Wprowadzenie do pracy w środowisku Linux

Część 4: Wprowadzenie do Systemu Kontroli Wersji Git

Opracowanie dla studentów matematyki

9 grudnia 2025

Spis treści

1 Wprowadzenie do Kontroli Wersji **3**
1.1 Problem: Chaos w plikach . . . 3
1.2 Rozwiązanie: System Kontroli Wersji (VCS) . . . 3
1.3 Git: Król Systemów Kontroli Wersji . . . 3
1.4 Dlaczego Git to standard w branży? . . . 3

2 Praca z Gitem: Podstawowe Koncepty i Komendy **4**
2.1 Repozytorium: Twój projekt z superpamięcią . . . 4
2.2 Zapisywanie historii: cykl `add` i `commit` . . . 4
2.3 Ignorowanie Plików: plik `.gitignore` . . . 4
2.4 Gałęzie (Branches): Bezpieczne eksperymenty . . . 5

3 Ekosystem Git: GitHub i Dobre Praktyki **5**
3.1 Git vs. GitHub: Narzędzie kontra Platforma . . . 5
3.2 Praca ze Zdalnym Repozytorium (GitHub) . . . 5
3.3 Anatomia wzorowego repozytorium na GitHubie . . . 7
3.4 GUI dla Gita: GitHub Desktop . . . 7
3.5 Uwierzytelnianie: Osobisty Token Dostępu (PAT) . . . 7

1 Wprowadzenie do Kontroli Wersji

1.1 Problem: Chaos w plikach

Prawdopodobnie każdy spotkał się z problemem utrzymywania wielu wersji tego samego pliku: `praca_v1.doc`, `praca_v2_poprawiona.doc`, `praca_FINALNA.doc`. Taki sposób pracy jest chaotyczny i prowadzi do błędów.

1.2 Rozwiązanie: System Kontroli Wersji (VCS)

System Kontroli Wersji (VCS)

To oprogramowanie, które śledzi i zarządza zmianami w plikach w czasie. Rejestruje każdą modyfikację w specjalnej bazie danych, co pozwala na przeglądanie historii, cofanie się do poprzednich wersji, eksperymentowanie bez ryzyka i, co najważniejsze, efektywną współpracę wielu osób.

1.3 Git: Król Systemów Kontroli Wersji

Git

Git to najpopularniejszy na świecie, darmowy i otwarty **rozproszony** system kontroli wersji. Słowo "rozproszony"oznacza, że każdy członek zespołu ma na swoim komputerze **pełną kopię całej historii projektu**.

1.4 Dlaczego Git to standard w branży?

Nauka Gita to nie tylko kwestia techniczna – to inwestycja w swoją przyszłość zawodową.

Git jako fundament nowoczesnej pracy zespołowej

W każdej firmie technologicznej praca nad kodem odbywa się w zespole. Git jest językiem, za pomocą którego ten zespół komunikuje się w sprawach technicznych. Jego znajomość pozwala na:

- **Bezpieczną współpracę:** Git pozwala dziesiątkom osób pracować nad tym samym projektem jednocześnie, bez ryzyka nadpisania czyjejś pracy.
- **Zarządzanie złożonością:** Nowoczesne oprogramowanie to miliony linii kodu. Git pozwala na organizację tej złożoności poprzez gałęzie.
- **Utrzymanie porządku i odpowiedzialności:** Każda zmiana w Gicie jest "podpisana"przez autora i opatrzona opisem, co tworzy przejrzystą historię projektu.

2 Praca z Gitem: Podstawowe Koncepty i Komendy

2.1 Repozytorium: Twój projekt z superpamięcią

Repozytorium (Repository, "repo")

To folder z Twoim projektem, który jest śledzony przez Gita. Zawiera on wszystkie pliki projektu oraz ukryty podkatalog o nazwie `.git`, w którym Git przechowuje całą historię zmian.

Terminal – Bash

```
# Wejdź do katalogu ze swoim projektem
$ cd moj-projekt
# Zainicjuj puste repozytorium Gita
$ git init
```

2.2 Zapisywanie historii: cykl `add` i `commit`

Praca w Gicie to powtarzalny cykl: **1. Modyfikuj pliki** → **2. Dodaj do poczekalni** (`git add`) → **3. Zatwierdź** (`git commit`).

- `git status` – **Twoja najważniejsza komenda!** Pokazuje, które pliki zostały zmienione, które są w poczekalni, a które nie są śledzone.
- `git add <plik>` – dodaje zmiany z konkretnego pliku do "poczekalni"(Staging Area).
- `git commit -m "Opis zmian"` – tworzy nowy "punkt zapisu"(commit) z plików w poczekalni.
- `git log` – wyświetla historię wszystkich commitów.

Terminal – Bash

```
# Zobaczmy status repozytorium
$ git status
# Stwórzmy nowy plik
$ echo "Pierwsza linia" > plik.txt
# Dodajmy plik do poczekalni, aby Git zaczął go śledzić
$ git add plik.txt
# Zatwierdźmy zmiany, tworząc pierwszy commit
$ git commit -m "Dodano plik.txt z pierwszą linią"
# Zobaczmy historię
$ git log
```

2.3 Ignorowanie Plików: plik `.gitignore`

Plik `.gitignore` to prosta lista wzorców, które Git ma ignorować. Jest kluczowy, aby do repozytorium nie trafiały pliki tymczasowe, dane wrażliwe (hasła!) czy pliki systemowe.

Dobra praktyka: Stwórz `.gitignore` na samym początku!

Nie musisz pisać tego pliku od zera. Serwisy takie jak `gitignore.io` pozwalają wygenerować gotowe szablony.

2.4 Gałęzie (Branches): Bezpieczne eksperymenty

Gałąź (Branch)

To **ruchoma etykieta** wskazująca na konkretny commit. Gałęzie pozwalają na tworzenie niezależnych, równoległych linii rozwoju. Zawsze twórz nową gałąź dla nowego zadania!

- `git branch <nazwa>` – tworzy nową gałąź.
- `git switch <nazwa>` – przełącza się na istniejącą gałąź.
- `git switch -c <nazwa>` – tworzy nową gałąź i od razu się na nią przełącza.
- `git merge <nazwa>` – łączy zmiany z gałęzi `<nazwa>` do tej, na której aktualnie jesteś.

Terminal – Bash

```
# Stwórz nową gałąź i przełącz się na nią
$ git switch -c nowa-funkcja
# ... pracuj na plikach, rób commity ...
# Wróć na gałąź główną
$ git switch main
# Połącz zmiany z `nowa-funkcja` do `main`
$ git merge nowa-funkcja
```

3 Ekosystem Git: GitHub i Dobre Praktyki

3.1 Git vs. GitHub: Narzędzie kontra Platforma

- **Git** to narzędzie działające w wierszu poleceń na Twoim komputerze. To "silnik".
- **GitHub** to platforma internetowa, która służy do **hostowania** repozytoriów Gita. To "chmura"dla Twojego kodu.

3.2 Praca ze Zdalnym Repozytorium (GitHub)

Twoje lokalne repozytorium może być połączone ze zdalną kopią, np. na GitHubie. Ta zdalna kopia, nazywana domyślnie `origin`, jest centralnym punktem dla współpracy.

Wysyłanie zmian na serwer: `git push`

Polecenie `git push` wysyła Twoje lokalne commity (z konkretnej gałęzi) do zdalnego repozytorium. To jak synchronizacja zapisów z chmurą.

Terminal – Bash

```
# Wyślij zmiany z lokalnej gałęzi `main` do zdalnej `origin`
$ git push origin main
```

Przy pierwszym wysłaniu nowej gałęzi, użyj `git push -u origin nazwa-galezi`. Opcja `-u` tworzy "powiązanie śledzące", dzięki czemu w przyszłości wystarczy wpisać samo `git push`.

Pobieranie zmian z serwera: `git pull`

Polecenie `git pull` robi odwrotną rzecz: pobiera najnowsze zmiany ze zdalnego repozytorium i łączy je z Twoją lokalną wersją. To kluczowa komenda w pracy zespołowej – **zawsze wykonuj ją przed rozpoczęciem pracy**, aby mieć pewność, że pracujesz na aktualnym kodzie.

Terminal – Bash

```
# Pobierz i połącz najnowsze zmiany z gałęzi `main` na serwerze
$ git pull origin main
```

Nie panikuj! Wstęp do rozwiązywania konfliktów

Czasami, podczas `git pull` lub `git merge`, Git napotka **konflikt**. Dzieje się tak, gdy Ty i inna osoba zmodyfikowaliście **te same linie w tym samym pliku**. Git nie wie, która wersja jest poprawna, więc zatrzymuje proces i prosi Ciebie o podjęcie decyzji.
W pliku objętym konfliktem zobaczysz specjalne znaczniki:

```
<<<<<<< HEAD
Twoja zmiana (to, co miałeś lokalnie)
=======
Zmiana, która nadeszła z serwera
>>>>>>> nazwa_commita_lub_galezi
```

Twoim zadaniem jest:

1. Otworzyć plik w edytorze.
2. Zdecydować, która wersja kodu ma zostać. Możesz wybrać swoją, cudzą, albo połączyć obie.
3. **Usunąć wszystkie znaczniki** dodane przez Gita (**<<<**, **===**, **>>>**).
4. Zapisać plik.
5. Dodać rozwiązany plik do poczekalni (`git add <plik>`) i zakończyć scalanie, tworząc nowy commit (`git commit`).

Konflikty są normalną częścią pracy w zespole. Najważniejsze to zrozumieć, dlaczego powstały, i spokojnie je rozwiązać.

3.3 Anatomia wzorowego repozytorium na GitHubie

Dobrze prowadzone repozytorium jest czytelne i łatwe w nawigacji. Powinno zawierać:

- **Plik `README.md`:** To "strona główna"Twojego projektu z opisem i instrukcjami.
- **Plik `.gitignore`:** Dba o to, by w repozytorium nie znalazły się niepotrzebne pliki.
- **Przejrzysta historia commitów:** Każdy commit jest mały i ma jasny, zrozumiały opis.

3.4 GUI dla Gita: GitHub Desktop

GitHub Desktop

To oficjalna, darmowa aplikacja od GitHuba, która pozwala na wykonywanie większości operacji w Gicie za pomocą kliknięć, a nie komend. Jest świetna na początek.

3.5 Uwierzytelnianie: Osobisty Token Dostępu (PAT)

Zamiast hasła, do uwierzytelniania w terminalu używa się **Osobistego Tokenu Dostępu (PAT)**.

- **Jak go wygenerować?:** Zaloguj się na GitHubie, wejdź w `Settings` → `Developer settings` → `Personal access tokens` i kliknij `Generate new token`.
- **Ważne:** Po wygenerowaniu tokena **skopiuj go i zapisz w bezpiecznym miejscu**. Już nigdy więcej go nie zobaczysz!
- **Jak go użyć?:** Gdy terminal poprosi o hasło podczas `git push`, wklej skopiowany token.

Pełny cykl pracy: od zera do GitHuba

Scenariusz A: Pierwsze wysłanie projektu na GitHuba

1. Stwórz nowe, puste repozytorium na stronie GitHub.com.
2. W swoim lokalnym folderze projektu, zainicjuj repozytorium Gita:
   ```
   git init
   ```
3. Dodaj wszystkie pliki do poczekalni i stwórz pierwszy commit:
   ```
   git add .
   git commit -m "First commit"
   ```
4. Połącz swoje lokalne repozytorium ze zdalnym na GitHubie (URL skopiuj ze strony GitHuba):
   ```
   git remote add origin link_do_twojego_repozytorium
   ```
5. Upewnij się, że główna gałąź nazywa się `main` (dobra praktyka):
   ```
   git branch -M main
   ```

6. Wyślij swoje commity na serwer GitHuba. Opcja `-u` ustawia zdalną gałąź jako domyślną dla przyszłych poleceń `push`:
   `git push -u origin main`

Scenariusz B: Codzienny cykl pracy w zespole

1. **Zsynchronizuj się:** Zawsze zaczynaj od pobrania najnowszych zmian.
   (`git pull origin main`)
2. **Stwórz nową gałąź:** Utwórz i przełącz się na nową gałąź dla swojego zadania.
   (`git switch -c moja-nowa-funkcja`)
3. **Pracuj:** Wprowadzaj zmiany i regularnie rób małe, dobrze opisane commity.
   (`git add .` → `git commit -m "..."`)
4. **Wyślij gałąź na serwer:** Gdy Twoja praca jest gotowa do oceny.
   (`git push origin moja-nowa-funkcja`)
5. **Stwórz Pull Request:** Na stronie GitHub.com, aby poprosić o włączenie Twoich zmian.
6. **Scal (Merge):** Po akceptacji, Twoje zmiany stają się częścią głównej gałęzi projektu.