Politechnika Wrocławska

Wprowadzenie do `git`-a w 5 minut

mgr inż. Łukasz Janiec

7 marca 2022

Spis treści

1 Czym jest git?

Git to najpopularniejszy z obecnych na rynku systemów kontroli wersji. System kontroli wersji to oprogramowanie pomagające nam zarządzać naszym projektem. Git to w pełni darmowe, wolne oprogramowanie na licencji GNU General Public License ver. 2.0.

1.1 Lokalne zastosowanie

Wyobraźmy sobie, że rozwijamy jakiś większy program. Wprowadzamy wiele zmian, pliki z kodem są dodawane i usuwane. W pewnym momencie chcielibyśmy z jakiegoś powodu wrócić się do wersji naszego programu sprzed miesiąca. Tutaj pojawia się problem - jeśli nie mamy backupu, to jest to praktycznie niemożliwe, stare pliki są już dawno nadpisane.

Taki problem nie wystąpi, jeśli od początku projektu będziemy korzystać z systemu kontroli wersji. Git zapamiętuje historię naszych zmian, zaczynając od pustego repozytorium (główny folder z naszym projektem będziemy nazywać `repozytorium`), aż do programu w w ich aktualnej

wersji. Projekt reprezentowany jest jako sekwencja małych zmian, które doprowadziły od stanu początkowego do stanu obecnego. Pojedyncza zmiana w repozytorium jest określana jako `commit`.

W tym miejscu pojawia się pytanie - jak często zmiany są zapisywane? Czy każda zmieniona linijka kodu traktowana jest jako osobny commit? O tym decyduje już użytkownik, commity nie są tworzone automatycznie. W momencie gdy uznamy, że wprowadzone przez nas zmiany są już wystarczająco istotne, wpisujemy odpowiednią komendę i nasza lokalna historia zmian zwiększa się o jednego commita. Być może brzmi to jeszcze zbyt abstrakcyjnie, jednak wszystko na pewno się rozjaśni, kiedy w sekcji 2 wykonamy krok po kroku wszystkie czynności potrzebne do utworzenia pierwszego commita.

Nie ma sztywnej reguły mówiącej jak często robić commity. Dobrze jest jednak zachowywać pewną spójność, nie robiąc ich w środku pracy nad jakąś małą zmianą, tylko po jej zakończeniu i testach działania.
Wszystko jest jednak wyłącznie kwestią preferencji użytkownika. Jednorazowe wrzucenie całego programu **NIE JEST** poprawnym zachowaniem.

1.2 Git w chmurze

Czynności opisane w sekcji 1.1 mogą być z powodzeniem wykonywane nawet na komputerze bez dostępu do internetu. Jednak Git ma znacznie więcej zastosowań. Istnieją serwisy, takie jak GitHub czy GitLab, które pozwalają na przechowywanie kopii naszego repozytorium w chmurze. Pozwala to dodatkowo na:

1. synchronizację podczas pracy na różnych komputerach,
2. dzielenie się naszym otwartym oprogramowaniem z resztą świata,
3. pracę nad tym samym projektem przez wielu programistów jednocześnie.

To właśnie zastosowanie 3 stanowi główną wartość systemów kontroli wersji w świecie IT. Dzięki niemu możliwe są ogromne kolaboracyjne projekty, jak np. rozwój jądra Linux. Jednak dla nas, w ramach zajęć z Podstaw Programowania/Programowania Obiektowego, nie będzie to jeszcze potrzebne. Skupimy się raczej na zastosowaniu 2, aby udostępniać swoje postępy wykonywania zadań prowadzącemu.

(a)  (b)  (c) 

Rys. 1: Loga: (a) `git`, (b) `GitLab`, (c) `GitHub`

2 Użytkowanie Gita

2.1 Tworzenie zdalnego repozytorium do udostępniania zadań

Stwórzmy konto na GitLabie (pod kursy uczelniane - konto z e-maila studenckiego(**!**)) i stwórzmy tam nowy projekt. Możemy dodać opis projektu i plik `README`, opisujący dany projekt i jego dokumentację. Przy publicznych projektach udostępnianych szerszej publice jest to wysoce wskazane.

My będziemy tworzyć raczej prywatne repozytoria, aby niepowołane osoby nie miały dostępu do naszego kodu.
Pamiętajmy jednak, aby dać wgląd w kod prowadzącemu za pomocą:

- `Settings > Members > Invite member`

Do swojego repozytorium należy dodać użytkownika `ljaniec` lub analogiczne konto Państwa prowadzącego. Proszę o dodawanie jako `Maintainer`, ponieważ jako `Guest` w repozytoriach prywatnych nie ma się wglądu do kodu. Państwa login powinien pozwolić mi na łatwą identyfikację (numer albumu, login na diablo/inyo od KCiR). W pliku `README` proszę o zapisanie nazwy kursu, prowadzącego i swojego imienia, nazwiska i numeru indeksu, obok numeru i nazwy zadania. Pierwsze Państwa repozytorium powinno zawierać obok pierwszego zadania także oświadczenie o samodzielnej pracy (strona prowadzącego).

2.2 Klonowanie repozytorium

Pod przyciskiem `Clone` na `GitLabie` powinien znajdować się adres HTTPS naszego repozytorium (kończący się na `.git`), skopiujmy go, użyjemy go później. Na koniec wygenerujmy zabezpieczający klucz SSH i przypiszmy go do naszego konta. Proces ten jest opisany w wielu istniejących już poradnikach.

2.3 Pierwszy commit

Polecenia zawarte w tej sekcji należy wykonywać w terminalu. Zalecanym systemem jest oczywiście GNU/Linux lub macOS. Istnieją narzędzia do robienia tego na Windowsie, jednak są one zdecydowanie mniej wygodne. Możliwe jest także użycie odpowiednich wtyczek np. w Visual Studio Code (wtyczka GitLens).

Na początku skonfigurujmy Gita wpisując podstawowe informacje o sobie:

```
git config --global user.name "nasze imie i nazwisko"
git config --global user.email "nasz mail"
```

W tym kroku sprawdzamy przy okazji, czy `git` jest w ogóle zainstalowany na naszym komputerze. Następnie otwórzmy terminal w miejscu, w którym chcemy utworzyć główny folder z naszymi projektami. Skopiujmy tam nasze utworzone zdalnie repozytorium:

```
git clone <adres>
```

Gdzie `<adres>` to adres naszego repozytorium skopiowany w punkcie 2.2. Po wpisaniu `ls` zauważymy, że pojawił nam się nowy folder. Na potrzeby tej instrukcji załóżmy, że nazywa się on `Kurs-Programowania`. Wejdźmy do niego i sprawdźmy jego stan.

```
cd Kurs-Programowania
git status
```

Wśród wyświetlonych informacji powinniśmy zobaczyć `Nothing to commit (create/copy files and use ’’git add’’ to track)`. Oznacza to, że nie dokonaliśmy jeszcze żadnych zmian. Skopiujmy więc do naszego repozytorium foldery z plikami projektu, na których do tej pory pracowaliśmy. W tym momencie `git status` zwróci nam informację o pojawieniu się nowych plików i folderów. Nie chcemy jednak śledzić zmian w każdym z tych plików, folder `obj/` zawiera pośrednie pliki binarne, które zupełnie nas nie interesują. Stwórzmy więc specjalny plik `.gitignore` zawierający reguły z wyrażeń regularnych, decydujące o tym które elementy naszego repozytorium git ma ignorować:

```
echo "obj/*" > .gitignore
```

W przyszłości można tam dodawać swoje własne reguły (np. `*.cpp~`, `*.hh~` itd.). Zauważmy, że teraz `git status` nie wypisuje folderu `obj/`.

Większość komend podanych w tej sekcji działa tylko wewnątrz gitowego repozytorium. Jeśli spróbujemy użyć ich gdzieś indziej otrzymamy komunikat `Fatal: not a git repository`

Jesteśmy już gotowi do stworzenia pierwszego commita. Wpiszmy:

```
git add .
```

Kropka na końcu polecenia jest istotna i oznacza, że do commita mają być dodane wszystkie pliki w bieżącym folderze i jego folderach podrzędnych. Wpisując teraz `git status` możemy podejrzeć które pliki wejdą w skład naszego commita. Jeśli znajduje się w nich coś niewłaściwego, to w tym momencie można to jeszcze w prosty sposób poprawić, usuwając niepożądane pliki z użyciem `git rm`.

Jeśli wszystko jest ok, wpisujemy:

```
git commit -m "Initial commit"
```

W cudzysłowach zawarty jest tytuł naszego commita. Zazwyczaj staramy się tam opisać zwięźle, czego dotyczy wprowadzana zmiana. Przyjęło się, że tytuły te mają formę poleceń, które zostały wykonane, a nie opisu wykonanej czynności. Napiszemy więc raczej `'Add displaying complex number'` zamiast `'I've implemented function for displaying complex number'`.

Jeśli wszystko poszło dobrze, to właśnie stworzyliśmy pierwszego commita. Możemy go zobaczyć, wpisując polecenie

```
git log
```

wyświetlające listę wszystkich commitów. Jednak, o ile nasza zmiana jest już zapisana lokalnie, to nie ma jej jeszcze w zdalnym repozytorium. Aby podzielić się swoimi postępami z resztą świata, trzeba jeszcze wpisać:

```
git push
```

I odpowiedni login i hasło. To wszystko. Kolejne commity tworzymy w ten sam sposób, podążając według schematu:

1. Wprowadź zmiany w kodzie
2. `git add .`
3. `git commit -m "nazwa commita"`
4. `git push`

Uwaga - pamiętanie o komendzie `git push` jest bardzo ważne. Bez niej prowadzący nie będzie mógł sprawdzić naszej pracy, mimo tego, że z naszej strony wszystko będzie wyglądać OK.

Przy pracy na różnych komputerach w tym samym repozytorium, zmiany wprowadzone do repozytorium online ściągniemy do lokalnego komendą `git pull`.

3 Postać repozytorium do kursu Podstaw Programowania

Proszę, aby na rzecz kursu Podstawy Programowania Państwa repozytorium (tutaj, na przykładzie `ljaniec`) wyglądało podobnie do poniższego drzewa folderów (z późniejszym analogicznym dodawaniem kolejnych zadań, programów, sprawozdań, itd.).

```
ljaniec
├── oswiadczeniePodpisane.txt
├── Z0
│   └── certyfikatBashDatacamp.pdf
├── Z1
│   ├── przygotowanie
│   │   ├── diagram.png
│   │   └── testyProponowane.pdf
│   ├── trojmianPoprawiony.c
│   ├── sprawozdanie.pdf
│   └── dodatkoweEmacs.txt
└── Z2
    ├── przygotowanie
    │   ├── algorytm.pdf
    │   ├── program1.c
    │   └── ...
    ├── main.c
    ├── sprawozdanie.pdf
    └── ...
```

I dalej, analogicznie jak powyżej - osobny folder na każde zadanie w pojedynczym repozytorium, w środku: folder na przygotowanie do zajęć, kod programu (potem folder projektu z `src`, `inc`, ...), sprawozdanie z opisanymi testami, różne dodatkowe rzeczy.

4 Postać repozytoriów do kursu Programowania Obiektowego

Na kursie Programowania Obiektowego do zadań będą zwykle przygotowane zalążki kodu, które będę starać przygotowywać się jako tzw. C++ project template do skopiowania na swoje konto. Będą tam skonfigurowane odpowiednie mechanizmy (CI, unit tests etc.), z którymi będę Państwa sukcesywnie zapoznawać.

Wymaga to jednak podejścia:

- `jedno zadanie == jedno repozytorium`

(inaczej niż na PProg). Do każdego z nich trzeba dodać prowadzącego jako Maintainera.

Jeśli ktoś nie będzie chciał wykorzystywać ww. wzorców projektu, to może przygotować swoje repozytorium do każdego z zadań, jednak należy trzymać się ogólnego schematu (folder `prj/` z `src/`, `inc/`, Makefile lub `CMakeLists.txt`, sprawozdanie z testami jednostkowymi itd.).

5 Podsumowanie

Przedstawiona wiedza powinna w pełni wystarczyć do skutecznego użytkowania Gita na zajęciach Podstaw Programowania lub (później) Programowania Obiektowego. Jeśli chodzi o ogólną wiedzę o tym narzędziu, to przedstawiono tu tylko wierzchołek góry lodowej. Przede wszystkim w instrukcji tej nie został poruszony temat *branch*-ów, które są używane bardzo powszechnie. Zainteresowanych i ambitnych zachęcamy do dalszego samodzielnego dokształcania się w tej kwestii, korzystając z udostępnianych już wcześniej źródeł, jak i dowolnych innych materiałów, których w sieci jest naprawdę sporo, np.:

- https://learn.datacamp.com/courses/introduction-to-git-for-data-science
- https://eu.udacity.com/course/how-to-use-git-and-github--ud775
- https://learngitbranching.js.org
- https://www.freecodecamp.org/news/git-for-professionals/
- https://www.biteinteractive.com/picturing-git-conceptions-and-misconceptions/
- https://www.youtube.com/watch?v=S9Do2p4PwtE - dzięki, Janek!

Powodzenia z programowaniem!