**Git - kurs**

# **Dodawanie pliku na githuba:**

Link - <https://www.youtube.com/watch?v=eDEccFYaBBw>

Przechodzimy do lokalizacji, gdzie znajduje się interesujący nasz folder (nie możemy uruchomić komendy Git Bash Here na poziomie pliku) 🡪 prawy przycisk myszy + Git Bash Here, otwiera nam się terminal

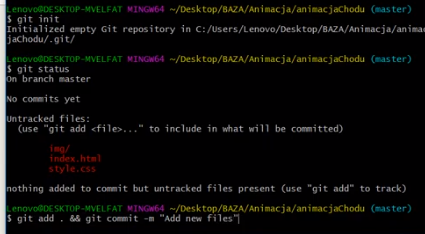
Jeśli pierwszy raz: ustalamy wstępną konfigurację z nazwą użytkownika i jego e-mailem



Potem: git init

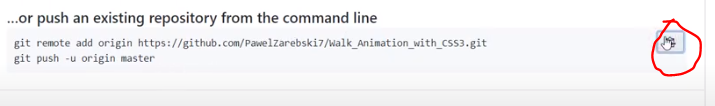
git status

git add . && git commit -m “Add new files”



Następnie Github - tworzymy nowe repozytorium

Po utworzeniu pojawia nam się kod, który następnie wklejamy do terminala:



Enter 🡪 rozpoczyna się przesyłanie plików na serwer…

Sprawdzamy w repozytorium, czy pliki się pojawiły

# **Odcinek 1**

Link - <https://www.youtube.com/watch?v=D6EI7EbEN4Q&list=PLjHmWifVUNMKIGHmaGPVqSD-L6i1Zw-MH&index=1>

System kontroli wersji - program zapisujący zmiany zachodzące w plikach (wersje) na przestrzeni czasu. W ten sposób tworzone są wersje tych plików. Dzięki temu możemy przejrzeć ich historię i w razie potrzeby przywrócić (kopie zapasowe). Wszystkie te informacje zapisywane są w tzw. repozytorium projektu.

**Rola systemów kontroli wersji:**

1. przegląd historii zmian wraz z informacją, kto i kiedy je wprowadził
2. przywrócenie dowolnej wersji pliku lub nawet całego projektu
3. praca zespołowa, poprzez wykorzystanie **zdalnych repozytoriów** (w takich serwisach, jak na przykład GitLab, Github, BitBucket)

**GIT -** najpopularniejszy system kontroli wersji, najważniejsze jego cechy to decentralizacja, prosta obsługa i szybkość działania.

Zarządzanie kodem źródłowym

**Podstawowe funkcje:**

1. Historia zmian - GIT umożliwia dostęp do historii zapisanych zmian w plikach projektu. W każdej chwili można podejrzeć lub przywrócić dowolną zmianę.
2. Rozgałęzianie kodu (branching) - jedną z najważniejszych funkcji GITa są gałęzie (branche). Umożliwiają rozwój nowych funkcji aplikacji niezależnie od siebie.
3. Przechowywanie kodu w chmurze (remote repository) - GIT umożliwia pracę ze zdalnym repozytorium, które może pełnić rolę kopii zapasowej projektu oraz stanowi podstawę pracy zespołowej.

*Każdy programista posiada własne lokalne repozytorium.*

1. Równoległa praca - GIT doskonale sprawdza się przy pracy zespołowej. Nie tylko zespoły, lecz nawet całe społeczności programistów mogą równocześnie pracować nad rozwojem projektu, nie tracąc kontroli nad źródłowym kodem aplikacji.
2. Możliwość pracy offline - nie ma żadnych przeszkód, by pracować nad repozytorium bez połączenia z internetem. Tak naprawdę nie istnieje ,,główne” repozytorium - a przynajmniej nie jest ono zdefiniowane przez GITa, lecz stanowi to kwestię umowną.
3. Społeczność - GIT w połączeniu z takimi serwisami, jak Github sprawia, że cała społeczność programistów może angażować się w rozwój oprogramowania, wymieniając doświadczenie i udostępniając efekty swojej pracy innym.

# **Odcinek 2 - Zapisywanie zmian**

Link - <https://www.youtube.com/watch?v=4bXuEv2R3W4&list=PLjHmWifVUNMKIGHmaGPVqSD-L6i1Zw-MH&index=2>

Lokalne repozytorium 🡪 **katalog roboczy** **przechowalnia** **repozytorium**

(working directory) (stage area) (.git folder)

**git add** 🡪 **git commit** 🡪

🡨 🡨 🡨 🡨 🡨 🡨 **git checkout**

Trzy stany plików:

1. Zmodyfikowany (modified) 🡪*git add*
2. Śledzony (staged) 🡪 *git commit*
3. Zatwierdzony (committed)

cd Desktop overment

overment git

overment git:(master) git status

.DS\_store

.idea/

**git init** - inicjowanie repozytorium

**git status** - podgląd aktualnego status

**git ignore**

**git add .** - dodawanie wszystkich plików do przechowalni

**git commit** - zapis zmian

**git commit -m ‘tresc\_komentarza’**

**clear** - czyszczenie terminala

**bilokacja** - plik posiada dwa stany jednocześnie

# **Odcinek 3 - Przestrzeń robocza i stage**

Link: <https://www.youtube.com/watch?v=gG80UsfMXsU&list=PLjHmWifVUNMKIGHmaGPVqSD-L6i1Zw-MH&index=3>

Aktywności podejmowane przed zapisaniem zmian

Chcemy pozbyć się zmian: git clean 🡪 nie działa od razu

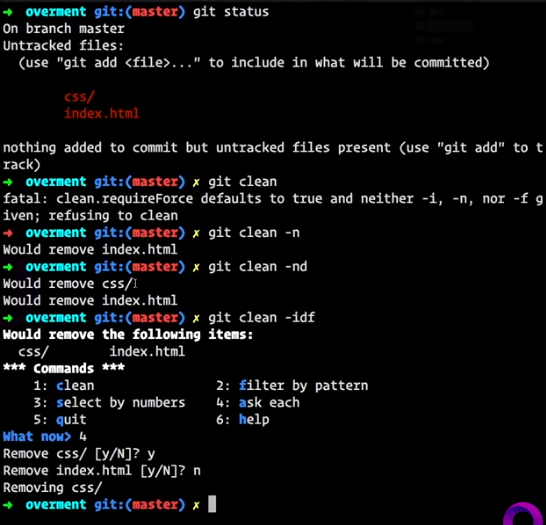
git status 🡪 git add 🡪 git status 🡪 git commit…

**git** **clean** - usuwanie nieśledzonych plików i katalogów (takich, które nigdy nie zostały dodane do repozytorium) 🡪 nieodwracalne!

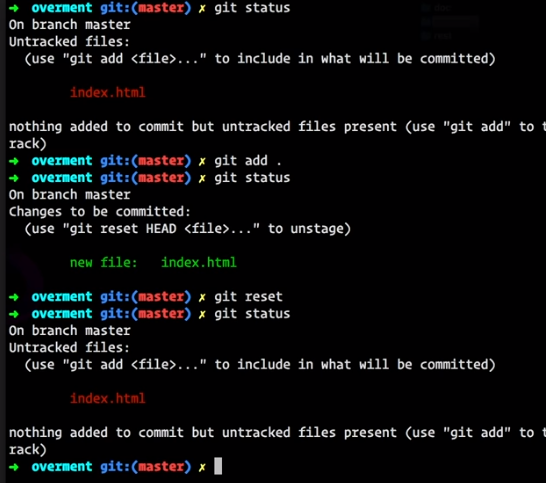
**git clean -n** - lista pików do usunięcia

**git clean -d** - kompletna lista elementów

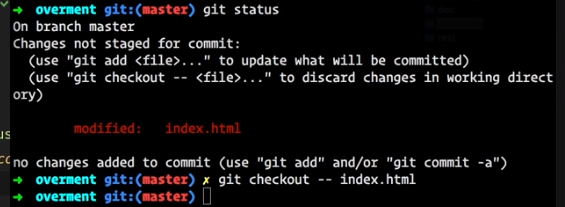
**git clean -idf** 🡪 w trybie interaktywnym (i), d - pliki i f-foldery



**git** **reset** 🡪 w podpowiedzi: odwrotność git add w tym przypadku



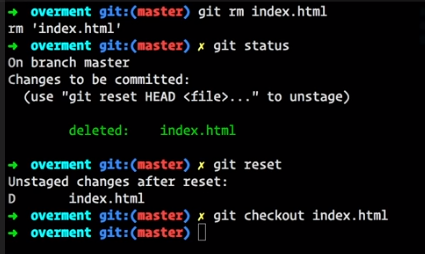
**git** **checkout** -- 🡪 usunięcie zmian (ma też inne zastosowania



**git rm** 🡪 usuwanie (następnie git status, git commit)

**git** **reset** 🡪 usunięcie zmian z kolejki oczekiwania

**git** **checkout** 🡪 tutaj: przywrócenie pliku



**git clean -x** (-xn???)🡪 uwzględnia pliki związane ze środowiskiem uruchomieniowym aplikacji (np. z folderu ignored)

**Podsumowanie:**

**git clean** - usuwa pliki, które nie zostały dodane do indeksu Gita.

**git reset** - przenosi pliki z kolejki oczekiwania (stage) do katalogu roboczego (working directory); pełni odwrotną rolę, niż git add

**git rm / git mv** - usuwa/przenosi pliki w repozytorium. Działają tak samo, jak rm i mv, równocześnie dodając zmiany na stage (kolejce oczekiwania)

**git checkout <path>** - przywraca stan pliku z indeksu gita; w praktyce przywraca wprowadzone zmiany. Komenda ta może być wykorzystana tylko wtedy, gdy plik istnieje już w indeksie