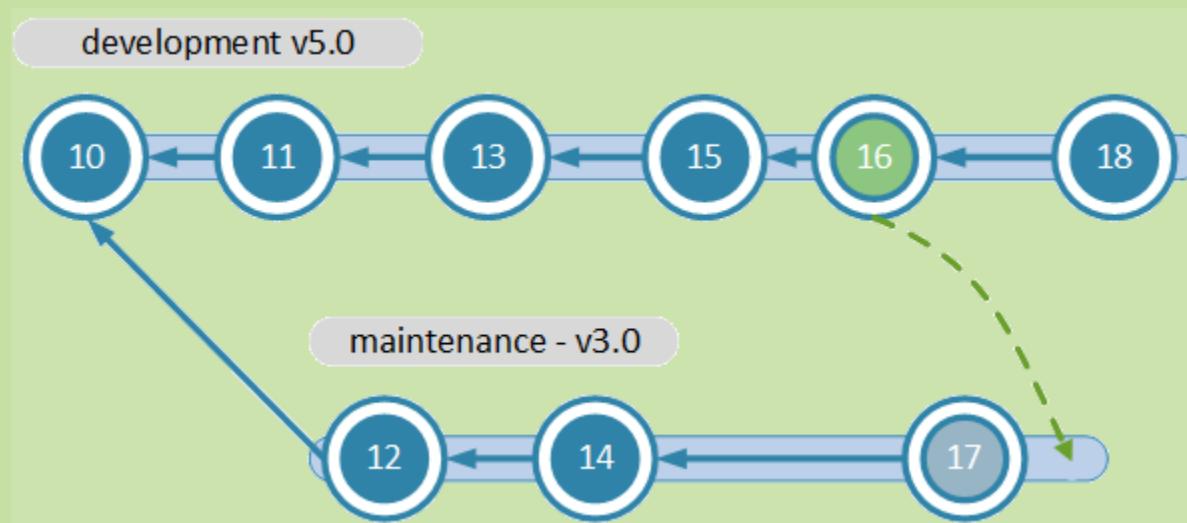
stosowane bardzo rzadko i tylko w gardłowych sytuacjach (np. commity zawierały poufne informacje)

UWAGA

Dobrą zasadą jest nie podmienianie commitów wysłanych do publicznego repozytorium !!

WYBIERANIE WISIENEK

git cherry-pick - pozwala dodać pojedynczy commit z innej gałęzi

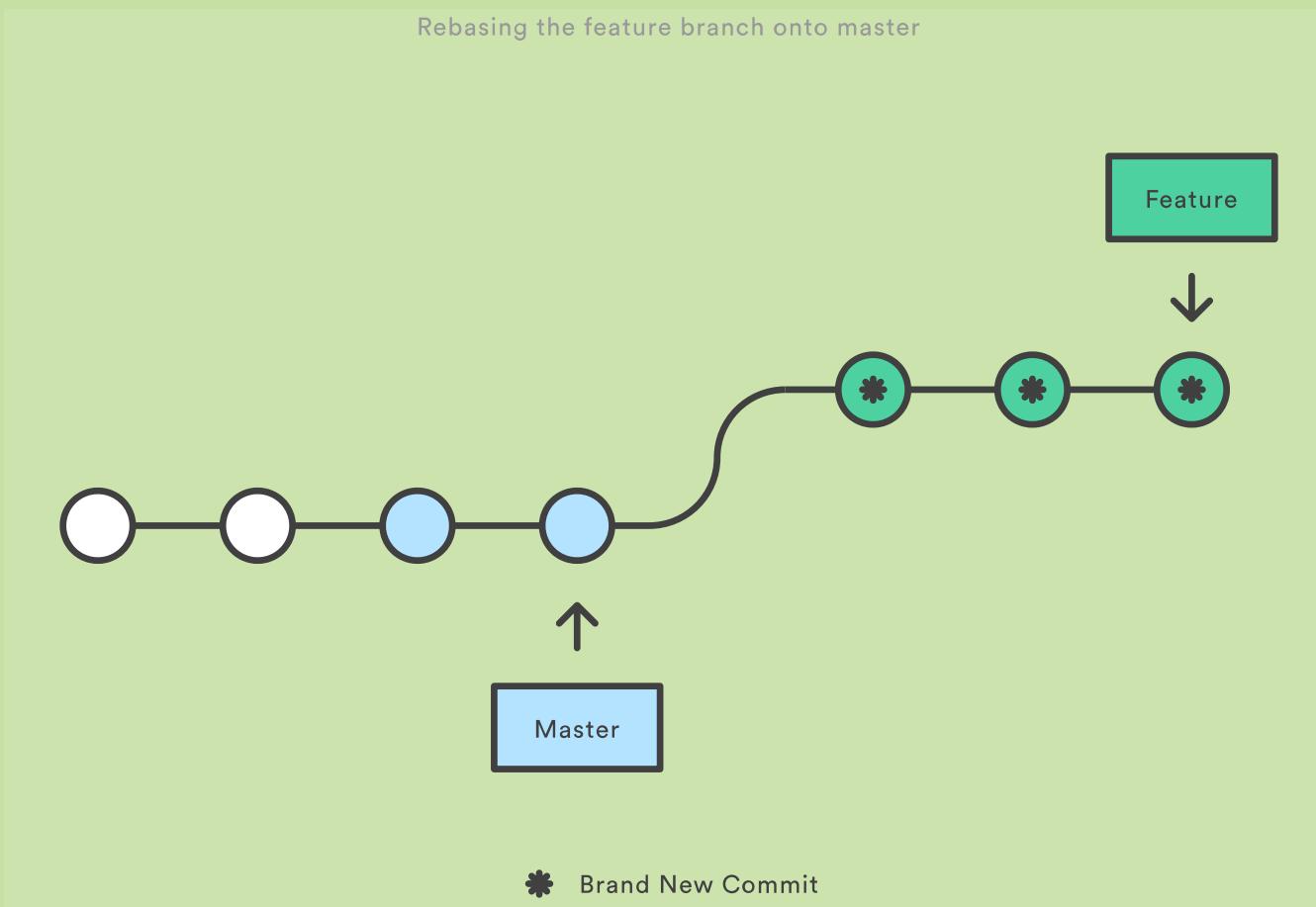


*plasticscm.com

REBASE

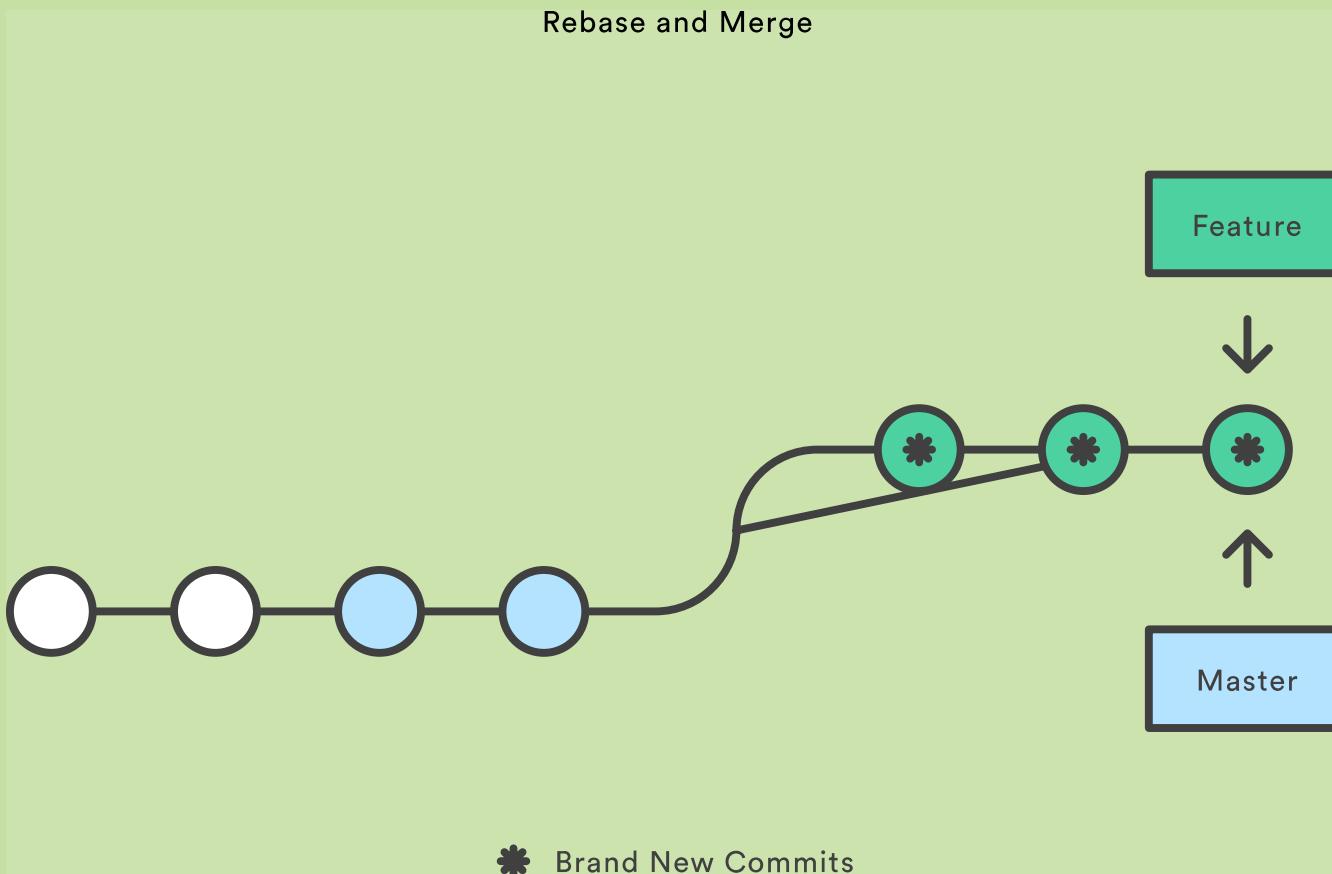
- Przepisuje commity w gałęzi od nowa
- Komity w gałęzi stają się dziećmi **HEAD** gałęzi master
- Rozwiązywanie konfliktów wykonujemy dla każdego z commitów
- Historia repozytorium pozostaje liniowa

REBASE



*atlassian.com

REBASE



*atlassian.com

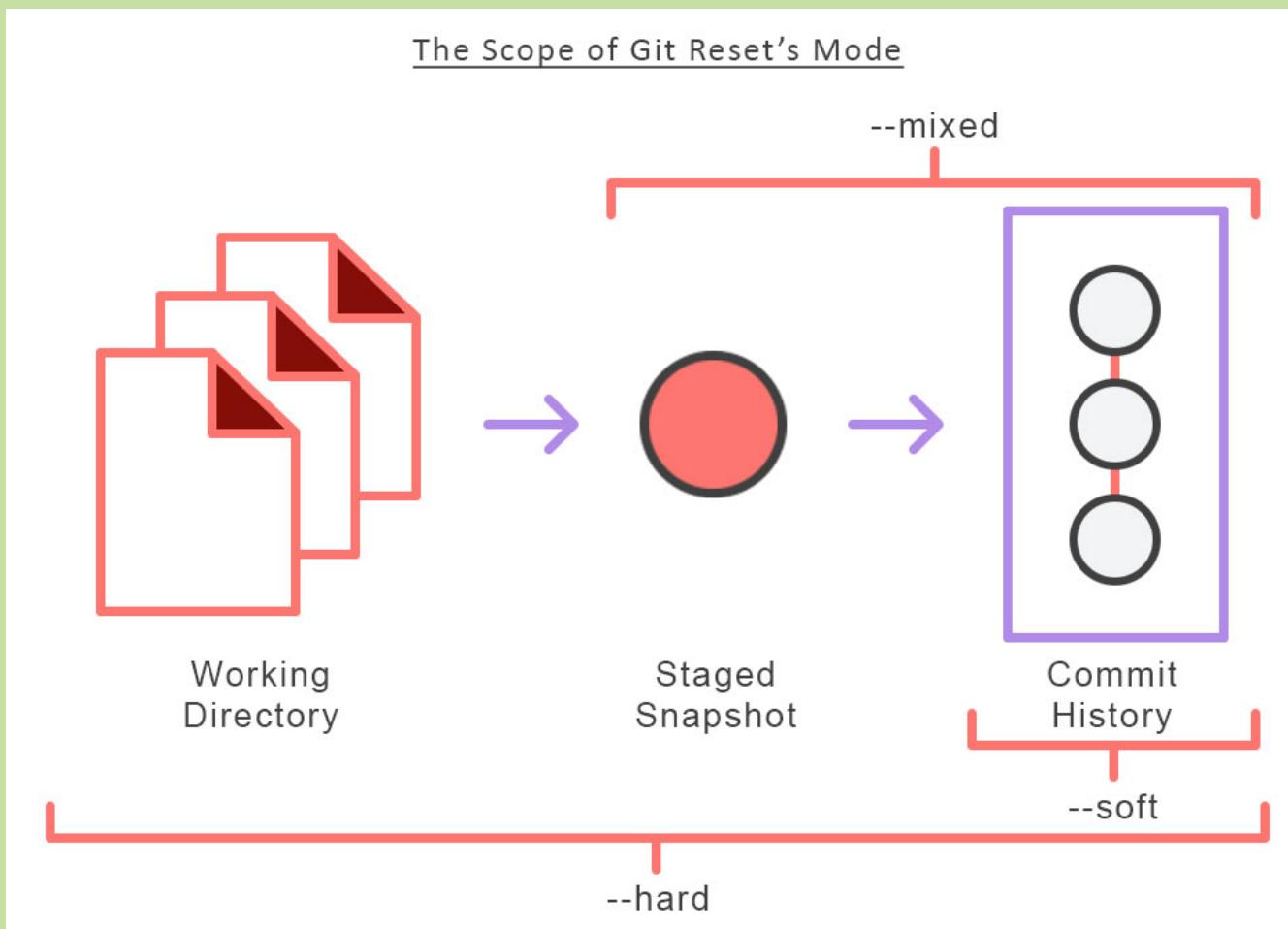
ĆWICZENIE REBASE

Cofamy merge w gałęzi master

```
git reset --hard HEAD^
```

Uwaga!!! git reset --hard usuwa pliki

GIT RESET



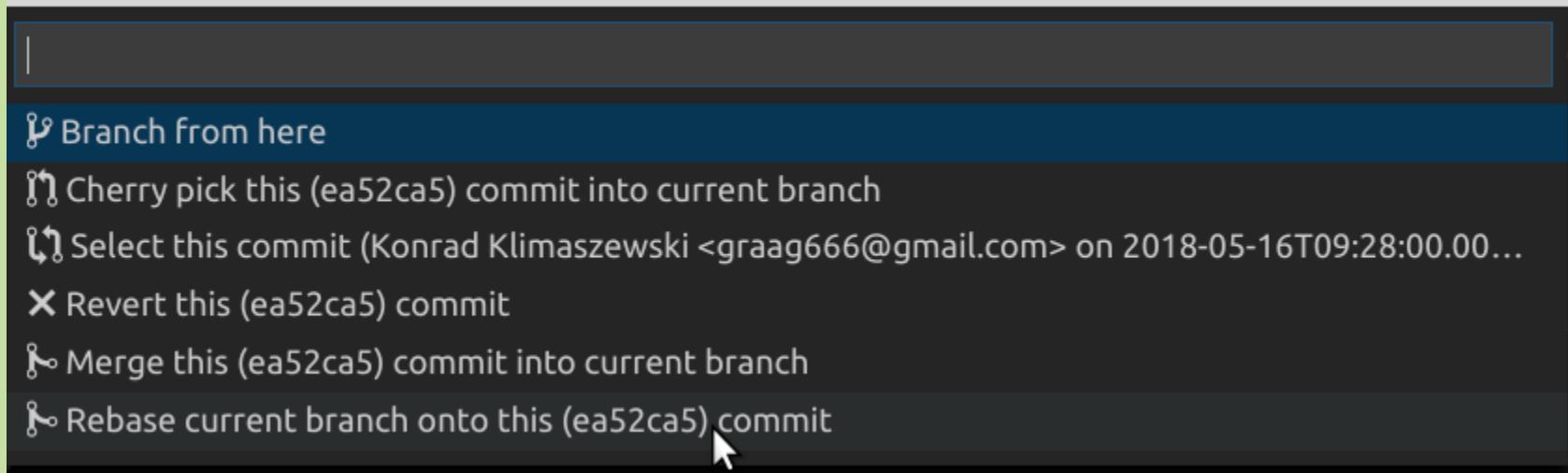
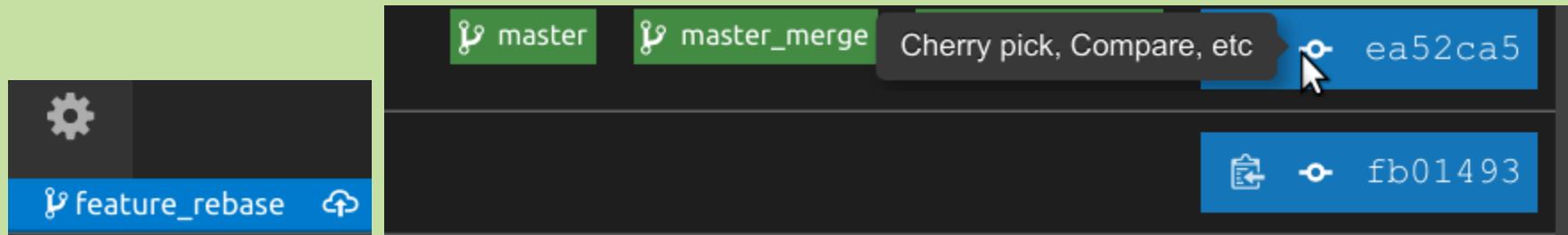
ĆWICZENIE REBASE

Będąc w gałęzi **feature** wykonujemy *rebase* zmian na **HEAD** gałęzi **master**

```
git checkout feature
git rebase master
gitk --all
```

Dla każdego commitu z gałęzi **feature** rozwiązujemy konflikty jeśli istnieją

ĆWICZENIE REBASE



Git History vекты_из_файла.py Git History main.py README.md

Search All branches All Authors Clear Refresh

Some logs
Konrad Klimaszewski on śr., 16 maj 2018, 11:29
feature_rebase 86a0de7

Add data plotter.
Konrad Klimaszewski on śr., 16 maj 2018, 00:02
4f1f872

Add input data
Konrad Klimaszewski on śr., 16 maj 2018, 00:01
b5950da

Some logs
Konrad Klimaszewski on śr., 16 maj 2018, 11:29
feature feature_merge a4b4144

More prints
Konrad Klimaszewski on śr., 16 maj 2018, 11:28
master master_merge master_rebase ea52ca5

Dodatkowe informacje w README
Konrad Klimaszewski on śr., 16 maj 2018, 11:25
fb01493

New data
origin/master 3749c83

GIT WORKFLOWS



*pinterest.com

PRACA ZESPOŁOWA

ZESPÓŁ JEDNOOSOBOWY ;-)

Pracuje samodzielnie nad projektem.

Po co mi Git?

- Backup
- Przenośność między maszynami
- Prostota udostępnienia innym poprzez GitHub, GitLab etc.



*buddy.works

PRACA W ZESPOLE

- Istnieje wiele modeli pracy zespołowej w oparciu o git:
 - Git Flow
 - Feature Branch
 - GitHub Flow
 - Forking Flow

PRACA W ZESPOLE

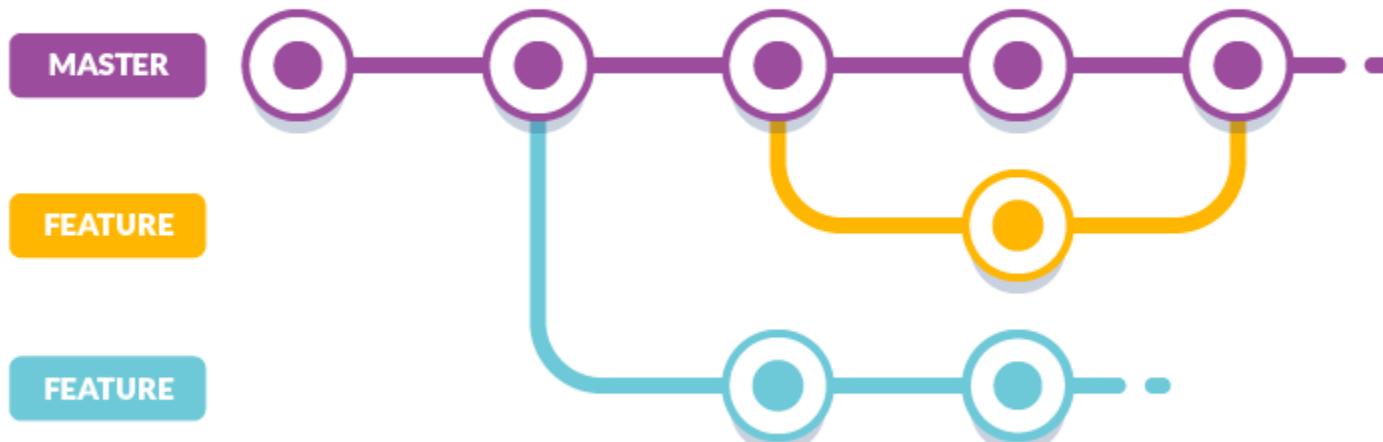
Wspólne cechy wszystkich *Flow:

- Jedna gałąź jest uznawana za **stabilną**
 - Kod który tu ląduje **musi** być działający i gotowy do wykorzystania
 - Często będzie automatycznie wysyłany do środowiska produkcyjnego
- Prace rozwojowe wykonujemy w **gałęziach**
- Operacje merge poprzedzone są **recenzją kodu**

FEATURE BRANCH / GITHUB WORKFLOW

HEAD gałęzi *master* zawsze jest działającą aplikacją

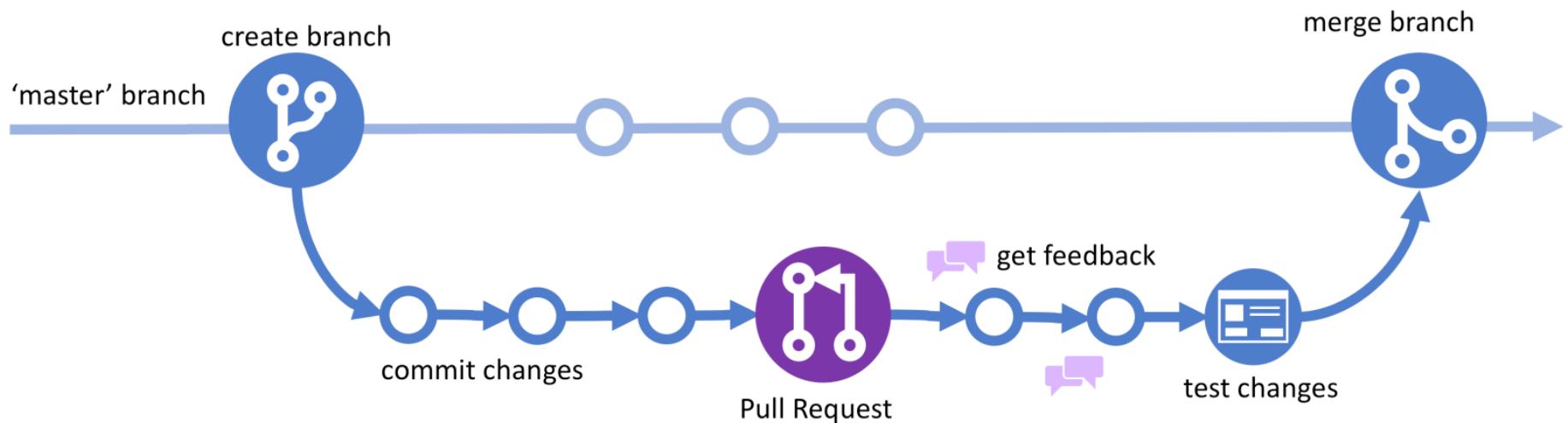
Każda zmiana zaczyna się od utworzenia nowej gałęzi



FEATURE BRANCH / GITHUB WORKFLOW

Każdy merge poprzedzony jest merge/pull request-em i recenzją kodu

GitHub Flow



Copyright © 2018 Build Azure LLC

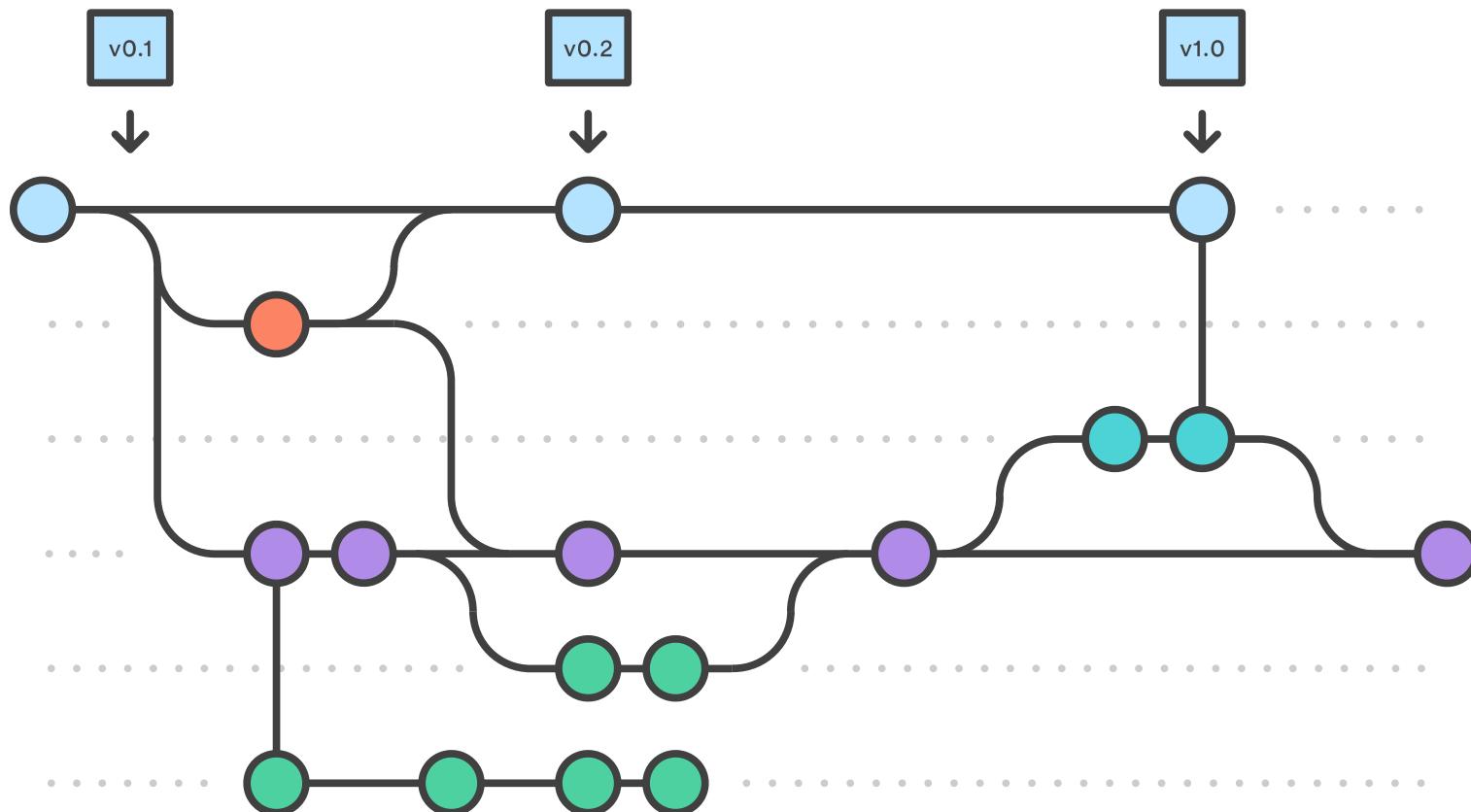
<http://buildazure.com>

*buildazure.com

GIT FLOW WORKFLOW

- Preferowany przez większe projekty
- Kilka zdefiniowanych gałęzi odpowiada różnym stadiom rozwoju projektu
- Bardzo często łączony z Forking Workflow

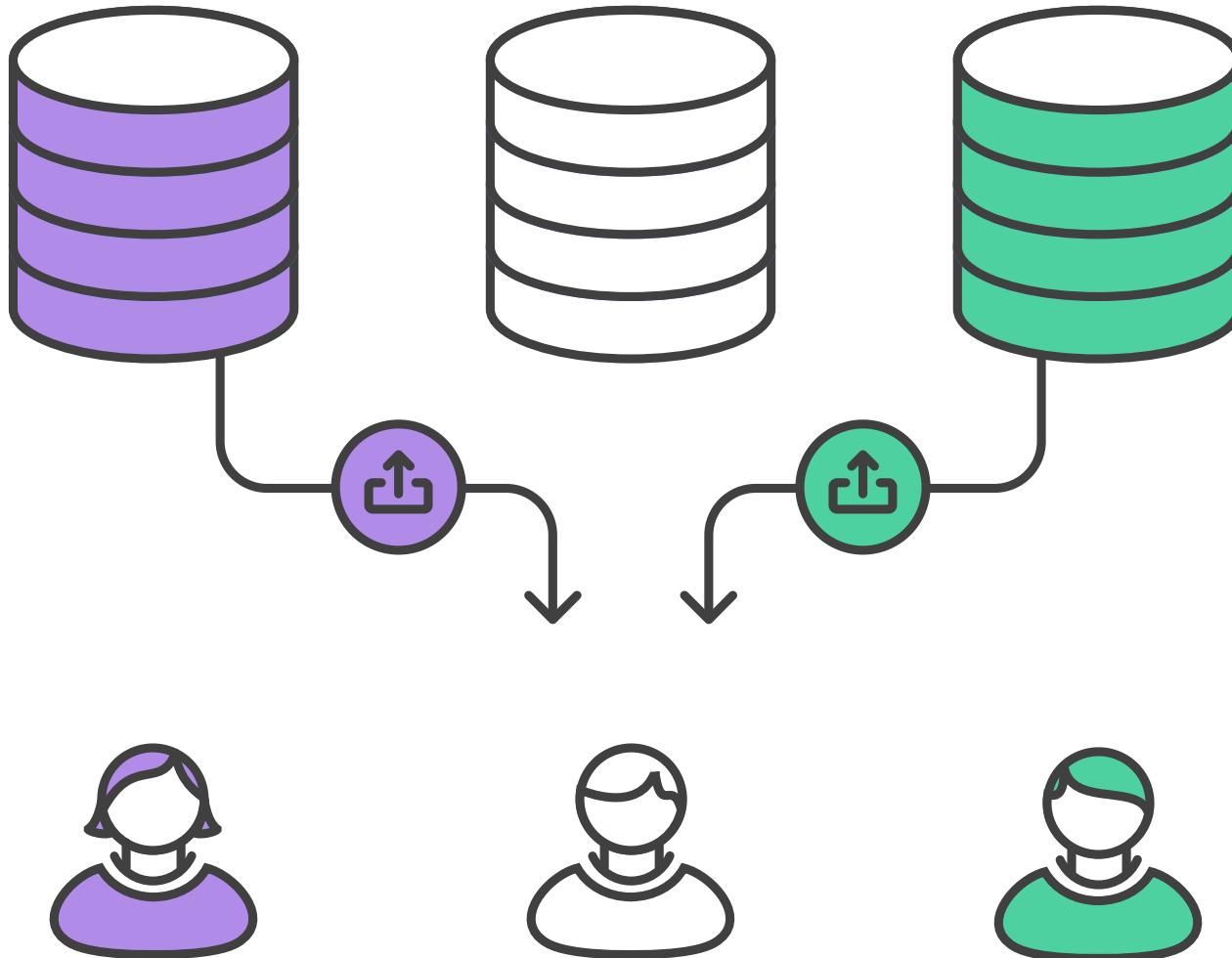
GIT FLOW WORKFLOW



FORKING WORKFLOW

- Bardzo podobny do GitHub workflow
- Zamiast gałęzi w głównym repozytorium wykorzystujemy fork-i
 - Każdy pracuje z własnym forkiem przesyłając pull requesty do maintainera
- Popularny wśród małych projektów na GitHub / BitBucket etc
- Jeden maintainer głównego repozytorium

FORKING WORKFLOW



*atlassian.com

HANDS ON LAB 4



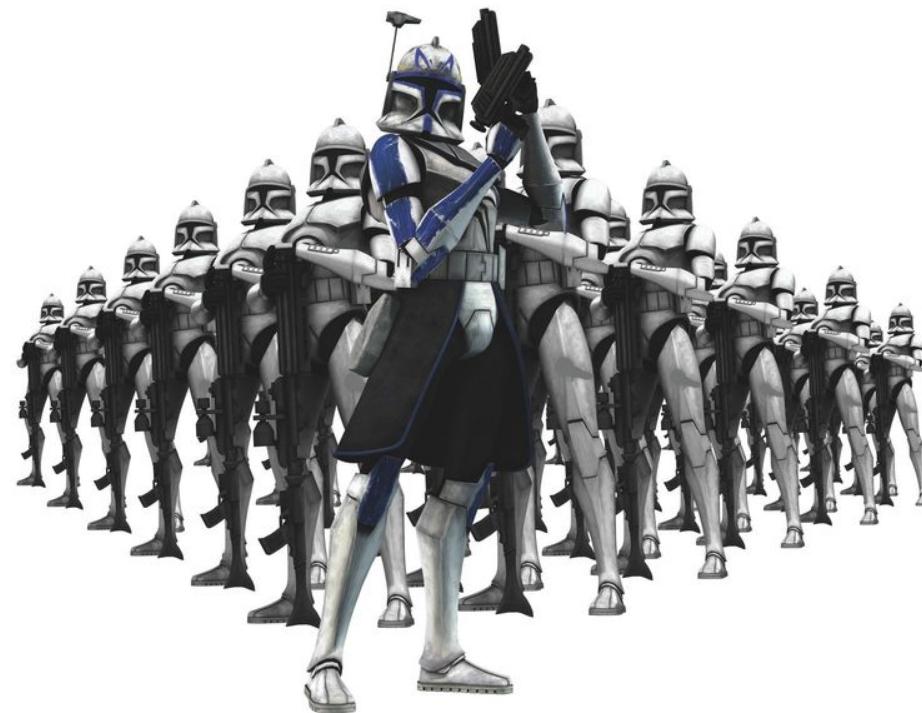
"Any problem working remotely?"

*mjms.net

PRACA ZESPOŁOWA

KLONY

git-for-scientists



*Lucasfilm Ltd.

KLONOWANIE

git clone - Klonuje / tworzy kopię zdalnego repozytorium lokalnie

Klonowanie istniejącego repozytorium (SSH):

```
git clone git@code.cis.gov.pl:developerscis/git-for-scientists.git
```

Klonowanie istniejącego repozytorium (HTTPS):

```
git clone https://code.cis.gov.pl/developerscis/git-for-scientists.git
```

Klonowanie istniejącego lokalnego repozytorium:

```
git clone file:///home/mkarpiarz/git/gitdemo.git
```

ŚLEDZENIE

Git pozwala na śledzenie zdalnych repozytoriów:

```
git remote --help
```

Podczas operacji `clone` domyślnie dodawane jest zdalne repozytorium
`origin`

```
cd git-for-scientists
git remote -v
```

POBIERANIE ZMIAN

git pull - Pobiera zdalne zmiany i integruje je z lokalnym repozytorium

```
git pull      # git fetch + git merge  
git pull --rebase # git fetch + git rebase
```

- W rzeczywistości synchronizacja z zdalnym repozytorium przebiega dwu etapowo:
 - Pobieramy aktualny stan zdalnego repozytorium: *git fetch*
 - Dołączamy zmiany do lokalnej gałęzi: *git merge* lub *git rebase*



FEATURE BRANCH

Tworzymy gałąź na poprawki które przygotowujemy do włączenia do
upstream

```
git checkout -b <username>
```

- Zapoznajmey się z `Main.md`
- Wybieramy jeden z "rozdziałów"
- Edytujemy plik rozdziału zgodnie z zawartym `TODO`

```
git add <rozdział>.md
git commit
```

FEATURE BRANCH

Wysyłamy nową gałąź do repozytorium zdalnego:

```
git push -u origin <username>
```

MERGE REQUESTS

The screenshot shows a GitLab interface for a repository. At the top, there's a header bar with a user icon (a cat), the repository name 'Konrad Klimaszewski / GitAndGitlabPresentation', and various navigation icons like search, help, and settings. Below the header is a navigation menu with links for Home, Files, Commits, Network, Graphs, Issues, Merge (which is highlighted in blue), Wiki, and Settings. The main content area is titled 'Merge Requests (0)' and features a green button labeled '+ New Merge Request'. At the bottom, there are three filter dropdowns: 'assignee: Any', 'milestone: Any', and 'sort: Newest'.

Merge Requests (0)

+ New Merge Request

assignee: Any

milestone: Any

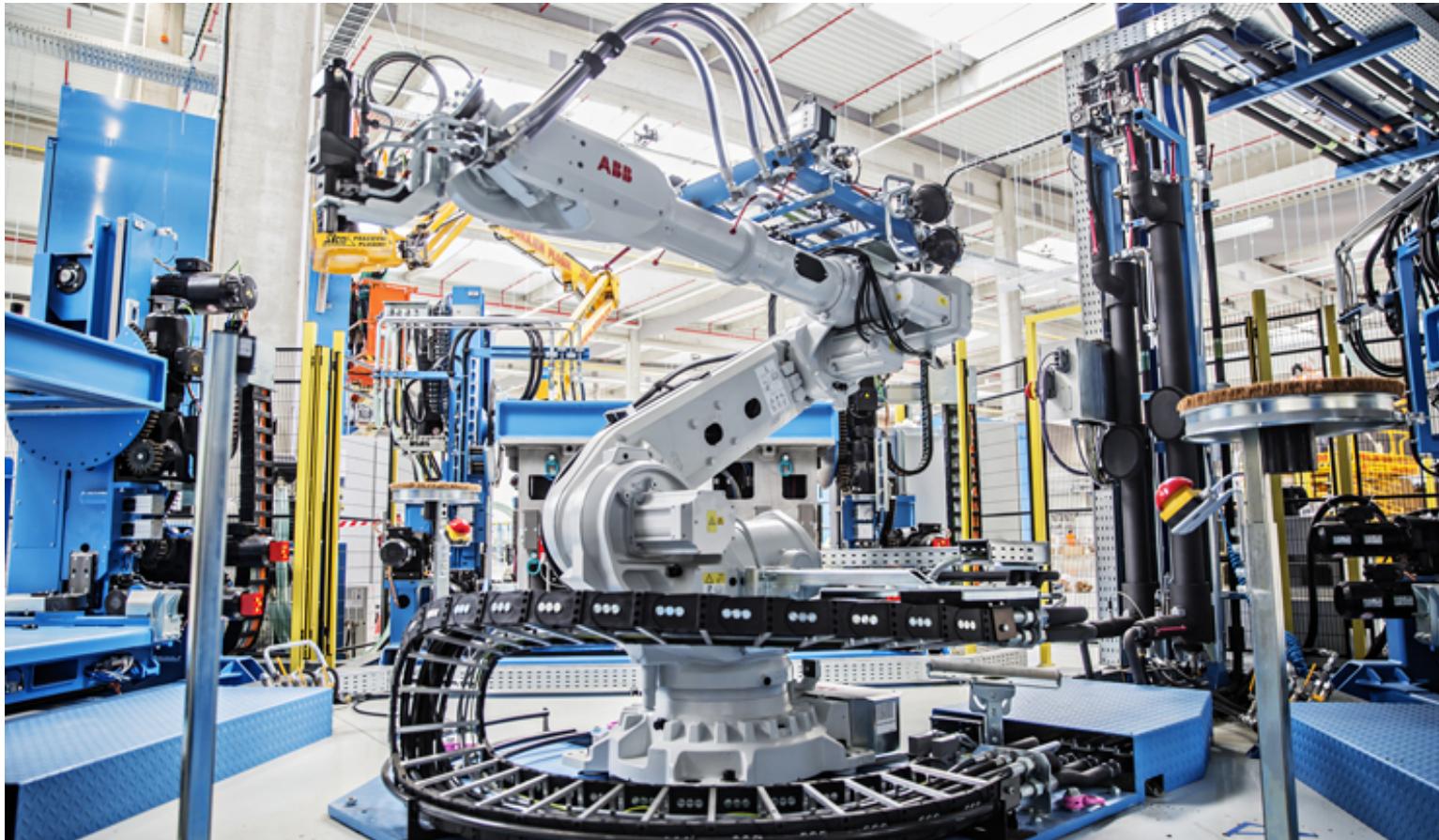
sort: Newest

MERGE REQUESTS

- Każdy zostanie przypisany do jednego MR
- Każdy dodaje min jeden komentarz dotyczący konkretnej linii kodu widocznej w "changes"

The screenshot shows the GitLab interface for a merge request. The top navigation bar includes links for GitLab, Projects, Groups, Activity, Milestones, Snippets, and a search bar. The main header shows the current tab is 'Changes' with 116 items. A summary indicates 16/26 discussions resolved. The left sidebar lists project structure under 'Models' and 'Services/Test'. The central area displays a diff view for the file '.gitignore'. A specific line of code is highlighted in purple, and a tooltip suggests adding a comment to that line. A modal window is open, prompting the user to 'Add a comment to this line' with fields for 'Write' and 'Preview', and a note that Markdown is supported. The right sidebar shows user activity metrics: 56 new commits, 4 merged pull requests, 4 issues, and 4 merge requests.

AUTOMATYZACJA



SZANUJ SWÓJ CZAS ;)

CONTINOUS INTEGRATION

BEST PRACTICES

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

CI



<https://jenkins.io>



<https://travis-ci.org>

CI



<https://circleci.com>



<https://gitlab.com>

TESTERY KODU



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



<https://coveralls.io/>



<https://snyk.io/>

PRZYKŁADY

- <https://github.com/tensorflow/tensorflow>
- <https://github.com/flutter/flutter>
- <https://github.com/facebook/react>

CODE.CIS.GOV.PL

<https://docs.gitlab.com/ce/ci/>

<https://docs.gitlab.com/ce/ci/yaml/README.html>

ĆWICZENIA

Aktywujemy CI dla projektu **git_handson**

- Settings -> General Settings -> Visibility ... -> Pipelines -> [On]
- Settings -> CI/CD -> Runners -> Shared Runners -> [Enable Shared Runners]

ĆWICZENIA

- Otwieramy projekt `git_handson`
- Dodajemy plik `.gitlab-ci.yml`

```
pylint: # Nazwa zadania
  tags: # Tagi określają predefiniowane środowiska
    - python3 # W CIŚ mamy python3 i docker
  stage: test # Domyślnie wyróżniamy trzy etapy: build -> test -> deploy
  script: # Skrypt który będzie wykonany w zadaniu
    - pip install pylint --quiet
    - pylint *.py
```

HANDS ON LAB 5



yandex.ru

SZTUCZKI GIT

GUIDELINES

- Zapisuj często swoją pracę: **commit**
- Dbaj by komentarze commit-ów opisywały zawarte zmiany
- Korzystaj z zdalnego repozytorium: **push**
- Nie nadpisuj opublikowanej historii: **git push --force**
- Korzystaj z automatyzacji (automatyczne testy, komplikacja, etc)
- W razie problemów nie panikuj
 - Zrób backup
 - Spokojnie poszukaj rozwiązań - w większości przypadków udaje się *naprostować* repozytorium git

<https://sethrobertson.github.io/GitBestPractices/>

GIT I PLIKI BINARNE

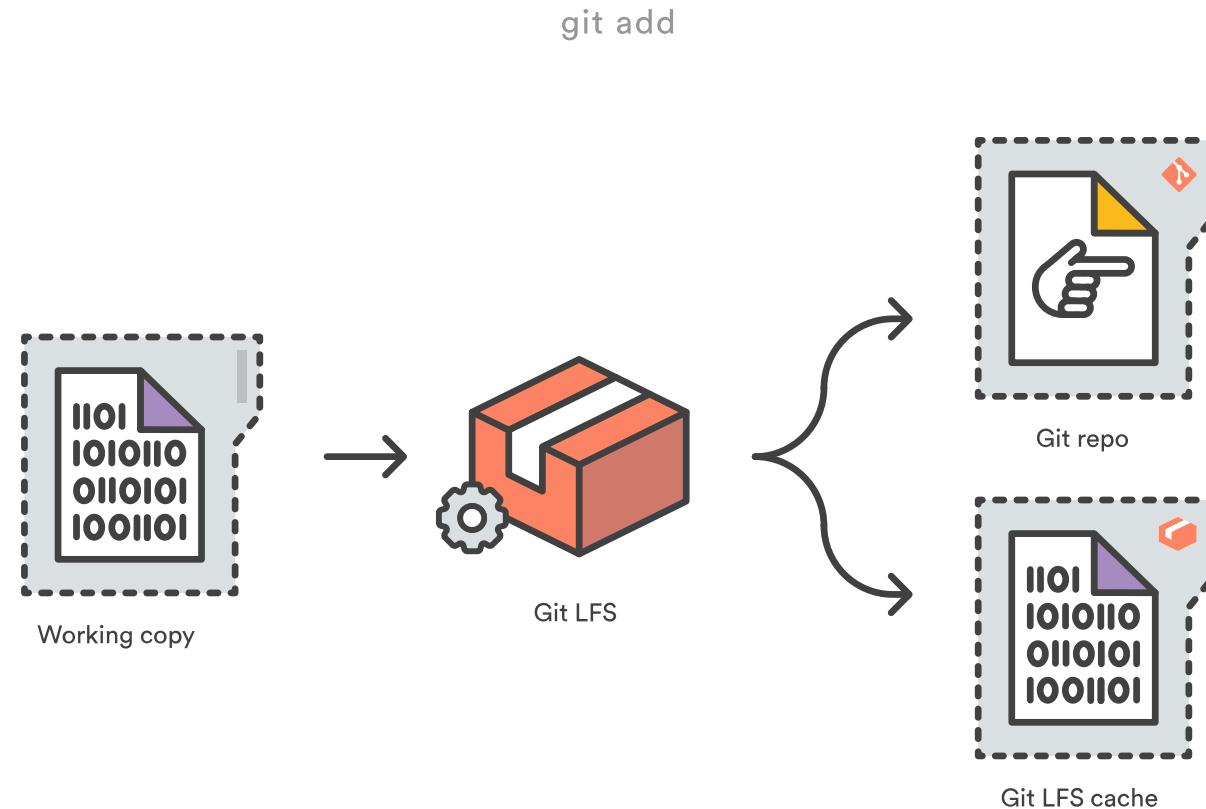
- Git słabo radzi sobie z dużymi plikami binarnymi
- Nawet małe zmiany zazwyczaj prowadzą do zapisania pełnej kopii pliku binarnego

ROZWIĄZANIE

GIT LARGE FILE STORAGE (LFS)

- <https://git-lfs.github.com/>
- https://docs.gitlab.com/ee/workflow/lfs/manage_large_binaries_with_git_

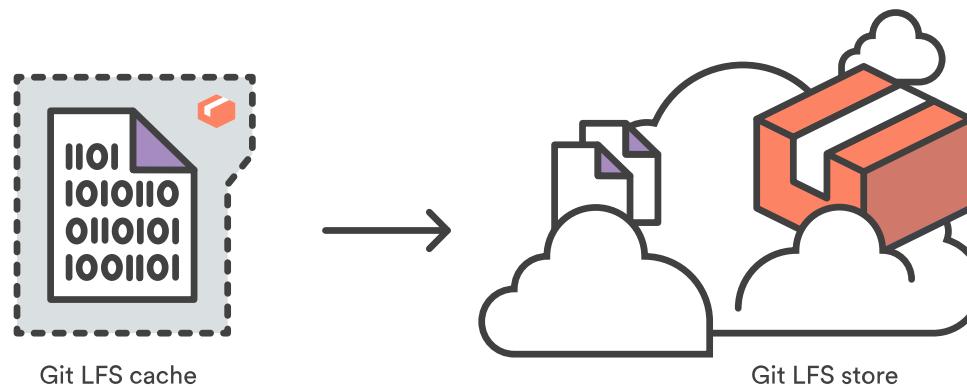
GIT LFS



*atlassian.com

GIT LFS

git push

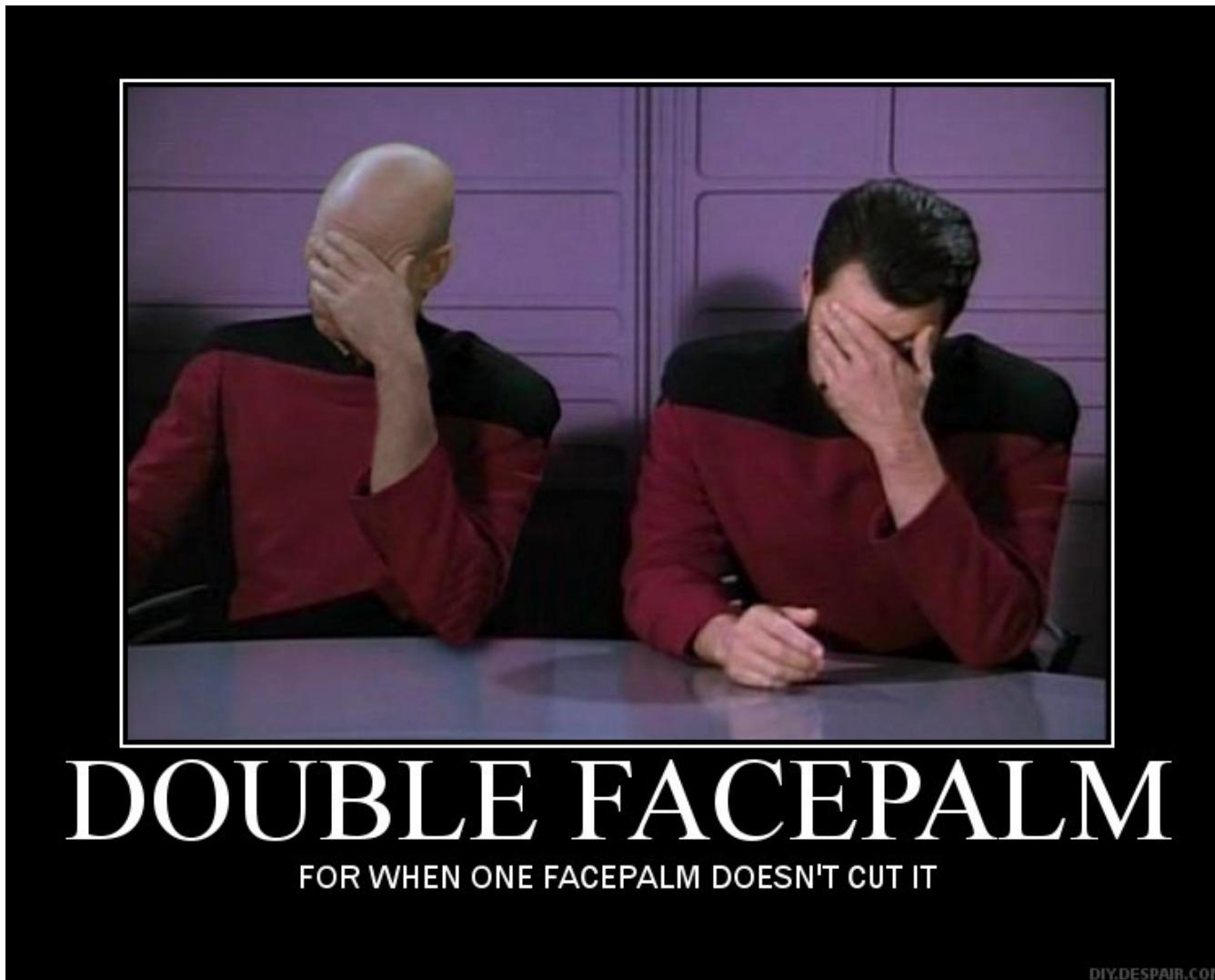


*atlassian.com

GIT LFS W PIŁCE

```
git lfs install      # initialize the Git LFS project
git lfs track "*.iso" # select the file extensions that you want to treat as large
git add .gitattributes
```

GIT - ZWALIŁEM OSTATNI COMMIT ;-(



DOUBLE FACEPALM

FOR WHEN ONE FACEPALM DOESN'T CUT IT

DIY.DESPAIR.COM

*diy.despair.com

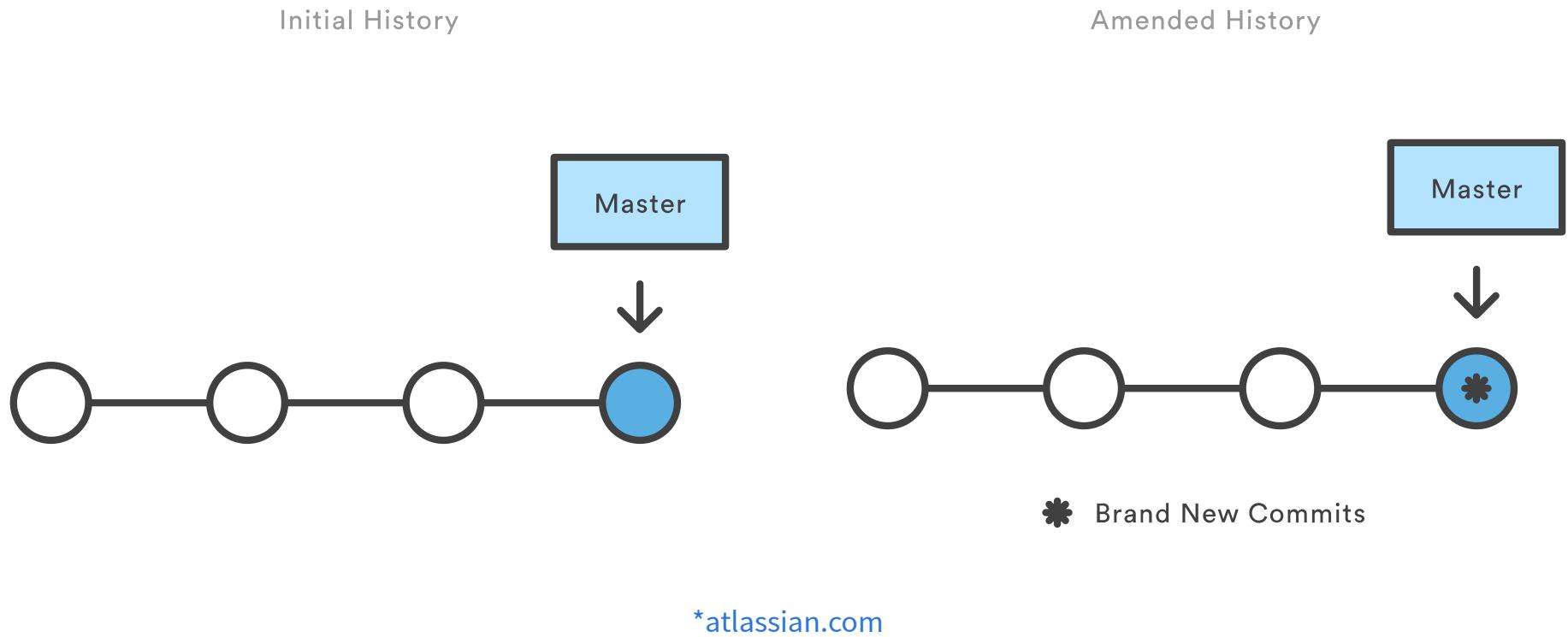
GIT - ZWALIŁEM OSTATNI COMMIT ;-(

git commit --ammend zamienia ostatni commit na nowy

- Ostatni commit jest wadliwy / Ma błędny komentarz
 - Poprawiamy kod w lokalnym katalogu
 - Zaznaczamy pliki do commitu (`git add`)
 - Wrzucamy poprawkę do repozytorium: `git commit --ammend`

Pozwala na prostą poprawkę / zmianę komentra

GIT - ZWALIŁEM OSTATNI COMMIT ;-(



GIT - ZMIANY W STARSZYCH COMMITACH

git rebase --interactive - pozwala na interaktywne wybieranie commitów
oraz ich edycję

```
git rebase --interactive <rodzic_commitu_z_błędem>
```

- Otwiera edytor -> Każda linia to jeden commit
- Usunięcie linii usuwa commit
- Możemy zmienić kolejność
- Dla każdej linii zostanie wykonane polecenie:

```
Commands:  
p, pick = use commit  
r, reword = use commit, but edit the commit message  
e, edit = use commit, but stop for amending  
s, squash = use commit, but meld into previous commit  
f, fixup = like "squash", but discard this commit's log message  
x, exec = run command (the rest of the line) using shell
```

GIT REBASE --INTERACTIVE

Konfigurujemy git-a aby korzystał z VSC jako diff-tool dla aktualnego projektu:

```
git config core.editor "code --wait"  
git config -e
```

Dodajemy nowe sekcje w pliku konfiguracyjnym:

```
[diff]  
  tool = default-difftool  
[difftool "default-difftool"]  
  cmd = code --wait --diff $LOCAL $REMOTE
```

GIT REBASE --INTERACTIVE

Możemy też zmienić globalną konfigurację git-a aby zawsze korzystał z VSC jako diff-tool:

```
git config --global core.editor "code --wait"  
git config --global -e
```

Dodajemy nowe sekcje w pliku konfiguracyjnym:

```
[diff]  
    tool = default-difftool  
[difftool "default-difftool"]  
    cmd = code --wait --diff $LOCAL $REMOTE
```

GIT REBASE --INTERACTIVE

Ćwiczenie - wyedytujmy 4 ostatnie commity:

```
git rebase --interactive HEAD~4
```

Przykładowe zmiany:

```
reword 9c58f7e Poprawki
squash 91814c9 Poprawki 2
edit dcc0216 Poprawka przekazywanie słownika
pick 2b43748 Aktualizacja dokumentacji
```

GIT REBASE --INTERACTIVE

Dla *reword* i *squash* otworzy się edytor z zawartością komentarza:

- Edytujemy komentarz
- Zapisujemy plik
- Zamykamy okno pliku
- Git automatycznie kontynuuje operację rebase

GIT REBASE --INTERACTIVE

W przypadku ***edit*** git zatrzyma się:

- Edytujemy wybrane pliki
- Dodajemy zmiany do indeksu:

```
git add
```

- Zapisujemy zmiany jako commit:

```
git commit --ammend
```

- Kontynuujemy:

```
git rebase --continue
```

GIT REBASE --INTERACTIVE

W przypadku wystąpienia konfliktów:

- Rozwiązujemy konflikty np.: z pomocą Visual Studio Code
- Oznaczamy rozwiążane konflikty:

```
git add
```

- Kontynuujemy:

```
git rebase --continue
```

GIT STASH

git stash zapis stanu katalogu roboczego (oraz stanu indeksu) w "schowku"

- polecenie `git stash` jest przydatne jeśli zaistnieje potrzeba powrotu do któregoś z poprzednich commitów w trakcie pracy nad nowym.

Zapisanie stanu katalogu roboczego:

```
git stash  
git stash list
```

Przywrócenie ostatniego stanu i usunięcie go ze stosu:

```
git stash pop
```

Inne:

```
git stash apply stash@{2}  # przywrócenie stanu katalogu roboczego  
git stash drop stash@{0}  # usunięcie zapisanego stanu ze stosu  
git stash pop stash@{1}  # przywrócenie konkretnego stanu i usunięcie go ze stosu
```

Komendy `git stash apply`, `git stash drop` i `git stash pop` wykonane bez określania konkretnego elementu stosu, będą działały na ostatnim dodanym elemencie

GIT - BUGI

git blame - pokazuje kto zmienił każdą linijkę w pliku i kiedy

git bisect - wykorzystuje metodę bisekcji (na grafach) do wykrycia w którym miejscu po raz pierwszy pojawił się błąd

GIT BISECT

Znamy ostatni dobry commit oraz commit gdzie objawia się błąd:



*paradigmadigital.com

GIT BISECT

Startujemy bisekcję:

```
git bisect start  
git bisect bad          # Aktualna wersja jest zła  
git bisect good v2.6.13-rc2 # Wiemy że tag v2.6.13-rc2 jest dobry
```

git bisect będzie checkout-ował kolejne commit-y z pomiędzy ostatniego dobrego i najstarszego złego starając się zminimalizować liczbę commitów do przejrzenia



GIT BISECT

Po weryfikacji oznaczamy commit jako dobry lub zły

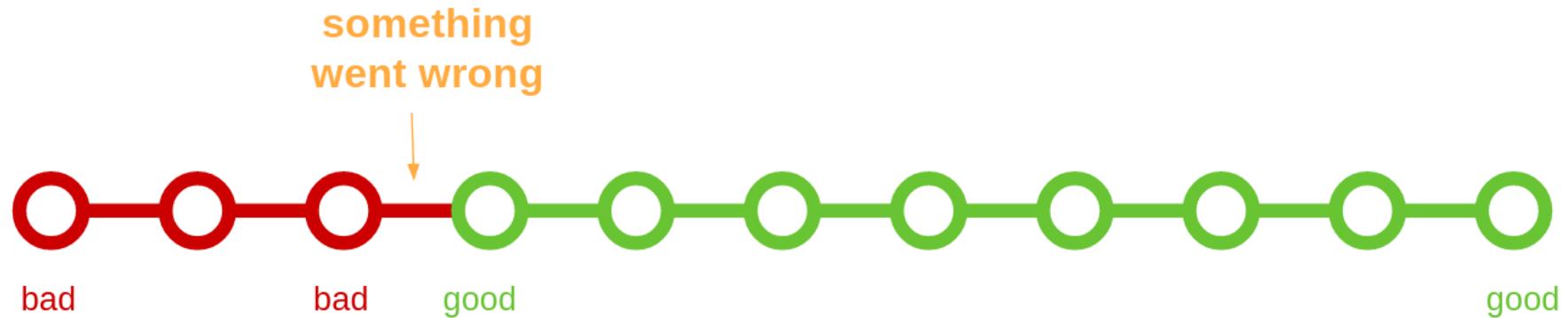
```
git bisect good  # Jeśli wersja jest dobra  
git bisect bad   # Jeśli wersja jest zła
```



*paradigmadigital.com

GIT BISECT

W końcu docieramy do commit-u który wprowadził błąd do kodu



*paradigmadigital.com

THANKS



```
git clone https://github.com/cis-ncbj/git-introduction.git
cd git-introduction
docker run -d -p 80:80 --name slides \
-v ${PWD}:/usr/share/nginx/html nginx
```