



Systemy kontroli wersji w TwinCAT 3

**Wykorzystanie narzędzi Azure Repos Git oraz Team
Foundation Version Control TFVC**

Wersja dokumentacji 1.0

Aktualizacja: 03.08.2020

Kontakt: *support@beckhoff.pl*

Beckhoff Automation Sp. z o. o.

Spis treści

1	Wstęp	5
1.1	Czym są systemy kontroli wersji?	5
1.2	Czym jest Azure DevOps?	5
1.3	Wymagania	5
2	Azure DevOps	6
2.1	Utworzenie konta i organizacji	6
2.2	Tworzenie projektu	7
3	Tworzenie repozytorium Git	10
3.1	Podstawy nomenklatury systemu Git	10
3.2	Tworzenie repozytorium Git w TwinCATcie 3 (TcXaeShell)	10
3.3	Dodanie wyjątków do .gitignore	11
3.3.1	Użytkownik posiadający Git for Windows (zaawansowany)	11
3.3.2	Użytkownik początkujący (zalecane)	13
3.3.3	Rozwiązanie globalne dla wszystkich repozytoriów Git (zaawansowany)	14
3.4	Synchronizacja z Azure Git Repos	15
3.4.1	Użytkownik początkujący (zalecane)	17
3.4.2	Użytkownik posiadający Git for Windows (zaawansowany)	18
3.5	Klonowanie repozytorium na komputer	19
3.5.1	Azure Repos	19
3.5.2	Inne	21
4	Podstawowe operacje na repozytorium Git	22
4.1	Wprowadzanie i akceptowanie zmian	22
4.2	Stage & Commit	22
4.3	Branch	24
4.4	Merge	25
4.5	Push	27
4.6	Fetch & Pull	28
4.7	Sync	28
4.8	Stash	28
5	Tips & Trics	30

5.1	Porada ogólna.....	30
6	Organizacja pracy nad projektem przy użyciu Team Foundation Version Control w środowisku Azure na przykładzie TcXaeShell 35	
6.1	Czym jest Team Foundation Version Control?.....	35
6.1.1	Cechy charakterystyczne	35
6.2	Utworzenie projektu TFVC w środowisku Azure.....	36
6.2.1	Utworzenie połączenia z repozytorium w Visual studio	36
6.2.2	Wprowadzenie nowego rozwiązania na serwer Azure używając TFVC.....	39
6.2.3	Dodawanie istniejącego projektu do repozytorium TFVC AZURE	42
6.3	Aktualizacja projektu na platformie Azure przy wykorzystaniu TFVC.....	45
6.4	Pobranie projektu z platformy Azure przy użyciu TFVC.....	46
6.5	Tworzenie odgałęzień oraz łączenie gałęzi	47
6.5.1	Tworzenie odgałęzienia	47
6.5.2	Łączenie gałęzi	49
6.6	Rozwiązywanie kolizji plików	51
6.7	Sprawdzanie historii wersji poszczególnych plików.....	54
6.8	Porównywanie zmian w kodzie	56

Uwaga! Poniższy dokument zawiera przykładowe zastosowanie produktu oraz zbiór zaleceń i dobrych praktyk. Służy on wyłącznie celom szkoleniowym i wymaga szeregu dalszych modyfikacji przed zastosowaniem w rzeczywistej aplikacji. Autor dokumentu nie ponosi żadnej odpowiedzialności za niewłaściwe wykorzystanie produktu. Dany dokument w żadnym stopniu nie zastępuje dokumentacji technicznej dostępnej online na stronie <https://infosys.beckhoff.com> .

© Beckhoff Automation Sp. z o.o.

Wszystkie obrazy są chronione prawem autorskim. Wykorzystywanie i przekazywanie osobom trzecim jest niedozwolone.

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® i XTS® są zastrzeżonymi znakami towarowymi i licencjonowanymi przez Beckhoff Automation GmbH. Inne oznaczenia użyte w niniejszym dokumencie mogą być znakami towarowymi, których użycie przez osoby trzecie do własnych celów może naruszać prawa właścicieli.

Informacje przedstawione w tym dokumencie zawierają jedynie ogólne opisy lub cechy wydajności, które w przypadku rzeczywistego zastosowania nie zawsze mają zastosowanie zgodnie z opisem, lub które mogą ulec zmianie w wyniku dalszego rozwoju produktów. Obowiązek przedstawienia odpowiednich cech istnieje tylko wtedy, gdy zostanie to wyraźnie uzgodnione w warunkach umowy.

1 Wstęp

W instrukcji tej omówione zostały systemy kontroli wersji zintegrowane w narzędziu inżynierskim TwinCAT 3 [Wersja 3.1.4024.10]. TwinCAT 3 bazuje na środowisku programistycznym Visual Studio, dzięki czemu użytkownik zyskuje łatwy dostęp z poziomu programu do Azure DevOps, w tym w szczególności do grupy narzędzi występujących pod nazwą Azure Repos.

1.1 Czym są systemy kontroli wersji?

Kontrola wersji to system, który rejestruje zmiany w pliku lub zbiorze plików w czasie, dzięki czemu można później przywołać konkretne wersje. Pozwala on przywrócić wybrane pliki do poprzedniego stanu, przywrócić cały projekt do poprzedniego stanu, porównać zmiany w czasie, zobaczyć kto ostatnio dokonał modyfikacji, co mogło być przyczyną problemu i wiele innych. Używanie systemu kontroli wersji (ang. VCS) oznacza również, że jeśli pliki zostaną utracone, można je łatwo odzyskać. W tej instrukcji, używany będzie projekt zawierający program PLC wraz z kodem źródłowym choć w rzeczywistości systemów kontroli wersji użyć można prawie do każdego rodzaju plików na komputerze. Jest to niewątpliwie ich zaleta.

Do najczęściej spotykanych systemów kontroli wersji należą **Git**, **CVS**, **TFS/TFVC**, **Mercurial**, **Darcs**, **Apache**. Systemów kontroli wersji nie należy mylić z programami czy też serwisami takimi jak: **GitHub**, **Azure DevOps**, **Jenkins**. Są to twory bardziej rozbudowane, których narzędzia kontroli wersji są tylko częścią.

1.2 Czym jest Azure DevOps?

Azure DevOps to produkt firmy Microsoft "w chmurze", który służy nie tylko do tworzenia kodu, ale również posiada szeroki wachlarz narzędzi do zarządzania projektem. W ich skład wchodzi: kontrola wersji, raportowanie, zarządzanie wymaganiami, zarządzanie projektami, zautomatyzowane budowanie, testowanie i zarządzanie wydaniem. Początkujący użytkownicy TwinCATa powinni jednak skupić się nad **Azure Repos**, które jest zestawem narzędzi stricte do kontroli wersji.

1.3 Wymagania

- a) **Dostęp do Internetu** w celu połączenia się z serwerem <https://dev.azure.com/>
- b) Zainstalowany **TwinCAT 3 (TcXaeShell)**
- c) Może wystąpić konieczność instalacji **Git for Windows** z domeny <https://git-scm.com/download/win>

2 Azure DevOps

2.1 Utworzenie konta i organizacji

Na początku należy utworzyć konto na Azure DevOps. Procedura jest następująca:

- Wchodzimy na stronę <https://azure.microsoft.com/pl-pl/services/devops/>
- Wybieramy opcję **Rozpocznij bezpłatnie**
- Logujemy się** za pomocą konta Microsoft lub tworzymy nowe
- (Opcjonalnie) Uzupełniamy informacje dodatkowe
- Wpisujemy nazwę naszej nowej **organizacji** np. "BeckhoffAutomation"
- Przepisujemy znaki z rysunku
- Klikamy **Continue**

Azure DevOps

Almost done...

Name your Azure DevOps organization

dev.azure.com/ BeckhoffAutomation

We'll host your projects in

West Europe

Wpisz znaki, które widzisz

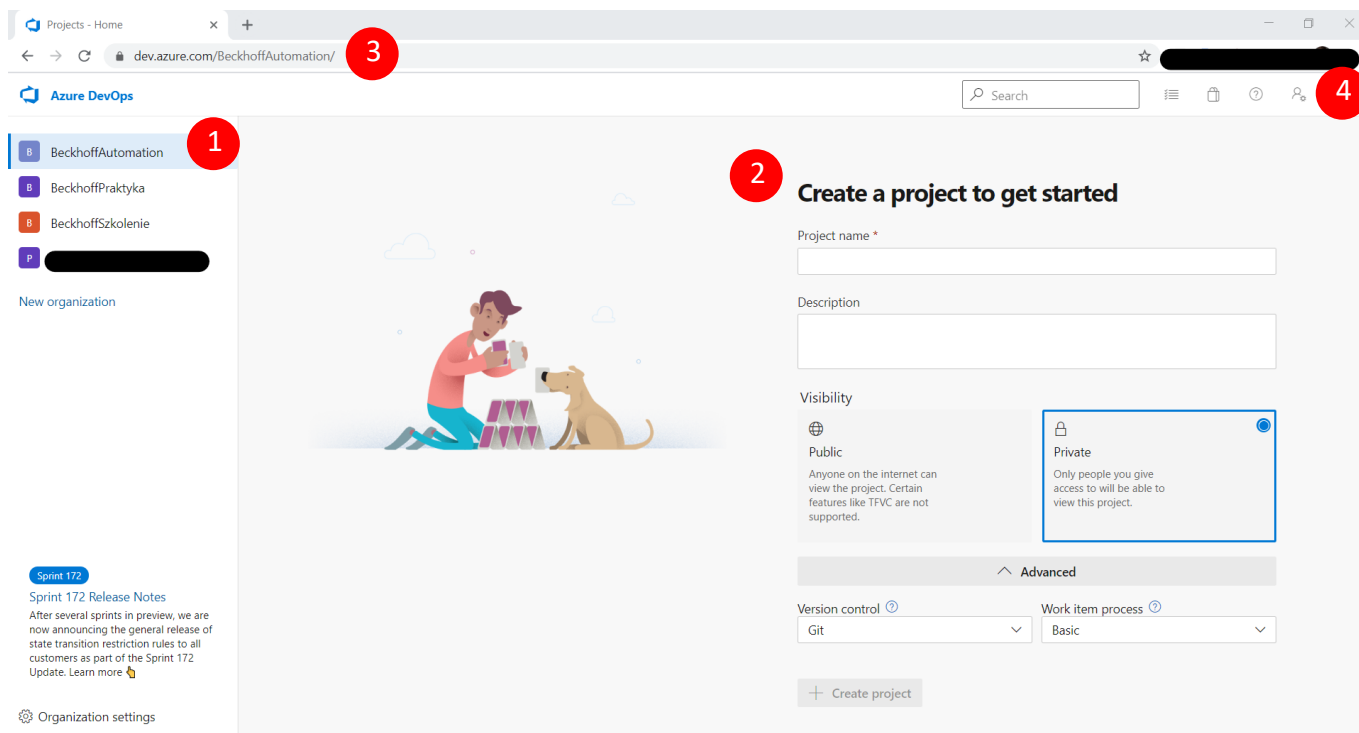
Nowy | Odsłuchaj

d3JW XGNV

d3JW XGNV

Continue

Po utworzeniu organizacji powinna się ona wyświetlić w kolumnie po lewej stronie ekranu [1]. Ponieważ nie mamy żadnego aktywnego projektu, program sugeruje utworzenie nowego [2]. Na tym etapie najważniejszy jest adres URL <https://dev.azure.com/BeckhoffAutomation/> [3], ponieważ jest to tym samym adres serwera naszej organizacji, z którym będziemy się łączyć z poziomu TwinCATa. W prawym górnym rogu widzimy ikonę konta Microsoft, na którym jesteśmy zalogowani [4].



2.2 Tworzenie projektu

- Wprowadź nazwę projektu – **Project name**
- (Zalecane) Dodaj opis - **Description**
- Określ widoczność projektu - **Visibility** – Wybieramy **Private**
 - Public – każdy posiadający link/adres naszego projektu ma do niego dostęp. Rozwiązanie przydatne przy projektach typu “open source”
 - Private – domyślne ustawienia chroniące nasz projekt. W celu udostępnienia jego zawartości oprócz wysłania zaproszenia należy nadać koledze/koleżance z zespołu uprawnienia, co zostanie opisane w dalszej części instrukcji
- Wybór systemu kontroli wersji – **Version control** - W tym punkcie omówiony został **Git**
- Work item process** – Pozostawiamy jako **Basic**
- Klikamy – **Create project**

Create a project to get started


Project name * **a**

Projekt Git Testowy ✓


Description **b**

Projekt zawierający repozytorium z systemem kontroli wersji - Git.

Visibility

 Public

Anyone on the internet can view the project. Certain features like TFVC are not supported.

 Private **c**

Only people you give access to will be able to view this project.

Advanced

d Version control **e**

Git Basic

f + Create project

Po utworzeniu projekt powinien pojawić się on w naszej organizacji. Po dwukrotnym naciśnięciu lewym przyciskiem myszy otwieramy projekt.



Po lewej stronie pod nazwą projektu wyświetlona zostaje lista narzędzi, z których najistotniejszym jest **Azure Repos**. Pozostała funkcjonalność nie jest tematem niniejszej instrukcji. Niemniej jednak, zachęcamy do zapoznania się z nią osoby zainteresowane projektem pod kątem zarządzania. Więcej informacji pod adresem: <https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops>

The screenshot shows the Azure DevOps web interface for a new repository named 'Projekt Git Testowy'. The left sidebar contains a navigation menu with options: Overview, Boards, Repos (selected), Files, Commits, Pushes, Branches, Tags, Pull requests, Pipelines, Test Plans, and Artifacts. The main content area displays the message 'Projekt Git Testowy is empty. Add some code!'. Below this, there are four sections: 'Clone to your computer' with buttons for HTTPS and SSH, a text input for the repository URL, a 'Generate Git Credentials' button, and a link to troubleshoot authentication; 'Push an existing repository from command line' with buttons for HTTPS and SSH, a text input for the command 'git remote add origin https://BeckhoffAutomation@dev.azure.com/BeckhoffAutomation/Projekt%20Git%20Testowy', and a copy icon; 'Import a repository' with an 'Import' button; and 'Initialize with a README or gitignore' with buttons for 'Initialize with README' and 'Initialize with gitignore'.

3 Tworzenie repozytorium Git

Jak już wspomniano Git jest systemem kontroli wersji, który w ostatnich latach stał się standardem. Przed przystąpieniem do dalszej części instrukcji zaleca się zaznajomienie z jego podstawami. Autoryzowane źródło informacji można znaleźć pod adresami:

- <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- <https://git-scm.com/docs>

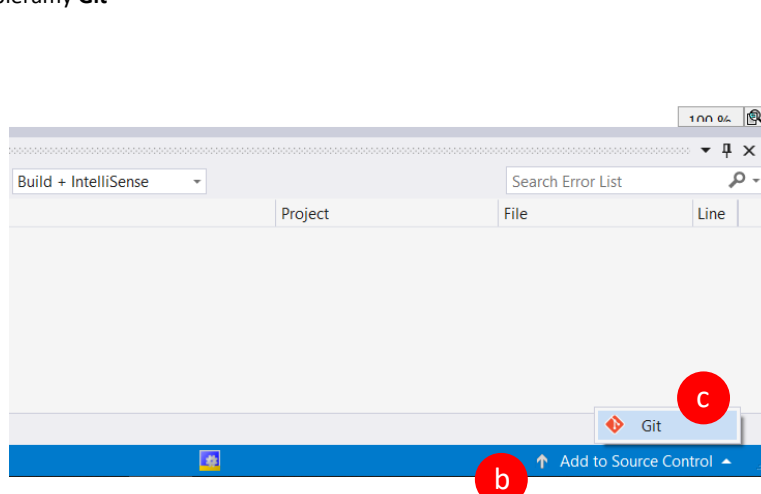
Dzięki znajomości Gita możliwe jest zarządzanie projektem lub zbiorem plików z poziomu Command Line. Umiejętność ta jest niezwykle cenna, ponieważ interfejs użytkownika nie ogranicza naszych możliwości. Operacje, które wykonywać będziemy w TwinCATcie są tylko częścią systemu, lecz wystarczającą dla standardowego użytkownika. Raz utworzone repozytorium w Gitcie udostępnić można za pośrednictwem wielu platform takich jak: Microsoft Azure DevOps, GitHub, Jenkins i innych.

3.1 Podstawy nomenklatury systemu Git

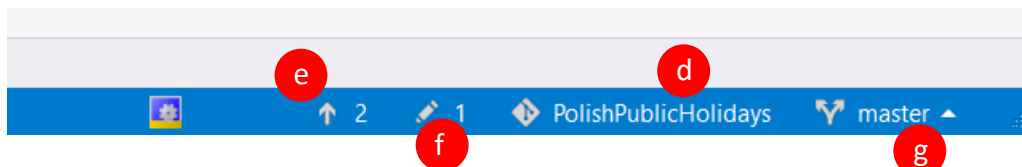
- **Distributed Version Control System** – zdytrybuowany system kontroli wersji; w praktyce sprowadza się do tego, że każdy użytkownik/członek zespołu, może mieć na swoim komputerze kopię repozytorium i rozwijać ją dowolnie bez dostępu do Internetu; (jest to podstawowa różnica względem TFVC, gdzie użytkownik na urządzeniu lokalnym ma zazwyczaj tylko jedną wersję każdego z plików; a nie całe repozytorium)
- **Repozytorium** – miejsce uporządkowanego przechowywania dokumentów, z których wszystkie przeznaczone są do udostępniania; innymi słowy zbiór plików projektu oraz zmienionych kopii (kolejnych wersji) które wchodzi w skład branchy
- **Clone** – klonowanie repozytorium; pobieranie kodu źródłowego z chmury do lokalnego katalogu roboczego
- **Sync** – synchronizowanie jednego repozytorium z innym poprzez porównanie różnic
- **Master** – główny branch repozytorium
- **Branch** – wersja projektu tworzona w celu wprowadzenia nowej funkcjonalności lub usunięcia błędów, tak aby nie uszkodzić głównej wersji, czyli master; w projekcie może występować wiele branchów.; to członkowie projektu decydują, które z nich zostaną połączone (**zmergowane**)
- **Push** – przesłanie zmian z lokalnego repozytorium do głównego, zazwyczaj w obrębie konkretnego branchu
- **Pull** – pobranie z serwera z wprowadzeniem zmian w lokalnym repozytorium (nadpisanie brancha)
- **Fetch** – pobranie z serwera bez automatycznego wprowadzania zmian w lokalnym repozytorium
- **Commit** – wprowadzenie i zatwierdzenie zmian w konkretnym branchu
- **Stage** – przygotowanie zmian do zatwierdzenia, czyli przygotowanie **Commitu**
- **Stash** – zachowanie zmian w schowku, bez uwzględnienia ich w **Commitcie**

3.2 Tworzenie repozytorium Git w TwinCATcie 3 (TcXaeShell)

- a) **Otwórz** solution z projektem
- b) W prawym dolnym rogu na pasku naciśnij **Add to Source Control**
- c) Następnie wybieramy **Git**



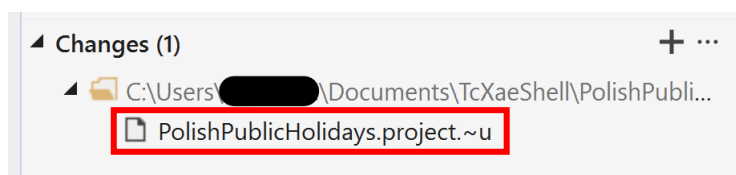
- d) Utworzone zostało repozytorium o nazwie tożsamej z nazwą Solution
- e) **Ikona strzałki** informuje nas o liczbie niezatwierdzonych Commitów
- f) **Ikona ołówka** informuje o liczbie zmian w projekcie, które nie należą do żadnego Commitu
- g) **Ikona branch** informuje nas na jakim branchu obecnie pracujemy, w tym miejscu możemy również dodawać nowe branchy oraz przełączać się pomiędzy istniejącymi; należy pamiętać, że aby przełączyć się na inny branch liczba przy ikonie ołówka musi być równa 0 (brak niez zaakceptowanych zmian w plikach projektu)



3.3 Dodanie wyjątków do .gitignore

Pomimo, iż od utworzenia repozytorium na komputerze nie dokonaliśmy żadnych zmian w plikach, przy ikonie ołówka widnieje cyfra 1. Po naciśnięciu LPM na ikonę ołówka w oknie **Team Explorer** zobaczyć możemy, iż zmiany dotyczą pliku z końcówką ".project.~u". Dzieje się tak ponieważ Visual Studio w trakcie pracy z projektem tworzy plik tymczasowy, w którym przechowywane są wszystkie zmiany dokonane na plikach, aż do ich zapisania. Dlatego **przed utworzeniem commitu pliki należy zapisać**. Sedno problemu natomiast leży w zasadzie działania Gita. System tworzy repozytorium ze wszystkich plików w danym folderze, a **pliki, które ma ignorować muszą być określone w pliku .gitignore**. Ponieważ tworzymy repozytorium za pomocą TwinCATa (Visual Studio), plik .gitignore zostaje stworzony automatycznie i nie zawiera wyjątku "*.project.~u". Są dwie możliwości, żeby to naprawić na poziomie projektu oraz jedna uniwersalna dla wszystkich projektów (wszystkie trzy przedstawiono poniżej).

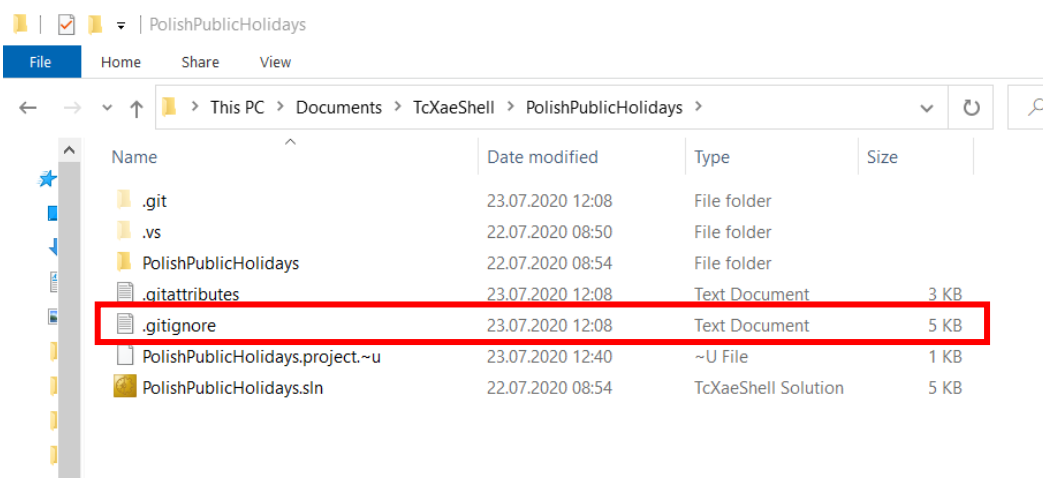
Tip Jeżeli nie widzisz okna Team Explorer: Górny pasek narzędzi->View->Team Explorer



3.3.1 Użytkownik posiadający Git for Windows (zaawansowany)

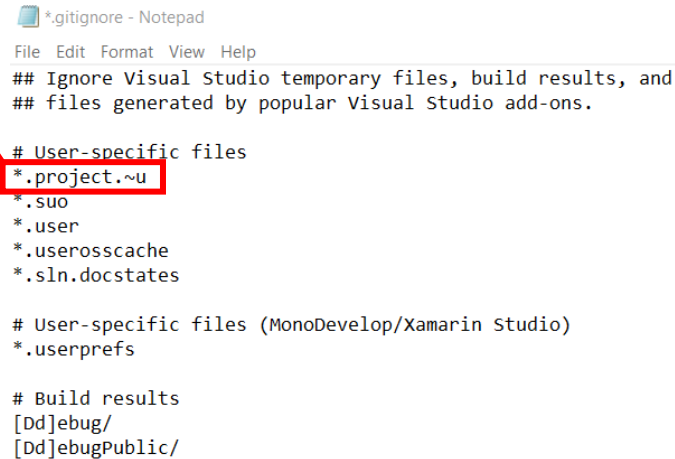
W celu usunięcia pliku z końcówką ".project.~u" z repozytorium należy:

- a) Otworzyć folder żądanego Solution
- b) Otworzyć plik **.gitignore** w dowolnym edytorze tekstu



- c) Dodać linijki kodu w dowolnym miejscu pliku tekstowego

```
# Linijki poprzedzone znakiem hash traktowane są jako komentarz
# Gwiazdka oznacza, że wszystkie pliki z daną końcówką będą ignorowane
*.project~u
```



```
*.gitignore - Notepad
File Edit Format View Help
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.project~u
*.suo
*.user
*.userosscache
*.sln.docstates

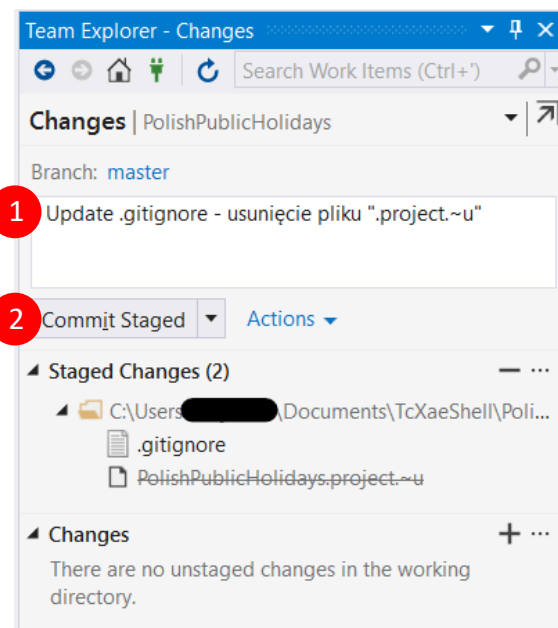
# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Build results
[Dd]ebug/
[Dd]ebugPublic/
```

- d) **Zapisujemy** plik .gitignore i zamykamy
- e) Otwieramy **Git Bash** lub podobne narzędzie np. konsolę obsługującą git (można skorzystać z wbudowanego w TwinCATa/Visual Studio **Package Manager Console** [View->Other Windows->Package Manager Console])
- f) Udajemy się do folderu z repozytorium
- g) Wykonujemy następujące komendy

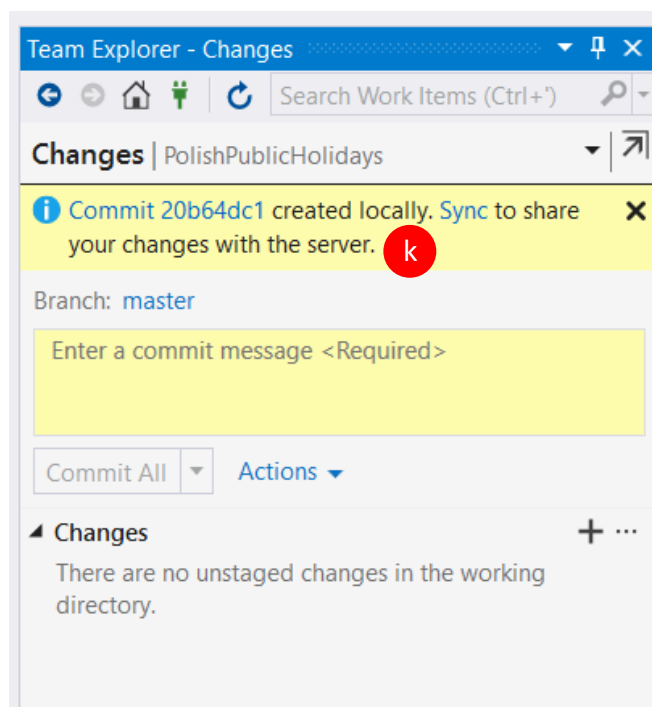
```
#usuwa wszystkie pliki z repozytorium, ale ich nie kasuje z folderu
git rm -rf --cached .
#dodaje na nowo wszystkie pliki w folderze - tym razem z uwzględnieniem nowego warunku w .gitignore
git add .
```

- h) Zamykamy Gita i otwieramy projekt
- i) Klikamy na ikonę ołówka w prawym dolnym rogu



- j) W oknie **Team Explorer** dodajemy **Commit Message** i tworzymy **Commit staged**

k) Commit został utworzony



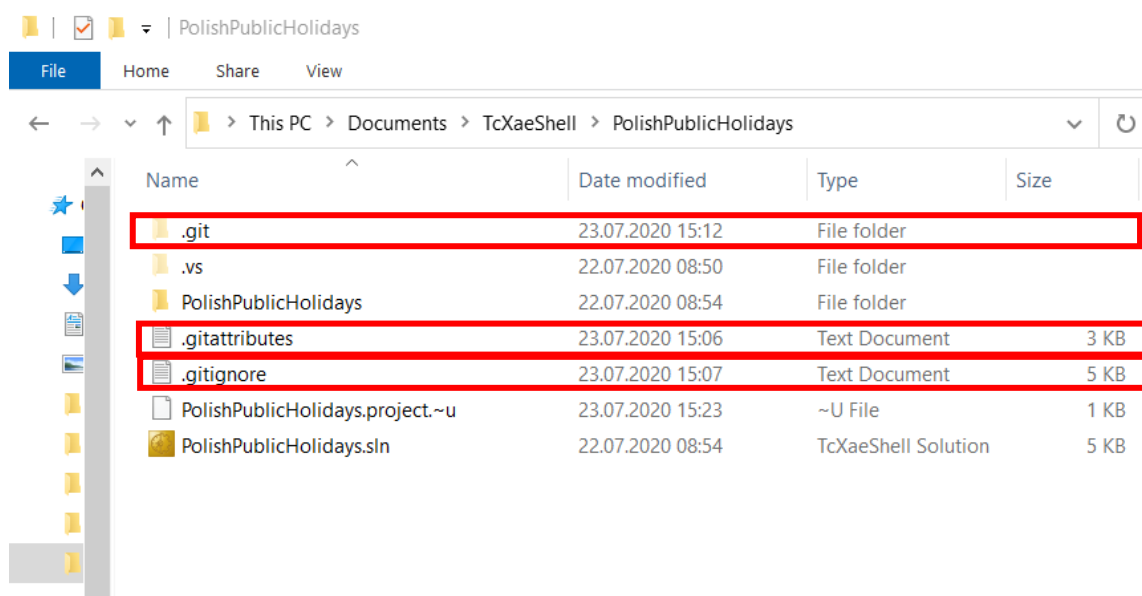
3.3.2 Użytkownik początkujący (zalecane)

Przedstawione poniżej rozwiązanie daje zamierzone efekty, trzeba jednak pamiętać dwie rzeczy:

- 1) Czynności należy wykonać **przed** utworzeniem repozytorium
- 2) W razie potrzeby należy ręcznie wydłużyć listę plików w .gitignore

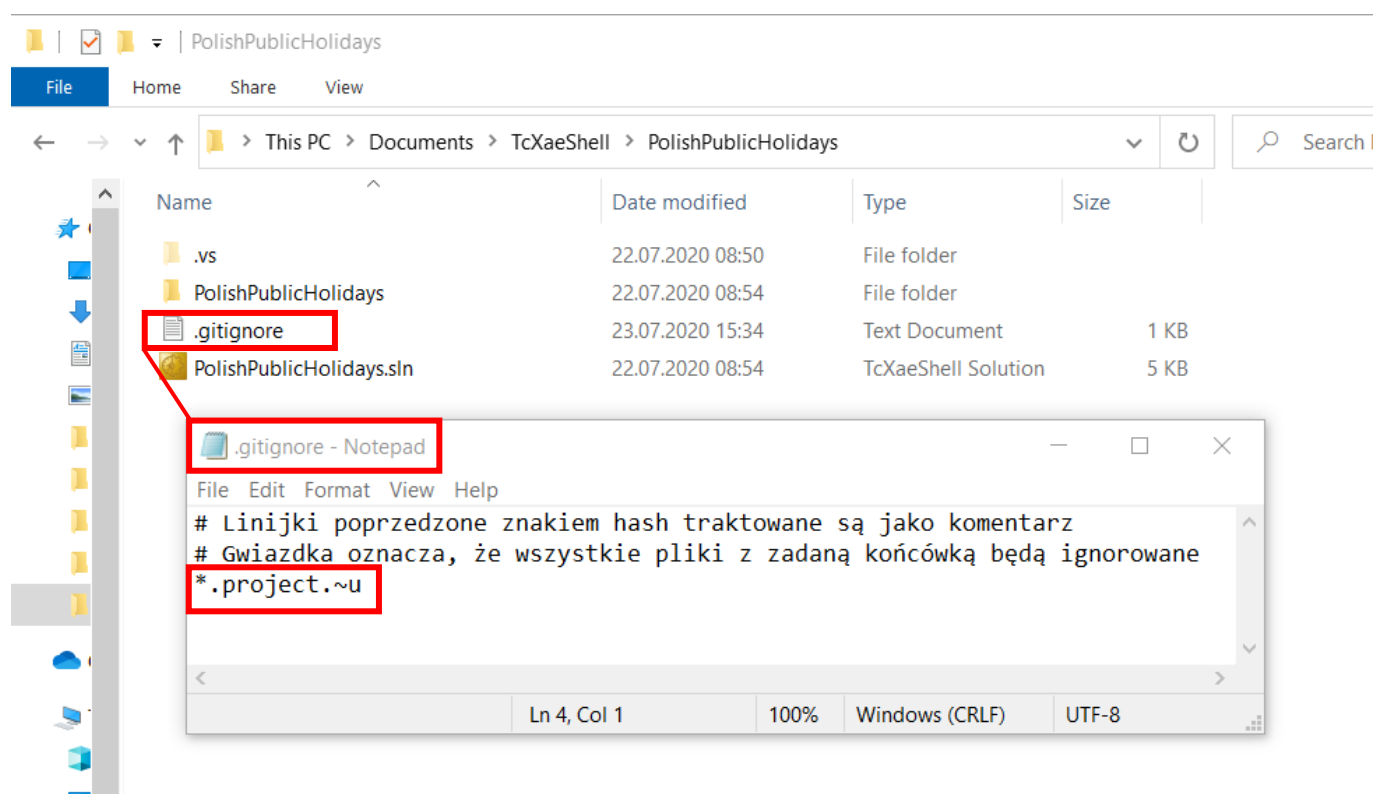
Jeżeli podążając za niniejszą instrukcją utworzono już repozytorium w [punkcie](#) oto sposób jak je usunąć:

- a) Zamknij projekt w TwinCATcie
- b) Otwórz folder projektu
- c) Usuń następujące pliki i foldery

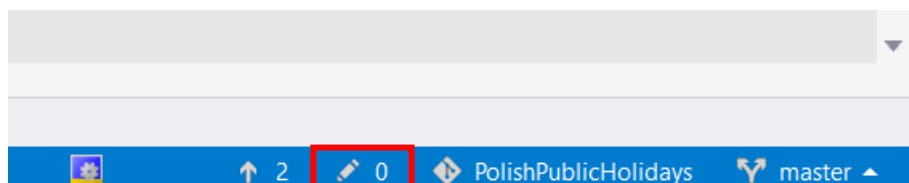


Następnie, w folderze projektu (Solution) należy utworzyć własny plik .gitignore i uzupełnić o poniższą linię. Jest to rozwiązanie ignorujące tylko konkretny rodzaj pliku. Można uzupełnić ten plik o dowolne rozszerzenia. Listy najczęściej stosowanych plików .gitignore są powszechnie dostępne w Internecie i tworzone pod konkretne rozwiązania. Można dla przykładu wyszukać zalecaną listę dla programu Visual Studio dodać jej zawartość do własnego pliku .gitignore. Tak przygotowany plik z powodzeniem można stosować w przyszłości do innych projektów.

```
# Linijki poprzedzone znakiem hash traktowane są jako komentarz  
# Gwiazdka oznacza, że wszystkie pliki z zadaną końcówką będą ignorowane  
*.project.~u
```



Zapisujemy plik .gitignore i otwieramy projekt w TwincATcie. Postępujemy analogicznie jak w [punkcie](#), tym razem z innym rezultatem.



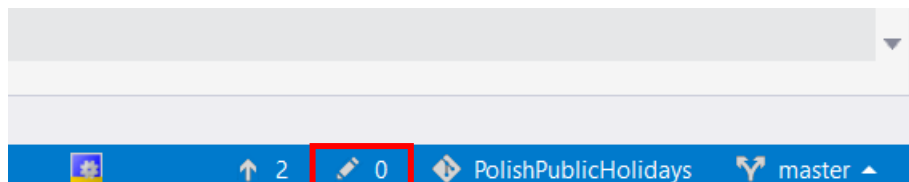
3.3.3 Rozwiązanie globalne dla wszystkich repozytoriów Git (zaawansowany)

Użytkownikom zaznajomionym z obsługą Gita rekomendujemy zdefiniowanie globalnej listy plików .gitignore i umieszczenia pozycji “*.project.~u” w jej wnętrzu. Należy stworzyć plik .gitignore np. folderze domowym (ang. home directory), a następnie wskazać na niego w Gitcie następującą komendą.

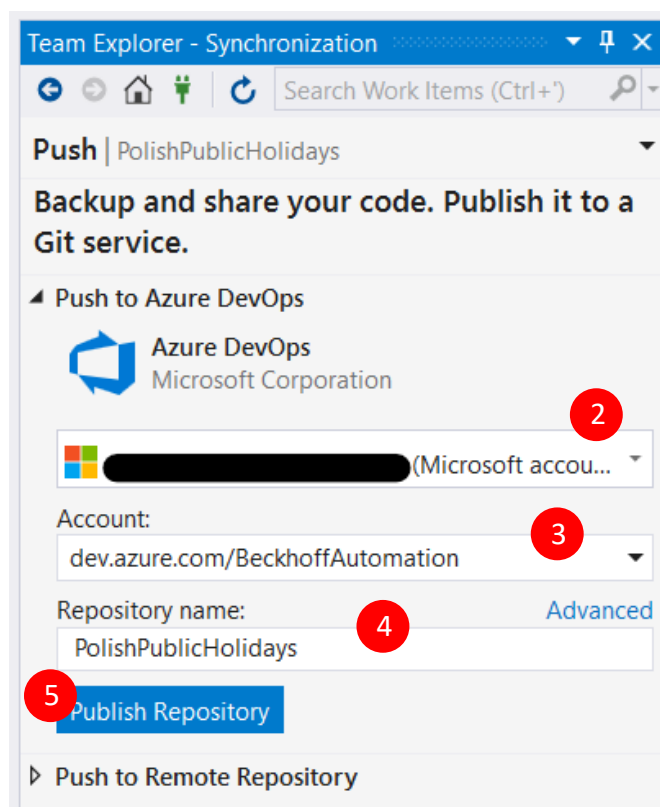
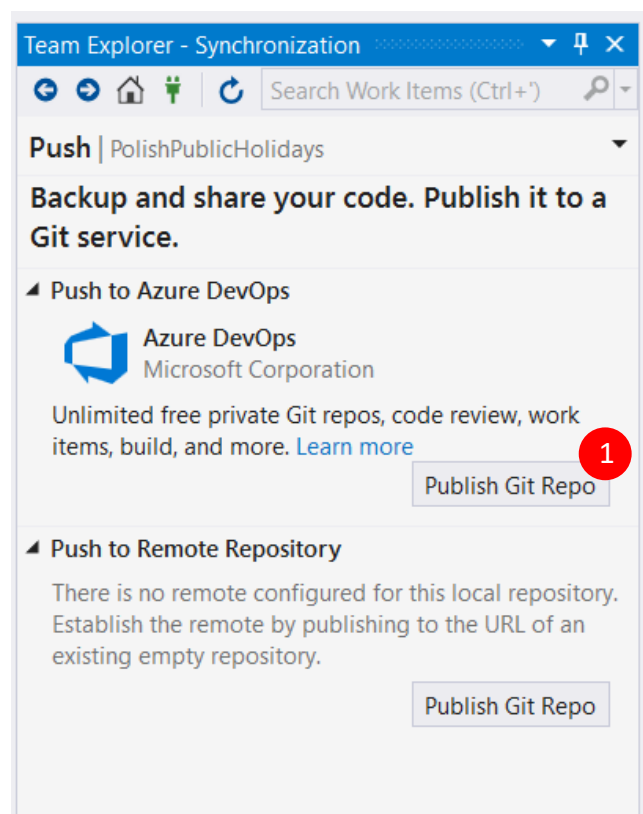
```
git config --global core.excludesfile %USERPROFILE%.gitignore
```

3.4 Synchronizacja z Azure Git Repos

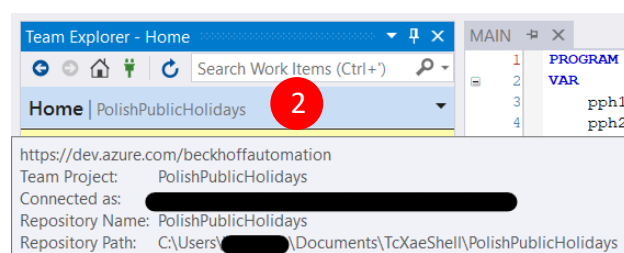
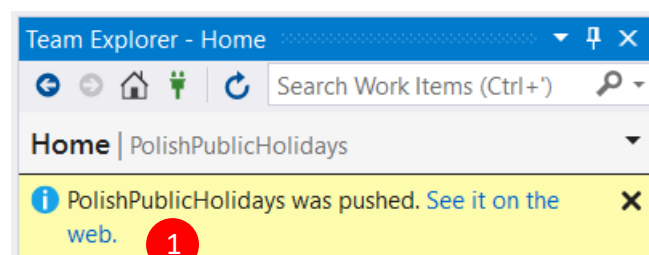
Jeżeli przy ikonie ołówka mamy 0, możemy przejść do przesłania repozytorium na serwer. W przypadku repozytorium Git mamy wiele serwisów i serwerów do wyboru (np. GitHub), rekomendujemy korzystanie z Azure Repos.



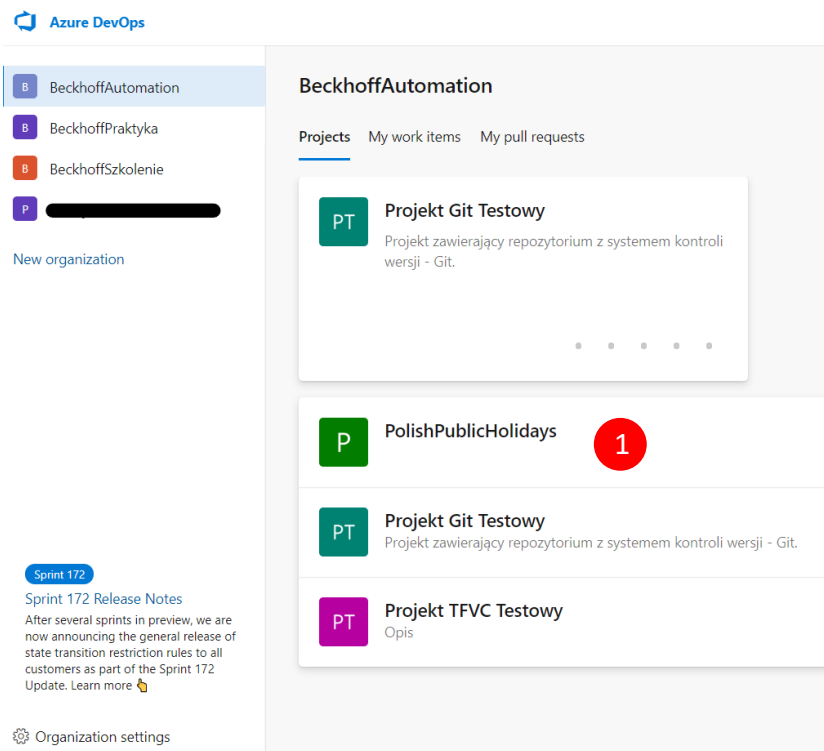
Aby przesłać repozytorium należy w zakładce **Team Explorer** nacisnąć **Publish Git Repo**. Jeżeli zakładka Team Explorer wygląda inaczej niż ta poniżej, naciśnij ikonę strzałki w prawym dolnym rogu (obok ikony ołówka). Następnie wybieramy konto Microsoft i logujemy się do niego. Z zakładki **Account** wybieramy adres serwera/organizacji, na który chcemy przesłać nasze repozytorium. Pole **Repository name** możemy pozostawić domyślnie wypełnione lub zmienić nazwę. Na koniec klikamy **Publish Repository**.



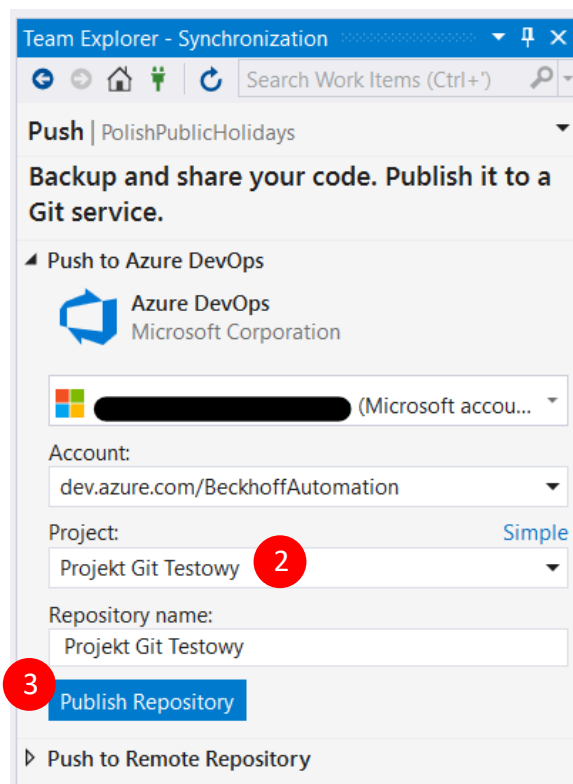
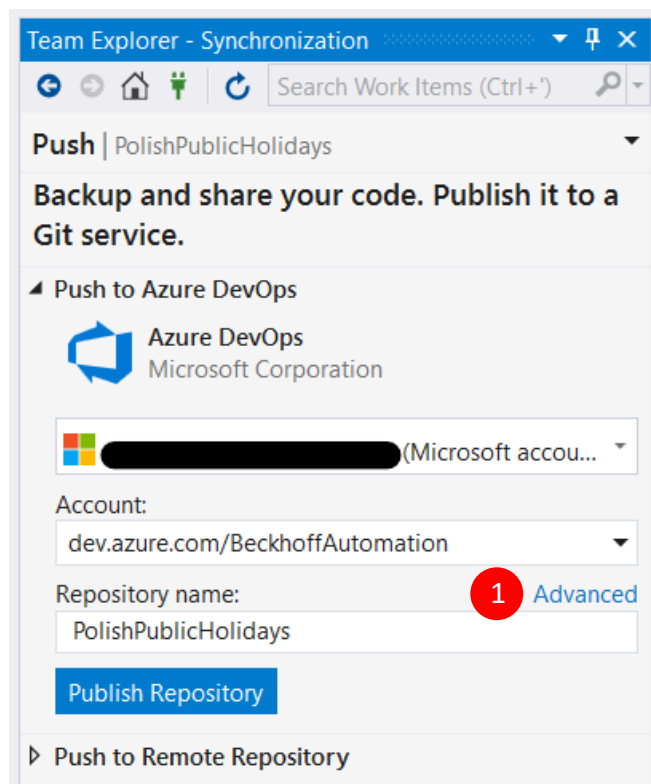
Po przesłaniu repozytorium na serwer otrzymamy potwierdzenie. Dodatkowo po najechaniu na nazwę repozytorium podejrzeć możemy status oraz dodatkowe informacje takie jak: adres repozytorium lokalny i serwera.



W organizacji na **Azure DevOps** pojawił się nowy projekt o nazwie **PolishPublicHoliday**. Jeżeli chcielibyśmy dodać repozytorium do wcześniej utworzonego projektu **Projekt Git Testowy**, należy to zdefiniować przed przesłaniem repozytorium na serwer.



W celach szkoleniowych proszę usunąć utworzone repozytorium z folderu projektu tak jak w [punkcie](#) oraz stworzyć repozytorium projektu na nowo (tym razem można oczywiście pozostawić już zmodyfikowany plik .gitignore). Następnie możemy postąpić analogicznie jak powyżej, tym razem rozwijając opcję **Advanced**. Wybieramy istniejący projekt **Git Testowy** i zatwierdzamy Publish Repository.

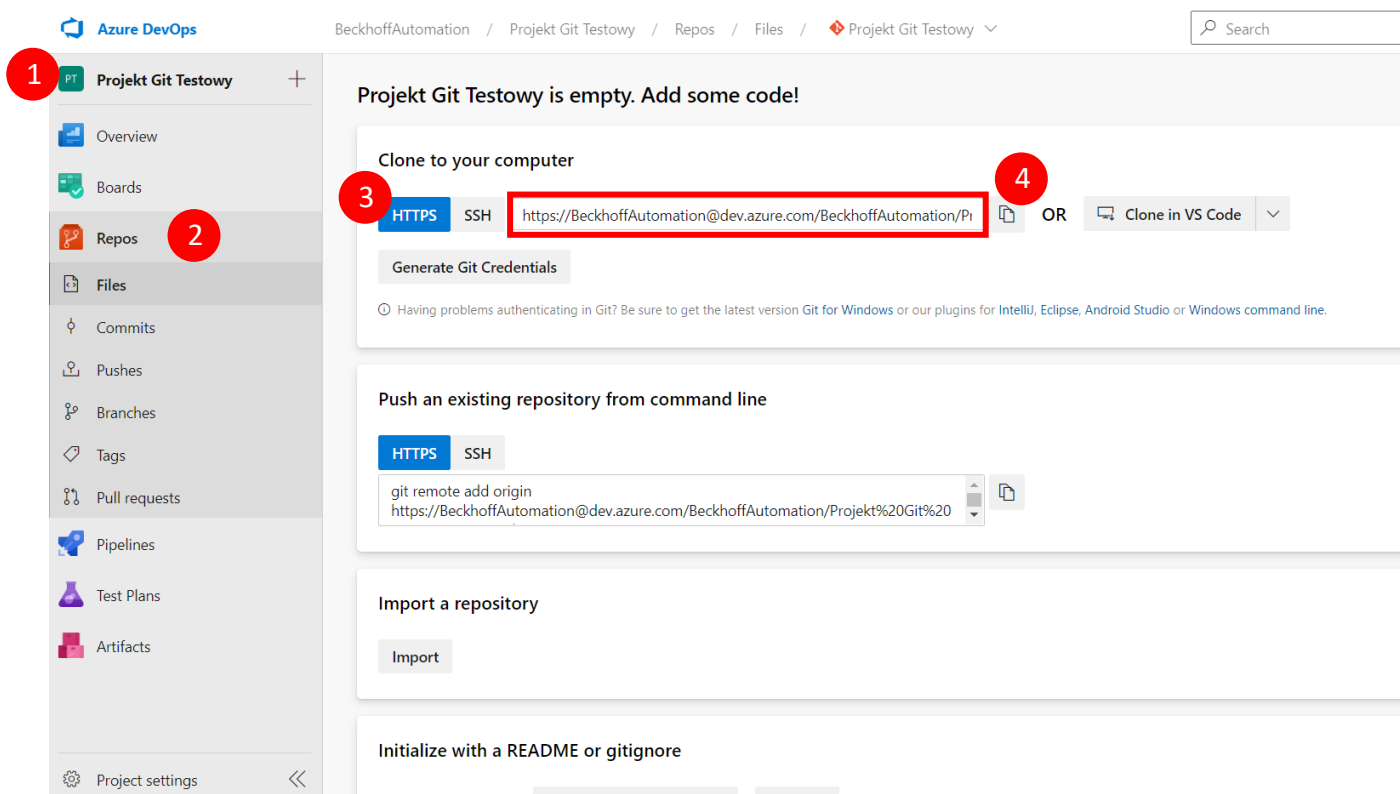


Mimo, iż takie postępowanie powinno przynieść efekt, pojawia się błąd, który bardzo szybko znika. Istnieją na szczęście sposoby, aby go obejść.

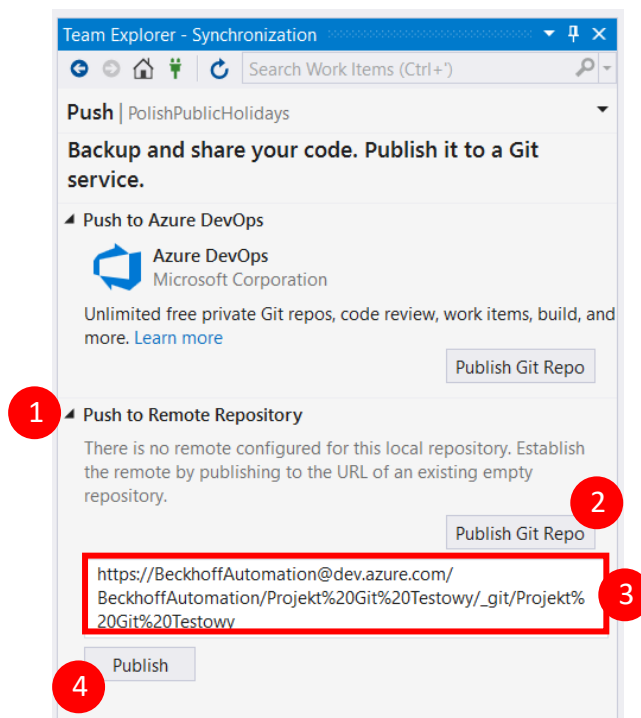


3.4.1 Użytkownik początkujący (zalecane)

W wyszukiwarce otwieramy interesujący nas projekt docelowy i otwieramy Azure Repos. Kopiujemy adres URL.

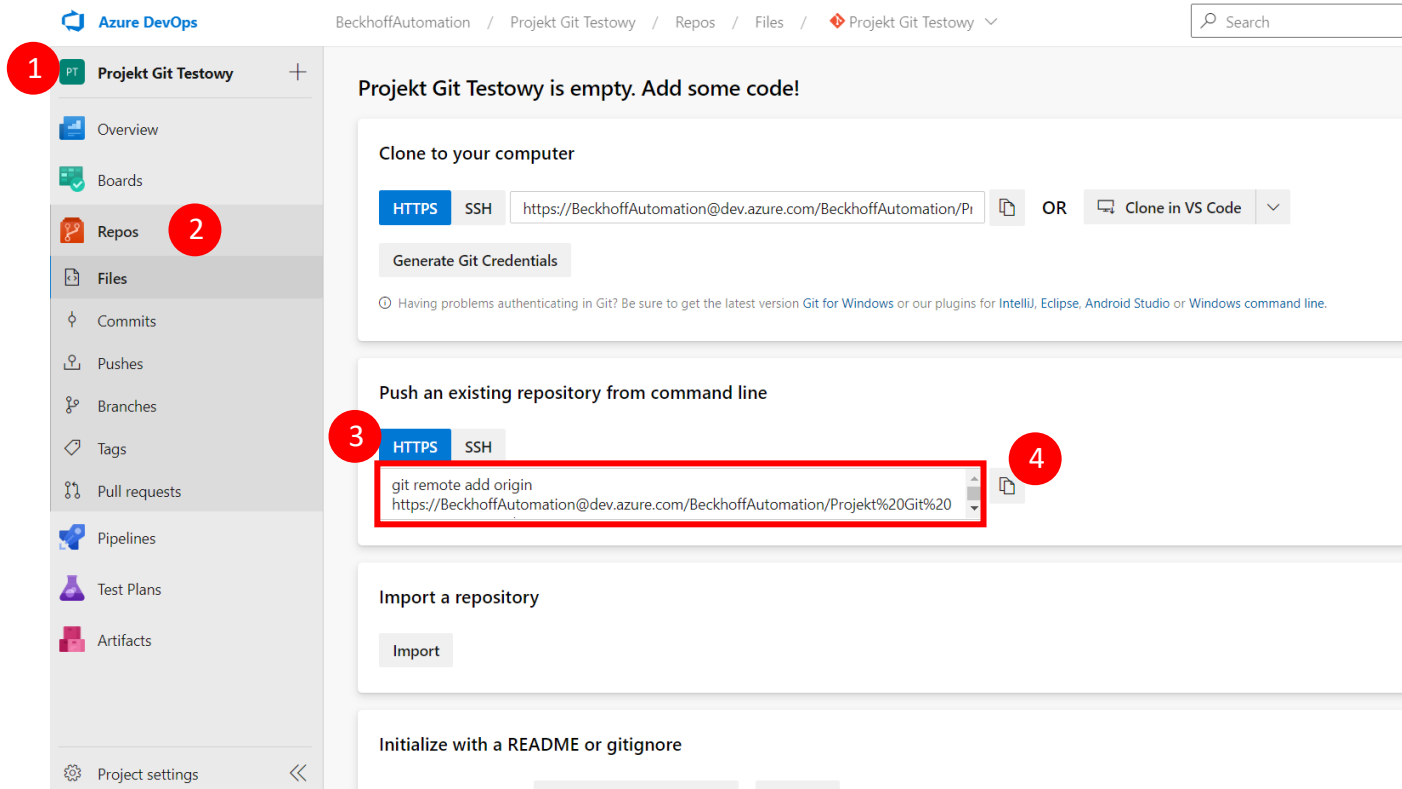


W TwinCATcie otwieramy okno **Team Explorer**. Tym razem zamiast Push to Azure DevOps wybieramy – **Push to Remote Repository**. Wklejamy wcześniej skopiowany **adres serwera** z repozytorium i klikamy **Publish**.

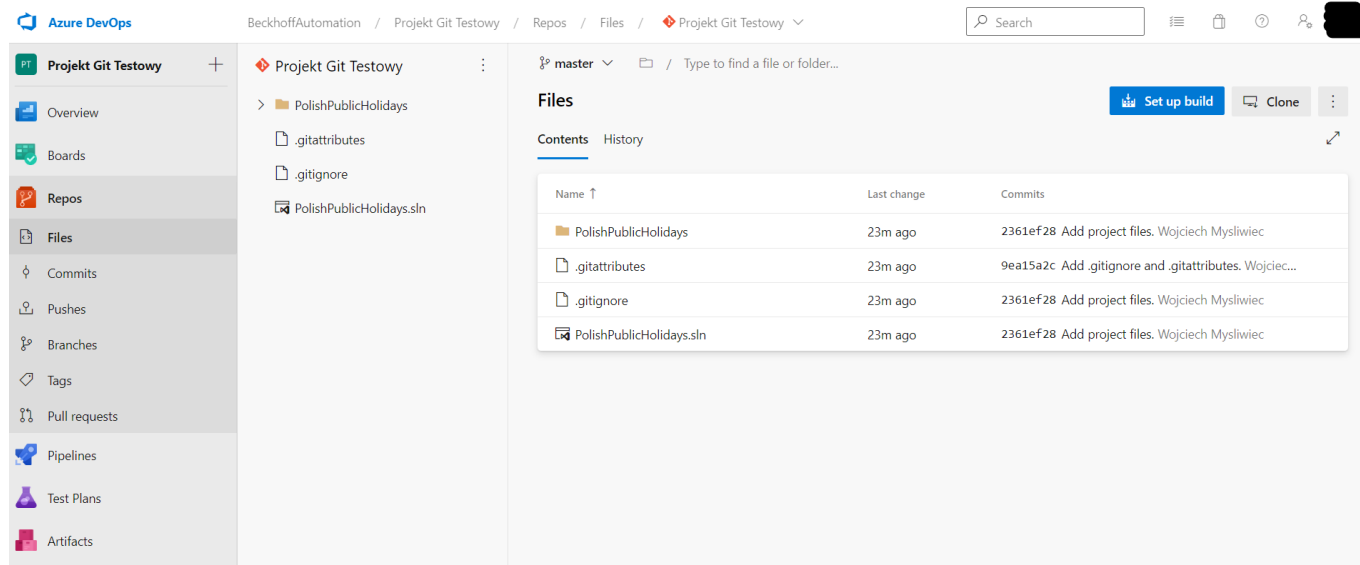


3.4.2 Użytkownik posiadający Git for Windows (zaawansowany)

Z poziomu folderu z repozytorium na komputerze wykonujemy komendy przygotowane w Azure DevOps. Można skorzystać z wbudowanego w TwinCATa/Visual Studio **Package Manager Console** [View->Other Windows->Package Manager Console].



Poprawność przesłania repozytorium możemy łatwo sprawdzić odświeżając Azure Repos. Pliki programu zostały pomyślnie przesłane na serwer.

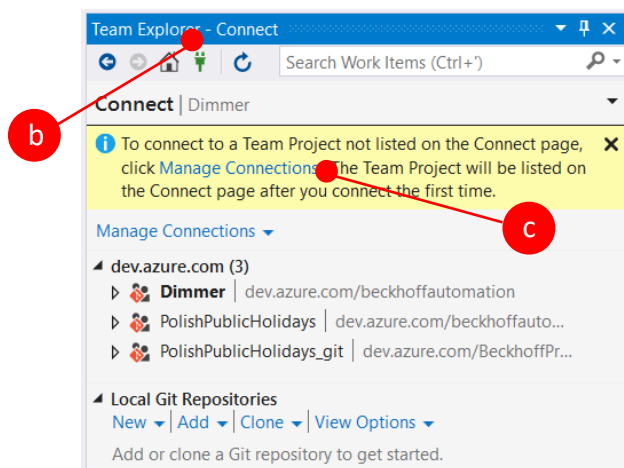


3.5 Klonowanie repozytorium na komputer

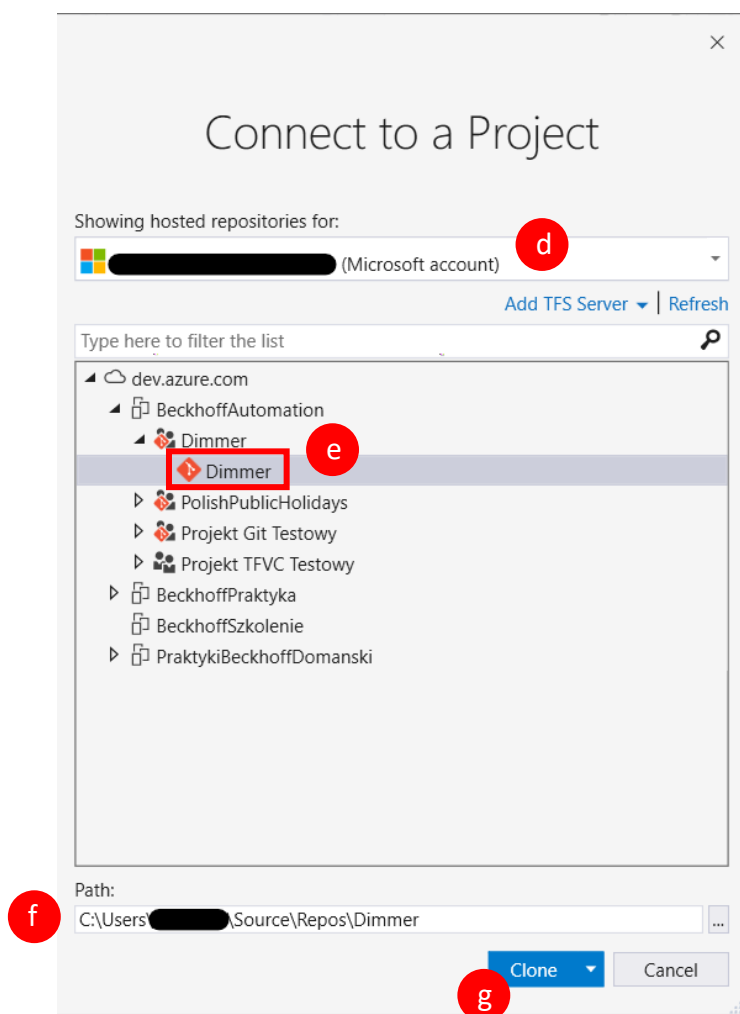
3.5.1 Azure Repos

Aby pobrać repozytorium z serwera – operacja **Clone**, należy:

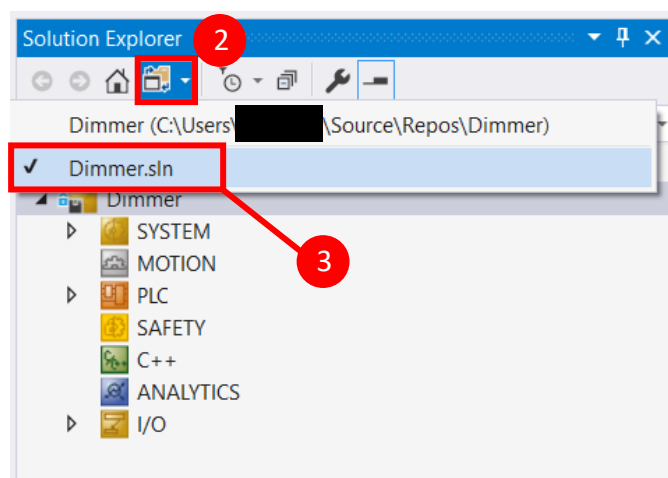
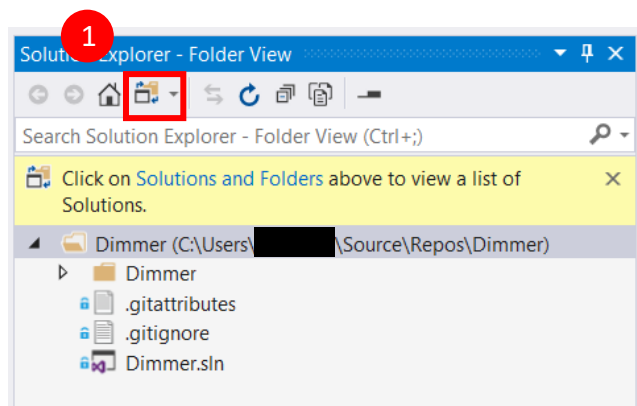
- Otworzyć TwinCATa, w razie potrzeby zamknąć dotychczasowe Solution
- W zakładce **Team Explorer** nacisnąć ikonę wtyczki



- Manage Connections**
- Logujemy się** do konta Microsoft
- Wybieramy interesujące nas repozytorium z listy
- Wpisujemy ścieżkę, gdzie ma ono zostać zapisane
- Zatwierdzamy **Clone**

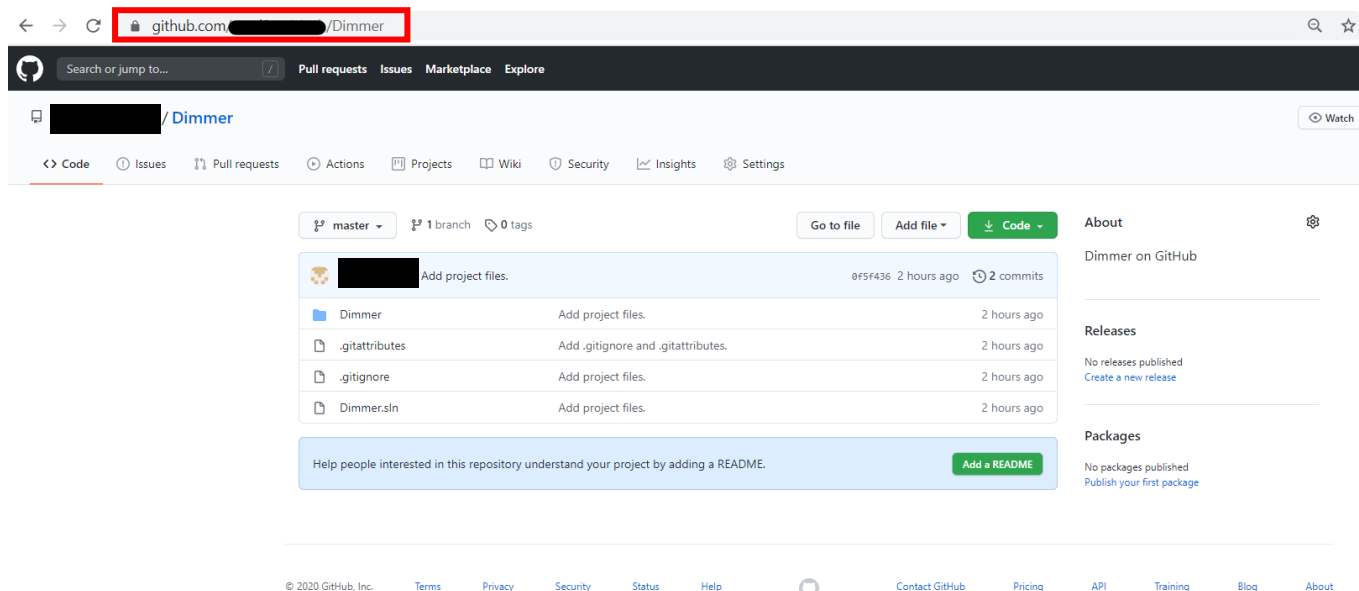


Po sklonowaniu ukaże nam się drzewko folderu z repozytorium. W celu przełączenia na widok projektu należy wybrać plik z rozszerzeniem .sln z rozwijanego menu.

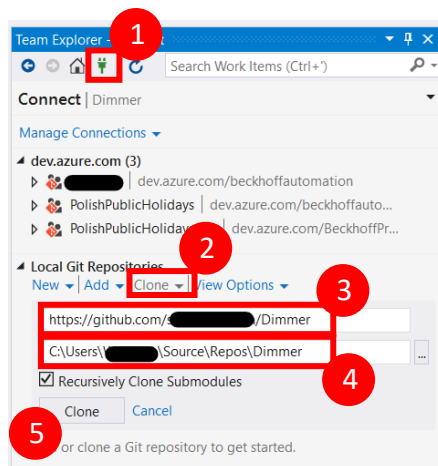


3.5.2 Inne

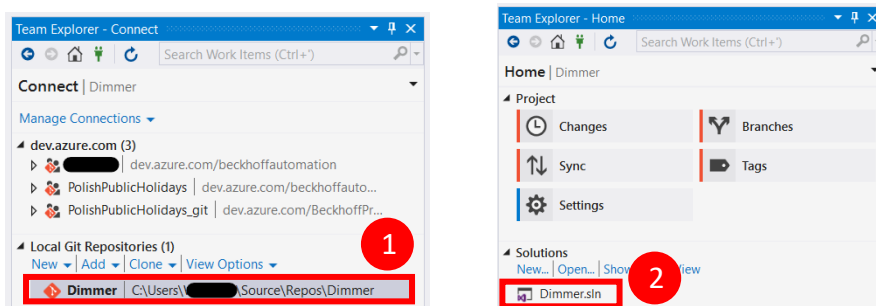
Zaletą Gita jest niewątpliwie jego popularność. Repozytorium można pobrać/sklonować z wielu popularnych serwisów. W tym przykładzie klonowania dokonamy bezpośrednio poprzez link do repozytorium, które znajduje się na GitHubie.



Podobnie jak wcześniej w Team Explorer klikamy na ikonę wtyczki. W Local Git Repositories wybieramy opcję Clone. Wpisujemy adres serwera oraz lokalną ścieżkę, gdzie ma ono zostać zapisane. Zatwierdzamy przyciskiem Clone.



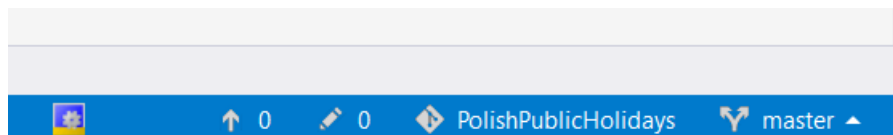
Repozytorium zostaje utworzone. Aby je otworzyć należy dwukrotnie kliknąć na nie LPM, a następnie w plik z rozszerzeniem .sln



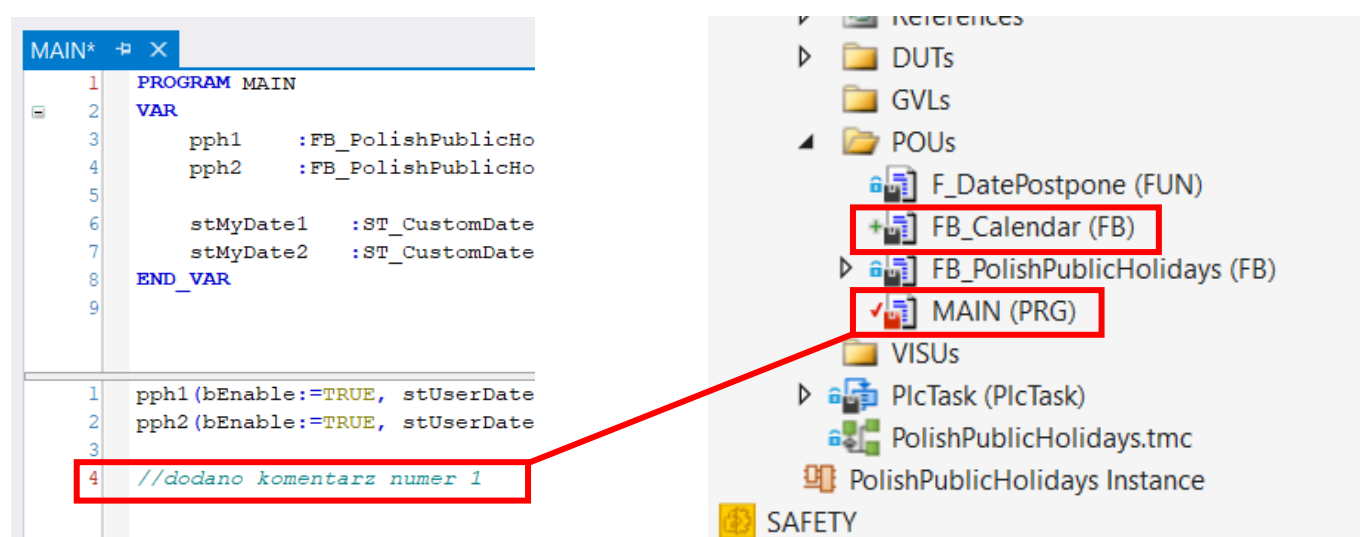
4 Podstawowe operacje na repozytorium Git

4.1 Wprowadzanie i akceptowanie zmian

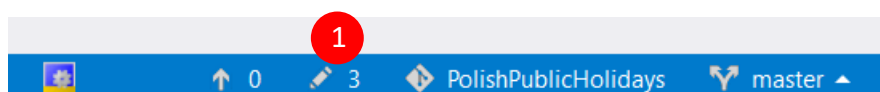
O aktualnym stanie branchu, w którym się znajdujemy informuje nas pasek w prawym dolnym rogu, który omówiony został w [punkcie](#). Poniżej zaprezentowany został stan repozytorium, który świadczy o zgodności branchu master, na którym pracujemy z branchem master na serwerze.



Wprowadźmy dla przykładu zmiany w kodzie – MAIN (PRG) oraz dodajmy nowy plik – FB_Calendar. Od razu widzimy zmiany przy nazwach plików w drzewku. **Zielony plus** oznacza, iż plik został utworzony i jeszcze nie należy do repozytorium. **Czerwony ptaszek** oznacza, że plik został zmieniony względem poprzedniego **Commitu**.

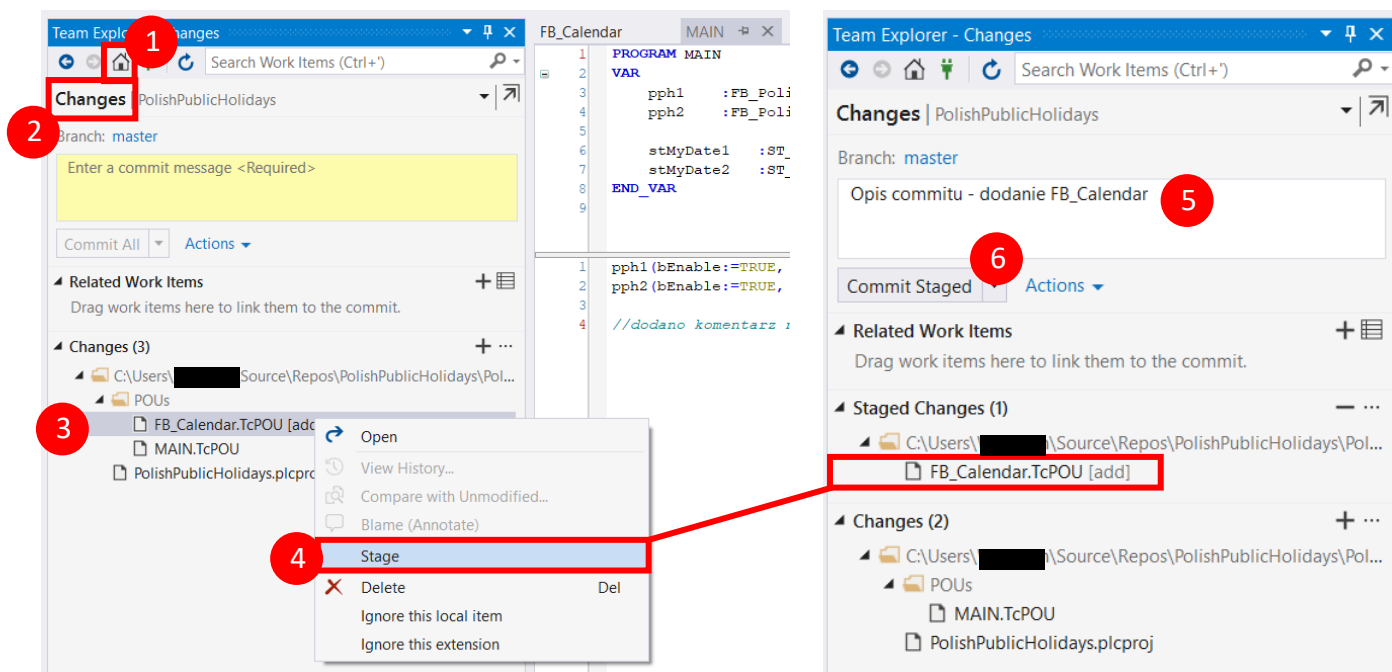


Dodatkowo informacja o zmianach w liczbie 2 program pokazuje na pasku po prawej stronie na dole. Klikamy w ikonę ołówka, w celu podejrzenia zmian i ich zaakceptowania.

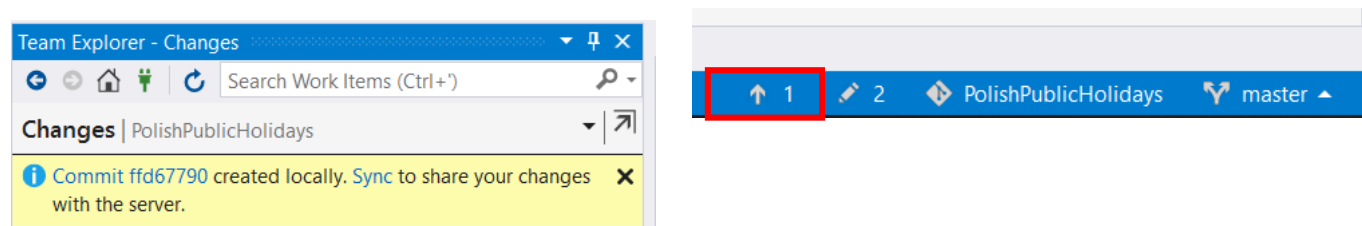


4.2 Stage & Commit

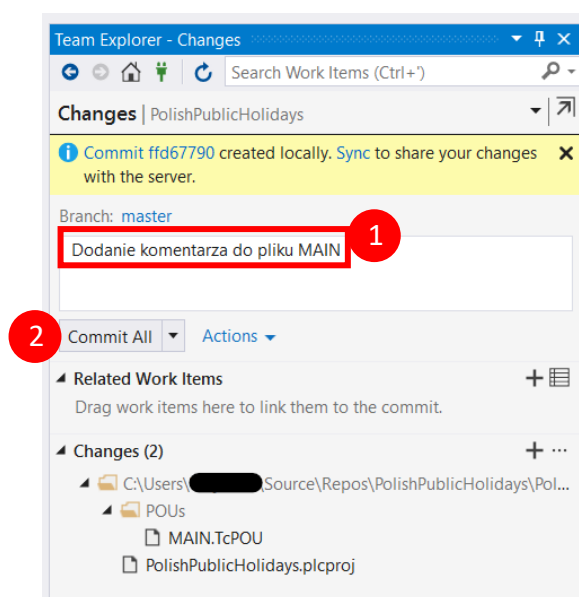
Stage oznacza wprowadzenie zmian na listę, które zostaną częścią kolejnego commitu. W **Team Explorer->Home->Changes** program pokazuje drzewo projektu, w którym wprowadzone zostały zmiany. Użytkownik ma możliwość wybrania konkretnych zmian, które zostaną zawarte w commitcie, oraz wyodrębnienia tych, które wymagają dalszej pracy, lub będą częścią osobnego commitu, a nawet brancha. Aby dodać zmieniony plik do listy należy nacisnąć PPM, a następnie wybrać **Stage**. W momencie, w którym nasza lista będzie gotowa tworzymy opis commitu (**każdy commit musi mieć swój opis** – tzw. commit message) i zatwierdzamy **Commit Staged**. Należy pamiętać, że commitu dokonujemy na konkretnym branchu – w tym przypadku master.



Komunikat informuje nas, że udało się utworzyć Commit **lokalny**. Oznacza to, że prowadziliśmy zmiany w lokalnym branchu, a ten na serwerze pozostał niezmieniony. Dodatkowo informuje nas o tym liczba przy ikonie strzałki. Należy ją interpretować jako liczbę nie przesłanych na serwer commitów.

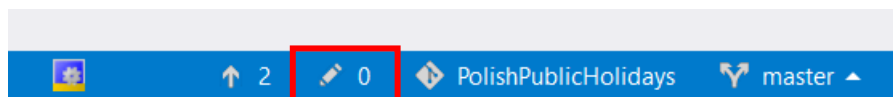


Pozostałe zmiany dodajmy w drugim commitcie, za pomocą przycisku **Commit All**. Oczywiście uprzednio nadając **commit message**.

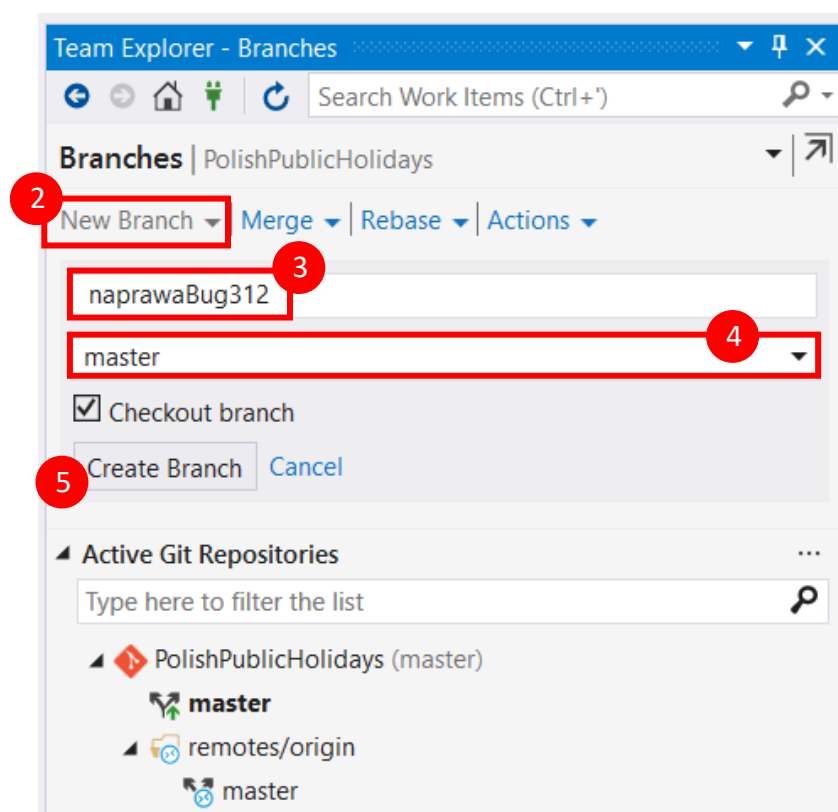
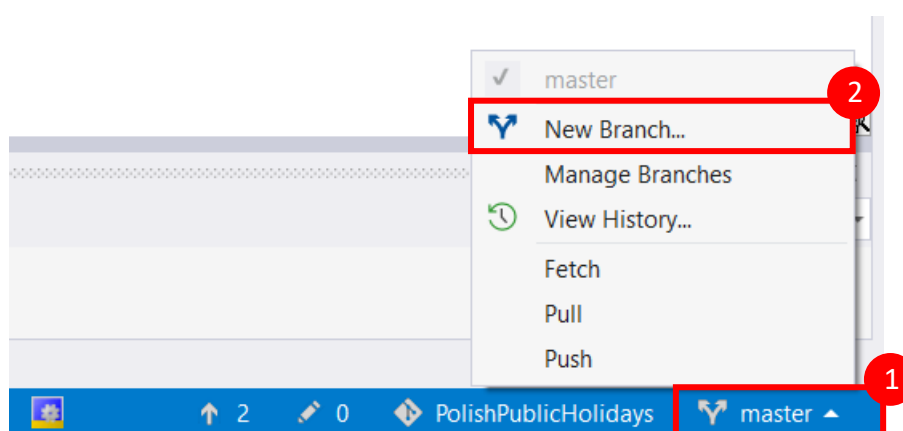


4.3 Branch

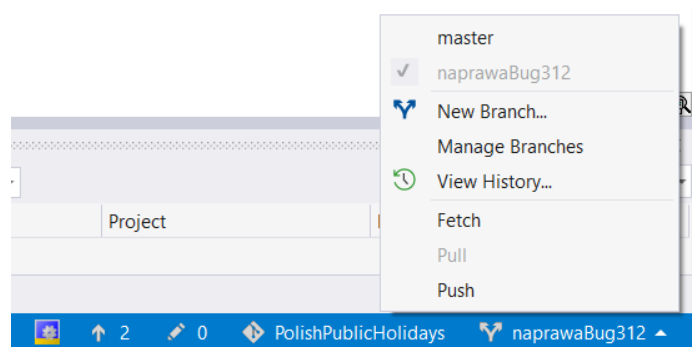
Branch służy w głównej mierze do dodawania nowych funkcjonalności, testowania oraz naprawiania bugów w sposób, który nie naraża podstawowej funkcjonalności projektu. Pracując na branchu pracuje się tak na prawdę na swego rodzaju kopii, która jest odgałęzieniem programu głównego. Kiedy fragment projektu przejdzie fazę testów branch można połączyć z innym branchem, a w szczególności z głównym branchem master, a niepotrzebny już branch usunąć. **Aby jednak utworzyć branch, trzeba jednak zdecydować o losie wszystkich wprowadzonych zmian. Uwaga ta dotyczy również przełączania się pomiędzy branchami.** Można je zaakceptować w ramach commitu, lub “odłożyć do schowka”, czyli użyć opcji **Stash**. Zostanie ona omówiona w dalszej części instrukcji. O gotowości do zmiany branchu świadczy 0 przy ikonie ołówka.



Aby dodać nowy branch należy rozwinąć menu branch i wybrać opcję **New Branch...** Można również wybrać **Team Explorer->Home->Branch->New Branch**

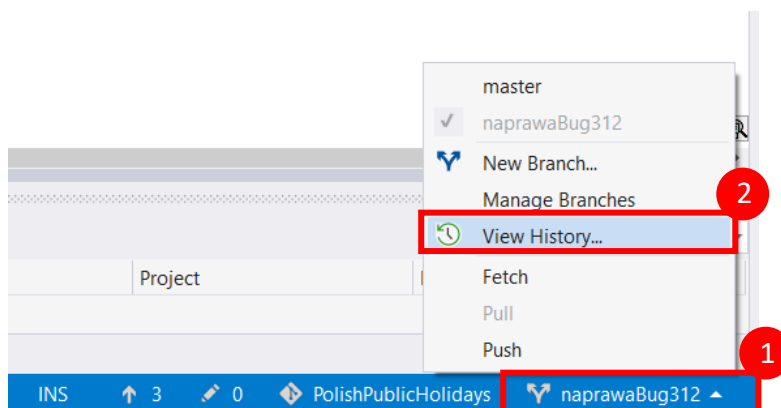


Wpisujemy nazwę nowego branchu, najlepiej taką która informuje o jego przeznaczeniu. Określamy bazę na podstawie, której powstanie nowy branch. Innymi słowy, od którego brancha powstanie rozgałęzienie – w tym przypadku master. Zatwierdzamy przyciskiem **Create Branch**. Opcja **Checkout branch** oznacza wyjście z poprzedniego branchu i wejście do nowo utworzonego. Status obecnego brancha nieodzwrotnie obserwować możemy w prawym dolnym rogu. Po rozwinięciu menu można powrócić do **master**.



4.4 Merge

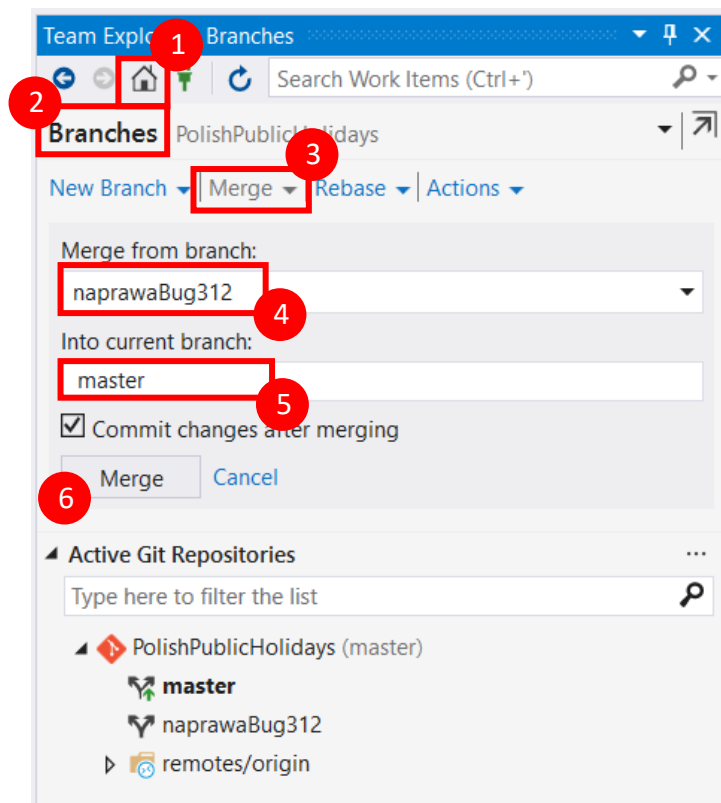
Merge odnosi się do łączenia branchy. Na potrzeby niniejszej instrukcji zasymulujemy naprawienie Bug312, a następnie połączenie obu branchy. Uczynimy to usuwając drugą linię kodu w pliku MAIN (PRG) w branchu naprawaBug312. Równocześnie w masterze dodany zostanie nowy program P_PRG (PRG). Aby podejrzeć historię commitów w branchach należy wybrać interesujący nas branch z menu w prawym dolnym rogu. Następnie wybieramy opcję **View History**. Tak samo postępujemy po wyborze branchu **master**.



History - naprawaBug312				
Graph	ID	Author	Date	Message
Local History				
	4262034c		24.07.2020 16:17:43	usunięto linię kodu numer 2 z MAIN (PRG) naprawaBug312
	47e3ff44		24.07.2020 15:21:36	Dodanie komentarza do pliku MAIN
	ffd67790		24.07.2020 15:09:59	Opis commitu - dodanie FB_Calendar
	2a4d3cae		24.07.2020 09:27:35	Add project files.
	2a12e95c		24.07.2020 09:27:33	Add .gitignore and .gitattributes.

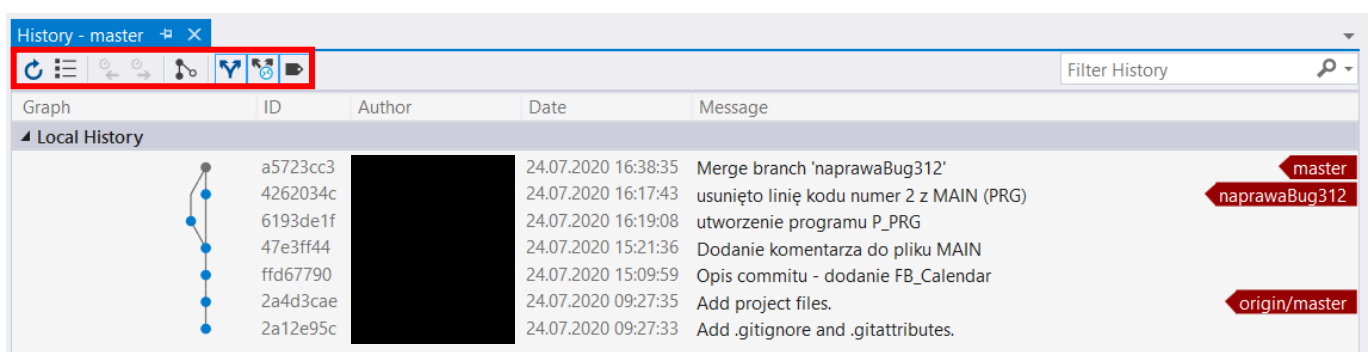
History - master				
Graph	ID	Author	Date	Message
Local History				
	6193de1f		24.07.2020 16:19:08	utworzenie programu P_PRG master
	47e3ff44		24.07.2020 15:21:36	Dodanie komentarza do pliku MAIN
	ffd67790		24.07.2020 15:09:59	Opis commitu - dodanie FB_Calendar
	2a4d3cae		24.07.2020 09:27:35	Add project files.
	2a12e95c		24.07.2020 09:27:33	Add .gitignore and .gitattributes.

Tak przygotowane branche chcemy teraz połączyć. W oknie **Team Explorer** wybieramy **Home->Branches->Merge**. Wybieramy branch, który chcemy połączyć – w naszym przypadku **naprawaBug312** oraz branch docelowy – **master** (obecnie na nim pracujemy). Zatwierdzamy **Merge**.



Rezultat połączenia dwóch branchy podejrzeć można na **grafie** w historii branchu docelowego (w tym przypadku master). Jeżeli historia branchu wygląda inaczej niż ta przedstawiona poniżej, proszę się upewnić czy zaznaczone są te same opcje w menu okna **History**. Ikony i ich opis według kolejności, zaczynając od lewej:

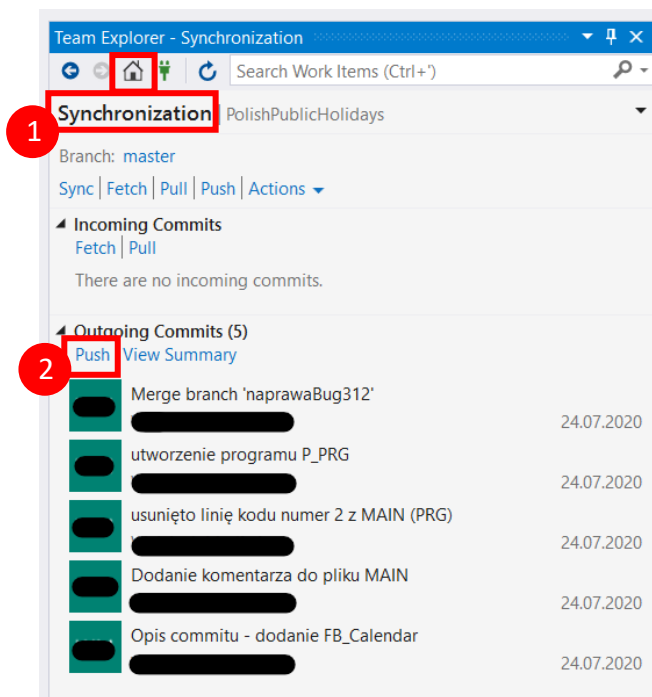
- Refresh** – odśwież historię, przydatne w celu sprawdzenia czy stan Remote Branches nie uległy zmianie
- Switch to Simple View** – włączone prezentuje commity w formie listy bez grafu
- Go to Child** – opcja aktywna po zaznaczeniu konkretnego commitu; przejście do commitu następnego
- Go to Parent** - opcja aktywna po zaznaczeniu konkretnego commitu; przejście do commitu poprzedniego
- Show First Parent Only** – ograniczenie historii tylko do jednego branchu będące “rodzicem”
- Show Local Branches** – pokazania branchy występujących lokalnie na komputerze
- Show Remote Branches** – pokazanie branchy występujących na serwerze (w repozytorium głównym)
- Show Tags** – pokaż etykiety branchy, gdzie obecnie znajduje się HEAD



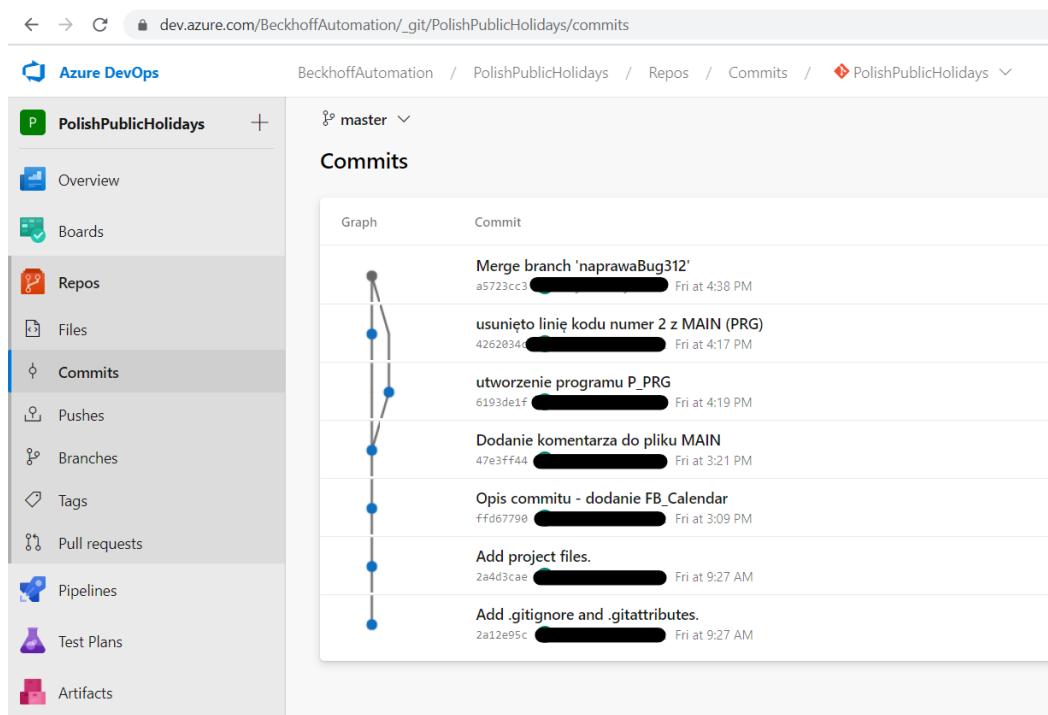
4.5 Push

Dotychczas poczyniane zmiany dokonywane były na repozytorium lokalnym. Widoczne jest to w historii branchów (screen powyżej), gdzie HEAD branchu **origin/master** wskazuje nadal na drugi commit z kolei. Branch **naprawaBug312** z kolei występuje tylko w repozytorium lokalnym. Aby przesłać zmiany do repozytorium głównego (na serwer) należy:

- Wybrać w oknie **Team Explorer->Home->Sync(hronization)**
- Następnie użyć opcji **Push**



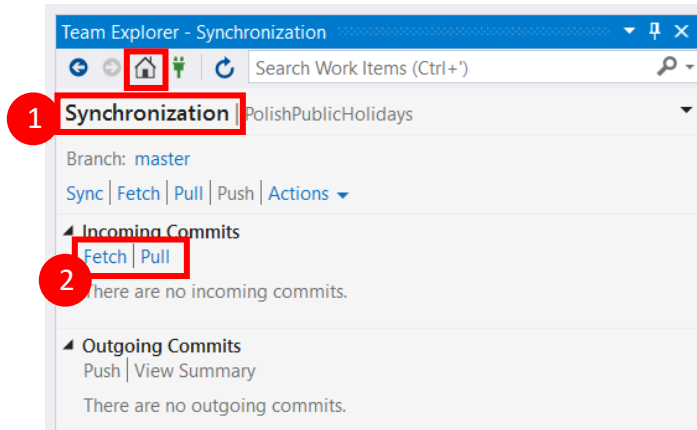
Po wykonaniu komendy Push repozytorium zdalne zostało zaktualizowane. Tym samym na platformie Azure w zakładce Repos w naszym projekcie podejrzeć można tą samą historię commitów, którą przeglądaliśmy w TwinCATcie jako historię repozytorium lokalnego.



4.6 Fetch & Pull

Oba narzędzia służą do pobierania obiektów/commitów z repozytorium zdalnego z tą różnicą, że komenda *Pull* automatycznie integruje pobrane zmiany z lokalnym branchem. *Pull* wykorzystywany jest do „zaciągnięcia” najnowszej wersji brancha do repozytorium lokalnego. *Fetch* pokazuje nam natomiast commity, które wyprzedzają nasz lokalny branch. Daje to możliwość podejrzenia zmian przed zsynchronizowaniem lokalnego brancha z tym z repozytorium zdalnego. Dodatkowo poprzez użycie komendy *cherry-pick* jesteśmy w stanie wcielić do lokalnego brancha zmiany tylko z wybranych commitów (zachęcamy do zapoznania się z dokumentacją Git w celu zgłębienia tajników instrukcji *cherry-pick*). W celu wywołania komend *Fetch & Pull*:

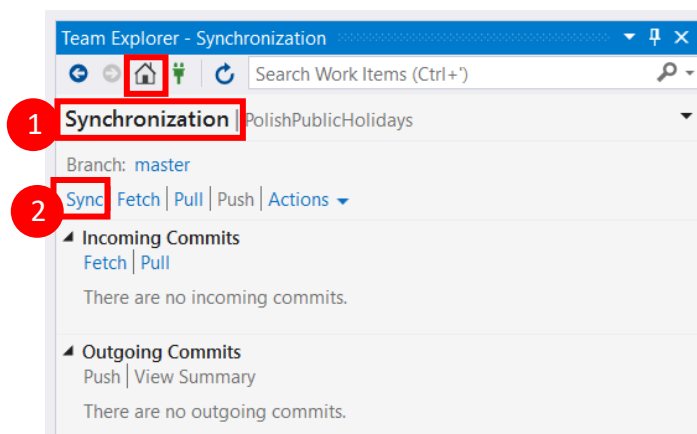
- Wybierz w oknie **Team Explorer->Home->Sync(hronization)**
- Wybierz **Fetch, Pull** lub **Fetch & Pull**



4.7 Sync

Komenda **Sync** powoduje synchronizację zdalnego repozytorium na serwerze oraz repozytorium lokalnego. Zmiany synchronizowane są w obydwu kierunkach jednocześnie. Oznacza to, że tak naprawdę Sync działa jak wywołanie obu komend *Push&Pull*. Z repozytorium lokalnego przesłane zostają commity na repozytorium zdalne oraz z repozytorium zdalnego „zaciągane” są zmiany do repozytorium lokalnego. Różnica w stosunku do komendy **Clone** jest taka, że **Clone** wykonywane jest raz przy pierwszym synchronizowaniu (de facto tworzeniu) repozytorium z serwera na komputer. Później używa się komend **Pull** lub **Sync**. W celu wywołania komendy **Sync**:

- Wybierz w oknie **Team Explorer->Home->Sync(hronization)**
- Wybierz **Sync**



4.8 Stash

Komenda **Stash** służy do zapisania obecnego stanu katalogu roboczego. Poczynione zmiany są zapisywane do schowka, a katalog przywrócony zostaje do HEAD commitu. Dzięki temu zabiegowi można m.in. zmienić branch, mimo że wprowadziliśmy do projektu zmiany,

które nie są częścią żadnego commitu. W dowolnej chwili można podejrzeć obiekty, które zostały zapisane w schowku i je przywrócić. Użycie komendy **Stash**:

- a) Wybierz w oknie **Team Explorer->Home->Changes**
- b) Zaznacz zmiany na liście, które chcesz zapisać w schowku
- c) Wybierz opcję **Stash** lub **Stash All**

5 Tips & Tricks

Rozdział ten poświęcony jest rozwiązywaniu błędów. Należy mieć na uwadze, że przyczyną błędów mogą być:

- Błędy instalacji TwinCATa
- Różnice programowe pomiędzy TwinCATem, a Visual Studio
- Błędy związane bezpośrednio z systemem kontroli wersji jakim jest Git
- Źle nadane uprawnienia dostępu w repozytorium zdalnym (np. Azure DevOps, GitHub)

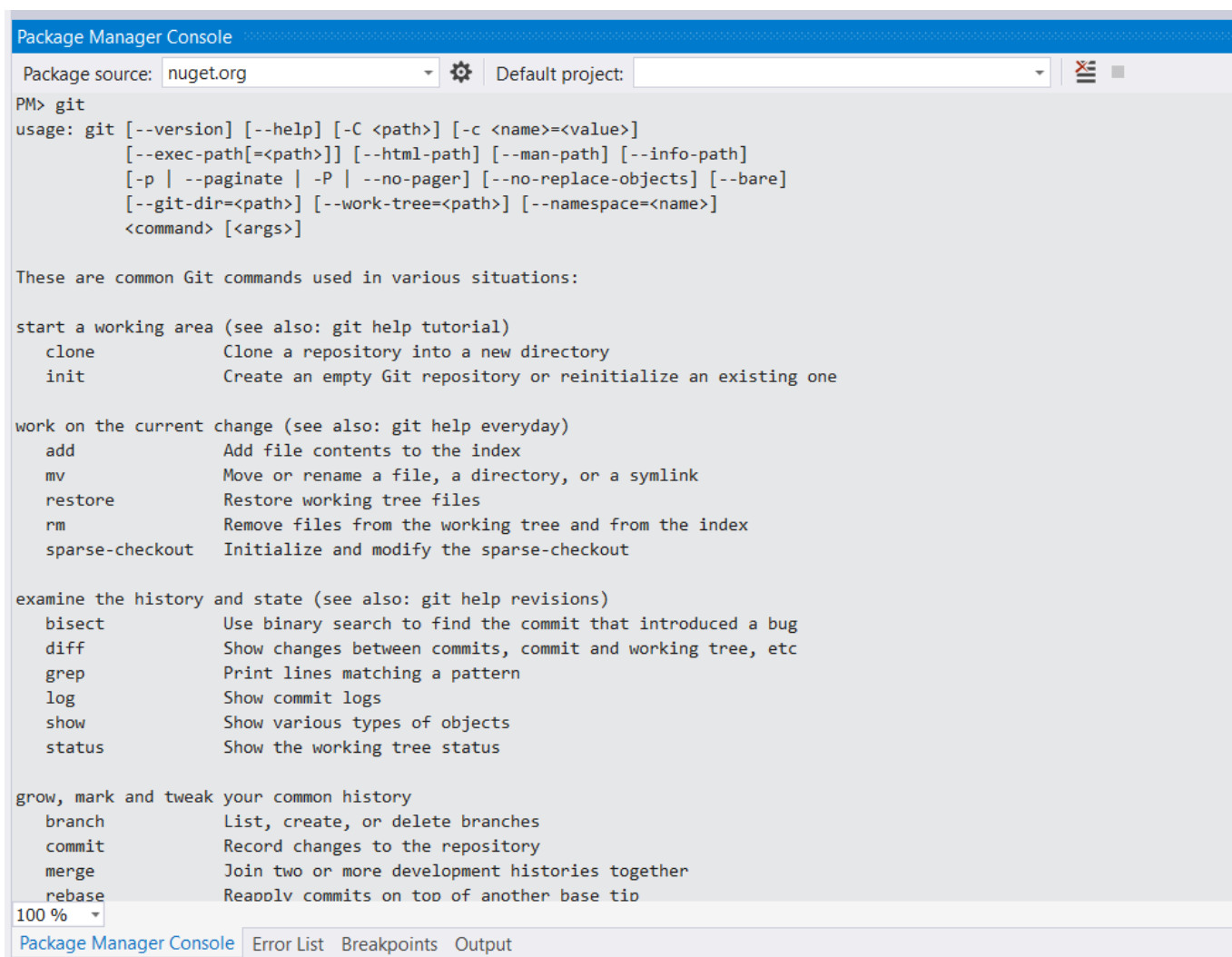
Począwszy od Visual Studio 2013 użytkownicy VS mają wbudowanego klienta Git bezpośrednio w IDE. Ponieważ TcXaeShell nie jest wierną kopią Visual Studio mogą wystąpić pewne problemy z systemem Git. Rozdział ten ma na celu podpowiedzieć jak je rozwiązać.

5.1 Porada ogólna

Podczas testów na repozytorium Git w narzędziu inżynierskim TwinCAT zauważono, że błędy mogą być spowodowany niepełną instalacją Gita w TwinCATcie TcXaeShell. Aby sprawdzić, czy Git został zainstalowany w TwinCATcie poprawnie należy:

- Otworzyć program TwinCAT TcXaeShell**
- Otworzyć okno **Package Manager Console** (View->Other Windows->Package Manager Console)
- Wpisać komendę `git` i zatwierdzić **Enter**

Jeżeli Git jest prawidłowo obsługiwany przez TwinCATa TcXaeShell konsola odpowie na komendę w zaprezentowany poniżej sposób.



```

Package Manager Console
Package source: nuget.org
PM> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone          Clone a repository into a new directory
  init           Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add            Add file contents to the index
  mv            Move or rename a file, a directory, or a symlink
  restore       Restore working tree files
  rm            Remove files from the working tree and from the index
  sparse-checkout Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
  bisect        Use binary search to find the commit that introduced a bug
  diff          Show changes between commits, commit and working tree, etc
  grep          Print lines matching a pattern
  log           Show commit logs
  show          Show various types of objects
  status        Show the working tree status

grow, mark and tweak your common history
  branch        List, create, or delete branches
  commit        Record changes to the repository
  merge         Join two or more development histories together
  rebase        Reapply commits on top of another base tip
  
```

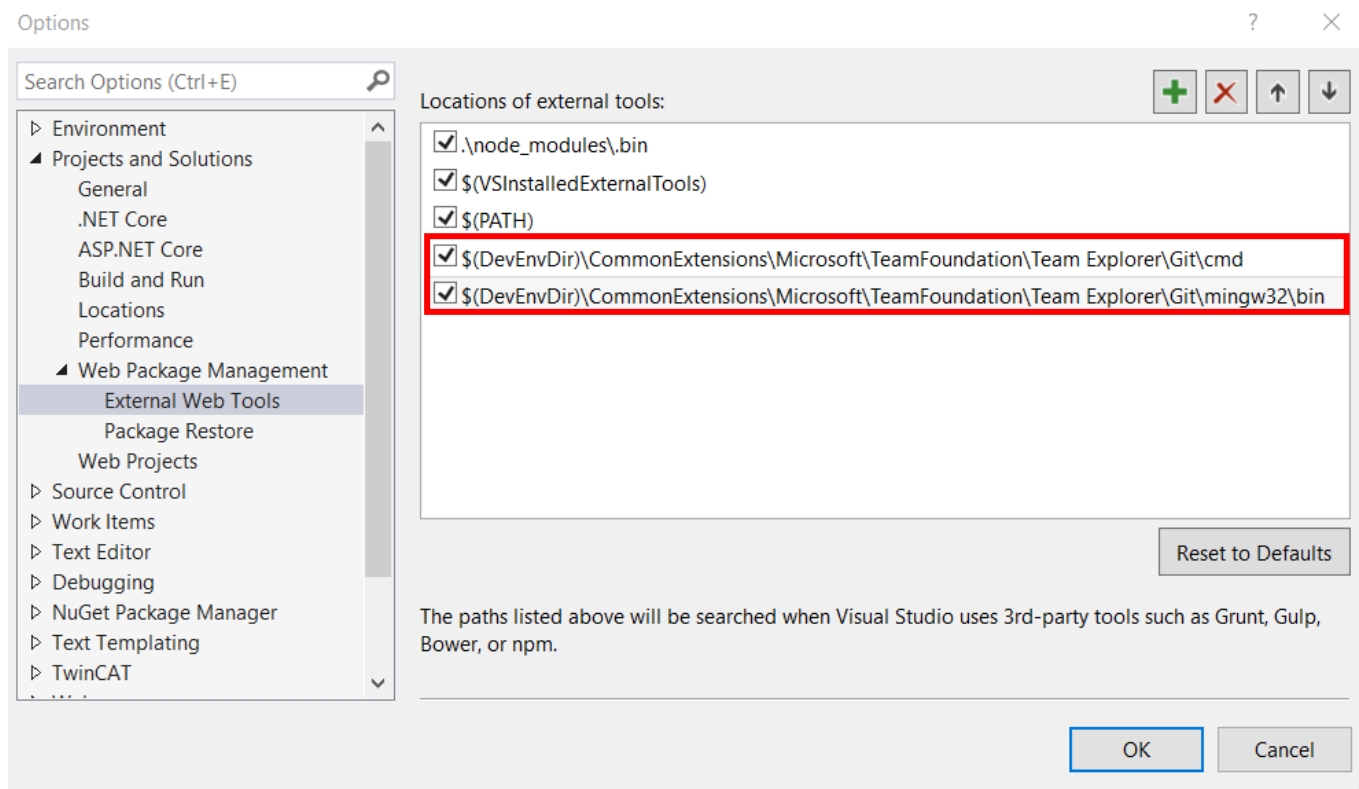
Jeżeli Git się nie zgłasza należy przeprowadzić instalację **Git for Windows**. Pliki instalacyjne dostępne pod adresem: <https://git-scm.com/download/win> . Instalację należy przeprowadzić z ustawieniami domyślnymi na każdym etapie. (Sprowadza się to do akceptowania ustawień proponowanych w Wizardzie instalatora). Następnie należy sprawdzić, czy w plikach instalacyjnych TwinCATa znajduje się **folder Git** pod zadaną ścieżką:

```
#Jest to zapis ścieżki skrócony, charakterystyczny dla Visual Studio
$(DevEnvDir)\CommonExtensions\Microsoft\TeamFoundation\Team Explorer\
#Pełna ścieżka może przyjąć postać
C:\Program Files (x86)\Beckhoff\TcXaeShell\Common7\IDE\CommonExtensions\Microsoft\TeamFoundation\Team Explorer
```

Jeżeli brakuje wspomnianego **folderu Git** należy odnaleźć na dysku zainstalowany przed chwilą **folder Git** i przekopiować w zadane powyżej miejsce. Domyślnie **Git for Windows** utworzy folder Git pod zadaną ścieżką:

```
# C jest w tym przypadku domyślnym dyskiem
C:\Program Files\
```

Jeżeli folder Git jest na swoim miejscu należy sprawdzić w zakładce **Tools->Options->Project and Solutions->Web Package Management->External Web Tools** zaznaczone dwie ścieżki są sprecyzowane. Jeżeli nie należy je dodać.



Jeżeli powyższe kroki nie pomogły (szczególnie jeśli folder Git był na swoim miejscu w plikach instalacyjnych TwinCATa) rozwiązaniem może być uzupełnienie folderu **git-core**. Podobnie jak w poprzednim kroku odnajdujemy w plikach instalacyjnych TwinCATa folder **Git**, a w nim szukamy podfolder **git-core**. Następnie należy go otworzyć i sprawdzić ilość plików w jego wnętrzu.

```
#Pełna ścieżka może przyjąć postać
C:\Program Files (x86)\Beckhoff\TcXaeShell\Common7\IDE\CommonExtensions\Microsoft\TeamFoundation\Team Explorer\Git\mingw32\Libexec\git-core
```

W osobnym oknie Eksploratora plików otwieramy zainstalowany poprzed **Git for Windows** folder Git i również odnajdujemy podfolder git-core. Otwieramy go oraz sprawdzamy ilość plików jak poprzednio.

```
# C jest w tym przypadku domyślnym dyskiem
C:\Program Files\mingw32\Libexec\git-core
```

Zaznaczamy wszystkie pliki z **git-core** z *Git for Windows* kopiujemy i wklejamy do folderu **git-core**, który znajduje się w *plikach instalacyjnych TwinCATa TcXaeShella*. Powtarzające się pliki pomijamy. Restartujemy TcXaeShella i wykonujemy komendę `git` ponownie w Package Manager Console. Należy sprawdzić, czy podstawowa funkcjonalność Gita działa (Push, Pull ect.).

Takim sposobem można poradzić sobie między innymi z poniższym błędem.

```
Git failed with a fatal error
NullReferenceException encountered.
cannot spawn /C/Program Files (x86)/Microsoft Visual
Studio/2017/Community/Common7/IDE/CommonExtensions/Microsoft/TeamFoundation/Team Explorer/Git/mingw32/Libexec/git-core/git-
askpass.exe: No such file or directory
could not read Username for ...
```


Który system kontroli wersji wybrać? Porównanie GIT/TFVC

MOŻLIWOŚCI	TFVC	GIT
DOSTĘPNE SERWERY	Azure DevOps Services, Team Foundation Server (instalowany lokalnie)	Azure DevOps Services, Team Foundation Server (instalowany lokalnie) oraz zewnętrzne oprogramowania np. Github
INFORMOWANIE	Członkowie zespołu mogą zostać poinformowani za pomocą wiadomości email o wpisach programowych na serwerze	Członkowie zespołu mogą zostać poinformowani za pomocą wiadomości email o wpływniu zmian na serwer w plikach programowych
BUDOWA PROJEKTÓW	Możliwa jest realizacja dowolnej ilości i połączenia plików źródłowych w obszarze roboczym jednego projektu	Możesz budować tylko jeden projekt w czasie wykorzystując różną liczbę repozytoriów jednocześnie
ZARZĄDZANIE PLIKAMI	Wszystkie pliki dostępne są w jednym folderze na serwerze. Można ustawić poziom dostępu do poszczególnych plików jak i całkowicie zablokować możliwość edytowania plików. Pliki można przeglądać w serwisie internetowym jak i przy użyciu Source Control Explorer wbudowanym w środowisko VS/TcXaeShell. Cały projekt występuje wyłącznie na serwerze. Dodatkowo wprowadzając zmiany na serwerze, wprowadzane są zmiany tylko w plikach edytowanych lokalnie przez użytkownika nawet jeżeli inne pliki były edytowane przez innych użytkowników na serwerze od ostatniej edycji.	Każdy projekt może zawierać wiele repozytoriów i każde repozytorium może zawierać wiele branchy. Dostęp do plików można edytować na poziomie dostępu do repozytoriów bądź też do poszczególnych branchy. Nie ma możliwości zablokowania edycji plików. Pliki można przeglądać przez portal internetowy. Całość można również przechowywać w lokalnym repozytorium na urządzeniu programistycznym w formie kopii danych. Brak możliwości przeglądania plików za środowiska programistycznego.
DOSTĘPNOŚĆ	Visual Studio, TcXaeShell, Eclipse	Visual Studio, TcXaeShell, Eclipse oraz wiele innych środowisk programistycznych.
ZARZĄDZANIE POSTĘPEM	Można odłożyć zmiany w plikach nieużywanych lub zawiesić zmiany wersji np. stabilnej „na półkę” z możliwością powrotu przed wysłaniem na serwer. Dodatkowo możliwość tworzenia branchy.	Możliwość tworzenia branchy.
ZGODNOŚĆ Z PROGRAMAMI VISUAL STUDIO	Możliwość obsługi na wszystkich wydanych wersjach Visual Studio. Możliwość obsługi w środowisku TcXaeShell.	Możliwość obsługi w wersjach Visual Studio 2013, 2015, 2017, 2019 oraz w wersji 2012 Update 4.
SKALA PROJEKTÓW	Możliwość efektywnej realizacji małych oraz ogromnych projektów (miliony plików na odgałęzienie o dużych rozmiarach). Warunkiem pracy w tym trybie jest użytkowanie gałęzi indywidualnych dla każdego użytkownika. TFVC nadpisuje pliki tylko edytowane przez programistę w danym etapie roboczym na serwerze. W sytuacji kiedy blok funkcyjny został w międzyczasie zmieniony przez innego programistę na serwerze system TFVC nie wykryje kolizji gdyż plik nie był lokalnie edytowany i może spowodować wystąpienie błędu w oprogramowaniu ze względu na zmianę w działaniu bloku funkcyjnego.	Można łatwo realizować mniejsze projekty. Jest możliwa realizacja większych projektów, jednakże trzeba planować z góry schemat realizacji projektu w celu uniknięcia kolizji programowych. Przykładowo w systemie TFVC branche są tworzone na serwerze z kolei w GIT są one tworzone lokalnie. TFVC blokuje utworzenie nowego brancha z identyczną nazwą co inny. Git nie posiada takiej funkcjonalności. Branche o takich samych nazwach utworzone lokalnie przez 2 różne osoby w trakcie wypychania na serwer kolizję która musi zostać rozwikłana. W TFVC taka kolizja nie występuje Git umożliwia pracę w lokalnym repozytorium tworząc commity lokalnie. Pozwala to na testowanie kodu na maszynie programisty przed wysłaniem wersji na serwer.

Podczas wysyłania na serwer wypychane jest całe repozytorium porównując wszystkie zawarte różnice w kodzie

POPRAWNOŚĆ

Podczas zatwierdzania kolejnych zmian w kodzie w postaci Commitów wypychamy fragmenty kodu które mogą być jeszcze niesprawdzone w celu zapisania danego etapu pisania oprogramowania. Drugi programista może pobrać niedokończoną wersję oprogramowania i na jej podstawie wprowadzać własne zmiany. W trakcie wprowadzania zmian może dojść do błędów niewynikających z nowych zmian w prowadzonym kodzie.

Wszelkie zmiany programistyczne są zatwierdzane lokalnie w postaci Commitów. Dopiero po ukończeniu danego etapu tworzenia oprogramowania wersję stabilną można wysłać na serwer gdzie będzie mogła być użytkowana przez innych programistów.

6 Organizacja pracy nad projektem przy użyciu Team Foundation Version Control w środowisku Azure na przykładzie TcXaeShell

6.1 Czym jest Team Foundation Version Control?

Team Foundation Version Control jest oprogramowaniem pozwalającym na śledzenie zmian w kodzie źródłowym podczas realizacji projektu. Podczas pisania kodu źródłowego masz możliwość wrzucenia wybranego stanu kodu źródłowego do historii. Systemy kontroli wersji zapisują wysłaną wersję na serwer umożliwiając jednocześnie dostęp do niej w przyszłości niezależnie od obecnego postępu w pisaniu programu.

6.1.1 Cechy charakterystyczne

Odmienne od GITa TFVC jest scentralizowanym systemem zarządzania plików. Jeden użytkownik posiada tylko jedną wersję plików w swojej przestrzeni roboczej. Dostęp do wersji wcześniejszych jest możliwy poprzez pobranie wybranej wersji z repozytorium na serwerze. Wszystkie odgałęzienia są umieszczone na serwerze, kiedy użytkownik posiada tylko 1 jednocześnie w obecnie edytowanej kopii.

TFVC posiada 2 modele pracy:

- **w przestrzeni serwerów** – wprowadzenie jakichkolwiek zmian wymaga zatwierdzenia obecnego stanu na urządzeniu stacjonarnym a następnie udostępnienia wersji na serwerze. Pozwala to na płynne prowadzenie dziennika zmian na serwerze ułatwiając obserwowanie postępu pracy nad danym projektem. Wykorzystując przestrzeń serwerową można zarządzać ogromną ilością plików wykorzystywanych w projekcie.

- **w przestrzeni lokalnej** – każdy użytkownik posiada wersję oprogramowania obecnie edytowaną w trybie offline. Użytkownicy mogą połączyć się z serwerem w celu sprawdzenia zmian w kodzie i rozwiązania konfliktów wersji programowych.

System Team Foundation Version Control jest sposobem kontroli wersji utworzonym przez firmę Microsoft. Opiera się ona o wykorzystanie platformy Azure zwanej wcześniej Team Foundation Server. Dodatkowo system Azure pozwala na przerobienie repozytorium TFVC na GIT. Jednocześnie możliwe jest posiadanie jednocześnie 2ch repozytoriów o charakterze GIT i TFVC w jednym projekcie jako 2 niezależne elementy.

6.2 Utworzenie projektu TFVC w środowisku Azure

Podczas tworzenia nowego projektu w środowisku Azure, w opcjach zaawansowanych należy zmienić nastawę kontroli wersji na Team Foundation Version Control[a].

Create a project to get started

Project name *
Projekt TFVC Testowy ✓

Description
Opis

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

Advanced

Version control ⓘ
Team Foundation Version Control

Work item process ⓘ
Basic

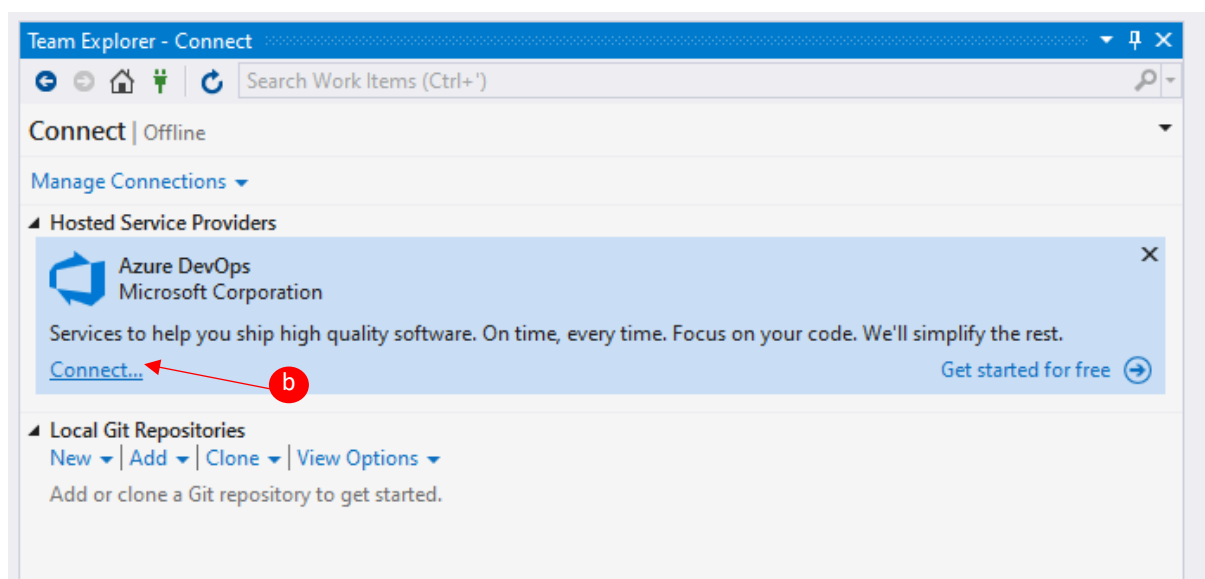
+ Create project

6.2.1 Utworzenie połączenia z repozytorium w Visual studio

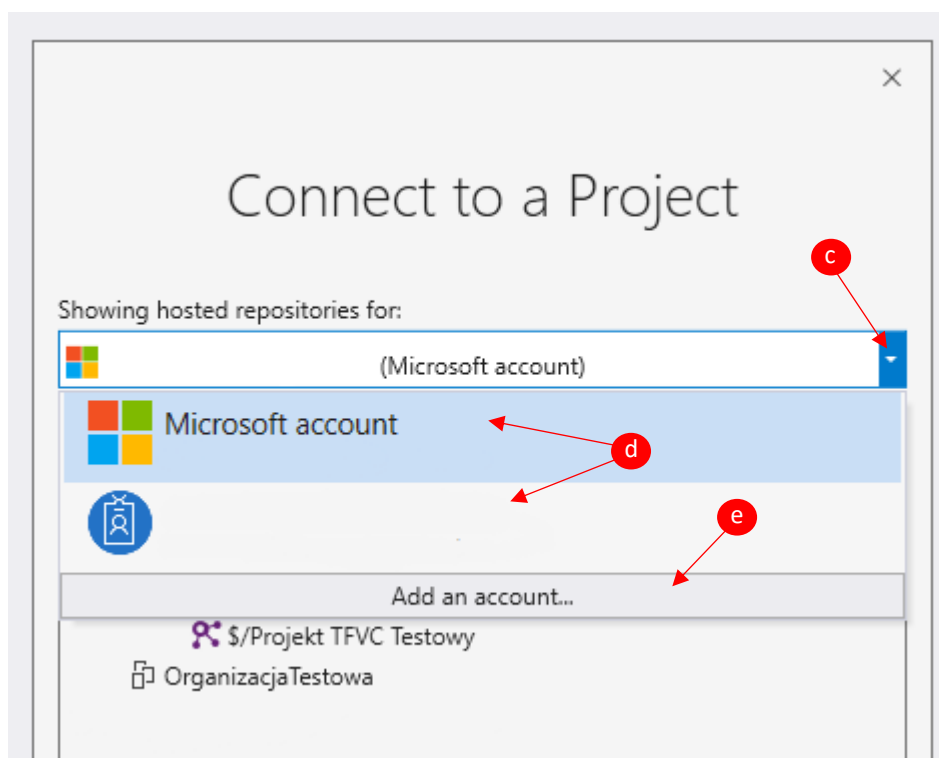
W oprogramowaniu Visual Studio używając okna oprogramowania Team Explorer należy wejść w ustawienia połączeń (Manage Connections) [a].



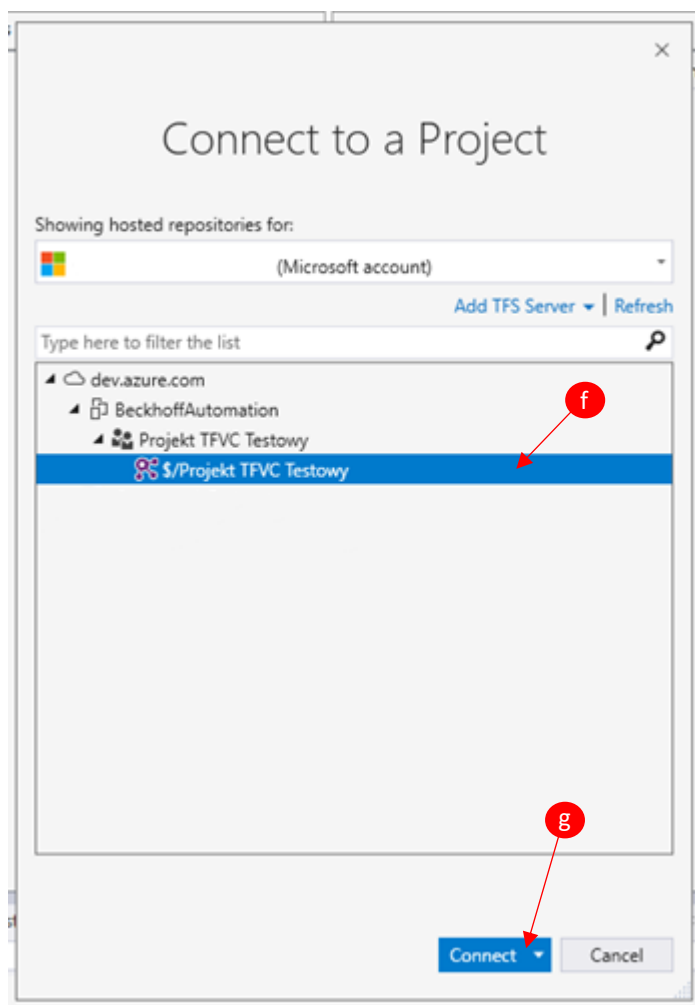
W następnym kroku należy kliknąć opcję Connect... w celu ustawienia połączenia z repozytorium [b]



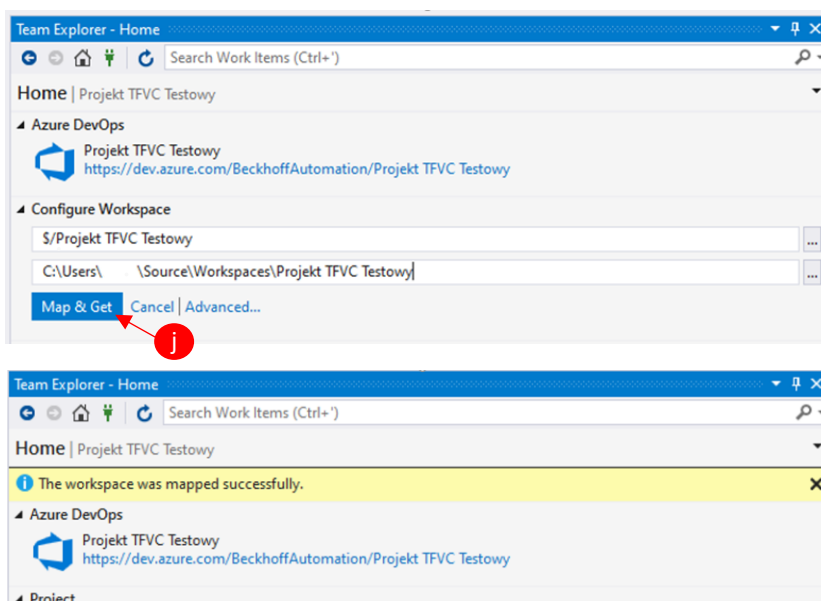
W obszarze okna Connect to a Project musimy wybrać interesujące nas konto użytkownika posiadające dostęp do repozytorium danego projektu [c]. W tym celu należy wybrać istniejące już konto [d]/dodać nowe logując się na konto użytkownika [e].



Po zalogowaniu się na konto użytkownika Wybieramy interesujące nas repozytorium z wybranego projektu [e]. Po wybraniu należy połączyć się klikając Connect [g].

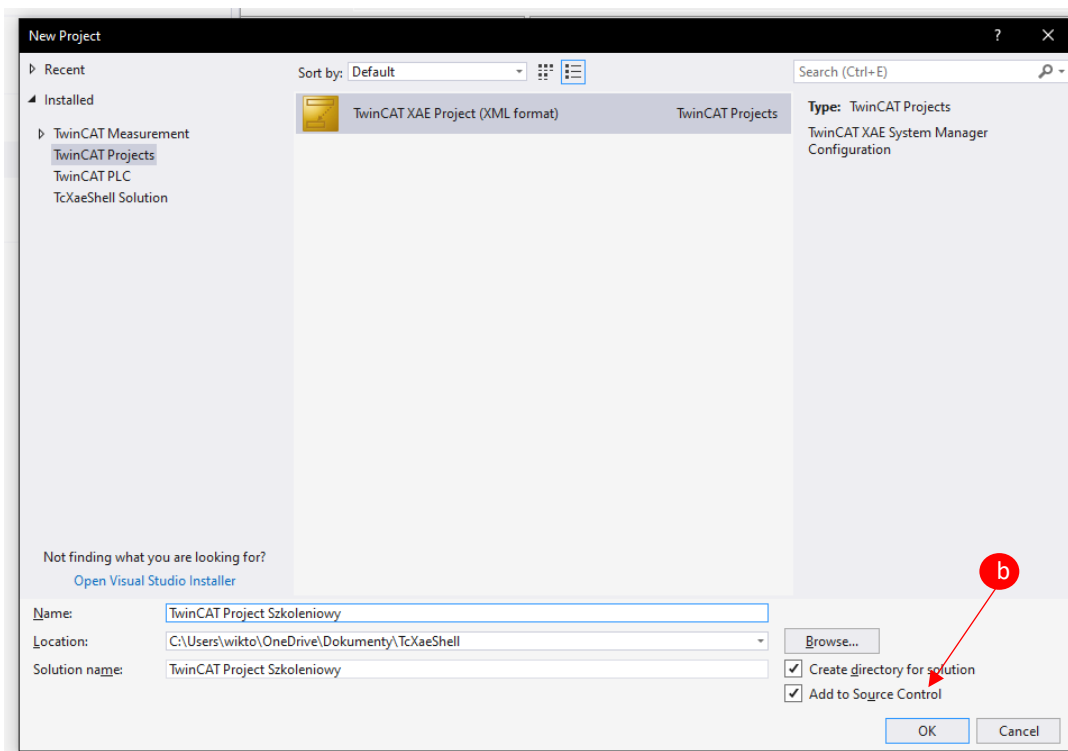
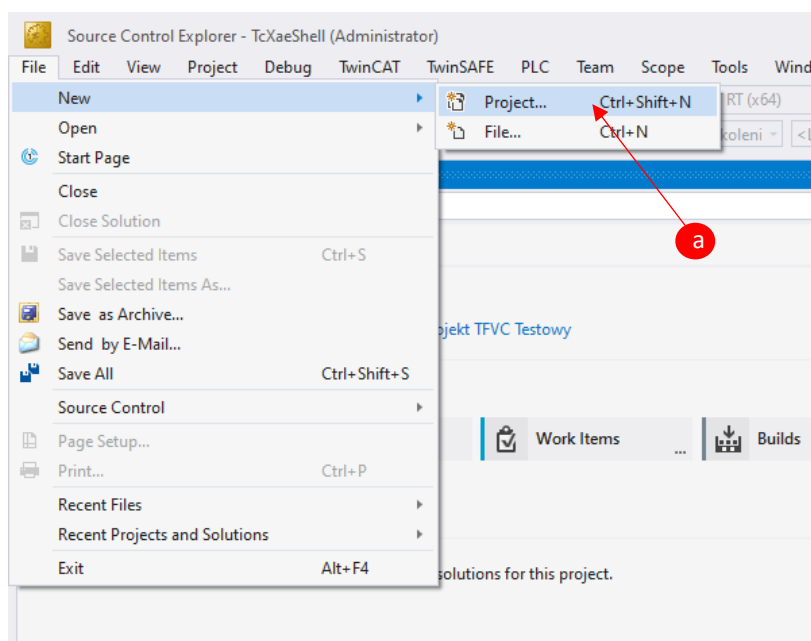


Po ustanowieniu połączenia należy zmapować obszar w przestrzeni lokalnej komputera w celu synchronizacji danych z serwerem [j].

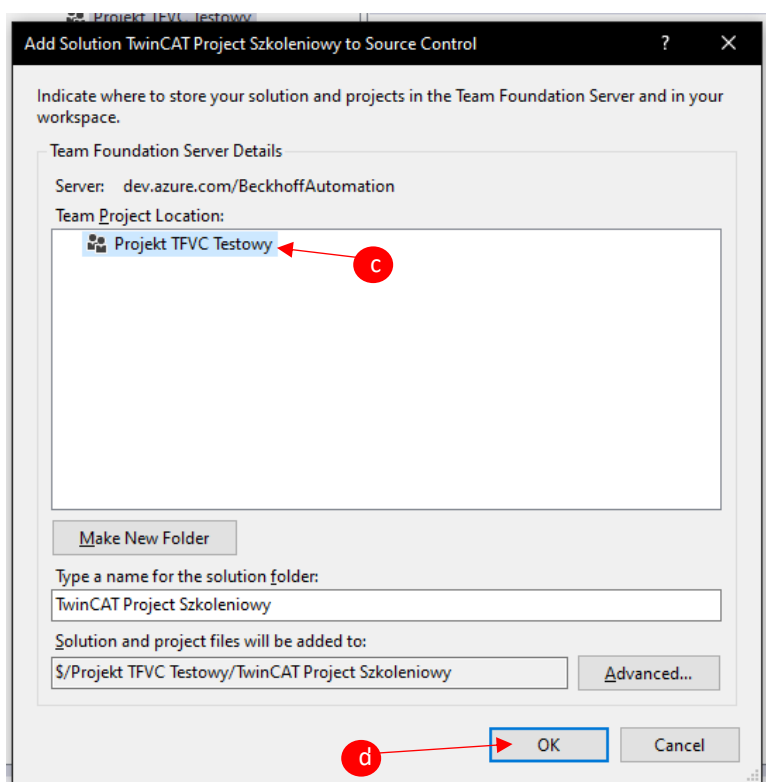


6.2.2 Wprowadzenie nowego rozwiązania na serwer Azure używając TFVC

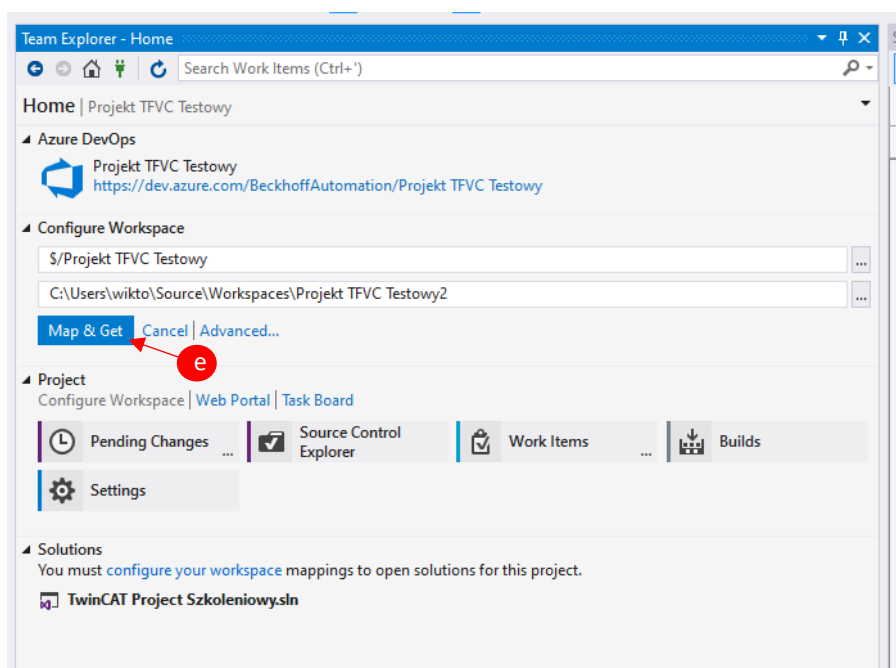
Będąc już połączonym z serwerem tworzymy nowy projekt [a] zaznaczając opcję dodania projektu do kontroli źródła [b].



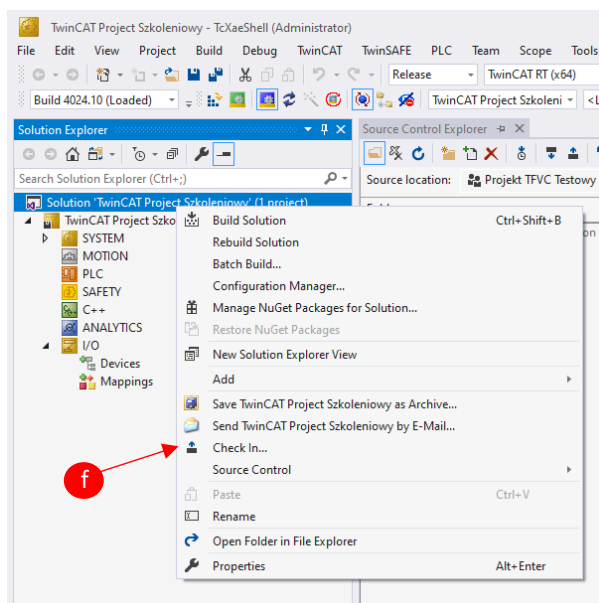
Oprogramowanie w następnej operacji pyta o wybranie miejsca docelowego repozytorium projektu. Wybieramy interesujący użytkownika projekt platformy Azure [c] a następnie zatwierdzamy wybór [d].



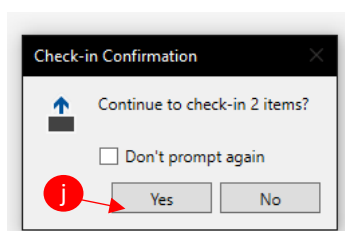
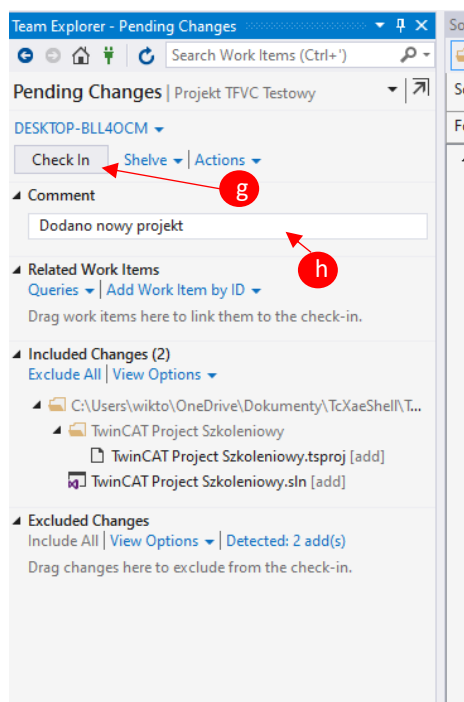
Po utworzeniu projektu należy zmapować miejsce występowania plików w celu dalszej synchronizacji, odbioru bądź też wysyłania plików [e].



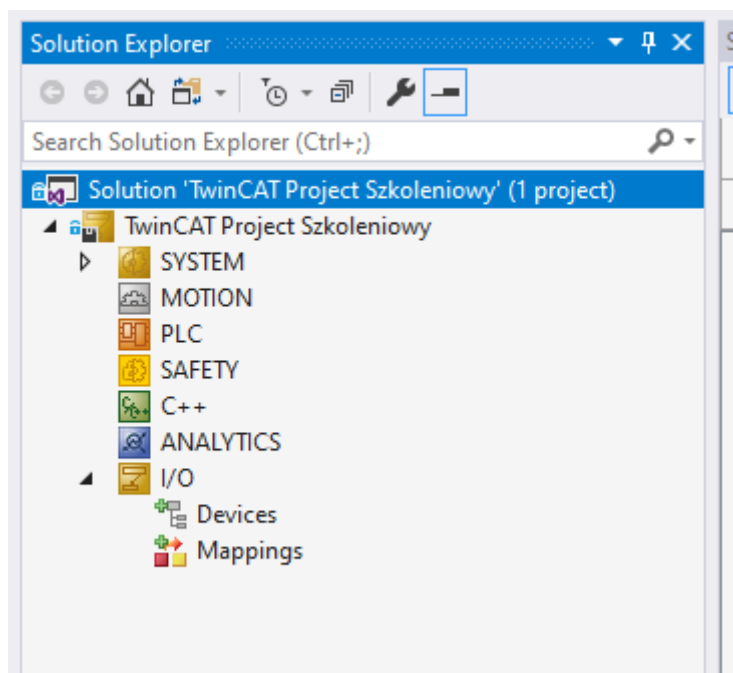
Nowo utworzony projekt znajduje się lokalnie na komputerze. W celu wysłania go do chmury należy wybrać Check in... z drzewka rozwijanego prawym przyciskiem myszki [f].



Przed samym wysłaniem należy zatwierdzić wprowadzenie zmian na serwerze [g][j]. Przed wysłaniem można jednocześnie wprowadzić komentarz opisujący co zostało zmienione w wersji [h].



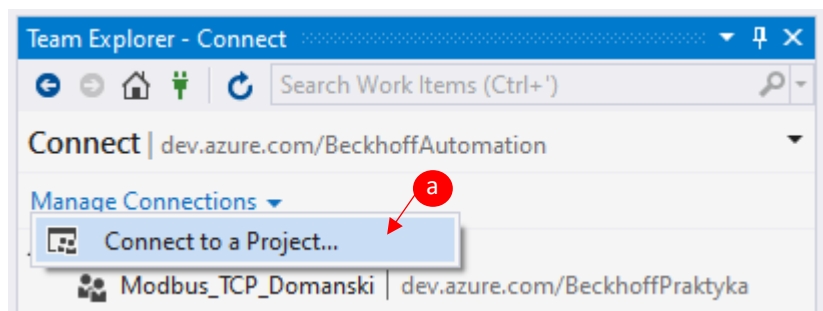
Po zaktualizowaniu plików na serwerze, przy nazwie projektu w drzewku nawigacji można zauważyć znak kłódki oznaczający, że wersja na komputerze jest najnowszą wersją projektu umieszczoną w chmurze.



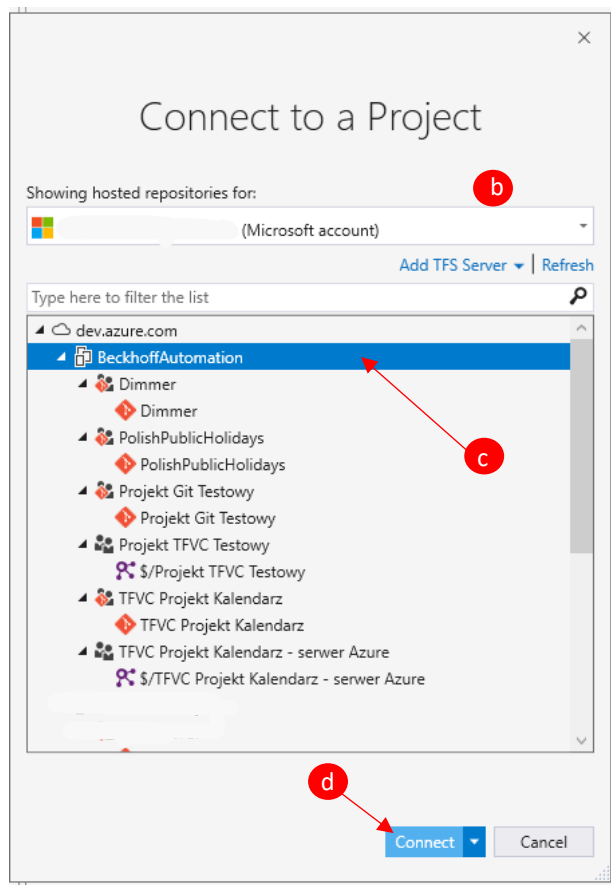
6.2.3 Dodawanie istniejącego projektu do repozytorium TFVC AZURE

Oprogramowanie TcXaeShell pozwala na dodanie już istniejącego projektu do repozytorium TFVC. W celu wysłania projektu na serwer należy wpięrow posiadać utworzony projekt w systemie TFVC na serwerze Azure. Rozwiązanie istniejące na serwerze lokalnym zostanie umieszczone docelowo w obszarze nowego repozytorium na serwerze Azure w istniejącym już projekcie. Informację jak utworzyć projekt w wersji TFVC na serwerze Azure można znaleźć pod nagłówkiem [7.2](#). Jeżeli projekt został utworzony/istnieje na serwerze Azure należy w następnym kroku otworzyć istniejący projekt oraz połączyć się z organizacją na serwerze Azure. Połączenie z organizacją można uzyskać w następujący sposób :

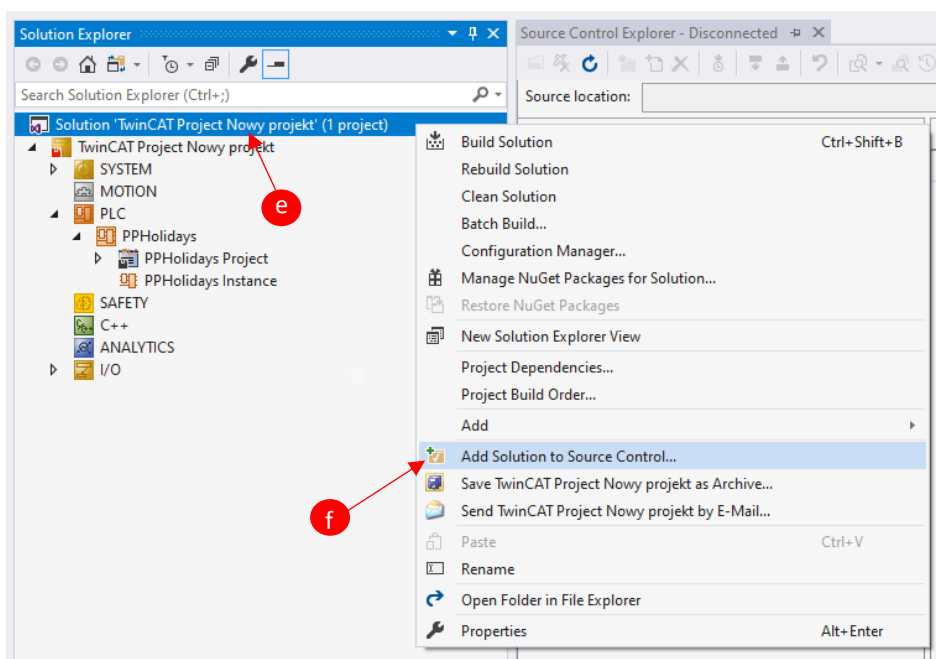
- W obszarze Team Explorera wybrać Manage Connections -> Connect to a Project [a].



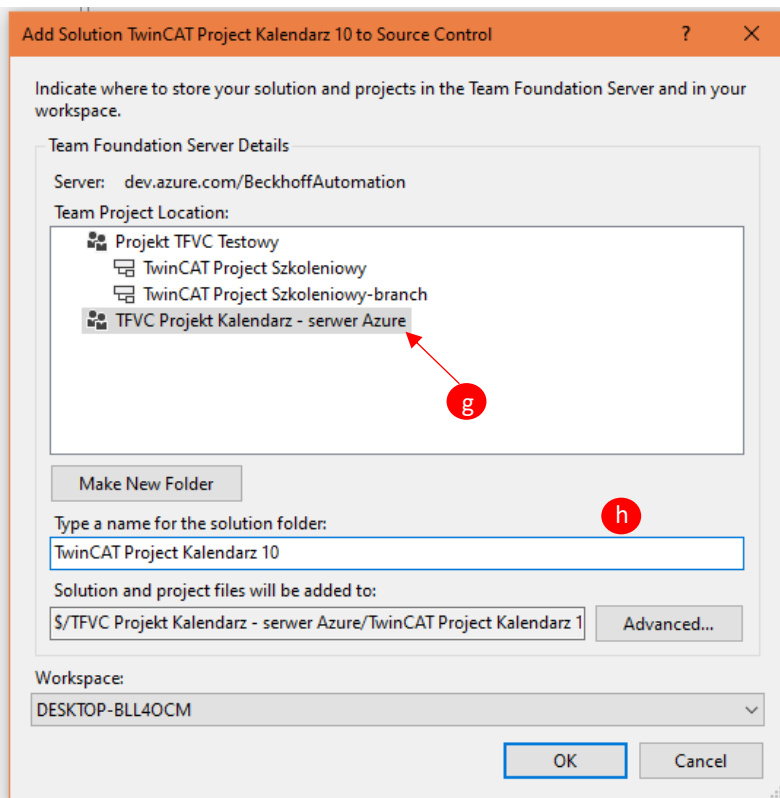
- Zalogować się na wybrane konto posiadające prawa edycji projektu w wybranej organizacji [b].
- Zaznaczyć wybraną organizację [c] oraz użyć przycisku Connect [d]



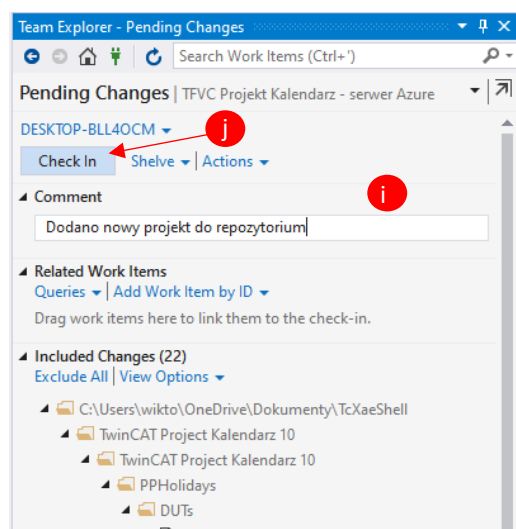
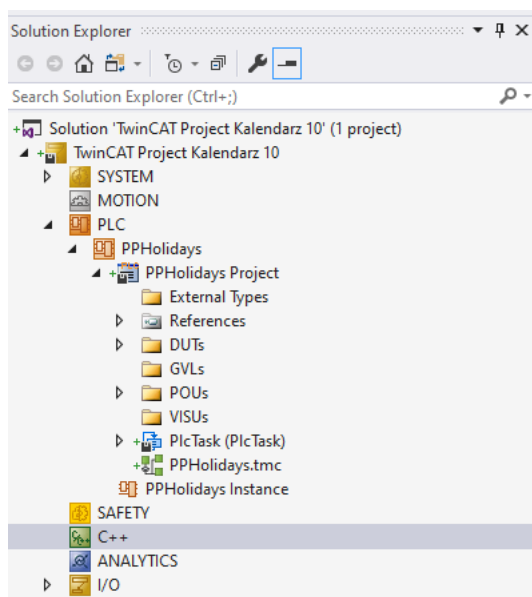
- Należy dodać obecny projekt do obszaru kontroli źródła poprzez wybranie PPM projektu [e] a następnie wybranie przycisku [f]



- Następnie należy wybrać istniejący projekt na serwerze Azure[g]
- Oraz wprowadzić nazwę nowego repozytorium które zostanie utworzone na serwerze [h]

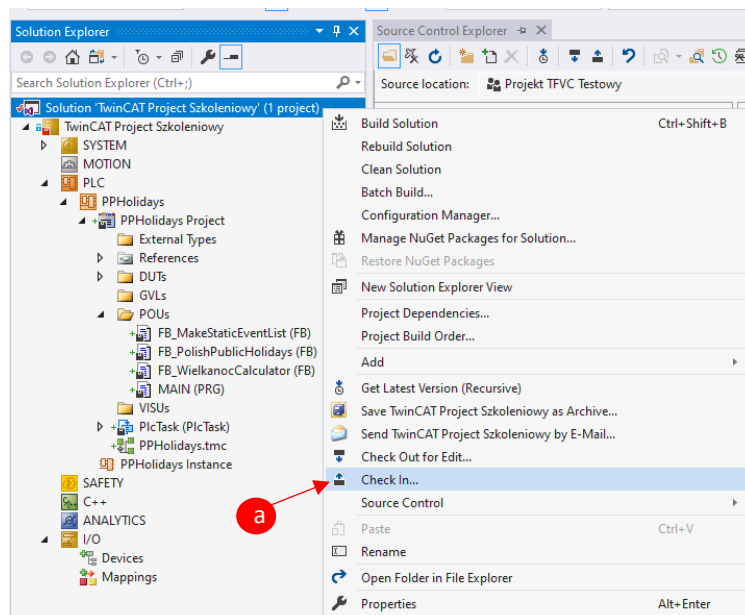


Jak można zauważyć projekt został oznaczony znakiem + jako obiekt nowo utworzony i nieistniejący jeszcze w obszarze serwera. W celu wysłania istniejącej wersji na serwer należy zatwierdzić zmiany w obszarze Team Explorer->Pending Changes. Po możliwym zakomentowaniu [i] i kliknięciu przycisku **Check In** [j] zmiany zostaną zatwierdzone i wysłane na serwer Azure.

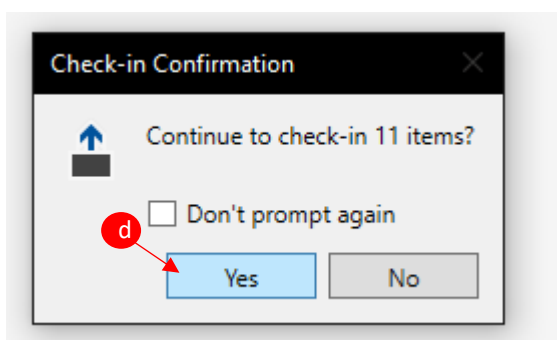
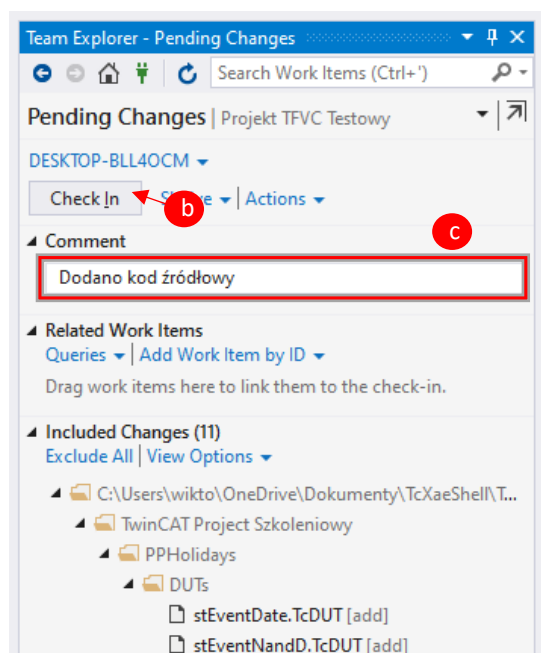


6.3 Aktualizacja projektu na platformie Azure przy wykorzystaniu TFVC

Wprowadzanie zmian w elementach projektu wprowadza różnicę w między wersją programu znajdującą się na komputerze jak i na serwerze. Nowo dodane pliki do projektu są oznaczone znakiem + obok ich nazwy w drzewku nawigacji. Niezmienione pliki pozostają oznaczone kłódką, natomiast edytowanie znakiem ✓ mówiącym o wprowadzonych zmianach lokalnych różniących się od wersji na serwerze. Do celu aktualizacji projektu z wersją w chmurze należy ponownie użyć przycisku Check in... w celu zatwierdzenia zmian [a]. Dostęp do elementu funkcyjnego można uzyskać poprzez kliknięcie prawym przyciskiem myszy na nazwę projektu.

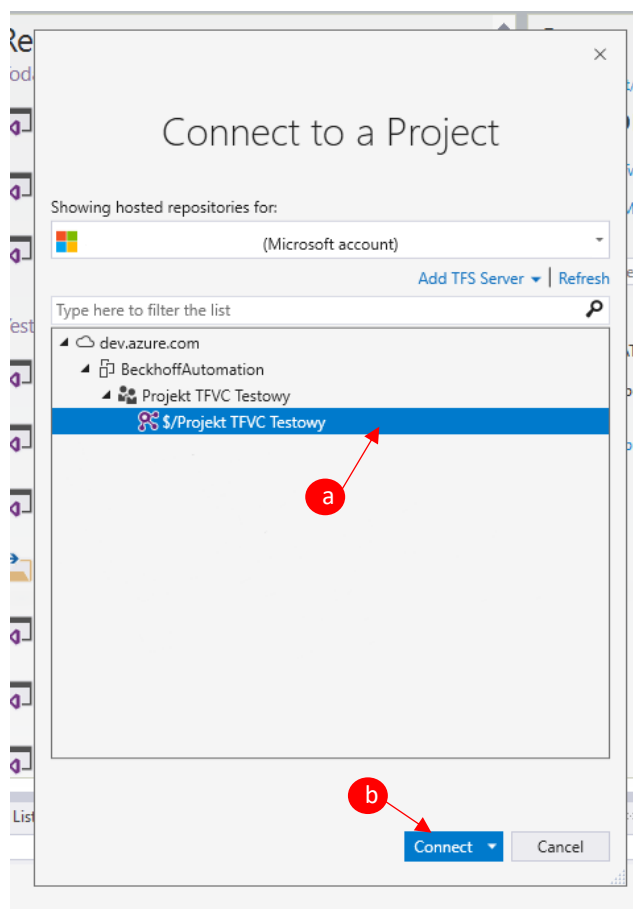


Następnie należy potwierdzić chęć wprowadzenia zmian w plikach na serwerze Azure[b][d]. Jednocześnie można uwzględnić podczas przesyłania komentarz określający jakie zmiany dokonano w projekcie[c]. Po zgraniu plików na serwer, ikony oznaczeń stanu plików zmieniają swój stan na „kłódkę” potwierdzając aktualność wersji

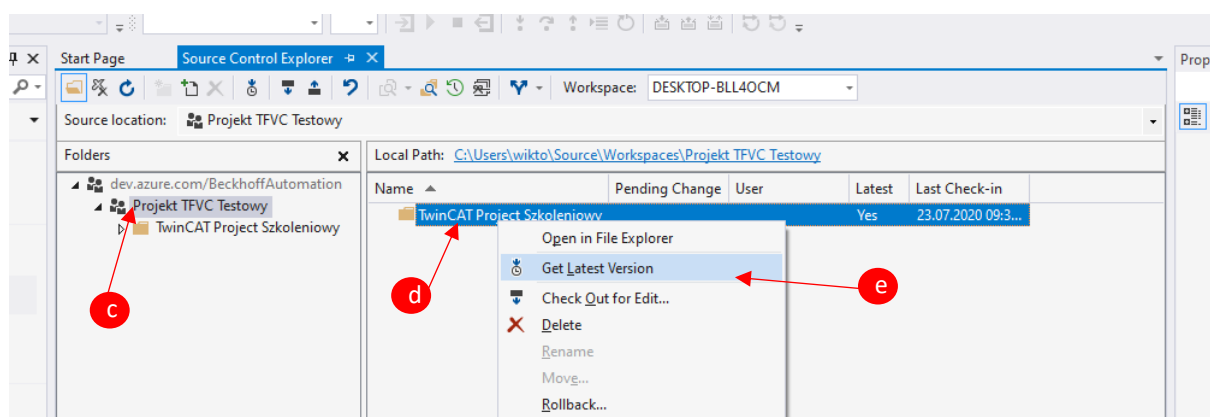


6.4 Pobranie projektu z platformy Azure przy użyciu TFVC

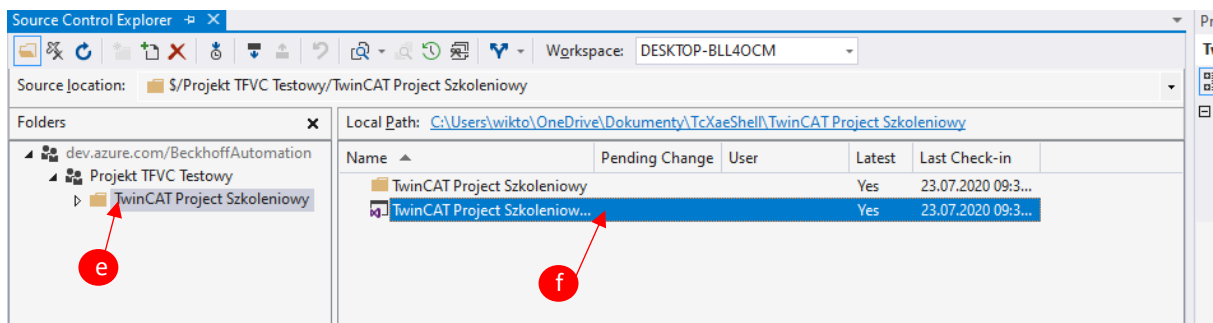
W celu pobrania najnowszej dostępnej wersji na nowej stacji roboczej należy połączyć się z serwerem Azure a następnie wybrać interesujące nas repozytorium projektowe[a][b].



Następnie w oknie kontroli źródła wybieramy pliki źródłowe[c][d] oraz pobieramy je do najnowszej wersji[e].



Następnie w celu edytowania plików otwieramy rozwiązanie[f][g]



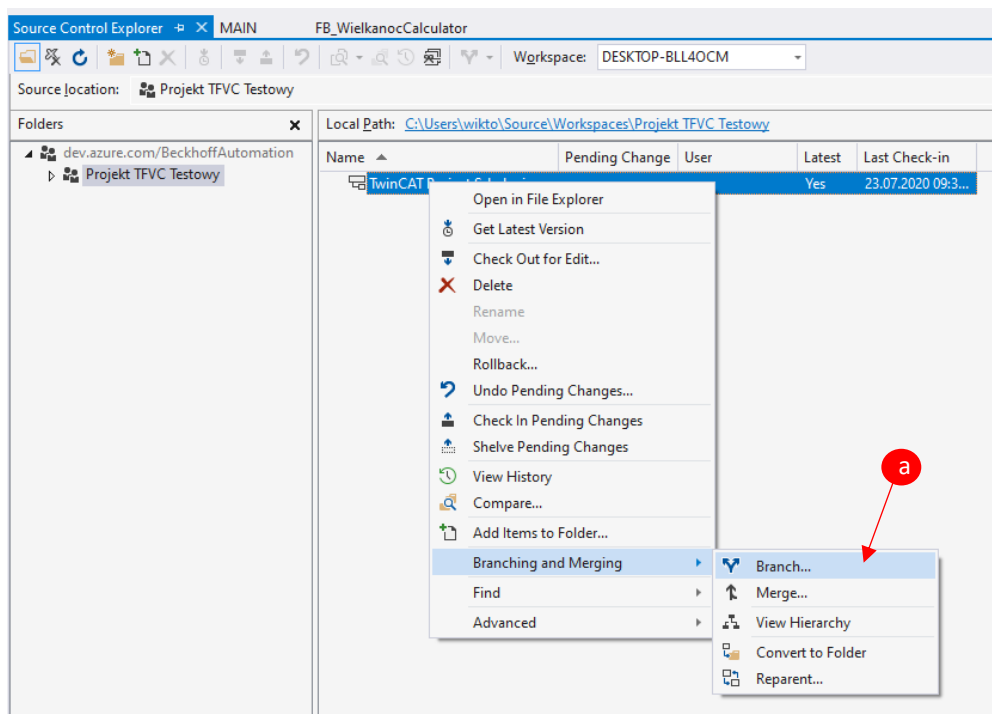
6.5 Tworzenie odgałęzień oraz łączenie gałęzi.

Operując na głównej gałęzi podczas edycji kodu bez używania odgałęzień narażamy główny kod programowy na utratę danych na serwerze oraz uszkodzenie programu. Podczas realizacji wspólnego projektu w jednej chwili wiele osób może edytować główny kod programowy. W celu uniknięcia nadpisania kodu źródłowego w sposób niepożądany używa się odgałęzień programowych.

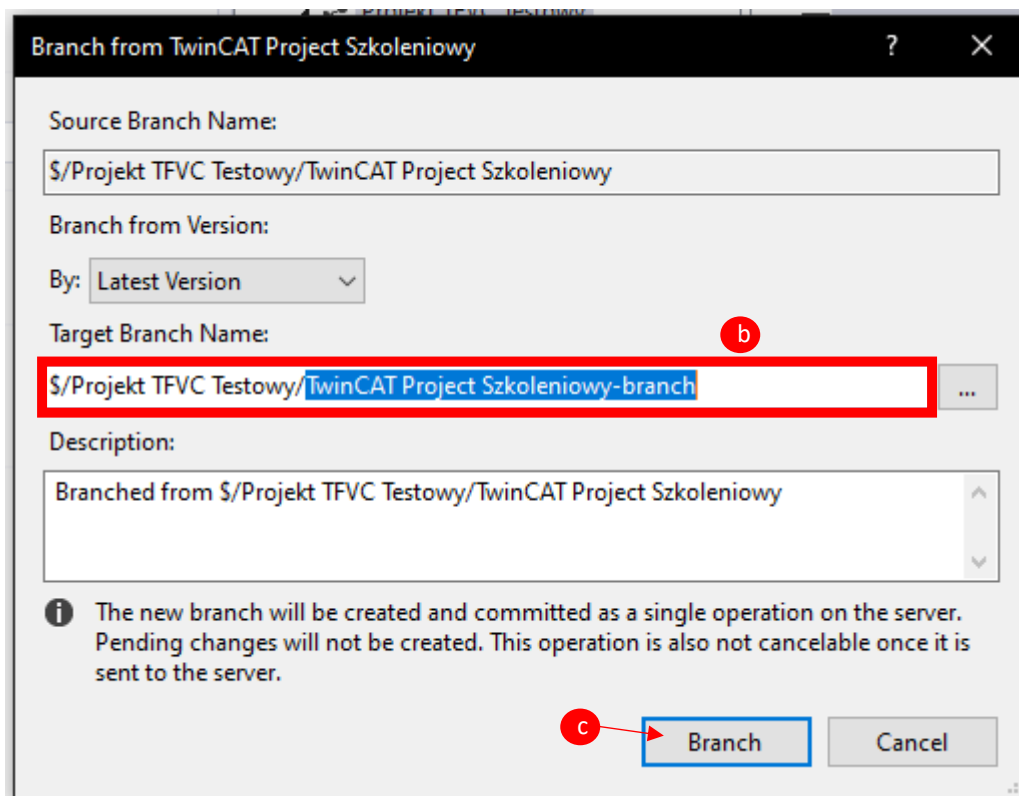
Odgałęzienie (tzw. Branch) - Kopia źródłowego kodu przeznaczona do dalszej edycji przez danego użytkownika. Tworząc odgałęzienie pracujemy na lokalnej kopii źródłowej gałęzi programowej. Pozwala to na modyfikowanie, dodawanie nowych elementów testowanie jak i naprawę błędów programowych niezależnie od obecnego stanu gałęzi głównej. Odgałęzienie umożliwia utworzenie danej wersji programu przeznaczonej do edycji w konkretnym celu bez ingerencji w gałąź źródłową w trakcie realizacji. Zaleca się, żeby jedno odgałęzienie programu przypadało na jednego użytkownika w trakcie edycji programu lub też na jedno sprecyzowane zadanie. Po osiągnięciu zamierzonego efektu gałąź można złączyć z gałęzią główną w celu dodania/zmodyfikowania funkcjonalności programu przy użyciu komendy **Merge**. Pozwala ona na porównanie obu wersji programowych i sprawdzenie kompatybilności. Operując na gałęziach można w ten sposób łatwo uniknąć wprowadzenia niepożądanych zmian w kodzie edytowanym przez inne osoby.

6.5.1 Tworzenie odgałęzienia

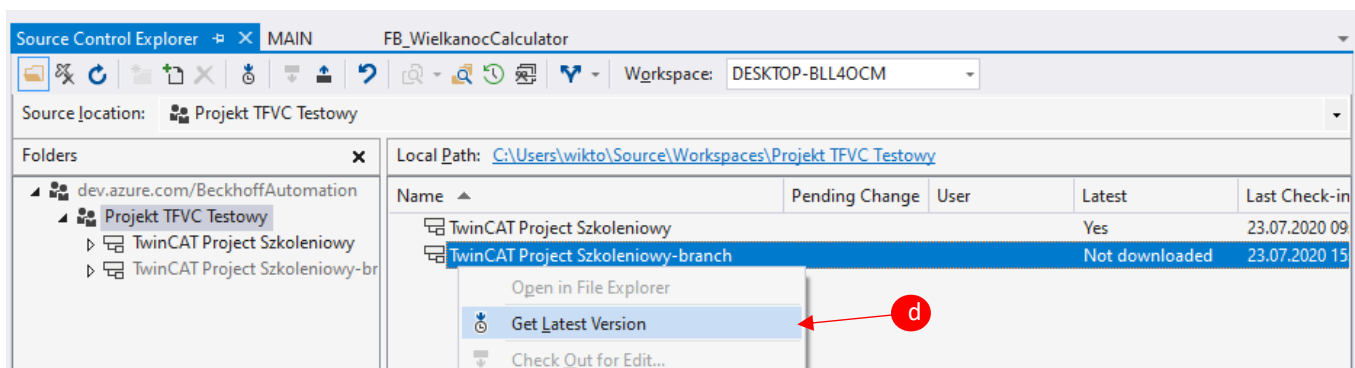
Aby dodać nową gałąź, używając prawego przycisku myszy na wybranym przez nas projekcie w eksploratorze kontroli źródła klikamy funkcję **Branch...** [a]



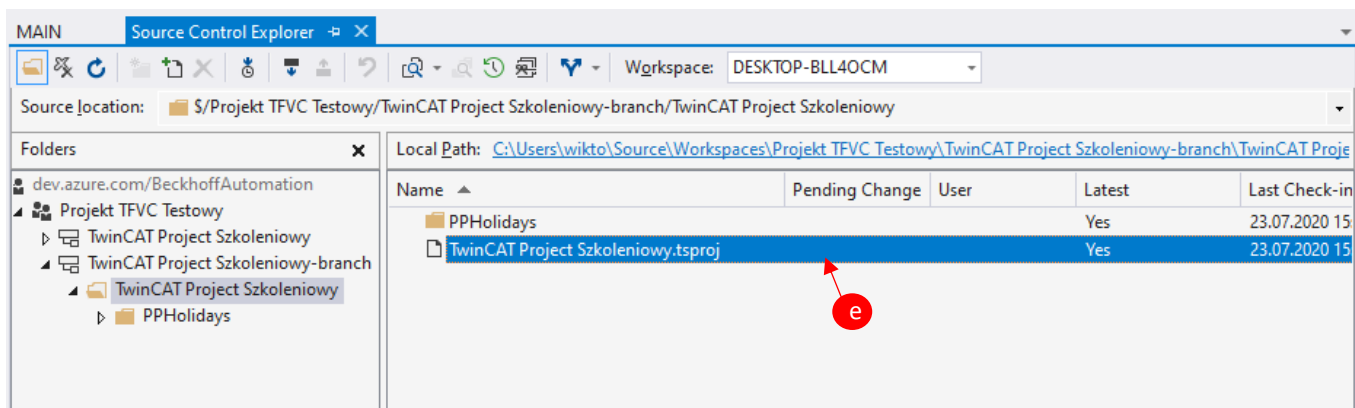
W następnym kroku wybieramy nazwę nowego odgałęzienia[b] a następnie zatwierdzamy wybór[c].



Gałąź jest tworzona w wersji sieciowej na serwerze Azure. Aby uzyskać do niej dostęp należy pobrać najnowszą wersję z sieci[d].



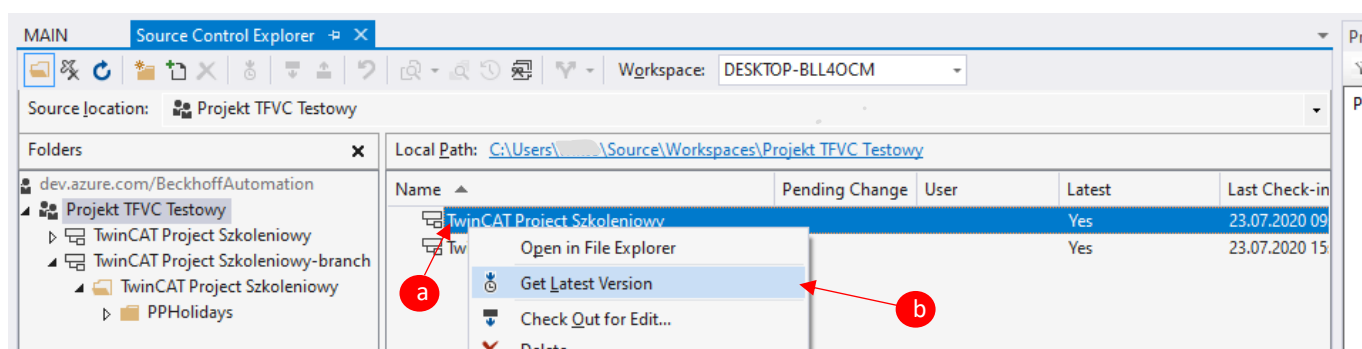
W celu edycji odgałęzienia należy otworzyć rozwiązanie dla gałęzi w eksploratorze kontroli źródła[e].



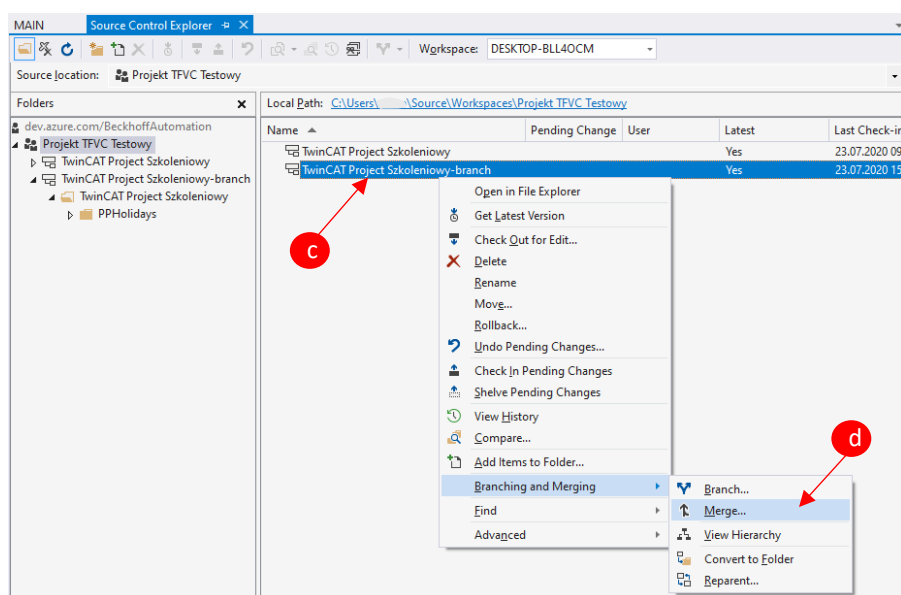
6.5.2 Łączenie gałęzi

Posiadając odgałęzienia programowe w realizowanym projekcie możemy zastosować operację łączenia. Realizuje ona dodanie nowych plików do gałęzi docelowej jak i porównanie zmian w kodzie. Używanie narzędzia łączenia odgałęzień pozwala na zabezpieczenie wersji programowych obecnych w chmurze przed wymuszonym nadpisaniem kodu. Podczas realizacji projektu wiele osób może zmienić kod gałęzi docelowej w trakcie realizacji projektu. Nieuwzględnienie zmian może prowadzić do utraty części zrealizowanej funkcjonalności jak i do uszkodzenia realizowanego projektu.

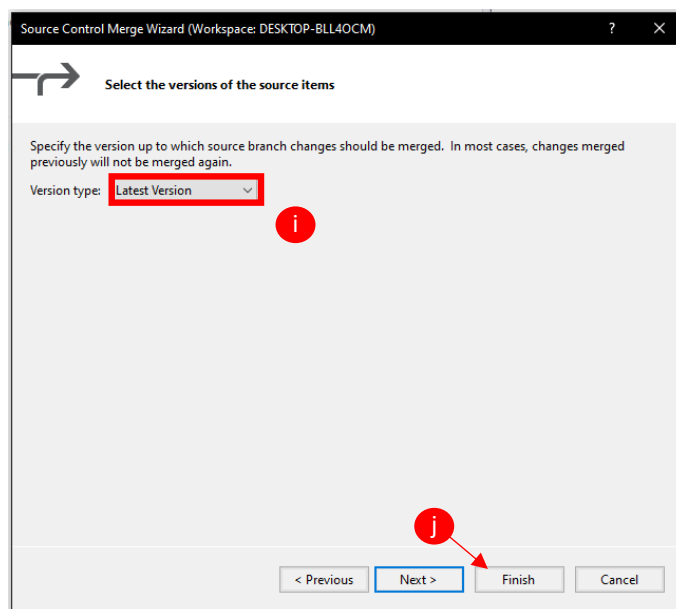
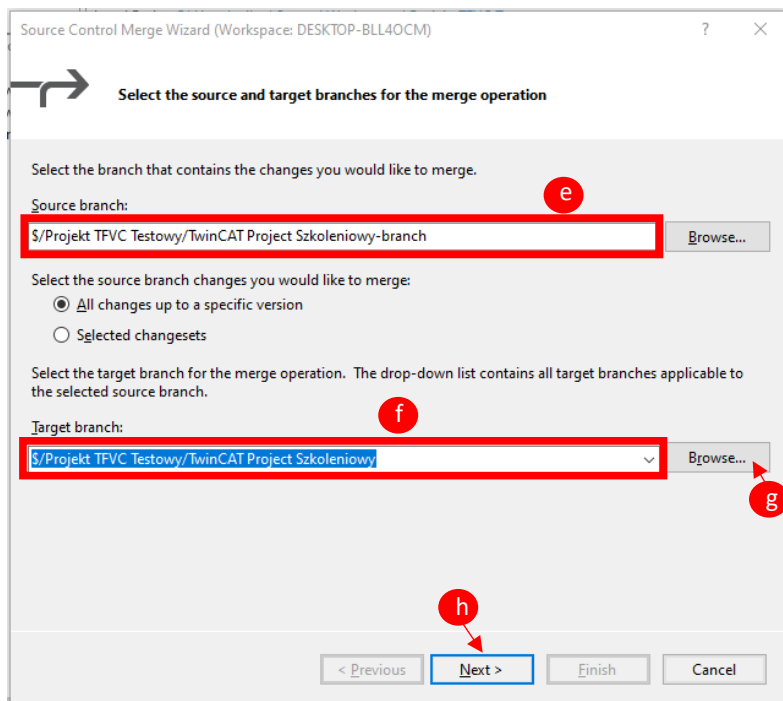
Przed przeprowadzeniem łączenia należy się upewnić, że pobrana jest najnowsza wersja gałęzi docelowej. Używając narzędzia Source Control Explorer możemy sprawdzić aktualność docelowej gałęzi. W przypadku nieposiadania najnowszej wersji należy ją pobrać używając PPM na gałęzi docelowej[a], a następnie klikając Get Latest Version [b].



TcXaeShell pozwala na łączenie gałęzi z poziomu środowiska programistycznego. W celu połączenia gałęzi należy w obszarze narzędzia Source Control Explorer gałąź przeznaczoną do łączenia [c]. Używając PPM na gałęzi wybieramy funkcję Merge... z okna [d].

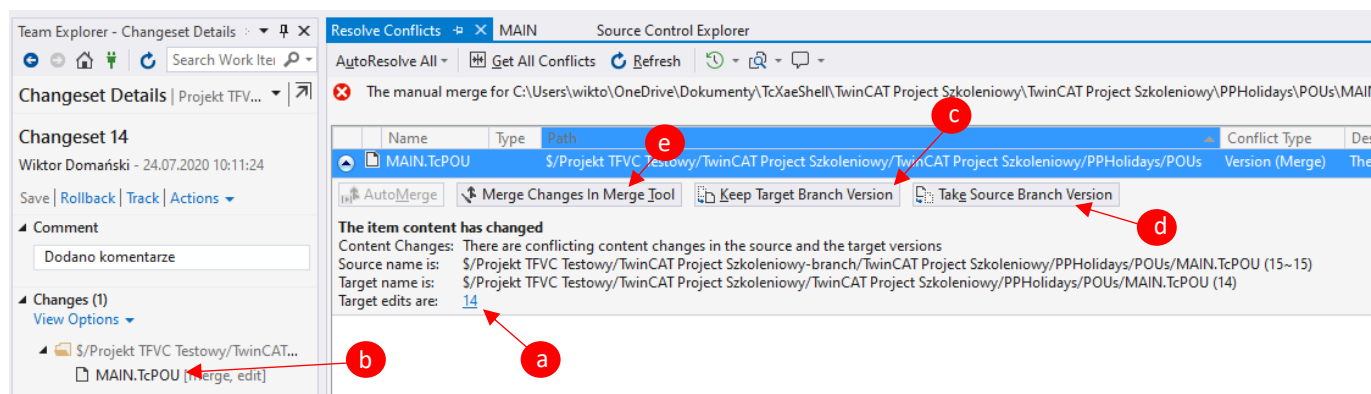


W następnym kroku Merge Wizard wymaga podania ścieżki gałęzi źródłowej [e] oraz docelowej [f]. Wizard domyślnie proponuje istniejące ścieżki. W celu zmiany ścieżki docelowej używamy przeglądarki plików Browse. Oprogramowanie domyślnie operuje w obszarze repozytorium zarządzając wersjami programowymi utworzonymi w chmurze. Po przejściu do następnego etapu [h] oprogramowanie pyta którą wersję gałęzi źródłowej ma dodać [i]. Zalecane jest używanie najnowszej wersji lub ostatniej stabilnej oprogramowania. Kończymy łączenie elementów [j].

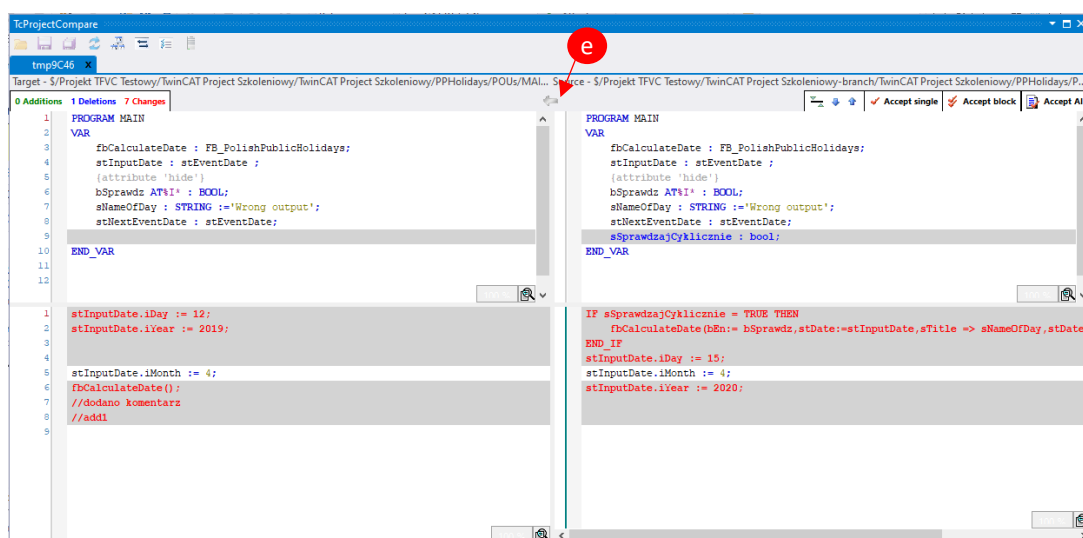


6.6 Rozwiązywanie kolizji plików

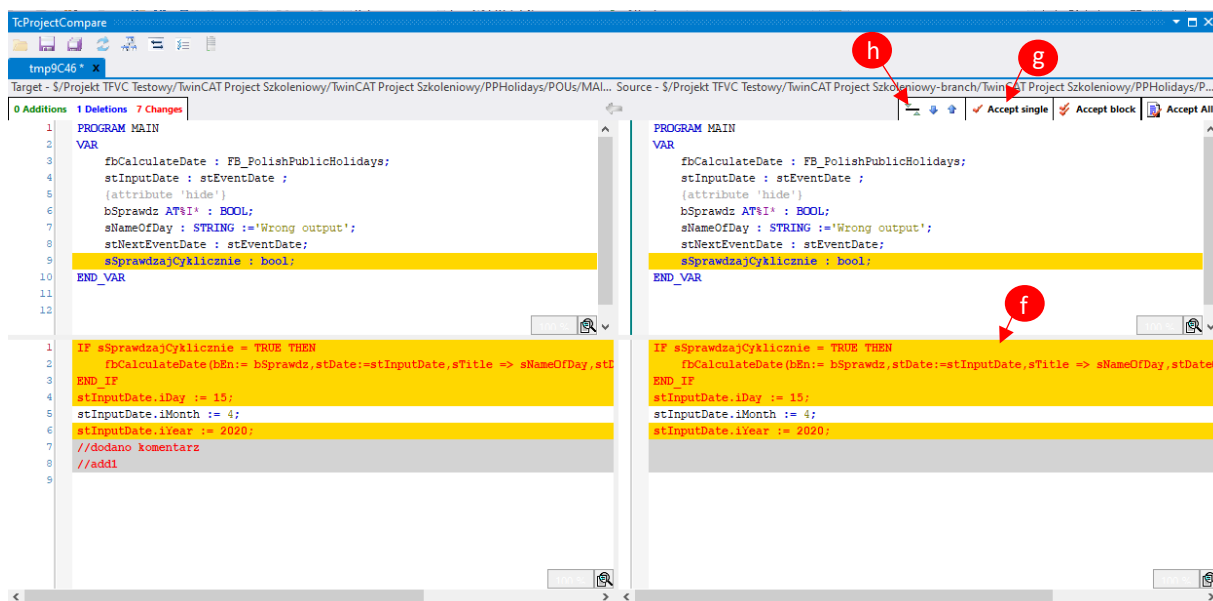
Podczas łączenia gałęzi może wystąpić konflikt. W celu przejrzenia obecnych różnic w plikach, edycji i podmiany klikamy w numer wprowadzanej edycji [a] oraz otwieramy plik różnicowy [b]. W celu podmiany bez edycji używamy Keep Target Branch [c] jeżeli chcemy zachować kod docelowy lub Take Source Branch [d] jeżeli chcemy podmienić plik docelowy na plik źródłowy. Oprogramowanie pozwala na kontrolowane łączenie plików przy pomocy wbudowanego narzędzia TwinCAT Project Compare. Aby wywołać narzędzie dla wybranej kolizji należy użyć **Merge Changes In Merge Tool** [e].



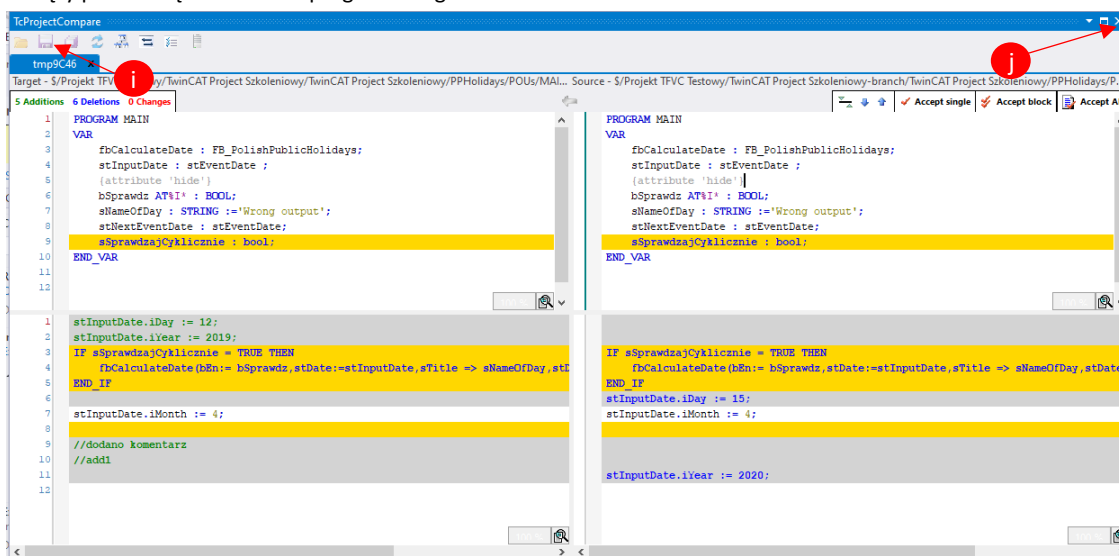
Narzędzie TcProjectCompare pozwala na edycję kodu w celu uzyskania zgodności. Edycja jest ograniczona do wybrania linijek kodu które mają być zastąpione przez wersję wypychaną. Kierunek nadpisywania zmian określa strzałka [e] widoczna nad oknami.



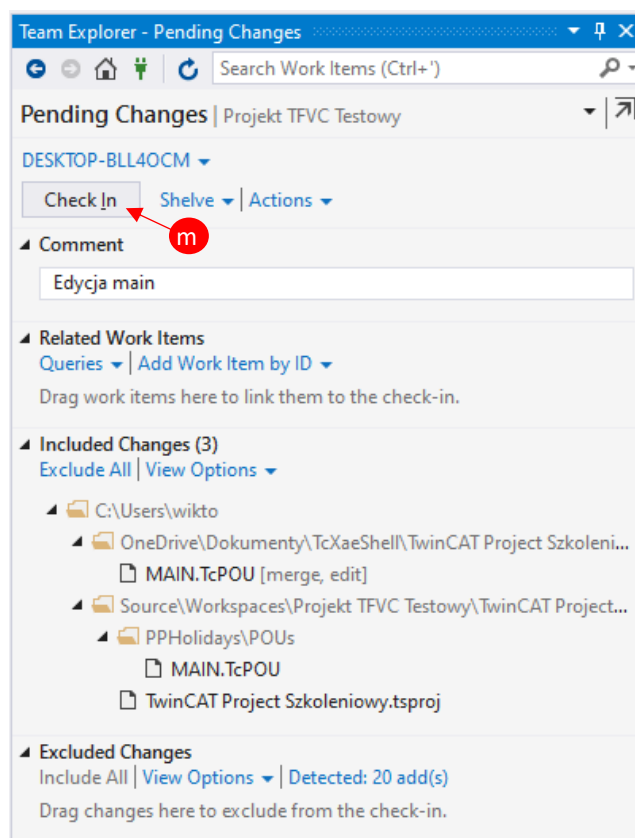
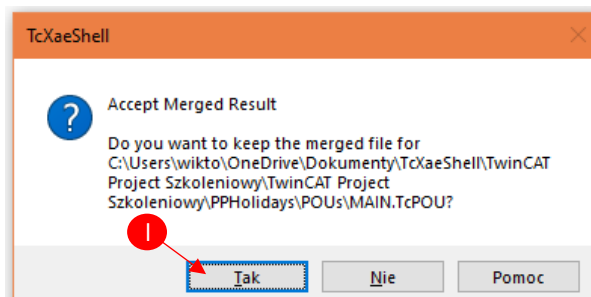
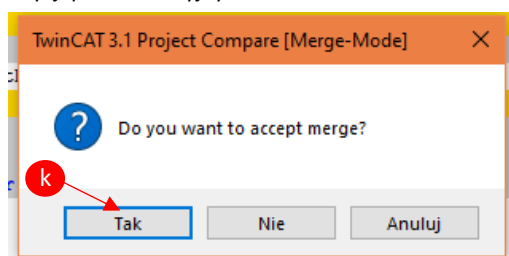
Wybór poszczególnych linii programowej przebiega poprzez oznaczenie wybranej linii kursorem [f] a następnie zatwierdzeniem zmiany przyciskiem Accept single [g] lub też klawiszem <Spacja>. Jak można kod docelowy został zastąpiony w wybranych liniach przez kod źródłowy. Oprogramowanie pozwala również na ułożenie naprzemiennie nowych zmian w kodzie w stosunku do istniejącego kodu w celu dodania nowych linii bez ingerencji w poprzednie. W celu uzyskania tej funkcjonalności należy wybrać ikonę [h].



Po przeprowadzeniu odpowiednich zmian możemy je zastosować poprzez zapisanie [i] porównania a następnie zamknięcie okna [j]. Kod wyprowadzony po lewej stronie okna (docelowy) zostanie wysłany na serwer w utworzonej postaci. Kod niezastosowany (na szaro po prawej) zostanie pominięty podczas łączenia bloku programowego.

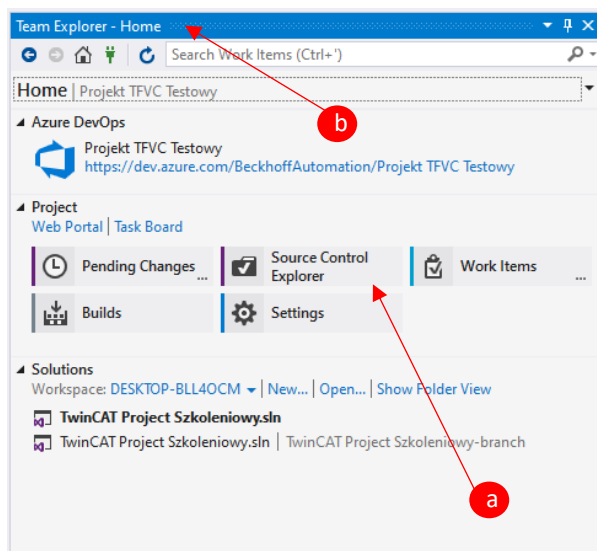


Oprogramowanie w następnych krokach pyta nas czy chcemy zastosować wprowadzone zmiany. W celu zastosowania zmian należy zaakceptować okienka [k] [l] oraz następnie ponownie zatwierdzić wprowadzenie zmian na serwerze Check in [m]. Oprogramowanie zostanie wówczas przesłane w najnowszej wersji po edycji plików kolizyjnych.

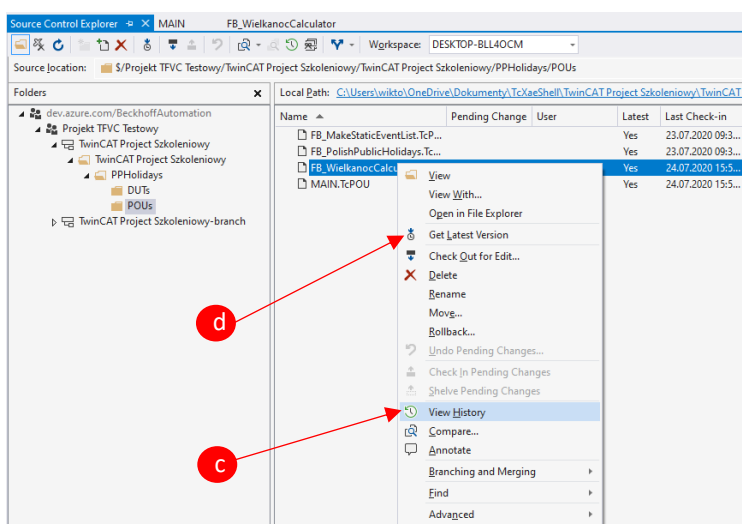
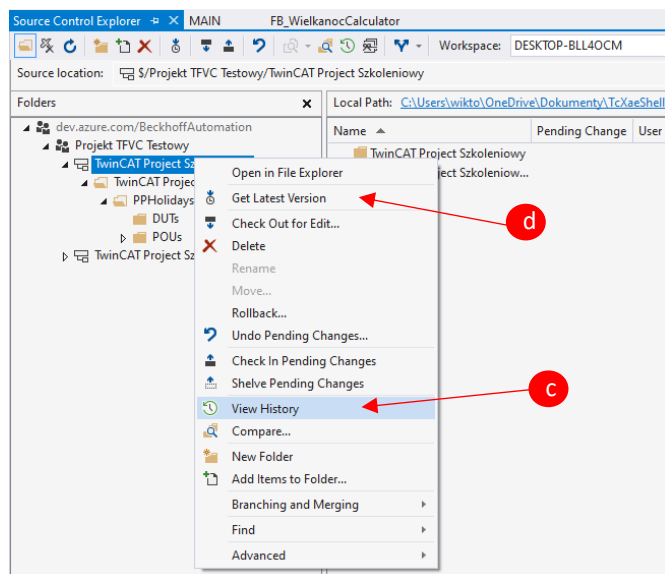


6.7 Sprawdzanie historii wersji poszczególnych plików

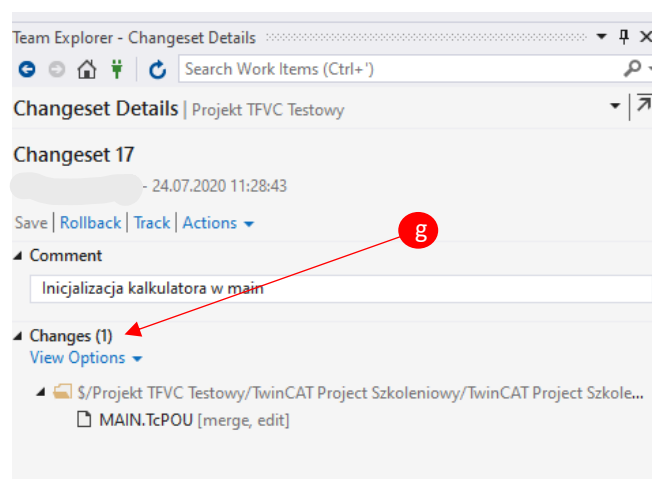
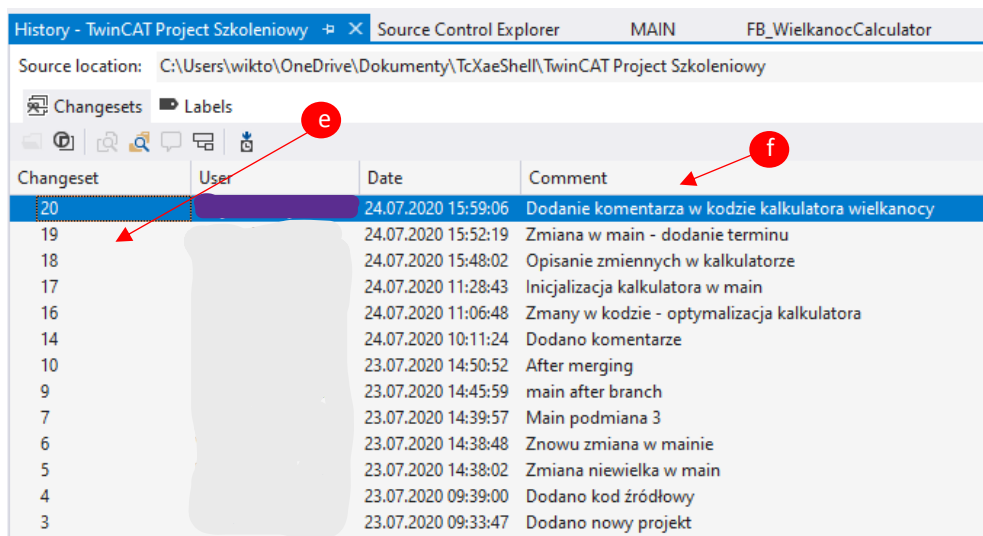
System TFVC pozwala na sprawdzenie historii aktualizacji poszczególnych plików/folderów/odgałęzień/przestrzeni roboczych na dowolnym etapie prac. Do sprawdzania wersji poszczególnych plików służy wbudowane w TcXaeShell narzędzie Source Control Explorer [a]. Narzędzie jest dostępne przez okno Team Explorer [b].



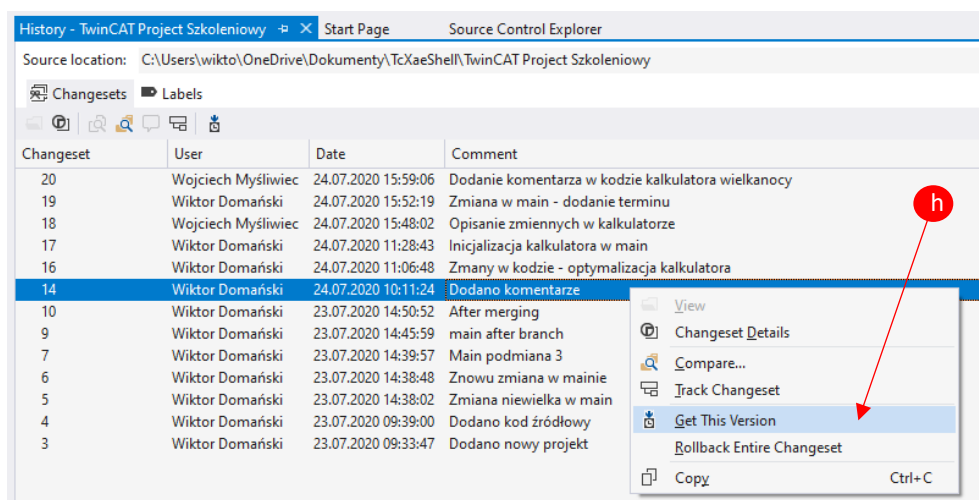
Klikając PPM na dowolny element drzewa projektu możemy wywołać opcję View History dla danego pliku lub projektu [c]. Jednocześnie możemy pobrać najnowszą wersję pliku/folderu w identyczny sposób [d].



Otwierając historię pliku/folderu otrzymujemy historię wypchanych wersji zmian dokonanych przez użytkowników [e]. Jednocześnie mamy podgląd komentarzy opisujących poszczególne zmiany poczynione na danym etapie programistycznym [f]. Otwierając jedną z wersji mamy możliwość podglądu listy zmienionych plików [g].

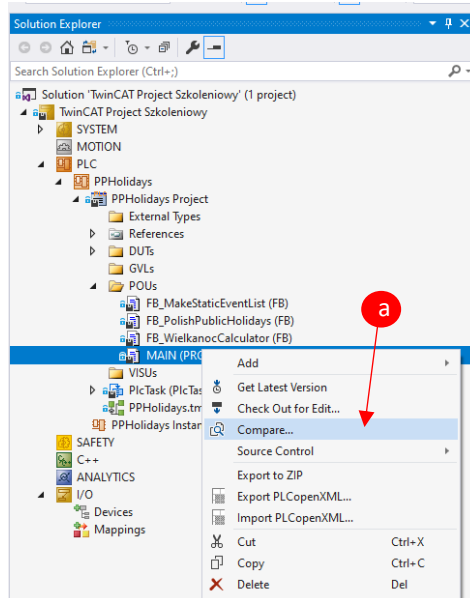


Jeżeli użytkownik potrzebuje do realizacji projektu danej wersji programu poprzednio zrealizowanej lub chce cofnąć zmiany do danego punktu w czasie może pobrać dowolną zatwierdzoną wersję programu używając funkcji Get This Version [h].

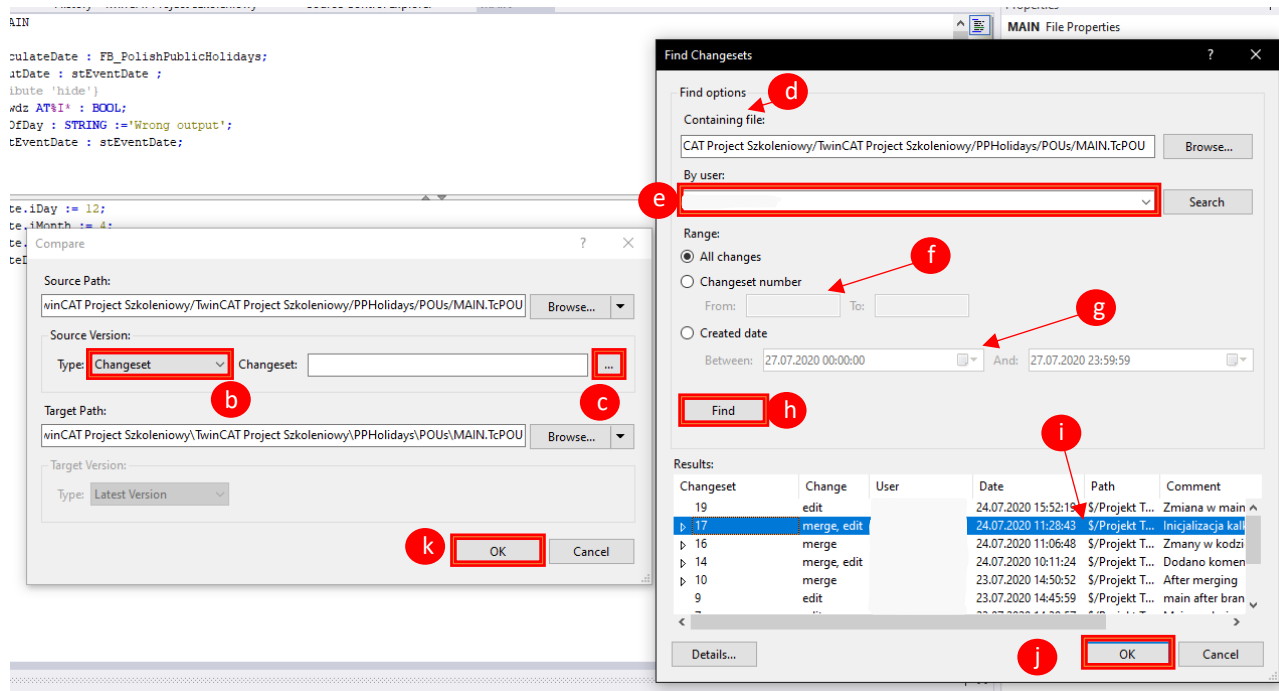


6.8 Porównywanie zmian w kodzie

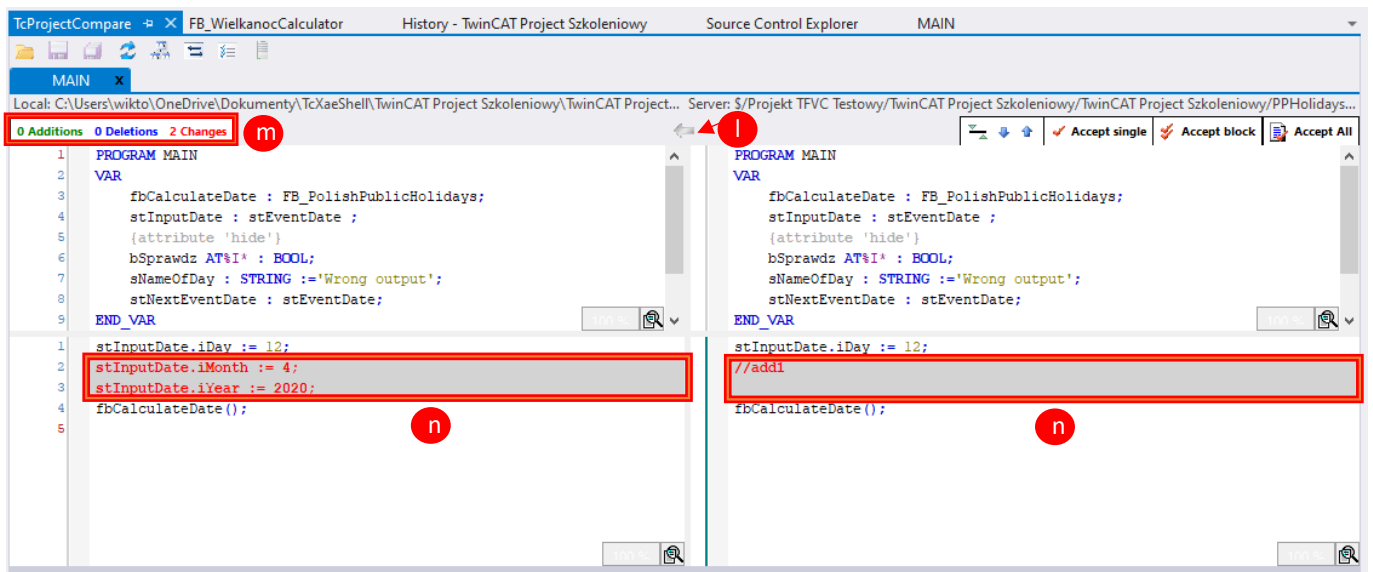
Środowisko programistyczne TcXaeShell używając systemu kontroli wersji TFVC ma możliwość porównania zmian w kodzie źródłowym dowolnego pliku. Porównanie może być przeprowadzone między dowolną wersją uwzględnioną w historii projektu. Możliwość porównania aktualnej wersji uzyskujemy poprzez użycie funkcji Compare... [a].



Operacja porównania pyta o podanie źródła porównania. Jako że porównujemy zmiany do ostatniej wersji źródłem powinien być wybrany changeset. W celu wybrania wersji programu z historii należy: [b] wybrać rodzaj źródła na Changeset, [c] wybrać edycję ścieżki, [d] wprowadzić zmiany informacyjne o poszukiwanej wersji w historii (np. który użytkownik [e], jaki zakres numerowy [f] lub data wprowadzenia zmiany [g]), [h] wyszukać wersję, wybrać interesujący changeset [i] oraz zatwierdzić [j] [k].



Realizacja porównania jest realizowana w narzędziu TcProjectCompare.



Opis:

- Kierunek przeprowadzonych zmian [l]
- przeprowadzone zmiany [m]
 - Nowe linie ●
 - Usunięcia linii ●
 - Zmiany w liniach ●
- obszar zmiany w kodzie [n]