Zawartość

[1. Przygotowanie do rozpoczęcia pracy z repozytorium na serwerze github. 1](#_Toc529123869)

[1.1. Ogólny opis pracy z repozytorium. 1](#_Toc529123870)

[1.2. Przygotowanie i konfiguracja TortoiseGit. 2](#_Toc529123871)

[1.3. Podstawowe pojęcia przy pracy z repozytorium. 3](#_Toc529123872)

[1.3.1. Commit – czyli zapisanie zmian w lokalnym repozytorium 3](#_Toc529123873)

[1.3.2. Push – zapisanie zmian na serwerze 5](#_Toc529123874)

[1.3.3. Konflikt – próba zmiany istniejącego kodu repozytorium. 6](#_Toc529123875)

[1.3.4. Pull – pobranie zmian z serwera 9](#_Toc529123876)

[2. Sklonowanie pierwszego repozytorium 9](#_Toc529123877)

# Przygotowanie do rozpoczęcia pracy z repozytorium na serwerze github

Na potrzeby informacyjne przyjmujemy, że repozytorium znajduje się na publicznym serwerze gihthub dostępnym pod adresem <https://github.com>

Do pracy z repozytorium na serwerze gita potrzebne jest konto dostępowe. Github oferuje zarówno repozytoria odpłatne jak i darmowe, które są publiczne i dostępne dla wszystkich . Publiczne repozytoria dostępne są dla wszystkich tylko do odczytu. Uprawnienia do modyfikacji nadaje właściciel zakładanego repozytorium.

Po utworzeniu konta serwis github oferuje interfejs WWW, przez który można zarządzać swoimi repozytoriami lub udostępnionymi przez innych użytkowników do edycji.

Lokalnie do pracy z repozytorium można wykorzystać konsolę lub interfejs graficzny np. [TortoiseGit](https://tortoisegit.org/). Poniższa instrukcja będzie opisywała obsługę z wykorzystaniem tortoisegit. Wiele narzędzi programistycznych np. Visual Studio posiada wbudowany mechanizm obsługi repozytorium.

## Ogólny opis pracy z repozytorium

Praca z repozytorium polega tworzeniu kod a następnie wersjonowaniu kolejno dodawany zmian. Mechanizm repozytorium przechowuje kolejno tworzone wersje w sposób sterowalny przez użytkownika. Git pozwala na cofanie zmian w obydwu kierunkach, łączenie pracy wielu członków zespołu, oraz zapewnia dostęp do dokumentów z różnych miejsc w sieci.

W przypadku gita tworzone jest lokalne repozytorium(na dysku lokalnym) oraz drugie na serwerze. Taki mechanizm pozwala na dowolne lokalne modyfikacje oraz szybki powrót do pierwotnej wersji umieszczonej na serwerze.

Zaimplementowane funkcje zwalniają użytkownika z opracowywania systemów przechowywania kopii a w konsekwencji zagubienia się w kolejnych wersjach.

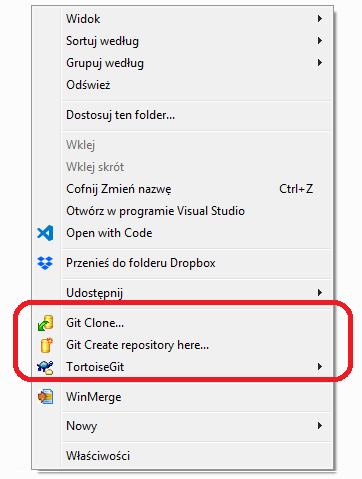
Przechowywanie kodu na serwerze jest również idealnym sposobem na jednoczesną pracę wielu osób na tych samych dokumentach. Dostępne narzędzia w znacznym stopniu ułatwiają łączenie powstałych fragmentów dokumentu w wersję ostateczną.

Do repozytorium na serwerze dodawane są zmiany w konkretnych liniach plików a nie całe pliki. Takie podejście znacznie zmniejsza ilość przechowywanych danych, co również znacznie przyspiesza poruszanie się po kolejnych wersjach dokumentu.

Repozytorium pozwala na tworzenie gałęzi (branch), które stanowią alternatywną wersję dla głównej. Funkcjonalność ta pozwala na tworzenie kodu tylko na potrzeby testów lub sprawdzenia innego rozwiązania jakiegoś problemu. Przy pracy grupowej można stworzyć osobną gałąź dla każdego a następnie łączenia sprawdzonego kodu. Pozwoli to łatwiej uniknąć przenoszenia błędów poza lokalne środowisko jednego użytkownika.

Mechanizm repozytorium pozwala na wykluczenie wybranych plików i katalogów z systemu wersjonowania. Takie zachowanie konfiguruje się dodając odpowiedni wpis w pliku .gitignore umieszczonym w katalogu repozytorium. Wykluczenia należy stosować do plików zawierających dane wrażliwe takie jak hasła, konfiguracje itp. Mechanizm gita ma pewne domyślne wartości które wyłączają część plików będących wynikiem np., kompilacji przez narzędzia programistyczne.

## Przygotowanie i konfiguracja TortoiseGit



Po pobraniu i zainstalowaniu tortoisegit(zarówno samego gita jak i interfejsu graficznego) w menu kontekstowym systemu powinny być widoczne dodatkowe opcje związane z obsługą repozytorium. Dostępne menu to:

* Git clone – tworzy repozytorium lokalnie na dysku na podstawie już istniejącego na serwerze
* Git create repository here – tworzy lokalne repozytorium
* TortoiseGit – pozostałe opcje

Katalogi zawierające repozytoria będą miały zmienione ikony. Nowe ikony będą informowały o stanie repozytorium, czy zostało zmienione czy nie.

## Podstawowe pojęcia przy pracy z repozytorium

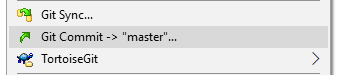
Aby dogłębnie poznać różne możliwości gita polecam profesjonalne tutoriale dostępne w sieci. Na potrzeby tej instrukcji zostaną wymienione i opisane podstawowe pojęcia, najczęściej wykorzystywane w pracy z repozytorium

* Commit – inaczej wczytanie kodu do lokalnego repozytorium
* Push – „zapchanie” zmian do repozytorium na serwerze na podstawie wcześniejszego commita
* Konflit – sytuacja kiedy linie kodu w docelowym repozytorium różnią się od tych, które próbujemy dodać operacją Push.
* Pull – pobranie zmian z serwera (jednocześnie dokunując merga)
* Merge – połączenie dwóch gałęzi tego samego repozytorium lub dołączenie zmian będących na serwerze do lokalnego repozytorium
* Branch – gałąź repozytorium będąca alternatywą dla właściwej wersji dokumentu
* Create branch – tworzy nową gałąź
* Switch/Checkout – zmiana brancha
* Stash save – zapisanie w celu tymczasowym dodanych zmian poza mechanizmem wersji
* Rebase – polecenie pozwalające połączyć kilka commitów w jeden
* Reverte change by this commit – cofa zmiany dokonane przez wskazany commit
* Reset „master” to this – cofa stan całego repozytorium lokalnego do wskazanego commita
* Show log – pokazuje okno z historią kolejnych zmian

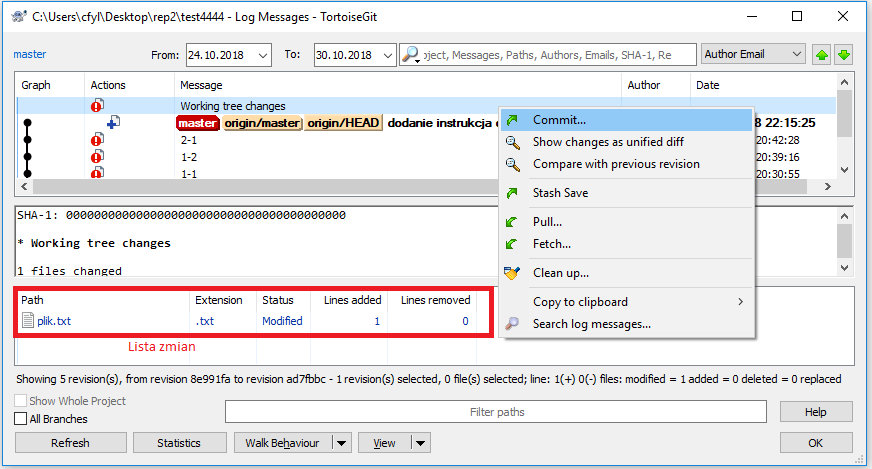
### Commit – czyli zapisanie zmian w lokalnym repozytorium

Mając pobrane lokalnie kody z repozytorium można rozpocząć. Praca z dokumentami może odbywać się z wykorzystaniem dowolnych narzędzi programistycznych (np., Visual Studio, NetBeans itp.). Oczywiście narzędzia takie lubią tworzyć dodatkowe pliki konfiguracyjne projektu, które można wyłączyć z systemu wersjonowania lub po prostu nie commitować ich do repozytorium.

Operację commit możemy wykonać w dowolnym momencie pracy wybierając „Git commit -> master” lub z wcześniej przedstawionego interfejsu Log Messages. Obydwa sposoby wywołują to samo okno jednak przy drugim sposobie można szybciej zobaczyć jakie zmiany zostały dokonane i mieć większą pewność, że czegoś nie przeoczyliśmy.



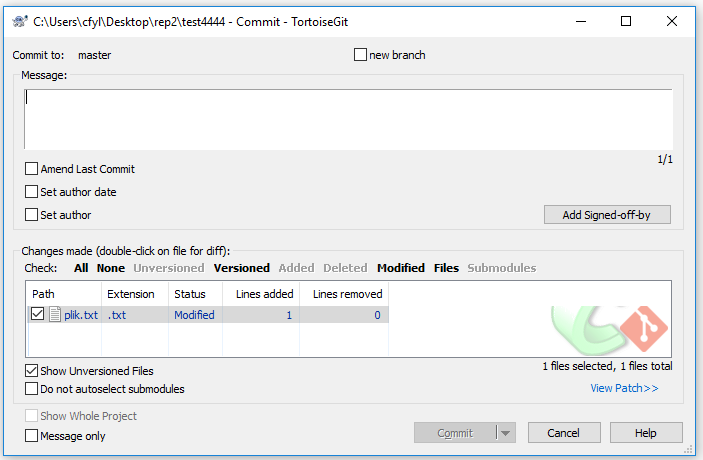
Metoda 1



Metoda 2

Jak widać w drugiej metodzie dodawania commita od razu widać w jakich plikach zostały dokonane zmiany.

Po wybraniu opcji commit otwiera się okno, w którym należy dodać opis wprowadzanej zmiany i zapisać przyciskiem „Commit”.



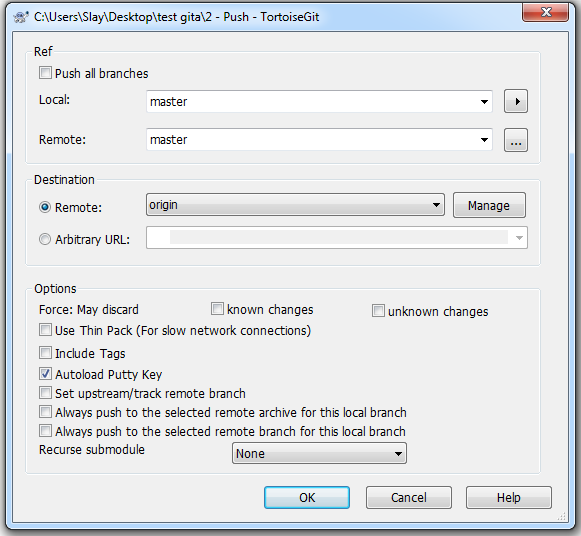
Tak dodany commit zapisuje zmiany jedynie w lokalnym repozytorium. Aby zapisać zmiany na serwerze potrzebna jest operacja „Push”.

### Push – zapisanie zmian na serwerze

Operację push wykonuje się w celu przeniesienia zmian dokonanych w lokalnym repozytorium na serwer, zazwyczaj zaraz po operacji commit.

Wykonując operację push należy się upewnić, czy rzeczywiście ma zostać wykonana. Po jej wykonaniu kod na serwerze ulegnie zmienia co będzie miało wpływ na innych użytkowników repozytorium. Operacji tej nie można cofnąć bez interakcji z innymi użytkownikami. Zmiany załadowane na serwer można wycofać, ale jeżeli zostały pobrane przez innych użytkowników to będą przez nich ponownie ładowane do czasu ich wycofania z repozytoriów pozostałych uzytkowników.

Wykonując operację push pojawia się następujące okno:



Przy standardowym scenariuszu należy zwyczajnie zatwierdzić przyciskiem ok. Dodatkowo w opcjach pojawia się „Force: May Giscard”. Jest to wymuszenie zmian na serwerze bez informacji zwrotnej o ewentualnych konfliktach z kodem innych użytkowników.

* „known changes” – zastępuje tylko kod, który występował również w repozytorium lokalnym ale uległ modyfikacji.
* „unknown changes” – zastępuje kod nieznany w lokalnym repozytorium, np. innych użytkowników.

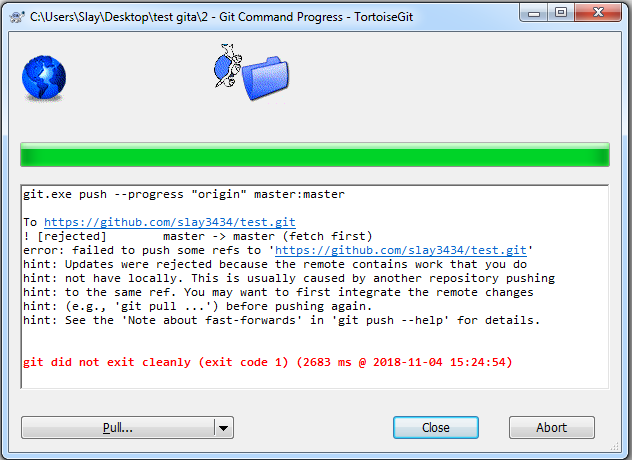
Wybierając opcję „Force” możemy usunąć permanentnie commity dodane na serwerze. W połączeniu z (dostępną w innym miejscu) operacją squash można w ten sposób połączyć wiele commitów w jeden w celu oczyszczenia repozytorium z nadmiarowej liczby operacji.

### Konflikt – próba zmiany istniejącego kodu repozytorium

Konflikt jest sytuacją kiedy próbujemy dokonać operację push, a inny użytkownik dodał swoje zmiany wcześniej i nie mamy ich lokalnie. Również kiedy zrobimy commita w lokalnym repozytorium a następnie operacją pull pobierzemy zmiany inne niż nasz własne.

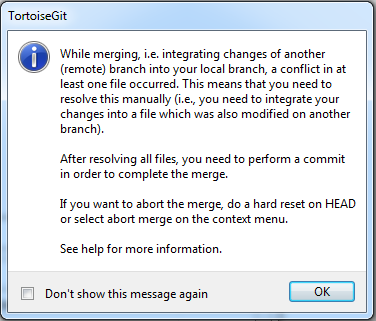
Taka funkcjonalność repozytorium zabezpiecza kod przed nadpisywaniem wzajemnie przez użytkowników.

Aby uniknąć konfliktu należy przed dodaniem swoich zmian do serwera wykonać operację pull. Oczywiście jest to niemożliwe w życiu codziennym ponieważ użytkownicy najczęściej pracują jednocześnie i nie jest możliwe żeby lokalny kod był stale na bieżąco. Z tego względu konflikty stają się nieuniknione dlatego git oferuje na to swoje rozwiązanie.

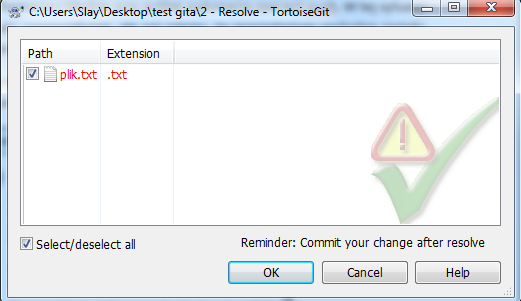


Na powyższym obrazku jest informacja o błędzie przy próbie wykonania operacji push. W tej sytuacji zmiany na serwerze nie zostały wprowadzone, ale już wiemy, że repozytorium centralne zostało zmienione przed naszymi zmianami.

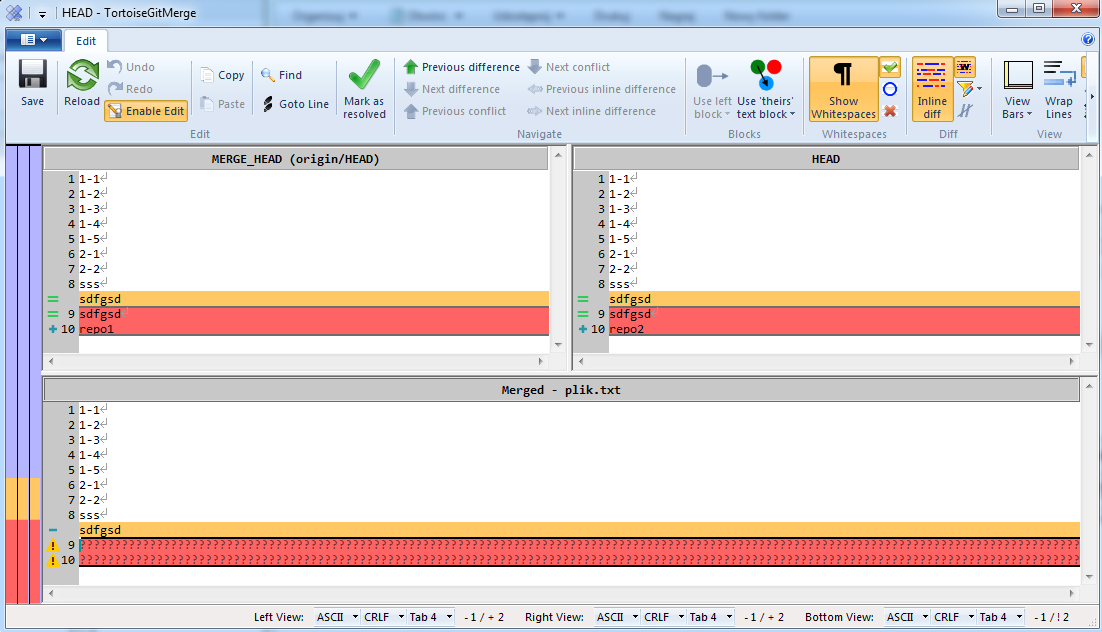
W tym momencie aby było możliwe dodanie lokalnych zmian na serwer należy rozwiązać powstały konflikt. W tym celu wykonujemy operację pull otrzymując poniższy komunikat:



W oknie wynikowym operacji pull pojawi się przycisk „Resolve”. Po Kliknięciu otrzymamy okno z listą plików będących w stanie konfliktu.



Klikając prawym przyciskiem myszy git daje szybkie możliwości rozwiązania konfliktu poprzez zastosowanie kodu z serwera lub lokalnego. Trzecia opcja to Edycja konfliktu za pomocą edytora tortoisa.



Po lewej stronie znajduje się kod z serwera, po prawej nasz lokalny kod. Na dolnym panelu przedstawiony jest planowany efekt końcowy.

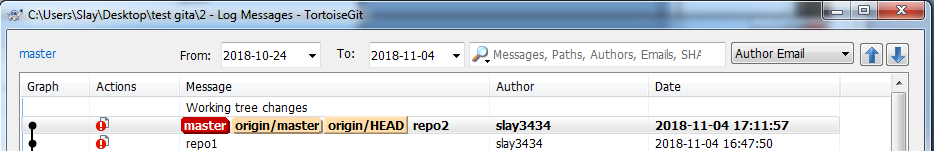
Dokonując poprawek w celu rozwiązania konfliktu trzeba mieć na uwadze, że kod dodany na serwerze przez innego użytkownika jest tam w jakimś celu i najlepiej go nie zmieniać. Należy również zwrócić uwagę na znaki końca linii, aby nie wystąpiły „sklejenia” linii kodu.

Po rozwiązaniu konfliktu powstanie merge ścieżki kodu na serwerze i lokalnej. Po takim rozwiązaniu otrzymamy jednak dodatkowy wpis w logu repozytorium;



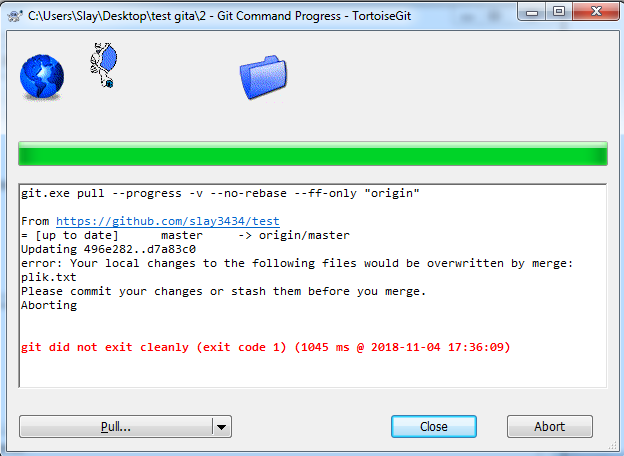
Innym rozwiązaniem na uniknięcie konfliktów jest skorzystanie z funkcji „stash”. Funkcja ta pozwala na zapisanie zmian dodanych do lokalnego repozytorium do zbioru tymczasowego.

Przed dodaniem commita robimy stasha własnych zmian. Po wybraniu funkcji stash z menu kontekstowego lub na panelu „log messages” lokalne zmiany będą przeniesione do zbioru tymczasowego a repozytorium wróci do pierwotnej wersji. W tym momencie możemy wywołać funkcje pull aby pobrać ewentualne zmiany innych użytkowników. Kiedy lokalne repozytorium będzie już w pełni zgodne z serwerem możemy zaaplikować nasze zmiany przechowywane w zbiorze tymczasowym. Wywołując funkcję „Stash pop” możemy otrzymać konflikt jeżeli nasz kod jest w tych samych liniach co kod innych użytkowników. W tym momencie rozwiązujemy konflikt według wcześniej opisanego scenariusza. Po rozwiązaniu konfliktu robimy commit, a następnie push. Dzięki takiemu podejściu otrzymujemy znaczniej przejrzystszy zapis w logu repozytorium bez dodatkowych zapisów. Wynik końcowy na obrazku poniżej.



### Pull – pobranie zmian z serwera

Operacja ta wymagana jest przy pracy w grupie aby zachować aktualność lokalnego kodu. Jeżeli zmodyfikowaliśmy lokalny kod pobranie zmian będzie niemożliwe i otrzymamy poniższy komunikat.



W tym momencie operacja pull nie dokonała żadnych zmian w lokalnym repozytorium. Możemy dodać commita a następnie pulla i tak może powstać konflikt. Innym rozwiązaniem jest skorzystanie z funkcji stash, w tym celu należy zapisać zmiany w zbiorze tymczasowy, pobrać zmiany z serwera następnie zaaplikować nasze zmiany i dodać commita bez żadnych konfliktów.

### Merge – łączenie

Pracując z repozytorium gita domyślnie posiadamy przynajmniej dwie gałęzie kodu, lokalną i centralną na serwerze. Dodatkowo mogą być tworzone inne gałęzie np. tymczasowe na przetestowanie jakiegoś rozwiązania. W takiej sytuacji nastąpi moment kiedy będzie trzeba przenieść wytworzony kod z jednej gałęzi do drugiej, taką operację określa się mergem. Oczywiście można przepisać kod jednak git daje sam rozwiązanie w postaci merga. Mechanizm zaimplementowany w gicie pilnuje aby nie wystąpiło nieświadome nadpisanie kodu i daje możliwości szybkiego przeniesienia różnić pomiędzy gałęziami.

Innym momentem wystąpienia tego zdarzenia może być operacja push lub pull. Jeżeli w trakcie, którejś z nich wystąpi konflikt, jego rozwiązaniem może być właśnie mege.

Merge jest bardzo przydatny w momencie kiedy na gałęzi tymczasowej powstaje kod, który może być użyty docelowo ale nie musi, lub kiedy tworzy się fragmenty kodu różne dla różnych odbiorców. W pierwszym przypadku, kiedy kod zostanie już ostatecznie dopracowany wystarczy dokonać merga do głównej gałęzi projektu. W drugim przypadku nie musimy myśleć o parametryzowaniu a zmiany wprowadzane w trzonie aplikacji mogą być szybko przenoszone na poszczególne gałęzie.

### Branch – gałęzie kodu

Branch jest niczym innym jak kolejnym repozytorium zbudowanym na podstawie głównej gałęzi lub innej. Jest on związany z głównym repozytorium ale może się zupełnie od niego różnić. Celem tworzenia gałęzi jest rozdzielenie kodu ,np. do testów, w celu tworzenia innych wersji kodu czy ze sposobu pracy zespołu developerskiego.

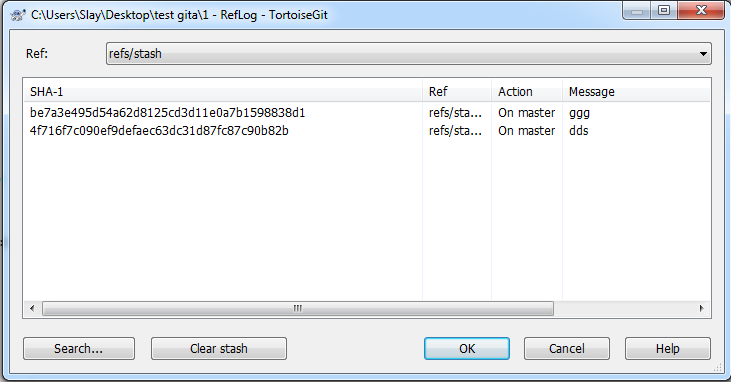
To dość proste rozwiązanie daje olbrzymie możliwości. W przypadku kiedy chcemy przetestować jakieś rozwiązanie ale wymaga ono wielu zmian, próba zapamiętania gdzie należy cofnąć zmiany może okazać się fatalna w skutkach. W tym wypadku utworzenie osobnej gałęzi pozwoli na dowolne modyfikacje nie mając wpływu na główny kod. Po zakończeniu pracy można połączyć gałęzie operacją merge.

Innym zastosowaniem jest utrzymanie „czystości” repozytorium. Jeżeli wytwarzamy kod dodając wiele commitów ze względu na skomplikowanie kodu w pewnym sensie zaśmiecamy repozytorium z punktu widzenia innych użytkowników. Dlatego warto w tym momencie trzymać swojego brancha, a do brancha głównego dodawać połączone commity (tzw. Squash) w postaci jednego commita.

### Stash – zbiór tymczasowy

Funkcjonalność pozwalająca zachować zmiany w kodzie nie wpływając na repozytorium. W momencie wykonania zdarzenia stash zmiany zostają zapisane w zbiorze tymczasowym i cofnięte w bieżącym stanie repozytorium. Pozwala to na bezproblemowe wykonanie polecenia pull lub może zastąpić tworzenie tymczasowego brancha.

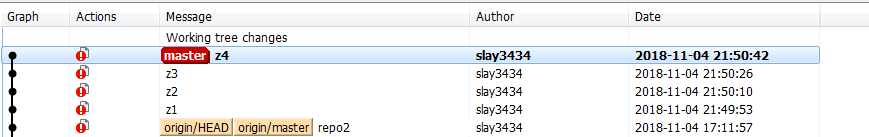
Takich zdarzeń możemy wykonać dowolną ilość i przeglądać je w postaci listy:



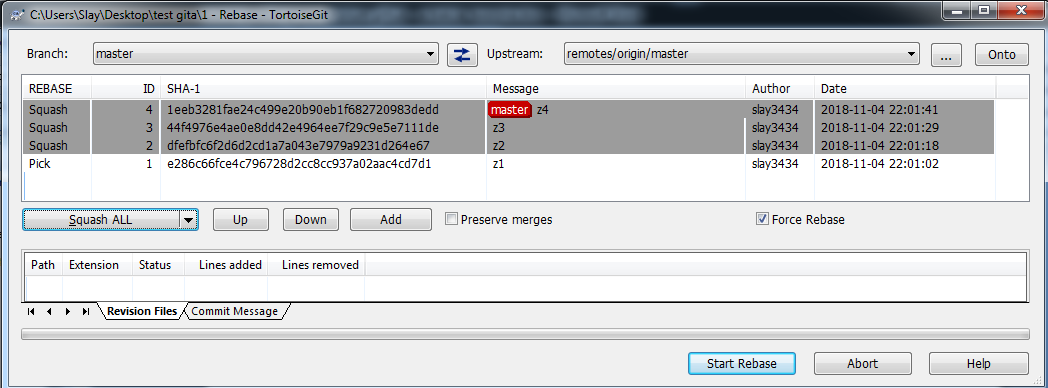
### Rebase – łączenie commitów

Przy dłużej pracy z gitem użytkownik zaczyna robić commit coraz częściej tylko dla bezpieczeństwa lub z innych przyczyn. Nie jest to niczym szkodliwy jednak prowadzi do rozrostu ilości wpisów w logu repozytorium, przez cos staje się on mało-czytelny. W tym celu warto wykonywać co jakiś czas czyszczenie repozytorium poprzez łączenie kilku commitów w jeden. Nie chodzi o łączenie wszystkiego w jeden commit tylko tych części które były wykonywane kolejno po sobie i dotyczyły konkretnych fragmentów projektu.

Operacja rebase pozwala połączyć kolejno występujące commity w jeden zaczynając od najstarszego. W ten sposób efekt nawet kilku dni pracy może ulec ściśnięciu do kilku linijek kodu.



Na powyższym obrazku widać 4 commity dodające po jednej linii w lokalnym repozytorium. Ponieważ nie są to kluczowe zmiany można połączyć je w jeden commit. Wybierając opcję rebase otrzymujemy okno, którym należy zaznaczyć opcję „ Force rebase” oraz „Squash All”.

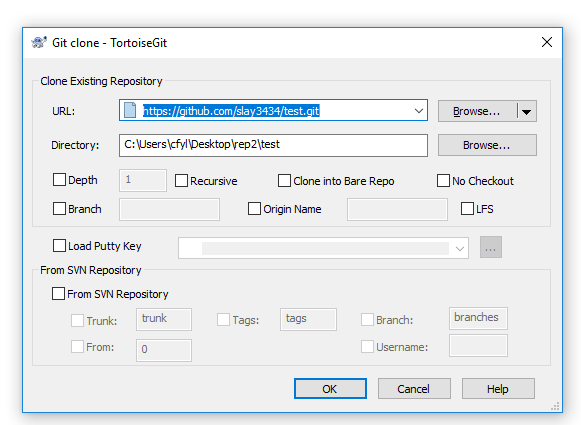


Wszystkie commity poza najstarszym zostały oznaczone atrybutem „Squash”. Po zatwierdzeniu przyciskiem „Start Rebase” otrzymamy jeden na bazie najstarszego commita. W zależności od tego czy kolejne commity modyfikowały ten sam kod mogą wystąpić konflikty, które trzeba rozwiązać.

# Sklonowanie pierwszego repozytorium

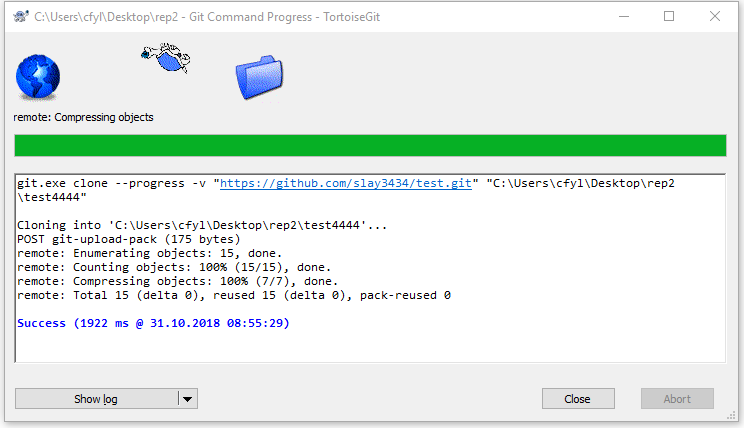
Mając konto na githubie oraz zainstalowanego lokalnie tortois wraz z gitem można rozpocząć pracę z repozytorium. Repozytorium na serwerze tworzy się w prosty sposób z wykorzystaniem interfejsu www. Dla każdego repozytorium powstaje **adres internetowy**, który służy do konfiguracji na stacjach klienckich. Na potrzeby instrukcji przyjmuję, że takie repozytorium jest już stworzone.

Wybierając z menu kontekstowego windowsa (prawy przycisk myszy) opcję „Git clone..” otrzymujemy okno konfiguracyjne dostępu do repozytorium na serwerze.



W polu „URL” należy wkleić adres internetowy naszego repozytorium, a w polu „directory” wybrać ścieżkę do katalogu w którym ma powstać lokalne repozytorium.

Po zatwierdzeniu program tortoise pobierze dokumenty zapisane na serwerze kończąc pracę komunikatem informacyjnym na temat pobranych danych.



We wskazanej lokalizacji powstanie katalog zawierający dane z serwera, oznaczony ikoną gita. 

Ikona z zielonym wskaźnikiem oznacza, że zawartość plików nie uległa modyfikacji względem lokalnego repozytorium. Nie oznacza to jednak, że lokalne pliki są w tej samej wersji co pliki na serwerze.

Czerwony wykrzyknik oznacza, różnicę pomiędzy lokalnym repozytorium a danymi w katalogu.

Wykrzyknik na żółtym tle oznacza „konflikty” czyli występowanie różnic pomiędzy lokalnym repozytorium a umieszczonym na serwerze. Prawdopodobną przyczyną najczęściej jest modyfikacja kodu na serwerze przez innego użytkownika. Przed dalszą pracą należy rozwiązać konflikty.

Do zarządzania sklonowanym repozytorium tortoise udostępnia kilka funkcji, podstawowym jest okno „Log Messages” pokazujące kolejne etapy modyfikacji repozytorium.

