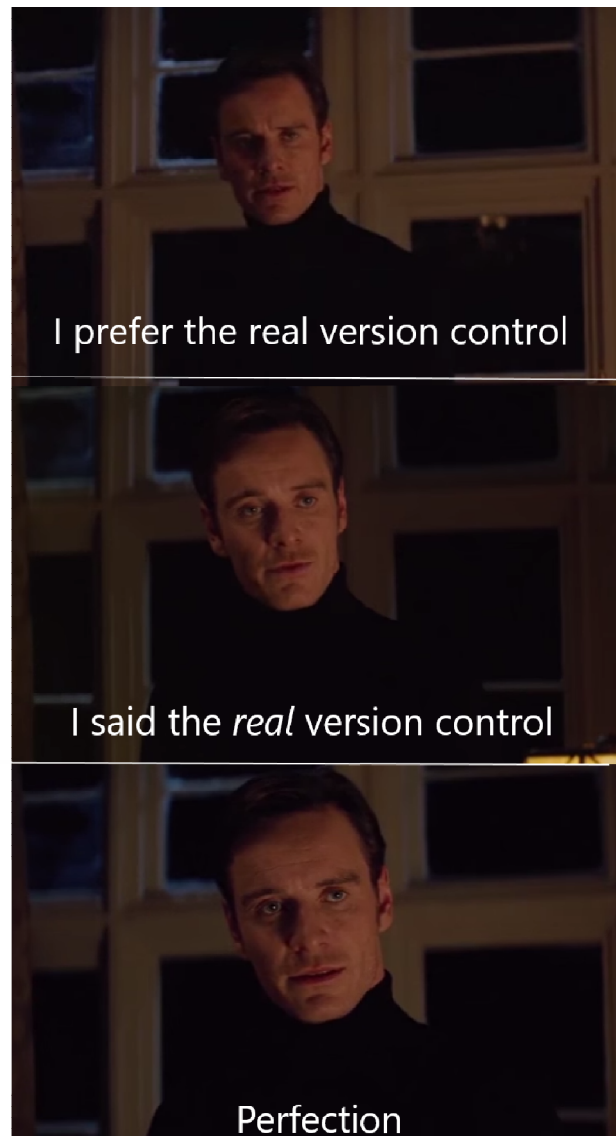
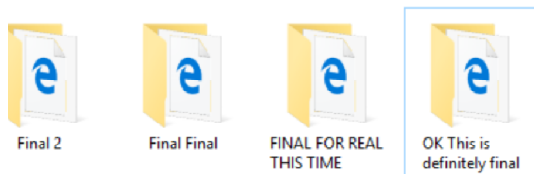


**Zawartość:**

- instalacja git
- inicjacja repozytorium git
- gitignore
- dodawanie zmian i ich zatwierdzanie
- historia commitów
- cofanie zmian
- gałęzie
- schowek
- zdalne repo

## GIT – system kontroli wersji



Każdy kiedyś spotkał się z problemem wersjonowania projektu. Każdy kiedyś naduszał CTRL+Z pięćdziesiąt razy, żeby wycofać się z wprowadzonych zmian. Każdy kiedyś cofając zmiany wspomnianym wcześniej skrótem cofnął nie tylko te zmiany, które chciał, ale też te których nie chciał. Każdemu zdarzyła się sytuacja kiedy chciał przetestować jakąś eksperymentalną zmianę przez co tworzył kopię projektu tylko po to by nie uszkodzić głównej wersji.

Wszystkie te problemy rozwiązuje system kontroli wersji GIT. Jako, że każdy prawidłny programista robi wszystko w konsoli, tutaj też przejdziemy przez niego za pomocą konsoli.

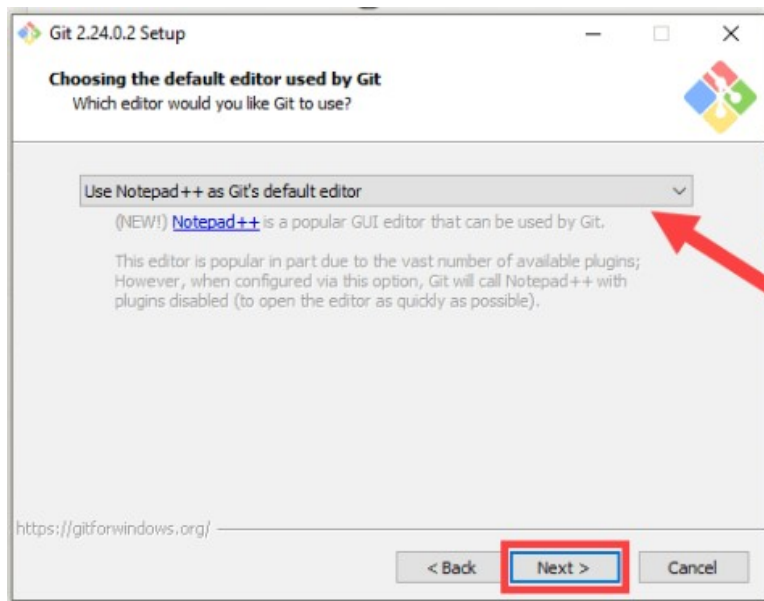
## Instalacja i konfiguracja:

Link do pobrania pliku instalacyjnego

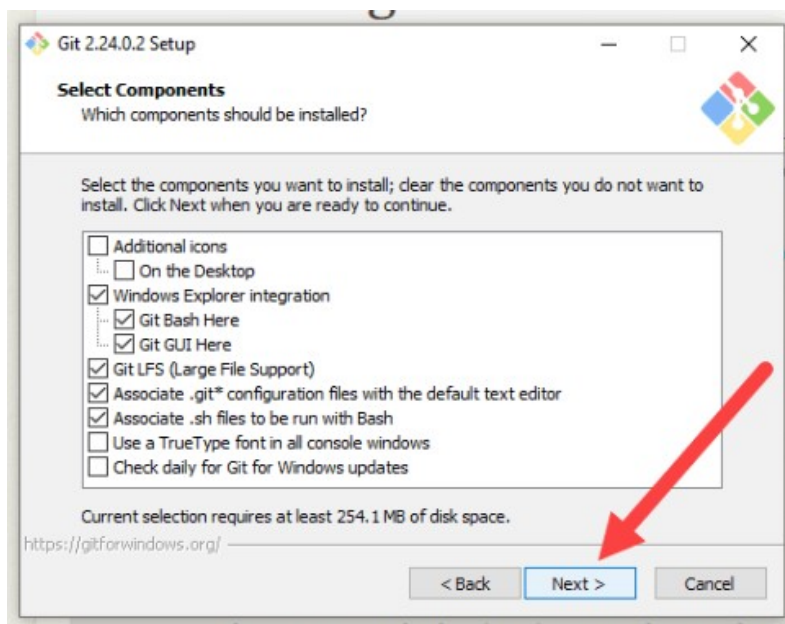
<https://git-scm.com/downloads>

Na co zwrócić uwagę w czasie instalacji?

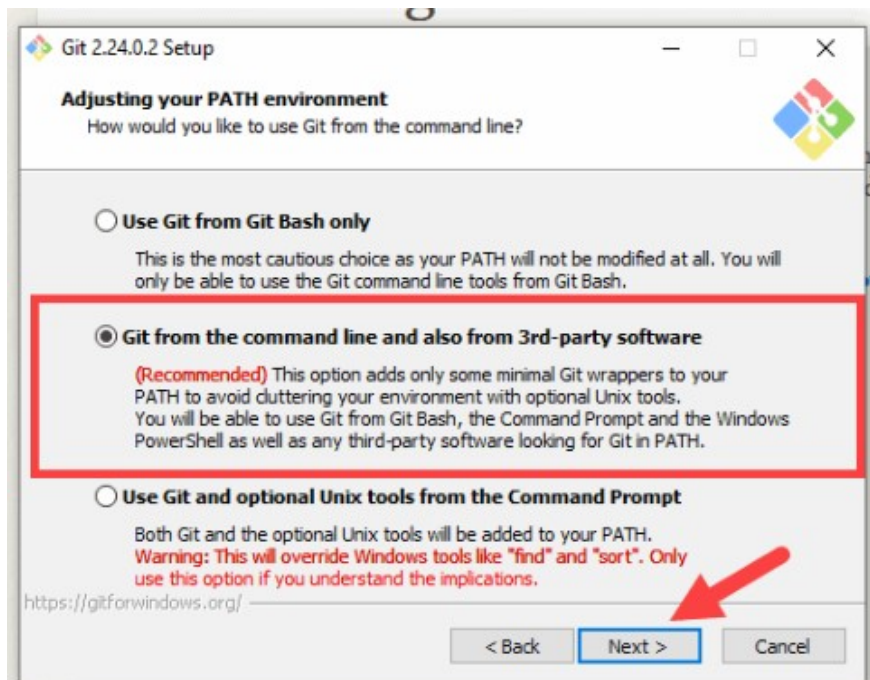
- W pewnym momencie GIT zapyta nas o domyślny dla niego edytor tekstu – warto wskazać tutaj jakiś Visual Studio Code lub coś innego, żeby nie utknąć kiedyś w vimie (kto wie ten wie)



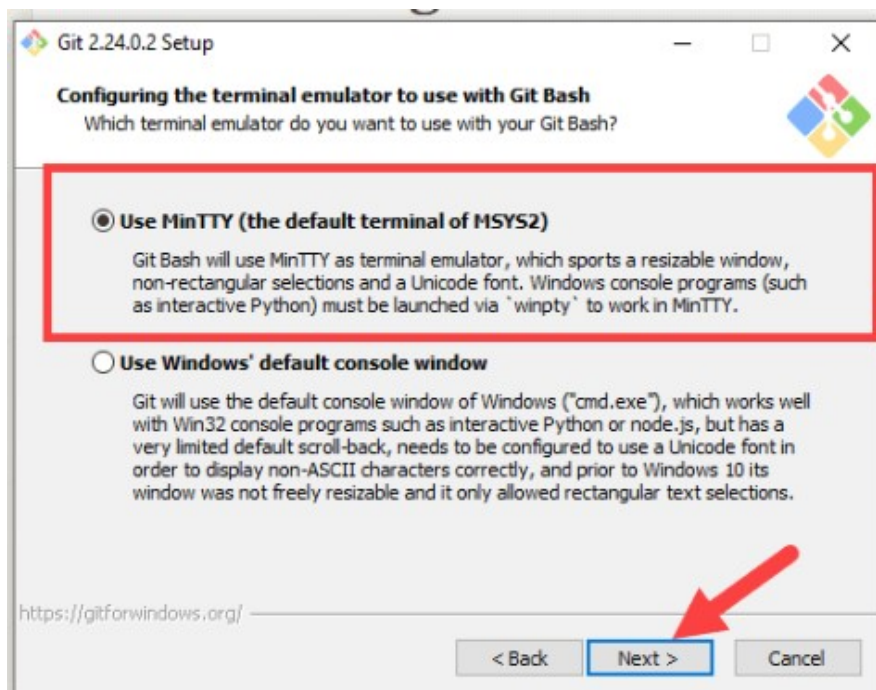
- Jako, że każdy prawdziwy programista pracuje w konsoli polecam odznaczyć instalację klienta GUI



- Warto Gita dodać do zmiennej środowiskowej PATH – co pozwoli używać go wszędzie, nie ważne w jakim katalogu się znajdujemy

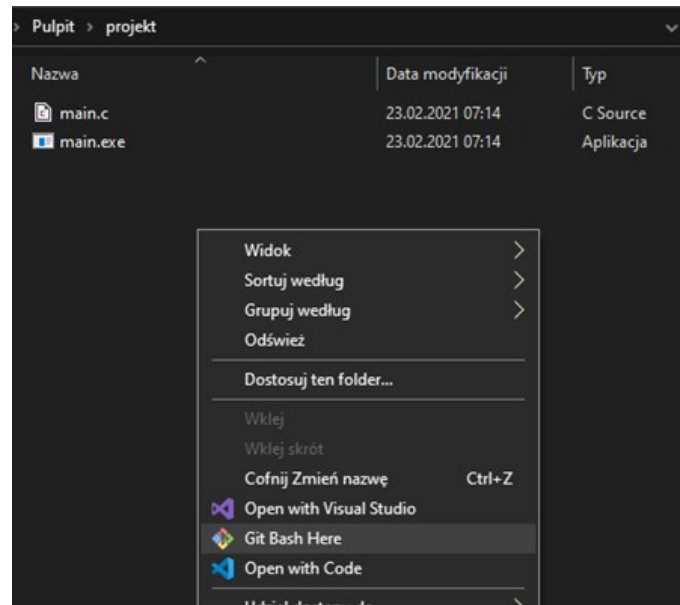


- Możemy jeszcze wybrać sobie rodzaj terminala (MSYS2 to domyślny terminal Gita, który w przeciwieństwie do normalnego wiersza poleceń pokazuje trochę więcej informacji)

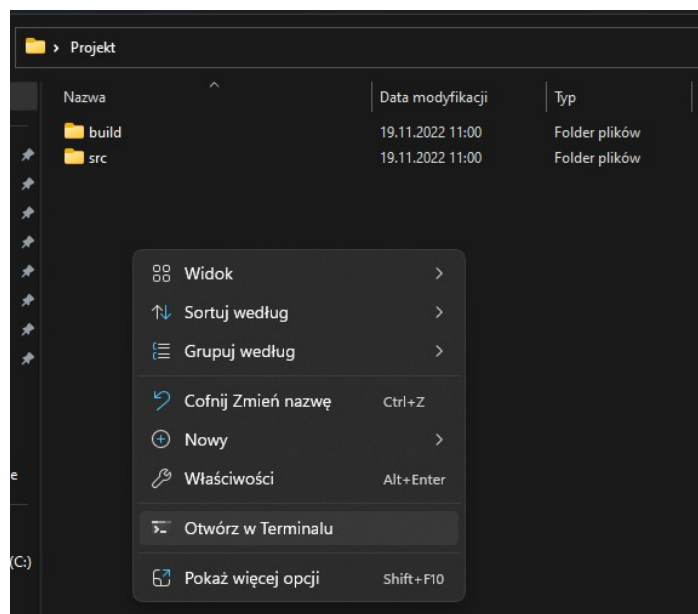


## Inicjowanie lokalnego repozytorium oraz plik .gitignore

Aby zainicjować repozytorium w naszym projekcie wystarczy kliknąć PPM na puste pole w eksploratorze Windows, a następnie wybrać „Git Bash Here”:



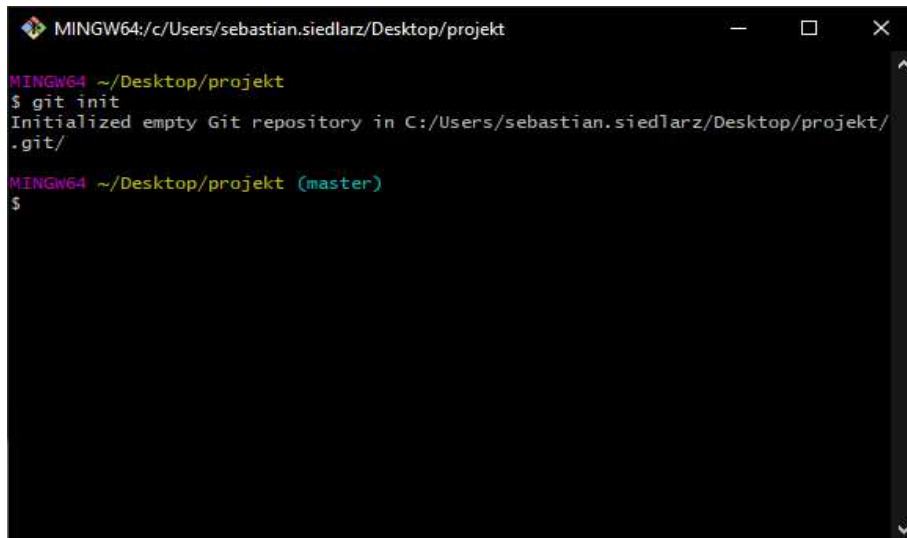
Jeżeli ktoś używa konsoli Windows (bo tak wybrał przy instalacji) po prostu otwiera terminal w tym miejscu:



Nie ma to znaczenia, ponieważ dodając do gita do zmiennej środowiskowej PATH umożliwiliśmy korzystanie z niego wszędzie tak jak z dowolnej innej komendy CMD.

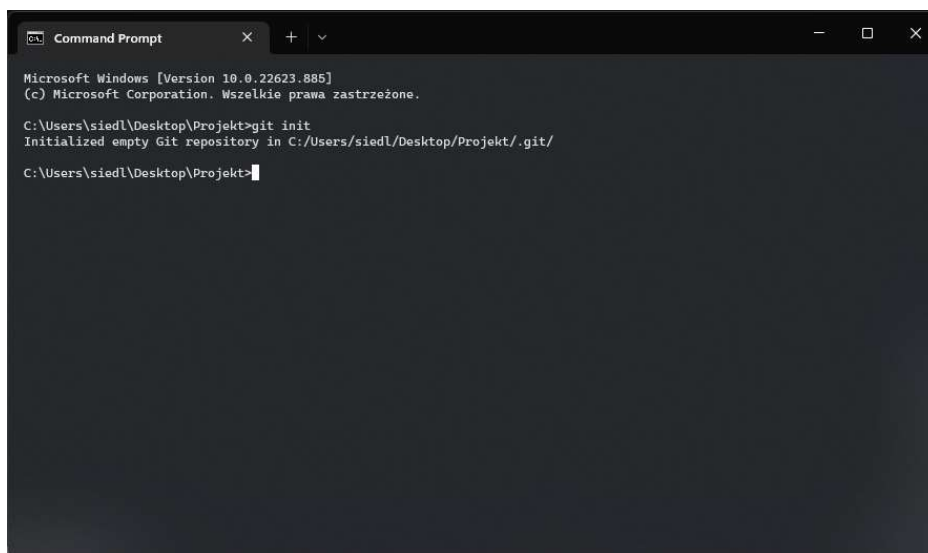
W dalszej części będziemy posługiwali się przykładową strukturą projektu składającą się z katalogu **src** (katalog z kodem) oraz **build** (katalog z wynikami np. kompilacji)

Tak wygląda Shell MSYS czyli domyślny wybór w czasie instalacji:



```
MINGW64/c/Users/sebastian.siedlarz/Desktop/projekt
MINGW64 ~/Desktop/projekt
$ git init
Initialized empty Git repository in C:/Users/sebastian.siedlarz/Desktop/projekt/.git/
MINGW64 ~/Desktop/projekt (master)
$
```

Jeżeli wybraliśmy wiersz poleceń Windows to zobaczymy zwykłą konsolę CMD. Możliwe jest jednak skonfigurowanie sobie aplikacji Windows Terminal by ta zastąpiła zwykły wiersz poleceń. Aplikacja ta dostępna jest do pobrania w sklepie Windows. Oferuje ona możliwość zmiany palety kolorów wiersza poleceń oraz widok kart.



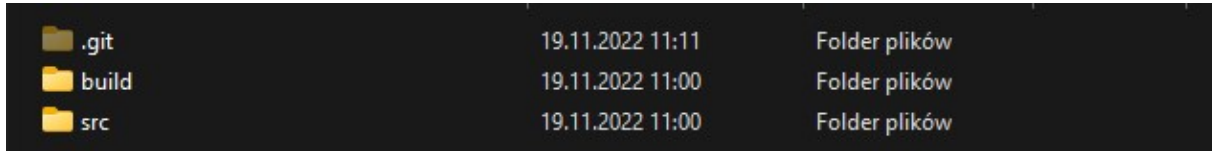
```
Command Prompt
Microsoft Windows [Version 10.0.22623.885]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\siedl\Desktop\Projekt>git init
Initialized empty Git repository in C:/Users/siedl/Desktop/Projekt/.git/

C:\Users\siedl\Desktop\Projekt>
```

Polecenie: **git init**

Inicjuje lokalne (ważne określenie) repozytorium. Jeżeli ustawimy sobie w Windowsie by ten pokazywał ukryte pliki i katalogi powinniśmy zauważyć w katalogu gdzie uruchomiliśmy konsolę katalog **.git**



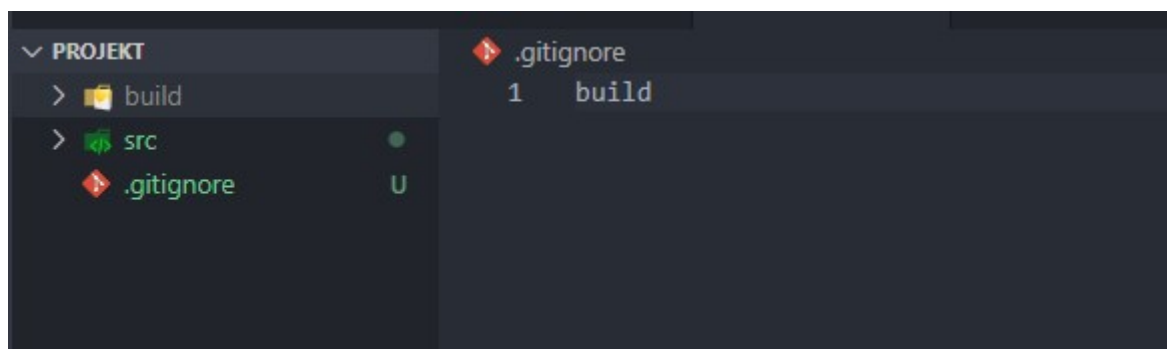
.git	19.11.2022 11:11	Folder plików
build	19.11.2022 11:00	Folder plików
src	19.11.2022 11:00	Folder plików

**Bardzo ważnym aspektem pracy z systemem GIT jest dodanie w katalogu gdzie otwarliśmy repozytorium pliku o nazwie:**

**.gitignore**

Utworzenie tego pliku może wymagać włączenia w windowsie by ten pokazywał rozszerzenia plików.

Plik ten pozwala stworzyć listę plików/katalogów, które mają być ignorowane przez system git. Raczej nie potrzebujemy śledzić plików kompilacji (one zmieniają się przy każdej kompilacji) lub pobranych bibliotek (te są np. pobierane automatycznie przez środowisko na bazie jakiegoś pliku konfiguracyjnego w projekcie)



Użyty w przykładzie projekt jest mega prosty, więc tutaj wykluczymy tylko katalog z wynikami kompilacji.

Jeżeli tworzymy projekt z użyciem jakiegoś bardziej zaawansowanego narzędzia (Visual Studio/NPM/Android Studio) to generuje on z automatu plik **.gitignore** pod siebie. Raczej ciężko pisać ręcznie gitignore dla Android Studio gdzie struktura projektu jest dosyć skomplikowana. Dodatkowo mało kto zna zastosowanie każdego pliku jaki tam można znaleźć.

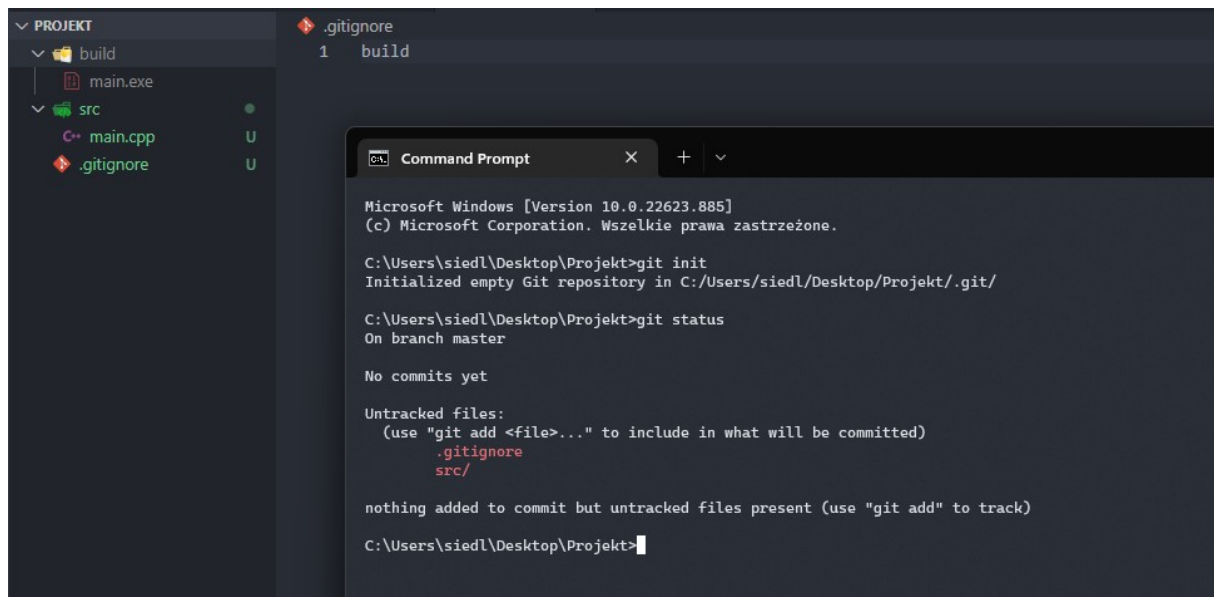
Jeżeli nasze narzędzie nie generuje plików gitignore automatycznie to możemy je pobrać np. z tego linku:

<https://github.com/github/gitignore>

### Polecenie: **git status**

Polecenie to pokazuje status lokalnego repozytorium. Głównie chodzi o zmiany, które zostały wykonane od ostatniego commita (o tym dalej).

Na obrazku widać, że dodano jeden plik z kodem źródłowym. Wykonano także kompilację do katalogu build. Komenda opisana wyżej wykonana w konsoli, pokazuje nam, że zaszły zmiany w dwóch miejscach (wymienione na czerwono). Jak widać zmiany w katalogu build są ignorowane tak jak sobie zażyczyliśmy.



```
Microsoft Windows [Version 10.0.22623.885]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\siedl\Desktop\Projekt>git init
Initialized empty Git repository in C:/Users/siedl/Desktop/Projekt/.git/

C:\Users\siedl\Desktop\Projekt>git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\siedl\Desktop\Projekt>
```

Polecenie statusu pokazuje zmiany widziane przez git od ostatniego commita. Zmiany podzielone są na dwie grupy:

- nieśledzone – czerwone
- śledzone – zielone

Jeżeli zmiana oznaczona jest na zielono oznacza to, że zostanie ona dodana do następnego commita (zatwierdzenia). O tym w dalszej części.

Polecenie `git status` jest bardzo ważne. Jesteśmy się w stanie dzięki niemu zorientować gdzie dokonaliśmy zmian i co zostanie dołączone do następnego commita.

### Poleceni: **git add**

Polecenie to dodaje pliki/katalogi do śledzenia. Pliki śledzone zostaną podłączone do następnego commita.



```

C:\Users\siedl\Desktop\Projekt>git add .gitignore

C:\Users\siedl\Desktop\Projekt>git status
On branch master

No commits yet

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

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

```

Na powyższym obrazku dodaliśmy do śledzenia plik gitignore. Natomiast zmiany w src zostawiliśmy na razie nie tknięte.

Moglibyśmy oczywiście je dodać poleceniem: **git add src**, ale dodamy je w następnym commicie.

Polecenie: **git add .**

Dodaje wszystko co jest nie śledzone do śledzenia.

Polecenie: **git commit**

Wykonuje ono zatwierdzenie śledzonych zmian. Zaleca stosować się je z przełącznikiem **-m**:

**git commit -m „jakaś krótka wiadomość co zostało zmienione”**

Jeżeli zapomnimy o przełączniku uruchomi się nam wskazany przy instalacji edytor tekstu z prośbą o wprowadzenie wiadomości/komentarza do commita.

```

C:\Users\siedl\Desktop\Projekt>git add .gitignore 1

C:\Users\siedl\Desktop\Projekt>git status 2
On branch master

No commits yet

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

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

C:\Users\siedl\Desktop\Projekt>git commit -m "Add gitignore" 3
[master (root-commit) 0305e9b] Add gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore

C:\Users\siedl\Desktop\Projekt>git status 4
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/

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

```

Proces zatwierdzania jakiejś zmiany:

1. Dodajemy wszystkie pliki, które z logicznego punktu widzenia należą do konkretnej zmiany
2. Za pomocą statusu podglądamy czy wszystkie pliki, które chcieliśmy są śledzone
3. Tworzymy commita (zatwierdzamy zmianę)
4. Sprawdzamy jakie zmiany są jeszcze nie zatwierdzone

Teraz możemy zatwierdzić dodanie pierwszego pliku źródłowego w src

```
C:\Users\siedl\Desktop\Projekt>git add src

C:\Users\siedl\Desktop\Projekt>git commit -m "Add main.cpp"
[master e1fd171] Add main.cpp
1 file changed, 5 insertions(+)
create mode 100644 src/main.cpp

C:\Users\siedl\Desktop\Projekt>git status
On branch master
nothing to commit, working tree clean
```

Po zatwierdzeniu widzimy, że status pokazuje nam iż nie zostało nic do zatwierdzenia.

**Ważne: Lepiej robić dużo małych/drobnych commitów, niż mało, ale za to dużego rozmiaru. Dlaczego? Wyjdzie niżej.**

Polecenie: **git log**

Polecenie to wyświetla listę wykonanych commitów w kolejności takiej, że najświeższe zmiany są na górze.

```
C:\Users\siedl\Desktop\Projekt>git log
commit e1fd1712dfa5b8383abadcd7799b9ca67fc75cc1 (HEAD -> master)
Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
Date: Sat Nov 19 12:37:07 2022 +0100

    Add main.cpp

commit 0305e9bf0d5f413e438937f13b7f59608ab19861
Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
Date: Sat Nov 19 12:33:31 2022 +0100

    Add gitignore
```

**Bardzo ważny jest wskaźnik HEAD. Pokazuje on jaka wersja projektu jest aktualnie widoczna w eksploratorze plików/naszym środowisku.**

Fajny przełącznik do polecenia log to **—online**

```
C:\Users\siedl\Desktop\Projekt>git log --online
e1fd171 (HEAD -> master) Add main.cpp
0305e9b Add gitignore
```

Wyświetla on listę w trochę bardziej skompresowanej formie.

Polecenie: **git diff**

Polecenie pokazuje zmiany jakie zaszły w konkretnym wskazanym pliku od ostatniego commita:

```
C:\Users\siedl\Desktop\Projekt>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
```

Ponownie zmieniliśmy coś w pliku main.cpp. Przypomnę, że żeby zatwierdzić tę zmianę należy dodać plik do śledzenia i wykonać commit.

Polecenie git diff pozwoli nam podejrzeć zmiany (nie ważne czy plik jest śledzony czy nie)

```
C:\Users\siedl\Desktop\Projekt>git diff src/main.cpp
diff --git a/src/main.cpp b/src/main.cpp
index d534429..9102bf4 100644
--- a/src/main.cpp
+++ b/src/main.cpp
@@ -1,5 +1,7 @@
#include <iostream>

int main(){
-   std::cout<<"Hello World!"<<std::endl;
+   std::cout<<"Hello World!!!"<<std::endl;
+
+   return 0;
+
+ }
\ No newline at end of file
```

Jak widać git ładnie wskazuje tutaj co zostało dodane a co zostało usunięte. Podobny widok oferuje portal GITHUB, ale o nim później.

Jeżeli nie wskażemy w poleceniu pliku, który chcemy podejrzeć git wyświetli zmiany, we wszystkich zmodyfikowanych plikach. Polecam sprawdzić co się stanie jeżeli nie mamy gitignore i wyświetlimy zmiany ogólnie.

Teraz bardzo ważne pytanie. Po co to wszystko? Commity możemy cofać/odwracać. I dlatego należy wykonywać małe i drobne commity.

Wyobraźmy sobie sytuację, że zmieniliśmy styl na stronie, dodaliśmy parę popupów oraz zmodyfikowaliśmy jedno zapytanie do bazy danych. Następnie zapisaliśmy to wszystko w jednym commicie.

Chcemy odwrócić teraz zmiany w stylach, ale nie możemy bo odwróci to wszystko. Sytuacja albo wszystko albo nic. Dlatego róbmy małe commity zawsze!!!

### Odwracać zmiany można na trzy sposoby:

Polecenie git checkout

Polecenie to właściwie nie cofa zmian. Polecenie to przenosi wskaźnik HEAD na konkretnego commita w historii (wersja projektu z tego commita widoczna jest w naszym środowisku).

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
4a30752 (HEAD -> master) Use printf instead of std::cout
6fedd2a Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore
```

W tym momencie załadowana jest wersja projektu z commita 4a30752.

```
C:\Users\siedl\Desktop\Projekt>git checkout 6fedd2a
Note: switching to '6fedd2a'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 6fedd2a Improve Hello World

C:\Users\siedl\Desktop\Projekt>git log --oneline
6fedd2a (HEAD) Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore
```

Podając numer commita do git checkout przenieśliśmy się do wersji projektu z danego commita.

Powrót na szczyt odbywa się poleceniem: **git checkout master**

Oczywiście jeżeli przeniesiemy się w dół i dokonamy zmian nie będziemy mogli tego tak po prostu zapisać. Do tego tematu wrócimy w dalszej części.

Polecenie: **git reset**

Cofa do wskazanego commita, czyli jeżeli coś zmieniliśmy, ale tego nie zatwierdziliśmy polecenie to wycofa te zmiany:

--hard ten przełącznik sprawi, że zmiany zostaną całkowicie usunięte

--soft zmiany zostaną w repozytorium jako zmiany śledzone

--mixed zmiany zostaną w repozytorium jako zmiany nie śledzone

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
4a30752 (HEAD -> master) Use printf instead of std::cout
6fedd2a Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore

C:\Users\siedl\Desktop\Projekt>git status
On branch master
nothing to commit, working tree clean
```

Mając taką sytuację wykonuje jakieś zmiany w projekcie, ale ich nie zatwierdzam.

Polecenie: **git reset --hard** powoduje całkowite utracenie zmian i powrót do sytuacji z obrazka

Polecenie: **git reset --soft** powoduje, że zmiany zostaną w repozytorium jako zmiany śledzone

Polecenie: **git reset --mixed** najczęściej stosowane jest gdy wykonamy **git add** na plikach, które nie chcemy, ponieważ wycofa ono zmiany do statusu czerwonego czyli nie śledzonego.

Przykład:

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
4a30752 (HEAD -> master) Use printf instead of std::cout
6fedd2a Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore

C:\Users\siedl\Desktop\Projekt>git reset --mixed e1fd171
Unstaged changes after reset:
M      src/main.cpp

C:\Users\siedl\Desktop\Projekt>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main.cpp

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\siedl\Desktop\Projekt>git log --oneline
e1fd171 (HEAD -> master) Add main.cpp
0305e9b Add gitignore
```

Wykonano cofnięcie do commita e1fd171. Wszystkie zmiany, które miały miejsce później są w katalogu jako nie śledzone. Spostrzegawczy zauważą, że pozwala to wykonać commita jeszcze raz pod inną nazwą albo zagregować wiele commitów w jeden.

Polecenie: **git revert**

Polecenie to odwraca wskazany commit i zapisuje tę zmianę jako nowy commit.

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
4a30752 (HEAD -> master) Use printf instead of std::cout
6fedd2a Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore

C:\Users\siedl\Desktop\Projekt>git revert 4a30752
hint: Waiting for your editor to close the file...
[15092:1119/132404.151:ERROR:service_worker_storage.cc(1904)] Failed to delete the database: Database IO error
[master ec57eb2] Revert "Use printf instead of std::cout"
 1 file changed, 2 insertions(+), 2 deletions(-)

C:\Users\siedl\Desktop\Projekt>git log --oneline
ec57eb2 (HEAD -> master) Revert "Use printf instead of std::cout"
4a30752 Use printf instead of std::cout
6fedd2a Improve Hello World
e1fd171 Add main.cpp
0305e9b Add gitignore
```

Commit 4a30752 zostało odwrócone. Wymagało to zmian w projekcie. Zmiany te zostały zapisane w nowym commicie.

Podsumowując:

- **git checkout** służy do skakania po wersjach projektu (skakania po commitach)
- **git reset** służy do cofania zmian od wskazanego commita do teraz
- **git revert** służy do odwracania wskazanego commita

**git reset** jest jedyną komendą, która nie powinna być już używana jeżeli wysłaliśmy zmiany np. do githuba

## Gałęzie

Czasami zachodzi potrzeba stworzyć drugą kopię projektu w celu np. przetestowania eksperymentalnych zmian. Czasami chcemy mieć więcej kopii projektu. Jedną „na elegancko”, a drugą do pracy.

Rozwiązaniem tego problemu są gałęzie (branches).

Każde repozytorium posiada z góry jedną gałąź, która jest gałęzią główną (nazywa się ona master lub main).

```
C:\Users\siedl\Desktop\Projekt>git branch
* master
```

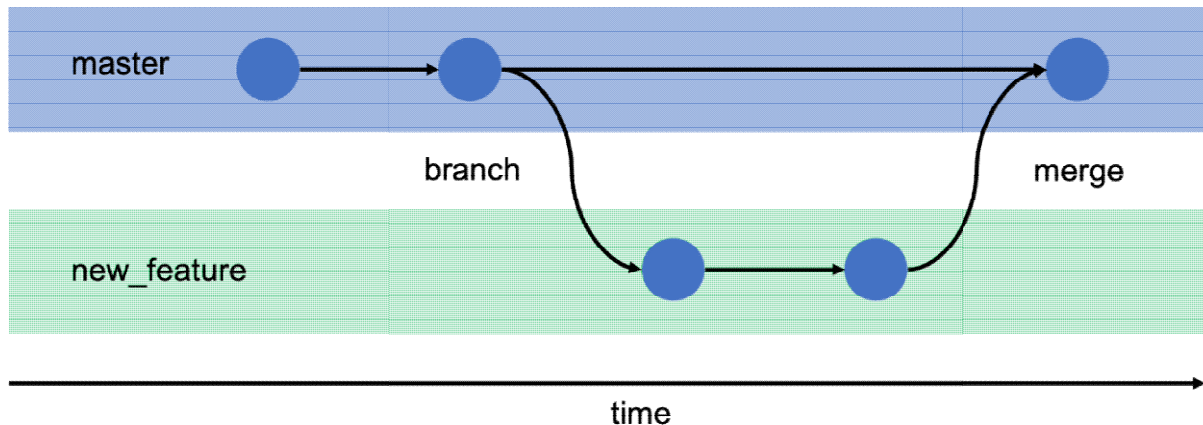


Polecenie: **git branch**

Wyświetla wszystkie gałęzie.

Polecenie: **git branch nazwa**

Tworzy branch o podanej nazwie na bazie wersji, na którą wskazuje HEAD gałęzi, na której się znajdujemy.



Aby możliwe było wykonanie nowego brancha/przełączenie między nimi gałąź, na której się znajdujemy musi mieć wszystko zatwierdzone (status musi pokazywać nothing to commit).

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
e1fd171 (HEAD -> master) Add main.cpp
0305e9b Add gitignore

C:\Users\siedl\Desktop\Projekt>git branch development

C:\Users\siedl\Desktop\Projekt>git checkout development
Switched to branch 'development'
M      src/main.cpp

C:\Users\siedl\Desktop\Projekt>git branch
* development
  master
```

Gałąź „development” została utworzona na bazie commita e1fd171.

Polecenie: **git branch -d nazwa**

Polecenie usuwa branch. Nie możemy znajdować się na gałęzi, którą chcemy usunąć.

```
C:\Users\siedl\Desktop\Projekt>git branch -d development
Deleted branch development (was e1fd171).
```

Gałęzie można tworzyć na bazie commitów, które są dalej w historii. Do tego przyda się polecenie **git checkout**, które w poprzednich przykładach użyto do skakania po commitach.

```
C:\Users\siedl\Desktop\Projekt>git log --oneline
e1fd171 (HEAD -> master) Add main.cpp
0305e9b Add gitignore

C:\Users\siedl\Desktop\Projekt>git checkout 0305e9b -b development
Switched to a new branch 'development'

C:\Users\siedl\Desktop\Projekt>git branch
* development
  master
```

Za pomocą polecenia **git checkout 0305e9b** przeskoczyliśmy do wskazanego commitu. Przełącznik **-b nazwa** pozwolił na bazie tego commitu, utworzyć nowy branch.

### Łączenie gałęzi:

Jak widać na powyższym schemacie, gałęzie można także łączyć. Bardzo częstą praktyką (nawet zalecaną praktyką) jest by na gałęzi głównej trzymać zawsze elegancką i funkcjonalną wersję projektu a ewentualne prace rozwojowe przeprowadzać na gałęziach pobocznych. W takiej sytuacji, gałęzie kiedyś trzeba będzie połączyć służy do tego

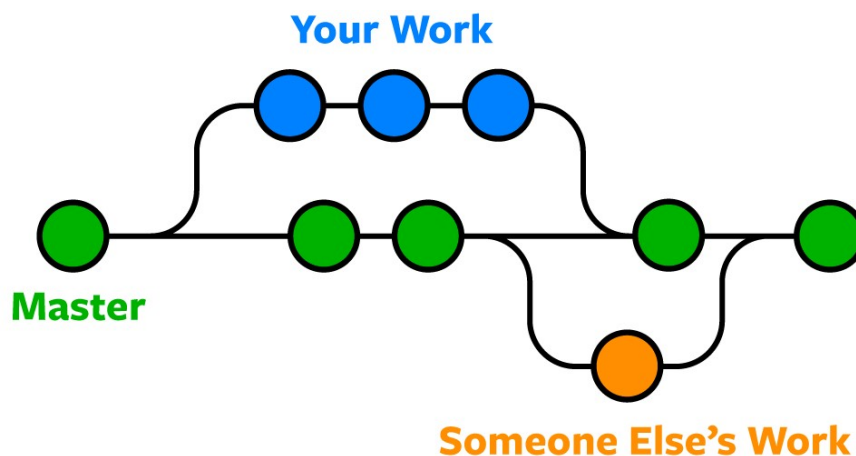
Polecenie: **git merge nazwa**

Aby dołączyć gałąź development do gałęzi master musimy przenieść się na gałąź master i wykonać polecenie:

**git merge development**

Teraz można usunąć gałąź development.

**Uwaga:** Proces ten przebiegnie bezproblemowo, jeżeli gałąź docelowa nie była modyfikowana między czasie. Jeżeli na obu gałęziach wykonano jakieś commity może dojść do konfliktu czyli sytuacji, w której git nie wie, którą wersję danego fragmentu kodu wybrać.





Łącząc gałąź zieloną i niebieską może dojść do konfliktu bo tutaj w przeciwieństwie do poprzedniego schematu zmiany zachodziły na obu gałęziach.

Przykład:

```
C:\Users\siedl\Desktop\Projekt>git log --graph --branches=*
* commit 03efa9a853b52d25c1045c7c52d0b77819c8cbbd (master)
  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
  Date: Sat Nov 19 14:42:14 2022 +0100

    Use \n instead of std::endl

* commit 956b44aeb3510896f4bd35c5975784e47ad642a7 (HEAD -> development)
  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
  Date: Sat Nov 19 14:41:39 2022 +0100

    Use printf

* commit e1fd1712dfa5b8383abadcd7799b9ca67fc75cc1
  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
  Date: Sat Nov 19 12:37:07 2022 +0100

    Add main.cpp

* commit 0305e9bf0d5f413e438937f13b7f59608ab19861
  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
  Date: Sat Nov 19 12:33:31 2022 +0100

    Add gitignore
```

Projekt został rozbity na dwie gałęzie. Na każdej po podziale wykonano jeden commit dotyczący tego samego kawałka kodu.

Próba mergu:

```
C:\Users\siedl\Desktop\Projekt>git merge development
Auto-merging src/main.cpp
CONFLICT (content): Merge conflict in src/main.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

W tym momencie uruchamia się nam domyślny edytor tekstu wskazany przy instalacji. Otwarte są w nim wszystkie pliki z konfliktami. Visual Studio Code ma do tego fajne funkcje, które przeprowadzają przez proces rozwiązywania konfliktów.

```
3  int main(){
4  <<<<<< HEAD (Current Change)
5      std::cout<<"Hello World!"<<"\n";
6  =====
7      printf("Hello World!!!");
8  >>>> development (Incoming Change)
9  }
```

Visual Studio Code pozwala wybrać, którą wersję kodu chcemy. Po wybraniu i zapisaniu wszystkich plików należy wykonać commita. Commit ten będzie punktem łączącym oba branche.

Jeżeli w przypadku merge nie ma konfliktów to połączenie zostanie wykonane bez tworzenia nowego commita.

```

*   commit c04554f0b88d9453dee522130dcdd4ab609eae47 (HEAD -> master)
| \
|  Merge: 03efa9a 956b44a
|  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
|  Date:   Sat Nov 19 14:54:01 2022 +0100
|
|      Merge
|
| *   commit 956b44aeb3510896f4bd35c5975784e47ad642a7 (development)
| | \
| |  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
| |  Date:   Sat Nov 19 14:41:39 2022 +0100
| |
| |      Use printf
| |
| *   commit 03efa9a853b52d25c1045c7c52d0b77819c8cbdd
| / \
|  Author: Sebastian Siedlarz <siedlarzsebastian409@gmail.com>
|  Date:   Sat Nov 19 14:42:14 2022 +0100
|
|      Use \n instead of std::endl

```

Na obrazku widać połączenie.

## Schowek

Schowek to miejsce gdzie można tymczasowo odłożyć zmiany wprowadzone od ostatniego commita

- dzięki temu możemy przetestować inną wersję tych zmian a później cofnąć poprzednie

- czasami chcemy się przełączyć na inną gałąź, ale mamy niezatwierdzone zmiany na aktualnej – schowek pozwala je odłożyć na czas wizyty na innej gałęzi by po powrocie je przywrócić

Polecenie: **git stash**

Umieszcza w schowku aktualnie nie zatwierdzone zmiany

Polecenie: **git stash pop**

Przywraca ze schowka zmiany, które zostały ostatnio odłożone

Schowek działa jak stos. Oznacza to, że zmiany możemy ciągle odkładać bo umieszczone zostaną one jedno na drugich (zasada działania jak stos FILO) oraz, że ściąganie odbywa się w odwrotnej kolejności względem umieszczania.

Polecenie: **git stash save „komentarz”**

Zachowuje zmiany w schowku z podanym komentarzem

Polecenie: **git stash list**

Wyświetla listę wpisów w schowku

Polecenie: **git stash pop stash@{x}**

Wykonuje pop elementu o numerze x z listy

## Zdalne repozytoria

Zdalne repozytoria pełnią rolę magazynu na nasz kod, ale też umożliwiają pracę w grupie na zasadach pracy zdalnej.

Przykładem serwisu, który umożliwia takie rzeczy jest [github.com](https://github.com). Zakładając repozytorium w serwisie przeprowadzi nas on przez proces łączenia repozytorium lokalnego i globalnego.

Polecenie: **git remote add origin https://github.com/user/repo.git**

Polecenie to dodaje połączenie między repozytoriami. Link ma zazwyczaj taką strukturę. Gdzie user to nazwa użytkownika a repo to nazwa repozytorium w serwisie.

Polecenie: **git remote remove origin**

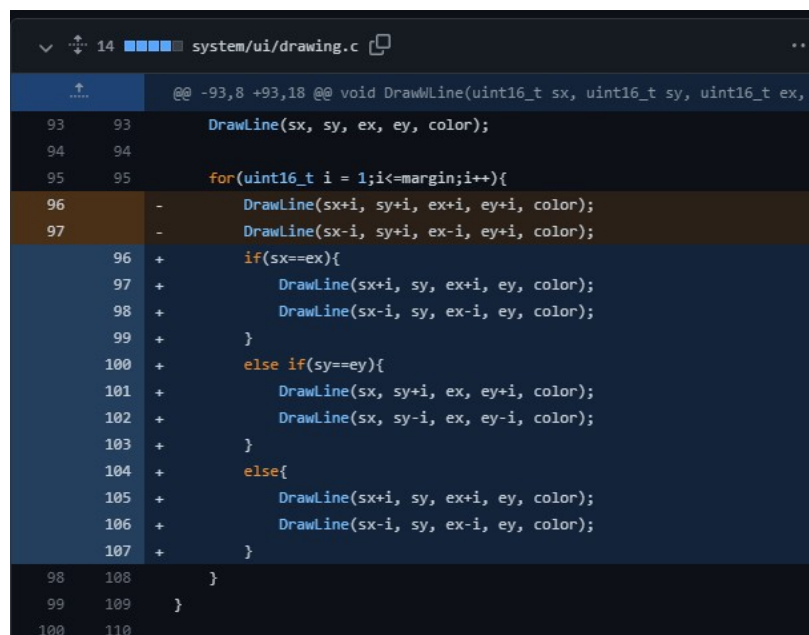
Usuwa wcześniej dodany link

Polecenie: **git push origin x**

Wysła brach x do zdalnego repozytorium

Polecenie: **git push --all origin**

Wysła wszystkie gałęzie do zdalnego repozytorium



```
@@ -93,8 +93,18 @@ void DrawWLine(uint16_t sx, uint16_t sy, uint16_t ex, uint16_t ey, uint16_t color);
93 93 DrawLine(sx, sy, ex, ey, color);
94 94
95 95 for(uint16_t i = 1; i <= margin; i++){
96 - DrawLine(sx+i, sy+i, ex+i, ey+i, color);
97 - DrawLine(sx-i, sy+i, ex-i, ey+i, color);
98 + if(sx==ex){
99 + DrawLine(sx+i, sy, ex+i, ey, color);
100 + DrawLine(sx-i, sy, ex-i, ey, color);
101 + }
102 + else if(sy==ey){
103 + DrawLine(sx, sy+i, ex, ey+i, color);
104 + DrawLine(sx, sy-i, ex, ey-i, color);
105 + }
106 + else{
107 + DrawLine(sx+i, sy, ex+i, ey, color);
108 + DrawLine(sx-i, sy, ex-i, ey, color);
109 + }
110 }
```

Na githubie można przeglądać zmiany, gałęzie, cofać się w projekcie, przeglądać statystyki i wiele więcej.