

# Podstawy Git/Github

## Warsztaty narzędzi inżynierii oprogramowania

### Co to jest Kontrola Wersji?

Kontrola wersji to system, który rejestruje zmiany w plikach na przestrzeni czasu, dzięki czemu można przywrócić wcześniejsze wersje, śledzić historię modyfikacji oraz współpracować z innymi osobami nad tym samym projektem. Jest niezbędnym narzędziem w programowaniu i zarządzaniu projektami, ponieważ:

- **Zapewnia bezpieczeństwo danych:** Pozwala odzyskać wcześniejsze wersje plików w przypadku błędów lub utraty danych.
- **Ułatwia współpracę:** Wielu programistów może pracować nad tym samym kodem jednocześnie, bez ryzyka nadpisania zmian innych osób.
- **Śledzi historię projektu:** Każda zmiana jest zapisywana z informacjami o autorze, czasie i opisie, co ułatwia analizę i debugowanie.

### Co to jest Git?

Git to rozproszony system kontroli wersji stworzony przez Linusa Torvaldsa w 2005 roku, pierwotnie na potrzeby rozwoju jądra Linux. Jest jednym z najpopularniejszych systemów kontroli wersji na świecie i stał się standardem w branży programistycznej.

### Kluczowe cechy Gita:

- **Rozproszony charakter:** Każde lokalne repozytorium jest pełną kopią projektu, co zwiększa bezpieczeństwo i niezależność od centralnego serwera.
- **Wydajność:** Git został zaprojektowany z myślą o szybkości i efektywności nawet przy bardzo dużych projektach.

### Krótką Historia Gita

- **2005:** Linus Torvalds tworzy Git jako narzędzie do zarządzania rozwojem jądra Linux po zerwaniu współpracy z komercyjnym systemem BitKeeper.
- **2005-2007:** Społeczność open-source szybko adaptuje Gita, rozwijając jego funkcjonalności i poprawiając dokumentację.
- **2008:** Powstaje GitHub, platforma hostingowa dla repozytoriów Gita, która popularyzuje system wśród programistów na całym świecie.
- **2010 i później:** Git staje się de facto standardem kontroli wersji w wielu projektach komercyjnych i open-source.

# Rozpoczynamy Pracę z Gitem

## Sprawdzenie Instalacji Gita

Zanim zaczniemy korzystać z Gita, upewnijmy się, że jest on zainstalowany na naszym komputerze.

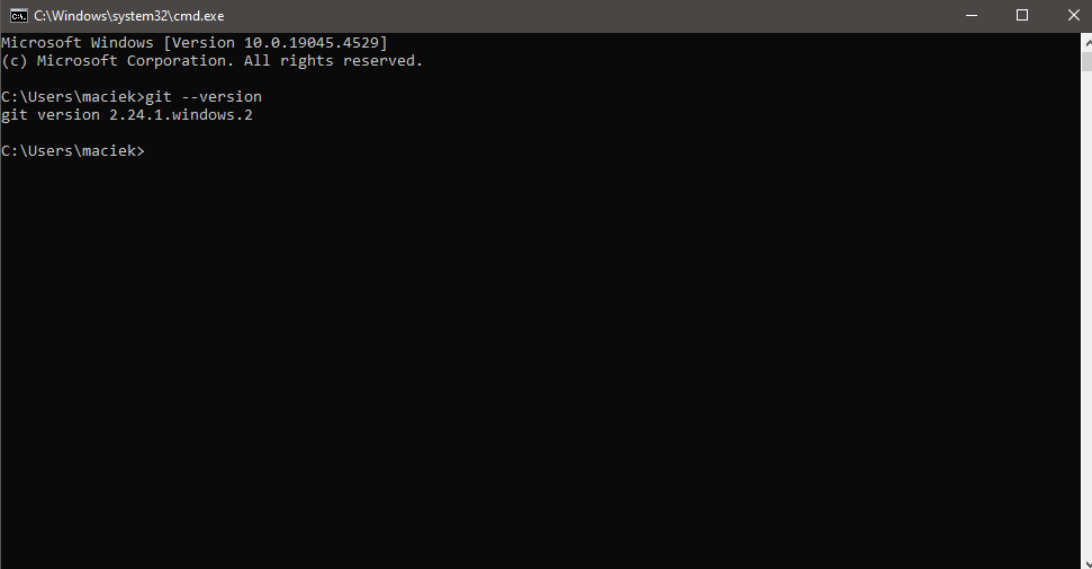
Krok 1: Otwórz Terminal lub Wiersz Poleceń

- **Windows:** Naciśnij **Win + R**, wpisz **cmd** i naciśnij **Enter**

Krok 2: Sprawdź Wersję Gita

Wpisz poniższe polecenie i naciśnij **Enter**:

**git --version**



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

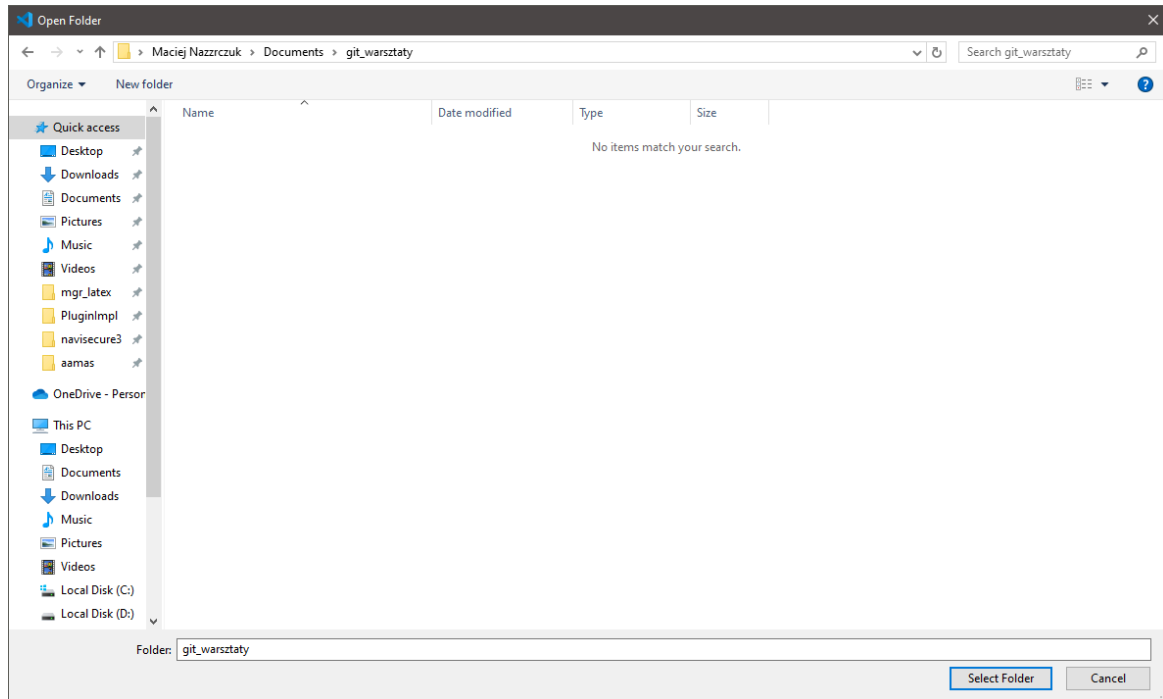
C:\Users\maciek>git --version
git version 2.24.1.windows.2

C:\Users\maciek>
```

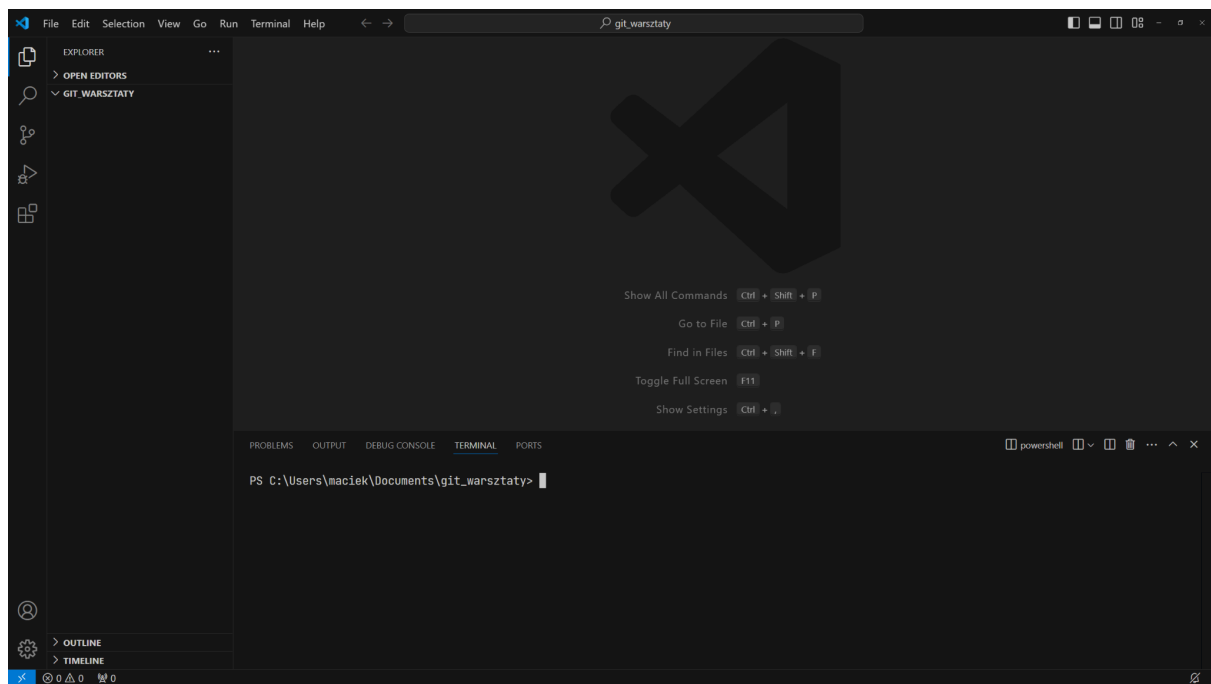
## Zainicjalizowanie nowego repozytorium

Zacznijmy od stworzenia nowego folderu, na którym będziemy dzisiaj pracować.

Na przykład w dokumentach stwórzcie nowy folder nazwany `git_warsztaty`, a następnie otwórzcie go w Visual Studio Code (**File** -> **Open Folder**).



Otwórzmy od razu okno terminala (**Terminal** -> **New Terminal**).



## Inicjalizacja repozytorium

Inicjalizacji repozytorium możemy dokonać z poziomu terminala, lub z poziomu interfejsu graficznego (GUI). Zacznijmy od wykorzystania terminala - dzięki temu lepiej zrozumiecie jak działa git, a nabyte umiejętności będą łatwiejsze do wykorzystania w przyszłości przy pracy z innymi edytorami.

W tym celu w terminalu wpisujemy:

```
git init
```

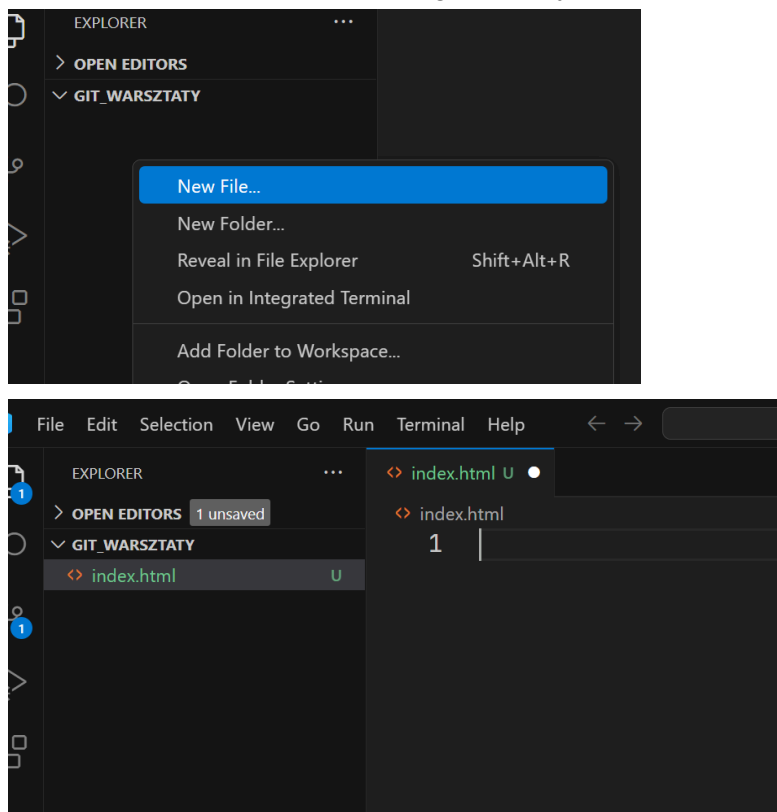
Powinniśmy uzyskać komunikat o sukcesie.

```
• PS C:\Users\maciek\Documents\git_warsztaty> git init
  Initialized empty Git repository in C:/Users/maciek/Documents/git_warsztaty/.git/
○ PS C:\Users\maciek\Documents\git_warsztaty> █
```

## Dodawanie plików do repozytorium i pierwszy commit

Nasze repozytorium jest puste. Dodajmy do niego plik.

W VS Code z menu kontekstowego klikamy New File, nazwijmy go index.html



W pliku możemy wpisać `html:5`, a następnie nacisnąć `tab`, żeby dostać szkielet strony w html.



```
> index.html
1  html:5
    html:5 Emmet Abbreviation
```

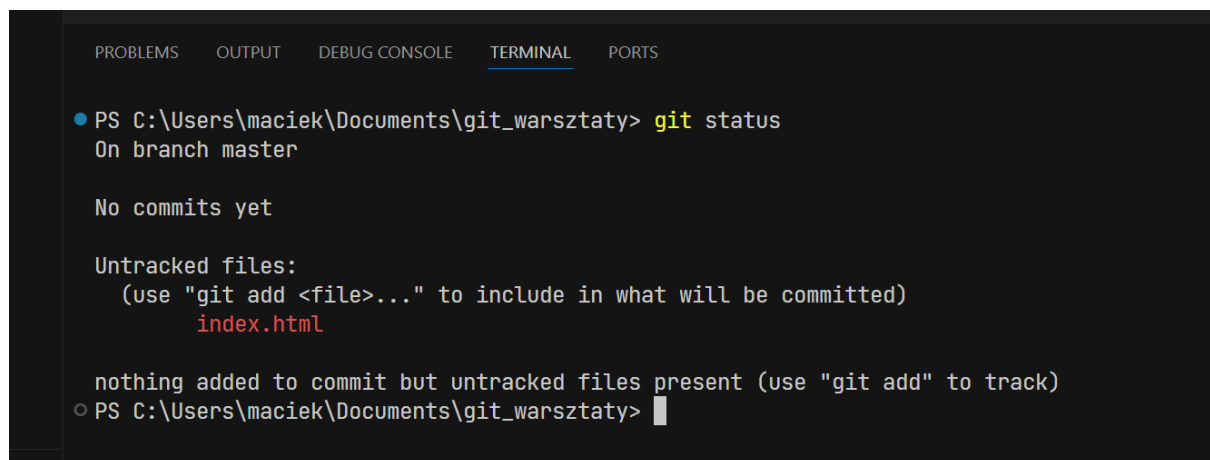
  

```
EXPLORER
  > OPEN EDITORS 1 unsaved
  > GIT_WARSZTATY
    <> index.html U

index.html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Document</title>
7  </head>
8  <body>
9
10 </body>
11 </html>
```

W tym momencie mamy pierwszy plik w naszym projekcie. Nie jest on jednak zapisany w naszym repozytorium. Możemy to sprawdzić poleceniem:

```
git status
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● PS C:\Users\maciek\Documents\git_warsztaty> git status
On branch master

No commits yet

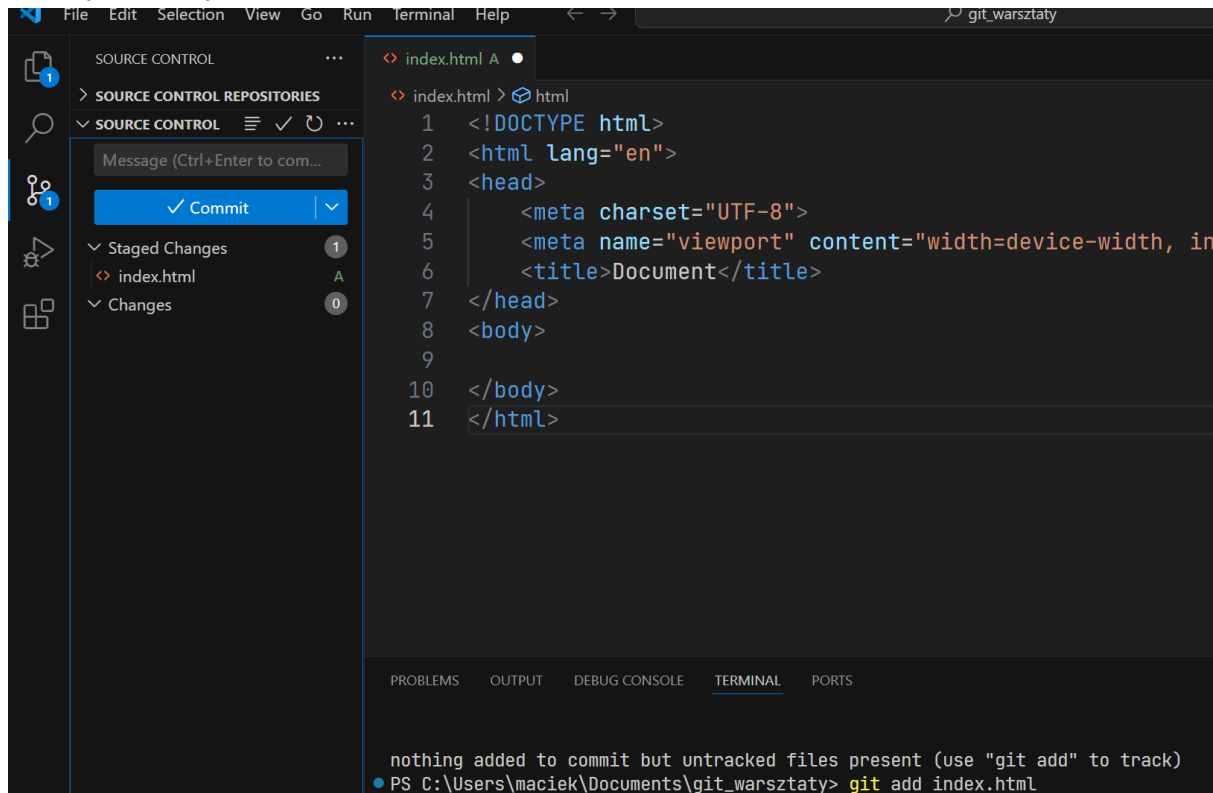
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    index.html

nothing added to commit but untracked files present (use "git add" to track)
○ PS C:\Users\maciek\Documents\git_warsztaty> 
```

Git mówi nam, że mamy jeden nieśledzony plik i wskazuje nam, że możemy dodać go do śledzonych plików poleceniem:

```
git add index.html
```

Zmiany możemy też śledzić z poziomu GUI



Plik index.html przeszedł z zakładki Changes do Staged Changes. Staged Changes są to zmiany gotowe do zapisania w naszym repozytorium (**commitowanie zmian**).

### Commit

W Git to zapis stanu całego projektu w określonym momencie. Można go porównać do zdjęcia (snapshotu) wszystkich śledzonych plików w repozytorium. Commit przechowuje informacje o zmianach dokonanych od ostatniego commita, a także metadane, takie jak:

- **Autor zmian:** kto dokonał commita.
- **Czas:** kiedy commit został wykonany.
- **Wiadomość:** krótki opis wprowadzonych zmian.
- **Unikalny identyfikator (hash):** 40-znakowy kod (np. `1a2b3c4d5e6f7g8h9i0jklmnopqrstuvw`), który jednoznacznie identyfikuje dany commit.

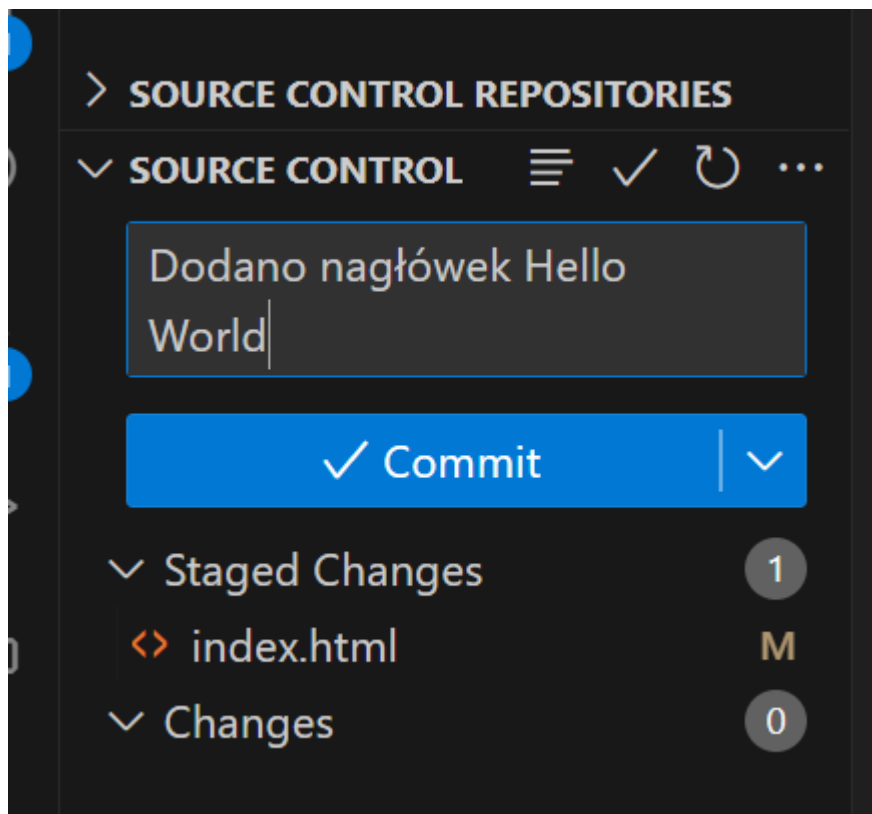
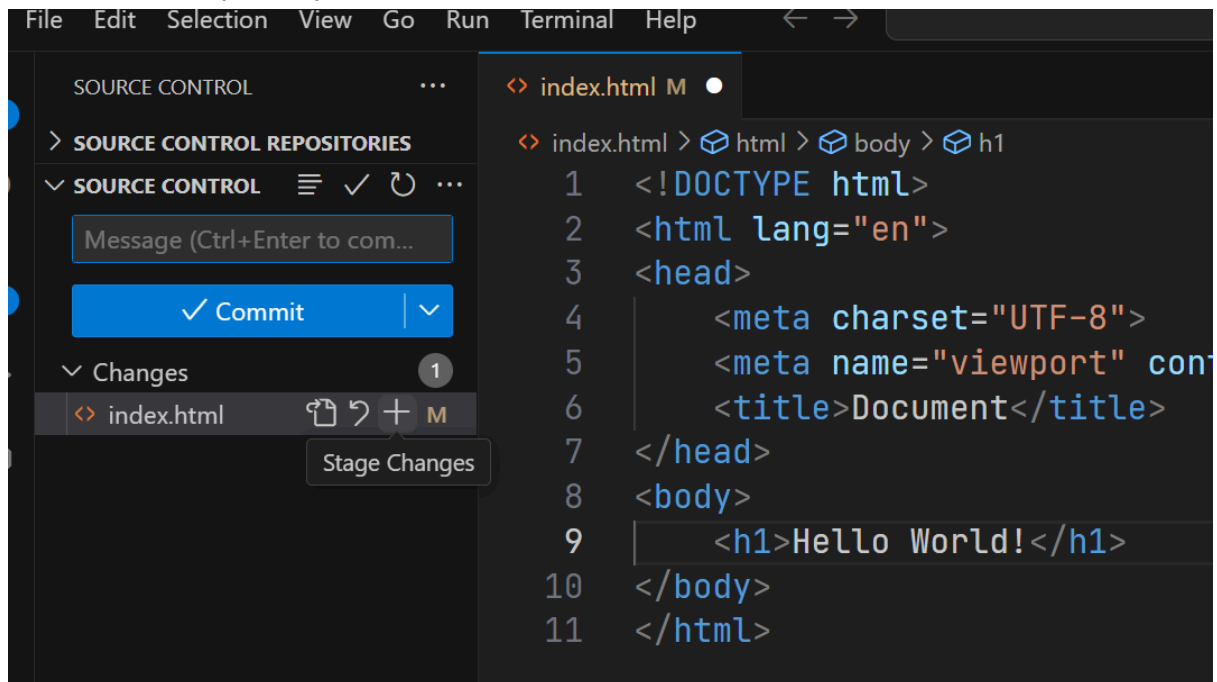
W celu zacommitowania zmian wpisujemy w terminalu:

```
git commit -m "Pierwszy commit"
```

```
PS C:\Users\maciek\Documents\git_warsztaty> git commit -m "Pierwszy commit"
• [master (root-commit) bdf45b4] Pierwszy commit
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 index.html
○ PS C:\Users\maciek\Documents\git_warsztaty>
```

Alternatywnie możemy commitować zmiany z poziomu GUI, w tym celu wprowadźmy zmiany w naszym pliku index.html (np. dodając nagłówek "Hello World!")

W zakładce Source Control dodajemy nasz plik do **Stage'a**, wprowadzamy opis naszego commita i klikamy w przycisk Commit.



## Zadanie 1:

Zacommittujcie przynajmniej 3 zmiany w pliku index.html.

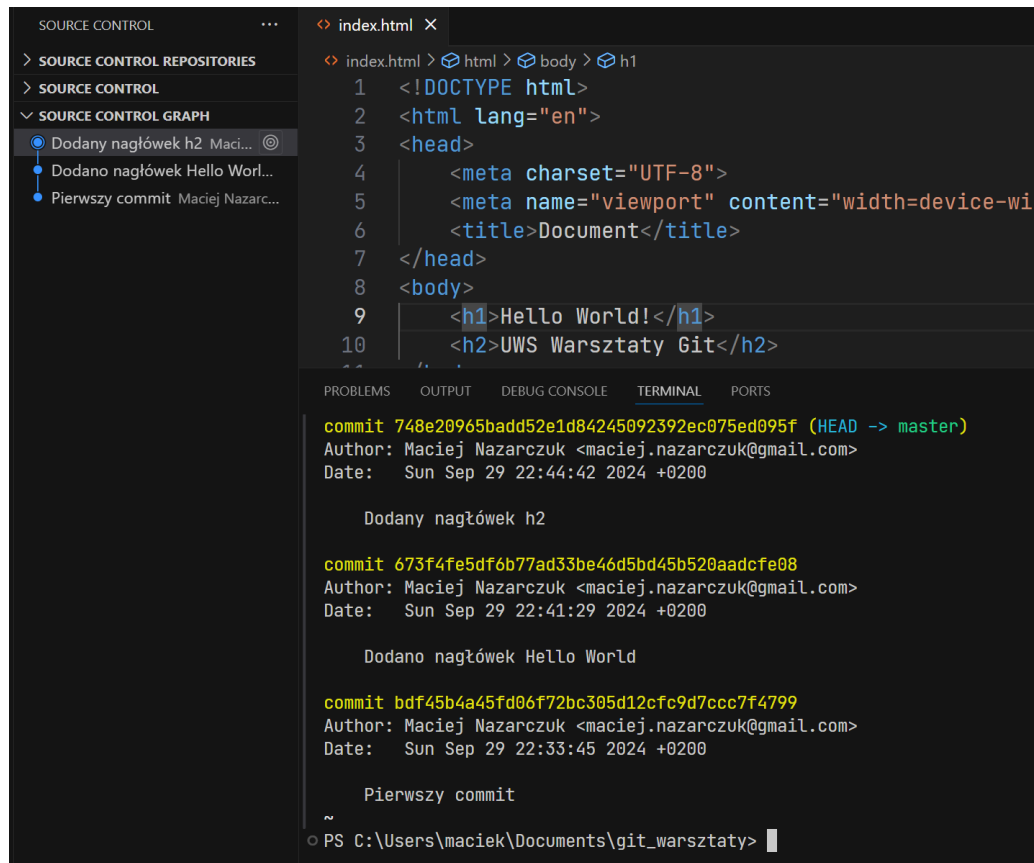
Wykorzystajcie zarówno GUI, jak i komendy w wierszu poleceń.

## Historia zmian

Historię commitów możemy zobaczyć w za pomocą polecenia:

```
git log
```

Lub w GUI:



```
commit 748e20965badd52e1d84245092392ec075ed095f (HEAD -> master)
Author: Maciej Nazarczuk <maciej.nazarczuk@gmail.com>
Date: Sun Sep 29 22:44:42 2024 +0200

    Dodany nagłówek h2

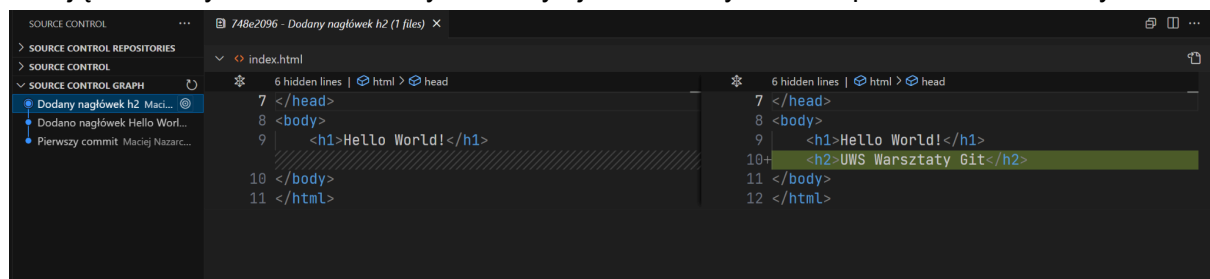
commit 673f4fe5df6b77ad33be46d5bd45b520aadcf08
Author: Maciej Nazarczuk <maciej.nazarczuk@gmail.com>
Date: Sun Sep 29 22:41:29 2024 +0200

    Dodano nagłówek Hello World

commit bdf45b4a45fd06f72bc305d12cfc9d7ccc7f4799
Author: Maciej Nazarczuk <maciej.nazarczuk@gmail.com>
Date: Sun Sep 29 22:33:45 2024 +0200

    Pierwszy commit
```

Klikając na dany commit możemy zobaczyć jakie zostały w nim wprowadzone zmiany:



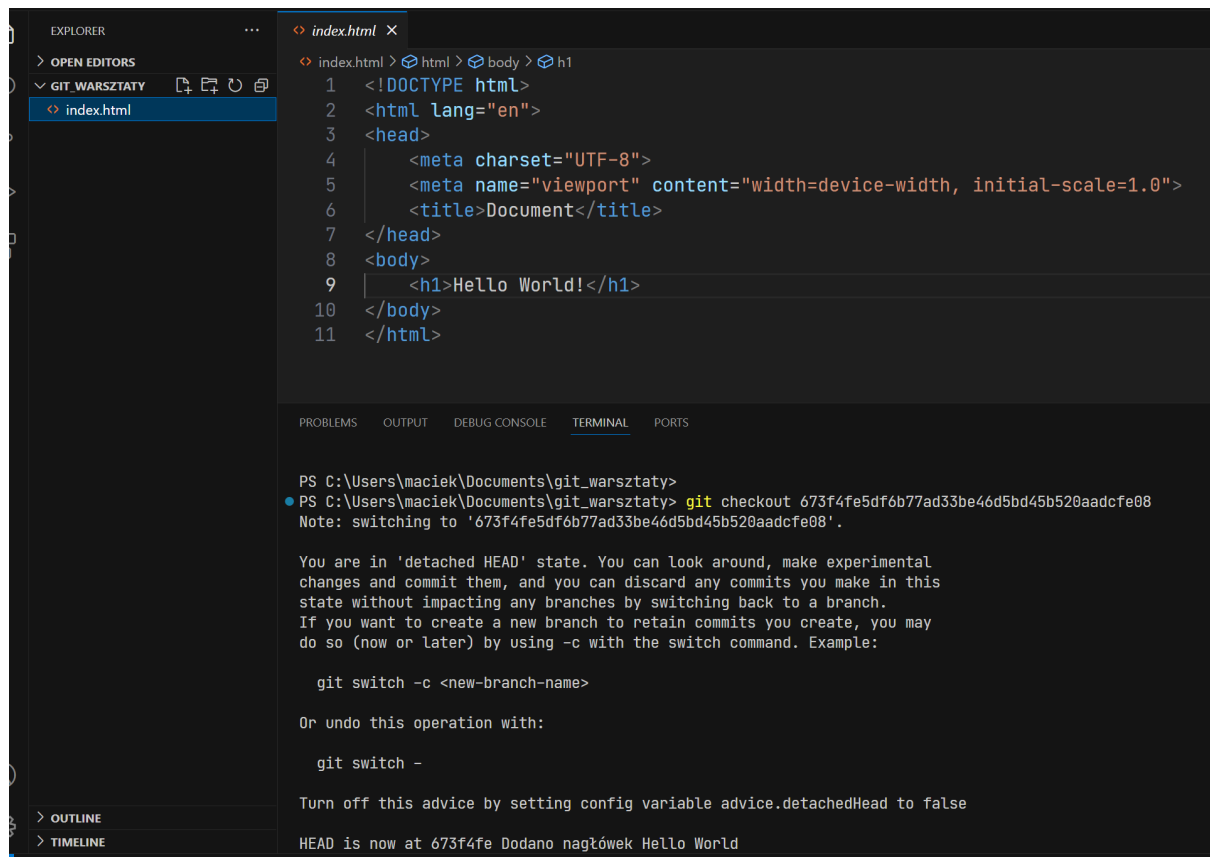
```
7 </head>
8 <body>
9 | <h1>Hello World!</h1>
10 </body>
11 </html>
```



Git pozwala nam na łatwe cofnięcie się do poprzedniej wersji naszego projektu. W tym celu możemy skopiować commit id dla commita, do którego chcemy wrócić, następnie poleceniem:

```
git checkout COMMIT_ID
```

Możemy wrócić do danego commita.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a file named 'index.html'. The main editor area displays the content of 'index.html', which is an HTML document with a single heading 'Hello World!'. Below the editor, the Terminal pane shows the output of a 'git checkout' command. The command was successful, and the terminal output indicates that the HEAD is now at a specific commit hash. The output also includes a warning about the 'detached HEAD' state and instructions on how to switch back to a branch or create a new one.

```
PS C:\Users\maciek\Documents\git_warsztaty>
● PS C:\Users\maciek\Documents\git_warsztaty> git checkout 673f4fe5df6b77ad33be46d5bd45b520aadcf08
Note: switching to '673f4fe5df6b77ad33be46d5bd45b520aadcf08'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 673f4fe Dodano nagłówek Hello World
```

Jak widać z pliku zniknął nagłówek 2giego poziomu.

## DETACHED HEAD

Co to jest "detached HEAD"?

W normalnej pracy z Gitem **HEAD** to wskaźnik, który wskazuje na ostatni commit w bieżącej gałęzi (branchu). O gałęziach powiemy za chwilę. Kiedy jesteśmy na jakiejś gałęzi, **HEAD** wskazuje na tę gałąź.

W stanie "**detached HEAD**":

- **HEAD** wskazuje bezpośrednio na konkretny commit, a nie na gałąź.
- Nie jesteśmy na żadnej gałęzi, tylko na konkretnym commicie.
- Zmiany, które wprowadzisz i zapiszesz (commitujesz), nie są powiązane z żadną gałęzią.

Co możemy robić w tym stanie?

- **Przeglądać kod:** Możesz zobaczyć stan kodu w konkretnym commicie.
- **Bez wpływu na gałęzie:** Zmiany nie wpłyną na żadną istniejącą gałąź.
- **Eksperymentować**

**Jednak uwaga:** Jeśli nie podejmiesz dodatkowych kroków, zmiany mogą zostać utracone po opuszczeniu tego stanu.

Jak zachować zmiany?

Jeśli chcesz, aby Twoje zmiany zostały zachowane i powiązane z gałęzią, musisz utworzyć nową gałąź lub przełączyć się na istniejącą.

Co to są gałęzie (branches) w Git?

**Gałęzie** w Git to niezależne linie rozwoju projektu. Pozwalają na pracę nad różnymi funkcjonalnościami, poprawkami czy eksperymentami bez wpływu na główną wersję kodu.

- **Gałąź `master` lub `main`:** Domyślna, główna gałąź projektu.
- **Inne gałęzie:** Tworzone w celu rozwijania nowych funkcji, naprawiania błędów czy testowania pomysłów.
- **Łączenie gałęzi (merge):** Pozwala na połączenie zmian z różnych gałęzi w jedną.

Gałęzie ułatwiają organizację pracy i współpracę między programistami.

Tworzenie nowej gałęzi, aby zachować zmiany

Aby zachować zmiany wykonane w stanie "detached HEAD", utwórz nową gałąź na podstawie bieżącego stanu.

Utwórz nową gałąź z obecnego stanu poleceniem:

```
git switch -c NAZWA_GALEZI
```

```
• PS C:\Users\maciek\Documents\git_warsztaty> git switch -c test  
Switched to a new branch 'test'
```

Teraz:

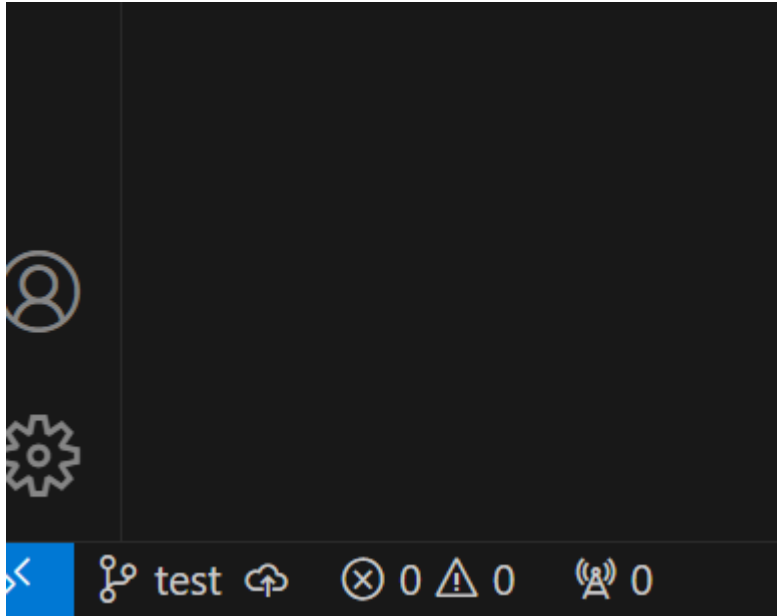
- Git utworzy nową gałąź o nazwie `NAZWA_GALEZI`
- **HEAD** będzie wskazywał na tę nową gałąź.
- Twoje zmiany zostaną zapisane na tej gałęzi.

Aktualną gałąź możemy sprawdzić poleceniem:

```
git branch
```

```
PS C:\Users\maciek\Documents\git_warsztaty> git branch
master
* test
```

Lub w GUI w lewym dolnym rogu ekranu:



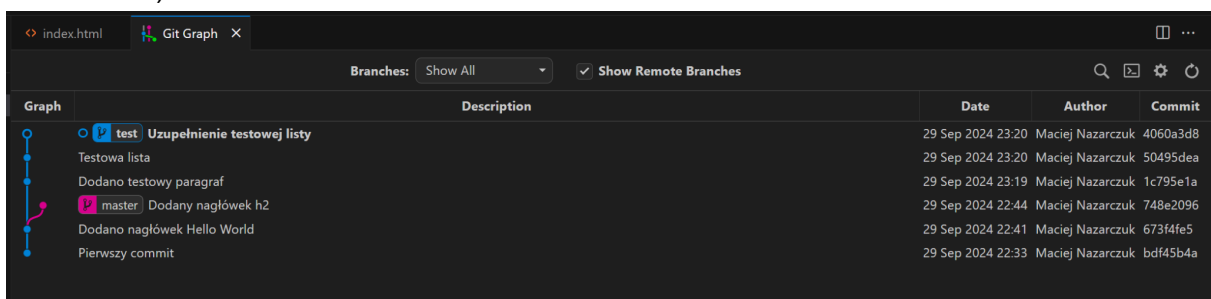
Dodajmy teraz, a następnie zacommitujmy kilka zmian na naszej testowej gałęzi.

Historię zmian w postaci drzewa możemy zobaczyć poleceniem:

```
git log --all --decorate --oneline --graph

PS C:\Users\maciek\Documents\git_warsztaty> git log --all --decorate --oneline --graph
* 4060a3d (HEAD -> test) Uzupełnienie testowej listy
* 50495de Testowa lista
* 1c795e1 Dodano testowy paragraf
| * 748e209 (master) Dodany nagłówek h2
|/
* 673f4fe Dodano nagłówek Hello World
* bdf45b4 Pierwszy commit
PS C:\Users\maciek\Documents\git_warsztaty>
```

Albo instalując rozszerzenie do VS Code, np. Git Graph (po zainstalowaniu znajduje się na dole ekranu).



# Łączenie zmian z dwóch gałęzi MERGE

## Co to jest Merging?

**Merging** w Git to proces łączenia zmian z jednej gałęzi do innej. Pozwala na integrację pracy wykonanej niezależnie na różnych gałęziach, tworząc spójną historię projektu.

## Dlaczego Merging jest ważny?

- **Współpraca:** Umożliwia integrację zmian dokonanych przez różnych członków zespołu.
- **Rozwój funkcjonalności:** Pozwala na pracę nad nowymi funkcjami na osobnych gałęziach, które można później połączyć z główną gałęzią.
- **Zarządzanie wersjami:** Ułatwia utrzymanie stabilnej wersji projektu, podczas gdy nowe funkcje są rozwijane i testowane na oddzielnych gałęziach.

## Podstawowe pojęcia:

- **Gałąź (branch):** Niezależna linia rozwoju projektu.
- **Gałąź bazowa:** Gałąź, do której chcesz wprowadzić zmiany (np. `master` lub `main`).
- **Gałąź źródłowa:** Gałąź, z której chcesz przenieść zmiany.

## Proces Mergowania

### Krok 1: Przejdź do gałęzi bazowej

Zanim zaczniesz proces mergowania, upewnij się, że znajdujesz się na gałęzi, do której chcesz wprowadzić zmiany.

```
PS C:\Users\maciek\Documents\git_warsztaty> git switch master
Switched to branch 'master'
PS C:\Users\maciek\Documents\git_warsztaty> 
```

### Krok 2: Wykonaj merge z gałęzi źródłowej

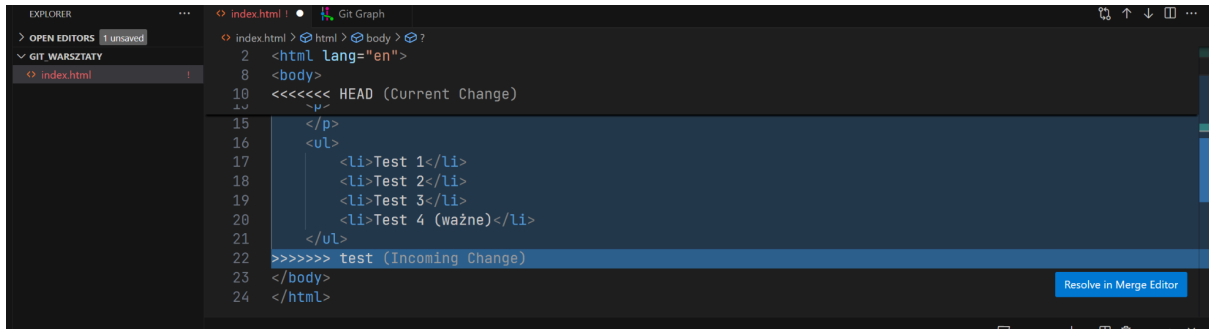
Użyj komendy `git merge`, aby połączyć zmiany z gałęzi źródłowej.

```
PS C:\Users\maciek\Documents\git_warsztaty> git merge test
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\maciek\Documents\git_warsztaty> 
```

Dostaliśmy informację, że wystąpiły **konflikty**.

Jeśli te same fragmenty plików zostały zmodyfikowane w obu gałęziach, Git nie będzie mógł automatycznie połączyć zmian i wystąpi konflikt.

Przy rozwiązywaniu konfliktów warto wykorzystać interfejs graficzny:

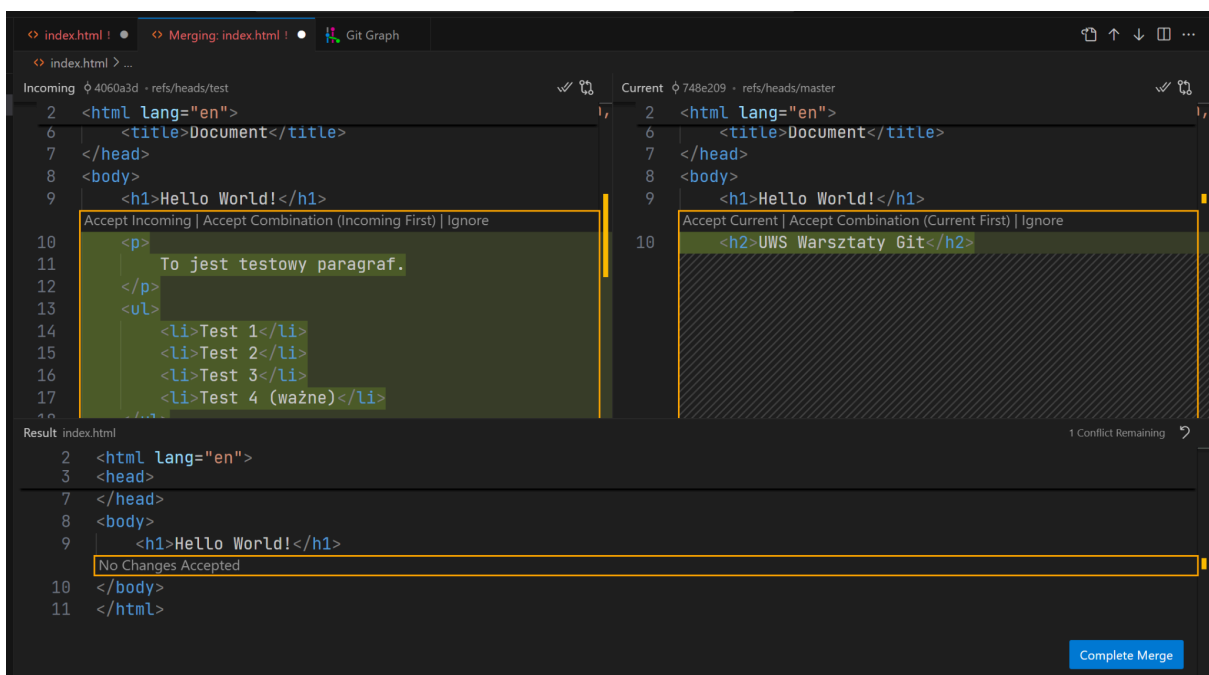


```
2 <html lang="en">
8 <body>
10 <----- HEAD (Current Change)
15 </p>
16 <ul>
17 <li>Test 1</li>
18 <li>Test 2</li>
19 <li>Test 3</li>
20 <li>Test 4 (ważne)</li>
21 </ul>
22 >>>>>> test (Incoming Change)
23 </body>
24 </html>
```

Po lewej stronie mamy zmiany przychodzące z gałęzi, którą chcemy zmerge'ować do gałęzi master.

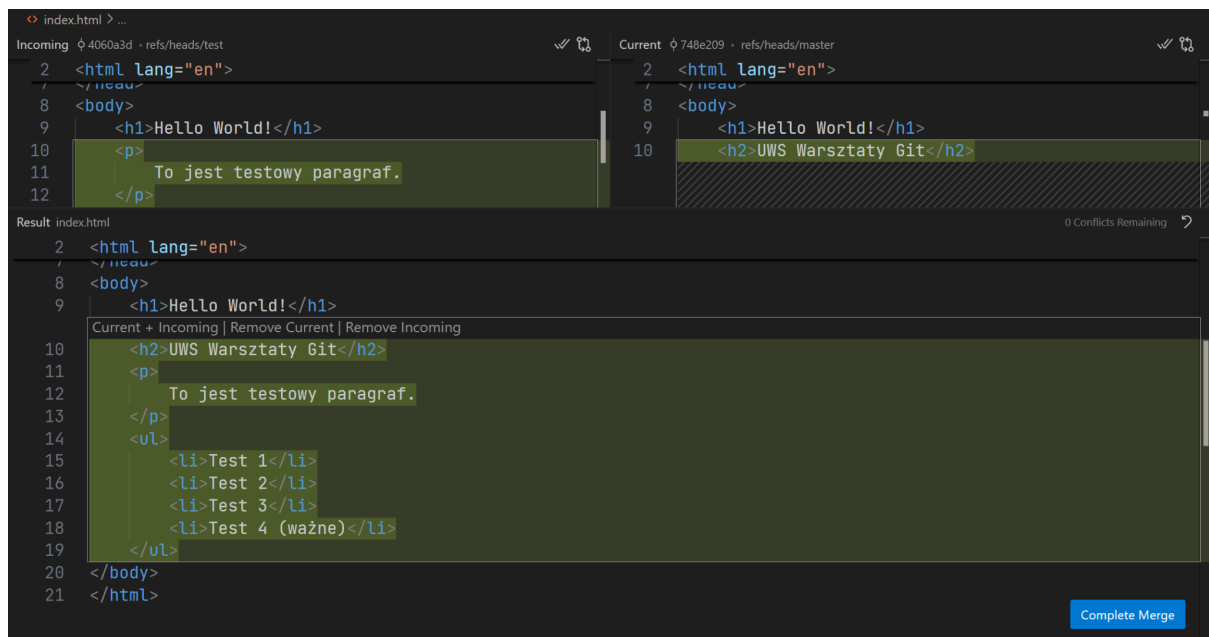
Po prawej stronie mamy gałąź master z zaznaczonym fragmentem konfliktującym.

Na dole mamy wynikowy plik po merge'u.



```
2 <html lang="en">
3 <head>
7 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <p>
11 To jest testowy paragraf.
12 </p>
13 <ul>
14 <li>Test 1</li>
15 <li>Test 2</li>
16 <li>Test 3</li>
17 <li>Test 4 (ważne)</li>
18 </ul>
19 </body>
20 </html>
```

Zaakceptujmy zmiany z obu gałęzi:



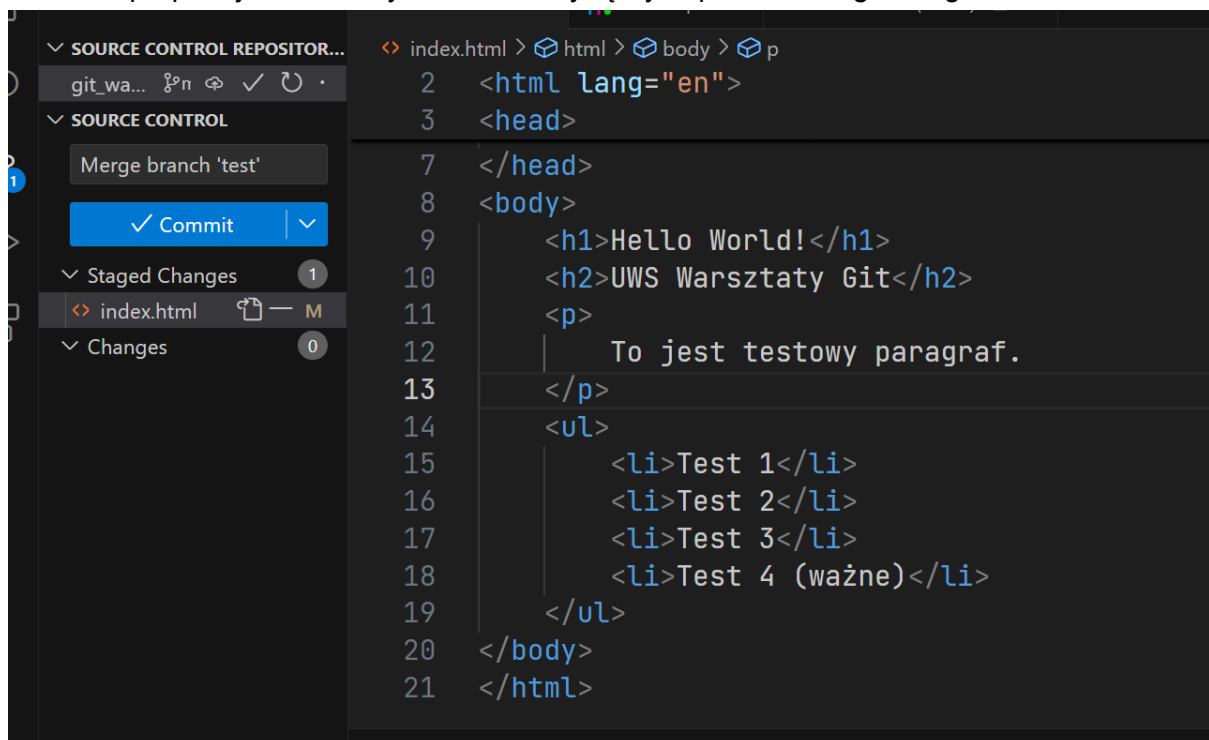
```
index.html > ...
Incoming 4060a3d · refs/heads/test
2 <html lang="en">
3 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <p>
11 | To jest testowy paragraf.
12 </p>

Current 748e209 · refs/heads/master
2 <html lang="en">
3 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <h2>UWS Warsztaty Git</h2>

Result index.html
0 Conflicts Remaining
2 <html lang="en">
3 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <h2>UWS Warsztaty Git</h2>
11 <p>
12 | To jest testowy paragraf.
13 </p>
14 <ul>
15 <li>Test 1</li>
16 <li>Test 2</li>
17 <li>Test 3</li>
18 <li>Test 4 (ważne)</li>
19 </ul>
20 </body>
21 </html>
Complete Merge
```

Gdy rozwiązaliśmy wszystkie konflikty możemy zakończyć mergowanie.

VS Code proponuje nam nowy commit, który łączy zapisze naszego merge'a.

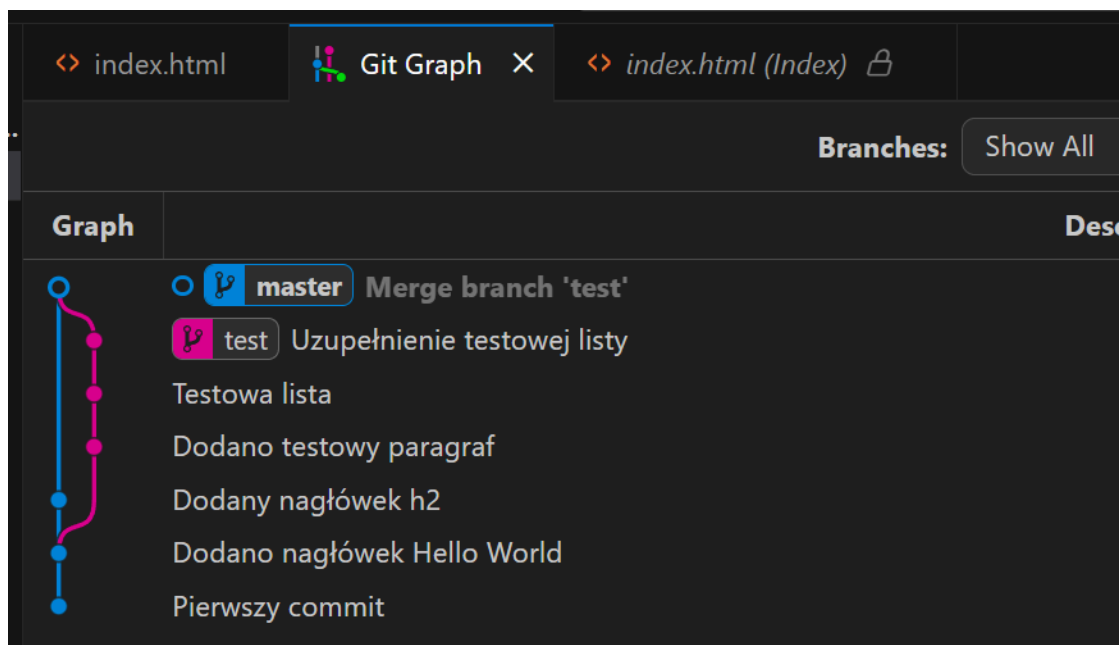


```
index.html > html > body > p
2 <html lang="en">
3 <head>
7 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <h2>UWS Warsztaty Git</h2>
11 <p>
12 | To jest testowy paragraf.
13 </p>
14 <ul>
15 <li>Test 1</li>
16 <li>Test 2</li>
17 <li>Test 3</li>
18 <li>Test 4 (ważne)</li>
19 </ul>
20 </body>
21 </html>
```

SOURCE CONTROL REPOSITORY: git\_wa...  
SOURCE CONTROL: Merge branch 'test', Commit, Staged Changes (1), index.html (M), Changes (0)

Tak wygląda historia po dołączeniu zmian:

```
PS C:\Users\maciek\Documents\git_warsztaty> git log --all --decorate --oneline --graph
*   b447a0b (HEAD -> master) Merge branch 'test'
| \
| * 4060a3d (test) Uzupełnienie testowej listy
| * 50495de Testowa lista
| * 1c795e1 Dodano testowy paragraf
| * | 748e209 Dodany nagłówek h2
| /
|
* 673f4fe Dodano nagłówek Hello World
* bdf45b4 Pierwszy commit
PS C:\Users\maciek\Documents\git_warsztaty>
```



### Zadanie 2 i 3:

Wprowadźcie i dodajcie 2 nowe branche. Zacomitujcie na nich pewne zmiany. Zmierzcie zmiany do głównej gałęzi, rozwiążcie przy tym konflikty.

# GitHub

## Co to jest GitHub?

**GitHub** to internetowa platforma hostingowa dla projektów programistycznych korzystających z systemu kontroli wersji Git. Umożliwia programistom przechowywanie kodu źródłowego w chmurze, współpracę nad projektami oraz korzystanie z narzędzi ułatwiających zarządzanie i rozwój oprogramowania.

### Kluczowe funkcje GitHuba:

- **Hostowanie repozytoriów Git:** Przechowuj swoje projekty online, dostępne z dowolnego miejsca.
- **Współpraca:** Pracuj z innymi programistami nad tym samym kodem, wykorzystując pull requesty, code review i inne narzędzia.
- **Zarządzanie projektami:** Korzystaj z issue trackerów, wiki, tablic Kanban i innych narzędzi do organizacji pracy.
- **Integracje:** Łącz GitHuba z innymi usługami, takimi jak Continuous Integration/Continuous Deployment (CI/CD), narzędzia do testowania i monitoringu.

### Dlaczego warto korzystać z GitHuba?

- **Centralne miejsce dla kodu:** Wszystkie Twoje projekty są dostępne w jednym miejscu, z możliwością zarządzania wersjami i historią zmian.
- **Współpraca społecznościowa:** GitHub ułatwia współpracę z innymi programistami na całym świecie, co jest szczególnie ważne w projektach open-source.
- **Widoczność i portfolio:** Publikując swoje projekty na GitHubie, budujesz swoje portfolio, które może być zauważone przez potencjalnych pracodawców.
- **Narzędzia wspomagające rozwój:** GitHub oferuje szereg narzędzi, takich jak automatyczne testy, analizę kodu, które pomagają w utrzymaniu wysokiej jakości oprogramowania.

### Zakładanie konta na GitHubie

1. Przejdź na stronę <https://github.com>.
2. Kliknij przycisk **Sign up** lub **Zarejestruj się**.
3. Wprowadź swój adres e-mail i kliknij **Continue**.
4. Ustaw nazwę użytkownika i hasło.
5. Przejdź przez proces weryfikacji i potwierdź adres e-mail.

### Zadanie 4:

Założcie konta na GitHubie



## Tworzenie nowego repozytorium na GitHubie

1. Po zalogowaniu się kliknij na ikonę **+** w prawym górnym rogu i wybierz **New repository**.
2. Uzupełnij pola:
  - **Repository name**: Nazwa Twojego repozytorium (np. **moj-projekt**).
  - **Description**: Krótki opis projektu (opcjonalnie).
3. Wybierz widoczność repozytorium:
  - **Public**: Każdy może zobaczyć Twoje repozytorium.
  - **Private**: Tylko Ty (i zaproszeni współpracownicy) mają dostęp.



## Połączenie lokalnego repozytorium z GitHubem

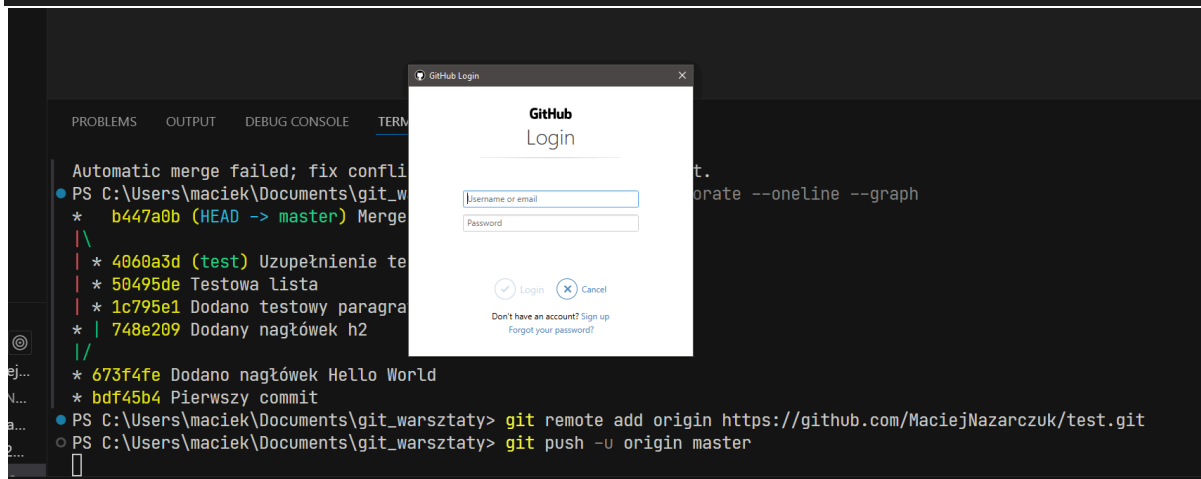
Jeśli masz już lokalne repozytorium Git, możesz je połączyć z nowo utworzonym repozytorium na GitHubie.

W terminalu dodajcie zdalne repozytorium:

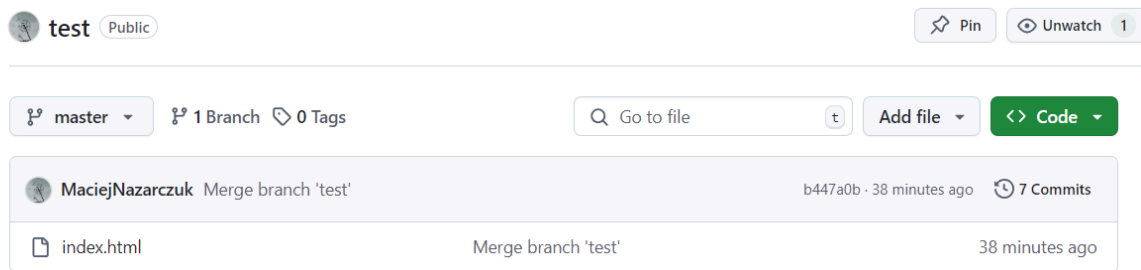
```
git remote add origin WASZ_ADRES_REPOZYTORIUM
```

Prześlijcie swoje zmiany na GitHuba:

```
git push -u origin master
```



W tym momencie nasz projekt powinien być widoczny na GitHubie.



Nie mamy na githubie naszej testowej gałęzi - nie jest nam ona potrzebna, w końcu zmergeowaliśmy wszystkie zmiany do gałęzi master.

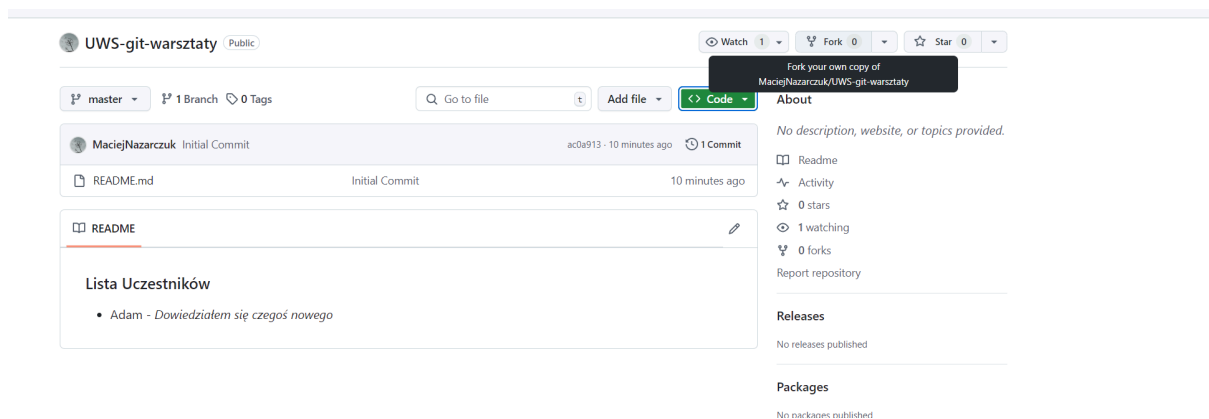
## Zadanie 5:

Zpushujcie swoje dotychczasowe repozytoria na GitHuba.

## Podstawowe operacje na GitHubie

### Forkowanie repozytorium

- **Fork** to kopia czyjegoś repozytorium na Twoim koncie GitHub.
- Umożliwia wprowadzanie własnych zmian bez wpływu na oryginalne repozytorium.
- Jak zforkować repozytorium:
  1. Przejdź do repozytorium, które chcesz zforkować.
  2. Kliknij przycisk **Fork** w prawym górnym rogu.



Pobranie repozytorium na lokalną maszynę:

W wierszu poleceń:

```
git clone WASZE_ZFORKOWANE_REPOZYTORIUM
```

```

PS C:\Users\maciek\Documents\git_warsztaty3> git clone https://github.com/MNazarczuk0/UWS-git-warsztaty.git
Cloning into 'UWS-git-warsztaty'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), done.
PS C:\Users\maciek\Documents\git_warsztaty3>

```

Po skopiowaniu repozytorium, dopiszcie siebie do listy uczestników, z jakimś miłym zdaniem komentarza ;).

Następnie zacommitujcie, a następnie zpushujcie zmiany na swoje zforkowane repozytorium.

```

PS C:\Users\maciek\Documents\git_warsztaty3\UWS-git-warsztaty> git push
info: please complete authentication in your browser...
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 336 bytes | 336.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MNazarczuk0/UWS-git-warsztaty.git
   ac0a913..5dc7823  master -> master
PS C:\Users\maciek\Documents\git_warsztaty3\UWS-git-warsztaty>

```

s
Projects
Wiki
Security
Insights
Settings

**UWS-git-warsztaty**
Public
Pin
Watch 0
Fork

forked from MaciejNazarczuk/UWS-git-warsztaty

master
1 Branch
0 Tags
Go to file
Add file
Code

This branch is 3 commits ahead of MaciejNazarczuk/UWS-git-warsztaty:master
Contribute
Sync fork

**MNazarczuk0**
Dodalem siebie
7b16b9a · now
5 Commits

README.md
Dodalem siebie
now

README

### Lista Uczestników

- Test - \*Napiszcie coś od siebie \* 😊
- Maciek - *Było super*

No description
Readme
Activity
0 stars
0 watch
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package

## Tworzenie pull requesta

- **Pull request** to prośba o włączenie Twoich zmian do oryginalnego repozytorium.
- Umożliwia współpracę i code review.
- Jak utworzyć pull request:
  1. Wprowadź zmiany w swoim forkowanym repozytorium.
  2. Przejdź do zakładki **Pull requests**.
  3. Kliknij **New pull request**.
  4. Wybierz gałąź, z której chcesz wprowadzić zmiany.
  5. Dodaj tytuł i opis, a następnie kliknij **Create pull request**.

Proces przedstawiono poniżej:

The screenshot displays the GitHub interface for creating and reviewing pull requests. At the top, the navigation bar includes links for Issues, Pull requests, Actions, Projects, Security, and Insights. A notification banner at the top center asks if the user is contributing for the first time. Below this, a search bar and filters are visible. The main content area features a 'Welcome to pull requests!' message, explaining that pull requests help collaborate on code. Below the welcome message, a comparison view is shown for a pull request. It includes a header with repository and branch information, a 'Create pull request' button, and a summary of changes (3 commits, 1 file changed, 2 contributors). The diff view shows changes to the README.md file, with a table of additions and deletions.

base repository: MaciejNazarczuk/UWS-git-war... base: master ← head repository: MNazarczuk0/UWS-git-warszt... compare: master

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) [Create pull request](#)

3 commits 1 file changed 2 contributors

Showing 1 changed file with 1 addition and 0 deletions. Split Unified

...	...	@@ -1,2 +1,3 @@
1	1	### Lista Uczestników
2	2	- Test - *Napiszcie coś od siebie 😊
3	+	- Maciek - *Było super*

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base repository: MaciejNazarczuk/UWS-git-war...

base: master

head repository: MNazarczuk0/UWS-git-warsztak...

compare: master

✓ Able to merge.

These branches can be automatically merged.

Add a title

Dodałem Siebie, MNazarczuk

Add a description

WritePreview

HBBIT=<>⌵≡≡≡≡≡≡≡@🔗↶✎

Add your description here...

📄 Markdown is supported

📁 Paste, drop, or click to add files

☑ Allow edits by maintainers

Create pull request

Remember, contributions to this repository should follow our GitHub Community Guidelines.

A poniżej jak to wygląda ze strony właściciela repozytorium, do którego trafia żądanie.

Dodałem Siebie, MNazarczuk #1

Open

MNazarczuk0 wants to merge 3 commits into #MaciejNazarczuk:master from #MNazarczuk:master

Conversation 0

Commits 3

Checks 0

Files changed 1

Changes from 1 commit · File filter · Conversations · Jump to ·

Dodałem siebie

MNazarczuk0 committed 4 minutes ago

▼ 1 README.md

```
-- @ -1,2 +1,3 @@  
1   ## Lista Uczestników  
2     Test - *Napiszcie coś od siebie *  
  
1   ## Lista Uczestników  
2     Test - *Napiszcie coś od siebie *  
2     . - Maciek - *Spis uczest.
```

[Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Filters

is:pr is:closed

Clear current search query, filters, and sorts

☐ 0 Open ☒ 1 Closed

☐ **Dodałem Siebie, MNazarczuk**  
#1 by MNazarczuk0 was merged now

**UWS-git-warsztaty** Public

[Pin](#) [Unwatch](#) 1

master 1 Branch 0 Tags

[Add file](#)

[Code](#)

**MNazarczuk0 and MaciejNazarczuk** Dodałem Siebie, MNazarczuk (#1)

9343e3e · 1 minute ago 3 Commits

README.md

Dodałem Siebie, MNazarczuk (#1)

1 minute ago

README

### Lista Uczestników

- Test - \*Napiszcie coś od siebie \* 😊
- Maciek - *Było super*

## Zadanie 6:

Podzielcie się w pary, a następnie niech 1 osoba utworzy repozytorium, 2ga osoba niech je sklonuje, doda swoje zmiany, a następnie wyśle pull request do zaakceptowania przez pierwszą osobę z pary. Śledźcie u kolegi/koleżanki jak przebiega ten proces z drugiej strony.