System kontroli wersji GIT

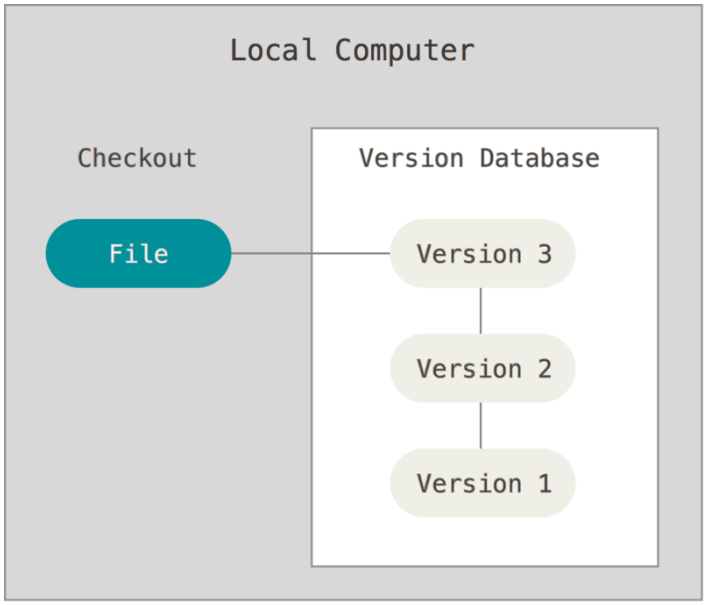
System kontroli wersji śledzi wszystkie zmiany dokonywane na pliku (lub plikach) i umożliwia przywołanie dowolnej wcześniejszej wersji. Przykłady w tej książce będą śledziły zmiany w kodzie źródłowym, niemniej w ten sam sposób można kontrolować praktycznie dowolny typ plików.

Jeśli jesteś grafikiem lub projektantem WWW i chcesz zachować każdą wersję pliku graficznego lub układu witryny WWW (co jest wysoce prawdopodobne), to używanie systemu kontroli wersji (VCS-Version Control System) jest bardzo rozsądnym rozwiązaniem. Pozwala on przywrócić plik(i) do wcześniejszej wersji, odtworzyć stan całego projektu, porównać wprowadzone zmiany, dowiedzieć się kto jako ostatnio zmodyfikował część projektu powodującą problemy, kto i kiedy wprowadził daną modyfikację. Oprócz tego używanie VCS oznacza, że nawet jeśli popełnisz błąd lub stracisz część danych, naprawa i odzyskanie ich powinno być łatwe. Co więcej, wszystko to można uzyskać całkiem niewielkim kosztem.

Lokalne systemy kontroli wersji

Dla wielu ludzi preferowaną metodą kontroli wersji jest kopiowanie plików do innego katalogu (może nawet oznaczonego datą, jeśli są sprytni). Takie podejście jest bardzo częste ponieważ jest wyjątkowo proste, niemniej jest także bardzo podatne na błędy. Zbyt łatwo zapomnieć w jakim jest się katalogu i przypadkowo zmodyfikować błędny plik lub skopiować nie te dane.

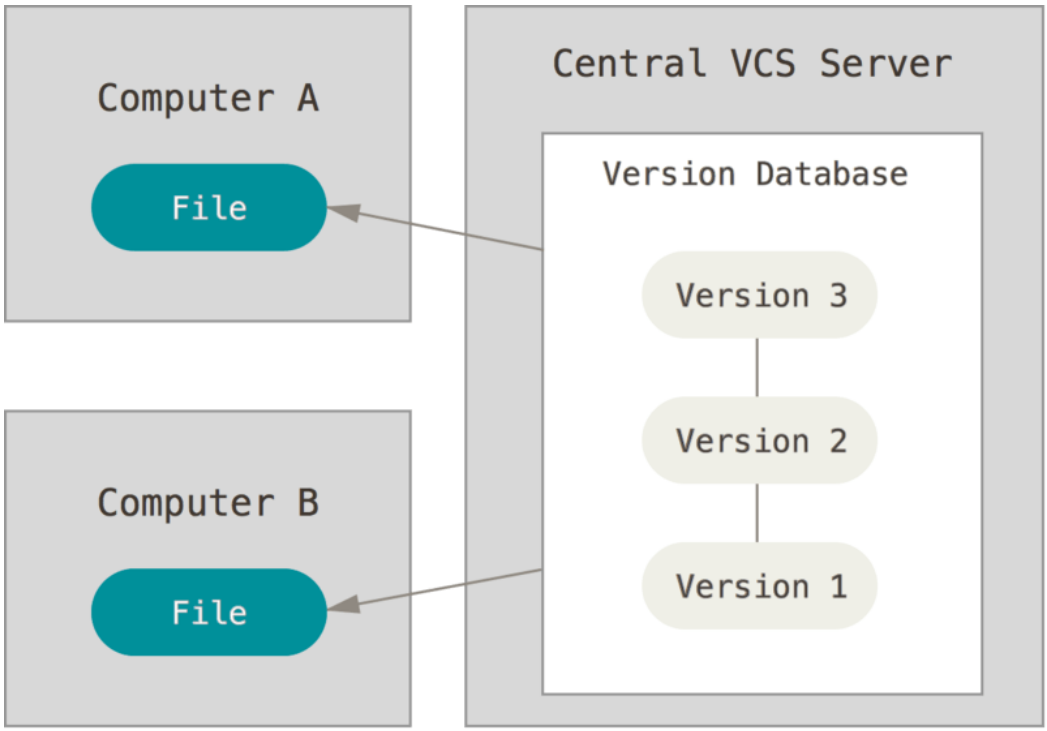
Aby poradzić sobie z takimi problemami, programiści już dość dawno temu stworzyli lokalne systemy kontroli wersji, które składały się z prostej bazy danych w której przechowywane były wszystkie zmiany dokonane na śledzonych plikach (por. Rysunek 1-1).



Lokalna kontrola wersji.

Scentralizowane systemy kontroli wersji

Kolejnym poważnym problemem z którym można się spotkać jest potrzeba współpracy w rozwoju projektu z odrębnych systemów. Aby poradzić sobie z tym problemem stworzono scentralizowane systemy kontroli wersji (CVCS - Centralized Version Control System). Systemy takie jak CVS, Subversion czy Perforce składają się z jednego serwera, który zawiera wszystkie pliki poddane kontroli wersji, oraz klientów którzy mogą się z nim łączyć i uzyskać dostęp do najnowszych wersji plików. Przez wiele lat był to standardowy model kontroli wersji



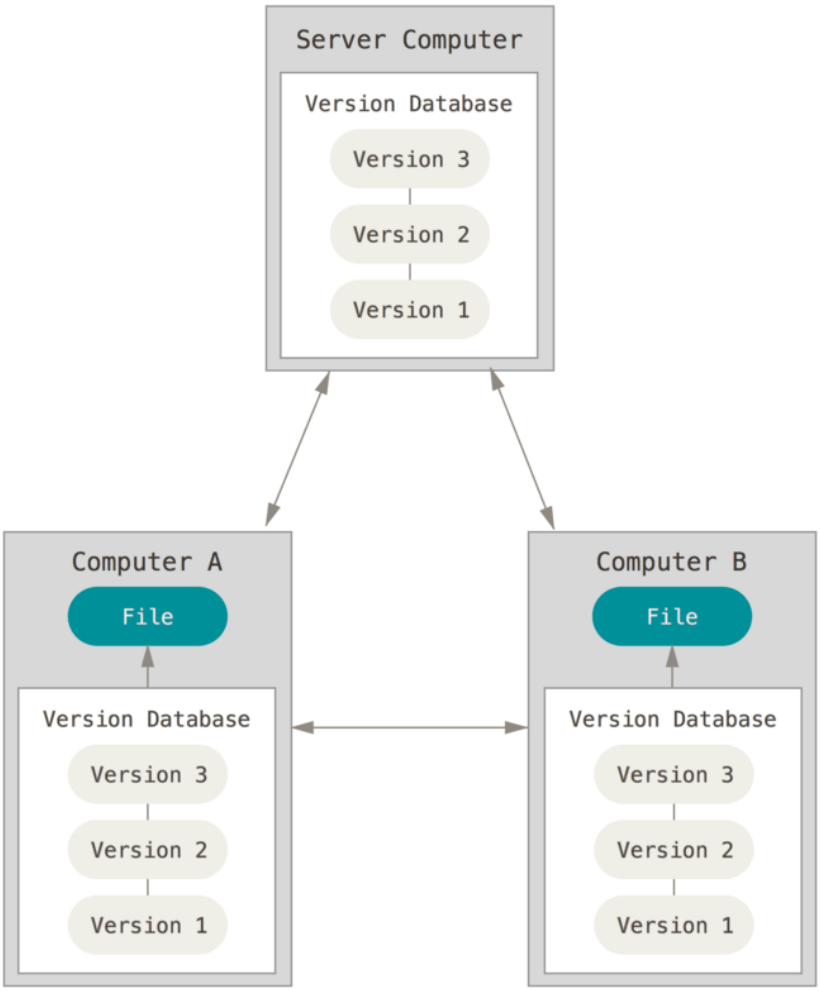
Zcentralizowana kontrola wersji.

Taki schemat posiada wiele zalet, szczególnie w porównaniu z VCS. Dla przykładu każdy może się zorientować co robią inni uczestnicy projektu. Administratorzy mają dokładną kontrolę nad uprawnieniami poszczególnych użytkowników. Co więcej systemy CVCS są także dużo łatwiejsze w zarządzaniu niż lokalne bazy danych u każdego z klientów.

Niemniej systemy te mają także poważne wady. Najbardziej oczywistą jest problem awarii centralnego serwera. Jeśli serwer przestanie działać na przykład na godzinę, to przez tę godzinę nikt nie będzie miał możliwości współpracy nad projektem, ani nawet zapisania zmian nad którymi pracował. Jeśli dysk twardy na którym przechowywana jest centralna baza danych zostanie uszkodzony a nie tworzono żadnych kopii zapasowych, to można stracić absolutnie wszystko - całą historię projektu, może oprócz pojedynczych jego części zapisanych na osobistych komputerach niektórych użytkowników. Lokalne VCS mają ten sam problem - zawsze gdy cała historia projektu jest przechowywana tylko w jednym miejscu, istnieje ryzyko utraty większości danych.

Rozproszone systemy kontroli wersji

W ten sposób dochodzimy do rozproszonych systemów kontroli wersji (DVCS - Distributed Version Control System). W systemach DVCS (takich jak Git, Mercurial, Bazaar lub Darcs) klienci nie dostają dostępu jedynie do najnowszych wersji plików ale w pełni kopiują całe repozytorium. Gdy jeden z serwerów, używanych przez te systemy do współpracy, ulegnie awarii, repozytorium każdego klienta może zostać po prostu skopiowane na ten serwer w celu przywrócenia go do pracy



Rozproszona kontrola wersji.

Co więcej, wiele z tych systemów dość dobrze radzi sobie z kilkoma zdalnymi repozytoriami, więc możliwa jest jednoczesna współpraca z różnymi grupami ludzi nad tym samym projektem. Daje to swobodę wykorzystania różnych schematów pracy, nawet takich które nie są możliwe w scentralizowanych systemach, na przykład modeli hierarchicznych.[1]

[1] https://git-scm.com/book/pl/v2/Pierwsze-kroki-Wprowadzenie-do-kontroli-wersji

Podstawowe pojęcia

SCM — to akronim od Source Code Management, czyli dosłownie kontrola kodu (w języku polskim funkcjonuje określenie system kontroli wersji); jest to system który pozwala na archiwizowanie i śledzenie zmian w kodzie, dzięki czemu możemy cofać się w historii lub podejrzeć, kto był autorem konkretnej zmiany

Repozytorium — 'kontener' na określony zbiór kodu, najczęściej jeden projekt; repozytorium pozwala grupować kod i zmiany, dzięki czemu możemy przeglądać wszystkie zmiany wykonane w ramach jednego repozytorium, przyznawać uprawnienia do repozytoriów oraz pobierać / kopiować je

Commit (lub rewizja) — jest to proces 'wysłania' na repozytorium określonych zmian w kodzie — jeśli pobierasz kod z repozytorium, następnie dokonujesz modyfikacji i wysyłasz te zmiany z powrotem do repozytorium, proces ten nosi nazwę commitowania, a same zmiany wysłane razem nazywamy commitem lub rewizją

pull / push — odpowiednio pobranie i wysłanie zmian (jednego lub wielu commitów) z/do innego repozytorium

diff — (ang. różnica) — jest to różnica pomiędzy różnymi rewizjami — dzięki temu możemy zobaczyć, które fragmenty uległy zmianie oraz w jaki sposób; pozwala to także zoptymalizować transfer danych pomiędzy repozytoriami

fork — kopia repozytorium; szczególnie popularne w przypadku projektów open-source, dzięki czemu możemy skopiować cały projekt i rozwijać go niezależnie (np. dopasowując do naszych potrzeb)

branch — odgałęzienie, wersja wewnątrz repozytorium; branche pozwalają na prace wielu osobom równocześnie, bez ciągłego wchodzenia sobie w drogę i nadpisywania zmian — każdy może pracować na swoim branchu, dopiero po zakończeniu pracy łącząc zmiany z innymi i rozwiązując problemy

merge — połączenie wielu zmian z różnych źródeł, które może skutkować niekompatybilnymi zmianami wymagającymi ręcznych modyfikacji; merge pozwala łączyć prace wykonywane w różnych obszarach, które mogą się zazębiać, w jedną całość w sposób kontrolowany i świadomy[2]

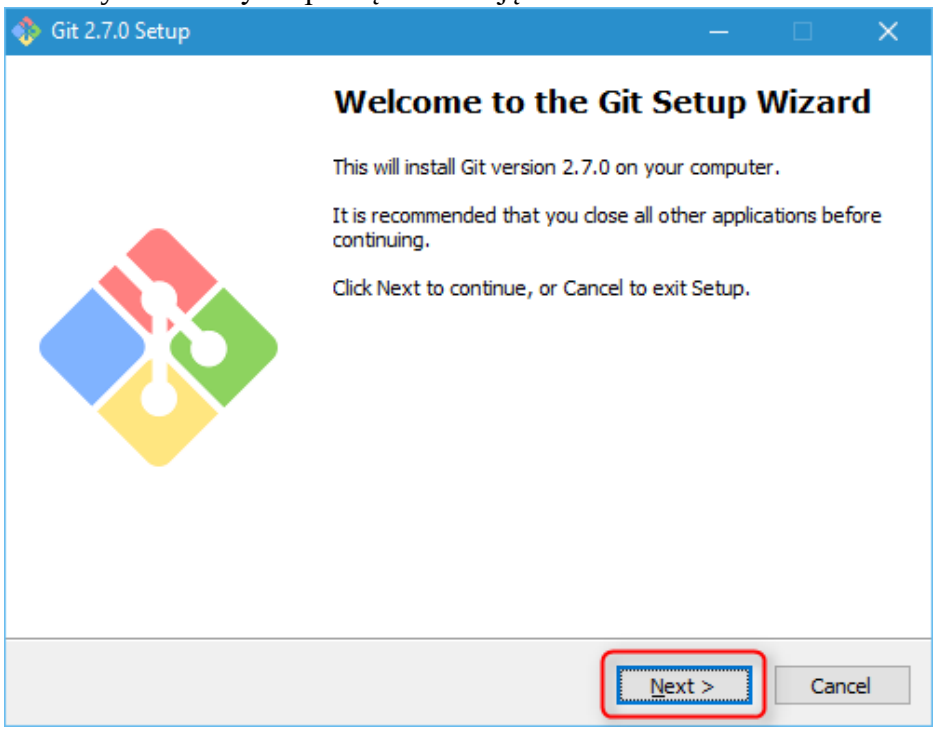
[2] https://kobietydokodu.pl/niezbednik-juniora-git-kontrola-wersji/

Instalacja Gita w systemie Windows

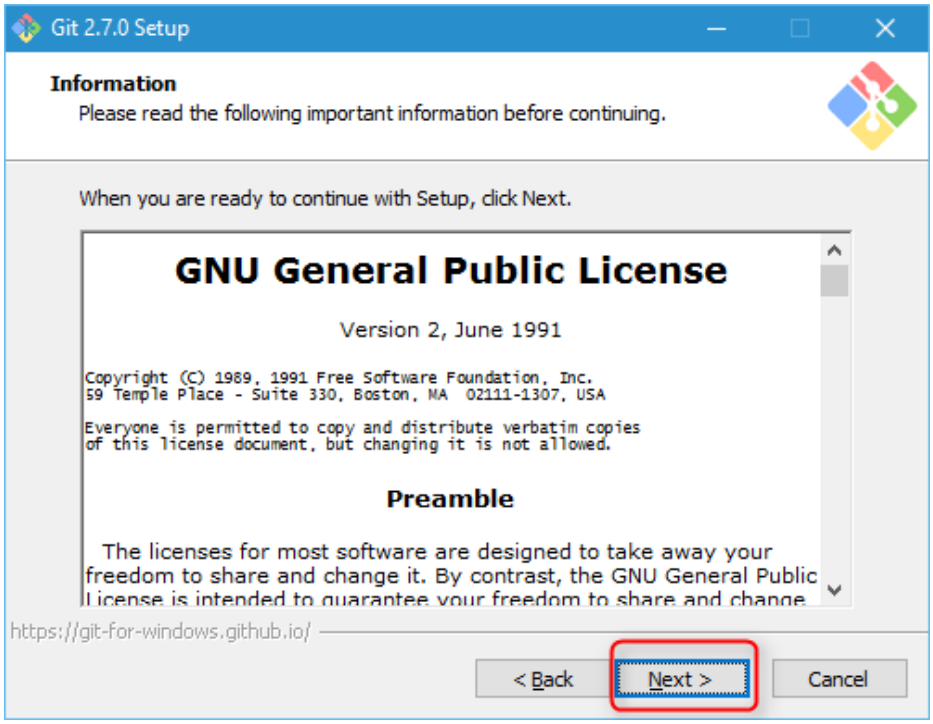
Jeżeli chcemy pracować z Gitem w systemie Windows, to konieczne jest doinstalowanie odpowiednich narzędzi. Jednym z częściej wybieranych rozwiązań jest Git for Windows, dostępny pod adresem https://git-for-windows.github.io/
Instalując to narzędzie zyskamy dostęp do konsoli tekstowej oraz do prostego interfejsu graficznego. Dzięki temu będzie można tworzyć lokalne repozytoria na naszym komputerze, jak również dokonywać synchronizacji ze zdalnymi repozytoriami na zewnętrznych serwerach.

W celu instalacji Git for Windows należy wykonać poniższe kroki:

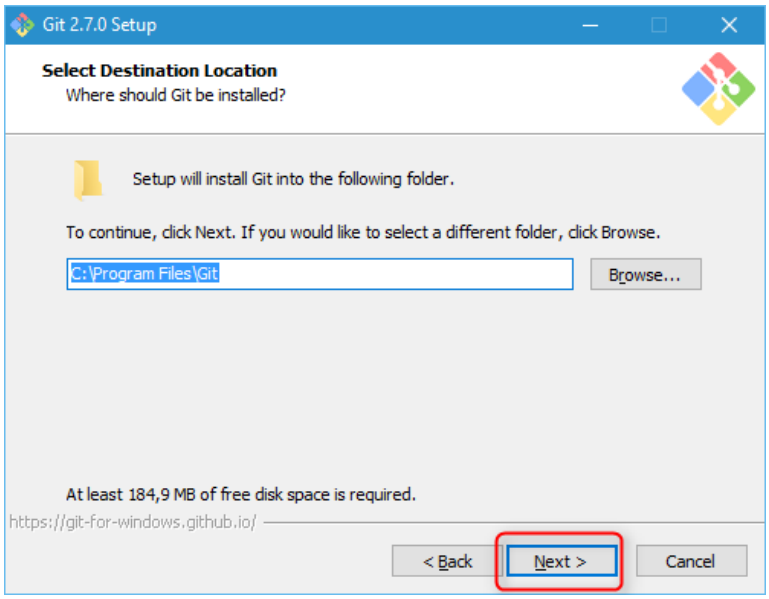
1. Pobieramy instalator ze strony https://git-for-windows.github.io/ i uruchamiamy go. Klikamy **Next** żeby rozpocząć instalację.



2. Czytamy licencję i klikamy **Next** zatwierdzając ją.

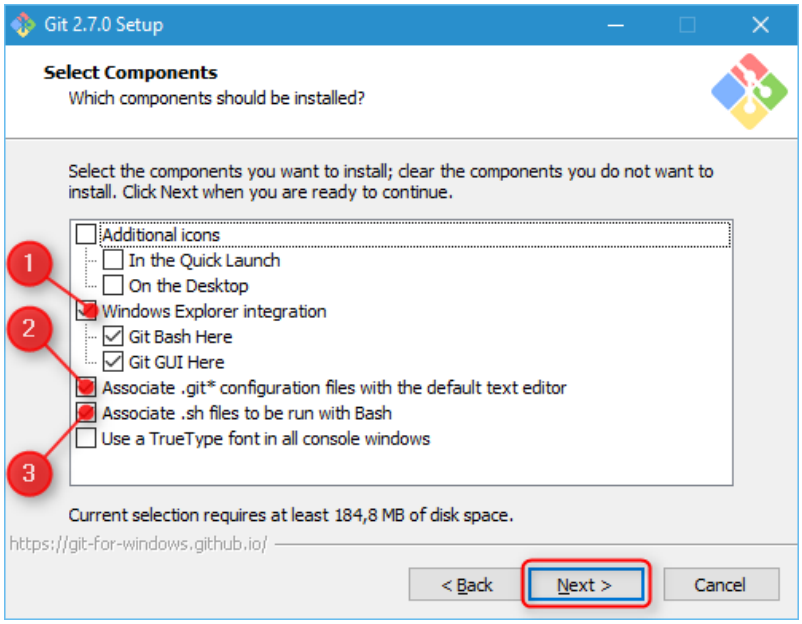


3. Podajemy ścieżkę pod którą zainstalowana ma zostać aplikacja. Zwykle możemy zostawić tą wartość bez zmian. Po uzupełnieniu klikamy **Next**.

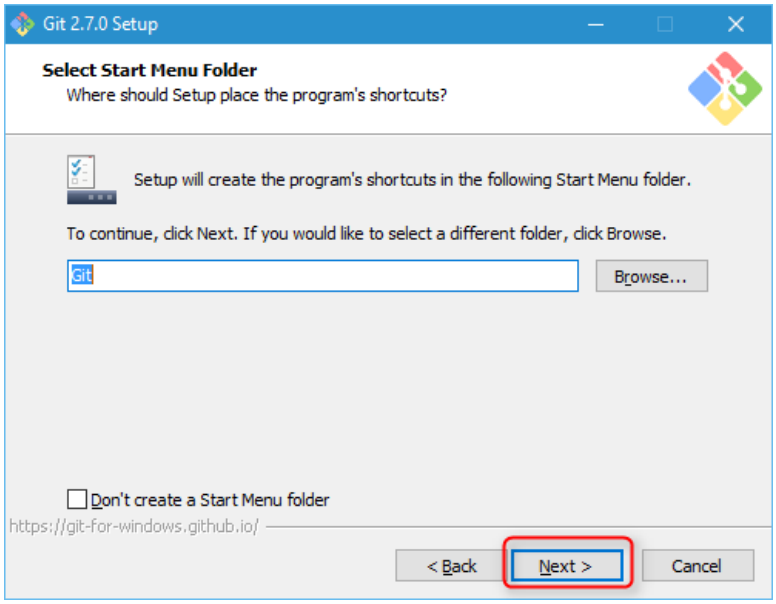


4. Zaznaczamy składniki jakie mają zostać zainstalowane. Zaznaczenie opcji 1 powoduje instalacje konsoli tekstowej i graficznego narzędzia do zarządzania repozytoriami. Opcje 2 i 3 powodują powiązanie odpowiednich plików z dostępnymi narzędziami, celem umożliwienia ich automatycznego uruchamiania po kliknięciu na nie. Po zaznaczeniu tych

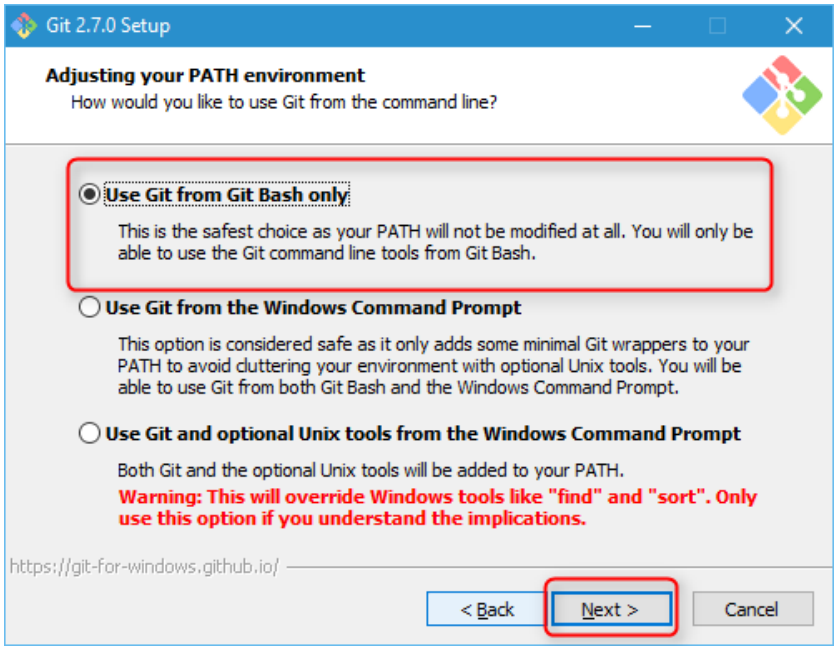
opcji klikamy **Next**.



5. Podajemy umiejscowienie w Menu Start. Zwykle można pozostawić tą wartość domyślną. Aby przejść do kolejnego kroku klikamy **Next**.



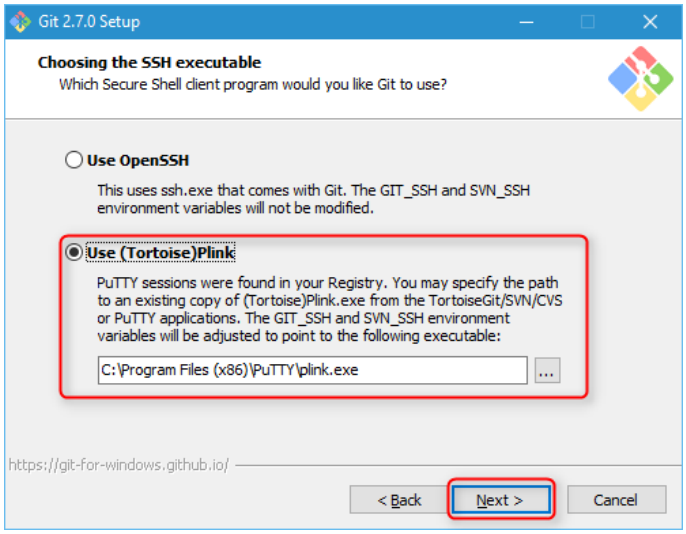
6. Decydujemy jak mocno zintegrowany z Windowsem ma być nasz git - polecamy wybrać pierwszą opcję. Dzięki niej, git będzie dostępny tylko w dodatkowo doinstalowanych narzędziach. Nie będzie on natomiast dostępny z wiersza polecenia systemu Windows. Jest to najbezpieczniejsza opcja, która nie powoduje zmian w systemie.



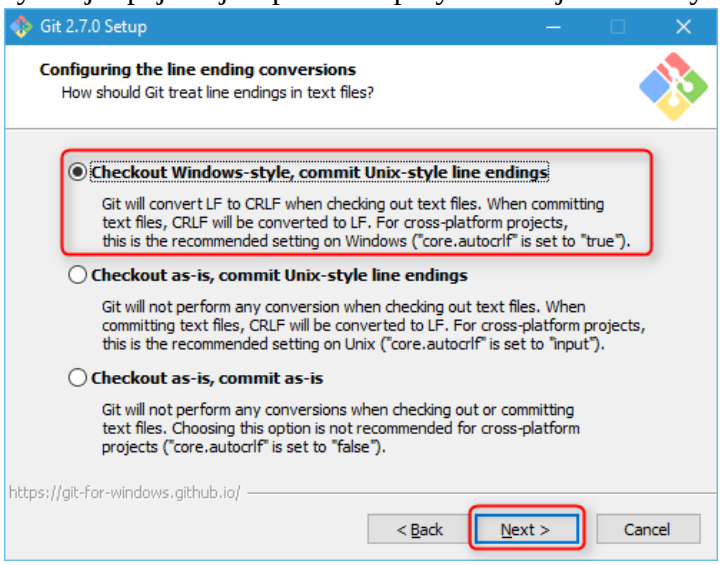
7. W tym kroku podajemy metodę przy pomocy której, git będzie nawiązywał połączenia SSH. Celem ułatwienia zarządzania kluczami SSH polecamy użycie i zainstalowanie w systemie programu PuTTY dostępnego pod adresem: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Warto pobrać ze wskazanej strony plik oznaczony jako "Installer", co spowoduje instalację pakietu narzędzi ułatwiających połączenia z użyciem protokołu SSH. Pakiet ten zawiera programy Pageant oraz Plink, które ułatwią nawiązywanie połączenie do repozytoriów z użyciem kluczy. **W przypadku serwerów hostingowych w Kylos.pl dostęp do repozytoriów możliwy jest tylko z użyciem kluczy.**

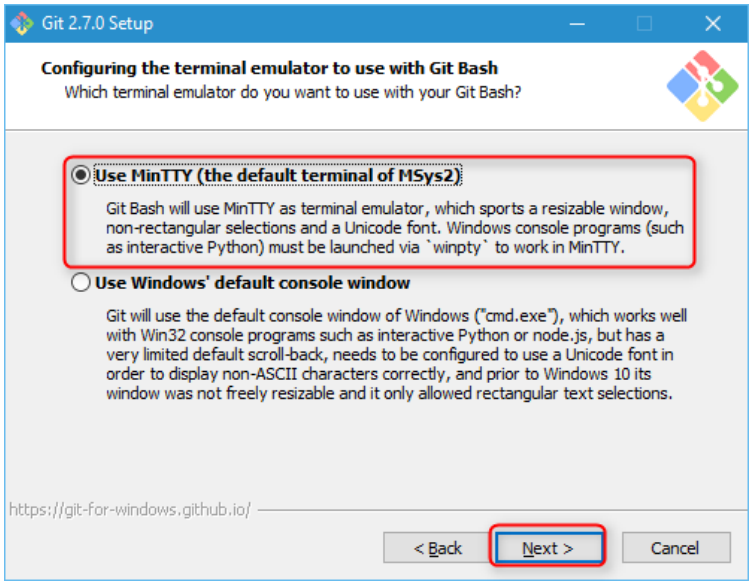
Po instalacji PuTTY, wybieramy **Use Plink** i wskazujemy ścieżkę do pliku plink.exe. Powinien się on znajdować w folderze w którym zainstalowaliśmy PuTTY. Przechodzimy do kolejnego kroku klikając **Next**.



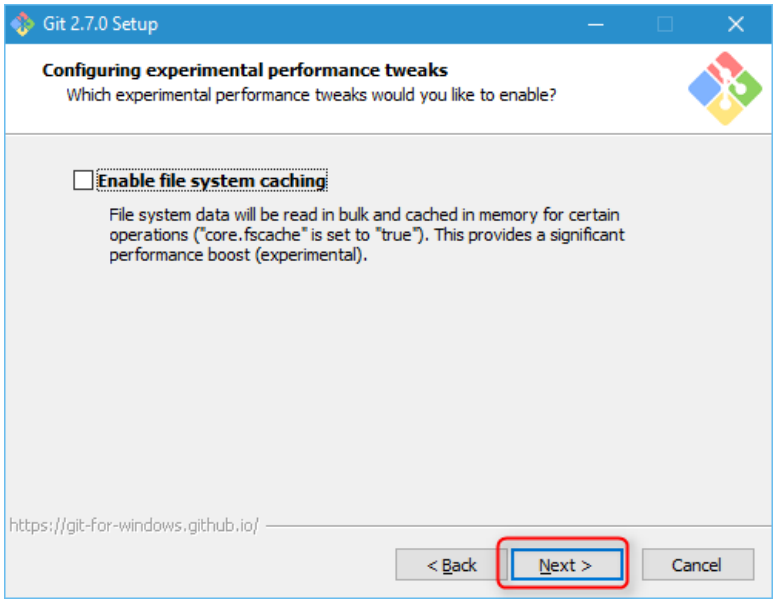
8. Proponujemy wybrać pierwszą z dostępnych opcji. Wskazują one mechanizmy konwertowania znaków końca linii w zależności od systemu operacyjnego. Git to narzędzie mocno związane z Linuxem, dlatego repozytoria na których będziemy pracować, zazwyczaj znajdują się na serwerze, gdzie środowisko jest oparte na systemie GNU/Linux. W takiej sytuacji opcja ta jest polecana przy instalacji Gita w systemie Windows.



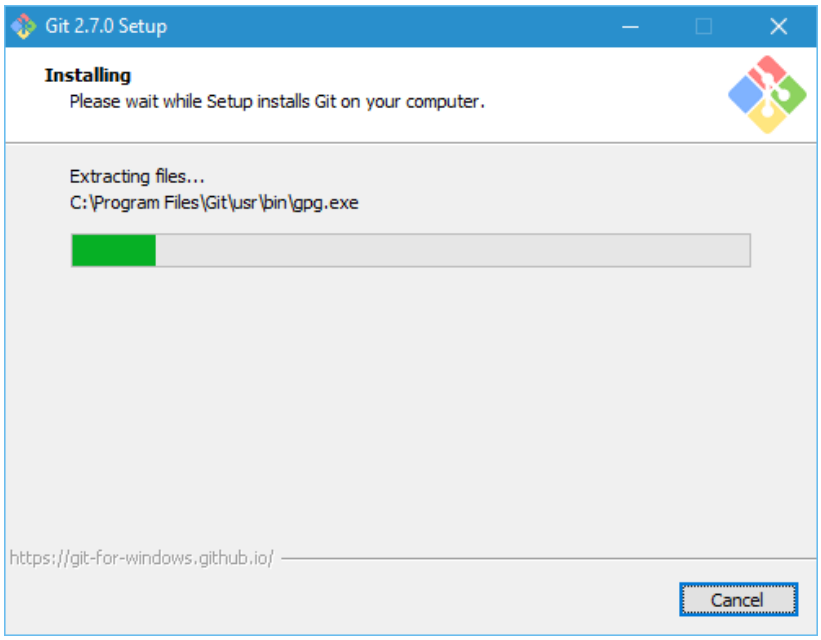
9. Pozostawiamy zaznaczoną pierwszą opcję, która dostarczy nam bardziej rozbudowaną konsolę.



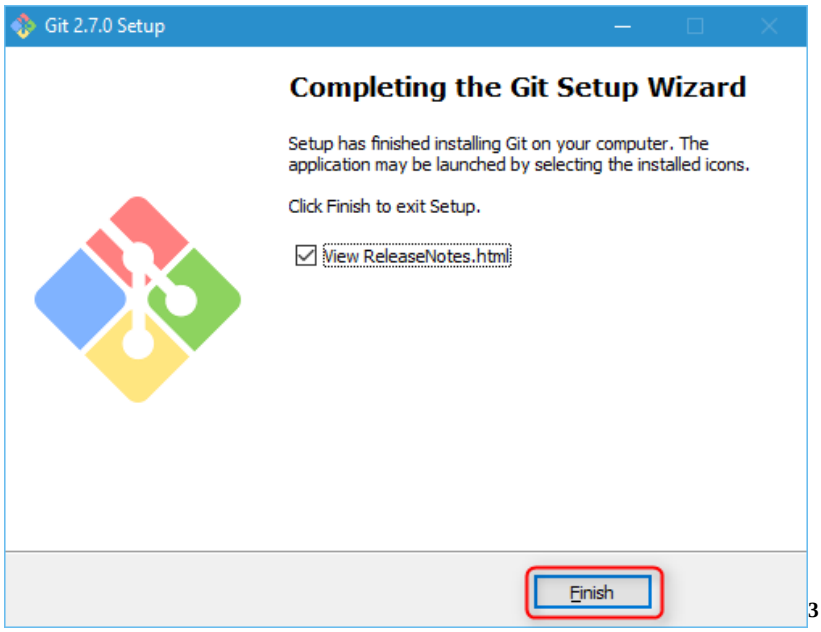
10. Kolejną opcję pozostawiamy niezaznaczoną i klikamy **Next** przechodząc do instalacji.



11. Cierpliwie czekamy aż wszystkie pliki zostaną skopiowane i oprogramowanie zostanie skonfigurowane.



12. Instalacja została zakończona. Celem zamknięcia instalatora klikamy **Finish.**



[3]

Podstawowe operacje wykonywane na repozytoriach Gita

Git to rozbudowane narzędzie, umożliwiające zawansowaną kontrolę nad wersjami dokumentów w naszym repozytorium. Do jego podstawowej obsługi wystarczy kilka prostych poleceń.

1. Konfiguracja danych użytkownika.

Ustawiamy globalne wartości dla naszej nazwy oraz dla naszego adresu e-mail. Dane te ułatwiają naszą identyfikacje w logach zmian. Celem ustawienia nazwy wydajemy polecenie:

```
git config --global user.name "Imię Nazwisko"
```

Aby ustawić nasz adres e-mail wykonujemy:

```
git config --global user.email imienazwisko@example.com
```

Użycie paramteru **--global** powoduje, że wartości te będą używane domyślnie w naszym systemie dla wszystkich repozytoriów.

[3] https://panel.kylos.pl/knowledgebase/189/Instalacja-Gita-w-systemie-Windows-.html

2. Sklonowanie repozytorium ze zdalnego serwera.

Aby sklonować istniejące w sieci repozytorium poprzez SSH wykonujemy polecenie o składni:

git clone nazwa_uzytkownika@example.com:/sciezka/do/rezpozytorium.git a na serwerach hostingowych kylos.pl poprawna będzie składnia:

```
git clone
nazwa_uzytkownika@nazwa_konta.kylos.pl:/home/nazwa_uzytko
wnika/sciezka/do/rezpozytorium.git
```

3. Dodawanie i usuwanie plików w repozytorium.

Po umieszczeniu w folderze rozboczym plików, musimy je jeszcze dodać do naszego repozytorium. Możemy je dodawać plik po pliku:

```
git add plik1
```

```
git add plik2
```

Lub dodać wszystkie niedodane pliki:

```
git add .
```

Aby usunąć plik z repozytorium wydajemy polecenie:

```
git rm nazwa_pliku
```

4. Wyświetlenie statusu repozytorium w którym aktualnie się znajdujemy.

Jeżeli chcemy sprawdzić czy w repozytorium były wprowadzone jakieś zmiany, które nie zostały jeszcze zatwierdzone commitem, wykonujemy polecenie:

```
git status
```

5. Commitowanie wprowadzonych zmian.

Jeśli chcemy dokonać commitu (zatwierdzenia) dla wszystkich zmodyfikowanych plików i przesłać naszą wiadomość wywołujemy w konsoli:

```
git commit -am "Wiadomosc"
```

6. Wysyłanie zmian do zdalnego repozytorium.

Jeśli chcemy wysłać wprowadzone zmiany na wcześniej sklonowane repozytorium, to wydajemy polecenie:

```
git push origin master
```

Gdzie **origin** to domyślnie alias do adresu naszego wcześniej sklonowanego zdalnego rezpozytorium. Parametr **master** reprezentuje gałąź do której następuje przesyłanie zmian.

7. Pobieranie zmian ze zdalnego repozytorium.

Gdy jesteśmy w danym repozytorium, najłatwiej w tym celu użyć polecenia:

```
git pull
```

które pobierze zmiany ze zdalnego repozytorium oraz automatycznie dokona ich scalenia.[4]

Tworzenie zdalnych repozytoriów Gita

W momencie pracy z systemem kontroli wersji, często chcemy tworzyć własne repozytoria, czyli przestrzeń wydzieloną pod konkretny projekt. W przypadku Gita możemy stworzyć dwa rodzaje repozytoriów:

- **lokalne** - pracujemy na nich tylko na urządzeniu na którym są stworzone i mamy dostęp do folderu roboczego, czyli widzimy nasze pliki
- **zdalne** - repozytoria takie znajdują sie na serwerach, nie posiadają folderu roboczego, natomiast można do nich przesyłać zmiany z innych komputerów

Dla serwerów hostingowych z cPanelem, można utworzyć rezpozytoria z poziomu konsoli, łączac się z serwerem po SSH. Sposób połączenia z serwerem hostingowym został opisany w artykule

1. Aby stworzyć repozytorium zdalne pod ścieżką **/home/nazwa_uzytkownika/sciezka/do/repozytorium.git** wykonujemy

[4] https://panel.kylos.pl/knowledgebase/190/Podstawowe-operacje-wykonywane-na-repozytoriach-Gita-.html

następujące polecenie w konsoli SSH konta hostingowego:

```
git init --
bare /home/nazwa_uzytkownika/sciezka/do/repozytorium.git
```

2. Następnie możemy sklonować nasze zdalne repozytorium na lokalną maszyne tak, aby móc na nim pracować. W tym celu na lokalnej maszynie możemy wykorzystać klienta git lub użyć w konsoli systemowej polecenia:

```
git clone
ssh://nazwa_uzytkownika_cpanel@nazwakonta.kylos.pl:/sciezka/
do/repozytorium.git cd folder_z_projektem
```

UWAGA!

Należy pamiętać o tym, że łącząc się klientem Gita z serwerem hostingowym, musimy mieć prawidłowo skonfigurowany i uruchomiony w systemie klucz - dokładnie ten sam którego użyto do połączenia po SSH.[5]

Aby stworzyć nowe repozytorium Git w systemie Windows, wykonaj następujące kroki:

Korzystając z Git Bash:

1. **Otwórz Git Bash:**
   - Kliknij prawym przyciskiem myszy w folderze, w którym chcesz utworzyć repozytorium, a następnie wybierz opcję "Git Bash Here".
2. **Inicjalizacja Repozytorium:**
   - W terminalu Git Bash wpisz następujące polecenie, aby zainicjować nowe repozytorium:

     bashCopy code
     `git init`
3. **Dodaj Pliki:**
   - Dodaj pliki do repozytorium, używając polecenia **git add**. Na przykład:

     bashCopy code
     `git add .`
4. **Zatwierdź Zmiany:**
   - Zatwierdź dodane pliki za pomocą polecenia **git commit**:

     bashCopy code
     `git commit -m "Pierwszy commit"`

Korzystając z Git GUI (Git Bash Here):

1. **Otwórz Git GUI:**
   - Kliknij prawym przyciskiem myszy w folderze, w którym chcesz utworzyć repozytorium, a następnie wybierz opcję "Git GUI Here".

[5] https://panel.kylos.pl/knowledgebase/183/Tworzenie-zdalnych-repozytoriow-Gita-.html

2. **Inicjalizacja Repozytorium:**
   - Kliknij na przycisk "Create a new repository" i postępuj zgodnie z instrukcjami.
3. **Dodaj Pliki:**
   - W zakładce "Repository" wybierz opcję "Add" i dodaj pliki do repozytorium.
4. **Zatwierdź Zmiany:**
   - W zakładce "Commit", wprowadź wiadomość commita i kliknij "Commit".

Korzystając z Git w Konsoli Systemu Windows:

1. **Otwórz Konsolę Systemu Windows:**
   - Wciśnij klawisz **Win + R**, wpisz **cmd** i naciśnij Enter.
2. **Przejdź do Folderu:**
   - Użyj polecenia **cd** do przejścia do folderu, w którym chcesz utworzyć repozytorium.
3. **Inicjalizacja Repozytorium:**
   - Wpisz polecenie **git init** i naciśnij Enter.
4. **Dodaj Pliki:**
   - Użyj polecenia **git add** do dodania plików do repozytorium.
5. **Zatwierdź Zmiany:**
   - Użyj polecenia **git commit -m "Pierwszy commit"** do zatwierdzenia zmian.

Teraz utworzyłeś nowe repozytorium Git w wybranym folderze na systemie Windows. Możesz kontynuować pracę nad swoim projektem, dodawać pliki, zatwierdzać zmiany i korzystać z funkcji kontroli wersji oferowanych przez Git.[6]

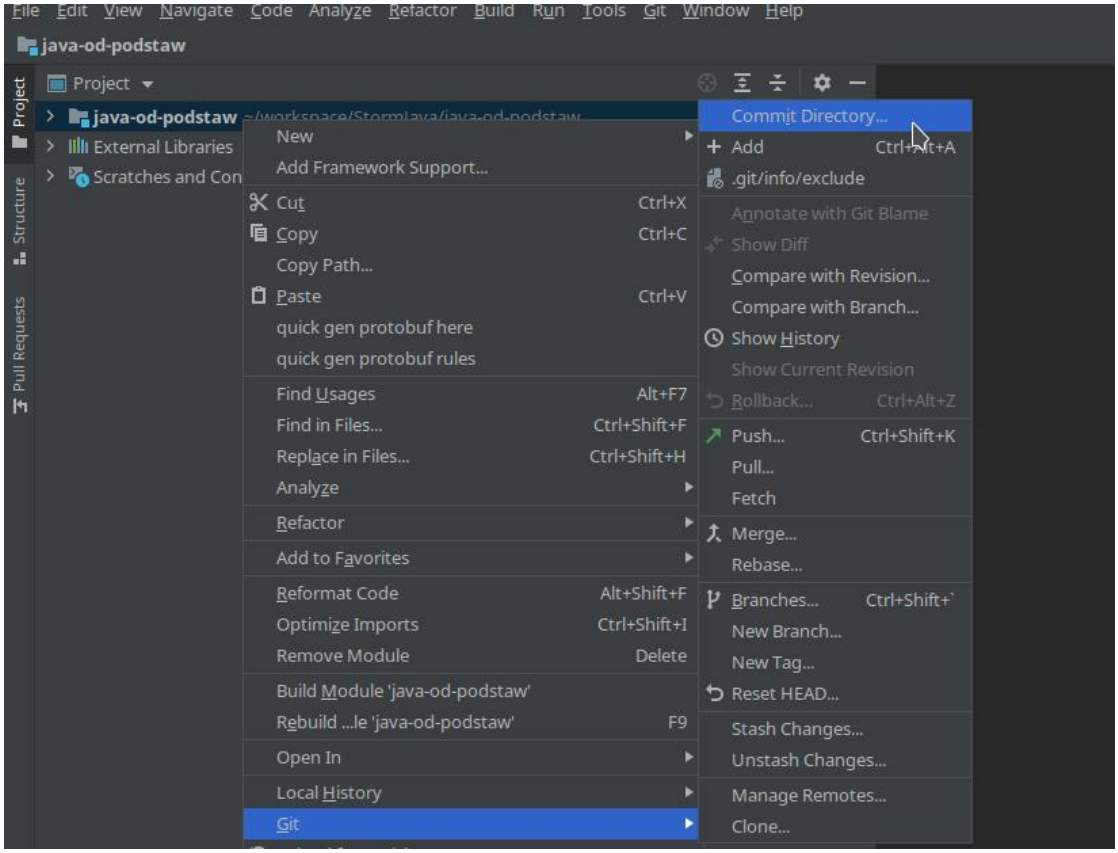
IntelliJ IDEA

Intellij IDEA jest jednym z popularniejszych IDE dla Java.

IDE – czyli **zintegrowane środowisko programistyczne** jest to aplikacja lub zespół aplikacji (środowisko) projektowany z myślą o maksymalizacji produktywności programisty. **IDE** charakteryzuje się tym, że udostępnia bardzo dużo różnorakich funkcjonalności, np. edycję kodu źródłowego, jego kompilację, automatyzację procesu budowania aplikacji, tworzenie baz danych, podpowiadanie składni, debugger, profiler – oraz wiele, wiele innych – czyli między innymi zarządzanie wersją kodu.

W tym wypadku mamy bardzo dobrą integrację naszego IDE z Gitem.

[6] https://chat.openai.com/



Powyżej widzisz zrzut ekranu z Intellij Idea z otwartym projektem

Więcej szczegółów: https://sii.pl/blog/jak-git-dziala-za-kulisami/

Zadania

Zadanie 1: Inicjalizacja Repozytorium

1. Zainicjuj nowe repozytorium Git w pustym folderze o nazwie "MojProjekt".

Zadanie 2: Dodawanie i Zatwierdzanie Zmian

1. Stwórz plik o nazwie "index.html" i dodaj kilka linijek kodu HTML.
2. Sprawdź status repozytorium i zobacz, że plik "index.html" jest nieśledzony.
3. Dodaj plik "index.html" do śledzenia.
4. Sprawdź ponownie status repozytorium.
5. Zatwierdź dodanie pliku "index.html" do repozytorium.

Zadanie 3: Tworzenie Brancha

1. Stwórz nowego brancha o nazwie "feature-navigation".
2. Przełącz się na nowo utworzony branch.

Zadanie 4: Modyfikacje na Branchu

1. Wprowadź zmiany w pliku "index.html" na branchu "feature-navigation".
2. Zatwierdź wprowadzone zmiany.

Zadanie 5: Scalanie Branchy

1. Przełącz się na branch "main".
2. Scalić zmiany z brancha "feature-navigation" do "main".
3. Rozwiązuj ewentualne konflikty, jeśli wystąpią.

Zadanie 6: Historia Repozytorium

1. Sprawdź historię repozytorium, wyświetlając listę commitów.

Zadanie 7: Tagowanie

1. Dodaj tag o nazwie "v1.0" do ostatniego commita.

Zadanie 8: Usuwanie Brancha

1. Usuń branch "feature-navigation".

Zadanie 9: Praca z Repozytorium na GitHub

1. Utwórz nowe repozytorium na GitHub.
2. Dodaj zdalne połączenie (remote) do swojego lokalnego repozytorium.
3. Wypchnij zmiany do repozytorium na GitHub.

Zadanie 10: Klonowanie Repozytorium

1. Sklonuj repozytorium "https://github.com/nazwa_uzytkownika/MojProjekt.git" na swój lokalny komputer.
2. Wprowadź jakieś zmiany, zatwierdź je i wypchnij na GitHub.