Warsztat Programisty – materiały do zajęć

Łukasz Kuszner, Radosław Ziemann

Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki
Zakład Optymalizacji Kombinatorycznej

13 stycznia 2025

7 Git - zapoznanie z systemem

Git to rozproszony system kontroli wersji (Version Control System) używany powszechnie przy tworzeniu projektów informatycznych. Pod następującym adresem URL znajdują się pomocne materiały na jego temat: https://git-scm.com/book/pl/v2. W szczególności proszę zwrócić uwagę na treści zawarte w 3.1 – 3.3.

7.1 Wstępna konfiguracja

Mając zainstalowanego Gita na naszym komputerze lokalnym lub serwerze musimy skonfigurować podstawowe dane dotyczące użytkownika: nazwę i mail. Aby to zrobić w konsoli wpisujemy następujące komendy:

```
git config --global user.name "Imię Nazwisko"
git config --global user.email imie.nazwisko@ug.edu.pl
```

Po ich wprowadzeniu powinien zostać nadpisany (lub powstać nowy) plik konfiguracyjny `.gitconfig` znajdujący się w katalogu domowym. Jego zawartość możemy sprawdzić za pomocą `cat ~/.config`. Oczywiście plik `.config` może zawierać więcej danych, jednak na tym etapie nie ma potrzeby ich ustawiania. Aby wyświetlić wszystkie ustawienia systemowe możemy wpisać `git config -l` lub `git config --list`.

7.2 Tworzenie repozytorium - katalog roboczy, katalog Gita i przechowalnia

Zaczynamy od stworzenia osobnego katalogu, w którym będziemy przechowywać własne repozytoria. Przykładowo nazwijmy go `repoGit`. W nim tworzymy katalog roboczy o nazwie `repo1`, w którym będzie się znajdować nasze pierwsze repozytorium, a następnie przechodzimy do niego.

`Katalog roboczy` (ang. working directory) stanowi obraz jednej wersji projektu. Zawartość tego katalogu pobierana jest ze skompresowanej bazy danych zawartej w katalogu Gita (o nim za chwilę) i umieszczana na dysku w miejscu, w którym można ją odczytać lub zmodyfikować. A więc katalog roboczy to po prostu katalog z plikami i podkatalogami, w którym oprócz plików projektu, które są już w bazie danych Gita, są także inne które możemy w przyszłości dołączyć do projektu. My tworzymy nasze repozytorium od zera - nasz katalog roboczy na razie jest pusty, nie ma także jeszcze żadnej wersji projektu. Żeby móc dodawać pliki do repozytorium i przechowywać je w jego bazie danych musimy stworzyć katalog Gita poleceniem `git init`.

`Katalog Gita` (.git) jest miejscem, w którym Git przechowuje własne metadane oraz obiektową bazę danych Twojego projektu, zawierającą wszystkie poprzednie oraz aktualną jego wersję. To najważniejsza część Gita i to właśnie ten katalog jest kopiowany podczas klonowania repozytorium (zobacz podsekcję klonowanie repozytorium).

Ćwiczenie 30. Stosując polecenie `ls -la` zaobserwuj powstanie nowego katalogu/katalogów. Przejrzyj jego zawartość.

UWAGA: Wszystkie kolejne polecenia będziemy wykonywać w katalogu roboczym `repo1`.

Zarówno nasz katalog roboczy jak i katalog Gita są "puste", tzn. nie dodaliśmy tam żadnych naszych plików. Stwórzmy więc dwa nowe puste pliki `one.sh` i `two.sh`. Pliki te są na razie tylko w katalogu roboczym (katalog Gita ich nie zawiera). Sprawdzamy zmiany w katalogu roboczym poleceniem `git status`. Po jego wykonaniu powinny nam się wyświetlić następujące komunikaty:

```
On branch master

No commits yet

Untracked files:
```

```
  (use "git add <file>..." to include in what will be committed)
        one.sh
        two.sh

nothing added to commit but untracked files present (use "git add" to track)
```

Git podpowiada nam, że utworzone pliki nie są "śledzone" (ang.tracked), czyli dodane do przechowalni (poczekalni).

`Przechowalnia` (ang. staging area) to prosty plik, zwykle przechowywany w katalogu Gita, który zawiera informacje o tym, czego dotyczyć będzie następna operacja zapisu (ang. commit), po której zmianie ulegnie katalog Gita - w naszym przypadku powstanie pierwsza wersja projektu.
Możemy teraz dodać pliki do przechowalni następującym poleceniem:

```
git add one.sh two.sh
```

Po kolejnym sprawdzeniu statusu w konsoli wyświetla się:

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   one.sh
        new file:   two.sh
```

Aby nasze zmiany (powstanie plików) trafiły do bazy danych (katalogu Gita) musimy je teraz zapisać (zatwierdzić) z odpowiednim komentarzem:

```
git commit –m "Dodanie plików one.sh i two.sh"
```

W konsoli powinniśmy otrzymać odpowiedni komunikat:

```
[master (root–commit) c611077] Dodanie plików one.sh i two.sh

 2 files changed, 1 insertion(+)
 create mode 100644 one.sh
 create mode 100644 two.sh
```

Za pomocą komendy `git log` możemy sprawdzić historię naszych zapisów ("commitów").

7.3 Modyfikacja plików

W czasie pracy nad projektem programista często zmienia zawartość plików, które trafiły już do bazy danych (są "zacommitowane") i są częścią jakiejś wersji (migawki, ang. snapshot) projektu. W takim przypadku zmiany trzeba także zatwierdzić, a Git będzie pamiętał kolejne wersje naszego projektu. Nasz poprzednio dodany skrypt `one.sh` był pustym plikiem.

Ćwiczenie 31. Zmień zawartość pliku one.sh, np. dopisz do niego komunikat: "To zmodyfikowany plik skryptu".

Sprawdzając status repozytorium zobaczymy, że nasz plik został zmodyfikowany:

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   one.sh

no changes added to commit (use "git add" and/or "git commit -a")
```

Plik one.sh jest już "śledzony" (jest w poczekalni), więc wystarczy tylko "zacommitować" zmiany.

Ćwiczenie 32. Zatwierdź zmiany i sprawdź czy nowy zapis ("commit") na pewno został dodany do bazy danych.

Wszystkie pliki z danego projektu możemy podzielić na trzy grupy:

1. *zmodyfikowany* – oznacza, że plik został zmieniony, ale zmiany nie zostały wprowadzone do bazy danych Gita (jest tylko w katalogu roboczym),
2. *oczekujący w przechowalni* (ang. staged) – oznacza, że zmodyfikowany plik został przeznaczony do zatwierdzenia (wysłania do bazy danych w katalogu Gita) w bieżącej postaci w następnej operacji zapisu ("commit").
3. *zatwierdzony* – oznacza, że dane zostały bezpiecznie zachowane w Twojej lokalnej bazie danych (w katalogu Gita),

Podsumowując zdobytą do tej pory wiedzę, możemy streścić ogólny sposób pracy z Gitem następująco:

1. dokonujemy modyfikacji plików (lub je tworzymy) w katalogu roboczym,
2. oznaczamy zmodyfikowane (stworzone) pliki jako śledzone (ang. tracked), dodając ich bieżący stan do przechowalni,
3. dokonujemy zatwierdzenia ("commit"), podczas którego zawartość plików z przechowalni zapisywana jest jako migawka projektu w katalogu Gita.
4. wracamy do 1.

7.4 Praca z gałęziami

7.4.1 Tworzenie nowej gałęzi

Wszystko co dotąd robiliśmy było zapisywane jako kolejne wersje (migawki) naszego projektu jedna po drugiej. Czasem jednak programista chce wypróbować nowe pomysły, które nie zawsze chce później umieścić w projekcie. Wtedy przydaje się utworzenie nowej gałęzi projektu, czyli alternatywnego ciągu zatwierdzeń kolejnych zmian, a tym samym pobocznej wersji powstającego projektu. Tworząc nową gałąź rozdzielamy repozytorium na dwie niezależne części, które mają jakąś wspólną historię zapisów ("commitów"), ale od ich rozdzielenia są niezależne.

Kolejne wersje naszego projektu były tworzone na gałęzi głównej (branch master), która powstaje przy tworzeniu nowego repozytorium. Załóżmy teraz, że na osobnej gałęzi chcemy stworzyć nowy plik `three.sh`, który będzie coś wyświetlał. Najpierw tworzymy nową gałąź: `git branch Nowa` i przechodzimy do niej: `git checkout Nowa`.

Ćwiczenie 33. Stwórz plik `three.sh`, prześlij do przechowalni i zatwierdź z komunikatem "Dodano plik three.sh". Sprawdź log i upewnij się, że są tam trzy commity, dwa z gałęzi `master` i jeden z gałęzi `Nowa`.

Poleceniem `git checkout master` możemy w każdej chwili wrócić na gałąź master i na niej dalej pracować.

7.4.2 Scalanie gałęzi

Po stworzeniu nowego pliku na gałęzi `Nowa` chcemy, żeby istniał on także na gałęzi master razem z jego wcześniejszymi "commitami" i zmianami. W takim celu służy nam operacja scalania, która polega na połączeniu dwóch gałęzi w jedną.

Po przejściu na gałąź master wykonujemy polecenie: `git merge Nowa` i jeśli wszystko się udało możemy teraz usunąć niepotrzebną gałąź: `git branch −d Nowa`.

7.5 Praca ze zdalnym repozytorium

Aby móc współpracować w jakimkolwiek projekcie opartym na Gicie, musisz nauczyć się zarządzać zdalnymi repozytoriami. Zdalne repozytorium to wersja Twojego projektu utrzymywana na serwerze dostępnym poprzez Internet lub inną sieć.

Zarządzanie zdalnymi repozytoriami obejmuje umiejętność dodawania zdalnych repozytoriów, usuwania ich jeśli nie są dłużej poprawne, zarządzania zdalnymi gałęziami oraz definiowania je jako śledzone lub nie, i inne. Możesz mieć ich kilka, z których każde może być tylko do odczytu lub zarówno odczytu jak i zapisu.

Zdalne repozytoria bardzo ułatwiają współpracę w grupie, a zarządzanie nimi polega głównie na wypychaniu (ang. push) zmian na zewnątrz i pobieraniu ich w celu współdzielenia pracy/kodu.

Ćwiczenie 34. Jeśli masz dostęp do serwera utwórz repozytorium `repo1` na swoim koncie.

Teraz z `repo1` można korzystać na każdym komputerze z zainstalowanym Gitem poprzez operację klonowania (stworzy się lokalna kopia repozytorium).

7.5.1 Klonowanie i aktualizacja

Klonowanie (ang. cloning) poprzez `ssh` wykonujemy za pomocą następującego polecenia (będąc w folderze, w którym chcemy mieć kopię `repo1`): `git clone login@nazwa_serwera:~\repoGit\repo1`.

Gdy dokonamy jakiejś zmiany i ją zatwierdzimy to możemy wysłać migawkę naszego zmodyfikowanego repozytorium na serwer, aby inni programiści mogli zobaczyć nasze zmiany (pobrać aktualny stan `repo1` za pomocą klonowania) . Wpisanie w terminalu `git push origin master` spowoduje wypchnięcie (ang.

push) zmodyfikowanego repozytorium (domyślnie pod `master` zawarta jest nazwa/adres naszego repozytorium) na gałąź `master`.

7.5.2 Gitlab

Jednym z hostingowych serwisów internetowych oferujących oprogramowanie przeznaczone dla projektów programistycznych jet GitLab (gitlab.com). Jest on oparty na systemie kontroli wersji Git oraz oprogramowaniu wspomagającym zarządzanie projektami opartymi na Git.

Ćwiczenie 35. Proszę założyć konto w serwisie Gitlab oraz utworzyć za pomocą przeglądarki internetowej i interfejsu graficznego nową publiczną grupę, a następnie utworzyć w niej w podobny sposób nowy projekt (repozytorium) o nazwie Repo2. Proszę także korzystając z ikonki + utworzyć nowy plik o nazwie program.c, w miejscu na komunikat (commit message) napisać "Dodanie pliku program.c" i zatwierdzić zmiany. W ten sposób utworzymy nowe repozytorium zawierające dokładnie jeden plik.

Ćwiczenie 36. Będąc w katalogu repoGit wykonaj polecenie klonowania repo2, sprawdzając wcześniej adres URL lub SSH repozytorium. Przykładowy adres SSH: git@gitlab.com:nazwa_grupy/repo2.git. Następnie proszę zmodyfikować plik program.c tak, aby wyświetlał w konsoli napis "Hello world" oraz dodać kolejny plik skrypt.sh, który wypisuje na ekranie "Tu skrypt". Zatwierdzić zmiany i wypchnąć je na serwer Gitlab'a. Sprawdzić w przeglądarce, że repo2 uległo aktualizacji.