



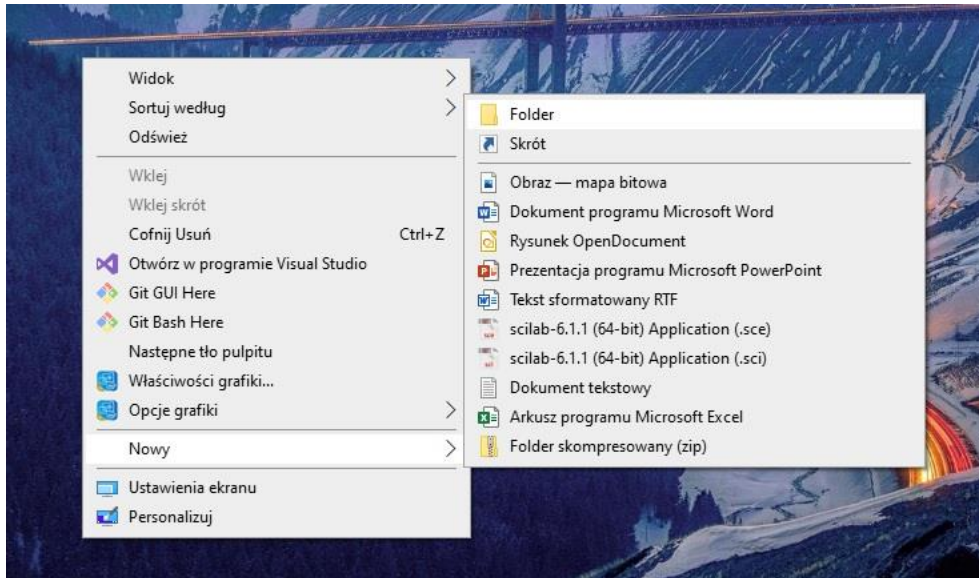
# **WSTĘP DO VCS GIT**

## **WARSZTATY**

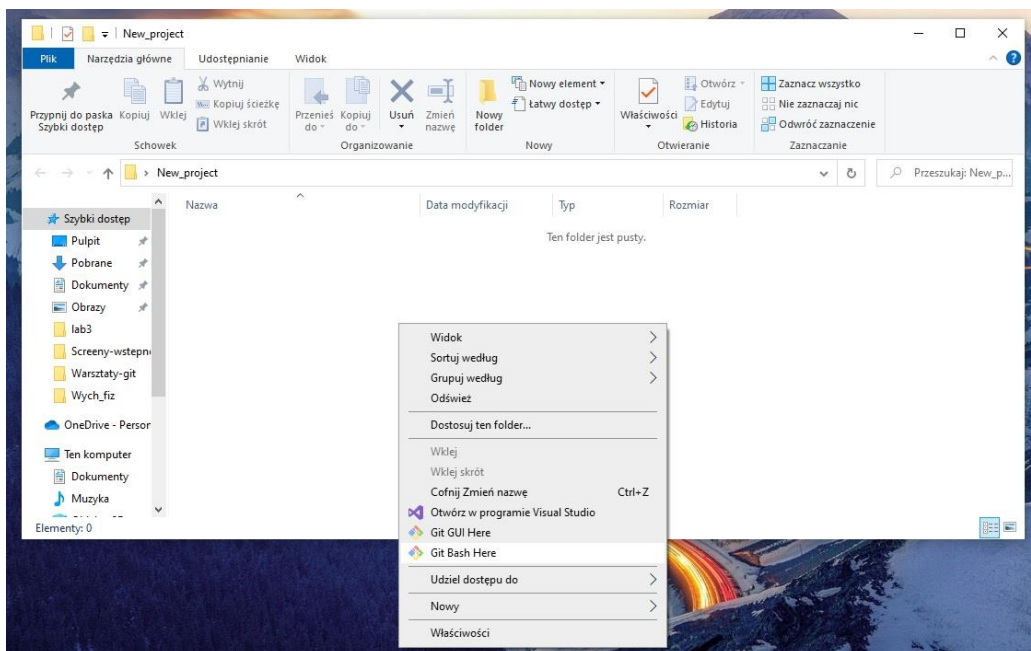
**Opracował:**  
**Hubert Baran**  
**Inżynieria i Analiza Danych**

# I. Podstawy

1. Utwórz na pulpicie folder o nazwie *New\_project*.



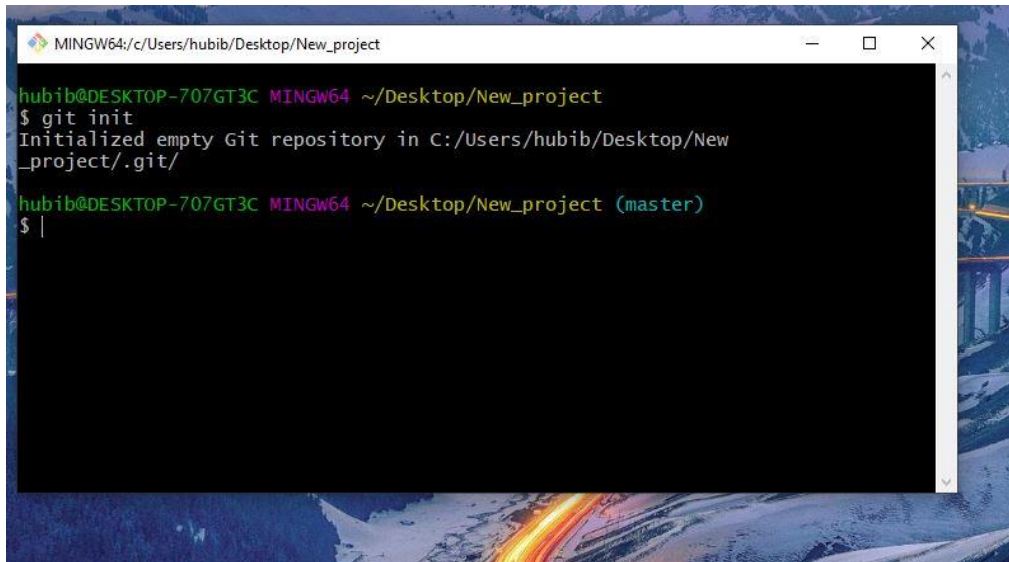
2. Otwórz folder, kliknij w okienku prawym przyciskiem myszy i wybierz *Git bash here*.



3. Otworzyliśmy w folderze *New\_project* Git Bash'a.

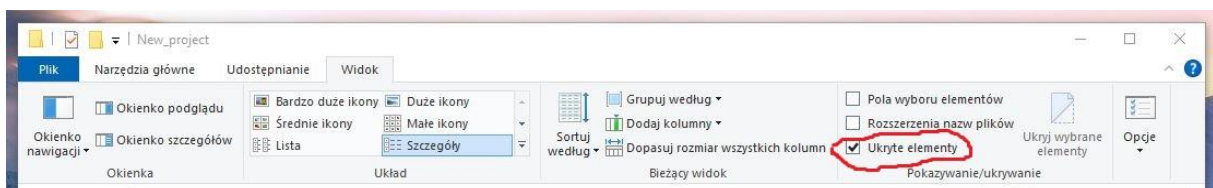
Zainicjalizuj w folderze repozytorium Gita (tzn. załóż projekt Gita). Wpisz w konsoli polecenie:

```
git init
```



```
MINGW64; c:/Users/hubib/Desktop/New_project
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project
$ git init
Initialized empty Git repository in C:/Users/hubib/Desktop/New_project/.git/
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ |
```

4. W okienku folderu, w zakładce widok zaznacz opcję *Ukryte elementy*.. Następnie sprawdź, co znajduje się w folderze. Co to takiego?



5. Możesz odznaczyć *Ukryte elementy*.

W konsoli wpisz:

```
git status
```

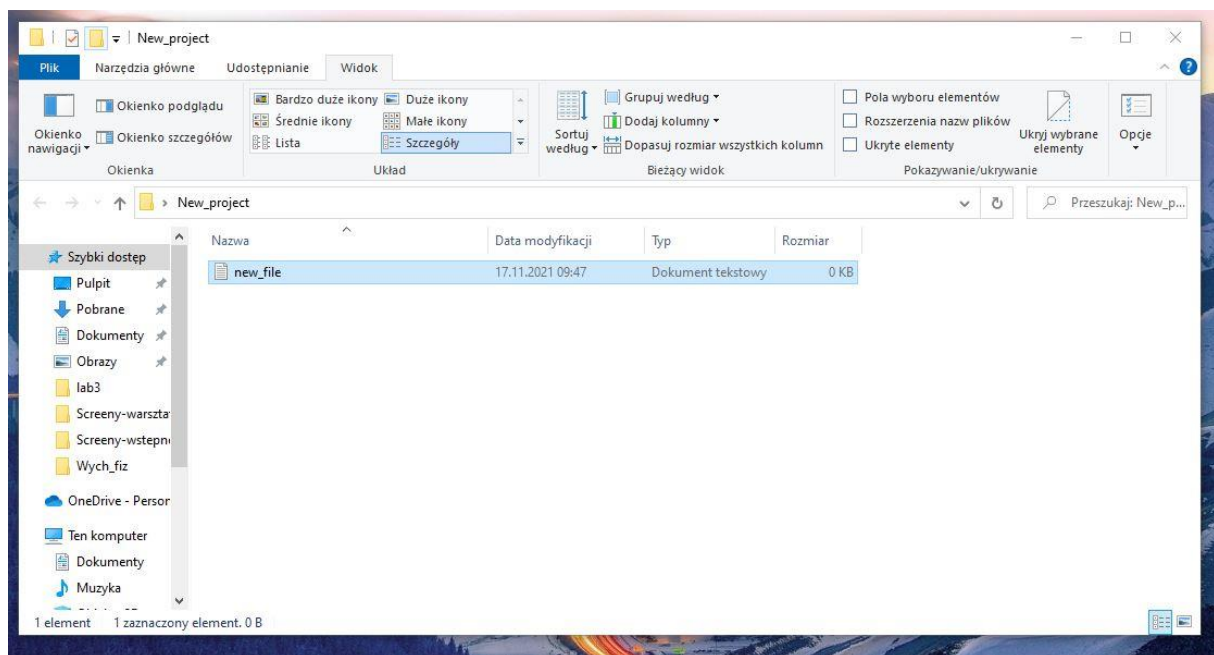
(Screen na następnej stronie.)

```
MINGW64: c:/Users/hubib/Desktop/New_project
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project
$ git init
Initialized empty Git repository in C:/Users/hubib/Desktop/New_project/.git/
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$
```

6. W folderze *New\_project* utwórz nowy plik tekstowy *new\_file.txt*  
(Uwaga Notatnik w Windowsie sam dodaje rozszerzenie)



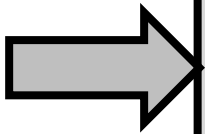
7. W konsoli ponownie wpisz  
`git status`  
Co się zmieniło?

8. Dodaj plik *new\_file.txt* do Gita:

```
git add new_file.txt
```

Następnie sprawdź zmiany:

```
git status
```

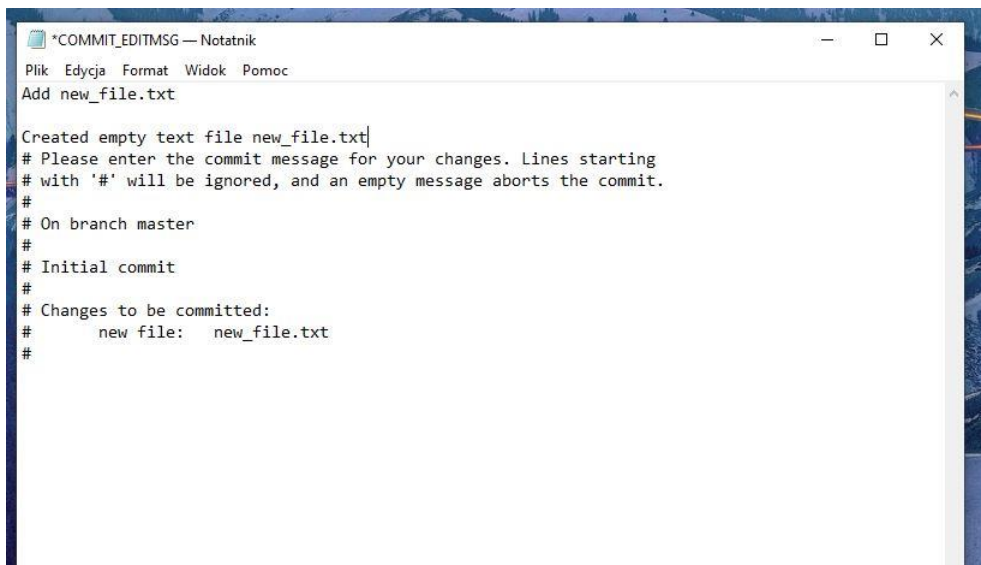


Nadużywanie polecenia *git status* to... bardzo dobra praktyka. Takie sprawdzanie zmian zmniejsza szansę tego, że zrobisz coś, co nie było Twoją intencją.

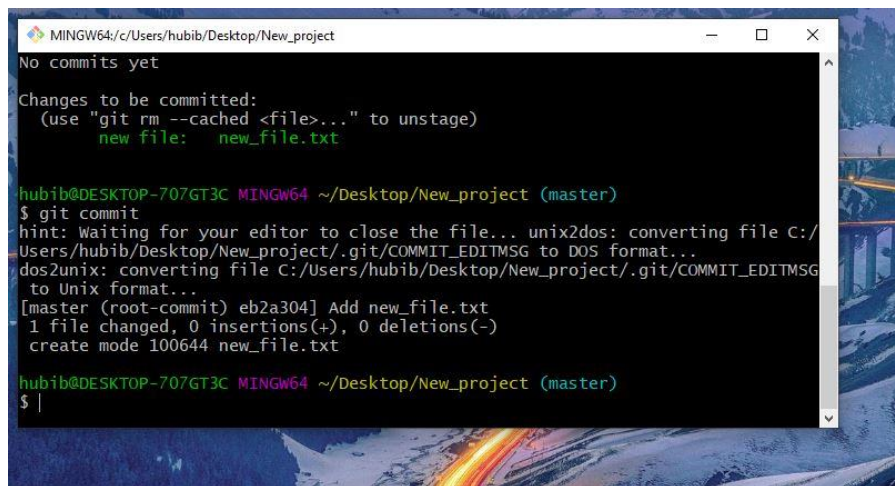
9. Zatwierdź commit:

```
git commit
```

Otworzy się edytor tekstu, w którym trzeba napisać opis commit'a. Zrób to w ten sposób:



Zapisz zmiany i zamknij plik.

A screenshot of a terminal window titled 'MINGW64: c:/Users/hubib/Desktop/New\_project'. The terminal shows the output of a 'git commit' command. It starts with 'No commits yet' and 'Changes to be committed: (use "git rm --cached <file>..." to unstage)' followed by 'new file: new\_file.txt'. Then, it shows the command 'hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New\_project (master)\$ git commit'. The output includes a hint about the editor, file format conversions (unix2dos and dos2unix), and the commit hash '[master (root-commit) eb2a304] Add new\_file.txt'. It also shows '1 file changed, 0 insertions(+), 0 deletions(-)' and 'create mode 100644 new\_file.txt'. The prompt returns to 'hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New\_project (master)\$'.

Gratulacje: właśnie wykonałeś swój pierwszy commit! 😊

10. Znów wpisz

```
git status
```

Co się stało?

11. Otwórz plik *new\_file.txt* w edytorze tekstu. Wpisz do niego linijkę:

*This is file: new\_file.txt!*

Zapisz plik.

12. Sprawdź zmiany:

```
git status
```

13. Dodaj do przechowalni plik *new\_file.txt*:

```
git add new_file.txt
```

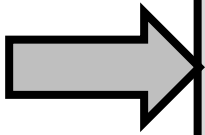
a następnie (na pewno się tego nie spodziewasz) sprawdź zmiany:

```
git status
```

14. Zatwierdź commit:

```
git commit
```

Jako opis wpisz: *Add title to new\_file.txt*



Dbaj, by długość tytułu komentarza nie przekraczała 50 znaków. Dla estetyki tytuł rozpoczynaj wielką literą, nie umieszczaj kropki na jego końcu. Wielu uznaje też zasadę, by tytuł pisać w trybie rozkazującym (np. *Add file.txt* zamiast *Added file.txt*).

### **Zadanie 1.**

Utwórz nowy plik tekstowy `another_file.txt`. Dodaj go do przechowalni (staged to commit) i zrób commit, dodając krótki, ale jednoznaczny opis tego, co zrobiłeś.

15. Wpisz polecenie:

```
git log
```

Poznajesz? 😊



```
MINGW64; c:/Users/hubib/Desktop/New_project
Date: Wed Nov 17 09:55:20 2021 +0100

Add new_file.txt

Created empty text file new_file.txt

~
~
~

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git log
commit c7b268fb488dfeacec59d2361db828b2a31056fb (HEAD -> master)
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 10:17:29 2021 +0100

Add another_file.txt

Created new text file

commit 6ce72dedb333abdeb4f4890c4617f0dfea0818f1
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 10:07:30 2021 +0100

Add title to new_file.txt

Title at the beginning of the file

commit eb2a304f55f835f367bd95320ebab46dd4b9fd18
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 09:55:20 2021 +0100

Add new_file.txt

Created empty text file new_file.txt

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ |
```

16. Poprzednie polecenie to oczywiście przeglądnięcie historii commit'ów naszego repozytorium. Wywołaj teraz to samo polecenie z dodatkową opcją:

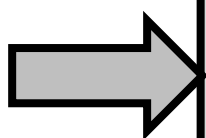
```
git log --oneline
```

Następnie z inną opcją:

```
git log --grep="title"
```

Okazuje się, że można połączyć obie opcje:

```
git log --oneline --grep="title"
```



Polecenie *git log* ma dużo różnych możliwości wyświetlania historii commit'ów. Oprócz dwóch powyższych opcji mamy też m. in. wyświetlenie konkretnej liczby ostatnich commit'ów, czy wyszukiwanie po autorze commit'a.



17. Teraz wyobraź sobie, że zaczynasz pracę wtedy, kiedy nie powinienes jej zaczynać (np. po całonocnej imprezie) i dodajesz zmianę, która ma niepożądany efekt w projekcie.

Otwórz plik *new\_file.txt* i dopisz linijkę:

*Jakas bardzo glupia zmiana*

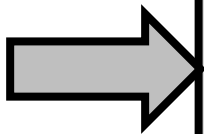
18. Zatwierdź commit:

```
git add .
```

```
git commit
```

Opis commit'a:

*Add enhancement in new\_file.txt*



Nawiasem mówiąc, powyższe polecenie jest przykładem, jak NIE NALEŻY opisywać zmian w commicie - z tytułu nie bardzo wiadomo, o co w tym commicie chodzi. Tytuł powinien od razu naprowadzać na konkretną zmianę, która została w ramach tego commit'u zatwierdzona.

19. Zatem, przełóżmy historię commit'ów:

```
git log
```

Znajdź commit, w którym wprowadziliśmy niepożądane zmiany.

20. Pora naprawić, co popsuliśmy. Musimy odwrócić zmiany, które miały miejsce w commicie *Add enhancement in new\_file.txt*.

W tym celu wyświetlamy historię z opcją *oneline*:

```
git log --oneline
```

Kopiuujemy hash wspomnianego commit'u (numerek z literkami).

(Screen na następnej stronie.)

The screenshot shows a terminal window titled 'MINGW64/c/Users/hubib/Desktop/New\_project'. The terminal output displays a series of Git commits: a commit with hash 'c7b268fb488dfeacec59d2361db828b2a31056fb' by 'Hubert1225' dated 'Wed Nov 17 10:36:07 2021 +0100' with the message 'Add enhancement in new\_file.txt', followed by 'Add another\_file.txt' and 'Created new text file'. Then, a second commit with hash '6ce72dedb333abdeb4f4890c4617f0dfea0818f1' by the same author dated 'Wed Nov 17 10:17:29 2021 +0100' is shown. A context menu is overlaid on the terminal, listing options like 'Open', 'Copy', 'Paste', 'Select All', 'Save as Image', 'Search', 'Reset', 'Default Size', 'Scrollbar', 'Full Screen', and 'Flip Screen'. Below the menu, the terminal shows the command '\$ git log' and its output, which lists the same commits in reverse chronological order. The prompt '\$' is visible at the bottom.

21. Wpisujemy polecenie z wklejonym hash'em:

```
git revert tu-wklej-hasha
```

Otworzy się edytor tekstu, by wpisać w nim opis commit'a - dzieje się tak dlatego, że nasze polecenie wprowadza zmiany, które zapisuje w kolejnym commicie.

(Można zostawić tytuł domyślny.)

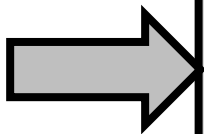
22. Otwórz plik *new\_file.txt*.

Co się stało?

23. Sprawdź historię commit'ów:

```
git log
```

Czy commit, w którym utworzyliśmy niechcianą zmianę, został usunięty? Co się właściwie stało?



Polecenie `git revert` to dość bezpieczny sposób przywracania zmian w Gicie - wycofywany commit pozostaje w historii Gita, przywrócenie zmian następuje jako nowy commit. Istnieje też inne polecenie: `git reset` - ono cofa zmiany, modyfikując historię Gita, zatem jeśli go będziesz używać, musisz dokładnie wiedzieć, co robisz!

### **Zadanie 2.**

Usuń plik `another_file.txt`, a następnie zrób commit. Następnie wycofaj zmiany poleceniem `git revert`. Sprawdź, czy operacje się udały.

## **II. Branch**

1. Sprawdź stan repozytorium:

```
git status
```

2. Sprawdź branch'e dostępne w repozytorium:

```
git branch
```

```
MINGW64:/c:/Users/hubib/Desktop/New_project
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git status
On branch master
nothing to commit, working tree clean

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git branch
* master

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ |
```

3. Utwórz nowy branch, nazwij go: *new\_feature*.

```
git branch new_feature
```

4. Ponownie sprawdź dostępne branch'e.

```
git branch
```

5. Przełącz się na branch *new\_feature*:

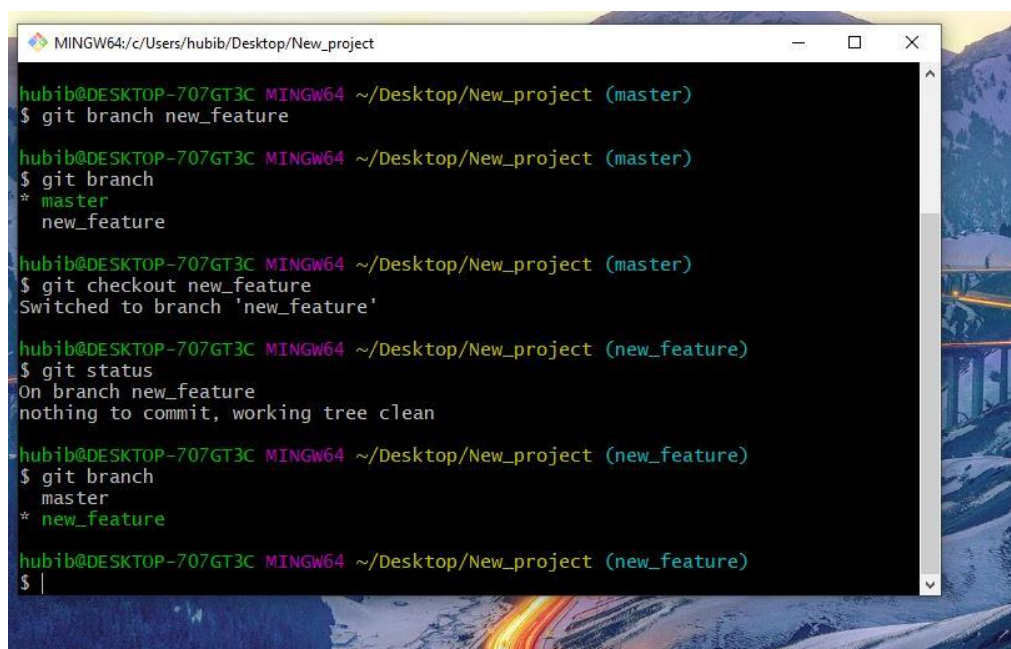
```
git checkout new_feature
```

6. Sprawdź, co teraz zwrócą polecenia:

```
git status
```

```
git branch
```

Co się zmieniło?

A screenshot of a terminal window titled 'MINGW64: c:/Users/hubib/Desktop/New\_project'. The terminal shows a series of Git commands and their outputs. The user is initially on the 'master' branch. They create a new branch named 'new\_feature', then switch to it using 'git checkout new\_feature'. After switching, they check the status, which shows 'On branch new\_feature' and 'nothing to commit, working tree clean'. Finally, they check the branches, showing 'master' and 'new\_feature' as available branches.

```
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git branch new_feature

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git branch
* master
  new_feature

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git checkout new_feature
Switched to branch 'new_feature'

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (new_feature)
$ git status
On branch new_feature
nothing to commit, working tree clean

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (new_feature)
$ git branch
* master
  new_feature

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (new_feature)
$
```

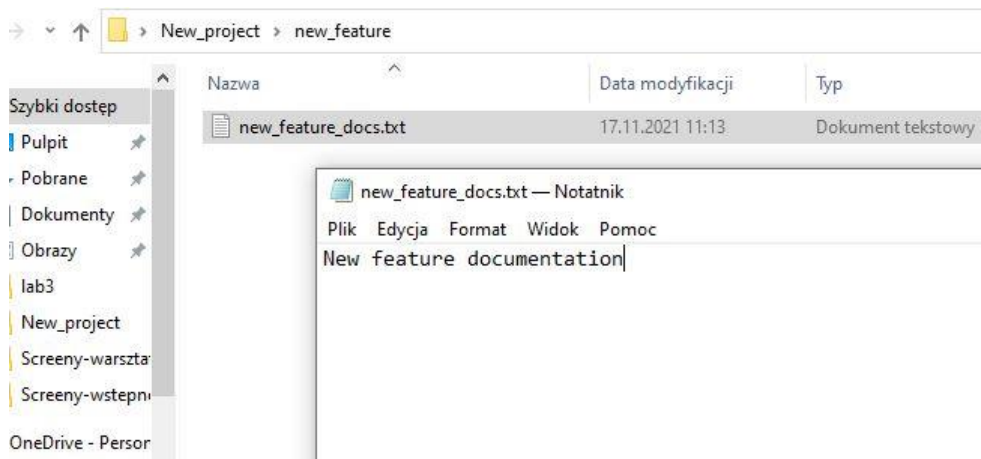
7. Czy coś się zmieniło, jeśli chodzi o zawartość naszego folderu *New\_project*?

Jeśli nie, to co właściwie działy powyższe polecenia?

8. W folderze *New\_project* utwórz folder *new\_feature*, a w nim plik tekstowy *new\_feature\_docs.txt*.

W pliku wpisz: *New feature documentation*

Zapisz plik.



9. Sprawdź git status i utwórz commit.

Tytuł commit'a: *Begin new feature*

10. Sprawdź historię zmian:

```
git log --oneline
```

11. Wróć na domyślną gałąź *master*:

```
git checkout master
```

```
git status
```

```
MINGW64/c/Users/hubib/Desktop/New_project
1 file changed, 1 insertion(+)
create mode 100644 new_feature/new_feature_docs.txt.txt

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (new_feature)
$ git log --oneline
ab6df38 (HEAD -> new_feature) Begin new feature
b36297a (master) Revert "Delete another_file.txt"
7665aca Delete another_file.txt
73411dc Revert "Add enhancement in new_file.txt"
7ed10ee Add enhancement in new_file.txt
c7b268f Add another_file.txt
6ce72de Add title to new_file.txt
eb2a304 Add new_file.txt

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (new_feature)
$ git checkout master
Switched to branch 'master'

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$ git status
On branch master
nothing to commit, working tree clean

hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/New_project (master)
$
```

12. Jak wyglądają pliki projektu? Czy jest obecny dodany przed chwilą folder a w nim plik?  
Jeśli nie, gdzie one są?

13. Przełączmy się na gałąź *new\_feature*

```
git checkout new_feature
```

14. Do pliku *new\_feature\_docs* dopisz linijkę: *Some description of new feature*.

(Oczywiście nie muszę mówić, byś zapisała/zapisał plik 😊)

Utwórz commit z tytułem: *Add description of new feature in docs*

15. Znow wracamy na branch *master*:

```
git checkout master
```

16. Teraz połączmy gałąź *master* z gałęzią *new\_feature*:

```
git merge new_feature
```

17. Sprawdź zawartość naszego folderu. Co się stało?

18. Sprawdź historię commitów.

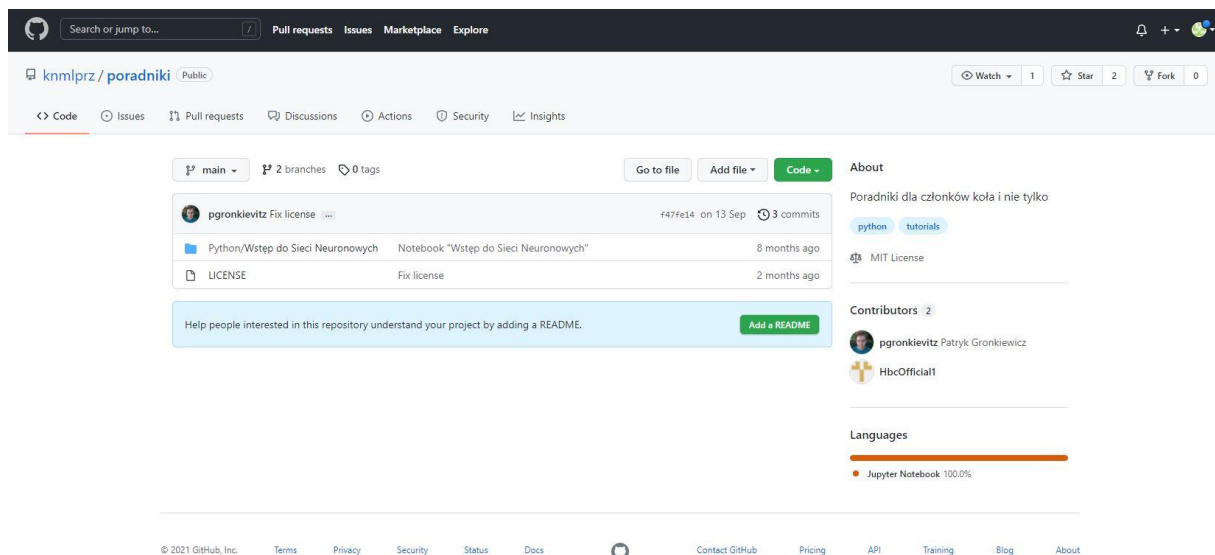
# III. Coś o repozytoriach zdalnych

1. Zaloguj się w serwisie GitHub:

<https://github.com/>

2. Wejdź na stronę repozytorium poradniki należącej do organizacji Koła ML na GitHubie:

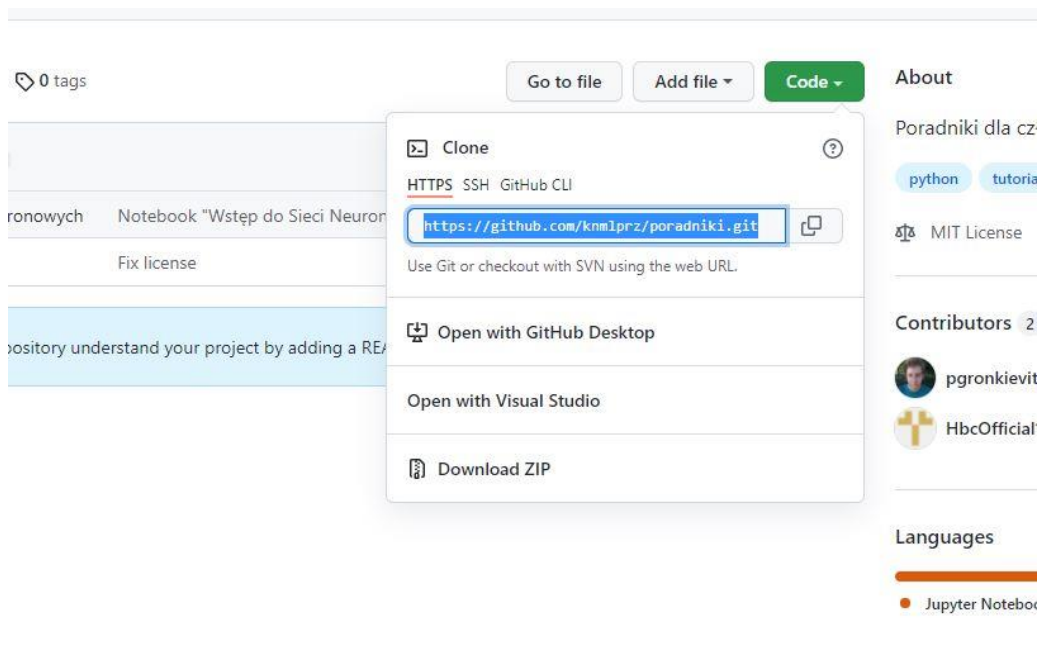
<https://github.com/knmlprz/poradniki>



3. Kliknij przycisk Code, zakładkę HTTPS i skopiuj link.

(Screen na następnej stronie)

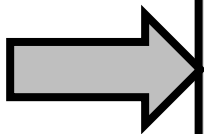




4. Utwórz na pulpicie folder o nazwie *Clone* i uruchom w nim Git Bash'a.
5. Wpisz w konsoli polecenie z wklejonym adresem, który skopiowałeś/łaś z GitHuba:  
`git clone tu_wklej_adres`

```
hubib@DESKTOP-707GT3C MINGW64 ~/Desktop/Clone
$ git clone https://github.com/knmlprz/poradniki.git
Cloning into 'poradniki'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 16 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), 3.24 MiB | 13.11 MiB/s, done.
```

6. Sprawdź zawartość katalogu Clone. Porównaj z tym, co widzisz na GitHubie.
7. W Git Bash'u wejdź do katalogu Poradniki, a następnie sprawdź repozytoria zdalne i stan repozytorium.  
`cd poradniki`  
`git remote`  
`git status`



W tym przykładzie klonujemy repozytorium z użyciem protokołu HTTPS, jednak bardziej zalecana opcja to użycie SSH. Do tego jednak potrzebna jest dodatkowa konfiguracja.

## IV. Inne sposoby zarządzania repozytorium Git

1. Otwórz wiersz poleceń systemu Windows.

Wejdź do katalogu *New\_project*

```
cd C:\Users\<twoja_nazwa>\Desktop\New_project
```

(*Uwaga.* Jeśli pracujesz na Linuksie, po prostu pomiń polecenia związane z wierszami poleceń w Windows. Nie zrobimy tu nic istotnego, tylko pokażemy, że możliwe jest obsługiwanie Gita również z tego miejsca.)

2. Sprawdź stan repozytorium, wpisując w wierszu poleceń:

(tego się nie spodziewasz)

```
git status
```

3. Sprawdź w wierszu poleceń historię commit'ów naszego repozytorium.

```
Wiersz polecenia - git log
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\hubib>cd C:\Users\hubib\Desktop\New_project

C:\Users\hubib\Desktop\New_project>git status
On branch master
nothing to commit, working tree clean

C:\Users\hubib\Desktop\New_project>git log
commit ccc34c2ccfdc6a9fbcd3705af1addb183c288b14 (HEAD -> master, new_feature)
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 11:25:36 2021 +0100

    Technic

commit 597ae4d49057d6e656379989c6ce5d757d278b73
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 11:24:07 2021 +0100

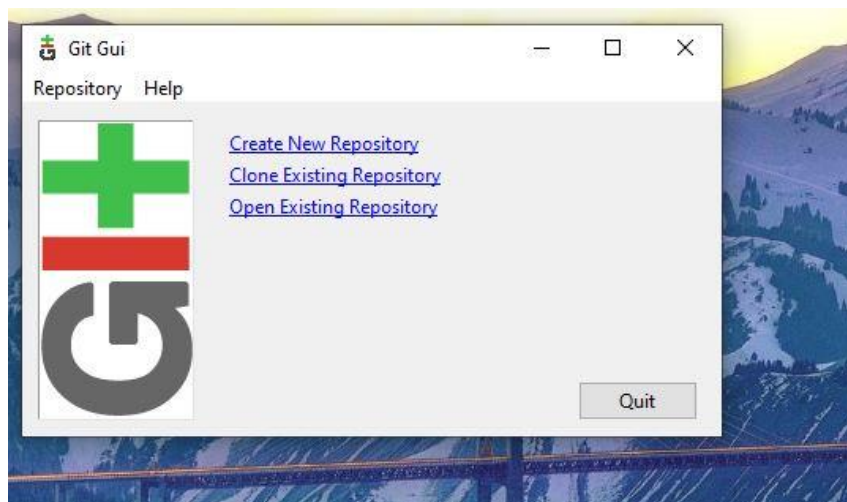
    Add description of new feature in docs

commit ab6df38b2302d6ccc195edfac3d212ac94f367cd
Author: Hubert1225 <164141@stud.prz.edu.pl>
Date: Wed Nov 17 11:16:44 2021 +0100

    Begin new feature

commit b36297a32f025789fa300d3d7f2a1ea17098b5d9
Author: Hubert1225 <164141@stud.prz.edu.pl>
```

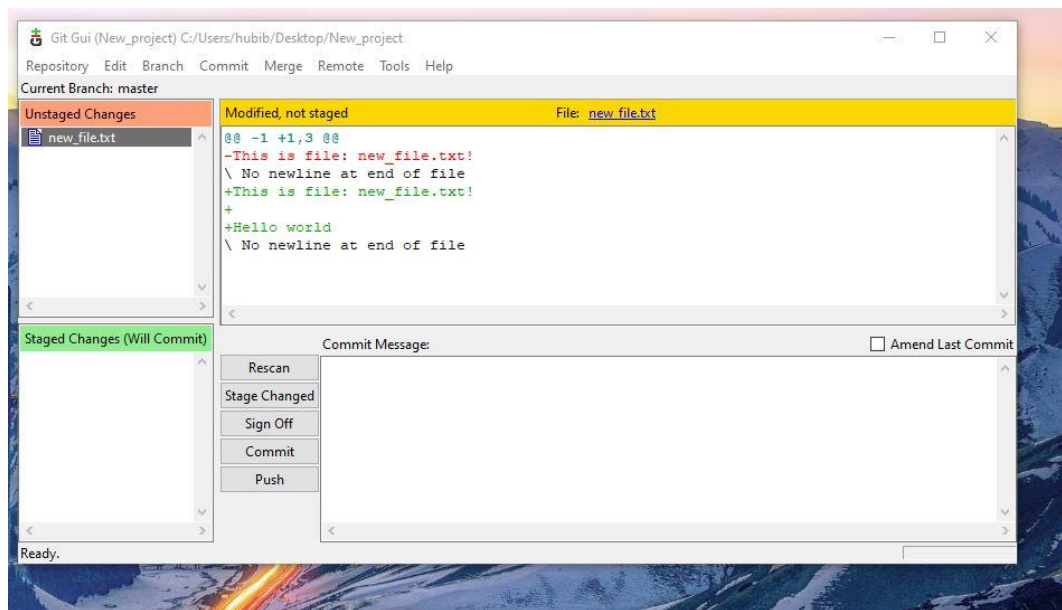
4. Otwórz aplikację Git Gui.



5. W aplikacji Git Gui otwórz nasze repozytorium *New\_project*.

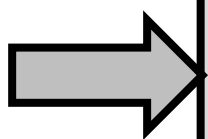
6. Otwórz plik *new\_file.txt* i dodaj kolejną linijkę: *Hello world*.

7. W Git Gui kliknij *Rescan*. Co się pojawiło?  
(Screen na następnej stronie).



8. Kliknij *Stage Changes*, wpisz *commit message* i zatwierdź commit.

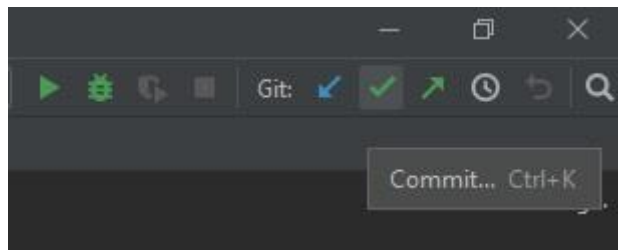
9. Sprawdź, czy commit jest widoczny, gdy sprawdzimy historię w Git Bash.



Jak zdążyłaś/łeś zauważyć, repozytorium Gita to jedno, a aplikacja, która je obsługuje, to drugie. Można obsługiwać jedno repozytorium za pomocą wielu aplikacji.

### Zadanie 3.

Otwórz Twoje ulubione IDE (program, w którym piszesz kod), otwórz w nim folder *New\_project* i poszukaj w IDE sekcji odpowiadającej za Gita. Jeśli ją znalazłeś, dodaj przykładowy plik z kodem do folderu, i zatwierdź commit z poziomu IDE.



*Git w programie Pycharm*

## V. Kolejne kroki (**WAŻNE !**)

1. Wygeneruj klucz SSH i skonfiguruj Gita do łączenia się ze zdalnymi repozytoriami przy użyciu SSH.

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/about-ssh>

2. Dowiedz się, jak kooperować ze zdalnym repozytorium (m. in. *git push*, *git fetch*, *fork*, *pull request*, ...)
3. Poglębiaj swoją wiedzę (np. *git stash*, *git rebase*, wycofywanie zmian z poczekalni, ...)
4. ĆWICZ, ĆWICZ, ĆWICZ 😊

**DZIĘKUJĘ ZA UWAGĘ 😊**