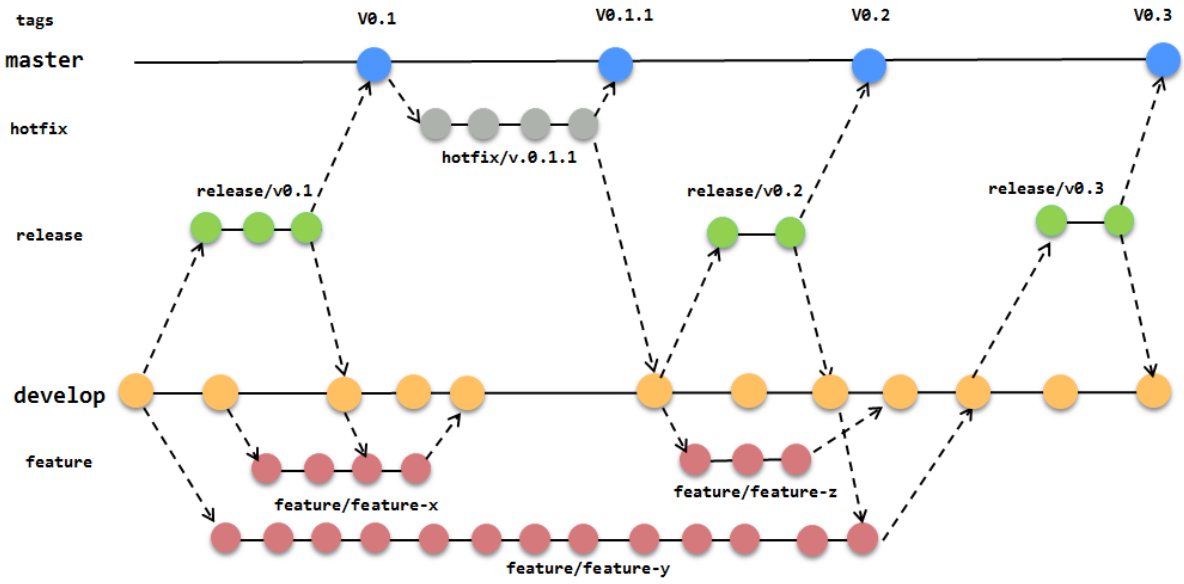
|  | Wydział Politechniczny<br><br>**Katedra Informatyki** |
|---|---|
| Prowadzący | Tomasz Gądek |
| Kurs | Narzędzia i środowiska programistyczne |
| Rok / Semestr | 1 / Letni |
| Temat | Lab07: Gitflow, .gitignore, README.md, rozgałęzianie i scalanie. |

Data ostatniej modyfikacji: **01-10-2024**

---

## Gitflow

Z wykorzystaniem **Git** wiąże się określona struktura repozytorium oraz schemat pracy (**workflow**), który został przedstawiony na poniższym diagramie. Jest to tzw. **Gitflow**.




Gitflow wykorzystuje 5 rodzajów gałęzi (**branch**), które opisuje poniższa tabela:

| Branch | Opis |
|---|---|
| master | **Master branch** odzwierciedla aktualną wersję systemu na środowisku produkcyjnym. |
| develop | **Develop branch** zawiera wszystkie nowe funkcjonalności i zmiany, które będą wdrażane w kolejnej wersji systemu. Odzwierciedla wersję systemu na środowisku testowym. |
| feature | **Feature branch** tworzone są na potrzeby implementacji nowych funkcjonalności. Zawsze tworzone są z branch develop. Po zakończeniu prac nad nową funkcjonalnością ostatecznie branch ten mergowany (scalany) jest do brancha develop. |
| release | **Release branch** jest tworzony w momencie gdy w branchu develop znajdują się już wszystkie nowe funkcjonalności zaplanowane do wdrożenia w kolejnej wersji systemu. Release branch tworzymy zawsze z brancha develop. |
| hotfix | **Hotfix branch** jest tworzony z brancha master na potrzeby wprowadzenia poprawki do krytycznego błędu wykrytego na środowisku produkcyjnym. |

Podstawowe polecenia Git

| Polecenie | Znaczenie |
|---|---|
| git init | Tworzenie nowego repozytorium Git |
| git clone [adres repozytorium] | Klonowanie repozytorium Git |
| git add [plik.rozszerzenie] \| [-A] \| [.] | Zaproponowanie zmian (lokalnie) |
| git commit -m "komentarz" | Zatwierdzanie zmian (lokalnie) |
| git push origin [nazwa brancha] | Wysłanie zmian do zdalnego repozytorium |
| git remote add origin [adres repozytorium] | Połączenie istniejącego repozytorium z serwerem |
| git branch [nazwa brancha] | Utworzenie nowej gałęzi |
| git checkout -b [nazwa brancha] | Utworzenie nowej gałęzi i przełączenie się na nowego brancha |
| git checkout [nazwa brancha] | Przełączenie się na brancha |
| git branch -d [nazwa brancha] | Usunięcie lokalnego brancha |
| git branch -m [old-name] [new-name] | Zmiana nazwy lokalnego brancha |
| git pull | Aktualizacja lokalnego repozytorium do ostatniego commita |
| git merge [nazwa brancha] | Scalanie aktywnej gałęzi ze wskazanym branchem |
| git log | historia commitów |
| git status | Stan plików w lokalnym repozytorium |
| git show-branch | Lista branchy |

Zanim rozpoczniesz...

- Utwórz nowe zdalne repozytorium na platformie **Bitbucket**.
- Otwórz **IntelliJ IDEA** oraz utwórz projekt **nisp_07**.
- Utwórz funkcję **main()**, który wyprowadzi na wyjście tekst **Hello World**.
- Skompiluj i uruchom projekt.
- Następnie otwórz terminal w IntelliJ IDEA. Uzyskasz do niego dostęp z poziomu menu IDE: **View / Tool Windows / Terminal**.
- Wykonaj polecenie *git init* i *git status*.

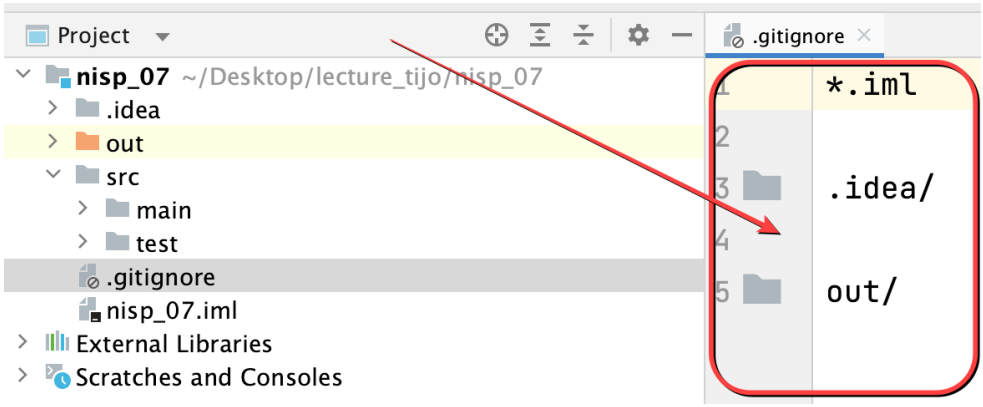
Status zmian powinien wyglądać następująco:

```
Nieśledzone pliki:
  (użyj „git add <plik>...”, żeby uwzględnić, co zostanie złożone)
        .idea/
        nisp_07.iml
        out/
        src/

nie dodano nic do złożenia, ale są nieśledzone pliki (użyj „git add”, aby śledzić)
```

Programistę interesują pliki źródłowe, które znajdują się w katalogu **src**. Pozostałe pliki i katalogi są niepotrzebne. Nie chcemy ich w zdalnym repozytorium. Aby wyłączyć śledzenie plików / katalogów skorzystamy z pliku .gitignore. Utwórz taki plik w głównym katalogu projektu (nie zapomnij o kropce na początku).

Zawartość **.gitignore**:




Dodaj pliki do poczekalni

```
git add .
```

Zatwierdź zmiany

```
git commit –m "Twoja wiadomość – NIE KOPIUJ! WYMYŚL!"
```

Następnie wyślij zmiany do zdalnego repozytorium utworzonego na platformie **Bitbucket** (tworzenie zdalnego repozytorium zostało opisane w **laboratorium 6**).

Po wysłaniu wszystkich zmian do zdalnego repozytorium powinieneś zobaczyć dodatkowy plik **.gitignore**. Plik **.gitignore** jest to specjalny plik, który zawiera informacje / reguły dotyczące plików, które **nie będą śledzone** przez rozproszony system kontroli wersji Git.

## README.md, dokumentacja programisty

Twój projekt może rozwijać się równolegle z dokumentacją. Utwórz plik **README.md**. Wklej do niego poniższy kod. Zamiast autora **Jan Kowalski** wprowadź własne dane.

````
# Programowanie w języku Kotlin

Język Kotlin - zorientowany obiektowo i funkcyjnie.

## Pierwszy program

```kotlin
fun main() {
       println("Hello World!")
}
```

## Kompilacja projektu

```bash
kotlinc Main.kt -include-runtime -d main.jar
```

## Uruchomienie projektu

```bash
java -jar main.jar
```

## Autor

Jan Kowalski

## Wikipedia
[O języku Kotlin](https://pl.wikipedia.org/wiki/Kotlin_(j%C4%99zyk_programowania))
````

Zatwierdź lokalne zmiany oraz prześlij je do zdalnego repozytorium. Otwórz plik **README.md** na platformie **Bitbucket**.

## Rozgałęzianie i scalanie

Zanim rozpoczniesz pracę utwórz branch (gałąź) develop:

```
git checkout –b develop
```

Polecenie skutkuje utworzeniem gałęzi develop oraz automatyczne przełączenie.

Zaprezentowane polecenie robi dokładnie to samo co dwa poniższe polecenia:

```
git branch develop
git checkout develop
```

Na tym etapie Twój kod jest identyczny, jak na branchu master.

Następnie wykonaj polecenie, aby zsynchronizować dane:

```
git push origin develop
```

Do weryfikacja gałęzi, na której się obecnie znajdujemy służy polecenie:

```
git branch
```

Dodaj kolejne funkcjonalności do projektu (**zanim zaczniesz cokolwiek robić doczytaj konspekt do końca!**):

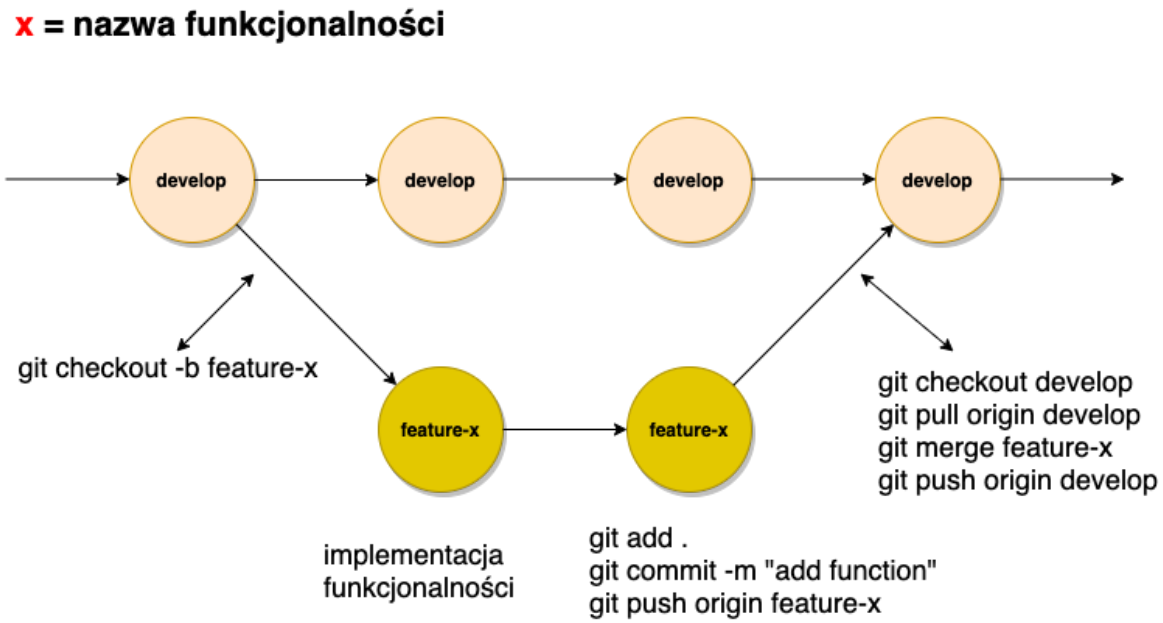
- Odejmowanie liczb całkowitych - funkcja: **sub(int, int)**
- Dodawanie liczb całkowitych - funkcja: **add(int, int)**
- Wyszukiwanie minimalnej wartości - funkcja **min(int, int)**
- Wyszukiwanie maksymalnej wartości - funkcja **max(int, int)**

Każda funkcjonalność to oddzielny **commit** oraz **branch**:

- feature-sub (x = sub - **spójrz na poniższą grafikę**).
- feature-add (x = add - **spójrz na poniższą grafikę**).
- feature-min (x = min - **spójrz na poniższą grafikę**).
- feature-max (x = max - **spójrz na poniższą grafikę**).

Podczas pracy postaraj się używać komend: **git status** oraz **git log**.

Wzoruj się na poniższym schemacie - to jest przepływ Twojej pracy:




Po skończonej pracy sprawdź wszystkie utworzone branche (gałęzie) na platformie **Bitbucket**.

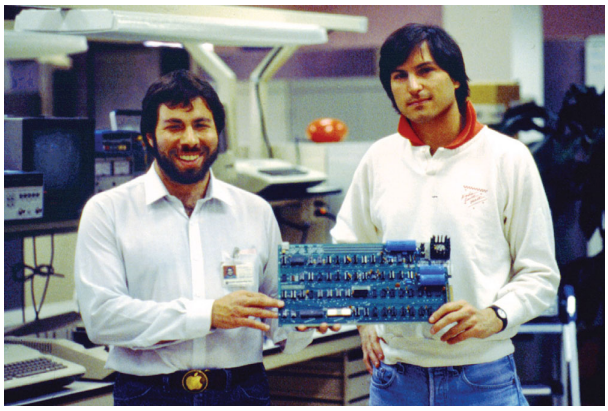
Lokalnie spróbuj przełączyć się pomiędzy różnymi gałęziami, np.:

```
git checkout feature-add
git checkout feature-sub
git checkout feature-min
git checkout feature-max
git checkout develop
```

Po każdej komendzie zweryfikuj, co takiego wydarzyło się w Twoim kodzie. Wyciągnij odpowiednie wnioski.

Czy wiesz, że...

Steve Wozniak, znany jako Woz, ma polskie korzenie? Jego dziadkowie ze strony ojca pochodzili z Polski. Wozniak sam jednak urodził się i dorastał w Stanach Zjednoczonych. Steve Wozniak odegrał kluczową rolę w rozwoju technologii komputerowej jako współzałożyciel Apple Inc.



Wspólnie z Steve'em Jobsem założyli Apple w 1976 roku, a pierwsze komputery Apple I i Apple II, zaprojektowane przez Wozniaka, odegrały kluczową rolę w ewolucji komputerów osobistych. Jego genialne projekty i innowacyjne podejście do technologii pomogły w tworzeniu nowych standardów dla przemysłu komputerowego. Wozniak jest powszechnie uznawany za jednego z pionierów technologii komputerowej.