Narzędzie git.

Git to rozproszony system kontroli wersji, który umożliwia śledzenie zmian w plikach, tworzenie historii projektów i współpracę nad kodem. Jego działanie opiera się na repozytorium (repo), które przechowuje wszystkie wersje plików i śledzi każdą zmianę w sposób niezawodny i wydajny. Aby lepiej zrozumieć, jak działa Git, przeanalizujmy kluczowe elementy i koncepcje.

Dokumentacja? git-scm.com.

Podstawowe informacje o systemie git.

1. **Repozytorium (repozytorium Git)**

Repozytorium Git to folder, który zawiera projekt i wszystkie dane potrzebne do śledzenia jego historii. Git przechowuje informacje w **katalogu** `.git` (ukryty folder w głównym katalogu projektu). W repozytorium znajdują się:

- **Indeks (staging area)**: Tymczasowe miejsce, w którym przechowywane są zmiany, które mają zostać zatwierdzone (commity).
- **Gałęzie (branches)**: Oddzielne linie rozwoju projektu, które mogą być scalane.
- **Commity**: Zatwierdzone zestawy zmian, które tworzą historię wersji.

2. **Snapshoty, a nie różnice (Diffs)**

Jednym z kluczowych aspektów Git jest to, że w przeciwieństwie do innych systemów kontroli wersji, Git **nie śledzi różnic między plikami**. Zamiast tego, Git **tworzy tzw. snapshoty** całego katalogu projektu przy każdym zatwierdzeniu (commicie). Oznacza to, że za każdym razem, gdy zatwierdzisz zmiany, Git zapisuje kompletny obraz plików w danej chwili. Jeśli plik nie został zmieniony, Git nie zapisuje nowej kopii pliku – zamiast tego tworzy wskaźnik do poprzedniej wersji pliku.

3. **Indeks (Staging Area)**

Indeks, zwany również obszarem staging, to tymczasowe miejsce, w którym przechowywane są pliki, które mają być zatwierdzone. Przed wykonaniem commitu, musisz dodać zmiany do indeksu, aby Git wiedział, które pliki mają zostać uwzględnione.

4. **Commit**

Commit to podstawowy element w Git, który oznacza trwałe zapisanie zmian w repozytorium. Każdy commit zawiera:

- Zestaw plików i ich wersje (snapshot).
- Metadane, takie jak autor commitu i czas jego wykonania.
- Wiadomość opisu zmian (commit message).
- **Hash** (unikalny identyfikator), który identyfikuje commit.

Podczas commitu Git tworzy nie tylko nową wersję plików, ale także śledzi relacje między commitami, co umożliwia odtwarzanie historii projektu.

5. **Hash SHA-1**

Każdy commit jest identyfikowany przez **unikalny hash SHA-1**, który jest 40-znakowym ciągiem liczb i liter. Hash ten jest generowany na podstawie zawartości commitu (pliki, ich wersje, metadane). Ten mechanizm sprawia, że każde drobne odchylenie w commitach tworzy całkowicie inny hash, co czyni Git bardzo dokładnym i bezpiecznym w śledzeniu wersji.

6. **Gałęzie (Branches)**

Gałąź w Git to ruchomy wskaźnik na określony commit. Domyślnie, kiedy tworzysz repozytorium, powstaje główna gałąź o nazwie **main** (lub historycznie `master`). Możesz tworzyć nowe gałęzie, które umożliwiają pracę nad różnymi funkcjami równocześnie, bez wpływu na kod w głównej gałęzi, gdyż w momencie utworzenia nowej gałęzi tworzona jest ona niejako kopia gałęzi, na której nowy branch jest właśnie tworzony.

Tworzenie i praca z gałęziami pozwala na:

- Tworzenie odrębnych ścieżek rozwoju.
- Współpracę nad projektami bez konfliktów.
- Scalanie gałęzi w jeden główny strumień, gdy funkcje są gotowe.

7. **Scalanie (Merging)**

Kiedy pracujesz na różnych gałęziach, możesz scalić zmiany z jednej gałęzi do innej. Scalanie to proces, w którym zmiany z jednej gałęzi (np. nowa funkcjonalność) są integrowane z inną gałęzią (np. `main`).

- **Scalanie bez konfliktów**: Git automatycznie łączy zmiany, jeśli nie ma konfliktów.
- **Konflikty**: Jeśli te same linie kodu zostały zmienione w obu gałęziach, Git poprosi cię o ręczne rozwiązanie konfliktów.

8. **Repozytorium lokalne a zdalne**

Git jest systemem rozproszonym, co oznacza, że każda kopia repozytorium jest pełnym repozytorium, które zawiera całą historię projektu. Możesz pracować na lokalnym repozytorium na swoim komputerze, ale możesz również współpracować z innymi, korzystając ze zdalnych repozytoriów (np. na GitHubie).

9. **Historia commitów**

Git pozwala na przeglądanie pełnej historii commitów, co ułatwia śledzenie zmian w projekcie, identyfikowanie błędów oraz przywracanie poprzednich wersji kodu. Polecenie `git log` wyświetla historię commitów, a dzięki różnym opcjom (takim jak `--oneline`), można przeglądać ją w kompaktowej formie.

Jak działają wersje repozytorium w Git?

Każda zmiana wprowadzona w projekcie, po dodaniu do indeksu i zatwierdzeniu w formie commitu, staje się nową wersją repozytorium. Git zapisuje te wersje w formie snapshotów, z możliwością ich identyfikacji za pomocą unikalnych hashy.

0. Instalacja

Instalacja Gita zależy od systemu operacyjnego, na którym pracujesz. Poniżej znajdziesz instrukcje instalacji na różnych platformach.

0.1. **Instalacja Git na systemie Linux**

W większości dystrybucji Linux Git jest dostępny w domyślnych repozytoriach. Aby zainstalować Git, wystarczy użyć menedżera pakietów odpowiedniego dla twojej dystrybucji.

- **Ubuntu/Debian:**

```
sudo apt update
sudo apt install git
```

- **Fedora:**

```
sudo dnf install git
```

- **Arch Linux:**

```
sudo pacman -S git
```

- **Slackware:** W przypadku Slackware Git może być zainstalowany za pomocą narzędzi `pkgtool`. Możesz pobrać paczkę Gita z SlackBuilds.org i zainstalować ją ręcznie. Instrukcja instalacji wygląda tak:

  1. Pobierz skrypt instalacyjny ze strony SlackBuilds.org.
  2. Rozpakuj archiwum:

```
tar xvfz git.tar.gz
cd git
```

  3. Uruchom SlackBuild:

```
./git.SlackBuild
```

  4. Zainstaluj paczkę:

```
sudo installpkg /tmp/git-<wersja>.tgz
```

0.2. **Instalacja Git na systemie macOS**

Na macOS można zainstalować Git na kilka sposobów. Najłatwiej jest skorzystać z menedżera pakietów Homebrew.

1. Zainstaluj Homebrew (jeśli jeszcze nie masz go zainstalowanego):

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Zainstaluj Git:

```
brew install git
```

Git jest również dostępny jako część Xcode Command Line Tools. Można go zainstalować, uruchamiając poniższe polecenie, które poprowadzi cię przez proces instalacji:

```
xcode-select --install
```

0.3. **Instalacja Git na systemie Windows**

Na Windows najlepiej jest użyć instalatora Git dla Windows, który dostarcza również narzędzie Git Bash, umożliwiające pracę z Git w środowisku przypominającym terminal Linux.

1. Pobierz instalator z oficjalnej strony Git: https://git-scm.com.
2. Uruchom pobrany plik `.exe` i przejdź przez proces instalacji, akceptując domyślne opcje.

Podczas instalacji będziesz miał możliwość skonfigurowania kilku opcji, takich jak używanie Git Bash lub integracja z PowerShell. Wybierz opcje według swoich preferencji.

0.4. **Sprawdzenie instalacji**

Po zakończeniu instalacji (na dowolnym systemie) możesz sprawdzić, czy Git został zainstalowany poprawnie, wpisując poniższe polecenie w terminalu (lub Git Bash na Windows):

```
git --version
```

Jeśli Git został poprawnie zainstalowany, zobaczysz numer wersji, np. `git version 2.42.0`.

Teraz, gdy Git jest zainstalowany na twoim systemie, możesz rozpocząć pracę, konfigurując swoje dane użytkownika za pomocą polecenia `git config`, jak opisano w sekcji konfiguracji.

1. **Tworzenie nowe repozytorium**

Aby rozpocząć śledzenie projektu, należy utworzyć nowe repozytorium w istniejącym folderze.

```
git init
```

W folderze, w którym polecenie zostało uruchomione pojawi się ukryty folder o nazwie `.git`, w którym znajdują się informacje o naszym repozytorium. Zostanie również utworzona gałąź (ang. branch) `master`.

2. **Konfiguracja użytkownika Git**

Przy rozpoczęciu pracy z Git, należy skonfigurować swoje dane, takie jak imię i adres e-mail, które będą używane do sygnowania commitów.

```
git config --global user.name Twoje Imię
git config --global user.email twoj_email@example.com
```

W przypadku pracu wielu osób na tym samym koncie systemu operacyjnego należy skonfigurować te parametry jako local zamiast global jak poniżej

```
git config --local user.name akowalski
git config --local user.email akowalski@gmail.com
```

Opcja *global* lub *system* ustawia te parametry dla każdego repozytorium na danym koncie, natomiast *local* jedynie dla tego konkretnego. Należy wtedy pamiętać, że wciąż lokalnie dla konkretnego repozytorium możemy te wartości nadpisać.

Kolejną rzeczą, JEŻELI PRACUJEMY NA KOMPUTERACH W SALI, którą należy wykonać jest wyłączenie domyślnej integracji Visual Studio Code z windowsowym menedżerem poświadczeń, który będzie przechowywał dane logowania do serwisu GitHub innych użytkowników utrudniając pracę kolejnym, którzy będą chcieli wypychać zmiany do swojego zdalnego repozytorium. Możemy to zrobić poleceniem:

```
git config --system --unset credential.helper
```

Warto również sprawdzić czy nie ma takiego ustawienia w innych zakresach konfiguracji.

```
git config --global -l
git config --local -l
```

Jeżeli tak to tutaj również usuwamy ten parametr.

3. **Klonowanie istniejące repozytorium**

Aby pobrać kopię zdalnego repozytorium, które już istnieje, użyj polecenia `git clone`.

```
git clone <url_repozytorium>
```

4. **Dodawanie plików do repozytorium**

Jeżeli dodałeś nowe pliki do projektu lub zmieniłeś istniejące, musisz dodać je do tzw. indeksu, aby były śledzone przez Git.

```
git add plik.py
```

Powyższe polecenie dodaje plik `plik.py` do poczekalni (ang. stage area) i w Visual Studio Code będzie on oznaczony literą `A` (added czyli dodany).

Aby dodać wszystkie zmodyfikowane pliki używamy komendy:

```
git add .
```

5. **Automatyczne wyłączanie plików z dodwania do repozytorium**

Należy jednak pamiętać, że różne narzędzia dodają 'coś od siebie' i mogą po pierwsze pojawić się pliki konfiguracyjne, które raczej nie będą potrzebne kiedy będziemy chcieli udostępnić repozytorium dla innych użytkowników, którzy mogą chcieć korzystać z innego narzędzia niż nasze i będą musieli samodzielnie nieco 'posprzątać' to repozytorium. Dlatego zalecanym krokiem przed dodaniem czegokolwiek do repozytorium jest przygotowanie pliku `.gitignore` zawierającego reguły (wyrażenia regularne), które spowodują, że spełaniające je zasoby będą przez narzędzie Git ignorowane przy sledzeniu zmian. Jednak należy pamiętać, że w przypadku modyfikacji `.gitignore` powinniśmy usunąć śledzenie zasobów, najwygodniej po prostu:

```
# UWAGA: poniższe polecenie usunie plik z poczekalni, ale też i z dysku
git rm *
# polecenie z przełacznikiem --casched usunie zasoby tylko z poczekalni
git rm --cached *
```

Tworzymy plik `.gitignore` a w jego treści wpisujemy nazwy plików bądź folderów, które nie chcemy aby były dodawane do naszego repozytorium. Ważne, aby każda nazwa pliku/folderu znajdowała się w nowej linii. Pamiętajmy o dodawaniu rozszerzenia pliku na końcu jego nazwy. Jeżeli chcemy wyłączyć z dodawania wszystkie pliki danego rodzaju możemy posłużyć się prostymi wyrażeniami regularnymi, czyli np. jeżeli chcemy wyłączyć z dodawanai wszystkie pliki typu `.txt` powinniśmy wpisać do pliku `*.txt`...

... i ponownie:

```
git add .
```

aby Git dodał wszystko ignorując te zawarte w `.gitignore`.

6. **Zatwierdzanie zmian (commit)**

Po dodaniu plików do indeksu, zmiany mogą zostać zatwierdzone.

```
git commit -m "Opis wprowadzonych zmian"
```

Opcja `-m` pozwala na dodanie wiadomości do commitu, która opisuje wprowadzone zmiany.

Komenda ta powoduje stworzenie nowej migawki zasobów znajdujących się w folderze roboczym (czyli wszystkiego tego co jest aktualnie widoczne w eksplorerze projektu) czyli z poczekalni. Każdy commit ma swój unikalny hashcode.

7. **Śledzenie stanu repozytorium**

Aby sprawdzić, jakie pliki zostały zmodyfikowane, a które czekają na zatwierdzenie:

```
git status
```

8. **Przeglądanie historii zmian**

Aby zobaczyć historię commitów, możesz użyć polecenia:

```
git log
```

Aby wyświetlić log w bardziej kompaktowej formie:

```
git log --oneline
```

9. **Porównywanie zmian**

Aby zobaczyć różnice między plikami lokalnymi a zatwierdzonymi zmianami:

```
git diff
```

10. **Tworzenie gałęzi (branching)**

Gałęzie pozwalają na rozwijanie nowych funkcjonalności bez ingerencji w główną wersję projektu.

```

```

```
git branch <nazwa_gałęzi>
```

Aby przełączyć się na nową gałąź:

```
git checkout <nazwa_gałęzi>
```

Możesz również utworzyć i przełączyć się na gałąź w jednym kroku:

```
git checkout -b <nazwa_gałęzi>
```

11. **Scalanie gałęzi (merge)**

Kiedy skończysz pracować na gałęzi, możesz scalić jej zmiany z główną gałęzią (zwykle `master` lub `main`).

```
git checkout main
git merge <nazwa_gałęzi>
```

12. **Pobieranie zmian ze zdalnego repozytorium**

Aby pobrać zmiany z zdalnego repozytorium bez automatycznego scalania:

```
git fetch
```

13. **Pobieranie i scalanie zmian**

Aby pobrać zmiany i automatycznie scalić je z bieżącą gałęzią:

```
git pull
```

14. **Wysyłanie zmian do zdalnego repozytorium**

Aby wysłać lokalne zmiany do zdalnego repozytorium:

```
git push origin <nazwa_gałęzi>
```

15. **Resetowanie i cofanie zmian**

Czasami konieczne jest cofnięcie lub odrzucenie zmian. Git oferuje kilka mechanizmów:

- **git reset**: Przywraca repozytorium do wcześniejszego stanu. Możesz cofnąć zmiany w staging area lub w pełni odrzucić commit.

```
git reset --hard <commit_hash>
```

- **git revert**: Tworzy nowy commit, który odwraca zmiany wprowadzone przez poprzedni commit, bez usuwania jego historii.

```
git revert <commit_hash>
```

16. **Tagi (Tags)**

Tagi to specjalne wskaźniki używane do oznaczania ważnych commitów, takich jak wersje oprogramowania. Tagi są niezmienne i pomagają łatwo odnaleźć określone punkty w historii projektu.

```
git tag <nazwa_tagu>
```

Podpięcie lokalnego repozytorium do nowego zdalnego repozytorium

Krok 1

Tworzymy nowe repozytorium zdalne, w trakcie zajęć korzystamy z serwisu GitHub. Po jego utworzeniu wyświetlona zostanie strona z komendami, które powinny nam pomóc przy procesie synchronizacji. Komendy włączając pierwszy commit można pominąć, gdyż zostało to już zrobione w nieco inny sposób. Dla lokalnego repozytorium wskazujemy zdalne repozytorum, do którego będziemy chcieli kod wypychać.

```
git remote add origin https://github.com/user/repo.git
```

Powyższe polecenie jest właściwe jeżeli nie korzystamy z klucza SSH (lepsze rozwiązanie, jeżeli tylko my korzystamy z danego konta w systemie operacyjnym), gdyż nie wymaga każdorazowego podawania poświadczeń przy wypychaniu zmian. Dodajemy informację o nowym zdalnym repozytorium o aliasie origin i podanym url-u.

Jeżeli chcemy sprawdzić jaki jest adres zdalnego repozytorium to

```
git remote get-url origin
```

a jeżeli trzeba go zmienić

```
git remote set-url origin https://nowy.adres.repo
```

Krok 2

Polecenie

```
git push -u origin master
```

próbuje wysłać aktualną gałąź (i jej stan z wykonanego ostatnio commitu) do zdalnego repozytorium z adresu przypisanego do aliasu (origin) do zdalnej gałęzi master. Tutaj może się pojawić kilka problemów (błędów), które mogą tę operację uniemożliwić. Jeżeli otrzymamy komunikat, mówiący, że wypchanie nie jest możliwe, bo użytkownik o nazwie xxx nie posiada uprawnień oznacza to, że w `menedżerze poświadczeń` są zapisane poświadczenia do serwisu GitHub innego użytkownika (mogą pochodzić z innego narzędzia). Należy usunąć wszystkie poświadczenia, które odwołują się do serwisu GitHub i ponowić próbę wypchania zmian. Można ponowić operację wypchania zmian.

Może się tutaj również pojawić problem jeżeli w zdalnym repozytorium znajdują się jakieś zasoby, których lokalnie nie ma (np. dodany domyślny plik `readme.md`). Wtedy możemy wykonać polecenie push z dodatkowym parametrem, wymuszającym wypchanie (force):

```
git push -f origin master
```

Pobranie repozytorium w inne miejsce/komputer

Jeżeli chemy kontynuować pracę z zawartością repozytorium na innym komputerze, narzędziu to wystarczy przejść do folderu, w którym chcemy umieścić repozytorium, np. folder z innymi projektami i wykonać polecenie:

```
git clone http://link.do.repo
```

To polecenie **UTWORZY** nowy folder o nazwie takiej jak nazwa zdalnego repozytorium i umieści tam już informacje o jego stanie (folder .git) oraz informacji o remote (ten sam, z którego został sklonowany). Teraz wystarczy skonfigurować `user.name` oraz `user.email` w przestrzeni `--local`, jeżeli nie ma poprawnych ustawień globalnych (pracujemy na tym samym koncie co inni użytkownicy) i można dalej pracować, zatwierdzać zmiany i wypychać je ponownie do zdalnego repozytorium.