

Git

Prowadzący:

Jakub Szyguła: jakub.szygula@polsl.pl

Dariusz Marek: dariusz.marek@polsl.pl

Pokój: 906

9 października 2023

Spis treści

1	Wprowadzenie	2
1.1	Cel laboratorium	2
1.2	Przechowywanie danych	2
1.3	Zasada działania	2
1.4	Gałęzie	3
1.5	Git workflow	3
1.6	Instalacja repozytorium Git	4
1.7	Usuwanie konta Github	7
1.8	Usuwanie konta Github z Windows 10	7
2	Zadania do wykonania	8
2.1	Część 1	8
2.2	Część 2	11
2.3	Część 3	16
3	Sprawozdanie	16

1 Wprowadzenie

Repozytorium Git to rozproszony system kontroli wersji. Umożliwia wsparcie dla rozgałęzionego procesu tworzenia oprogramowania. Dostarcza kilka algorytmów łączenia zmian z dwóch gałęzi, a także daje możliwość dodawania własnych algorytmów.

1.1 Cel laboratorium

Celem laboratorium jest zaprezentowanie podstaw obsługi repozytorium Git.

1.2 Przechowywanie danych

Git umożliwia rozproszone przechowywanie danych. Każdy użytkownik pracujący na danym repozytorium posiada własną kopię repozytorium, do której może zapisywać zmiany bez połączenia z siecią. Następnie zmiany mogą być wymieniane między lokalnymi i zdalnymi repozytoriami.

Repozytorium Git nie wymaga aplikacji serwerowej choć w rzeczywistości zdalne repozytoria są bardzo często wykorzystywane. Istnieje wiele pakietów rozszerzających bazowe oprogramowanie, o kontrolę dostępu, możliwość zarządzania wieloma repozytoriami, czy np. interfejs WWW. Przykładami mogą być: GitHub, Bitbucket, GitLab i Gitorious.

1.3 Zasada działania

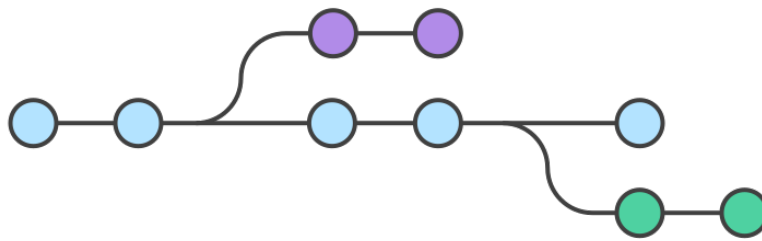
Podczas pracy z repozytorium najczęściej wykonuje się następujące kroki:

1. Tworzenie repozytorium (*git init*) - obecnie najczęściej wykorzystuje się zdalne repozytoria do tworzenia i przechowywania danych, z których pobiera się oraz wysyła zmiany w przechowywanych danych.
2. Klonowanie (*git clone*) - pobieranie zdalnego repozytorium do lokalnego repozytorium.
3. Zapisywanie stanu danych (*git commit*) - bardzo ważną częścią pracy z repozytorium jest zapisywanie stanu tworzonego oprogramowania. Umożliwia to odtworzenie stanu w razie zniszczenia aktualnej wersji danych, ale daje także możliwość przesłania zapisanego stanu do zdalnego repozytorium.
4. Wysyłanie zapisanego stanu z lokalnego do zdalnego repozytorium (*git push*) - przesyłanie stanu lokalnego repozytorium do zdalnego umożliwia rozsyłanie aktualnej wersji oprogramowania np. do innych współpracowników.
5. Pobieranie aktualnego stanu zdalnego repozytorium do lokalnego repozytorium (*git pull*).

6. Dodawanie śledzenia zmian danych plików przez oprogramowanie git (*git add*). Mechanizm śledzenia zmian w plikach to jedna z przydatniejszych funkcjonalności repozytorium. Umożliwia on sprawdzanie za pomocą odpowiednich komend (*git status*) stanu aktualnego repozytorium i zmian w plikach.
7. Sprawdzanie zmian lokalnego repozytorium względem zdalnego (*git fetch*).

1.4 Gałęzie

Podczas tworzenia oprogramowania najczęściej poza wykorzystaniem samego repozytorium do przechowywania danych (kodu źródłowego) wykorzystuje się mechanizm gałęzi (*branch*). Gałęzie to ścieżki historii, na których zapisywane są stany oprogramowania. Mechanizm ten umożliwia również tworzenie nowych gałęzi rozpoczynających się w danym zapisanym stanie repozytorium. Tworzenie kolejnych gałęzi wykorzystywane jest najczęściej do dodawania kolejnych funkcji oprogramowania bez wprowadzania bałaganu w aktualnej działającej wersji oprogramowania.



Rysunek 1: Źródło: <https://www.atlassian.com/git/tutorials/using-branches>

1.5 Git workflow

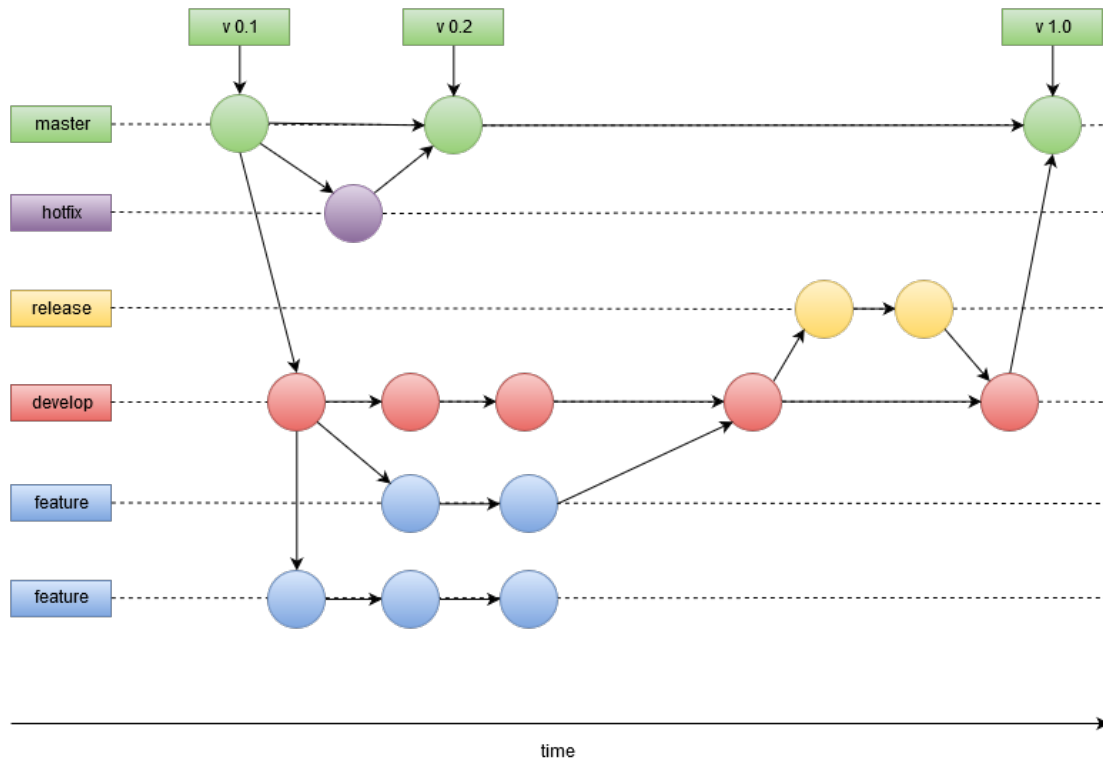
Git workflow polega na wprowadzeniu określonej struktury gałęzi w repozytorium. Wykorzystywana struktura może być zależna od potrzeb, ale w większości przypadków jest podobna. Podział gałęzi repozytorium umożliwia odseparowanie pracy jednego użytkownika od innych użytkowników.

Najczęściej wykorzystywaną strukturą gałęzi jest podział na:

1. Master – gałąź, na której znajduje się ostatnia **działająca wersja aplikacji**. Jest to również domyślna gałąź, na którą trafia się klonując repozytorium. W firmach często spotyka się specjalne automaty (testery), które same budują, testują oraz dodają aktualną wersję na tę gałąź i nie robi tego żaden człowiek.
2. Release - to jest gałąź przygotowawcza pod wydanie wersji aplikacji. Najczęściej gałąź ta wykorzystywana jest do testowania tworzonego oprogramowania.
3. Develop - główna gałąź na której składana jest praca użytkowników. W rzeczywistości rzadko wykonuje się zapisy (commit'y) bezpośrednio na tę gałąź, gdyż głównie prace wykonywane są na gałęziach typu "Feature".

4. Feature - to gałąź, na której wykonuje się wszelkie prace związane z implementacją oprogramowania. Najczęściej każdy użytkownik lub pracownik dostając zadanie tworzy nową gałąź bazującą na aktualnym stanie gałęzi "develop", następnie wykonuje na nowo utworzonej gałęzi zadanie, a po jego zakończeniu łączy swój kod z aktualnym kodem dostępnym na gałęzi "develop".
5. Hotfix - to poprawka, która musi się znaleźć jak najszybciej w wersji programu.

Graficzny przykład struktury gałęzi został zaprezentowany na rysunku 2



Rysunek 2: Graficzny przykład struktury gałęzi.

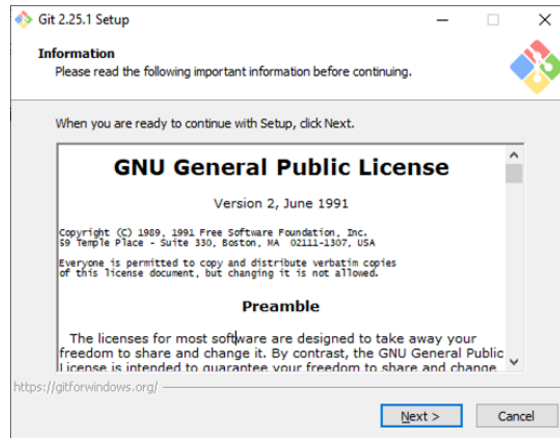
Źródło: <https://codecouple.pl/2016/02/11/gitflow-workflow-model-pracy-z-gitem/>

1.6 Instalacja repozytorium Git

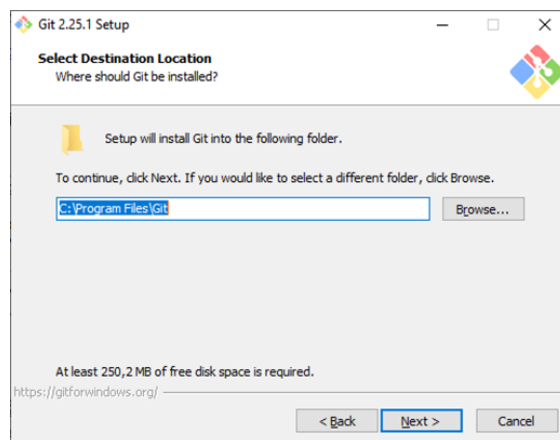
Oprogramowanie Git dostępne jest pod adresem <https://git-scm.com/downloads>.

Dla ułatwienia pracy z zarządzaniem repozytorium można użyć dowolnego edytora tekstowego, np. Visual Studio Code, który wspiera niektóre funkcjonalności repozytorium git (<https://code.visualstudio.com/>)

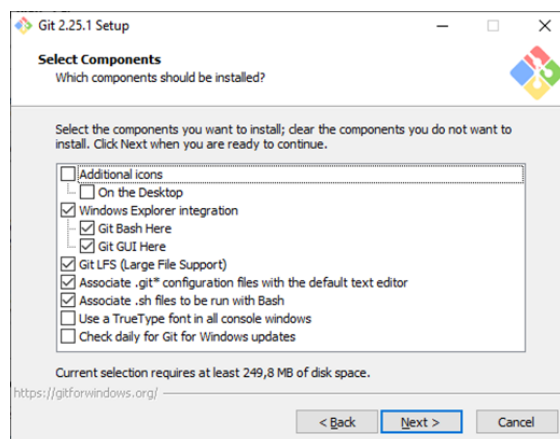
1. Rozpoczęcie instalacji



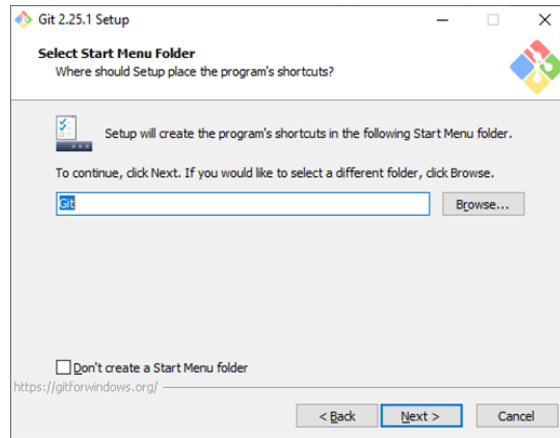
2. Wybór folderu instalacji



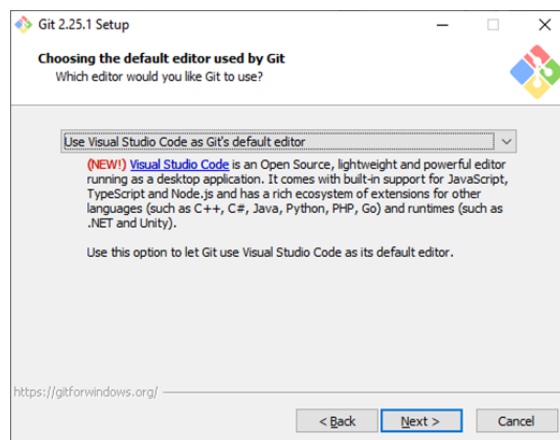
3. Wybór instalowanych komponentów



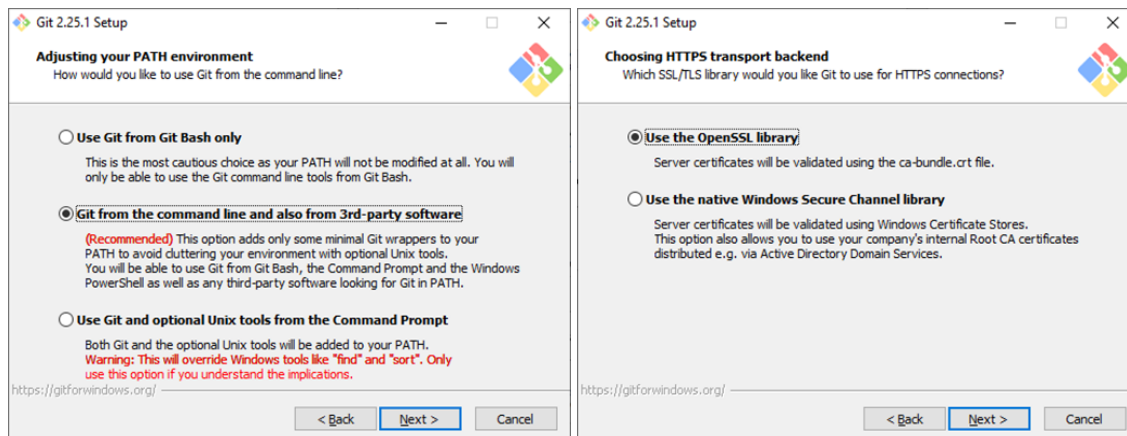
4. Folder w menu start

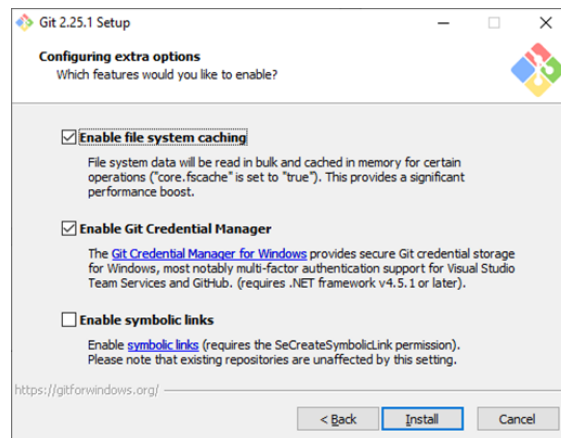
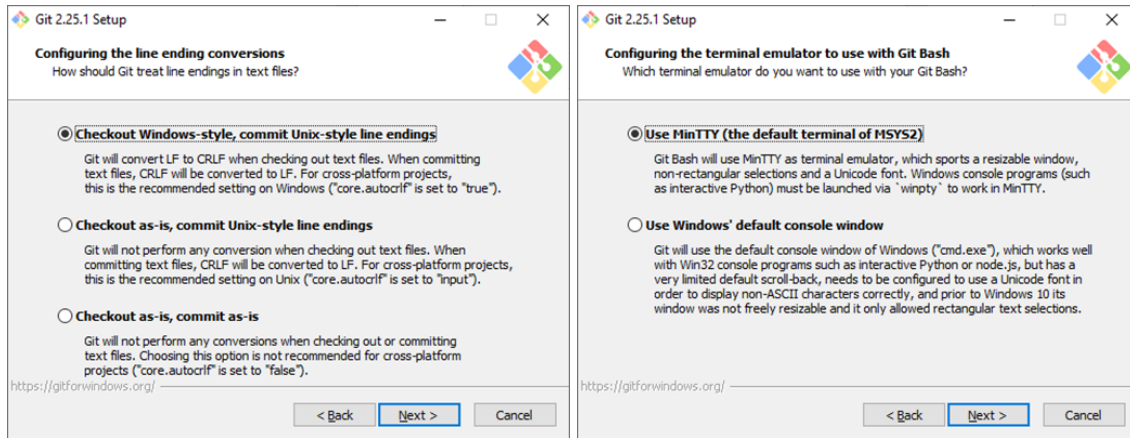


5. Wybór podstawowego edytora tekstu



6. Kolejne kroki





1.7 Usuwanie konta Github

W razie potrzeby usunięcia konta Github należy usunąć dane z konfiguracji. Dostęp do konfiguracji możliwy jest za pomocą komendy: **git config --global --edit**.

1.8 Usuwanie konta Github z Windows 10

W razie potrzeby usunięcia konta z systemu Windows 10 należy użyć **Menadżera Poświadczeń Systemu Windows** (Panel sterowania -> Konta użytkowników -> Menadżer poświadczeń -> Poświadczenia systemu Windows).

2 Zadania do wykonania

Do wykonania laboratorium będzie potrzebne konto w serwisie **Github** (github.com).

2.1 Część 1


1. Stwórz puste zdalne repozytorium w serwisie github.com.

- Zaloguj się w serwisie github.com
- Przejdź do strony z repozytoriami
- Naciśnij przycisk *New*
- Wpisz nazwę repozytorium
- Nie zaznaczaj** *Initialize this repository with a README*

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

 **dariuszmarek** ▾

Repository name *


terst-repo

✓


/

Great repository names are short and memorable. Need inspiration? How about **expert-disco**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

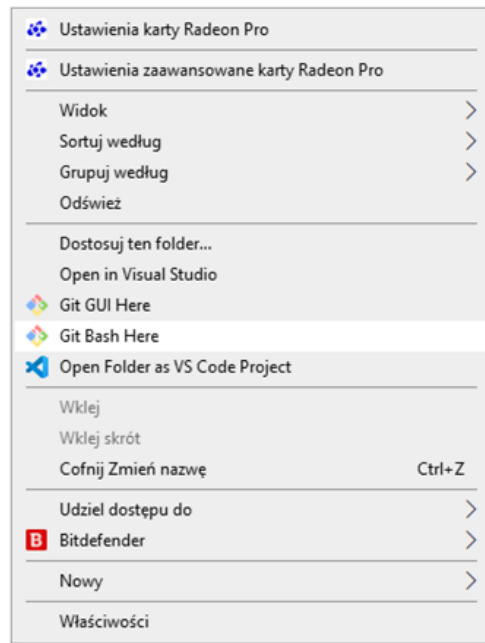
Add .gitignore: **None** ▾

Add a license: **None** ▾ 

Create repository

2. Stwórz lokalne repozytorium

- (a) Stwórz folder, w którym będzie znajdować się lokalne repozytorium.
- (b) Otwórz konsolę **git bash** w stworzonym folderze.



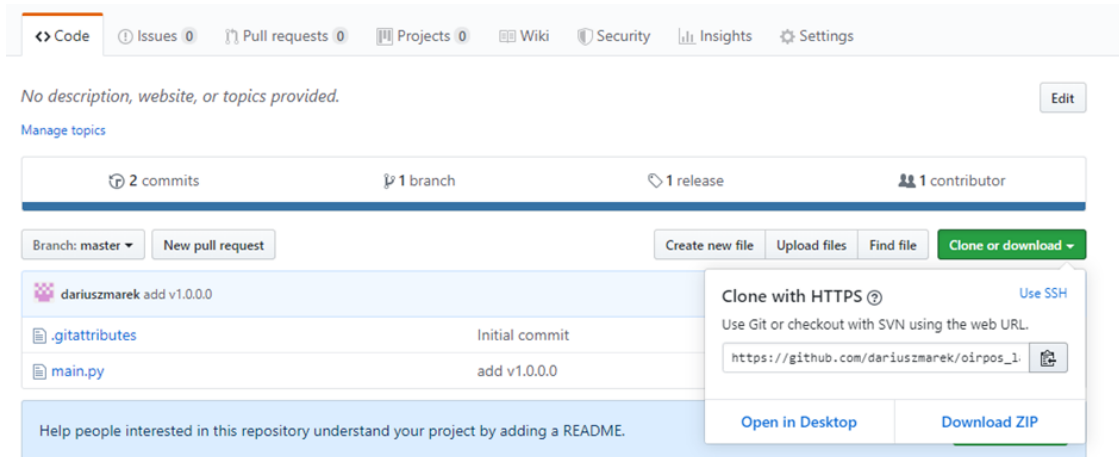
- (c) Stwórz repozytorium za pomocą komendy **git init**.

3. Pierwszy commit

- a. Dodaj do folderu plik tekstowy z dowolnym tekstem wewnątrz.
- b. Za pomocą komendy **git status** sprawdź aktualny status lokalnego repozytorium.
- c. Dodaj śledzenie dodanego pliku za pomocą komendy **git add**.
- d. Sprawdź aktualny status lokalnego repozytorium.
- e. Zapisz stan lokalnego repozytorium za pomocą komendy **git commit -m „wiadomosc”**
- f. Przeanalizuj błąd i użyj komend w nim zawartych do dodania konta github.
- g. Sprawdź status repozytorium.

4. Wysłanie lokalnego repozytorium na serwer

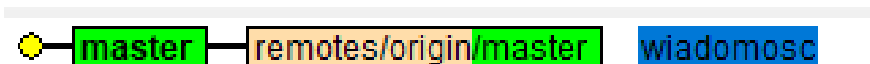
- a. Sprawdź i skopiuj adres stworzonego zdalnego repozytorium na github.com



- b. Dodaj odwołanie do zdalnego repozytorium w lokalnym repozytorium za pomocą komendy **git remote add origin <URL>**. W git'cie przyjęło się, że odwołanie do głównego repozytorium (jedno repozytorium lokalne może odwoływać się do wielu zdalnych) nazywa się **origin**.
- c. Spróbuj wykonać aktualizację zdalnego repozytorium za pomocą komendy **git push** i sprawdź błąd.
- d. Zaktualizuj zdalne repozytorium wraz z ustawieniem śledzenia zdalnego repozytorium przez lokalne za pomocą komendy wyświetlonej w tekście błędu. Można również użyć przełącznika **-u** zamiast **--set-upstream**.

5. Okno kontroli wersji **gitk**.

- a. Za pomocą komendy **gitk** uruchom aplikację używaną do kontroli wersji / historii zmian repozytorium.
- b. Zapoznaj się z wyświetlanymi informacjami. Zauważ, że żółta kropka oznacza zapis stanu repozytorium. Dodatkowo obok kropki są informacje o nazwie lokalnej gałęzi oraz zdalnej (jeżeli istnieje).



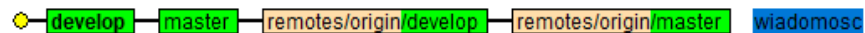
2.2 Część 2

1. Tworzenie gałęzi *develop*.

- Za pomocą komendy **git checkout master** przejdź na gałąź **master** (jeżeli zaczynasz pracę z repozytorium prawdopodobnie na niej właśnie się znajdujesz)
- Następnie stwórz gałąź **develop** za pomocą komendy:
git branch develop
- Uruchom komendę **git status** i sprawdź gdzie jesteś. Zauważ, że konsola podpowiada, który branch jest aktywny.

```
darju@DESKTOP-DU9VUG6 MINGW64 ~/Documents/GitHub/gitlab (master)
```

- Przejdź na stworzoną gałąź za pomocą komendy:
git checkout develop
- Uruchom **gitk** zauważ, że stworzyłeś gałąź, ale tylko lokalnie.
- Za pomocą komendy **git push -u origin develop** zaktualizuj zdalne repozytorium i ustaw automatyczne śledzenie lokalnego repozytorium <> i zdalnego repozytorium. Gałęzie w repozytoriach (lokalnym i zdalnym) nie muszą nazywać się tak samo – natomiast podczas tego laboratorium dla ułatwienia zrozumienia zagadnień nie będzie to wykorzystywane.
- Uruchom **gitk** i sprawdź rezultaty.

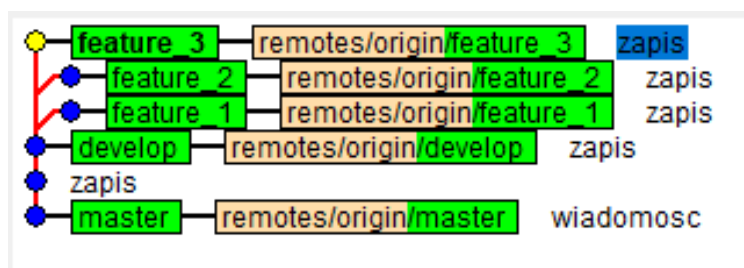


A horizontal sequence of colored boxes representing Git commits. From left to right: a yellow circle, a green box labeled 'develop', a green box labeled 'master', an orange box labeled 'remotes/origin/develop', an orange box labeled 'remotes/origin/master', and a blue box labeled 'wiadomosc'.

2. Tworzenie gałęzi (branch'ów) *feature*.

- Za pomocą komendy **git checkout develop** przejdź na gałąź **develop**
 - Stwórz plik *main.txt* w folderze z repozytorium.
 - Dodaj tekst **developcommit** do stworzonego pliku.
 - Dodaj stworzony plik do śledzenia za pomocą komendy **git add**.
 - Dodaj commit'a do lokalnego repozytorium za pomocą komendy **git commit -m „wiadomosc”**
 - Wyślij zmiany do zdalnego repozytorium używając komendy **git push**
- Za pomocą komendy **git checkout develop** przejdź na gałąź **develop**
 - Stwórz gałąź **feature_1** rozpoczynając się w gałęzi **develop**.
 - Przejdź na stworzoną gałąź **feature_1**.
 - Dodaj do pliku *main.txt* w nowej linii tekst **feature1commit**.
 - Sprawdź stan repozytorium za pomocą **git status**.
 - Zapisz stan repozytorium (*git add / git commit*).

- g. Wyślij zmiany do zdalnego repozytorium
- c)
- a. Za pomocą komendy **git checkout develop** przejdź na gałąź **develop**
 - b. Stwórz gałąź **feature_2** rozpoczynając się w gałęzi **develop**.
 - c. Przejdź na stworzoną gałąź **feature_2**.
 - d. Dodaj do pliku *main.txt* w nowej linii tekst **feature2commit**.
 - e. Sprawdź stan repozytorium za pomocą **git status**.
 - f. Zapisz stan repozytorium (*git add / git commit*).
 - g. Wyślij zmiany do zdalnego repozytorium
- d)
- a. Za pomocą komendy **git checkout develop** przejdź na gałąź **develop**
 - b. Stwórz gałąź **feature_3** rozpoczynając się w gałęzi **develop**.
 - c. Przejdź na stworzoną gałąź **feature_3**.
 - d. Dodaj do pliku *main.txt* w nowej linii tekst **feature3commit**.
 - e. Sprawdź stan repozytorium za pomocą **git status**.
 - f. Zapisz stan repozytorium (*git add / git commit*).
 - g. Wyślij zmiany do zdalnego repozytorium
- e) Przełącz się pomiędzy 4 stworzonymi gałęziami i zaobserwuj zmiany w pliku *main.txt*.
- f) Sprawdź stan repozytorium za pomocą aplikacji **gitk** i porównaj wyświetlane dane, kiedy **gitk** uruchamiane jest z przełącznikiem **--all** (**gitk --all**)



3. Łączenie (Merge'owanie) gałęzi (branch'ów) feature.

- a. Przejdź na gałąź **develop**.
- b. Wykonaj merge branch'a **feature_1**. Merge wykonuje się z jakiegoś do aktualnego aktywnego branch'a. Użyj do tego komendy **git merge feature_1** (za pomocą „TAB” wyświetlane są podpowiedzi)
- c. Przeczytaj wyświetlony komunikat.
Fast-forward – oznacza, że dołączana gałąź wychodziła z tego samego miejsca, w którym się aktualnie znajdowaliśmy i można było wykonać przeniesienie obecnej gałęzi na dołączaną.

- d. Wykonaj merge drugiego feature'a
git merge feature_2
- e. Przeczytaj wyświetlony komunikat.
Wystąpił konflikt, który nie mógł być automatycznie rozwiązany.
- f. Uruchom narzędzie do łączenia plików **git mergetool**. Jeżeli uruchomił się VIM to spokojnie! Wyjdź z niego, a następnie przejdź do punktu 7, dodaj vscode i wróć tu :).
- g. Zapoznaj się z otrzymanym wynikiem.

```
developcommit
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
feature1commit
=====
feature2commit
>>>>>> feature_2 (Incoming Change)
```

- h. Używając dostępnych „przycisków” (Accept...) spraw, aby wynikowo otrzymać obie zmiany.

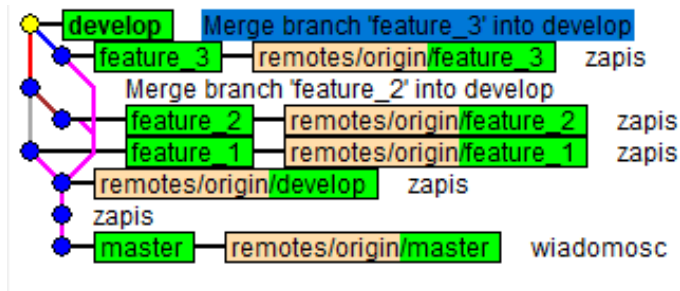
```
developcommit
feature1commit
feature2commit
```

- i. Zapisz zmiany i wyłącz vscode.
- j. Zauważ, że nadal aktywna gałąź jest podczas „łączenia”

```
dariu@DESKTOP-DU9VUG6 MINGW64 ~/Documents/GitHub/gitlab (develop|MERGING)
```

- k. Użyj komendy **git merge --continue** aby przejść do następnego etapu łączenia. Powinien wyświetlić się ponownie vscode umożliwiający wpisanie wiadomości commit'a. Wpisz wiadomość lub zostaw bez zmian, zapisz i wyjdź z vscode.
- l. Przeanalizuj otrzymane komunikaty.
- m. Uruchom **gitk**
- n. Branch został połączony i commit wykonany, ale nadal zmiany nie zostały wysłane do repozytorium zdalnego.
- o. Wykonaj polecenia:
git fetch – sprawdzenie czy na zdalnym repozytorium zaszły jakieś zmiany, ale bez ich pobierania
git pull - pobranie zmian dla aktualnie aktywnego branch'a
git push – wysłanie zmian dla aktualnie aktywnego branch'a do zdalnego repozytorium
git status – wyświetlenie statusu lokalnego repozytorium

- p. Uruchom **gitk**
- q. Wykonaj merge 2 pozostałych funkcjonalności (feature'ów). **Synchronizuj dane z repozytorium zdalnym**
- r. Po połączeniu 3 funkcjonalności uruchom **gitk** i przeanalizuj historię.



4. Gitignore.

- a. przejdź na gałąź **develop**
- b. Stwórz gałąź **release** i prześlij ją na zdalne repozytorium.
- c. Dodaj dowolny plik z rozszerzeniem ***.cpp** do repozytorium.
- d. Uruchom **git status** i sprawdź aktualny stan repozytorium.
- e. Dodaj commit'a zawierającego dodany plik i zsynchronizuj repozytoria.
- f. Dodaj plik **.gitignore** do repozytorium, a w treści wpisz ***.cpp**

```

dariu@DESKTOP-DU9VUG6 MINGW64 ~/Documents/GitHub/gitlab (release)
$ touch .gitignore

dariu@DESKTOP-DU9VUG6 MINGW64 ~/Documents/GitHub/gitlab (release)
$ echo "*.cpp" > .gitignore

```

- g. Dodaj drugi plik z rozszerzeniem ***.cpp**
- h. Uruchom **git status** i zauważ, że po dodaniu pliku **gitignore** z rozszerzeniem **cpp** wewnątrz, git nie podpowiada o dodaniu pliku z rozszerzeniem **cpp** do repozytorium. **Github** udostępnia wiele plików **gitignore** do różnych języków programowania.
<https://github.com/github/gitignore>
- i. Umożliwiają one niedodawanie zbędnych plików np. wynikowych z budowania aplikacji do repozytorium.
- j. Dodaj commit'a za pomocą **git add *** i przeczytaj komunikat. Podczas dodawania nie został również dołączony plik **gitignore**, gdyż ten trzeba dodać ręcznie.
- k. Dodaj plik **.gitignore** do commit'a, utwórz commit'a i zsynchronizuj repozytoria.

5. Tag'owanie commit'ów.

- Przejdź na gałąź **release**
- Za pomocą komendy **git tag -a v1.0.0.0 -m 'release v1.0.0.0'** dodaj tag do commit'a.
- Uruchom **gitk** i naciśnij na tag, sprawdź wyświetlane informacje. Tag'owanie polega na oznaczaniu danego commit'a. Mogą być 2 typy tagów lightweight oraz annotated. Stworzony tag w punkcie b jest annotated. Do taga annotated można się odwoływać jak do nazwy branch'a, a do lightweight nie ma takiej możliwości. Tagi nie są wysyłane automatycznie do zdalnego repozytorium, trzeba to robić manualnie, za pomocą **git push --tags**. Usuwanie lokalnych tagów następuje poprzez **git tag --delete <nazwa>**. Natomiast zdalnych za pomocą wysłania „niczego” w miejsce tag'a **git push origin :<nazwa>**
- Prześlij tag do zdalnego repozytorium.

6. Tworzenie aliasów.

- Uruchom **git config --global -e**
- Dodaj przykładowe aliasy i przeanalizuj ich działanie uruchamiając poszczególne komendy np. **git st**

```
[alias]
st = status
co = checkout
sync = "!f() { git pull --all; git push; }; f"
cos = "!f() { git co $1; git sync; }; f"
```

7. Dodawanie aplikacji do porównywania i łączenia plików.

- Uruchom globalne ustawienia git'a w edytorze za pomocą komendy **git config --global -e**
- Następnie dodaj do niego kod odpowiedzialny za dodanie vscode jako program do merg'owania

```
# Add this to you gitconfig
# Comment: Start of "Extra Block"
# Comment: This is to unlock VSCode as your
# git diff and git merge tool
[merge]
tool = vscode
[mergetool "vscode"]
cmd = code --wait $MERGED
[diff]
```

```
tool = vscode  
[difftool "vscode"]  
cmd = code --wait --diff $LOCAL $REMOTE  
# VSCode Difftool  
## End of extra block
```

2.3 Część 3

- a. Usuń folder zawierający stworzone repozytorium.
- b. Stwórz folder, do którego zostanie ponownie stworzone repozytorium ze zdalnego repozytorium.
- c. Użyj komendy **git clone <url>** do sklonowania repozytorium zdalnego do lokalnego.
- d. Przeanalizuj pobrane repozytorium za pomocą aplikacji **gitk**.

3 Sprawozdanie

Sprawozdanie prosimy przesłać na Platformę Zdalnej Edukacji (<https://platforma.polsl.pl/>):

1. Kody źródłowe zadań,
2. Krótkie sprawozdanie z wykonanych zadań,
3. Skład sekcji.