



Autor : Sebastian Waśkiewicz

Rozproszony system kontroli wersji

### Spis treści

- 1.Czym jest Git?
- 2.Najpopularniejsze hostingowe serwisy internetowe przeznaczone dla projektów programistycznych wykorzystujących Git.
3. github.com
4. Instalacja i konfiguracja Gita
- 5.Git : Najważniejsze komendy

## 6. GIT – ignorowanie plików.

### 1.Czym jest Git?

Git jest rozproszonym systemem kontroli wersji. Stworzył go pan Torvalds który odpowiedzialny jest za stworzenie jądra Linux.

Git służyć miał właśnie do rozwijania tego systemu operacyjnego.

System kontroli wersji pozwala nam śledzić wszystkie zmiany dokonywane na plikach na których pracujemy i umożliwia nam przywołanie dowolnej wcześniejszej wersji.

2.Najpopularniejsze hostingowe serwisy internetowe przeznaczone dla projektów programistycznych wykorzystujących Git.

# GitHub



# GitLab

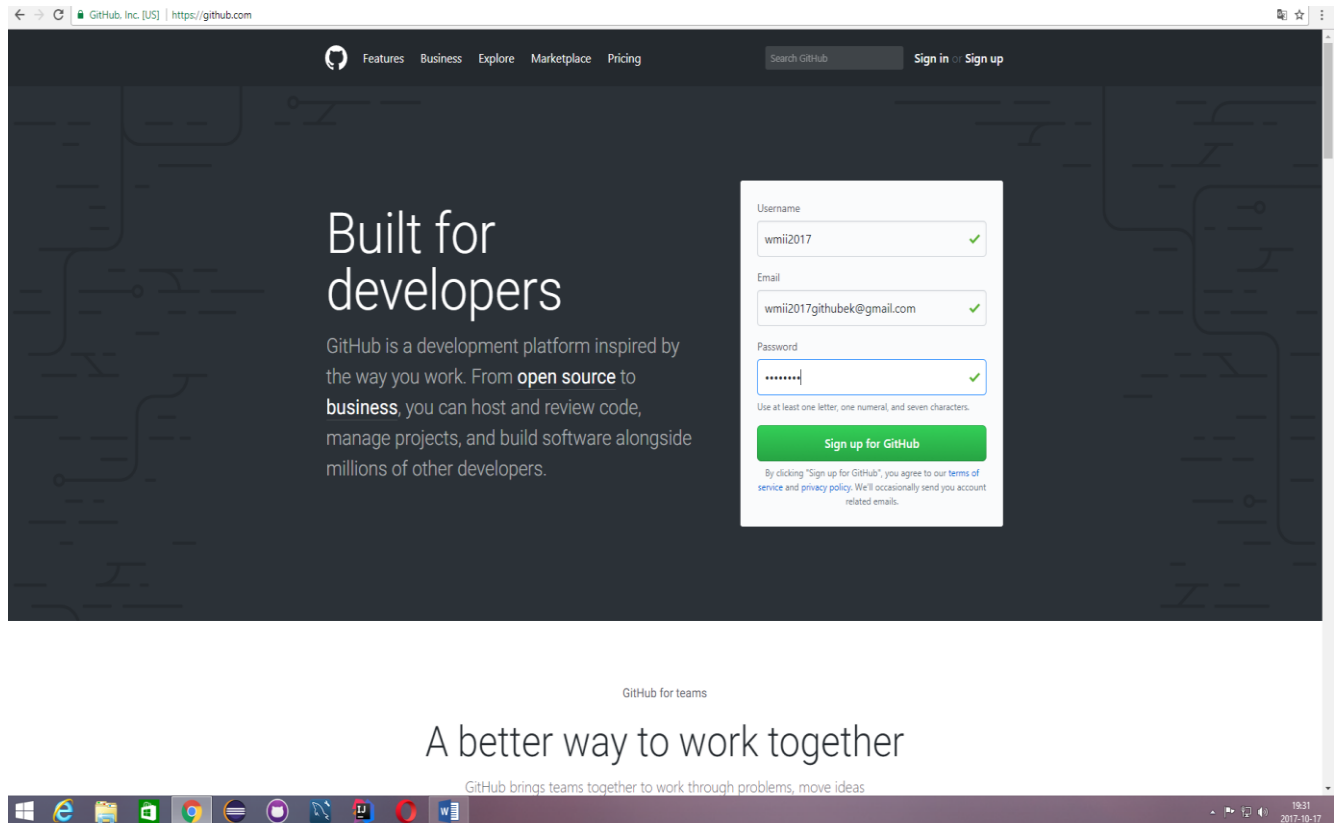


# Bitbucket

W moim referacie posłużę się w 100% GitHubem – jest to serwis bardzo intuicyjny i darmowy.

## 3.github.com

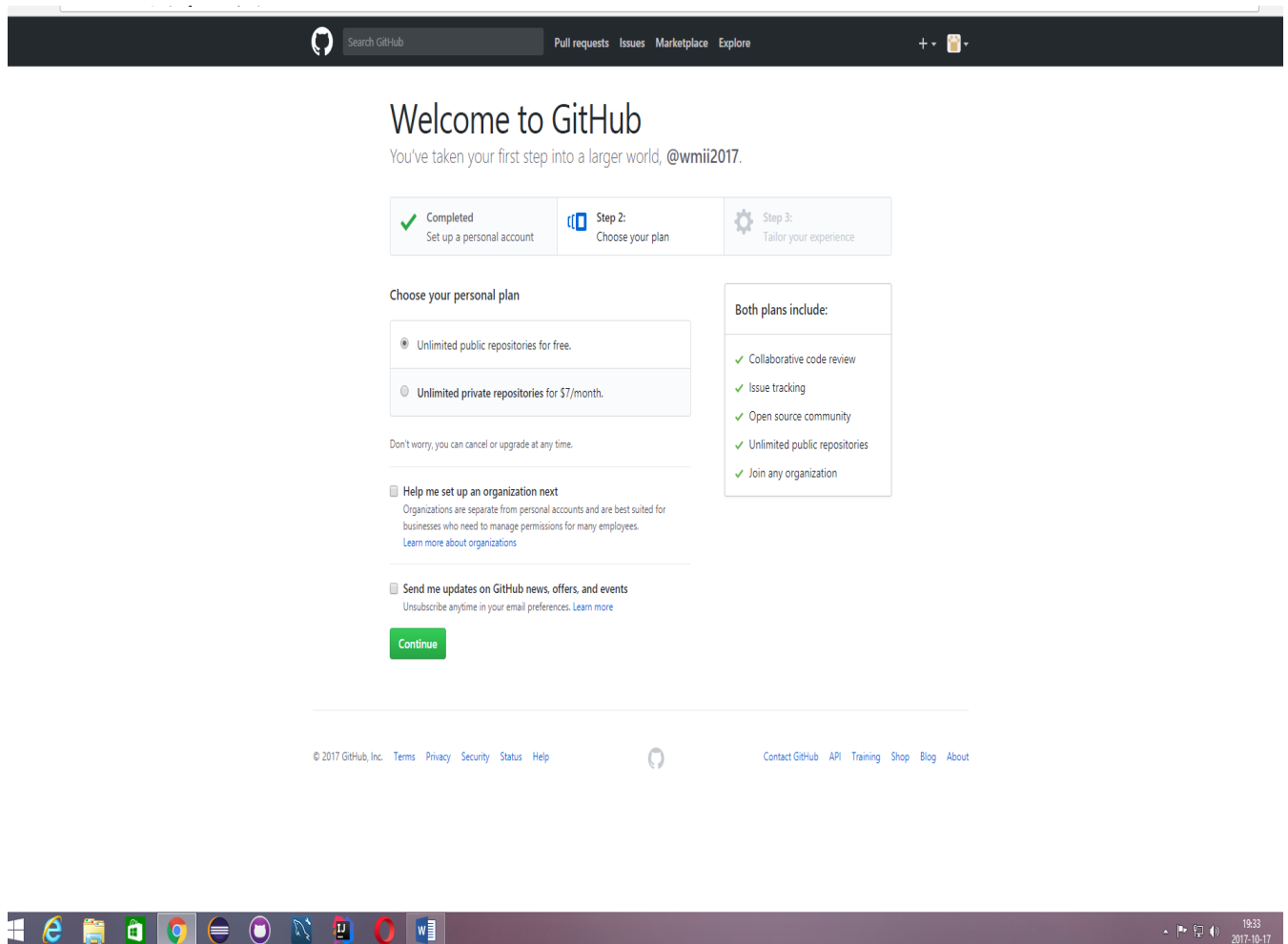
### 3.1) Zakładanie konta



wmii2017

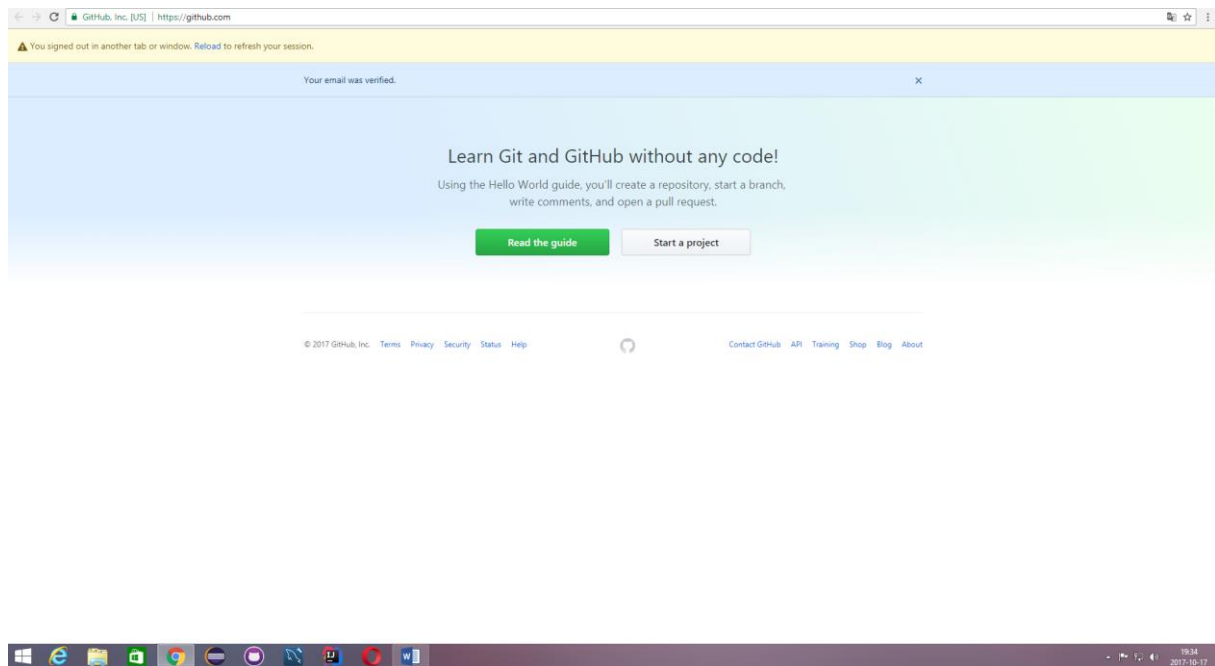
wmii2017github@gmail.com

Uzupełniamy pola i klikamy w zielony przycisk.



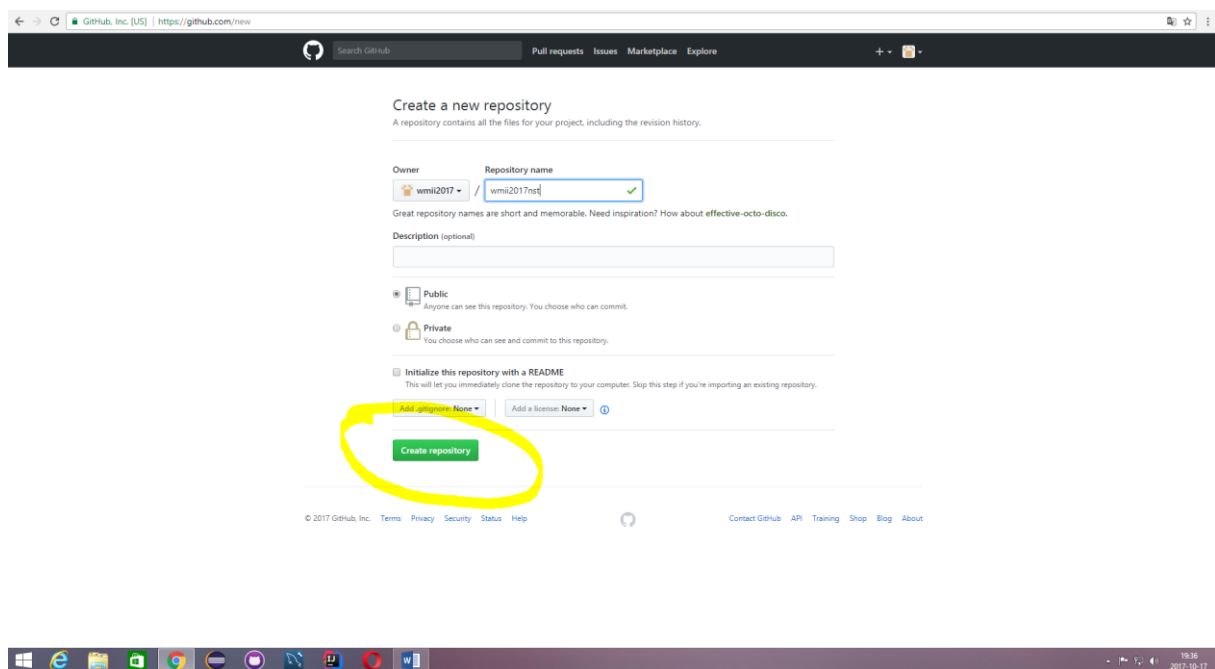
Otrzymujemy taki ekran.

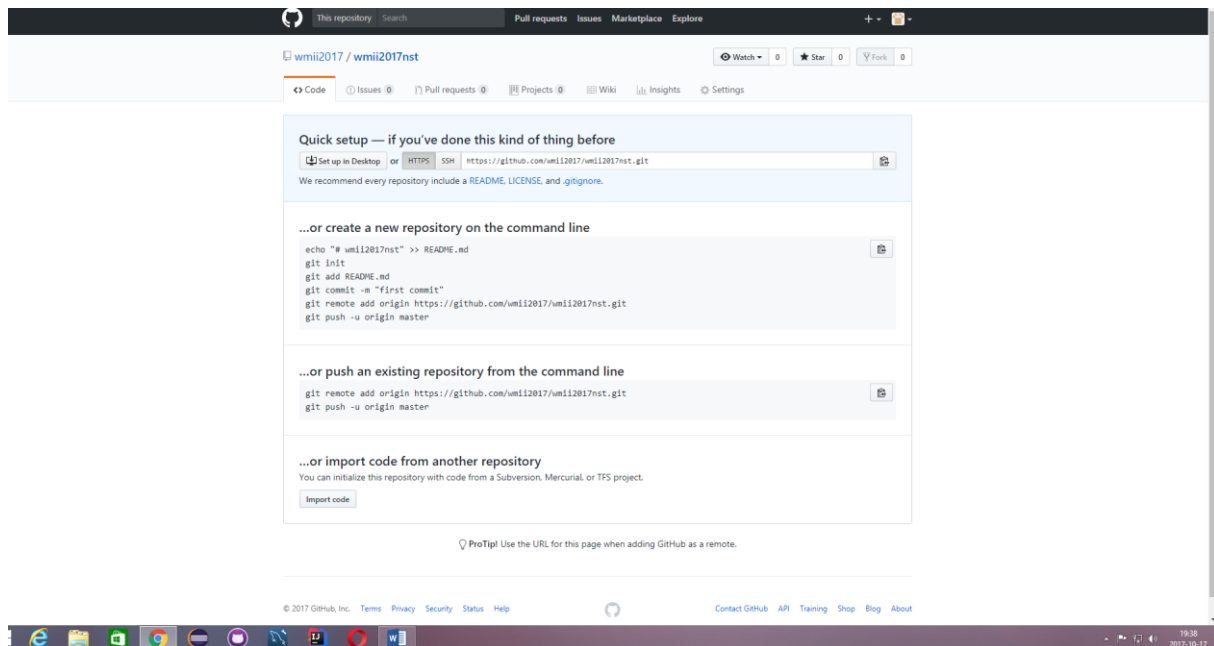
**W opcji Choose your personal plan wybieramy Darmowe publiczne repozytorium i idziemy dalej.**



Wybieramy opcję Start a project.

## 3.2 Stwórzmy nasze pierwsze repozytorium.





Repozytorium zostało stworzone.

## 4. Instalujemy Gita

Przechodzimy na stronę <https://git-scm.com/downloads> i wybieramy wersję programu zgodną z naszym systemem operacyjnym.

Klikamy ciągle Next aż do rozpoczęcia instalacji 😊

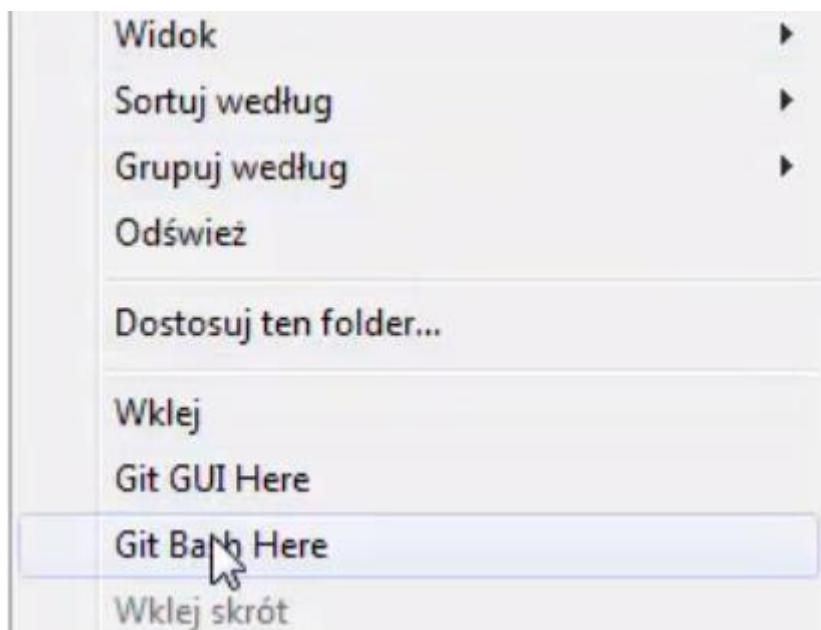
## 5.Tworzenie repozytorium

Na pulpicie utwórz w folder – proponuje nazwę repozytorium. W środku zakładamy folder np. o nazwie Pliki.

Przechodzimy w folder Pliki i klikamy prawym przyciskiem myszy w jego „wnętrznosciach”



Z listy wybieramy opcję Git Bash Here.



Otrzymaliśmy konsolę podobną do tej która jest domyślną systemową Windowsa.



```
Sebastian@Sebastian MINGW64 ~/Desktop/wmii/wmiinstgit
$ git init
Initialized empty Git repository in C:/Users/Sebastian/Desktop/wmii/wmiinstgit/.git/

Sebastian@Sebastian MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ |
```

Wpisujemy git init – To polecenie stworzy nam nowy podkatalog zawierający wszystkie niezbędne pliki – szkielet repozytorium.

Następnie konfigurujemy ustawienia użytkownika :

Git config –global user.name „wmii2017”

```
Sebastian@Sebastian MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git config --global user.name "wmii2017"
```

Git config –global user.email

[wmii2017github@gmail.com](mailto:wmii2017github@gmail.com)

```
Sebastian@Sebastian MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git config --global user.email "wmii2017github@gmail.com"

Sebastian@Sebastian MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ |
```

Ponownie wpisujemy komendę git init

## Teraz musimy utworzyć klucz SSH.

### Czym one są?

Klucze SSH zapewniają większe bezpieczeństwo logowania do serwera poprzez protokół SSH, niż w przypadku użycia samego hasła. W procesie generowane są dwa klucze: prywatny i publiczny. Tak przygotowany klucz publiczny umieszcza się na dowolnym serwerze, który chcemy zabezpieczyć natomiast klucz prywatny wykorzystywany będzie w kliencie jaki wykorzystujemy do połączenia SSH. Prawidłowe uwierzytelnienie nastąpi, gdy klucz prywatny i publiczny będą zgodne.

Wchodzimy w C:\Users\nazwa\_uzytkownika i klikamy ponownie Git Bash Here.

Wykonujemy polecenie `ssh-keygen -t rsa -C "adres-email@uzytkownika"`

Klikamy enter i potwierdzamy dwa razy naszym hasłem.

```
MINGW64:/c/Users/Sebastian/.ssh
Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-ODqmhkNDZyJK/agent.1976; export SSH_AUTH_SOCK;
SSH_AGENT_PID=5416; export SSH_AGENT_PID;
echo Agent pid 5416;

Sebastian@Sebastian: MINGW64 ~/.ssh
$ eval "$(ssh-agent)"
Agent pid 6684

Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-add ir_rsa
ir_rsa: No such file or directory

Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-add id_rsa
Enter passphrase for id_rsa:
Identity added: id_rsa (id_rsa)

Sebastian@Sebastian: MINGW64 ~/.ssh
$ |
```

Teraz przechodzimy do naszego folderu który zostanie wygenerowany automatycznie i jego nazwa to .ssh. Zawartością tego folderu będą dwa pliki:

Nazwa	Data modyfikacji	Typ	Rozmiar
id_rsa	2017-10-17 20:19	Plik	2 KB
id_rsa.pub	2017-10-17 20:19	Microsoft Publish...	1 KB

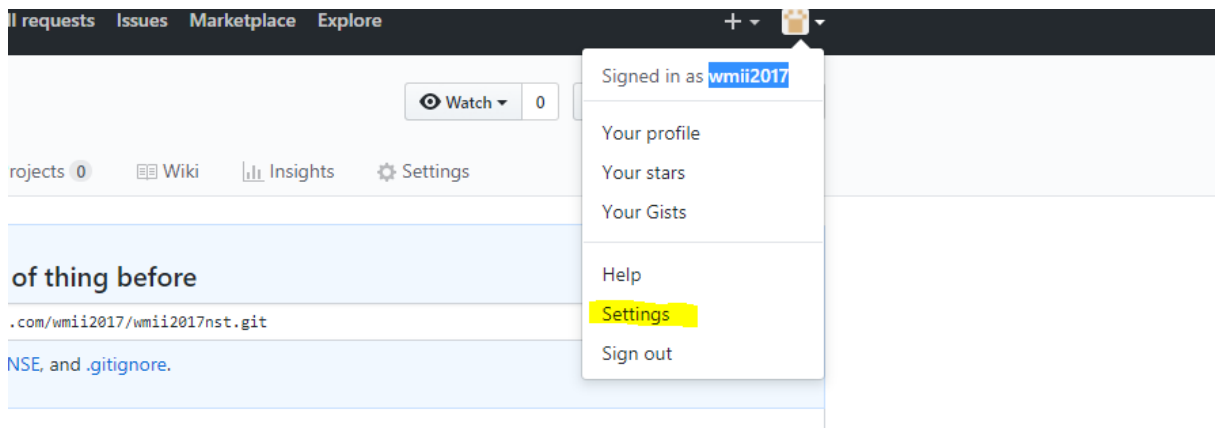
Otwieramy plik id\_rsa.pub za pomocą dowolnego edytora tekstu i kopiujemy jego zawartość.

W moim przypadku klucz ma następującą wartość :

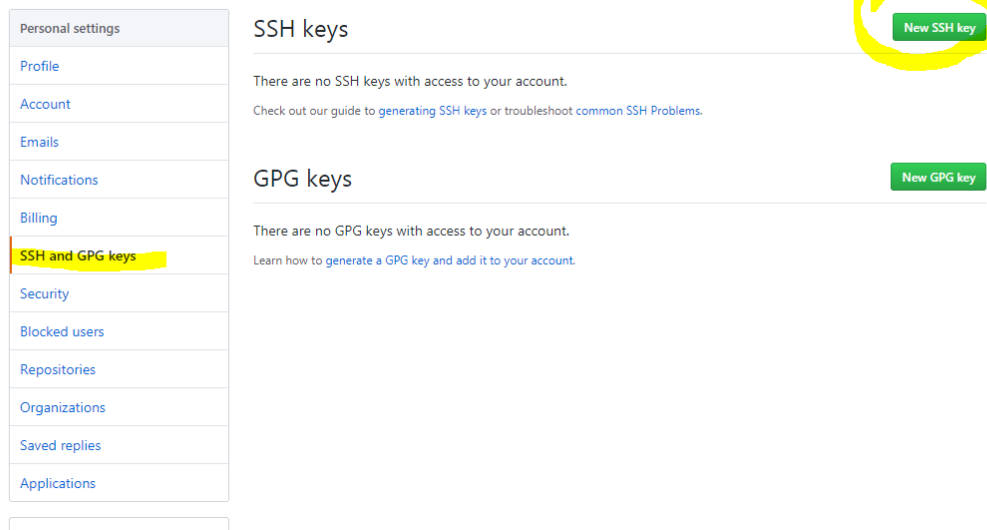
ssh-rsa

AAAAB3NzaC1yc2EAAAADAQABAAQCC7hbfDO1uuBhvKu4ymqyBZk4Xx0uD  
6EbDvUp+DWt47daMbc0tiK2daybB2fKWmVCRAJ2x1yyXGrd0FVIVvs2aHNpw5S  
giyxj5nuFHW3wpC3DpHSudx7SEsXBNhh/KA1/zRwd6Xm9C0OpBsPYi6A2mYrOx  
ZRh2eQoXsz0aITcKfjh2NWx2+aQZ7DMFHsYq2mvQpJnJHJPoKSLdAupUjVLICZQP  
bVrVx/0v4u16hzkwNMACKGmjxn7Fu2qAlNJb8Z+sZZMIH1IpsAG8fZMSnR+SIDB  
UPQqXKIzu01UWzGJNayCbF3yJsxOzs3zAvwtw8RwqAQRj93cQA5C96nQ7K1IX  
[wmii2017github@gmail.com](mailto:wmii2017github@gmail.com)

Wracamy na stronę github.com i wybieramy zakładkę Settings.



Wybieramy zakładkę SSH Key - > Add SSH key



Title

Moj kluczyk na wykład

Key

ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQCC7hbfDO1uuBhvKu4ymqyBZk4Xx0uD6EbDvUp+DWt47daMbc0tiK2  
daybB2fKWmVCRAJ2x1yyXGrd0FVIVvs2aHNpw5Sgiyxj5nuFW3wpC3DpHSudx7SEsXBNhh/KA1/zRwd6Xm9C0  
OpBsPYi6A2mYrOxZRh2eQoXsz0alTcKfjh2NWx2+aQZ7DMFHsYq2mvQpJnJHJPoKSLdAupUjVLICZQPbVrVx/0v4  
u16hzkwNMACKGmjxn7Fu2qAINJb8Z+sZZMIH1lpsAG8fZMSnR+SIDBUPQqXKIZu01UWzGjNayCbF3yJsxOzs3z  
Avwtw8RwqAQRj93cQA5C96nQ7K1IX wmii2017github@gmail.com  
|


Add SSH key

Tytuł uzupełniamy dowolnie , w Key wklejamy zawartość pliku id\_rsa.pub

## SSH keys

[New SSH key](#)

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



Moj kluczyk na wykład

Fingerprint: 3a:71:86:32:15:24:35:ce:b7:7c:6c:3c:c1:dd:d2:7c

Added on 17 Oct 2017

Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

Jeśli wszystko pójdzie prawidłowo otrzymamy powyższy widok.

Wracamy do konsoli GitBash i wywołujemy poniższe polecenia :

```
MINGW64:/c/Users/Sebastian/.ssh
Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-ODqmhkNDZyJK/agent.1976; export SSH_AUTH_SOCK;
SSH_AGENT_PID=5416; export SSH_AGENT_PID;
echo Agent pid 5416;

Sebastian@Sebastian: MINGW64 ~/.ssh
$ eval "$(ssh-agent)"
Agent pid 6684

Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-add ir_rsa
ir_rsa: No such file or directory

Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh-add id_rsa
Enter passphrase for id_rsa:
Identity added: id_rsa (id_rsa)

Sebastian@Sebastian: MINGW64 ~/.ssh
$ |
```

W folderze .ssh :

Ssh-agent

Eval "\$(ssh-agent)"

Ssh-add id\_rsa

Potwierdzamy hasłem.

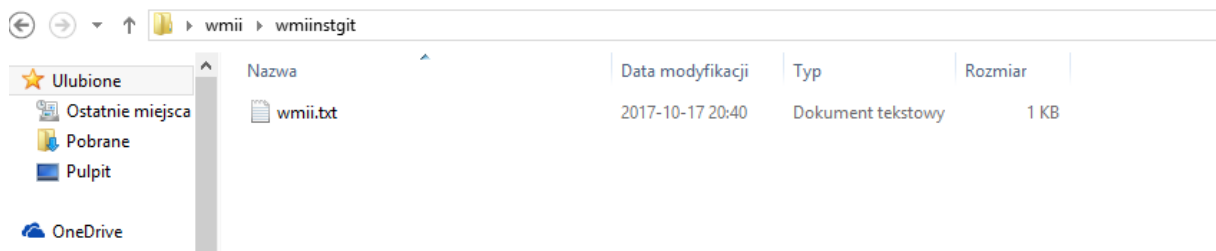
Ostatnim krokiem jest nasze uwierzytelnienie z GitHub.com

```
Sebastian@Sebastian: MINGW64 ~/.ssh
$ ssh -T git@github.com
The authenticity of host 'github.com (192.30.253.113)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'github.com,192.30.253.113' (RSA) to the list of known hosts.
Hi wmi2017! You've successfully authenticated, but GitHub does not provide shell access.

Sebastian@Sebastian: MINGW64 ~/.ssh
$ |
```

Wracamy do naszego repozytorium:

Tworzymy dowolny plik tekstowy:



Klikamy prawym i wybieramy Git Bash Here.

```
Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git init
Reinitialized existing Git repository in C:/Users/Sebastian/Desktop/wmii/wmiinstgit/.git/

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git add .

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git commit -m "Moj pierwszy commit"
[master (root-commit) 5ac33a6] Moj pierwszy commit
1 file changed, 1 insertion(+)
create mode 100644 wmii.txt

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git push -u origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git remote add origin https://github.com/wmii2017/wmii2017nst.git

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 227 bytes | 113.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/wmii2017/wmii2017nst.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

Sebastian@MINGW64 ~/Desktop/wmii/wmiinstgit (master)
$ |
```

Po czym wywołujemy polecenia :

Git init

Git add.

## Git commit -m "Mój pierwszy commit"

git remote add origin Link\_do\_naszego\_repozytorium ->

Quick setup — if you've done this kind of thing before

 Set up in Desktop or 

HTTPS SSH `https://github.com/wmii2017/xd214.git`



We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# xd214" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/wmii2017/xd214.git
git push -u origin master
```



...or push an existing repository from the command line

```
git remote add origin https://github.com/wmii2017/xd214.git
git push -u origin master
```




...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Git push -u origin master

Po wejściu na GitHub.com i wybraniu naszego repozytorium otrzymamy następujący widok :

No description, website, or topics provided. 

[Add topics](#)

1 commit

1 branch

0 releases

1 contributor

Branch: master ▾


New pull request


Create new file


Upload files

Find file

Clone or download ▾

 `wmii2017` `Mój pierwszy commit` Latest commit 5ac33a6 7 minutes ago

 `wmii.txt` `Mój pierwszy commit` 7 minutes ago

Help people interested in this repository understand your project by adding a README. 



## 5.Git : Najważniejsze komendy – źródło : [blog.piotrnalepa.pl](http://blog.piotrnalepa.pl)

**git init** - Inicjalizuje repozytorium GIT w danym katalogu

**git add [nazwa\_pliku]** - Dodaje zmiany we wskazanym pliku do commita

**git add .** - Dodaje wszystkie zmienione pliki do commita

**git add -p [nazwa\_pliku]** - Udostępnia możliwość dodania wybranych linii w zmodyfikowanym pliku do commita

**git commit -m "[treść\_commita]"** Dodaje opis do commita. Dobrym zwyczajem jest opisanie co ta zmiana wprowadza do kodu w zakresie funkcjonalnym

**git add origin [adres\_repozytorium, np. <https://github.com/username/moje-repozytorium.git>]** Ustawia konkretny adres zdalnego repozytorium jako główne repozytorium

**git push origin master** - Wysłanie zmian do brancha zdalnego

**git push -f** - Wysłanie zmian do zdalnego repozytorium ignorując konflikty, to znaczy, że jeśli wystąpią konflikty to pliki zostaną nadpisane właśnie wysłaną wersją. Trzeba stosować to bardzo ostrożnie.

**git checkout [nazwa\_brancha]** - Zmienia aktywny branch na wybrany przez użytkownika

**git checkout [nazwa\_pliku]** - Usuwa zmiany w wybranym pliku

**git checkout .** - Usuwa zmiany we wszystkich zmienionych plikach

**git checkout -b [nazwa\_brancha]** Tworzenie nowego brancha z aktywnego brancha i przełączenie się na niego

**git rebase master** - Zaciągnięcie zmian z brancha głównego do brancha aktywnego

**git push origin :[nazwa\_brancha]** - Usunięcie zdalnego brancha

**git branch -d [nazwa\_brancha]** - Usuwanie brancha lokalnie. Nie można usunąć w ten sposób aktywnego brancha

**git stash** - Dodanie zmienionych plików do pamięci/stosu i usunięcie ich z aktywnego brancha

**git pull --rebase** - Pobranie najnowszych zmian z aktywnego brancha zdalnego

**git stash pop** - Przywrócenie zmodyfikowanych plików z pamięci/stosu

**git stash clear** - Czyszczenie pamięci/stosu

**git remote prune origin** - Pobranie aktualizacji o usuniętych branchach zdalnych

**git fetch --all** - Pobranie listy zdalnych branchy

**git branch** - Wyświetlenie listy lokalnych branchy

**git branch -r** - Wyświetlenie listy zdalnych branchy

**git status** - Wyświetlenie listy zmienionych plików

**git diff [nazwa\_pliku]** - Szczegółowe wyświetlenie zmian w wybranym pliku

**git reset HEAD** - Resetowanie przygotowanych commitów (przed wysłaniem). Zmodyfikowane pliki są dostępne do ponownego dodania.

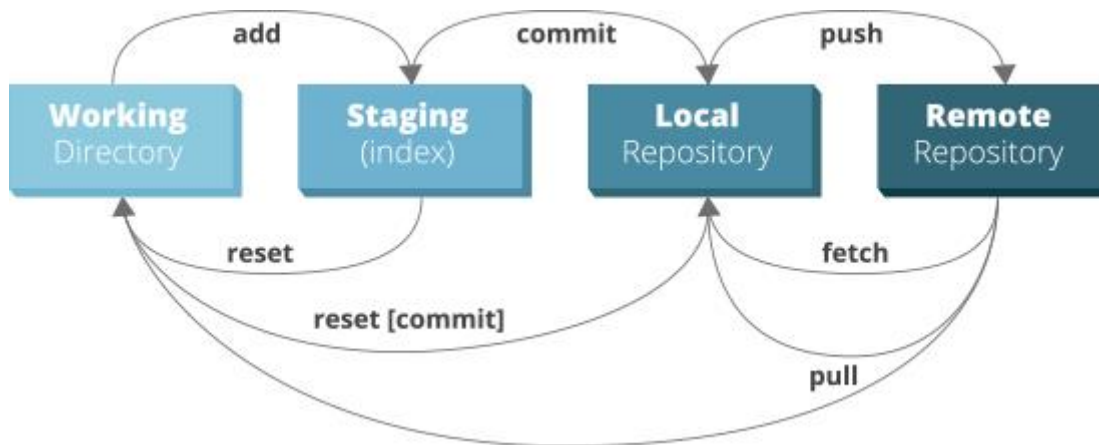
**git reset HEAD --hard** - usuwanie wszystkich zmian z brancha lokalnego i przywrócenie zmian z brancha zdalnego

**git reset HEAD^ --hard** - Usuwanie ostatniego commita z brancha

**git reset HEAD^^**

**git reset HEAD~2** - Obydwie komendy usuwają ostatnie 2 zmiany z brancha. Im więcej daszków (^) tym więcej commitów zostanie usuniętych.

**git rebase -i HEAD~3** - Interaktywne zmienianie zawartości, opisów commitów. Commity można łączyć wtedy w jeden duży, zmienić jego opis, itd.



## 6.GIT – ignorowanie plików.

Często spotkasz się z klasą plików, w przypadku których nie chcesz, by Git automatycznie dodawał je do repozytorium, czy nawet pokazywał je jako nieśledzone. Są to ogólnie pliki generowane automatycznie, takie jak dzienniki zdarzeń, czy pliki tworzone w czasie budowania projektu. W takich wypadkach stworzysz plik zawierający listę wzorców do nich pasujących i nazywasz go `.gitignore`.

Źródła :

Wikipedia.org

GitHub.com

git-scm.com/