Git i Gitub

Zasady pracy

- Pracujecie wyłącznie w PARZE**.**
- Ustalcie role: **Osoba A (Owner)** i **Osoba B (Collaborator)**.
- Każde działanie jasno wskazuje, co robi **A** i co robi **B**.

Zasady tworzenia commitów

1. **Język:** wszystkie komunikaty po **angielsku**.
2. **Forma:** zaczynamy od **czasownika w trybie rozkazującym** (imperative).
    - `Add`, `Create`, `Fix`, `Update`, `Remove`

3. **Treść:** commit opisuje **co zostało zrobione**, a nie co się wydarzyło.
    - `Add Gandalf to heroes list` - POPRAWNY
    - `Gandalf został dodany do listy` - NIEPOPRAWNY

4. **Długość:**

    - Maks. **50 znaków** w pierwszej linii (krótko i zwięźle).
    - Bez kropki na końcu.

5. **Spójność:** każdy commit powinien obejmować **jedną logiczną zmianę**.

    - Jeśli zmieniasz kilka plików w różnych celach, to zrób kilka commitów.

6. **Styl:** używaj **wielkiej litery na początku**.

7. **Przykłady poprawnych commitów:**
    - `Add base heroes.txt file with header`
    - `Create folder heroes/ with Gandalf details`
    - `Fix conflict in team.txt after merge`
    - `Remove old draft file`
8. **Przykłady niepoprawnych commitów:**
    - `Added heroes file`
    - `Creating new folder`
    - `Fixed conflit`
    - `remove file`

Etap 0 – Przygotowanie ról

- Wybierzcie w parze: A = Owner, B = Collaborator.
- Ustalcie identyfikatory do nazw gałęzi, np. asia i kuba.

Etap 1 – Założenie repozytorium na GitHubie

1. **A** tworzy publiczne repo: `heroes-pair-<A>-<B>` (np. `heroes-pair-asia-kuba`).
2. Zaznacz „Initialize this repository with a README”.
3. W **Settings** → **Collaborators** dodaj **B** z uprawnieniami **Write**.
4. Skopiujcie URL repo (HTTPS).

Rezultat: repo istnieje, **B** ma dostęp.

Etap 2 – Klon i pierwszy commit

- **A** i **B**: git clone <URL> i wejdźcie do katalogu roboczego.
- **A** tworzy lokalnie: heroes.txt z zawartością:

```
Lista bohaterów

---------------
```

- **A** dodaje pusty folder heroes/
- **A** przygotowane repo lokalne wysyła na githuba
- gdy to zrobi, **B** pobiera aktualne dane z Githuba do swojego repo lokalnego

Rezultat: wspólna baza na master.

Etap 3 – Gałęzie osobiste

- **A i B** pracują na **własnych gałęziach** (feature-asia-hero, feature-kuba-hero).
- Każdy edytuje **ten sam plik heroes.txt**.
- Każdy ma dodać swojego bohatera **dokładnie w 3. linii** (tuż pod nagłówkiem ----------).

np.

A pisze

```
Lista bohaterów
---------------
Gandalf
```

więc B wpisuje

```
Lista bohaterów
---------------
Batman
```

Etap 4 – Opisy bohaterów

- A i B pracują lokalnie na swoich gałęziach
- W katalogu `heroes/` tworzą plik `<bohater>.txt` z trzema informacjami:
    - Imię bohatera,
    - Świat (uniwersum),
    - Jedna cecha.

np.

```
Imię: Gandalf
Świat: Władca Pierścieni
Cecha: Mądry
```

- Commit + push (na swojej gałęzi `feature-...-hero`).

Etap 5 – Rebase i konflikty

- **A** jako pierwszy scala swoją gałąź z masterem (najpierw rebase na master, potem merge i push).
- **B** robi rebase swojej gałęzi na master po scaleniu zmian A → pojawia się konflikt w `heroes.txt` (bo oboje dopisali bohatera w tej samej linii).
- **B** ręcznie rozwiązuje konflikt, kontynuuje rebase, a następnie scala swoją gałąź do mastera i pushuje zmiany.

Etap 6 – Dodatkowy konflikt (A rozwiązuje)

- **A i B** równolegle tworzą nowe gałęzie (`feature-asia-team`, `feature-kuba-team`).
- Oboje tworzą i edytują ten sam plik `team.txt` – **w tej samej linii** wpisują różne drużyny bohaterów (celowo, aby wymusić konflikt).
- **B** jako pierwszy robi rebase na master i merge swojej gałęzi, więc jego zmiany lądują w repo.
- **A** dopiero teraz robi rebase na master, więc u niego pojawia się konflikt w `team.txt`.
- **A** rozwiązuje konflikt ręcznie, kończy rebase, scala gałąź do mastera i pushuje zmiany.