

System kontroli wersji GIT

1. Wprowadzenie

System kontroli wersji (ang. version/revision control system) – oprogramowanie służące do śledzenia zmian głównie w kodzie źródłowym oraz pomocy programistom w łączeniu zmian dokonanych w plikach przez wiele osób w różnym czasie.

1.1. Linus Torvalds – twórca systemu git

GIT to system kontroli wersji, będący wolnym oprogramowaniem o otwartym kodzie. Projektowany był z myślą o jednoczesnej pracy wielu ludzi nad jednym kodem.

System GIT stworzył Linus Torvalds jako narzędzie wspomagające rozwój jądra Linux ponieważ wszystkie istniejące (darmowe) systemy kontroli wersji były niewystarczające. System ten miał być rozproszony, szybki, miał chronić przed błędami w repozytorium i nie posiadać wad CVS-a.

Prace nad Gitem rozpoczęły się 3 kwietnia 2005 roku, projekt został ogłoszony 6 kwietnia, 7 kwietnia Git obsługiwał kontrolę wersji swojego własnego kodu, 18 kwietnia pierwszy raz wykonano łączenie kilku gałęzi kodu, 27 kwietnia Git został przetestowany pod względem szybkości z wynikiem 6,7 lat na sekundę, a 16 czerwca Linux 2.6.12 był hostowany przez Gita.

Jak widać w pełni funkcjonalny system kontroli wersji powstał w niecały miesiąc. A ojcem tego sukcesu jest oczywiście sam Linus Torvalds, który większość projektu wykonał sam!

1.2. Najważniejsze cechy GITa

Do najważniejszych cech systemu GIT należą:

- Dobre wsparcie dla rozgałęzionego procesu tworzenia oprogramowania: jest dostępnych kilka algorytmów łączenia zmian z dwóch gałęzi, a także możliwość dodawania własnych algorytmów.
- Praca off-line: każdy programista posiada własną kopię repozytorium, do której może zapisywać zmiany bez połączenia z siecią; następnie zmiany mogą być wymieniane między lokalnymi repozytoriami.
- Wsparcie dla istniejących protokołów sieciowych: dane można wymieniać przez HTTP(S), FTP, rsync, SSH.
- Efektywna praca z dużymi projektami: system Git według zapewnień Torvaldsa, a także według testów fundacji Mozilla, jest o rzędy wielkości szybszy niż niektóre konkurencyjne rozwiązania.
- Wsparcie dla nieliniowego programowania (branche)
- Adresowanie przez zawartość (SHA-1)

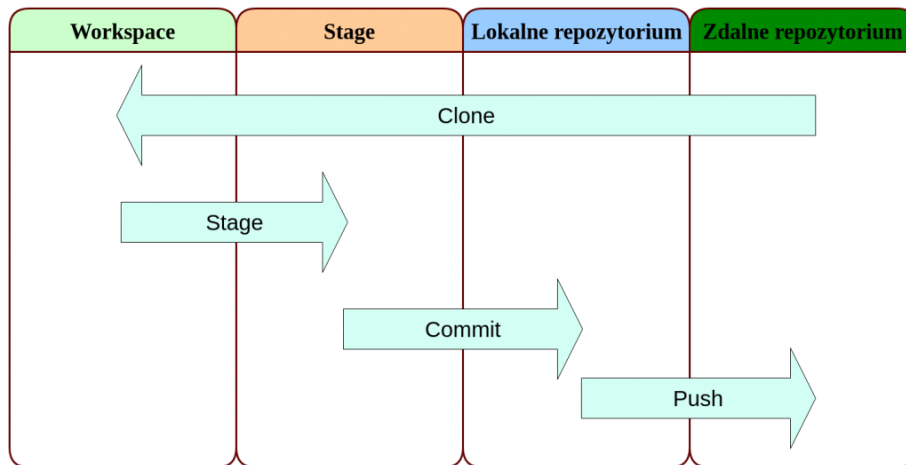
1.3. Stany plików w GIT

Aby móc pracować z GITem trzeba zrozumieć, w jakich stanach mogą znajdować się zarządzane przez system pliki. Git wprowadza trzy główne stany dla zmian: zmodyfikowany, śledzony oraz zatwierdzony.

- zmodyfikowany – plik był edytowany, ale zmiana o tym nie została jeszcze nigdzie zapisana;
- śledzony – zmodyfikowany plik został oznaczony do zatwierdzenia przy najbliższej operacji commit;
- zatwierdzony – dokonana zmiana została zapisana i utrwalona w lokalnej bazie danych;

Przesłanie zmian do zdalnego repozytorium jest już operacją opcjonalną.

Stany GIT



- katalog Git – to trzon lokalnego repozytorium. W nim Git przechowuje metadane o plikach oraz obiektową bazę danych. Ten katalog jest kopiowany podczas klonowania repozytorium.
- katalog roboczy – jest to odtworzony obraz wersji projektu. To właśnie zawartość tego katalogu jest modyfikowana przez użytkownika.
- przechowalnia (stage) – to miejsce pośrednie, między katalogiem roboczym, a lokalną bazą danych. Dzięki niej można utrwalić tylko wybrane zmiany.

1.4. Terminologia

Podczas pracy z GITem możemy spotkać się z następującym nazewnictwem:

- **Branch** - równoległa gałąź projektu rozwijana oddzielnie od głównej.
- **Tag** – marker konkretnej wersji (rewizja w SVN'ie) projektu.
- **Working Dir** – katalog roboczy na którym pracujemy
- **Index** – rodzaj „cache”, czyli miejsca gdzie trzymane są zmiany do commita
- **Master Branch** – główny branch z którym łączymy (merge) nasze zmiany przed wysłaniem do zdalnego repozytorium.
- **Development Branch** – gałąź na której łączone są gałęzie feature. Przy wyjściu kolejnej wersji mergowany jest z Master Branchem.
- **Feature Branch** – gałąź na której rozwijane jest konkretne narzędzie bądź dodatek do głównego projektu.
- **HotFix** – branch tworzony na potrzeby szybkich poprawek, naprawienia niezgodności lub bugu.

1.5. Obiekty GITa

- **Commit** – wskazuje na tree oraz ojca, zawiera przykładowo takie informacje jak autor, data i treść wiadomości.
- **Tree** – reprezentuje stan pojedynczego katalogu (lista obiektów blob oraz zagnieżdżonych obiektów tree)
- **Blob** – zawiera zawartość pliku bez żadnej dodatkowej struktury
- **Tag** – wskazuje na konkretny commit oraz zawiera opis taga.

1.6. Git Bash

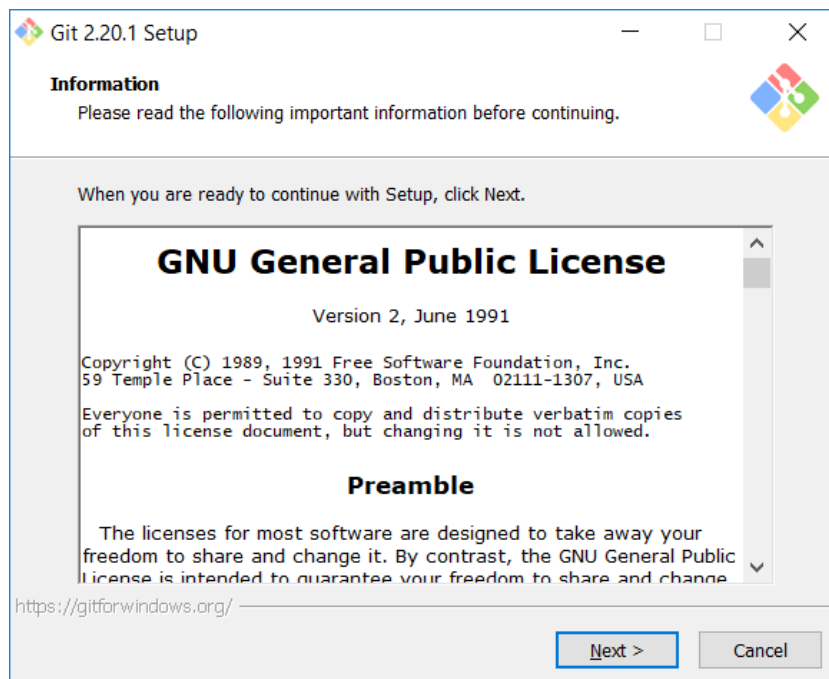
Git Bash to konsola systemu GIT, która umożliwia w pełni funkcjonalne zarządzanie repozytorium. Za pomocą odpowiednich komend konsolowych można m.in. tworzyć repozytorium, dodawać pliki, śledzić zmiany itd.

Do najważniejszych poleceń Git Basha należą:

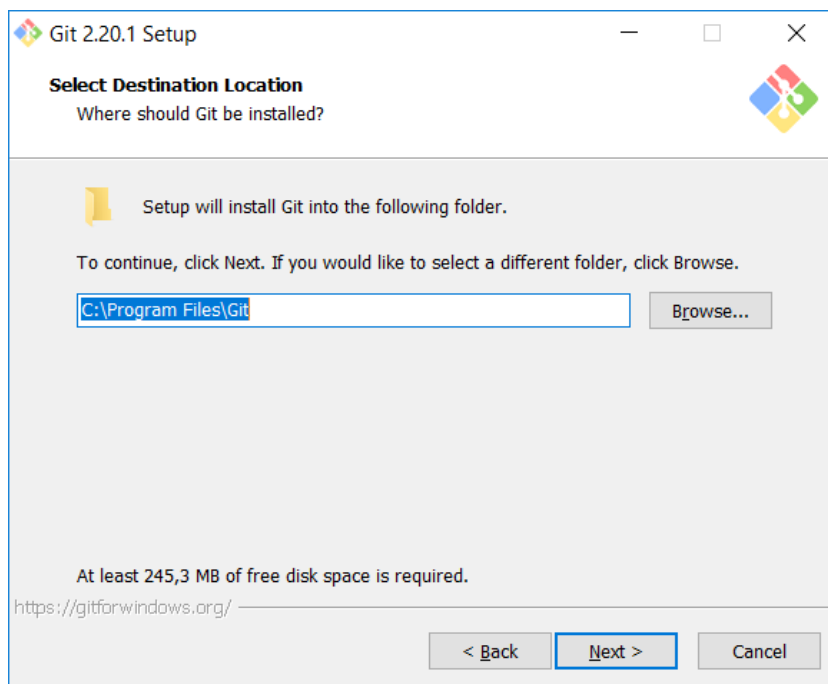
- **git init [nazwa]** – tworzy nowe repozytorium lokalne.
- **git clone origin [link]** – klonuje repozytorium z serwera zdalnego na komputer.
- **git remote add origin [link]** – dodaje repozytorium zdalne do repozytorium lokalnego.
- **git add [plik]** – dodaje wszystkie zmienione pliki do staged area
- **git checkout -- .** – usuwa wszystkie pliki z staged area.
- **git branch [nazwa]** – tworzy nowy branch
- **git checkout -b [nazwa]** – tworzy nowy branch i ustawia go jako aktualny branch.
- **git checkout [nazwa]** – przełącza na wybrany branch
- **git commit -m 'tytuł' -m 'opis'** – tworzy commit z plików w staged area o wybranym tytule oraz opisie.
- **git push** – wypycha zmiany lokalne na serwer zdalny
- **git pull** – zaciąga oraz przyłącza zmiany z serwera zdalnego na serwer lokalny
- **git fetch** – pobiera zmiany z repozytorium zdalnego, ale nie przyłącza ich do working directory.
- **git merge** – łączy zmiany z dwóch różnych branchy, ścieżek, lub zmiany pobrane z repozytorium zdalnego za pomocą git fetch
- **git stash save [plik]** – dodaje zmieniony plik do schowka
- **git stash apply** – dodaje zmiany ze schowka do working directory.

2. Instalacja GITa

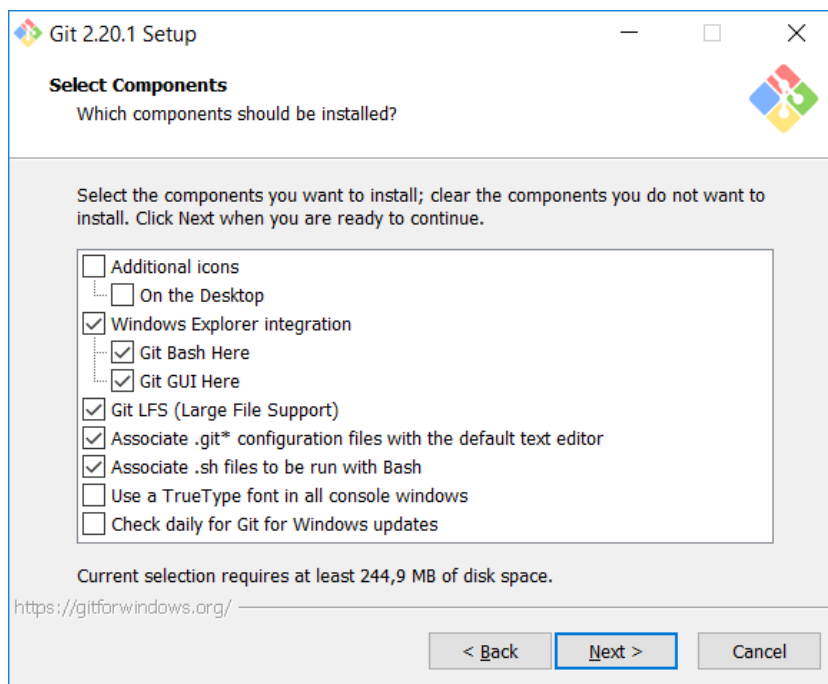
Klikamy **Next**:



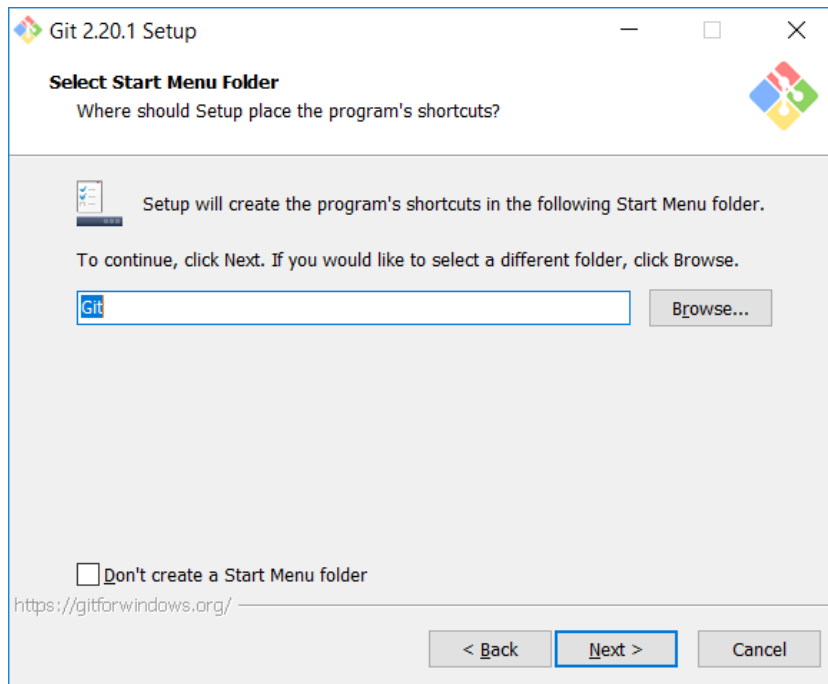
Wybieramy folder instalacji i klikamy **Next**:



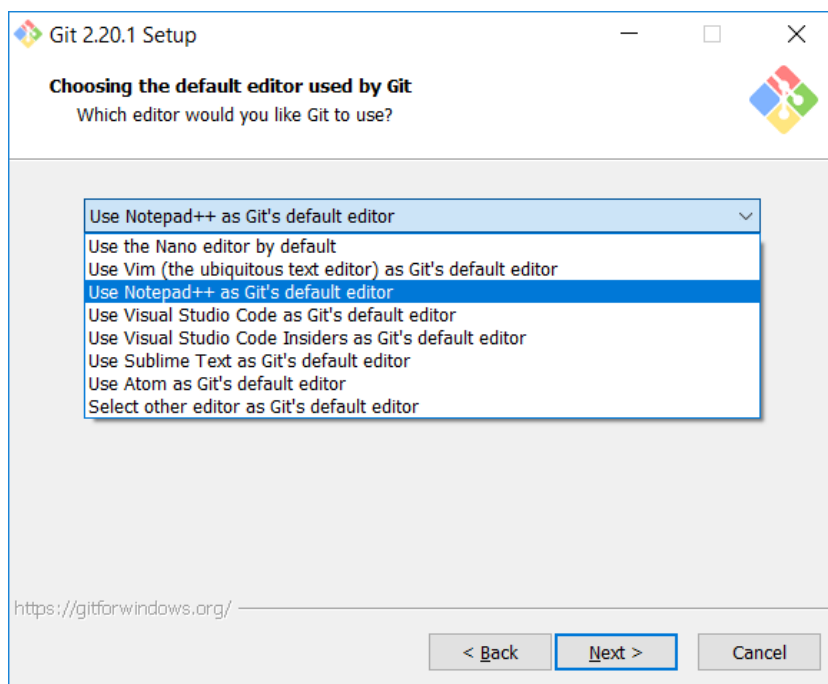
Wybieramy komponenty do zainstalowania (pozostawiamy domyślne) i klikamy **Next**:



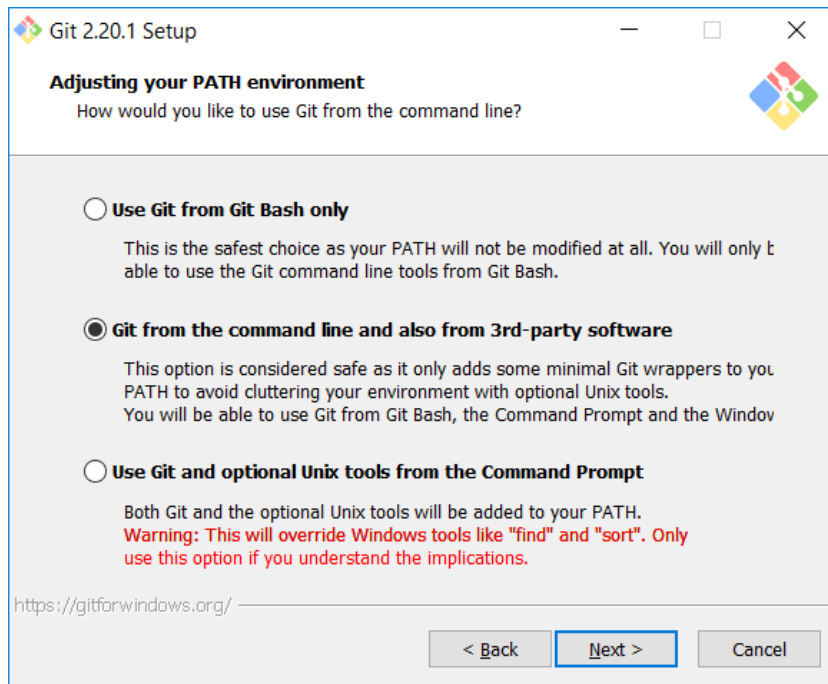
Tworzenie folderu dla **Gita** w menu **Start**:



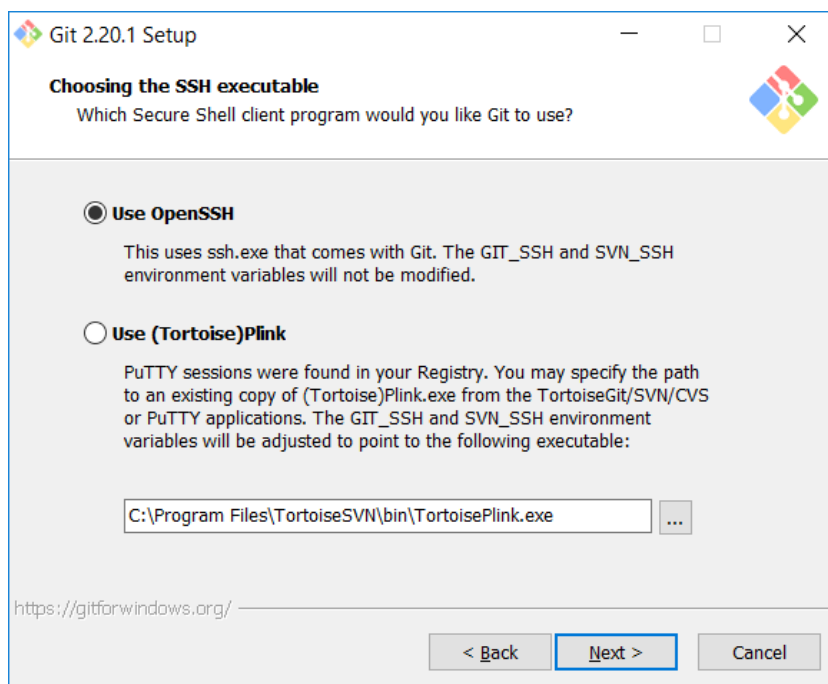
Wybieramy edytor dla Gita np. **Notepad++**:

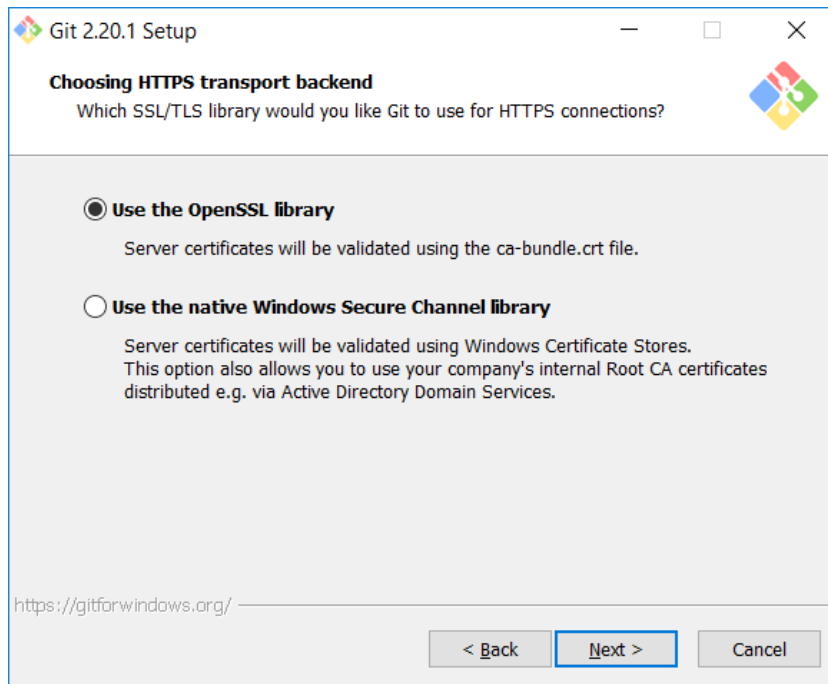


Wybieramy ustawienia zmiennej środowiskowej **PATH**:

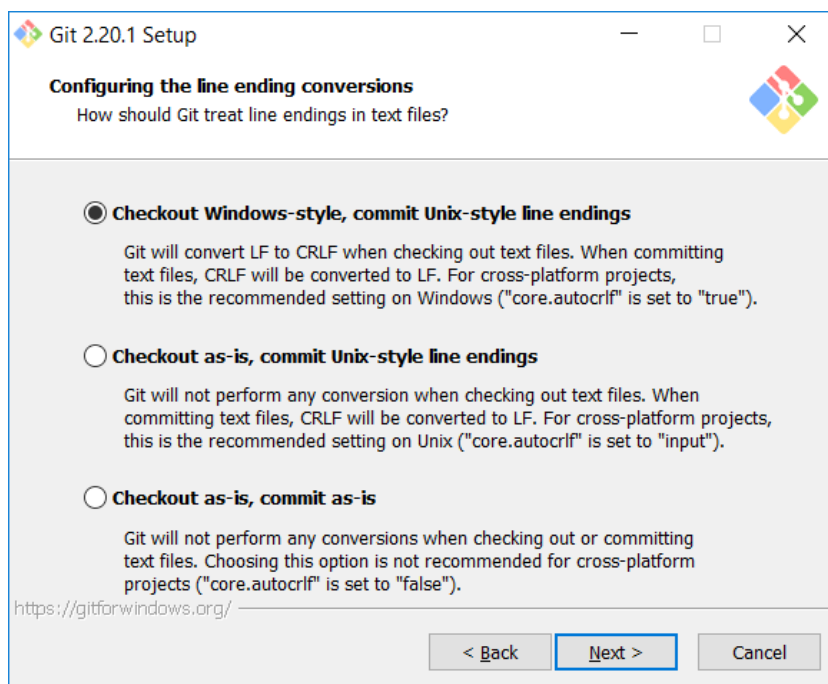


Wybieramy ustawienia sieciowe dla Gita:

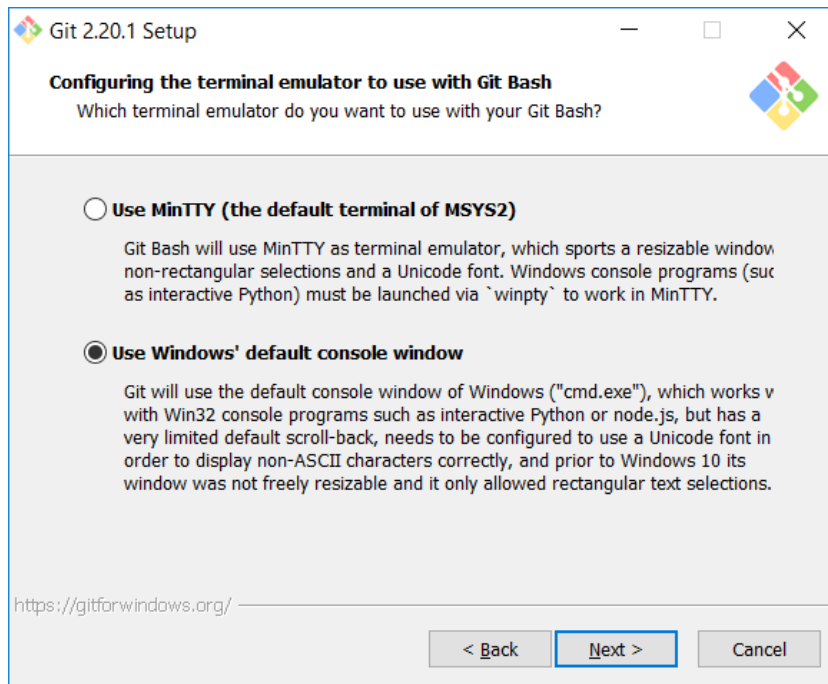




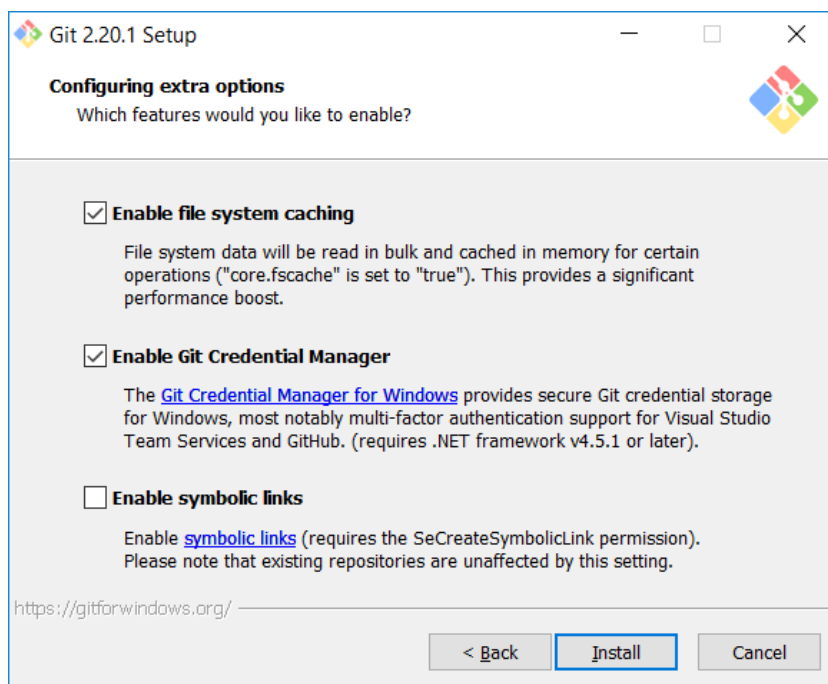
Wybieramy ustawienia znaku końca wiersza dla plików tekstowych:



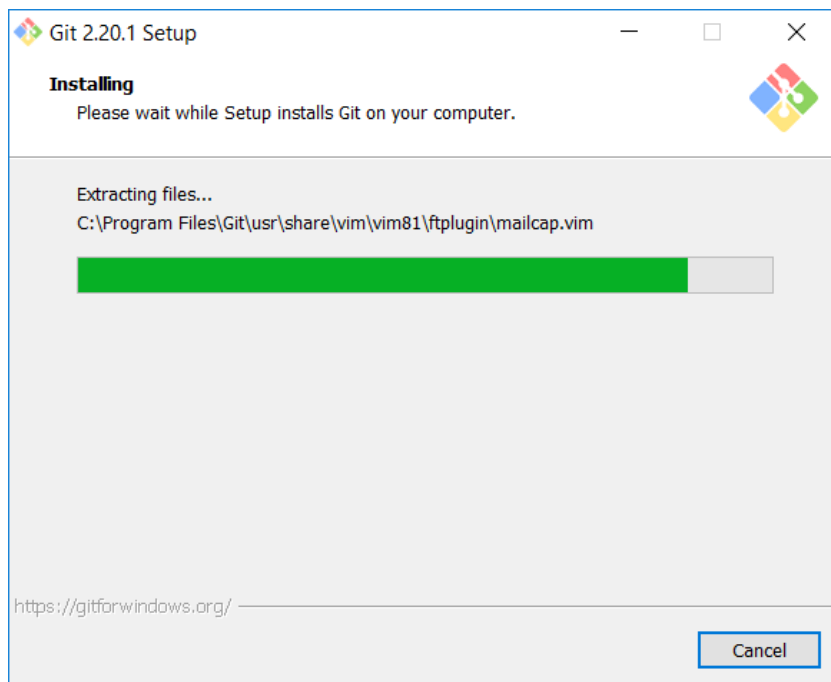
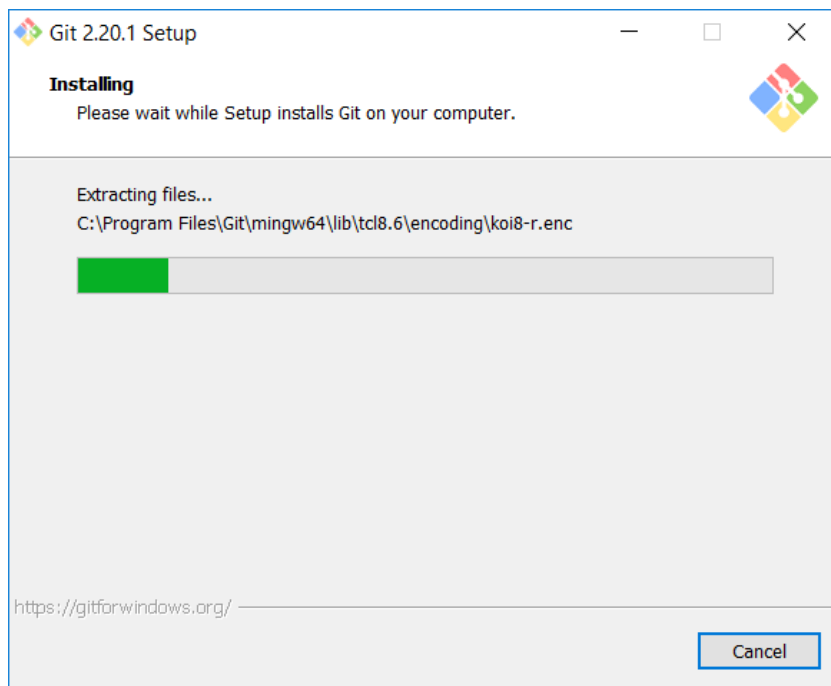
Wybieramy terminal:



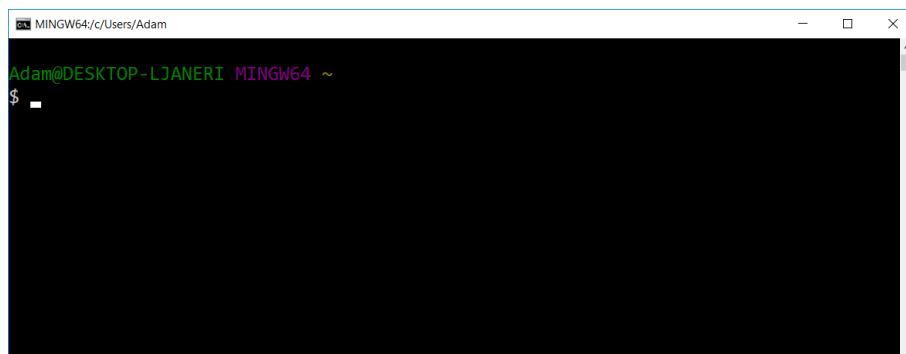
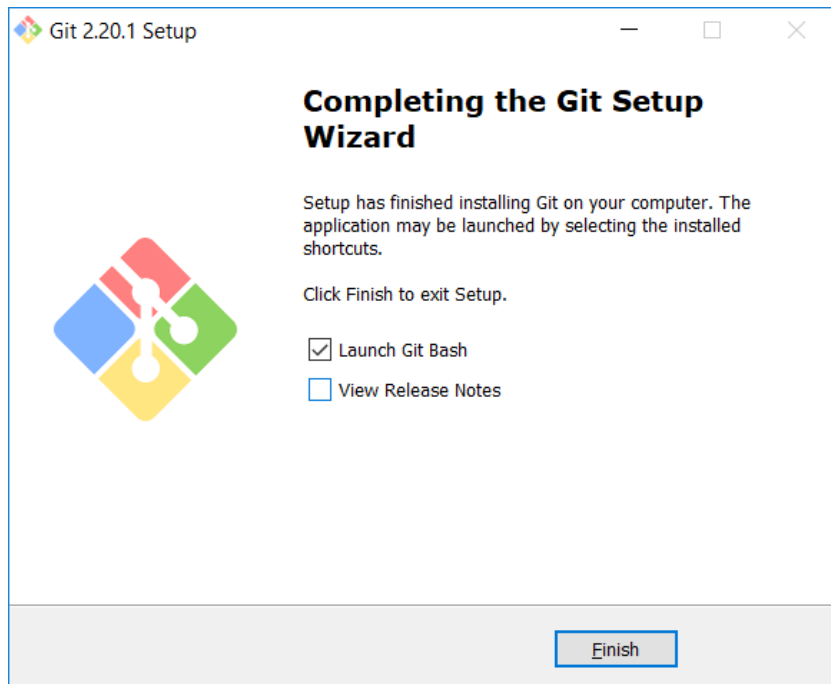
Opcje dodatkowe (domyślne):



Instalacja:



Kończymy instalację i uruchamiamy **Git Bash**:



3. Git Bash

3.1. Konfiguracja GITa

Konfiguracja GITa sprowadza się do ustawienia nazwy użytkownika, adresu e-mail oraz inicjalizacji repozytorium.

Do tego celu służą kolejno polecenia:

```
git config --global user.name "Twoje imię i nazwsko"
```

```
git config --global user.email twój@email.com
```

```
git init
```

Polecenia te należy wywołać będąc w katalogu, w którym chcemy utworzyć repozytorium.

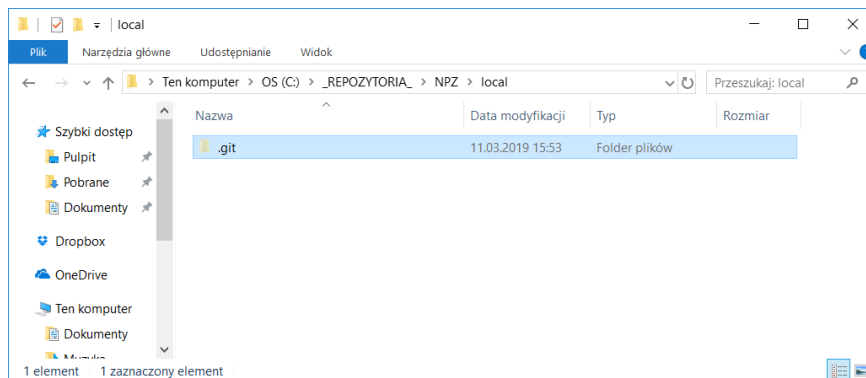
```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local
$ git config --global user.name Adam

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local
$ git config --global user.email adamszczur8@gmail.com

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local
$ git init
Initialized empty Git repository in C:/_REPOZYTORIA_/NPZ/local/.git/

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

W wyniku w wybranym folderze utworzony został ukryty katalog **.git** zawierający pliki konfiguracyjne GITa.



ZADANIE

Utwórz na pulpicie katalog o nazwie **moje_repozytorium** i zainicjalizuj go jako repozytorium GITa.

3.2. Rejestrowanie zmian w repozytorium

Rejestrowanie zmian w repozytorium podzielone jest na kilka etapów bezpośrednio związanych z cyklem życia zmian.

3.2.1. Sprawdzenie stanu plików w repozytorium

Do sprawdzenia stanu plików w repozytorium (statusu) służy polecenie:

```
git status
```

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Jak można zauważyć, w naszym repozytorium nie ma żadnych plików, które są śledzone.

ZADANIE

Utwórz w katalogu repozytorium nowy plik tekstowy **text.txt** i zapisz w nim tekst *Ala ma kota*. Następnie sprawdź status plików w repozytorium.

W wyniku wykonanych czynności otrzymujemy komunikat:

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    text.txt

nothing added to commit but untracked files present (use "git add" to track)

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

GIT wykrył, że dodaliśmy nowy plik i ostrzega nas, że nie jest on jeszcze śledzony przez repozytorium.

3.2.2. Śledzenie nowych plików

Aby rozpocząć śledzenie naszego pliku posłużymy się poleceniem

git add

Składnia polecenia:

- **git add text.txt** – dodaje jeden wybrany plik
- **git add -A** – dodaje wszystkie nieśledzone pliki
- **git add .** – dodaje do śledzenia bieżący katalog ze wszystkimi plikami i katalogami, które się w nim znajdują

```
MINGW64:/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git add text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Jak można zauważyć, status pliku **text.txt** zmienił się.

ZADANIE

Dodaj do śledzenia plik **text.txt** i następnie sprawdź status repozytorium.

3.2.3. Dodawanie zmian do poczekalni

Jak zostało wcześniej napisane, GIT operuje na pojedynczych zmianach, a nie na plikach. Bardzo dobrze to widać podczas dodawania plików do poczekalni.

ZADANIE

Zmodyfikuj plik **text.txt** dodając do niego kolejną linię tekstu: *Kot ma Alę*. Następnie sprawdź status plików w repozytorium.

Ponowne wywołanie `git status` pokazuje, że plik **text.txt** jest jednocześnie w dwóch sekcjach:

```
MINGW64:/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   text.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Dzieje się tak, ponieważ GIT umieszcza plik w poczekalni w dokładnie takiej wersji, w jakiej znajdował się podczas odpalenia komendy `git add`. Jeżeli w tym momencie zostanie uruchomiona komenda `git commit` to zatwierdzona zostanie wersja z poczekalni, a nie ta widoczna w katalogu roboczym.

Żeby zaktualizować plik w poczekalni, trzeba go zwyczajnie jeszcze raz dodać przez `git add`.

```
MINGW64~/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git add text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Wykorzystanie poczekalni w procesie zatwierdzania zmian daje ogromne możliwości, dzięki temu można wybrać, które konkretnie modyfikacje chce się zatwierdzić i utrwalić w repozytorium.

ZADANIE

Zaktualizować plik **text.txt** poprzez wywołanie komendy `git add`. Sprawdzić status plików.

3.2.4. Podgląd dokonanych zmian

Jest to bardzo ważna funkcjonalność. Przed zatwierdzeniem zmian zawsze warto zweryfikować, czy wszystko poszło zgodnie z planem. Może się zdarzyć, że zmiany zostały zrobione nie w tym miejscu, co trzeba lub pojawiły się jakieś dodatkowe wygenerowane pliki, których nie chcemy utrwać w repozytorium. Dzięki weryfikacji zmian w podglądzie można uniknąć tego typu błędów.

Polecenie `git status` dostarczy tylko informacji, które pliki zostały zmodyfikowane, natomiast dzięki `git diff` można dokładnie zobaczyć te zmiany.

- różnica między katalogiem roboczym a poczekalnią – `git diff`
- różnica między poczekalnią a repozytorium – `git diff --cached`

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   text.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git diff

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git diff --cached
diff --git a/text.txt b/text.txt
new file mode 100644
index 0000000..a3d9b0f
--- /dev/null
+++ b/text.txt
@@ -0,0 +1,2 @@
+Ala ma kota.
+Kot ma AlKEA.
\ No newline at end of file

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
```

ZADANIE

Sprawdzić działanie poleceń z punktu 3.2.4

3.2.5. Zatwierdzanie zmian

Jeżeli mamy już pewność, że dokonane zmiany są poprawne, można utrwalić je w lokalnym repozytorium. W tym celu posłużymy się komendą

```
git commit
```

Wywołanie komendy bez żadnych argumentów uruchomi najpierw domyślny edytor tekstowy w celu podania komentarza dla zatwierdzanych zmian.

Opcjonalnie można podać komentarz jako argument już przy samym wywołaniu komendy:

```
git commit -m „komentarz”
```

Po potwierdzeniu zmiany zostaną zapisane w repozytorium w postaci nowej migawki.

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git commit --m "komentarz do commita"
[master (root-commit) 925edbd] komentarz do commita
1 file changed, 2 insertions(+)
create mode 100644 text.txt

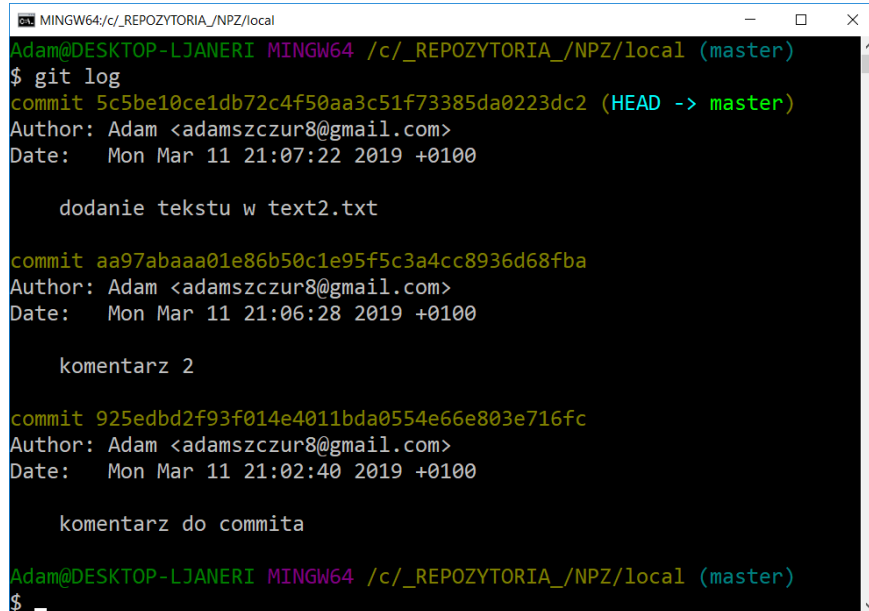
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

3.2.6. Historia zmian

Do przeglądania historii zmian służy polecenie:

`git log`

Domyślnie log bez podania żadnych argumentów wyświetla zmiany od najnowszego do najstarszego.



```
MINGW64:/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git log
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

commit 925edbd2f93f014e4011bda0554e66e803e716fc
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:02:40 2019 +0100

    komentarz do commita

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Polecenie log jest bardzo rozbudowane i zawiera wiele opcji konfiguracyjnych, ich pełną listę można znaleźć korzystając z pomocy:

`git help log`

Jedną z najprzydatniejszych opcji jest `-p`. Pokazuje ona różnice wprowadzone z każdą rewizją. Dodatkowo można użyć opcji `-2` aby ograniczyć zbiór do dwóch ostatnich wpisów:


```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git log -p -2
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

diff --git a/text2.txt b/text2.txt
index e69de29..c9c1808 100644
--- a/text2.txt
+++ b/text2.txt
@@ -0,0 +1 @@
+Narzedzia Pracy Zespo<B>owej
\ No newline at end of file

commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

diff --git a/text2.txt b/text2.txt
new file mode 100644
index 0000000..e69de29

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
```

Opcja spowodowała wyświetlenie tych samych informacji z tą różnicą, że bezpośrednio po każdym wpisie został pokazywany tzw. *diff*, czyli różnica. Jest to szczególnie przydatne podczas recenzowania kodu albo szybkiego przeglądania zmian dokonanych przez współpracowników. Dodatkowo można skorzystać z całej serii opcji podsumowujących wynik działania `git log`. Na przykład, aby zobaczyć skrócone statystyki każdej z zatwierdzonych zmian należy użyć opcji `--stat`:

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git log --stat
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

text2.txt | 1 +
1 file changed, 1 insertion(+)

commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

text2.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)

commit 925edbd2f93f014e4011bda0554e66e803e716fc
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:02:40 2019 +0100

    komentarz do commita

text.txt | 2 ++
1 file changed, 2 insertions(+)

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Przydatnymi opcjami są również `--since` oraz `--until`. Wprowadzają one ograniczenia czasu dla wyświetlanych logów, np. wyświetlenie informacji o zmianach dokonanych w ciągu ostatnich kilkunastu minut:

```
MINGW64/c/_REPOZYTORIA/_NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$ git log --since =17.minutes
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$ git log --since =18.minutes
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$
```

Polecenie to obsługuje mnóstwo formatów - można uściślić konkretną datę (np. "2008-01-15") lub podać datę względną jak np. 2 lata 1 dzień 3 minuty temu.

Można także odfiltrować listę pozostawiając jedynie rewizje spełniające odpowiednie kryteria wyszukiwania. Opcja `--author` pozwala wybierać po konkretnym autorze, a opcja `--grep` na wyszukiwanie po słowach kluczowych zawartych w notkach zmian.

```
MINGW64/c/_REPOZYTORIA/_NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$ git log --author=Adam
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

commit 925edbd2f93f014e4011bda0554e66e803e716fc
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:02:40 2019 +0100

    komentarz do commita

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$ git log --author=Ala

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA/_NPZ/local (master)
$
```

```
WybierzMINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git log --grep="komentarz"
commit aa97abaaa01e86b50c1e95f5c3a4cc8936d68fba
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:06:28 2019 +0100

    komentarz 2

commit 925edbd2f93f014e4011bda0554e66e803e716fc
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:02:40 2019 +0100

    komentarz do commita

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

Jeżeli potrzebujemy określić zarówno autora jak i słowa kluczowe, musimy dodać opcję `--all-match` - w przeciwnym razie polecenie dopasuje jedynie wg jednego z kryteriów.

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git log --author=Adam --grep="dodanie" --all-match
commit 5c5be10ce1db72c4f50aa3c51f73385da0223dc2 (HEAD -> master)
Author: Adam <adamszczur8@gmail.com>
Date: Mon Mar 11 21:07:22 2019 +0100

    dodanie tekstu w text2.txt

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$
```

ZADANIE

Utworzyć kolejny plik **text2.txt** i dodać go do repozytorium oraz zacommitować zmiany (komentarz dowolny). Następnie zmodyfikować plik **text2.txt** umieszczając w nim tekst: *Stoi na stacji lokomotywa*. Dodać zmiany (add). Dodać kolejną linie tekstu w pliku **tekst2.txt**: *Ciężka, ogromna i pot z niej spływa – tłusta oliwa*. Dodać zmiany i wykonać commit. Sprawdzić działanie polecenia `git log` z różnymi opcjami, o których mowa w tym podrozdziale.

3.3. Rozszerzone mechanizmy

3.3.1. Cofanie zmian

W celu wycofania zmian, które zostały już wysłane do zdalnego repozytorium, można skorzystać z commitów wycofujących. Ten mechanizm nie modyfikuje historii, a generuje commit, który jest przeciwieństwem zmiany, którą chcemy wycofać. Służy do tego polecenie:

`git revert [opcje]`

```
MINGW64/c/_REPOZYTORIA_/NPZ/local
Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
$ git revert HEAD~0
[master 125c02f] Revert "ciuch"
1 file changed, 1 insertion(+), 2 deletions(-)

Adam@DESKTOP-LJANERI MINGW64 /c/_REPOZYTORIA_/NPZ/local (master)
```

Powyższe wywołanie cofa ostatni commit w rewizji HEAD.

ZADANIE

Dodać do pliku **text2.txt** kolejną linię tekstu: *ciuch, ciuch*. Dodać zmiany i wykonać commit. Następnie dodać jeszcze jedną linię: *lokomotywa odjeżdża* i również dodać zmiany i wykonać commit. Dalej wycofać ostatnie 2 commity i sprawdzić zawartość pliku **text2.txt**.

Usunąć plik **text.txt** i zatwierdzić zmiany wykonując polecenia `add` i `commit`. Następnie wycofać ostatni commit i sprawdzić zawartość repozytorium. Sprawdzić historię zmian w repozytorium.

3.4. Tagowanie źródeł

Tagowanie (etykietowanie) źródeł to mechanizm pozwalający na oznaczenie ważniejszych miejsc w historii zmian projektu. Najczęściej jest wykorzystywany do oznaczania wersji aplikacji (np. wersja 3.1.5, itp). Git posiada dwa rodzaje etykiet: lekkie oraz opisane. Dla nas ważne będą lekkie.

W celu oznaczenia aktualnych źródeł nowym tagiem wpisujemy komendę:

```
git tag [nazwa-tagu]
```

Natomiast sama komenda `git tag`, bez podania żadnych argumentów, wyświetli listę wszystkich znanych tagów.

ZADANIE

Utworzyć etykietę dla bieżącej wersji repozytorium. Następnie dokonać kilku zmian w repozytorium wraz z commitami i otagować każdy commit. Wyświetlić listę tagów.

3.5. Ignorowanie plików

W większości projektów mamy do czynienia z plikami, których nie chcemy wersjonować. Są to np. pliki generowane automatycznie. Dodanie ich do repozytorium powoduje tylko zaciemnienie obrazu wprowadzanych zmian.

Można, co prawda, pomijać tego typu pliki przy zatwierdzaniu zmian, jednak nie jest to zbyt pragmatyczne podejście. Dużo lepszym wyjściem jest oznaczenie takiej klasy plików jako ignorowane. Od tego momentu nie będą nawet widoczne jako pliki zmodyfikowane.

Mechanizm ignorowania plików oparty jest o plik tekstowy **.gitignore**. Poniżej przykładowa zawartość:

```
*.tmp
```

```
tmp
```

Kolejne klasy ignorowanych plików wpisujemy w osobnych liniach. Pierwsza linijka odpowiedzialna jest za ignorowanie wszystkich plików o rozszerzeniu `.tmp`, natomiast druga za cały katalog `tmp` oraz jego zawartość.

Warto już na starcie zdefiniować, które pliki mają być ignorowane. Pozwoli to w przyszłości na uniknięcie zabawy z niepotrzebnymi plikami.

Ponieważ plik **.gitignore** jest zwykłym plikiem tekstowym przechowywanym w głównym katalogu repozytorium, on również może podlegać wersjonowaniu. Po jego dodaniu lub modyfikacji warto zacommitować naniesione zmiany lub jego też oznaczyć do ignorowania.

ZADANIE

Utworzyć plik **.gitignore** i umieścić w nim kilka rozszerzeń plików (np. `*.tmp`, `*.inf`) oraz katalog np. `tmp`. Wykonać commit. Następnie dodać do repozytorium po 1 pliku z każdym z ignorowanych

rozszerzeń oraz katalog podany do ignorowania. W katalogu tym powinno znajdować się kilka plików. Sprawdzić status repozytorium, tzn. czy są jakieś zmiany widziane przez GITa.