

Postawy obsługi rozproszonego systemu kontroli wersji Git

Systemy kontroli wersji



1. Lokalne
 - RCS – Revision Control System
2. Scentralizowane
 - CVS – Concurrent Versios System
 - SVN – Subversion
3. Rozproszone
 - **Git**
 - Mercurial
 - Bazaar



<https://git-scm.com/video/what-is-version-control>

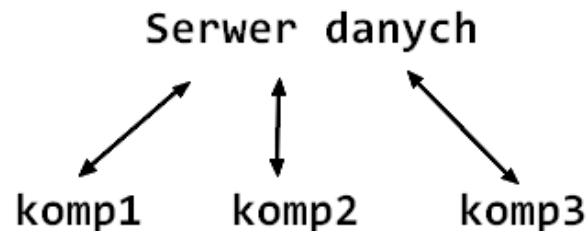
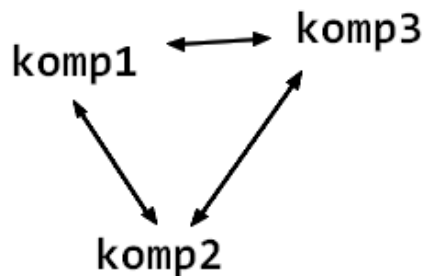
Rozproszona architektura



Ważną zaletą Gita jest **rozproszona architektura**. Polega ona na tym, że repozytoria znajdują się na wielu komputerach, a nie na jednym scentralizowanym serwerze (jak np. w przypadku SVN).



Inne oprogramowanie



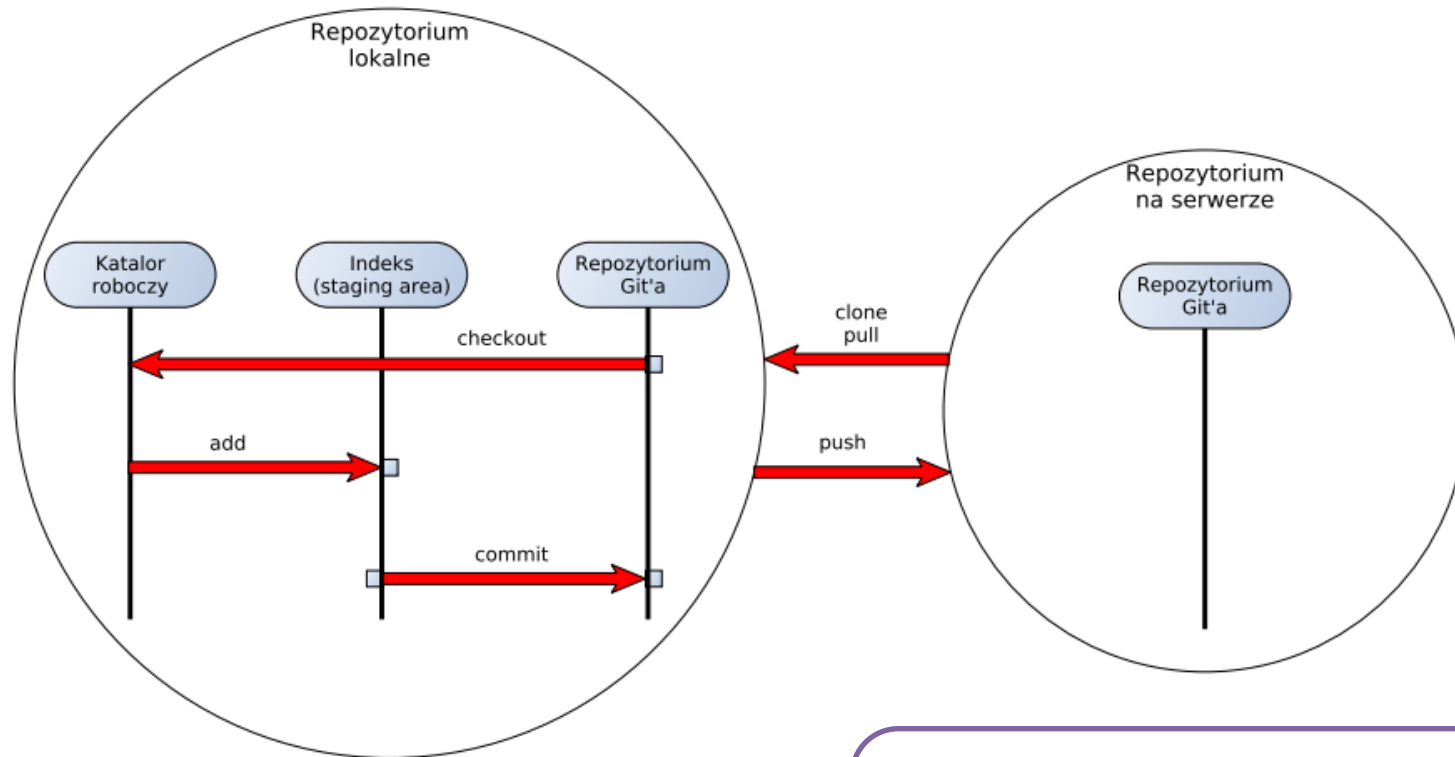
W przypadku repozytoriów scentralizowanych (takich jak np. SVN) **awaria serwera danych oznacza utratę wszystkich danych**. Informacji nie da się odtworzyć z komputerów użytkowników tych repozytoriów.

Git – podstawowe cechy



- **efektywna praca z dużymi projektami** - jest jednym z najszybszych systemów kontroli wersji
- **wsparcie dla protokołów sieciowych** - dane można wymieniać przez HTTP(S), FTP, rsync, SSH
- **każda kopia repozytorium to obraz całego projektu** - Git nie zapamiętuje zmian między kolejnymi rewizjami lecz kompletne obrazy (snapshots)
- **możliwość tworzenia oprogramowania z rozgałęzieniami**
- **tryb pracy off-line** - każdy pracuje na własnej kopii repozytorium, a następnie zmiany mogą być wymieniane między lokalnymi repozytoriami jak również serwerem.

Git – idea



<https://git-scm.com/video/what-is-git>

Git rozróżnia trzy stany dla plików:

- zatwierdzony
- zmodyfikowany
- śledzony

Git – idea cd.



Git rozróżnia trzy stany dla plików:

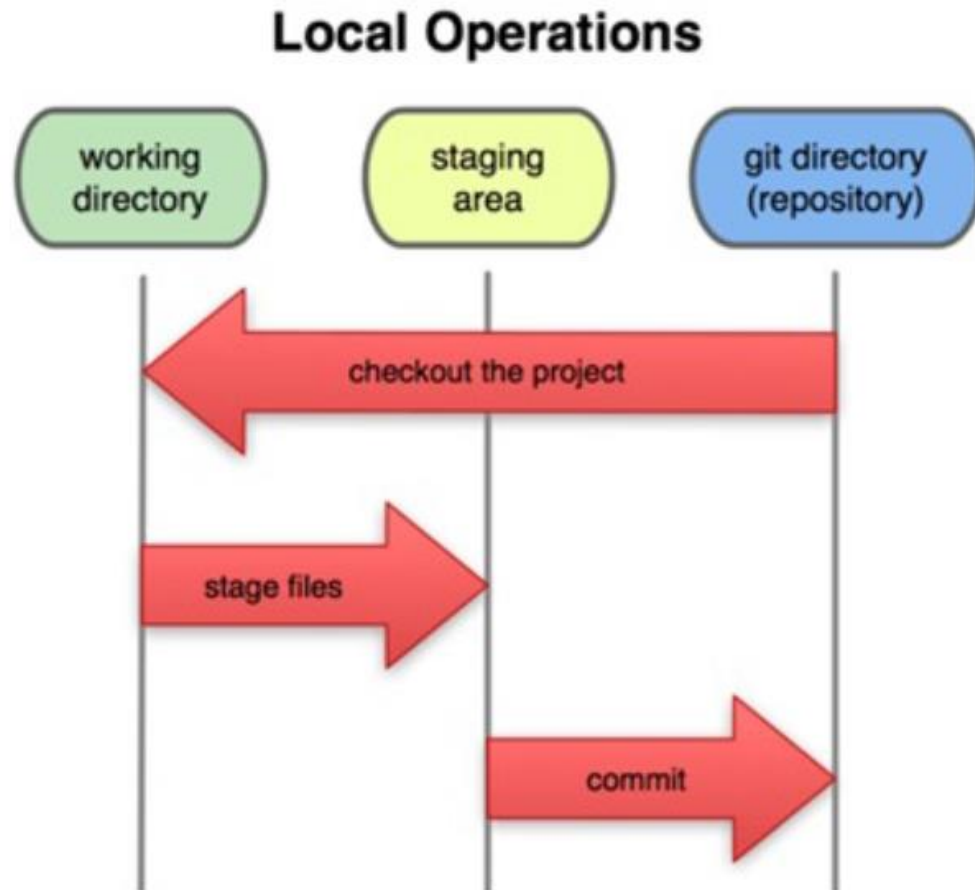
- zatwierdzony
- zmodyfikowany
- śledzony

- Zatwierdzony oznacza, że dane zostały bezpiecznie zachowane w lokalnej bazie danych.
- Zmodyfikowany oznacza, że plik został zmieniony, ale zmiany nie zostały wprowadzone do bazy danych.
- Śledzony - oznacza, że zmodyfikowany plik został przeznaczony do zatwierdzenia w bieżącej postaci w następnej operacji commit.

Główne sekcje projektu



- główne sekcje projektu Git:
 - katalog Git,
 - katalog roboczy
 - przechowalnia (ang. staging area).



Główne sekcje projektu cd.



- Katalog Git jest miejscem, w którym Git przechowuje własne metadane oraz obiektową bazę danych projektu.
 - To najważniejsza część Git i to właśnie ten katalog jest kopiowany podczas klonowania repozytorium z innego komputera.
- Katalog roboczy stanowi obraz jednej wersji projektu. Zawartość tego katalogu pobierana jest ze skompresowanej bazy danych zawartej w katalogu Git i umieszczana na dysku w miejscu, w którym można ją odczytać lub zmodyfikować.
- Przechowalnia to prosty plik, zwykle przechowywany w katalogu Git, który zawiera informacje o tym, czego dotyczyć będzie następna operacja commit.

Praca z Git



- Podstawowy sposób pracy z Git wygląda mniej więcej tak:
 - Dokonujesz modyfikacji plików w katalogu roboczym.
 - Oznaczasz zmodyfikowane pliki jako śledzone, dodając ich bieżący stan (migawkę) do przechowalni.
 - Dokonujesz zatwierdzenia (commit), podczas którego zawartość plików z przechowalni zapisywana jest jako migawka projektu w katalogu Git.

Git vs Github



Git sam w sobie jest aplikacją konsolową i nie posiada żadnego graficznego ani webowego interfejsu. Możesz pracować w Gicie lokalnie lub łącząc się z innym komputerem poprzez sieć.

Aby usprawnić pracę użytkownikom Gita powstało wiele serwisów hostujących repozytoria Gitowe. Do najpopularniejszych należą:

- github.com
- gitlab.com
- bitbucket.org

Dzięki tym serwisom można zarządzać swoimi repozytoriami z poziomu wygodnego, webowego interfejsu. Dodatkowo w bardzo ładny sposób prezentują one historię zmian, treść plików, hierarchię plików, członków danego repozytorium i tak dalej. Wprowadzają także warstwę organizacyjną i autoryzacyjną.

Github – strona główna



[Why GitHub?](#) [Enterprise](#) [Explore](#) [Marketplace](#) [Pricing](#)

Search GitHub



[Sign in](#)

[Sign up](#)

Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside 40 million developers.

Individuals

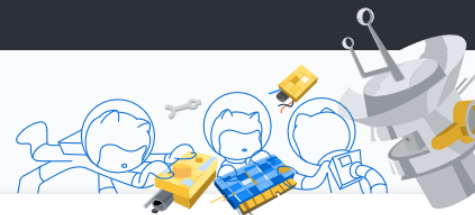
Everything a developer needs to contribute to the future of software.

[View individual plans](#) →

Organizations

Essential management and security for teams of all sizes.

[View organization plans](#) →



Get started with GitHub Enterprise

Take collaboration to the next level with security and administrative features built for teams.



Enterprise

Deploy to your environment or the cloud.

[Start a free trial](#)



Talk to us

Need help?

[Contact Sales](#) →

Gitlab – strona główna

[Product](#)[Pricing](#)[Resources](#)[Blog](#)[Support](#)[Jobs](#)[Get free trial](#)[Explore](#)[Sign in](#)[Register](#)

Empower your team to thrive remotely

At GitLab, our all-remote team works from 65+ countries to build the leading collaboration tool for DevOps. Discover everything we've learned in our [Remote Work Playbook](#).

[Try GitLab for FREE](#)[▶ Watch a demo](#)[DOWNLOAD THE REMOTE PLAYBOOK](#)

What is your biggest DevOps dilemma?

Complex Toolchains

Your DevOps toolchain is complex, expensive to maintain, and brittle

With GitLab, you can simplify your

Speed

Your developers are slowed down by bottlenecks, hand-offs, and re-work

With GitLab, SCM, CI, security and more

Security

You are forced to trade speed for security... or security for speed

With GitLab, you can move security "left"

Bitbucket – strona główna

[Why Bitbucket](#) ▾[Product Guide](#) ▾[Self-Hosted](#)[Pricing](#)[Log in](#)[Get started](#)

Tap into more advanced security permissions and admin settings with Bitbucket Cloud Premium. [Learn more.](#)

Built for professional teams

Bitbucket is more than just Git code management. Bitbucket gives teams one place to plan projects, collaborate on code, test, and deploy.

[Get started for free](#)

Or host it yourself with [Bitbucket Data Center](#) →

bugfix/123-bug remove extra padding

[Approve](#)[Merge](#)

bugfix/123-bug



master

[OPEN](#)

I made a few changes to tidy up the code. Please let me know if all looks good!



js / core.js

...

```
12 - // Takes an input date string and related date format
12 + // by parsing the date and converting to a string using
13 + function reconstructDateMatches(date, DateFormat) {
```



Free unlimited private



Best-in-class Jira & Trello



Built-in Continuous

Obsługa Gita

Sposoby zarządzania repozytorium



1. Z poziomu konsoli
2. Przy użyciu klientów Git
 - TortoiseGit
 - Git GUI
 - GitHub Desktop
3. Stosując IDE z wbudowaną obsługą Gita
 - Eclipse
 - Clion
 - Visual Studio
 - Visual Code
4. Bezpośrednio poprzez stronę internetową

Sposoby zarządzania repozytorium



1. Z poziomu konsoli

2. Przy użyciu klientów Git

- TortoiseGit
- **Git GUI**
- **GitHub Desktop**

Te sposoby omówimy.

3. Stosując IDE z wbudowaną obsługą Gita

- Eclipse
- Clion
- Visual Studio
- Visual Code

4. Bezpośrednio poprzez stronę internetową

Instalacja narzędzia Git



1. System Windows:

- Git (razem z Git GUI) <https://git-scm.com/download/win>
- GitHub Desktop <https://desktop.github.com/>

2. System Linux Ubuntu:

- Git (bez GUI) `sudo apt install git`

3. System MacOS:

- Git (razem z Git GUI) <https://git-scm.com/download/mac>
- GitHub Desktop <https://desktop.github.com/>

Pobranie adresu repozytorium



rberdyga / repo1

Unwatch 1

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

some description

Edit

Manage topics

2 commits

1 branch

0 packages

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

rberdyga Commit1

17.1.cpp

Commit1

17.2.cpp

Commit1

17.3.cpp

Commit1

README.md

Initial commit

Clone with HTTPS

Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/rberdyga/repo1.git



Open in Desktop

Download ZIP

7 minutes ago

README.md



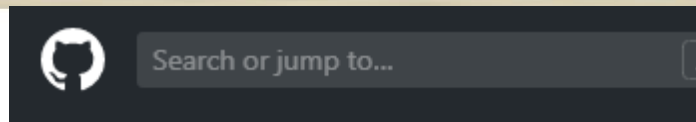
repo1


some description

Zarządzanie repozytorium przez stronę internetową



Obsługa przez stronę internetową 1




 rberdyga ▾

Repositories


 New

Find a repository...

 [mmejer/EvolvingCars](#)

 [mmejer/osm_projekt1](#)

 [Kurs-Cpp-2019-2... /Zajecia17](#)

 [rberdyga/payout_uber_bolt_drivers](#)

 [Kurs-Cpp-2019-2... /Zajecia18](#)

 [Kurs-Cpp-2019-2... /Zajecia16](#)

 [Kurs-Cpp-2019-2... /Zajecia12](#)

Show more

Your teams

You don't belong to any teams yet!

Po zalogowaniu się wybieramy po lewej stronie ekranu przycisk **New** w zakładce *Repositories* aby utworzyć nowe repozytorium.

Obsługa przez stronę internetową 2





Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)



Owner Repository name *

 rberdyga ▾ / repo1 

Great repository names are short and memorable. Need inspiration? How about [curly-succotash](#)?

Description (optional)

some description

- ☒  **Public**
Anyone can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

- ☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾ | Add a license: **None** ▾ 

Create repository

Uzupełniamy
informacje
na temat
repozytorium
i klikamy **Create
repository.**

Obsługa przez stronę internetową 3



rberdyga / repo1

Unwatch ▾

1

★ Star

0

🍴 Fork

0

↔ Code

! Issues 0

🔗 Pull requests 0

▶ Actions

📁 Projects 0

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

some description

Edit

[Manage topics](#)

🔗 1 commit

🌿 1 branch

📦 0 packages

📄 0 releases

👤 1 contributor

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



rberdyga Initial commit

Latest commit 04ff544 now



README.md

Initial commit

now

📖 README.md



repo1

some description

Aby dodać nowe pliki klikamy **Upload files**.

Obsługa przez stronę internetową 4



rberdyga / repo1

Unwatch 1

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

repo1 /



Drag files here to add them to your repository

Or [choose your files](#)



Commit changes

Add files via upload

Add an optional extended description...

☒ Commit directly to the `master` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

Cancel

Wrzucamy pliki,
wpisujemy treść
commita,
możemy wybrać
brancha (gałąź)
projektu.

Na koniec
klikamy **Commit
changes.**

Aplikacja Git GUI



Obsługa za pomocą klienta Git GUI 1



 Git Gui



Repository Help



[Create New Repository](#)

[Clone Existing Repository](#)

[Open Existing Repository](#)

Open Recent Repository:

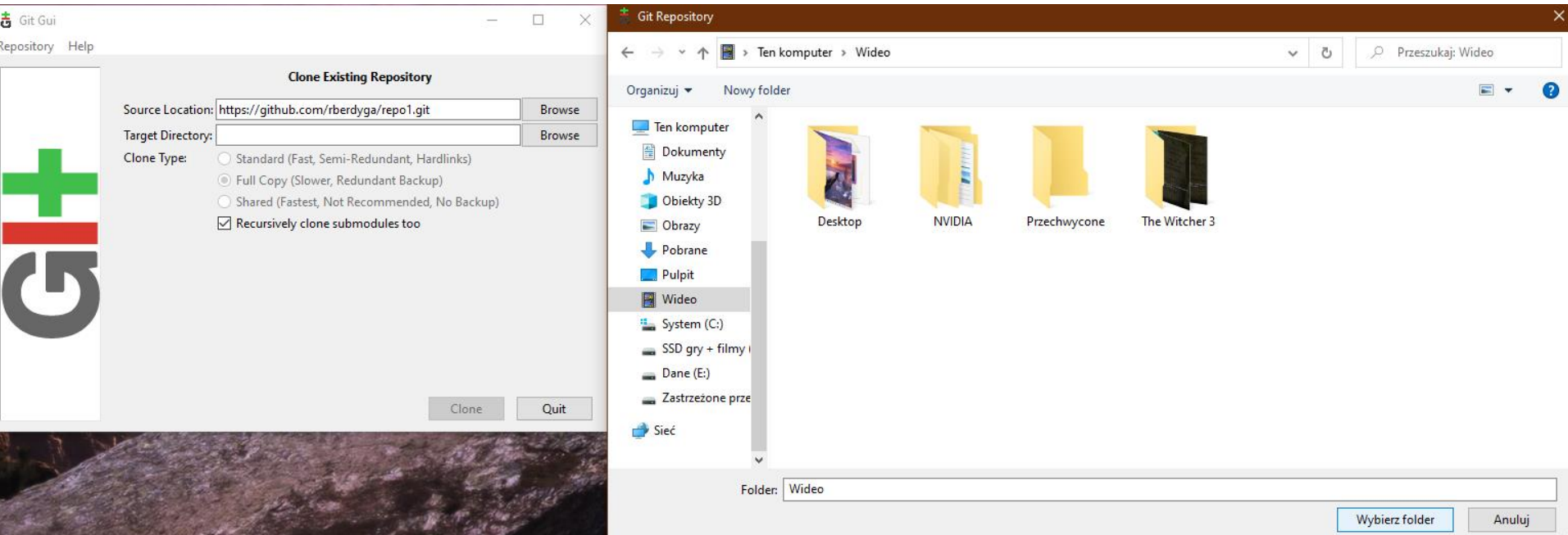
[E:/Pliki/GITLAB/osm](#)

Wybieramy pożądaną opcję. Ja wybrałem w tym momencie opcję 'clone'.

Clone – funkcja, która pobiera repozytorium z serwera na komputer lokalny

Quit

Obsługa za pomocą klienta Git GUI 2

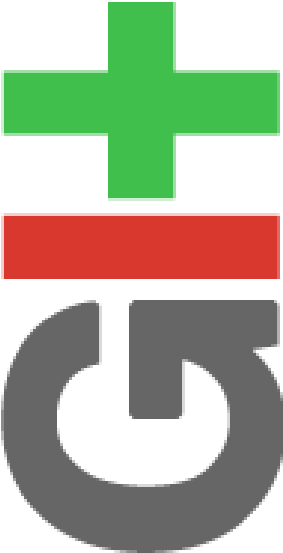


Jako **Source Location** wpisujemy adres repozytorium z serwera (np. z GitHuba)

Jako **Target Directory** wybieramy dowolny katalog na dysku.

Obsługa za pomocą klienta Git GUI 3





Git Gui

Repository Help

Clone Existing Repository

Source Location:

Target Directory:

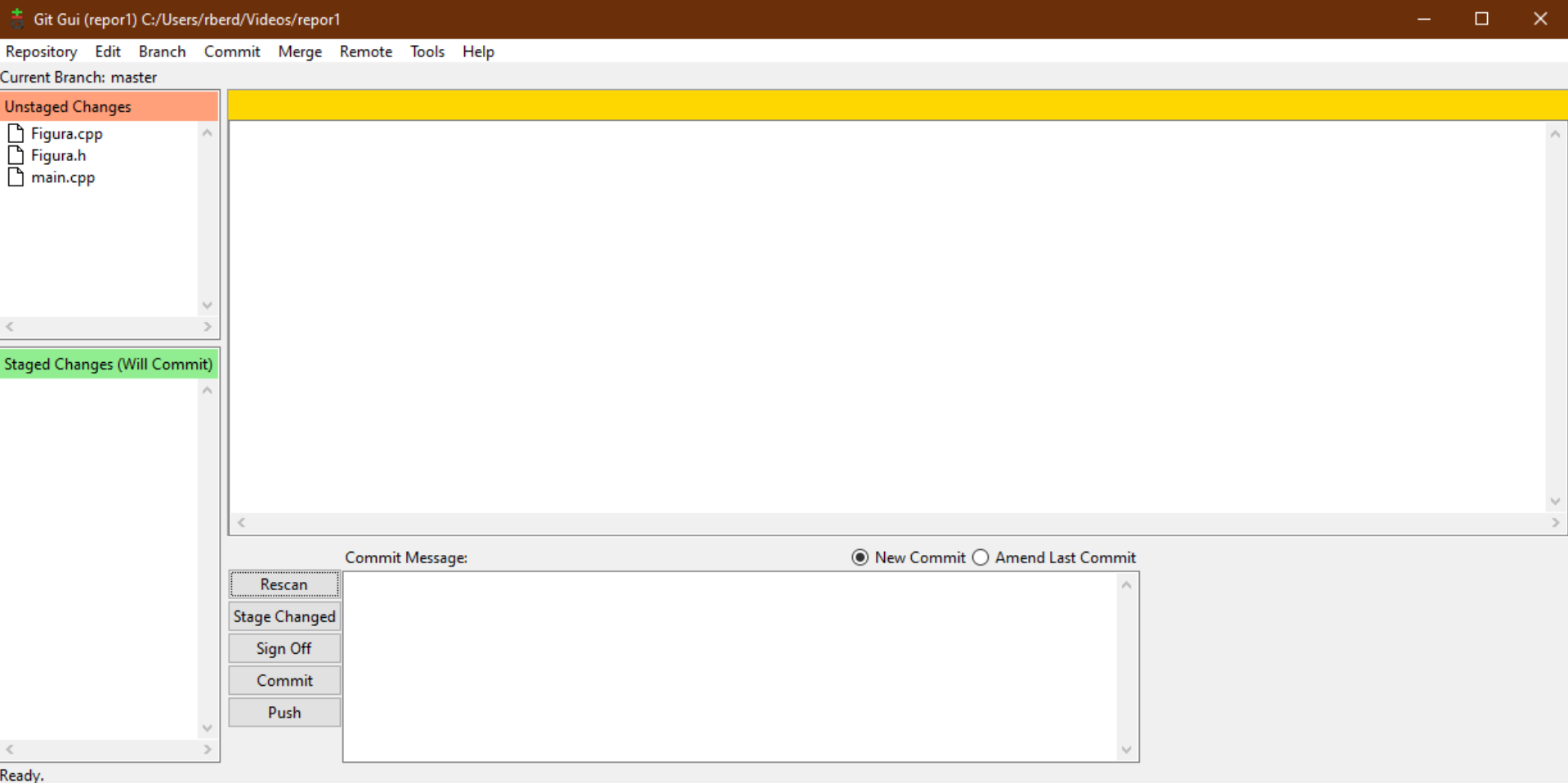
Clone Type:

- ☐ Standard (Fast, Semi-Redundant, Hardlinks)
- ☒ Full Copy (Slower, Redundant Backup)
- ☐ Shared (Fastest, Not Recommended, No Backup)
- ☒ Recursively clone submodules too

Po wybraniu folderu musimy w polu **Target Directory** wpisać nazwę katalogu, która zostanie utworzona na potrzeby repozytorium, poprzedzając ją backslashem, np.

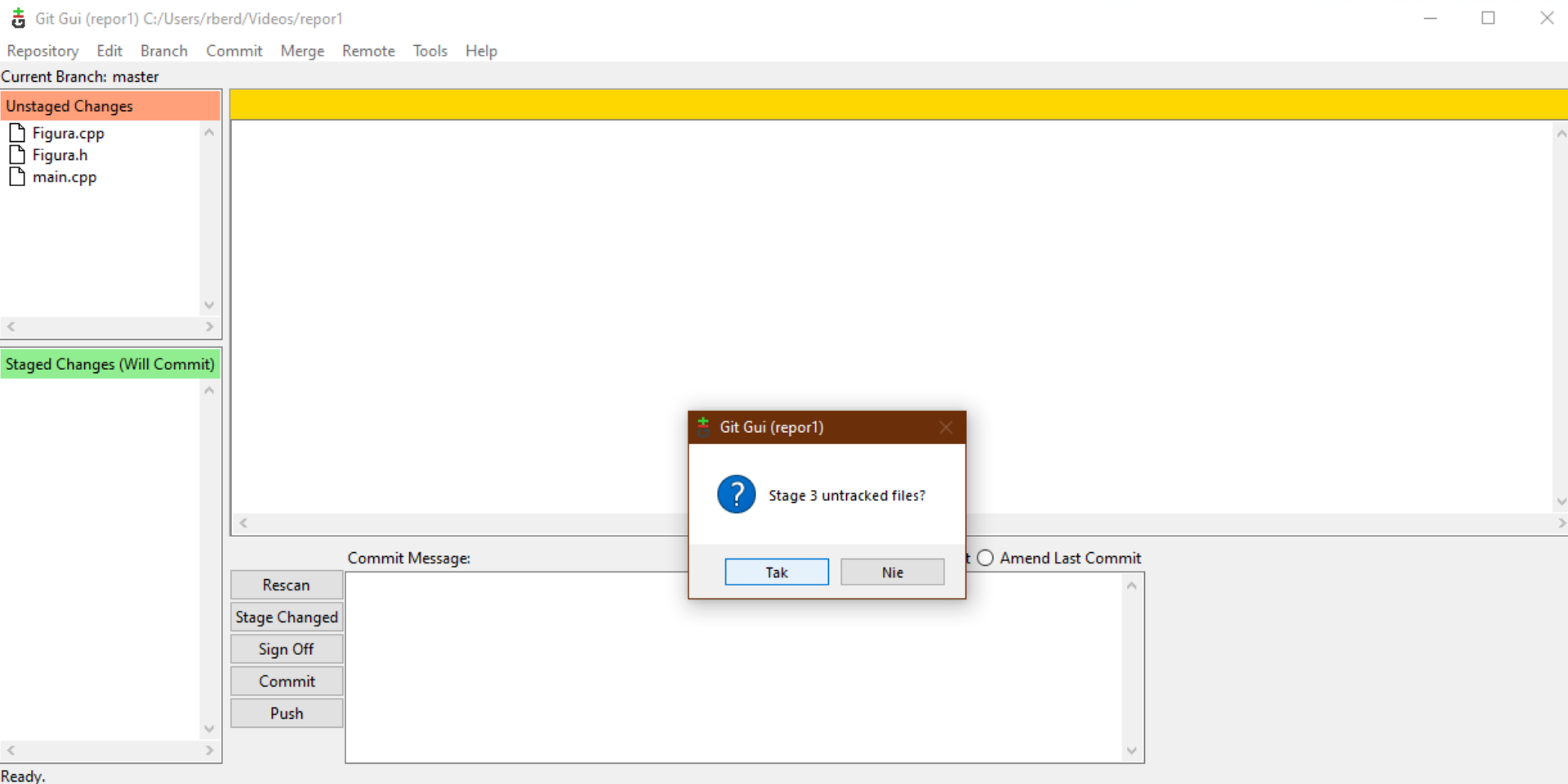
/repor1

Obsługa za pomocą klienta Git GUI 4



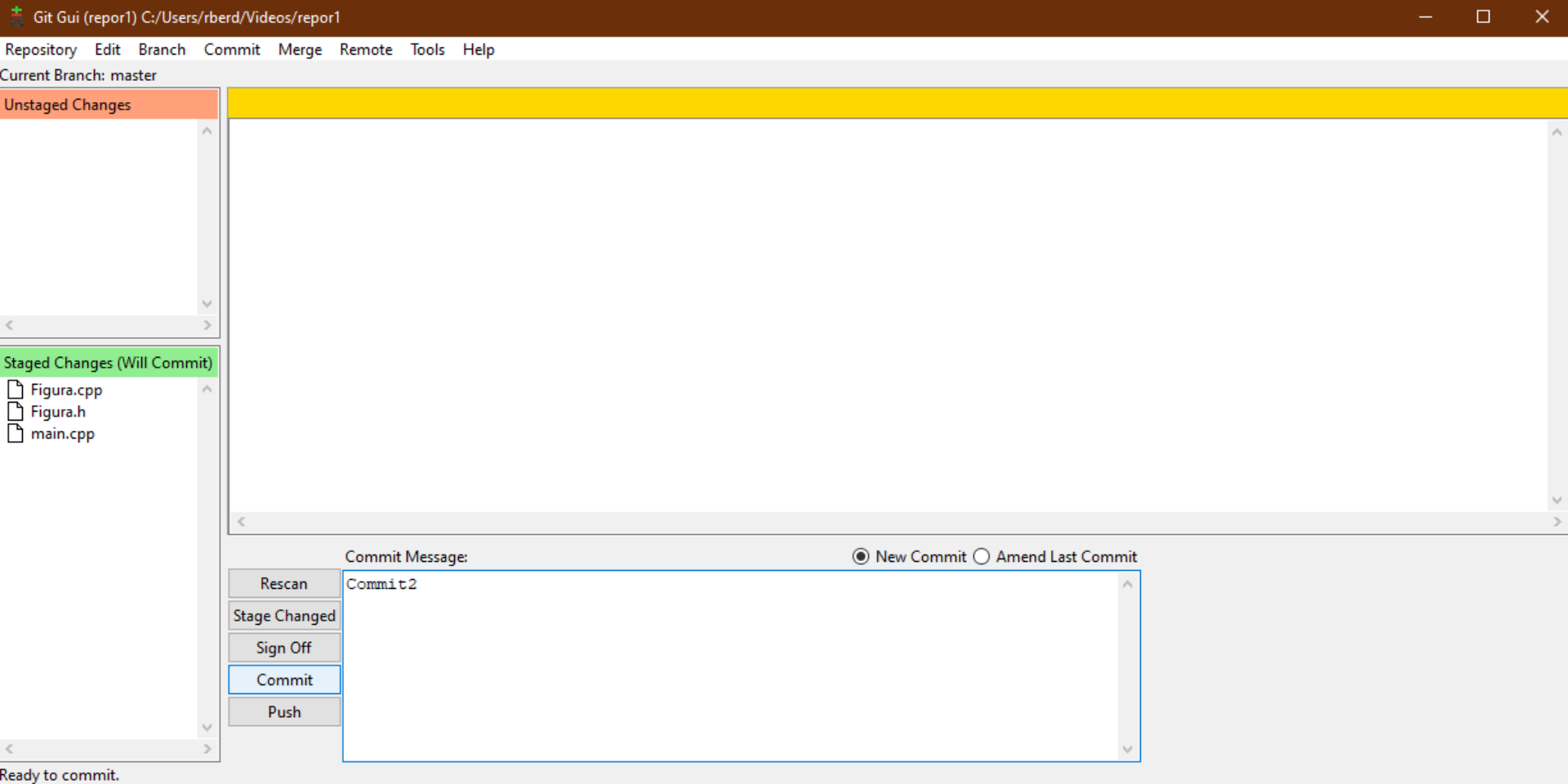
Powinno ukazać się nam takie okno. Gdy dodamy pliki **LOKALNIE** do folderu z repozytorium i klikniemy opcję **Rescan**, po lewej stronie system wykryje nowe pliki lub katalogi oraz ewentualne zmiany w plikach, które już w repozytorium były.

Obsługa za pomocą klienta Git GUI 5



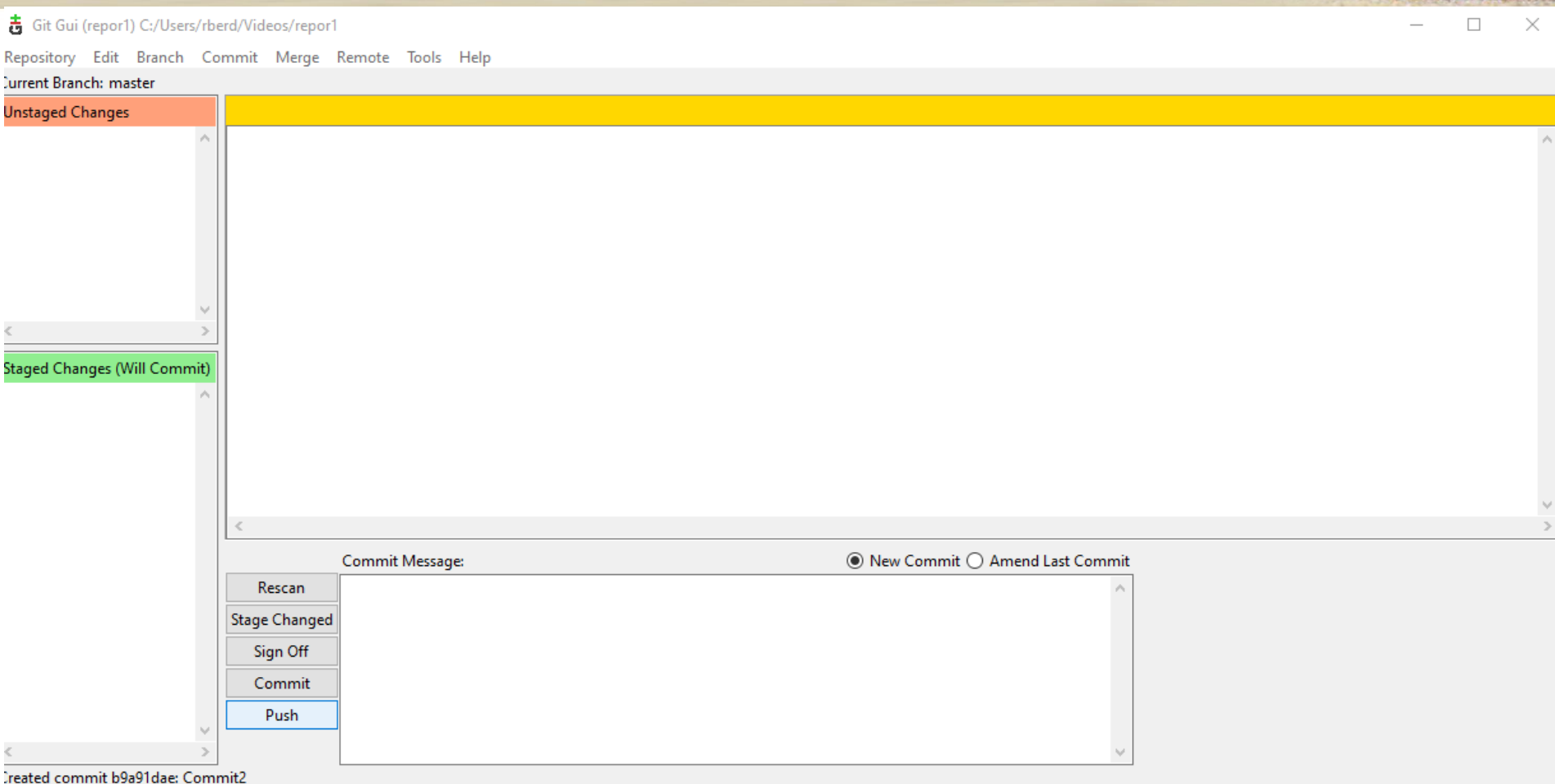
Gdy będziemy gotowi, możemy dodać nasze pliki lub foldery do poczekalni. DO tego celu służy przycisk **Stage Changed**.

Obsługa za pomocą klienta Git GUI 6



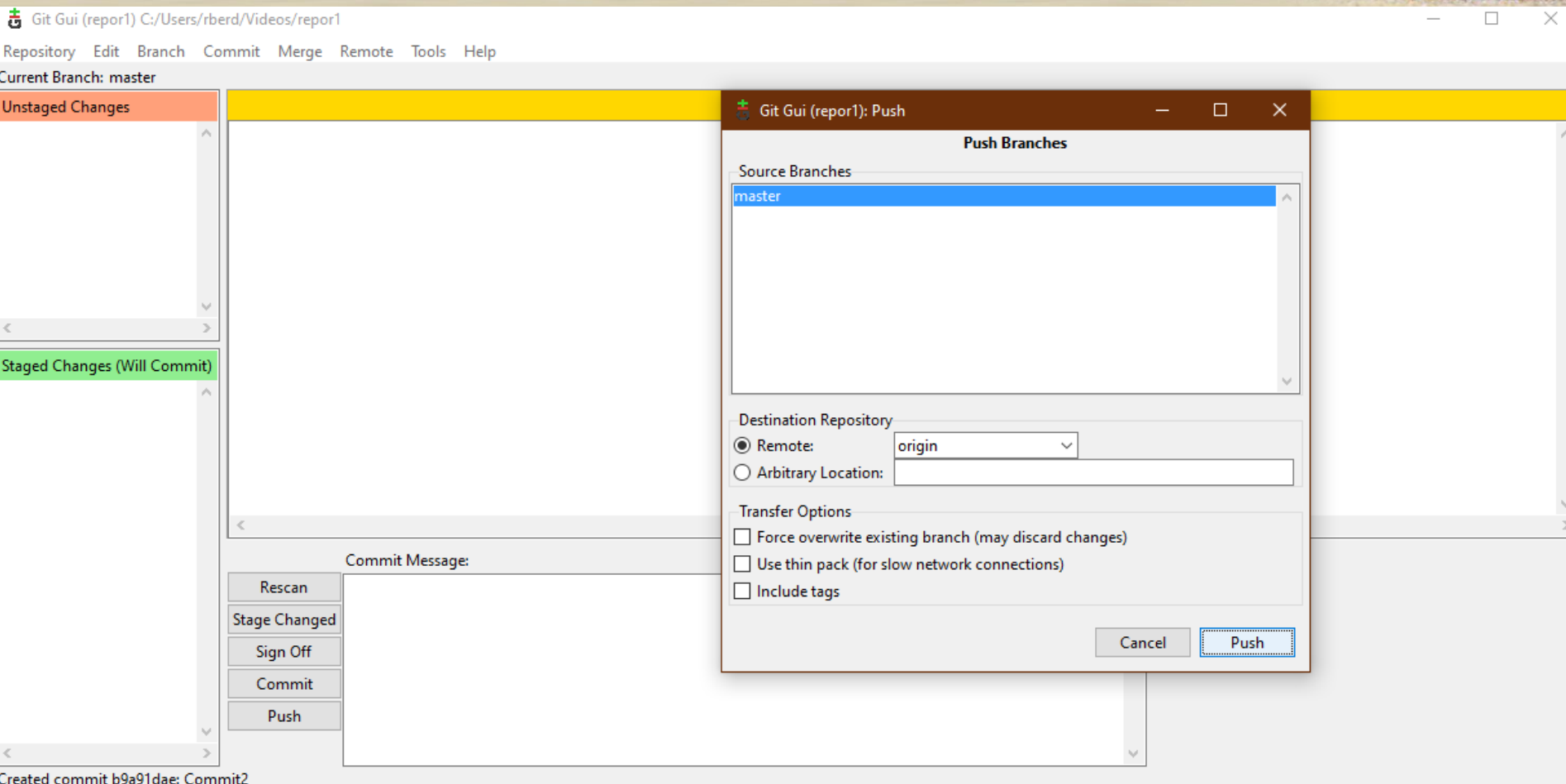
Jeśli mamy już nasze pliki w poczekalni (po lewej na zielono) możemy wykonać commit. Wpisujemy jego treść oraz klikamy przycisk **Commit**.

Obsługa za pomocą klienta Git GUI 7



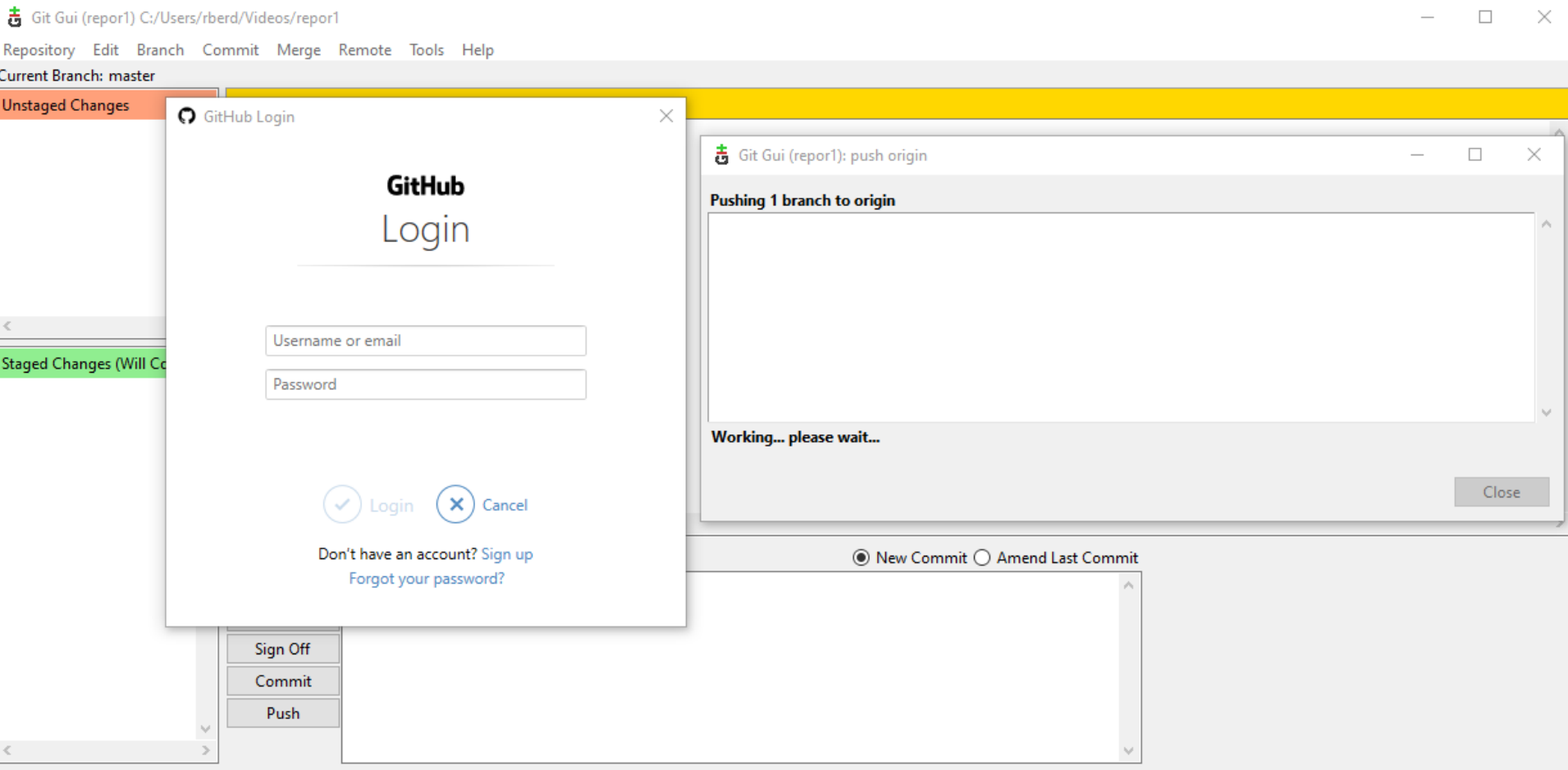
Po wykonaniu commita musimy wrzucić wszystko na serwer. Do tego służy komenda **Push**.

Obsługa za pomocą klienta Git GUI 8



System pozwala na wybór **brancha** (gałęzi) projektu. Domyślna gałąź to *master*. Wybieramy gałąź, wybieramy **Remote** *origin* i klikamy **Push**.

Obsługa za pomocą klienta Git GUI 9

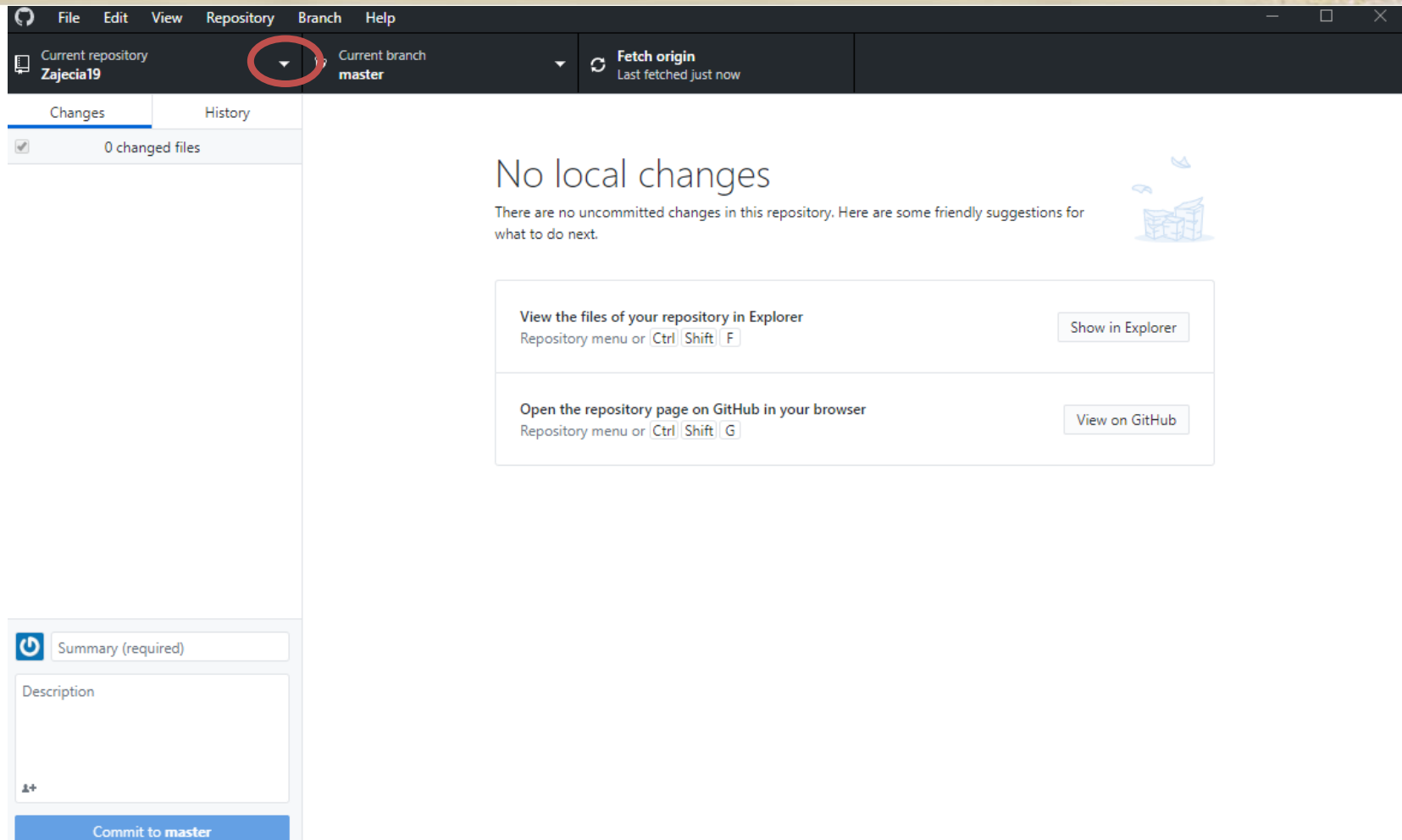


W tym momencie (jeśli wcześniej tego nie zrobiliśmy) serwer poprosi nas o logowanie, aby sprawdzić czy możemy wprowadzać zmiany repozytorium.

Aplikacja GitHub Desktop

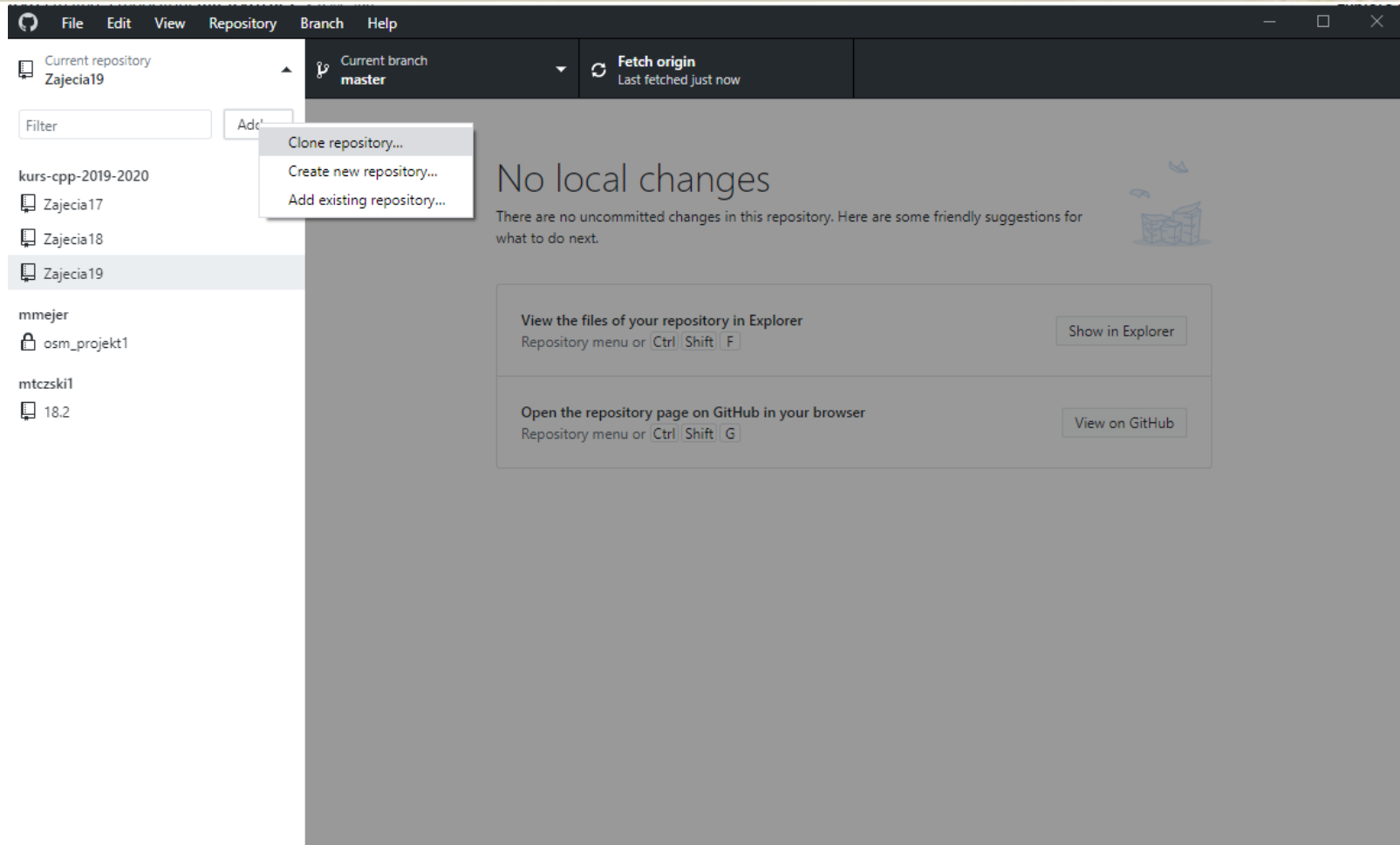


Obsługa za pomocą klienta GitHub Desktop 1



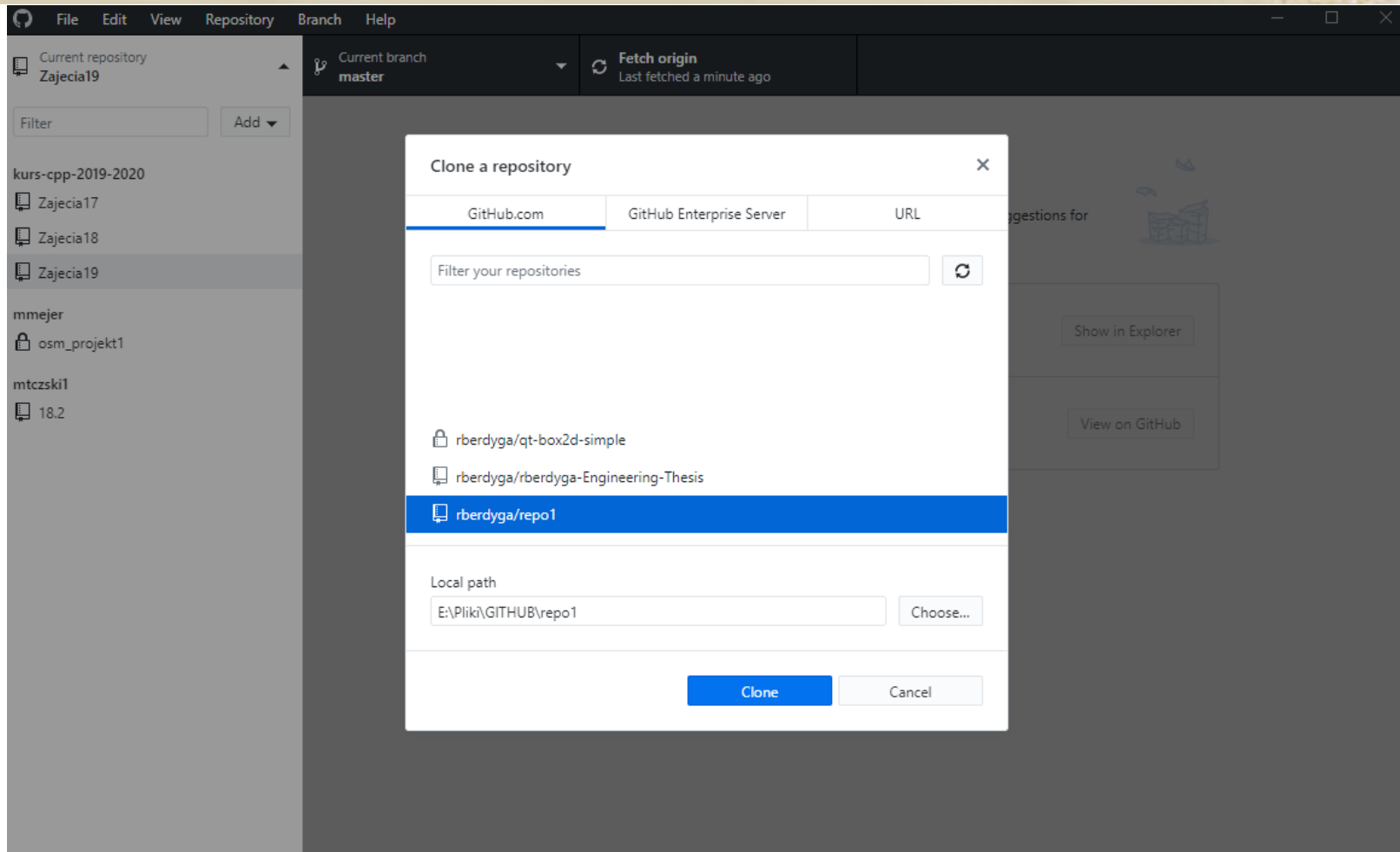
Po zalogowaniu powinniśmy zobaczyć mniej więcej to na ekranie. Klikamy po lewej stronie na niewielką strzałkę skierowaną w dół.

Obsługa za pomocą klienta GitHub Desktop 2



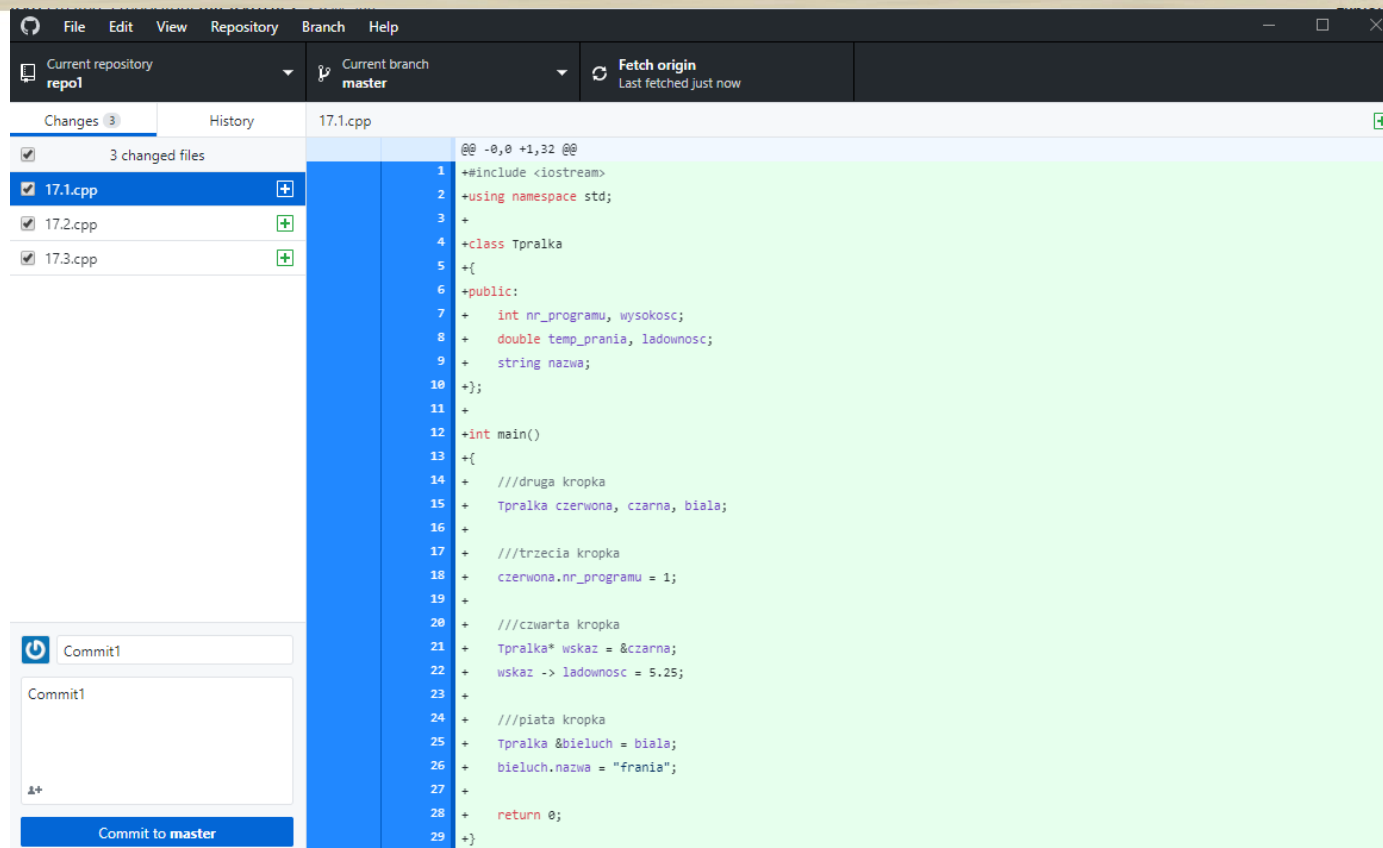
Wybieramy pożądaną opcję. Ja wybrałem w tym momencie opcję 'clone'.

Obsługa za pomocą klienta GitHub Desktop 3



Wybieramy repozytorium lub podajemy jego adres URL w zakładce URL. Wybieramy także ścieżkę lokalną, gdzie repozytorium zostanie sklonowane.

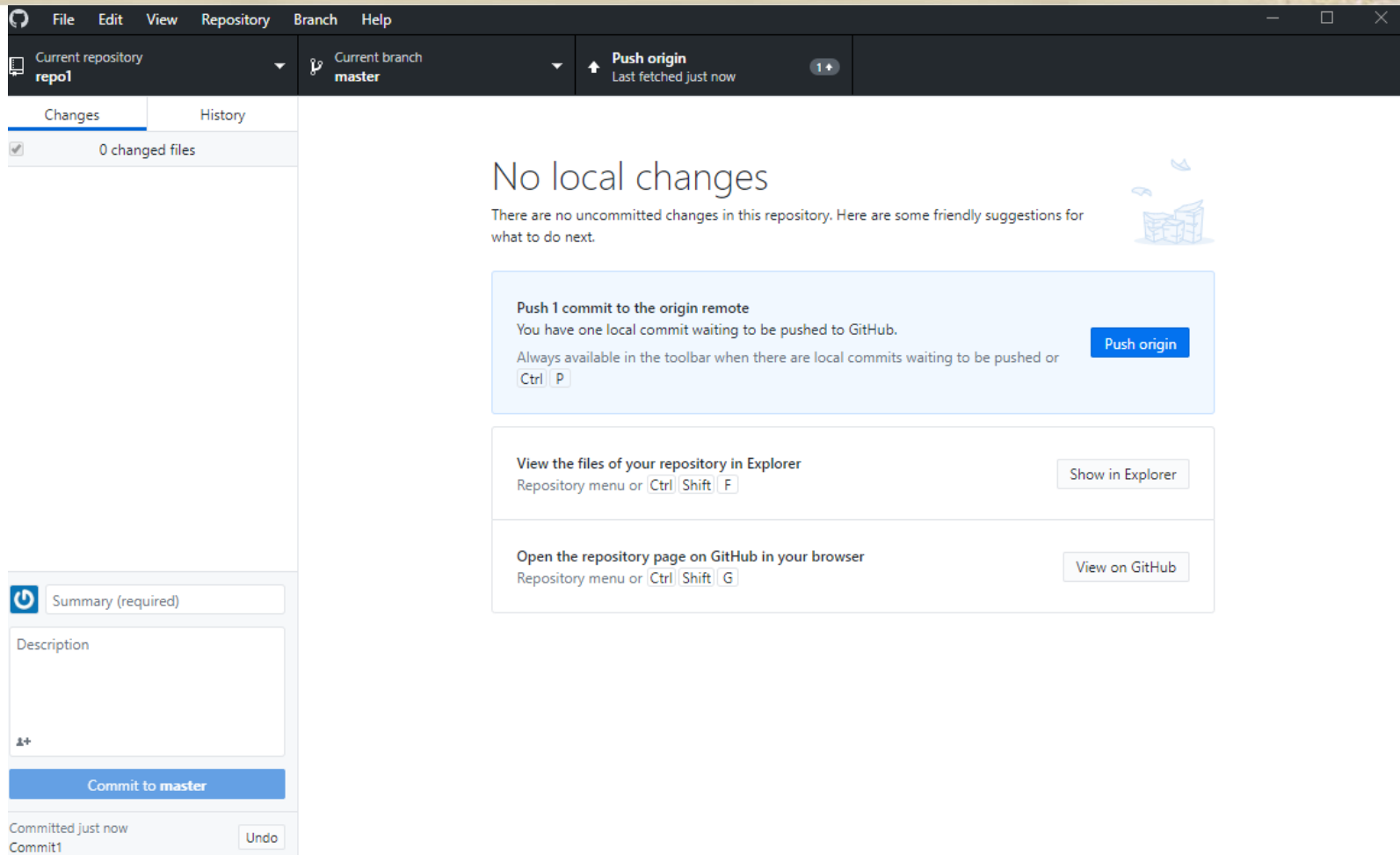
Obsługa za pomocą klienta GitHub Desktop 4



Gdy dodamy pliki **LOKALNIE** do folderu z repozytorium system sam wykryje nowe pliki lub katalogi oraz ewentualne zmiany w plikach, które już w repozytorium były.

Następnie możemy zaznaczyć pliki, które chcemy dodać (domyślnie wszystkie będą zaznaczone), wpisać treść commita, wybrać w panelu górnym branch, a na końcu **Commit to *master*** (gdzie *master* to nazwa wybranej gałęzi projektu).

Obsługa za pomocą klienta GitHub Desktop 5



Po wykonaniu commita musimy wrzucić wszystko na serwer. Do tego służy komenda **Push** w środkowej części ekranu.

Poziom konsoli – najważniejsze komendy



Obsługa z poziomu konsoli 1



Sprawdzenie zainstalowanej wersji git:

- `git version`

Uwaga! Podkreślone elementy oznaczają, że w Waszym repozytorium mogą się one inaczej nazywać.

Konfiguracja tożsamości:

- `git config -global user.name "Rafał Berdyga"`
- `git config -global user.email "rberdyga@gmail.com"`

Utworzenie pustego repozytorium:

- `git init`

Sprawdzenie statusu lokalnego repozytorium:

- `git status`

Warto upewnić się, że będąc w konsoli lub terminalu, znajdujemy się w odpowiednim katalogu z projektem!

Obsługa z poziomu konsoli 2



Pobranie aktualnej wersji repozytorium z serwera:

- `git clone git://adres-serwera/nazwa-repozytorium.git`

Dodawanie pliku:

- `git add file` ← Dodanie pojedynczego pliku
- `git add *` ← Dodanie wszystkich plików i katalogów (poza usuniętymi)
- `git add -A` ← Dodanie wszystkich plików i katalogów

Usunięcie pliku z indeksu:

- `git rm --cached file`

Wykonanie commita ze zmianami w pliku:

- `git commit -m "nazwa commita"`

Obsługa z poziomu konsoli 3



Tworzenie nowego rozgałęzienia (brancha):

- `git branch name`

Przełączenie się do innej gałęzi:

- `git checkout name`

Usunięcie rozgałęzienia (trzeba być od niego odłączonym):

- `git branch -D name`

Wysłanie lokalnego repozytorium na serwer:

- `git remote add origin git://adres-serwera/nazwa-repozytorium.git`
- `git push origin master`

Pracując w kilka osób można pobrać aktualną wersję repozytorium za pomocą:

- `git pull origin master`