

# Wprowadzenie do systemu Git

Wszystkie stworzone treści oparte są na poradniku spółki Atlassian. Po przeczytaniu tych poradników będziesz wiedzieć w praktyce wszystko, czego potrzeba ci do pracy z Gitem.

<https://www.atlassian.com/git/tutorials/>

## 1. Czym jest Git ?

Jest to jeden z Systemów Kontroli Wersji ( ang. System Version Control – SVN ) , których zadaniem jest dostarczenie narzędzi do monitorowania zmian w naszej pracy ( dokładnie kodzie źródłowym ) na przestrzeni czasu. Rejestruje każdą zmianę i dokumentuje ją. Dzięki temu kilku developerów może pracować bez obaw nad projektem, na własnych kopiach plików, a następnie scalać je z głównym "folderem". Git wykłnie wszystkie błędy i konflikty które nastąpiły podczas pracy ( np. dwóch programistów zedytowało ten sam plik, każdy w inny sposób ). Git powstał również z myślą o bezpieczeństwie - wszystko przechowujemy na zewnętrznym serwerze, dzięki monitorowaniu zmian zawsze możemy cofnąć nieoczekiwane błędy, a także posiadać kilka wersji naszych plików.



## 2. Dlaczego muszę znać Gita ?

Git powstał w roku 2005 i został stworzony przez nikogo innego jak ojca linuxa – Linusa Torvaldsa. Od tego czasu stał się wręcz standardem w pracy każdego programisty. Poświęć w zespole kilka dni na jego poznanie, a znacznie poprawisz efektywność pracy twojej jak i twoich współpracowników. Dla współczesnego developera nieznajomość Gita jest brakiem elementarnej wiedzy, i w praktyce każda firma wymaga umiejętności jego używania. Jest to oprogramowanie na tyle przemyślane, że na naukę podstaw stracisz jeden dzień, i wystarczą one do większości zadań w 90 % zajęć. Zaawansowane techniki Gita stosowane są przeważnie tylko przy bardzo dużych zespołach i projektach.

### 3. Słowniczek

**Zanim rozpoczniemy zabawę na dobre warto znać te kilka pojęć :**

**Repozytorium** to struktura zawierająca historię projektu – pliki, wersje, gałęzie, historię zmian itd.

**Kopia robocza** – to lokalna wersja repozytorium na którym pracuje programista

**Obiekt** – cokolwiek w repozytorium, pliki, branche, commity.

**Blob** – plik w gicie

**Drzewo** – katalog w gicie

**Commit** – jest to uchwycony stan projektu w danej chwili czasu. Zawiera informację o stanie repozytorium przed zmianami i po ich wprowadzeniu, a także komentarz programisty do zmian. Pierwszy commit to nic innego jak inicjacja projektu.

**Push** – jest to wprowadzenie commita do repozytorium na serwerze.

**Merge** – Proces scalania dwóch branchy.

**Branch** – jest to oddzielna ścieżka pracy nad projektem, stworzona z danego stanu repozytorium. Więcej o gałęziach w następnym punkcie

**HEAD** – nagłówek jest informacją, na której gałęzi wykonujemy operacje.

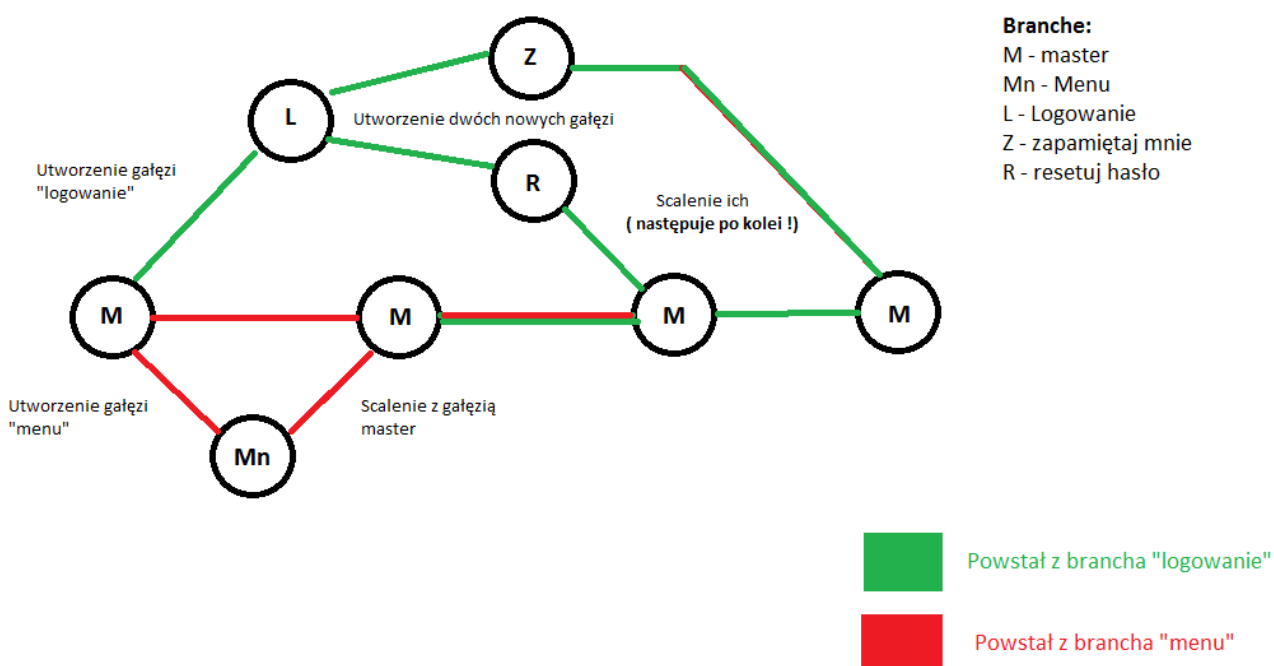
### 4. Podstawowe operacje, ich przebieg i zasada działania

Przedstawię tu dwa podstawowe procesy, które należy znać na początek – aktualizację repozytorium, i jak wygląda operacja na branchach. Pierwszy przebiega bardzo prosto. Najpierw informujemy gita które pliki zostały zmienione tak, aby wiedział co wysłać do repozytorium a także sprawdzić pod względem konfliktów ( wysyłanie dużego repozytorium w całości, biorąc pod uwagę wiele branchy, nie byłoby mądrym pomysłem). Następnie opisujemy to co zrobiliśmy, czyli commitujemy zmiany. Na zakończenie pushujemy je do główny repozytorium, i cieszymy się nową wersją projektu .

Przejdźmy teraz do najważniejszego punktu, czyli gałęzi. Tworzymy je, aby móc pracować nad własną kopią repozytorium, “nie wtaczając nosa w nie swoje sprawy”, niech każdy developer pracuje nad swoją częścią kodu. Gotową gałąź scalamy następnie z inną by stworzyć kolejną wersję repozytorium. Ostatecznie pod koniec projektu pozostaje jedna, główna gałąź “master” która jest w gruncie rzeczy gotowym produktem. Zastosowań branchy jest bardzo wiele, np. do przygotowania innej wersji programu czy też bezpiecznego środowiska do testowania niebezpiecznej zmiany w kodzie. Ogranicza nas jedynie wyobraźnia.

Wydawać może się to z początku lekko przytłaczające, jednak obrazek wraz z wyjaśnieniem, powinien rozwiązać wątpliwości

1. Główną gałęzią repozytorium jest branch 'master', tworzony na początku projektu.
2. Pierwszy programista postanawia stworzyć logowanie do strony internetowej – tworzy więc branch „logowanie” z brancha 'master'.
3. Drugi programista postanawia stworzyć menu na stronie – tworzy branch „menu”. Tydzień później, po licznych commitach naprawiających bugi postanawia scalić swój branch, z branchem master, czyli wykonać merge. Zmienił linijkę z imieniem „Jacek” na „Franek” w pliku który istniał w pierwotnej wersji brancha master. Podczas pushowania zmian Git daje o tym znak, a programista będąc świadomy tych zmian używa przełącznika –force w poleceniu konsoli, by wymusić swe zmiany. Powstaje nowa wersja gałęzi master.
4. Do developera pracującego nad logowaniem dołącza trzeci programista, gdyż stwierdza, że inaczej nie zdążą do logowania dodać opcji „zapamiętaj mnie” oraz „zresetuj hasło”. Jako że generalnie system logowania jest już skończony obydwójce tworzą swoje branchy z aktualnej na której pracują (**branch „logowanie” nie master !**), i rozpoczynają kodowanie w pocie czoła. Po zakończeniu prac robią to samo co programista pracujący nad gałęzią „menu” – mergują swoje gałęzie do mastera, naprawiając po drodze konflikty.
5. Powstaje finalna wersja projektu, czyli piękna strona z menu, oraz systemem logowania.



## 5. Praktyka – polecenia ( WRESZCIE ! )

Tak duża dawka teorii powinna zobrazować w 100 procentach to co pokażę teraz na poleceniach konsoli .

Dodatkowo dodam jeszcze parę poleceń od siebie które będziesz musiał wykonać na początku projektu. Nie powinno być żadnych wątpliwości co dzieje się na screenach, jeżeli taka pozostaje, nie masz innego wyboru jak użyć mej pomocy :D

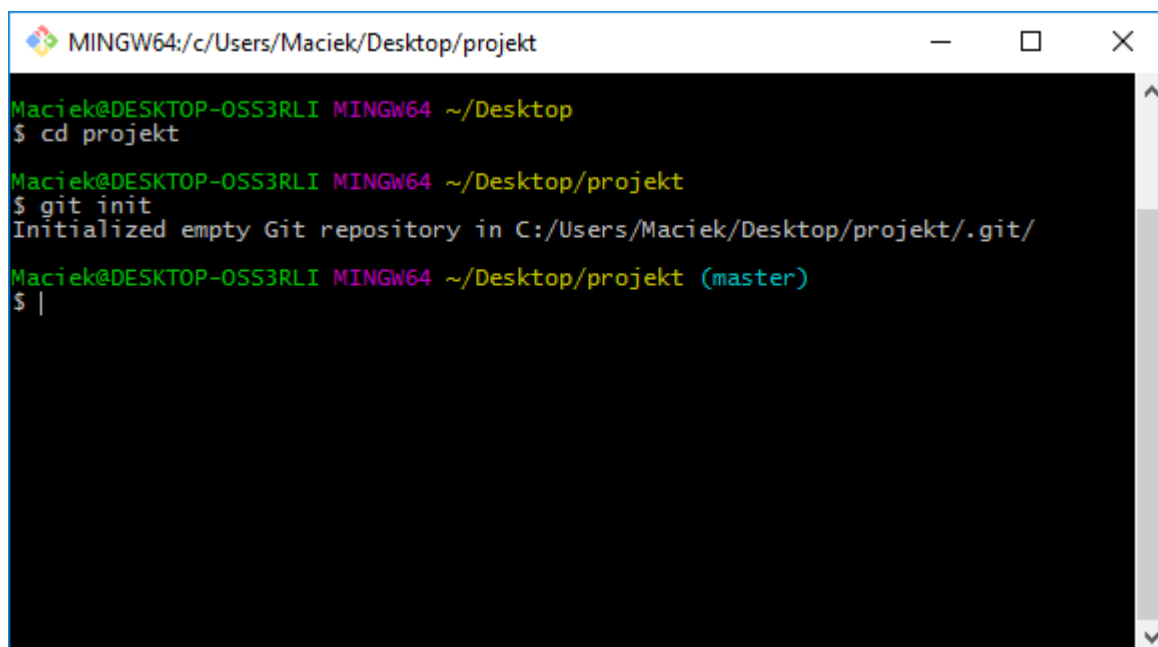
Przed rozpoczęciem pracy po raz pierwszy ( czynność tą trzeba wykonać tylko po świeżej instalacji gita), uruchom dwa polecenia. Użyj emaila z konta bitbucket, i nicku również z tego serwisu ☺. Git będzie wiedział jaka osoba będzie podczas wysyłania zmian prosiła o autoryzację na wprowadzenie zmian. Jest to operacja obowiązkowa

***git config --global user.email „twój@email.com”***

***git config --global user.name „Nick”***

Aby stworzyć nowe repozytorium lokalnie, wejdź do dowolnego katalogu w konsoli i uruchom polecenie

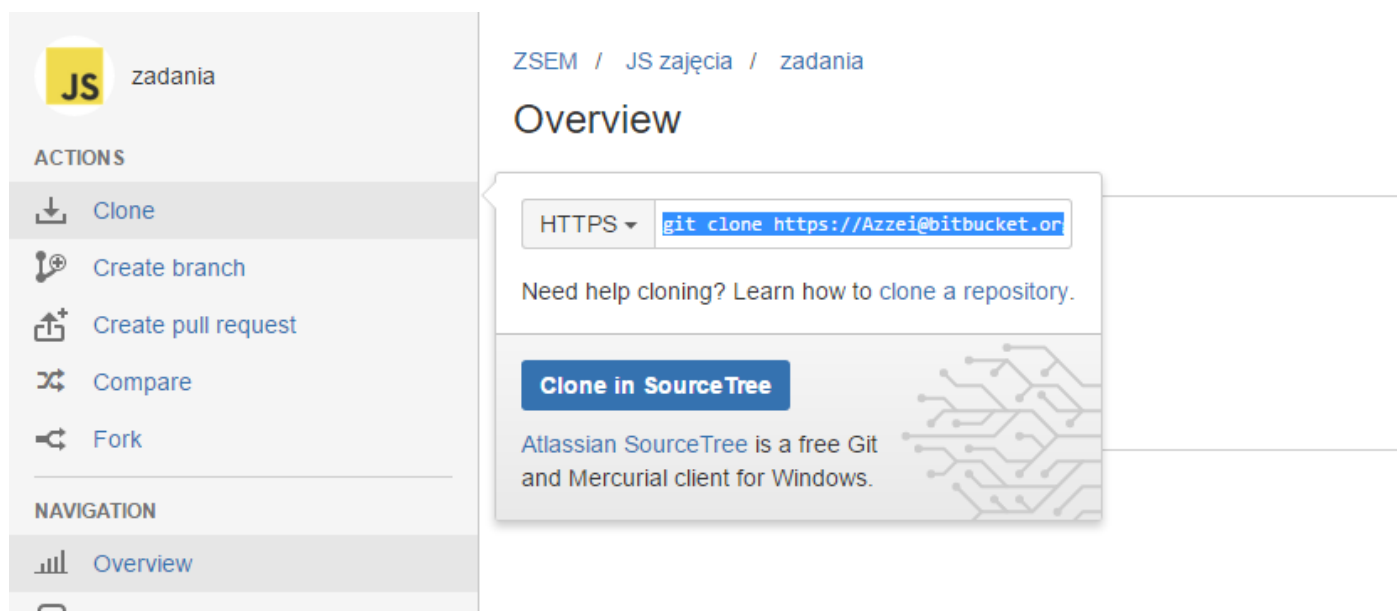
## ***git init***



```
MINGW64:/c/Users/Maciek/Desktop/projekt
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop
$ cd projekt
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt
$ git init
Initialized empty Git repository in C:/Users/Maciek/Desktop/projekt/.git/
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ |
```

Jeżeli wolisz otrzymać gotowy folder, z zainicjowanym repozytorium, utwórz takie na dowolnym serwisie z systemem git (github, gitlab, w naszym przypadku bitbucket). Następnie wykonaj polecenie git clone. Bitbucket pozwala generować to polecenie. Nie będę opisywał procesu tworzenia repozytorium na bitbuckecie, gdyż przeprowadzę ten krok dla każdej nowej osoby na zajęciach.

## ***git clone źródło***



The screenshot shows the Bitbucket 'Clone' dialog box. On the left, there's a sidebar with 'JS zadania' and a list of actions: 'Clone', 'Create branch', 'Create pull request', 'Compare', and 'Fork'. The 'Clone' action is selected. The main area shows the 'Overview' section with the 'HTTPS' protocol selected and the URL 'git clone https://Azzei@bitbucket.or'. Below the URL, there's a link to 'Learn how to clone a repository.' and a button labeled 'Clone in SourceTree'. At the bottom, there's a note about Atlassian SourceTree being a free Git and Mercurial client for Windows.

```
MINGW64:/c/Users/Maciek/Desktop/projekt

Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ git clone https://Azzei@bitbucket.org/zsem/materialy.git
Cloning into 'materialy'...
remote: Counting objects: 12, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 12 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (12/12), done.
Checking connectivity... done.

Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$
```

Jeżeli chcesz podpiąć gotowy już projekt pod repozytorium, służy do tego polecenie:

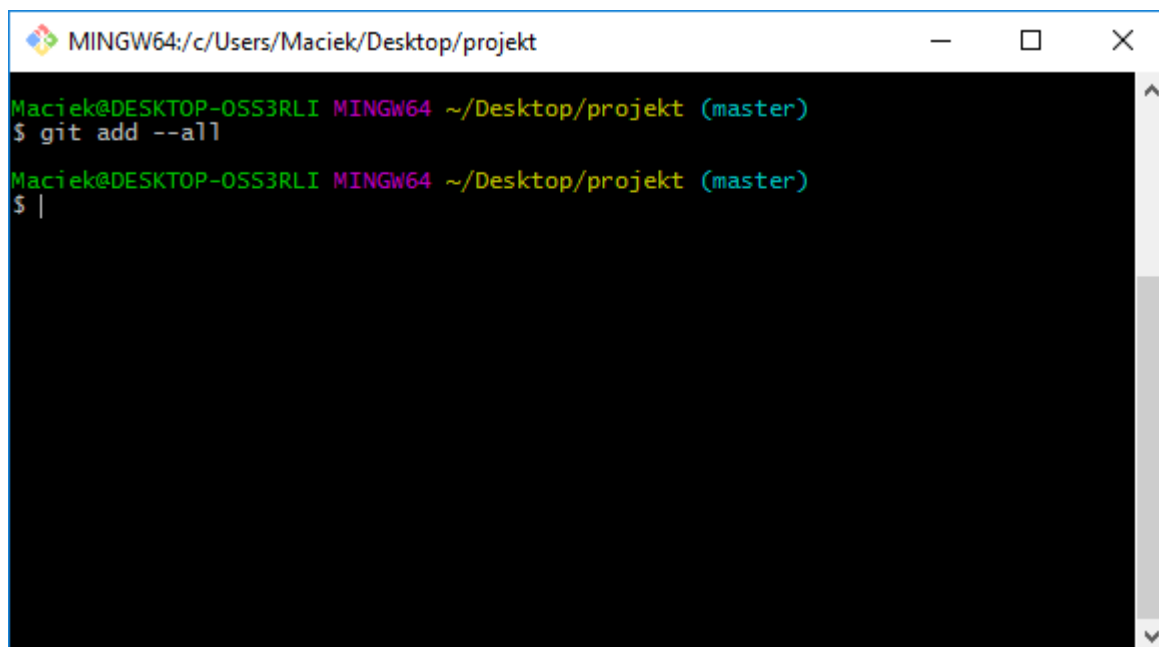
***git remote add origin źródło***

```
MINGW64:/c/Users/Maciek/Desktop/projekt

Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ git remote add origin https://Azzei@bitbucket.org/zsem/materialy.git
```

Pliki do wysłania przygotowujemy poleceniem `git add`. Prostą komendą na dodanie wszystkich zmienionych blobów (słowo użyte celowo ☺) jest `git add -all`.

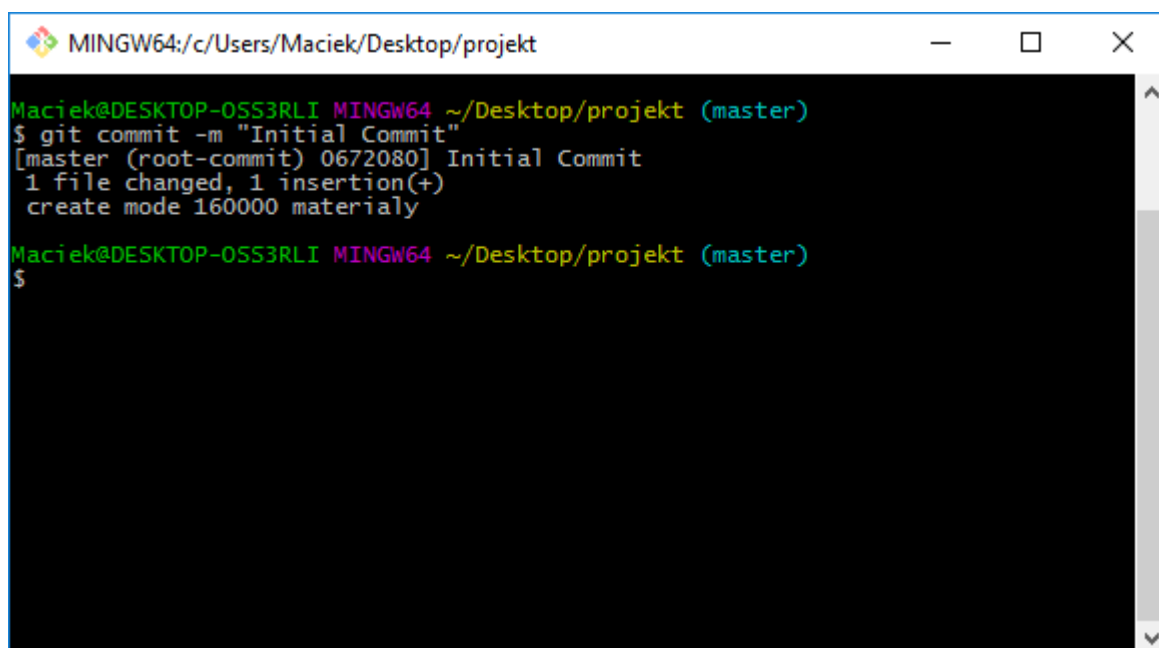
## *git add pliki*



```
MINGW64:/c/Users/Maciek/Desktop/projekt
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ git add --all
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ |
```

Nie pozostaje nam nic innego jak commit i push zmian do repozytorium na serwerze. Polecenie `git commit` przyjmuje przełącznik `-m` (od Message), jeżeli go nie użyjemy otworzony zostanie edytor tekstu VIM, w którym możemy opisać zmiany. Aby z niego wyjść naciśnij klawisz escape, wpisz „:q” i naciśnij enter. Aby zapisać zmiany które wpisałeś do edytora postąp tak samo, lecz zamiast „:q” użyj „:wq” (q – quit, w – write). Jeżeli zaczniesz linijkę w vimie od znaku „#”, git nie uwzględni jej i nie wyśle do repozytorium. Możesz dzięki temu lokalnie sprawdzać twoje komentarze do commita, dostępne tylko dla ciebie.

## *git commit -m „wiadomość”*

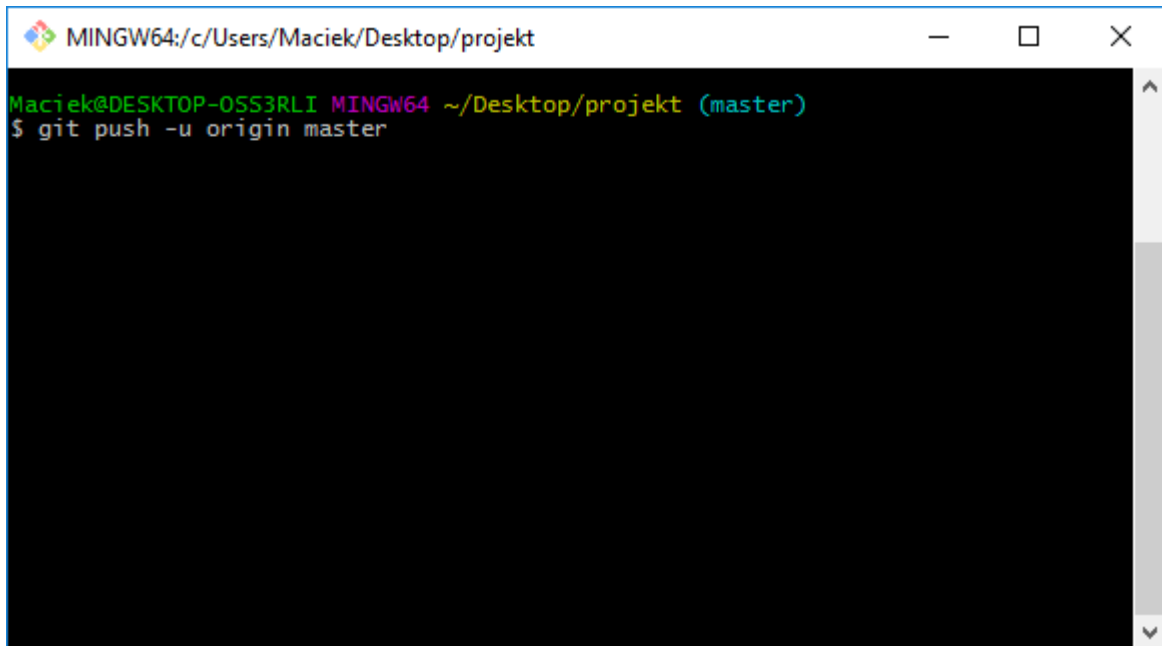


```
MINGW64:/c/Users/Maciek/Desktop/projekt
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$ git commit -m "Initial Commit"
[master (root-commit) 0672080] Initial Commit
1 file changed, 1 insertion(+)
create mode 160000 materialy
Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)
$
```

Pozostaje już tylko wyłać zmiany na serwer poleceniem `git push`. Pojawi się w nim część o nazwie „origin”, jest to źródło. „Origin” w tym przypadku, jest to główny serwer repozytorium, gdyż jego kopie mogą się znajdować również na innych serwerach. Nazwa gałęzi umożliwia przekierowanie zmian na inny branch,

niekoniecznie o tej samej nazwie. Przełącznik `-u`, jest to skrót od upstream (swojego rodzaju zadanie domowe, jest on nieistotny, ale możesz o nim poczytać).

### ***git push -u źródło nazwa\_gałęzi***

A screenshot of a terminal window titled 'MINGW64:/c/Users/Maciek/Desktop/projekt'. The prompt is 'Maciek@DESKTOP-OSS3RLI MINGW64 ~/Desktop/projekt (master)'. The command '\$ git push -u origin master' has been entered and executed. The terminal background is black with green and white text. The window has standard Windows-style title bar controls (minimize, maximize, close) in the top right corner.

Brawo ! Twoja pierwsza zmiana została wprowadzona do repozytorium !

A teraz jeszcze kilka poleceń do pobierania zmian, i operacji na branchach.

### ***git pull źródło (np.origin) nazwa\_gałęzi***

Pobiera zawartość podanej gałęzi do tej w której się znajdujemy, aktualizując wszystkie pliki. Operacja ta może być wymagana przed pushem, jeżeli wystąpiły w repozytorium jakieś zmiany od czasu pobrania go lokalnie.

### ***git checkout -b nazwa\_gałęzi***

Tworzy LOKALNIE nową gałąź z tej na której się znajdujemy, i przełącza nas na nią (czyli przenosi HEAD). Aby utworzyć gałąź w repozytorium na serwerze, stwórz ją lokalnie z innej gałęzi, a następnie zpushuj na serwer, zostanie ona utworzona automatycznie, i co więcej, git jest na tyle mądry, że uwzględni na statystykach z której gałęzi powstała.

### ***git branch***

Listuje wszystkie gałęzie w repozytorium LOKALNYM (dzięki przełącznikom można zobaczyć te z serwera)

### ***git checkout -d nazwa\_gałęzi***

Usuwa wskazaną gałąź (musisz być na innej niż ta usuwana)

## ***git checkout nazwa\_gałęzi***

Przełącza nas między gałęziami.

## **6. Na zakończenie**

Kilka rzeczy które należy zauważyć. Po użyciu git init, pojawi się w waszym katalogu ukryty folder .git. Pod żadnym pozorem nie należy go usuwać ani modyfikować, gdyż tak naprawdę to on decyduje o tym czy dany katalog jest repozytorium. Usuwając go usuwasz dane o repozytorium, i drzewo staje się zwykłym folderem.

Po zmianie brancha, zauważyć można że poprzednie pliki znikają i pojawiają się nowe. Git ukrywa je, dlatego też nie przejmuj się dużą wagą katalogów, jeżeli taka wystąpi.

Możesz utworzyć specjalny plik .gitignore który umożliwia ustalenie, których plików i katalogów git ma nie brać pod uwagę przy commitowaniu i wysyłaniu. Bardzo przydatna opcja, której używa się w bardzo wielu przypadkach. Jak wygląda środek takiego pliku ? Mniej więcej tak:

Folder/\* (wszystko w katalogu, ale nie on, sam)

Plik1.txt

Plik2.txt

Folder2

Kolejne pozycje do ignorowania pojawiają się w nowej linijce. Popróbuj sam !

**Mam nadzieję, iż artykuł się spodobał !**



**IN CASE OF FIRE**

🔦 **GIT COMMIT**  
🔦 **GIT PUSH**  
🚒 **LEAVE BUILDING**