

Git w zespole — gałęzie, pull requesty i konflikty

Czas: 2 × 45 min

praca w zespołach 3-osobowych (lider A, osoby B i C); przy 2 osobach lider wykonuje też zadania C

Cel: Praca zespołowa w jednym repozytorium GitHub: Issues jako lista zadań, gałęzie funkcji, pull requesty z przeglądem kodu, rozwiązywanie konfliktu scalania i oznaczenie wersji tagiem.

Po zajęciach potrafisz:

- planuje pracę zespołu w GitHub Issues i łączy zmiany z zadaniami (Closes #n)
- tworzy gałęzie funkcji, wysyła je na GitHub i otwiera pull request
- przegląda kod kolegi, komentuje konkretne linie i zatwierdza zmiany
- rozwiązuje konflikt scalania lokalnie i rozumie znaczniki <<<<<< ===== >>>>>>
- oznacza wersję tagiem z adnotacją i pisze README projektu

Sprzęt i materiały

- komputer z Windows 11, Git i VS Code
- konto GitHub dla każdej osoby (darmowe)
- folder `r2_24_start` z Pliki_do_zadan

Oprogramowanie

- **Git for Windows** — Git Bash i polecenia `git` w terminalu VS Code
git-scm.com/downloads
- **GitHub** — repozytorium, Issues, pull requesty, Releases
github.com
- **Visual Studio Code** — panel Source Control i edytor konfliktów (Accept Current / Incoming / Both)
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie polecenia z zadania 28 roku 1 (commit, gałąź, merge, push).
- Sprawdź logowanie do GitHub z Git (np. `git push` do własnego repozytorium testowego); ustaw `git config --global user.name` i `user.email`.

Bezpieczeństwo

- Nie zapisuj haseł ani tokenów w plikach repozytorium; przy logowaniu używaj okna Git Credential Manager.
- Po zajęciach na komputerze szkolnym wyloguj się z GitHub i usuń zapisane poświadczenia (Menedżer poświadczeń Windows → Poświadczenia systemu Windows → `git:https://github.com`).

Przebieg zadania

1 Repozytorium zespołu i lista zadań (lider A)

Podział: #1 podstrona Projekty (B), #2 podstrona Kontakt (C), #3 styl aktywnego odnośnika w menu (C), #4 README (A).

1. Utwórz na GitHubie publiczne repozytorium `koło-programistów` (bez README). Skopiuj do lokalnego folderu pliki z `r2_24_start`, dodaj `.gitignore` (np. `node_modules/`, `.vscode/`, `Thumbs.db`), wykonaj pierwszy commit i `git push -u origin main`.
2. Dodaj członków zespołu: Settings → Collaborators → Add people. Każdy musi zaakceptować zaproszenie (e-mail lub github.com/notifications).
3. Utwórz Issues #1–#4 z opisem i przypisz osoby (Assignees). Dodaj etykiety `enhancement` lub `documentation`.
4. Włącz ochronę gałęzi main: Settings → Rules → Rulesets → New branch ruleset, cel: Default branch, reguła „Require a pull request before merging” z 1 wymaganą akceptacją, status Active.

2 Gałęzie funkcji (B i C równolegle)

B i C pracują jednocześnie, każde na swojej gałęzi. Oboje dopisują odnośnik w menu zaraz pod „Start” — konflikt jest częścią ćwiczenia.

```
git clone https://github.com/<lider>/koło-programistów.git
cd koło-programistów
git switch -c feature/projekty      # osoba C: feature/kontakt
# ... zmiany ...
git add .
git commit -m "Dodaj podstronę Projekty (#1)"
git push -u origin feature/projekty
```

1. B: utwórz `projekty.html` (ten sam nagłówek, menu i stopka, sekcja z 3 projektami koła) i dopisz w menu obu stron odnośnik „Projekty” bezpośrednio pod „Start”. Zrób co najmniej 2 commity.
2. C: utwórz `kontakt.html` z formularzem (imię, e-mail, wiadomość, przycisk) i dopisz w menu obu stron odnośnik „Kontakt” bezpośrednio pod „Start”. Zrób co najmniej 2 commity.
3. B i C: wyślij gałąź na GitHub i otwórz pull request do main. W opisie napisz, co zmieniono, i dodaj `Closes #1` (lub `Closes #2`).

3 Przegląd kodu i scalenie

Autor nie może zatwierdzić własnego pull requesta — robi to inna osoba z zespołu.

1. A: otwórz PR osoby B, karta Files changed. Dodaj komentarz do co najmniej jednej konkretnej linii (np. brak atrybutu alt, literówka), a następnie Review changes → Request changes lub Approve.
2. B: popraw kod zgodnie z uwagą, zrób commit i push na tę samą gałąź — sprawdź, że commit pojawił się w PR. Po akceptacji A scala PR (Merge pull request) i usuwa gałąź (Delete branch). Sprawdź, że Issue #1 zamknęło się automatycznie.
3. Zrób zrzut PR z komentarzem do linii i statusem Merged.

4 Konflikt scalania (C)

Po scaleniu PR #1 GitHub pokaże w PR osoby C: „This branch has conflicts that must be resolved”.

```
git switch feature/kontakt
git fetch origin
git merge origin/main          # CONFLICT (content): Merge conflict in index.html
# popraw pliki, usuń znaczniki konfliktu
git add index.html
git commit
git push
```

1. C: scal aktualny main do swojej gałęzi poleceniami z ramki. Otwórz `index.html` w VS Code i zrób zrzut znaczników konfliktu. Zapisz, co oznacza część HEAD, a co część origin/main.
2. Rozwiąż konflikt tak, aby menu miało kolejno: Start, Projekty, Kontakt (opcja Accept Both Changes i ewentualna poprawka ręczna). Sprawdź, że w żadnym pliku nie zostały znaczniki `<<<<<<` (`git grep "<<<<<<"`).
3. Zakończ scalanie commitem i wyślij gałąź. Po akceptacji B scal PR #2. Wykonaj w tym samym trybie Issue #3 (osobna gałąź i PR).

5 README, tag wersji i historia (A)

1. Na gałęzi `docs/readme` uzupełnij `README.md`: opis projektu, tabela zespołu (osoba, rola, Issues), uruchomienie, zasady pracy z gałęziami i PR, sekcja Wersje. Scal przez PR (Closes #4).
2. Wszyscy: `git switch main` i `git pull`. Lider: `git tag -a v1.0 -m "Wersja 1.0"` i `git push origin v1.0`, a następnie Releases → Draft a new release z tagiem v1.0 i krótkim opisem zmian.
3. Każdy: wykonaj `git log --oneline --graph --all` i zrób zrzut. Wskaż w historii commit scalający rozwiązujący konflikt.
4. Sprawdź w Insights → Contributors (lub Insights → Network), że każda osoba ma commity w repozytorium. Wpisz adres repozytorium w sprawozdaniu.

W sprawozdaniu umieść

- ☐ adres repozytorium zespołu i podział ról
- ☐ zrzuty: lista Issues, PR z komentarzem do linii, znaczniki konfliktu, historia `git log --graph`, Release v1.0
- ☐ opis rozwiązania konfliktu (co było po obu stronach i co zostawiono)
- ☐ odpowiedzi na pytania kontrolne (indywidualnie)

Pytania kontrolne

1. Dlaczego zespół pracuje na gałęziach funkcji zamiast commitować bezpośrednio do main?
2. Kiedy powstaje konflikt scalania, a kiedy Git scala zmiany automatycznie?
3. Czym różni się tag od gałęzi? Dlaczego `git push` nie wysyła tagów?
4. Co powinno się sprawdzać podczas przeglądu kodu (podaj 3 rzeczy)?

Kryteria oceny

Kryterium	Punkty
Repozytorium z .gitignore, współpracownicy, 4 Issues przypisane do osób, reguła ochrony main	3
Gałęzie funkcji z min. 2 commitami każdej osoby, PR z opisem i Closes #n	4
Przegląd kodu: komentarz do linii, poprawka w PR, scalenie, zamknięte Issue	3
Konflikt rozwiązany poprawnie (menu Start, Projekty, Kontakt, brak znaczników), zrzut i opis	4
README wg wzoru, tag v1.0 wysłany i Release, zrzut historii	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).