Pludry, 14.06.2013r

Programowanie Komputerów 4
Sprawozdanie z projektu
„GitUI”

Autor: Przemysław Thomann
AEI INF, sem. 4 gr.5

Prowadzący: dr inż. Roman Starosolski

1. Analiza problemu.

Tematem projektu było stworzenie prostego i miłego dla oka okienkowego interfejsu użytkownika (GUI) dla rozproszonego systemu kontroli wersji oprogramowania jakim jest GIT i któremu domyślnie polecenia przekazywane za pośrednictwem linuksowej powłoki. Celem natomiast było usprawnienie lub przynajmniej uproszczenie użytkowania programu GIT, który niewątpliwie jest potężnym narzędziem choć na pierwszy rzut oka może wydawać się skomplikowany. Program *GitUI* nie zawiera „logiki" działania – jest jedynie pośrednikiem między użytkownikiem a programem GIT – pozwala zarówno wykonywać polecenia jak i odczytywać odpowiedź od programu GIT. Całym problemem projektu było zapewnienie prawidłowej interakcji i współpracy między użytkownikiem wykonującym określone działania na nakładce a odpowiednim wywoływaniem poleceń konsolowych GIT'a.

Projekt napisany został w języku C++ z wykorzystaniem biblioteki QT i wykorzystuje technikę programowania obiektowego. Biblioteka QT pozwala na tworzenie graficznego interfejsu użytkownika oraz definiowania odpowiednich połączeń i interakcji między zdarzeniami. Pozwala także na wygodne tworzenie elementów takich jak przyciski, paski zadań itp. oraz ich odpowiednie ułożenie w okienku.

Założeniem projektu *GitUI* było utożsamienie najważniejszych funkcji GIT'a, które miały być udostępniane przez nakładkę z odpowiednią klasą, zapewniającej wszystko co potrzebne do obsługi danej funkcji GIT'a. W związku z tym założeniem projekt posiada 13 klas będących głównie odpowiedzialnymi za definicje metod i pól związanych z obsługą danej instrukcji GIT'a oraz tworzenia odpowiedniego interfejsu okienka, obsługiwanego przez użytkownika.
Udostępnione funkcje to :
- git init – inicjacja repozytorium git'a w danym miejscu
- git add – dodanie pliku do „śledzonych" przez repozytorium
- git branch – tworzenie gałęzi, wyświetlanie gałęzi, usuwanie gałęzi
- git checkout – umożliwia przeskok na inną gałąź,
- git commit – commit, czyli stworzenie migawki aktualnego stanu śledzonych przez nas GIT'a plików
- git log – sprawdzenie aktualnego stanu śledzonych i nie śledzonych plików
- git remote – operacje związane z repozytoriami zdalnymi, dodawanie, usuwanie, modyfikacja
- git clone – klonuje repozytorium w nowe
- git push - „wciska" nasze repozytorium, uaktualnia ze zdalnym (modyfikuje zdalne)
- git fetch – pobiera zmiany ze zdalnego repozytorium, uaktualnia nasze
- git merge – scala zmiany jednej gałęzi z inną
- git config – konfiguracja programu GIT, danych użytkownika obsługującego repozytorium itp.

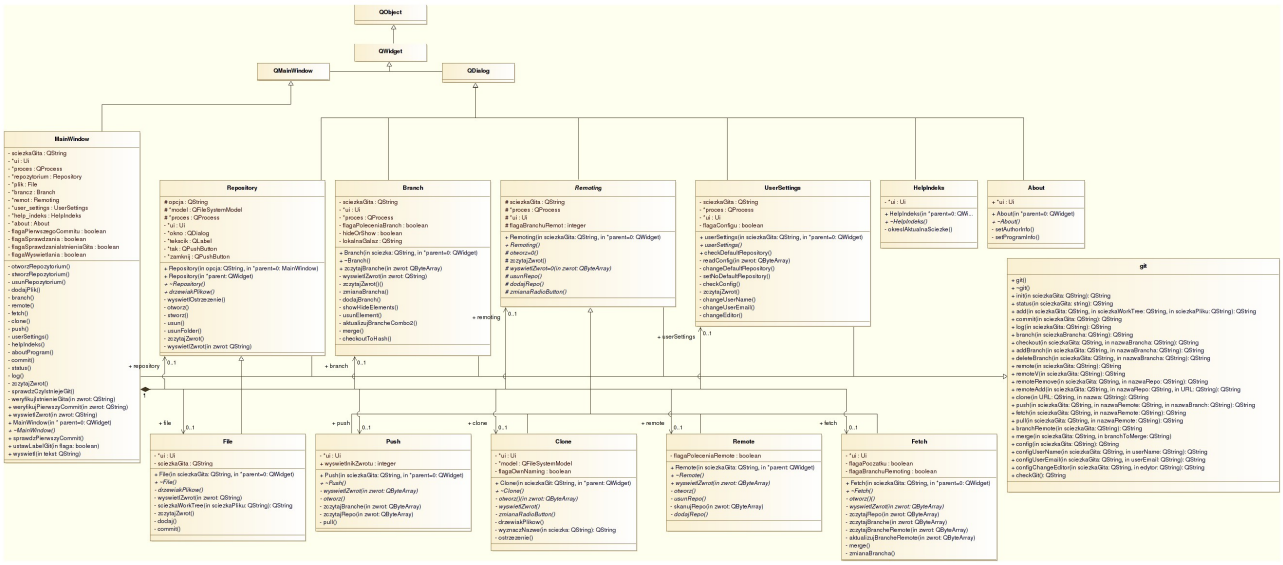
Powyższe polecenia są wykorzystywane w programie zazwyczaj w najprostszej formie, choć często są to dość rozbudowane instrukcje – bardziej zaawansowane operacje można wykonać za pośrednictwem konsoli.
Szczegółowe dane dotyczące poleceń programu GIT można znaleźć w dokumentacji dostępnej pod adresem: http://git-scm.com/documentation

Schemat interfejsu sprowadza się do 1 okna głównego, które udostępnia tworzenie innych okienek udostępniających określone funkcje. Poniżej okno główne programu:

Główne okno zawiera „wyświetlacz” , który zwraca odpowiedzi od programu GIT (dokładnie takie same, jakie otrzymałby użytkownik wpisując dane polecenie w konsoli linuksowej). Odpowiednie przyciski pozwalają na wybranie odpowiedniego dla użytkownika polecenia. Dokładny opis funkcji udostępnionych za pośrednictwem nakładki można znaleźć w części *3. Specyfikacja zewnętrzna.*

2. Specyfikacja wewnętrzna.

Poniżej znajduje się diagram UML projektu:



Projekt zawiera 13 klas oraz plik main.cpp w którym znajduje się główna funkcja main() odpowiedzialna za stworzenie głównego okna programu (MainWindow). Każda klasa poza *git* oraz *remote* posiada formularz .ui definiujący wygląd okna. .

Działanie programu sprowadza się do tworzenia obiektu klasy **QProcess**, który pozwala na uruchomienie i komunikację z procesem. Następnie nakładka reaguje na reakcje użytkownika w postaci np. wciśnięcia klawisza i wywołuje odpowiednią funkcję GIT'a łącząc się z procesem, przekazując mu w parametrach dokładną postać komendy, łącznie z uwzględnieniem ścieżki repozytorium do którego się odwołujemy.

Każda klasa wykonująca polecenia GIT'a posiada obiekt klasy **QProcess** – *proces.* Chcąc wykonać jakąś komendę GIT'a wywołujemy metodę proces->start() ,przekazując jako argument metody obiekt klasy **QString**, a najczęściej będzie to tymczasowy obiekt tworzony przy wykorzystaniu obiektu zwracanego przez wywołanie statycznej metody klasy git, definiującej określoną postać wywołania komendy GIT'a. Każda klasa posiada

connect(proces,SIGNAL(readyRead()), this, SLOT(zczytajZwrot()));

dzięki czemu w momencie pojawia się odpowiedzi procesu (takiej samej jaką otrzymalibyśmy w konsoli) zostanie wywołany slot zczytajZwrot() określony następująco:

```
void Klasa:: zczytajZwrot(){
    QByteArray bytes = proces->readAllStandardOutput();
    wyswietlZwrot(bytes);
}
```

gdzie Klasa jest nazwą klasy w której metoda jest zdefiniowana a wyswietlZwrot(bytes) jest funkcją odpowiedzialną za wyświetlenie tekstu w obiektach klasy do tego przeznaczonych.

Ten schemat działania powtarza się we wszystkich klasach wykorzystujących obiekty klasy **QProcess**.

Każda klasa wywołująca komendy programu GIT posiada pole QString sciezkaGita będące odpowiedzialne za pamiętanie ścieżki do repozytorium GIT'a, które użytkownik musi otworzyć przed wykonaniem operacji używając programu GitUI (można skorzystać także z automatycznego otwierania repozytorium – dokładne informacje można znaleźć w pliku *HelpIndeksGitU.html*). Każde nowo tworzone okno, poza oknem głównym jest powiązane z oknem głównym, z obiektem klasy MainWindow poprzez przekazanie konstruktorowi nowo tworzonego okna wskaźnika na obiekt okna głównego. W związku z tym, jeżeli dane metody nie są prywatne (a publiczne) możemy wywoływać metody okna głównego z okna stworzonego poprzez wykorzystanie metody *parentWidget()* dostępnej dzięki klasie **QWidget,** która dziedziczona jest poprzez dziedziczenie przez każdą klasę tworzącą okno pochodnej klasy **QWidget** (wykorzystane w projekcie klasy **QMainWindow** oraz **QDialog**). Dokładny przebieg dziedziczenia przedstawia diagram UML. W konstruktorach klas tworzących okienka, wywoływane są konstruktory klas **QDialog l**ub **QMainWindow** – w przypadku dziedziczenia stworzonych klas, konstruktory te są wywoływane tylko w klasach nadrzędnych.

Działanie i reagowanie programu na określone sygnały wykorzystuje technologię **sygnałów i slotów** udostępnioną przez framework QT. Zasada działania polega na wywoływaniu odpowiednich slotów (metod) jako reakcji na odpowiednie działanie np. kliknięcie klawisza. Sloty te nie przyjmują parametrów oraz w większości przypadków nie zwracają wyniku, dlatego są void.

Omówienie poszczególnych klas:

```
class MainWindow : public QmainWindow, public git
```

Klasa odpowiedzialna za definiowanie metod oraz pól okna głównego, pozwalającego na dostęp do wszystkich pozostałych okien. Obiekt tej klasy tworzony jest w funkcji main() od razu po uruchomieniu programu. Obiekt okna zawiera w sobie obiekty klas **QGridLayout** (odpowiedzialny m.in. za rozmieszczenie obiektów w oknie w przypadku zmiany rozmiaru okna), **QLabel** (napisy w oknie), **QTextBrowser** (wyświetlacz, w którym wyświetlane są odpowiednie informacje) oraz **QMenuBar** (będący paskiem zadań, wyboru poszczególnych opcji programu, tworzenia innych okien).

```
class Repository : public QDialog, public git
```

Klasa definiuje metody i pola związane z obsługą repozytoriów GIT'a, czyli tworzeniem, usuwaniem i otwieraniem folderów „..git" w którym zawarte są niezbędne do działania GIT'a pliki. Obiekt klasy tworzony w przez obiekt klasy MainWindow, w reakcji na wybranie odpowiedniej opcji w pasku zadań (Repository → odpowiednia opcja) . W zależności od wybrania opcji, tworzone okno pozwala na stworzenie repozytorium, otwarcie istniejącego lub usunięcie istniejącego. Opcja ta jest przekazywana jako parametr wywołania konstruktora klasy,
opcje: „otworz" - modyfikuje działanie odpowiednich przycisków oraz elementów okna tak aby pozwalały na otwarcie danego repozytorium
„stworz" - jak wyżej, z tym że do zapisu, stworzenia nowego repozytorium
„usun" - pozwala usunąć repozytorium , korzystając z linuksowego polecenia *rm*
Obiekty klas QT wykorzystane w klasie: **QGridLayout**, **QLabel**, **QPushButton**, **QTreeView** (tworzy drzewo plików, pozwalając na operacje na plikach).

```
class File : public Repository
```

Klasa definiuje metody i pola związane z dodawaniem plików do repozytorium GIT'a oraz robieniem „commitów". Wykorzystuje obiekt klasy QtreeView pozwalający zaznaczanie plików i

dzięki temu dodawanie ich do śledzonych przez repozytorium GIT'a. Wykorzystane obiekty klas QT: **QGridLayout**, **QLabel, QPushButton**, **QTextBrowser**, **QTreeView** .

`class Branch : public QDialog, public git`

Klasa odpowiedzialna za definiowanie metod związanych z operacjami na GIT'owych gałęziach. Pozwala na tworzenie, usuwanie i przeskakiwanie między gałęziami. Obiekty klas QT wykorzystane w klasie: **QFormLayout**, **QComboBox** (pozwala na stworzenie obiektu, z którego możemy jak z listy wybrać odpowiedni element), **QLabel, QLineEdit** (obiekt pozwala na wpisanie, edycję linii tekstu) , **QPushButton**, **QTextBrowser**, **QRadioButton** (tworzy guzik „radio").

`class Remoting : public QDialog, public git`

Klasa abstrakcyjna odpowiedzialna za definiowanie podstawowych metod i pól, obsługiwanych przez klasy pochodne (**Push, Clone, Remote, Fetch**), związana z operacjami sieciowymi GIT'a. Najważniejsza metoda to virtual void *otworz*() =0; będąca czysto wirtualną metodą i przedefiniowana w każdej klasie pochodnej, na najważniejszą funkcję charakteryzującą daną klasę. Klasa zawiera formularz .ui, który wykorzystywany jest przez klasę pochodną Remote, definiującą wygląd i zachowanie jej okna. Obiekty klas QT wykorzystane w klasie: **QComboBox**, **QLabel**, **QPushButton**, **QTextBrowser, QLineEdit, QRadioButton.**

`class Remote : public Remoting`

Klasa pochodna po Remoting, definiuje m.in. metody pozwalające na dodawanie, usuwanie i wyświetlanie informacji o zdalnych repozytoriach. Metoda *otworz()* w tej klasie odpowiedzialna jest za wyświetlanie informacji o zdalnych repozytoriach, dodanych do danego repozytorium lokalnego GIT'a. Jako, że klasa ta korzysta z formularza .ui klasy nadrzędnej, sama klasa nie wykorzystuje obiektów klasy QT (tylko nadrzędne).

`class Fetch : public Remoting`

Klasa pochodna po Remoting, definiuje pola i metody związane z pobieraniem zmian ze zdalnych repozytoriów do lokalnego oraz łączeniem ich z lokalnymi repozytoriami. Metoda *otworz()* przedefiniowania w klasie odpowiedzialna jest za pobieranie zmian z wybranego zdalnego repozytorium. Obiekty klas QT wykorzystane w klasie: **QComboBox**, **QLabel**, **QPushButton**, **QTextBrowser.**

`class Clone: public Remoting`

Klasa pochodna po Remoting, pozwala na wykonywanie operacji związanych z klonowaniem repozytoriów w nowe repozytoria. Metoda otworz() przedefininowana z klasy bazowej odpowiada za klonowanie jednego rezpozytorium w nowe. Obiekty klas QT wykorzystane w klasie: **QLabel**, **QPushButton**, **QTextBrowser, QTreeView, QRadioButton, QlineEdit.**

`class Push : public Remoting`

Klasa pochodna po Remoting, odpowiada za definiowanie metod i pól odpowiedzialnych za „wciskanie" aktualnego repozytorium, aktualnych zmian na repozytorium zewnętrzne, zdalne. Metoda otworz() przedefiniowana z klasy bazowej pozwala na wysyłanie, uaktualnianie zdalnego repozytorium o zmiany z naszego lokalnego. Obiekty klas QT wykorzystane w klasie**: QLabel, QPushButton, QTextBrowser, QComboBox.**

`class UserSettings : public QDialog, public git`

Klasa definiuje metody i pola związane z tworzeniem obiektu pozwalającego na edycję ustawień programu GitUI oraz modyfikację ustawień repozytorium GIT'a. Pozwala na
- ustawienie aktualnie otwartego repozytorium na domyślne – wykorzystywany jest do tego obiekt klasy *QFile* tworzący powiązanie z plikiem tekstowym r*epozytorium.txt*, który musi znajdować się w katalogu z programem GitUI. Za pomocą strumieni wejścia-wyjścia, obiekt klasy *UserSettings* pozwala zapisywać aktualną ścieżkę obsługiwanego repozytorium jak i ją odczytywać.

- zmianę nazwy oraz adresu e-mail użytkownika obsługującego repozytorium – poprzez wykorzystanie odpowiednich funkcji GIT'a
- ustawienie edytora tekstowego wykorzystywanego w tworzeniu „commitów” - poprzez wykorzystanie odpowiedniego polecenia GIT'a

Obiekty klas QT wykorzystane w klasie: **QLabel, QLineEdit, QpushButton.**

`class HelpIndeks : public QDialog`

Klasa odpowiedzialna za tworzenie okna wyświetlającego pomoc w obsłudze programu. Klasa wykorzystuje obiekt QT klasy **QWebView**, który wczytuje plik *HelpIndeksGitUi.html* zawierający treść indeksu i wyświetla w oknie. Plik .html musi znajdować się w folderze wraz z programem GitUI, bowiem obiekt klasy HelpIndeks sprawdza aktualną ścieżkę programu wykonywalnego i na tej podstawie określa ścieżkę pliku *HelpIndeksGitUi.html.*

Obiekty klas QT wykorzystane w klasie**: QLabel, QPushButton, QWebView.**

`class About : public QDialog`

Klasa określa metody i pola obiektu będącego oknem zawierającym informacje o programie i autorze programu. Obiekty klasy QT wykorzystane w klasie: **QLabel, QpushButton**.

`class git`

Klasa określa metody zwracające odpowiednio zmodyfikowane komendy GIT'a w postaci obiektu **QString**, które mogą być wykorzystane przez obiekt klasy **QProcess** do wywoływania odpowiednich komend w pozostałych obiektach. Klasy wywołujące komendy GIT'a dziedziczą po niej, dzięki czemu mają dostęp do wszystkich metod w niej zaimplementowanych.

Informacje dotyczące kompilacji i uruchamiania:

Wykorzystane środowisko: **QT Creator 2.7.1** based on QT 5.0.2 (64 bit)
Wykorzystany system operacyjny: Arch Linux 64-bit, kernel version 3.7.7-1
Wykorzystana wersja QT: 4.8.4
Wykorzystana wersja programu git: version 1.8.3.1

Wykorzystane techniki obiektowe:
- **dziedziczenie** – klasy ***Push, Clone, Remote*** oraz ***Fetch*** dziedziczą po abstrakcyjnej klasie Remoting, korzystając z jej metod oraz pól.
- **polimorfizm** – zastosowanie , przykład (w klasie MainWindow):

```
remot= new Remote(sciezkaGita,this);
remot->show();
```

gdzie *Remoting* * remot; - wskaźnik na obiekt klasy *Remoting*. Jako, że klasy pochodne dziedziczą metodę czysto wirtualną *otworz()*, każda z nich w definicji swoich sygnałów i slotów wywołuje odpowiednią metodę (swoją metodę) mimo że obiekt jest typu *Remoting*.
- **dziedziczenie wielobazowe** – klasy *MainWindow*, *Branch, Repository* i *UserSettings* dziedziczą po **QDialog**, ponieważ zawierają jej cechy związane z tworzeniem i wyglądem okien i jest dla nich wywoływany konstruktor klasy nadrzędnej. Ponadto, klasy te wywołują komendy GIT'a za pomocą *procesu*, przez co potrzebują mieć dostęp do odpowiednio zmodyfikowanych postaci poszczególnych komend – zapewnione je mają dzięki dziedziczeniu po klasie ***git*.**
- strumienie – klasa *UserSettings* wykorzystuje plik tekstowy do zapisywania oraz odczytywania linii tekstu z pliku, dotyczącej repozytorium automatycznie otwieranego przy starcie programu GitUI.

```
out<<ui->lineEdit_3DefaultRepo->text();
```

- gdzie *out* to strumień wyjściowy związany z plikiem *repozytorium.txt*

- **wyjątki** – program GitUI przewiduje, że użytkownik może nie mieć zainstalowanego programu GIT – wtedy nakładka jest w ogóle nie potrzebna. Dlatego przy każdym uruchomieniu wykonywane jest sprawdzenie czy program GIT jest zainstalowany:

```
 try{
        sprawdzCzyIstniejeGit();
    }
    catch(QString wyjatek){
        ui->labelGit->setText(wyjatek);

    }
```

gdzie : `if(proces->state()!=2){     //sprawdzam czy proces jest uruchomiony i gotowy do zapisu-odczytu`

```
        QString wyjatek="No git program found! Install it very quickly and come back :)";   //jesli nie rzucam wyjatek
        throw wyjatek;
    }
```

Dzięki temu zabezpieczeniu w przypadku gdy program GIT nie jest zainstalowany – *proces->state* nie zwróci wartości 2, czyli potwierdzenia gotowości do zapisu i odczytu – wyrzucany jest wyjątek w postaci obiektu **QString**.
Ponadto wyjątek jest rzucany w przypadku niepomyślnego otwarcia pliku tekstowego, w klasie *UserSettings.*

Niewykorzystane techniki zostały zaimplementowane w projekcie technikiPk4 (napisanym w czystym C++, system: Arch Linux 64-bit, kompilator GCC).

Zmiany w stosunku do założeń projektu:

Większa ilość klas w stosunku założeń projektu – nowe klasy to : *About* oraz *HelpIndeks.*

Klasy wykorzystujące polecenia GIT'a dziedziczą po klasie *git*, a nie wykorzystują jej metod statycznych.

Klasa *Remoting* jest klasą abstrakcyjną. W założeniu wszystkie klasy miały korzystać z forumlarza .ui klasy *Remoting* – w praktyce okazało się to niemożliwe lub byłoby bardzo trudne do zrealizowania – wynikało to z tego, że każda klasa miała różną ilość elementów jakie miały być zawarte w oknie i ciężko było upchać to w 1 formularz. Finalnie tylko klasa *Remote* korzysta z formularza *Remoting,* pozostałe klasy mają swój formularz.

3. Specyfikacja zewnętrzna.

Folder, w którym znajduje się plik wykonywalny *GitUI* musi zawierać dodatkowo pik *repozytorium.txt* oraz *HelpIndeksGitUi.html*– wymagane do poprawnej pracy programu GitUI.
Elementy każdego okna są intuicyjne oznaczone – opis odpowiada reakcji nakładki w związku z wywołaniem odpowiedniego działania przez użytkownika.
Guzik **Back** – zamknięcie okna.
Informacja nad *wyświetlaczem* danego okna informuje z jakim repozytorium aktualnie pracujemy.
Okna zawierają *wyświetlacze*, czyli obiekty przystosowane do wyświetlania danych – przedstawiają one zwrot od programu GIT, w reakcji na odpowiednie akcje oraz inne informacje związane z obsługą nakładki GitUI.

Wykonując operacje związane z plikami – zapisem i odczytem użytkownik musi mieć na uwadze, że GitUI nie weryfikuje możliwości zapisu czy odczytu pliku przez danego użytkownika.

W przypadku nietypowych ustawień niektóre operacje wykonywane na nakładce mogą nie

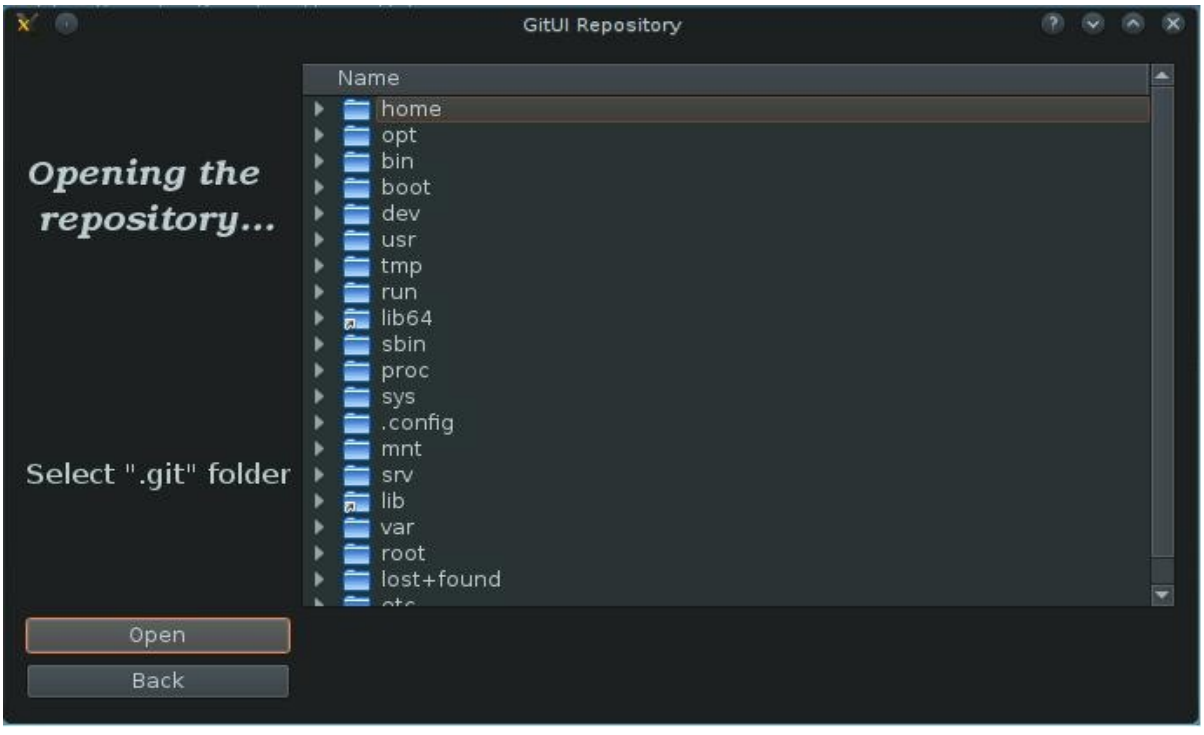
działać prawidłowo.

1. Okno główne

Okno główne pozwala na dostęp do wszystkich opcji nakładki. Zawiera pasek zadań za pomocą którego można wybrać interesującą użytkownika opcję. *Wyświetlacz* informuje o zwrotach od programu GIT oraz o informacjach związanych z operacjami i problemami.
Przyciski: *status* - „git status” , *log*- „git log”, *commit*-”git commit”.
Ready to go...! oznacza, że repozytorium jest pomyślnie otwarte i można wykonywać operacje.

Zanim będzie można uzyskać dostęp do wszystkich opcji zarówno nakładki jak i programu GIT, trzeba wykonać pierwszy *commit*. Dopiero wtedy tworzona jest gałąź **master* i można uzyskać dostęp do wszystkich opcji programu git.

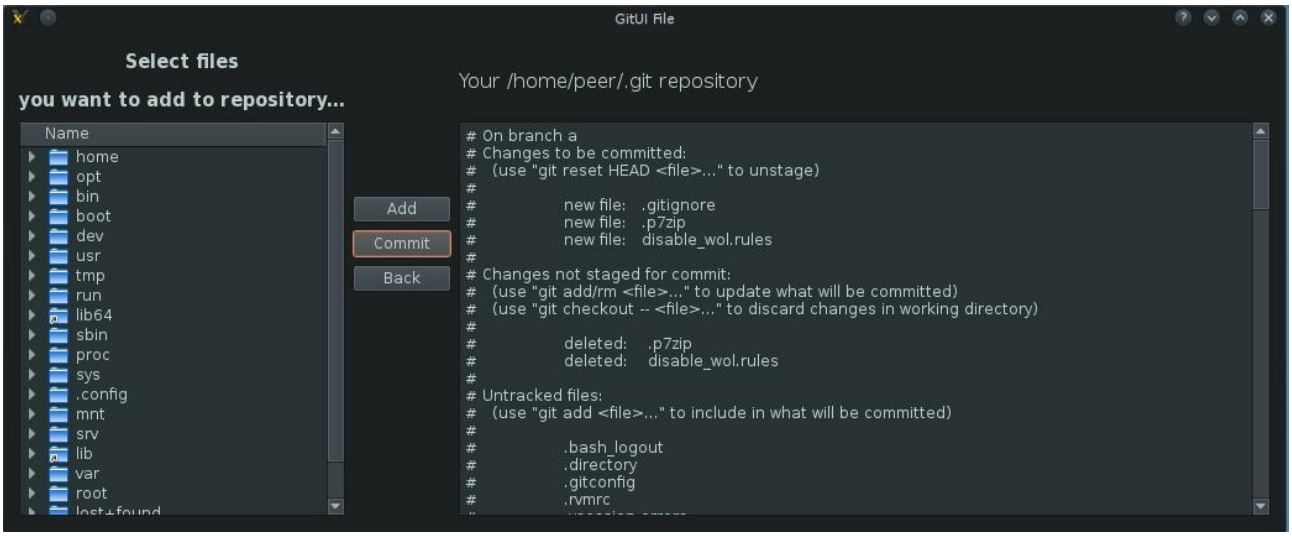
2. Repository.



Okno pozwala na otwieranie, tworzenie, usunięcie repozytorium „.git” w zależności od opcji wybranej z *MainWindow*. Po wybraniu ścieżki z drzewa plików, wciśnięcie odpowiedniego guzika spowoduje określoną reakcję, w wypadku wyżej – otwarcie repozytorium.

Przy usuwaniu repozytorium zostanie wyświetlone okno, które wymaga potwierdzenia czy operacja na pewno ma być wykonana. Na tym etapie trzeba potwierdzić, bądź można zrezygnować z operacji usuwania repozytorium.

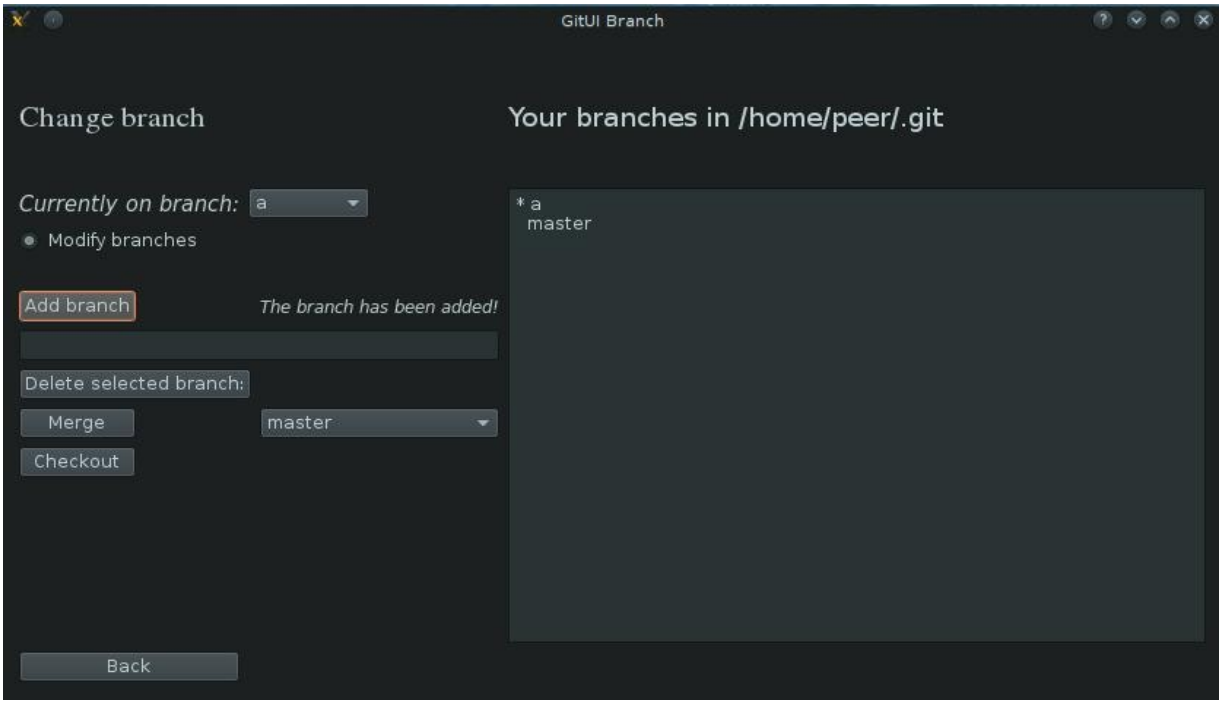
3. File.



Okno pozwala na dodanie plików do śledzonych przez GIT'a. Po otwarciu od razu uruchamiane jest polecenie „ git log” a wynik wyświetlany w oknie. Wybrane pliki mogą być wybrane z drzewa plików, przy czym możliwe jest wybranie większej ilości plików poprzez zaznaczenie ich. Po wybraniu użytkownik może kliknąć przycisk *Add* aby dodać pliki do śledzonych przez GIT. Ekran pokazuje reakcję GIT'a na określone działania.

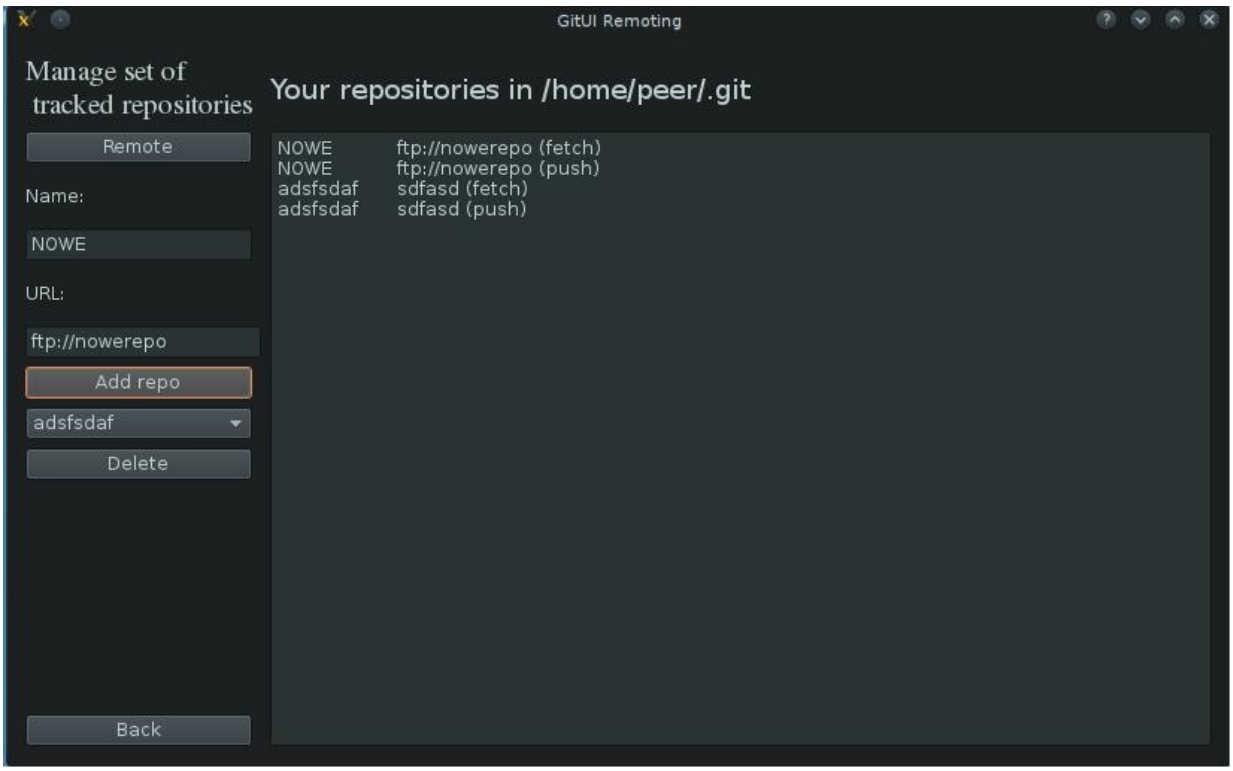
Wciśnięcie przycisku *Commit* wywoła komendę „git commit”, przy czym zostanie otwarty zewnętrzny edytor tekstu.

4. Branch.



Okno pozwala na dodawanie, modyfikowanie i usuwanie gałęzi GIT'a. *Wyświetlacz* przedstawia wynik komendy „git branch”. Powyżej ścieżka repozytorium, do którego się to odnosi.
Za pomocą *combo box'a* pierwszego możemy przeskoczyć na inny branch (wykonując polecenie *checkout*), Kliknięcie w opcje *Modify Branch*, rozszerza menu do możliwości dodania nowego repozytorium (po wpisaniu nazwy w edytorze linii) i kliknięciu w guzik *Add branch*.
Po kliknięciu guzika *Delete selected branch*, gałąź spod *combo box'a* drugiego (znajdującego się obok guzika) zostanie usunięta. Kliknięcie w guzik *Merge* spowoduje złączenie zmian wybranej gałęzi z główną. Guzik *Checkout* pozwala na przeskoczenie nie na gałąź (choć także jest to możliwe przy wpisaniu nazwy gałęzi) lecz na kod SHA1 danego commitu. Kod powinien być wpisany w linii edycji.

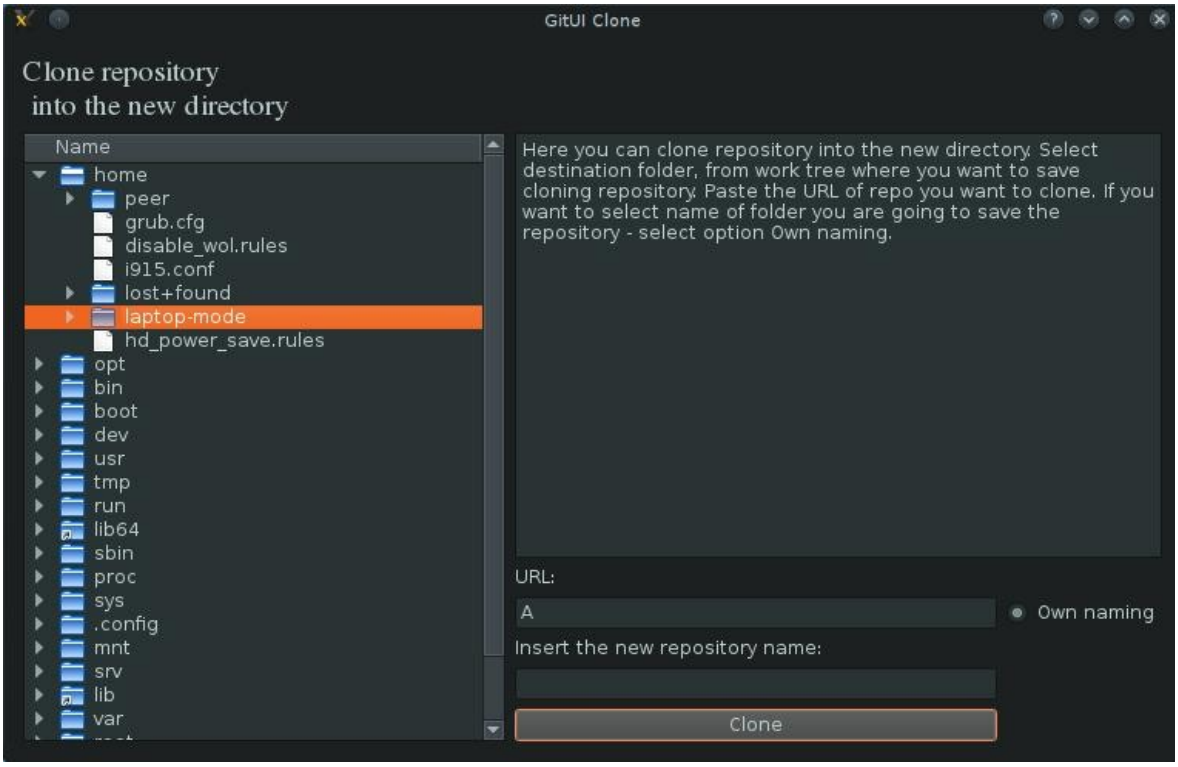
4. Remote.



Okno pozwala dodać zdalne repozytoria (znajdujące się np. na serwerze). Pozwala także usunąć te zapisane w historii danego lokalnego repozytorium. *Wyświetlacz* przedstawia zwrot GIT'a na wywołanie komendy „git remote”. W linii *Name* należy wpisać nazwę repozytorium, a w linii URL adres repozytorium, które chcemy dodać do lokalnego. Wciśnięcie klawisza *Add repo* spowoduje dodanie danego repozytorium do listy repozytoriów aktualnie otwartego (o ile nie dodano już wcześniej repozytorium o takiej nazwie).

Wciśnięcie klawisza *Delete* spowoduje usunięcie danego repozytorium z *combo box'a*.

5. Clone.

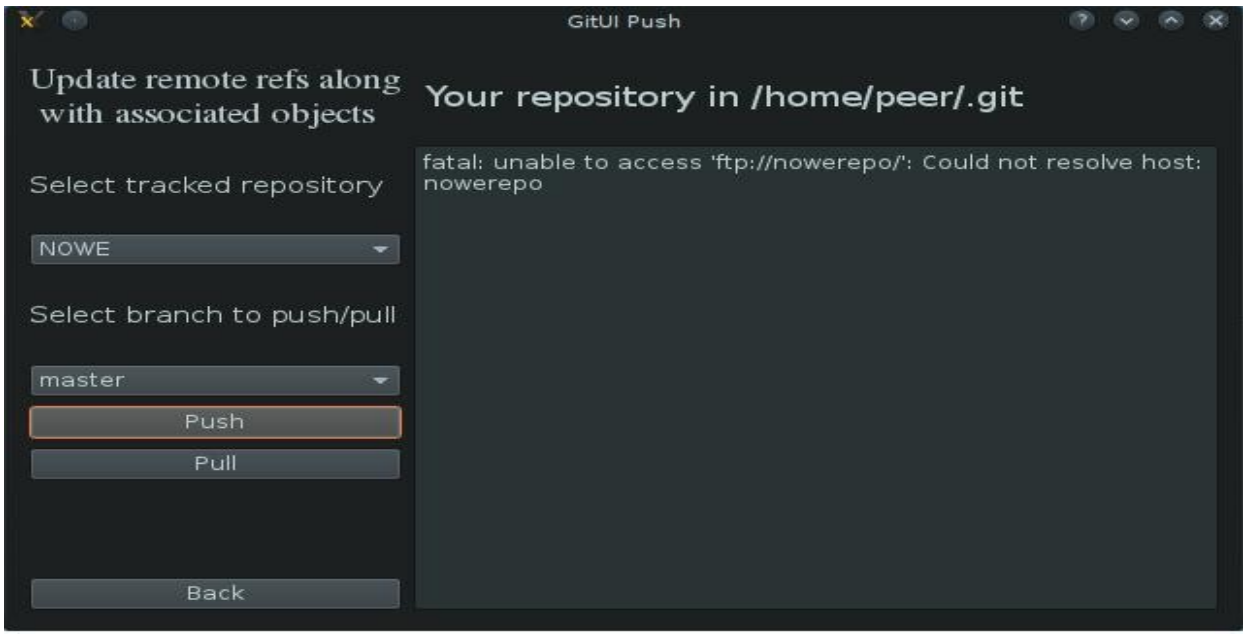


Okno pozwala na sklonowanie repozytorium w nowe. Ścieżka docelowego zapisu nowego repozytorium musi być określona w drzewie plików. W linii *URL* należy wpisać adres repozytorium, które będzie klonowane. Domyślnie w wybranej ścieżce zostanie stworzony katalog o nazwie repozytorium które klonujemy, a w nim dopiero zainicjowany katalog „.git”.

Użytkownik ma możliwość własnego nazwania folderu, w którym zostanie zainicjowany „.git”. Aby to zrobić należy wybrać opcję *Own naming* (*radio button*) i wpisać w drugiej linii wybraną nazwę katalogu.

Wciśnięcie klawisza *Clone* spowoduje uruchomienie komendy „git clone” z odpowiednimi parametrami. Zwrot zostanie przedstawiony w wyświetlaczu.

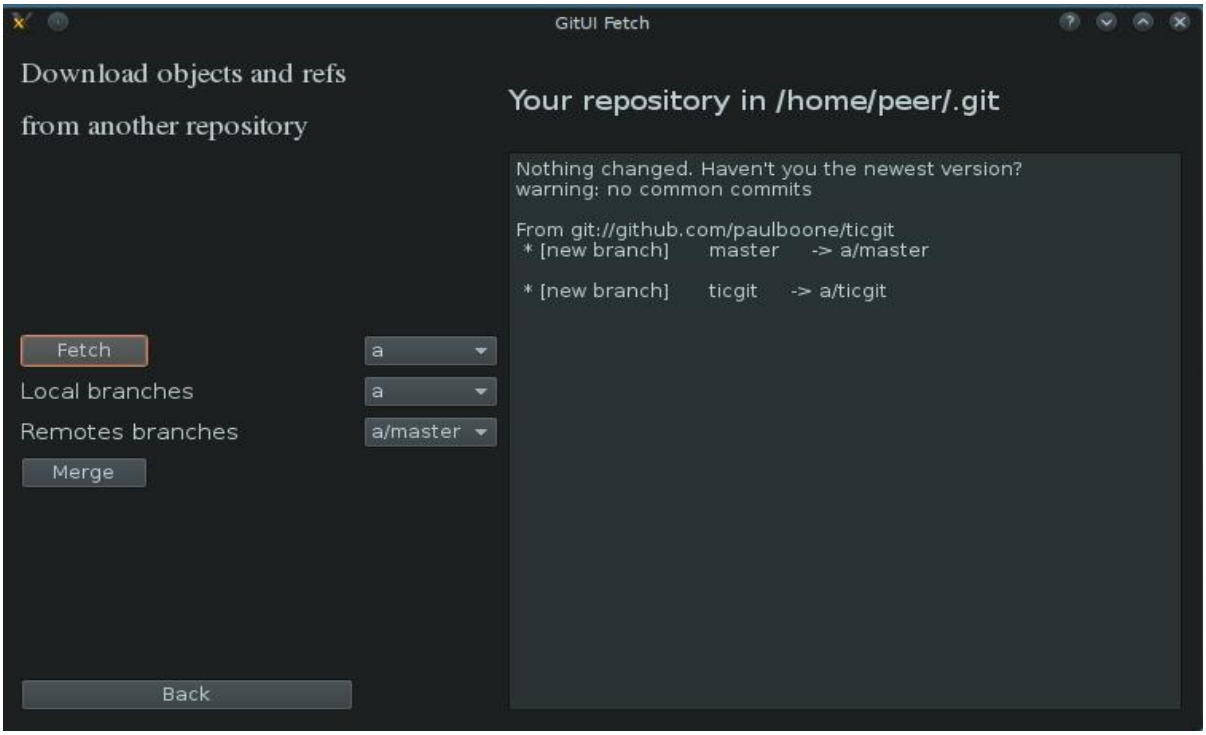
6. Push.



Okno pozwala na „wciśnięcie” aktualnych zmian na swoim lokalnym repozytorium – na zewnętrzne repozytorium. Docelowe repozytorium należy wybrać z *combo box'a Select tracked repository*. Nazwa gałęzi, związana ze zmianami jakie mają być „wciśnięte” do zdalnego repozytorium musi być wybrana w drugim *combo box'ie*. Wciśnięcie guzika *Push*, spowoduje wywołanie komendy „git push” z odpowiednimi parametrami. Zwrot GIT'a można zobaczyć w *wyświetlaczu*.

Guzik *Pull* pozwala pobrać zmiany z repozytorium i automatycznie scalić je z naszym repozytorium. Wymaga to odpowiednich ustawień w plikach konfiguracyjnych GIT'a, zanim będzie można korzystać z tej opcji za pomocą nakładki GitUI.

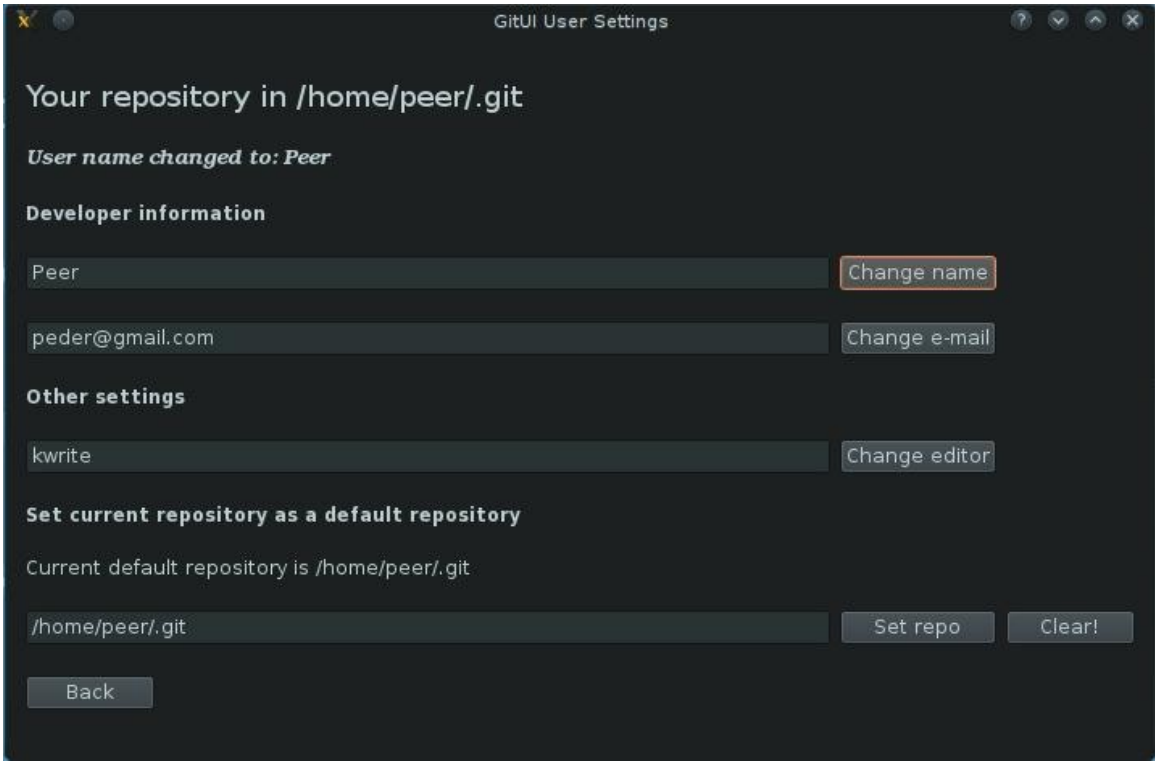
7. Fetch.



Okno pozwala na pobieranie zmian ze zdalnych repozytoriów, uaktualnianie ich ze swoim, lokalnym. Nazwa repozytorium musi być wybrana w *combo box'ie*. W przypadku gdy zmiany zostaną pomyślnie pobranie pojawią się dwa kolejne *combo box'y*: *Local branches* oraz *Remotes branches*, zawierające kolejno branche lokalnego repozytorium oraz branche repozytorium zdalnego, z którego pobieramy dane. Aby połączyć zmiany ze zdalnego repozytorium z naszym, należy wcisnąć klawisz *Merge*.

Reakcje programu GIT można śledzić w *wyświetlaczu.*

8. User settings.



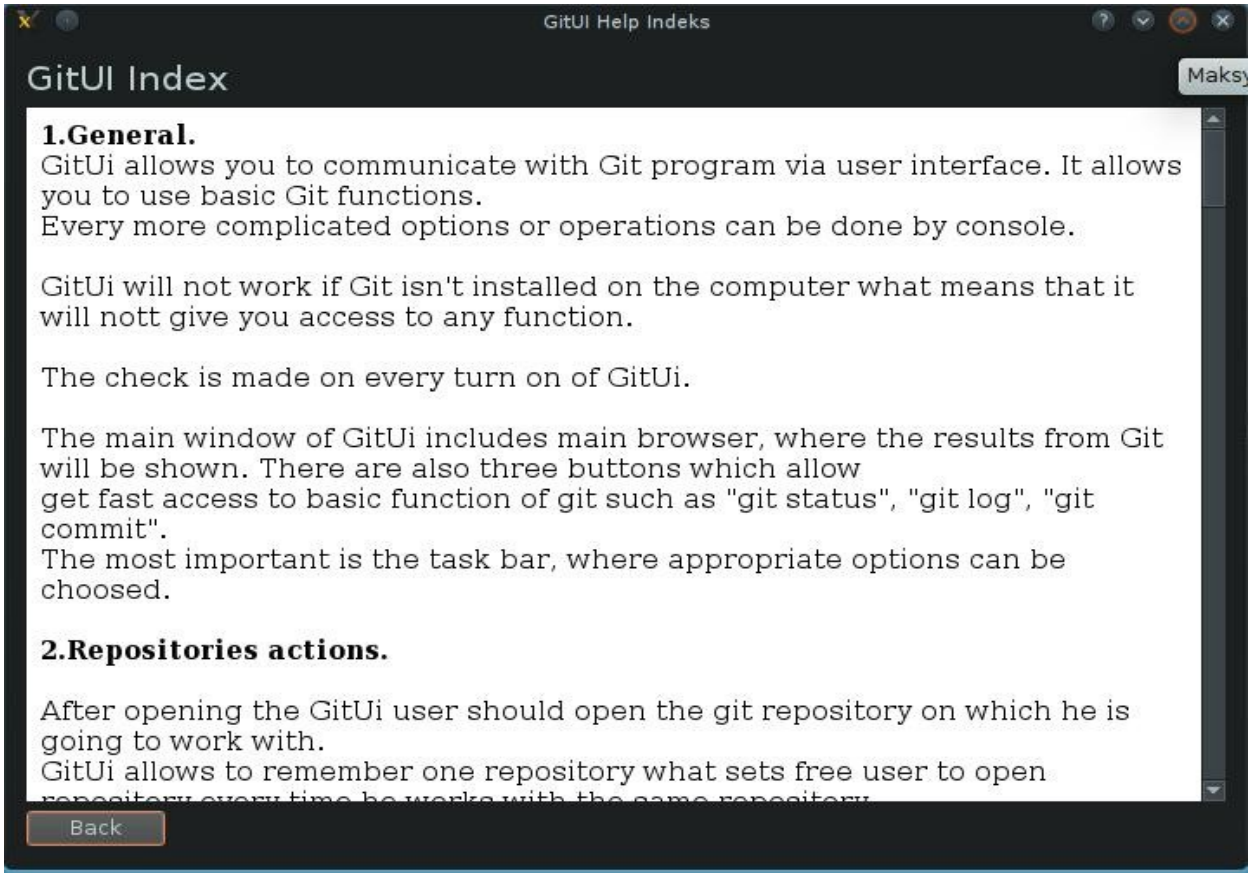
Okno pozwala na zmianę ustawień użytkownika danego repozytorium oraz nakładki GitUI. Sekcja *Developer Information* zawiera dwie linie, kolejno w których zawarte są informacje: nazwa użytkownika danego repozytorium oraz adres e-mail. Aby je zmienić wystarczy edytować daną linię oraz wcisnąć odpowiedni klawisz *Change name* lub *Chane e-mail*.

Sekcja *Other settings* pozwala na zmianę obsługiwanego przez nakładkę edytora tekstu. Jego nazwę należy wpisać w linii pod napisem *„Other settings”* oraz potwierdzić operację guzikiem *Change editor*.

Ostatnia sekcja *Set current repository as a default repository* pozwala na zapisanie aktualnie otwartego repozytorium w pliku *repozytorium.txt* (znajdującego się w katalogu z plikiem wykonywalnym programu). Dzięki zapisanej ścieżce, nie będzie trzeba za każdym razem otwierać repozytorium, tylko zostanie ono automatycznie otwarte po włączeniu programu GitUI.

Aby ustawić dane repozytorium jako domyślnie należy wcisnąć przycisk *Set repo,* aby usunąć ustawienia dotyczące domyślnego repozytorium należy wcisnąć klawisz *Clear!*

9. Indeks.



Okno zawiera instrukcję obsługi programu GitUI. *Wyświetlacz* przedstawia zawartość pliku *HelpIndeksGitUi.html*, który musi znajdować się w folderze z plikiem wykonywalnym. *Rolka* obok *wyświetlacza* pozwala na wygodne przesuwanie względem tekstu.

10. About.

Okno zawiera informacje na temat wersji nakładki oraz autora.

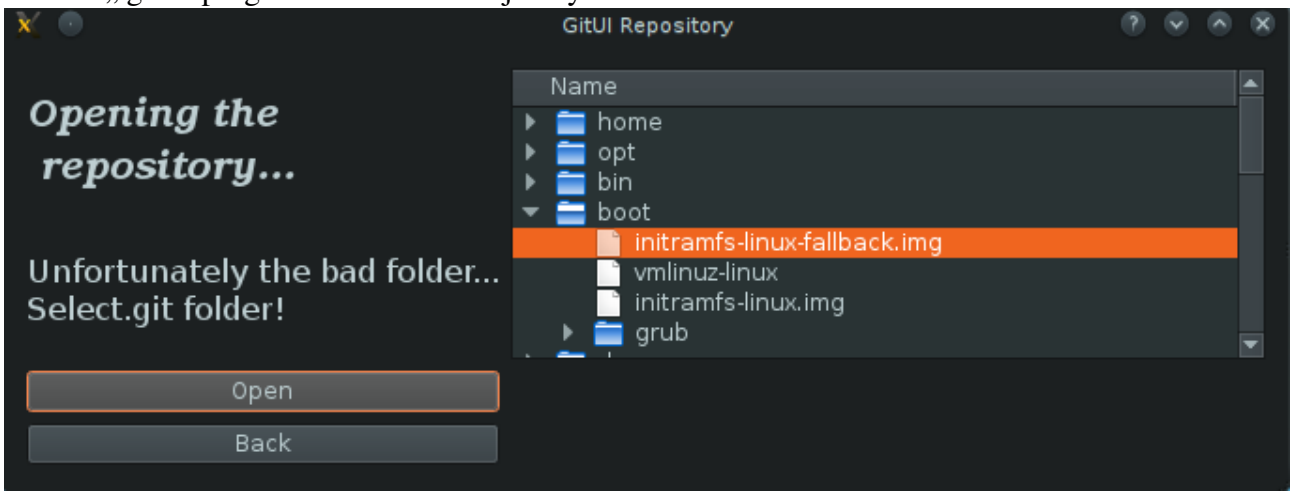
4. Testy.

Program testowany był pod wieloma kątami, zarówno pod kątem poprawności działania jak i nieprzewidzianych akcji użytkownika oraz czynników zewnętrznych (np. usunięcie plików).

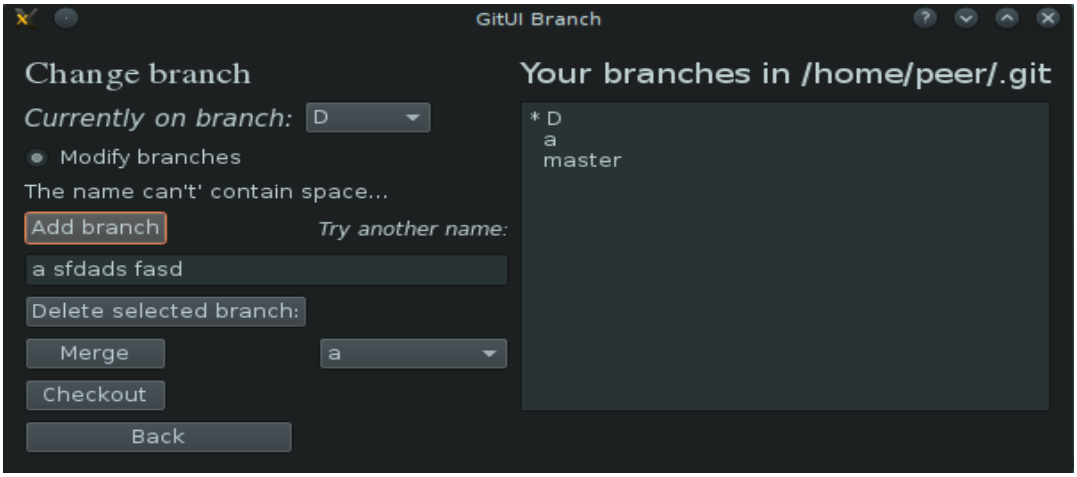
W przypadku gdy program GIT nie jest zainstalowany na komputerze, GitUI informuje o tym:

żadne opcje nie są dostępne.

W sytuacji gdy w momencie wyboru repozytorium, użytkownik otworzy plik zamiast folderu „.git” - program GitUI informuje o tym:

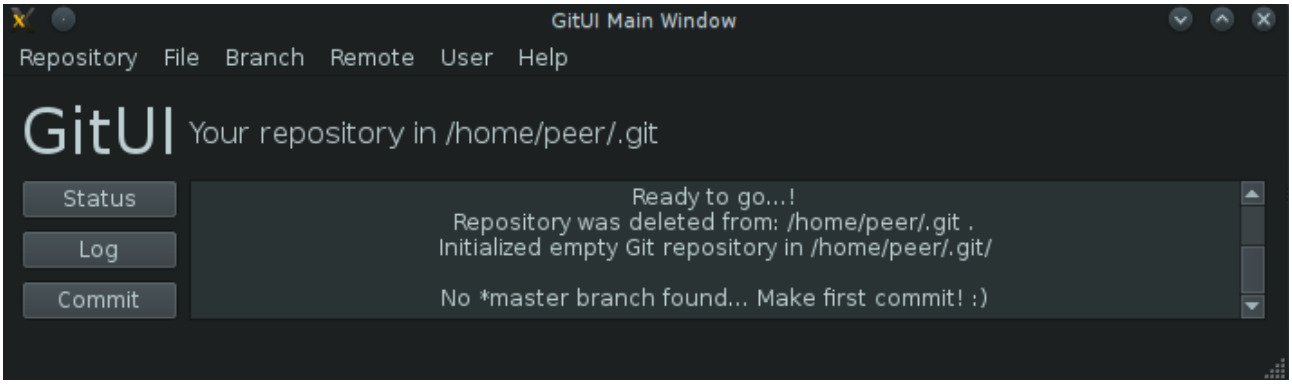


W sytuacji gdy przy dodawaniu nowej gałęzi użytkownik wpisze nieprawidłową postać nazwy gałęzi, GitUI informuje o tym:



Opcje powiadamia o nieprawidłowości wpisywanych danych bądź o ich braku są zaimplementowane w każdym oknie.

W sytuacji gdy dane repozytorium nie ma wykonanego pierwszego *commitu*, użytkownik nie ma dostępu do żadnych opcji programu związanych z repozytorium, prócz *File* , gdzie może dodać pliki do śledzonych przez GIT'a oraz zrobić pierwszy *commit*.



Jak widać użytkownik jest poinformowany o braku pierwszego *commitu*. Należy go wykonać aby uzyskać dostęp do wszystkich funkcji.

Teoretycznie niepoprawne działanie nakładki GitUI jest możliwe w sytuacji gdy dana komenda GIT'a nie zadziała, a okno ma zaimplementowaną aktualizacje elementów swojego interfejsu niezależnie od zwrotu programu GIT. Taka sytuacja może zajść (choć nie zdarzyła się), lecz otwarcie takiego okna ponownie spowoduje wczytanie poprawnych danych na podstawie zwrotu z programu GIT.

Program GitUI może także przestać działać poprawnie w momencie gdy format instrukcji programu GIT się zmieni.

Znane błędy:

Program GitUI nie weryfikuje praw użytkownika do zapisu/odczytu z danej ścieżki. Wybranie niedozwolonej ścieżki przy tworzeniu repozytorium (do zapisu w której użytkownik nie ma praw) spowoduje zamknięcie okna i przekazanie ścieżki do okna głównego, mimo że repozytorium nie zostało zainicjowane. Ścieżka ta będzie widoczna nad wyświetlaczem, mimo że nie jest poprawna. Choć użytkownik nie ma w tym momencie dostępu do żadnych funkcji programu git, jest to błąd działania.

Podobna sytuacja pojawia się w trakcie dodawania plików do śledzonych przez GIT'a, gdzie w sytuacji dodania plików, do których użytkownik nie ma prawa – pliki nie zostaną dodane, ale można to stwierdzić tylko i wyłącznie na podstawie zwrotu od programu GIT, który po prostu się nie zmieni. Program GitUI nie zawiera funkcji weryfikujących praw do plików.

Informacja *Ready to go...!* jest wyświetlana każdorazowo, przy kliknięciu klawisza *Commit* w oknie File (wyświetlana jest w oknie *MainWindow*). Związane jest to z weryfikacją programu, dotyczącą istnienia gałęzi **master*, powstałej przy tworzeniu pierwszego commitu.

5. Uwagi końcowe.

Realizacja projektu przebiegła zgodnie z założeniem projektowym, nieznacznie tylko, odbiegając od planu. Ilość klas nieznacznie wzrosła, główne metody zostały nie zmienione.

Implementacja przekroczyła ramy planu pracy a także przewidywany czas realizacji zwiększył się, była potrzebna większa ilość czasu.

Wnioski dotyczące projektu:

- na zakładany czas realizacji zawsze trzeba wziąć poprawkę, zazwyczaj ciężko zmieścić się w mocno okrojonym czasie
- implementacja projektu zgodnie ze stworzonym wcześniej założeniem i planem może być pomocna i usprawniać pracę, jeśli dokument projektowy został zrobiony z odpowiednią starannością. Stworzenie szkieletu a także określenie ram programu jest trudne i wymaga dużej ilości czasu na przemyślenia, przed realizacją

Dzięki wykorzystaniu frameworku QT, nauczyłem się używać tej biblioteki tworząc aplikacje *GUI* a także poznałem obiektowe techniki programowania.