Autor : Sebastian Waśkiewicz

Rozproszony system kontroli wersji

Spis treści

6. GIT – ignorowanie plików.

1.Czym jest Git?

Git jest rozporoszonym 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

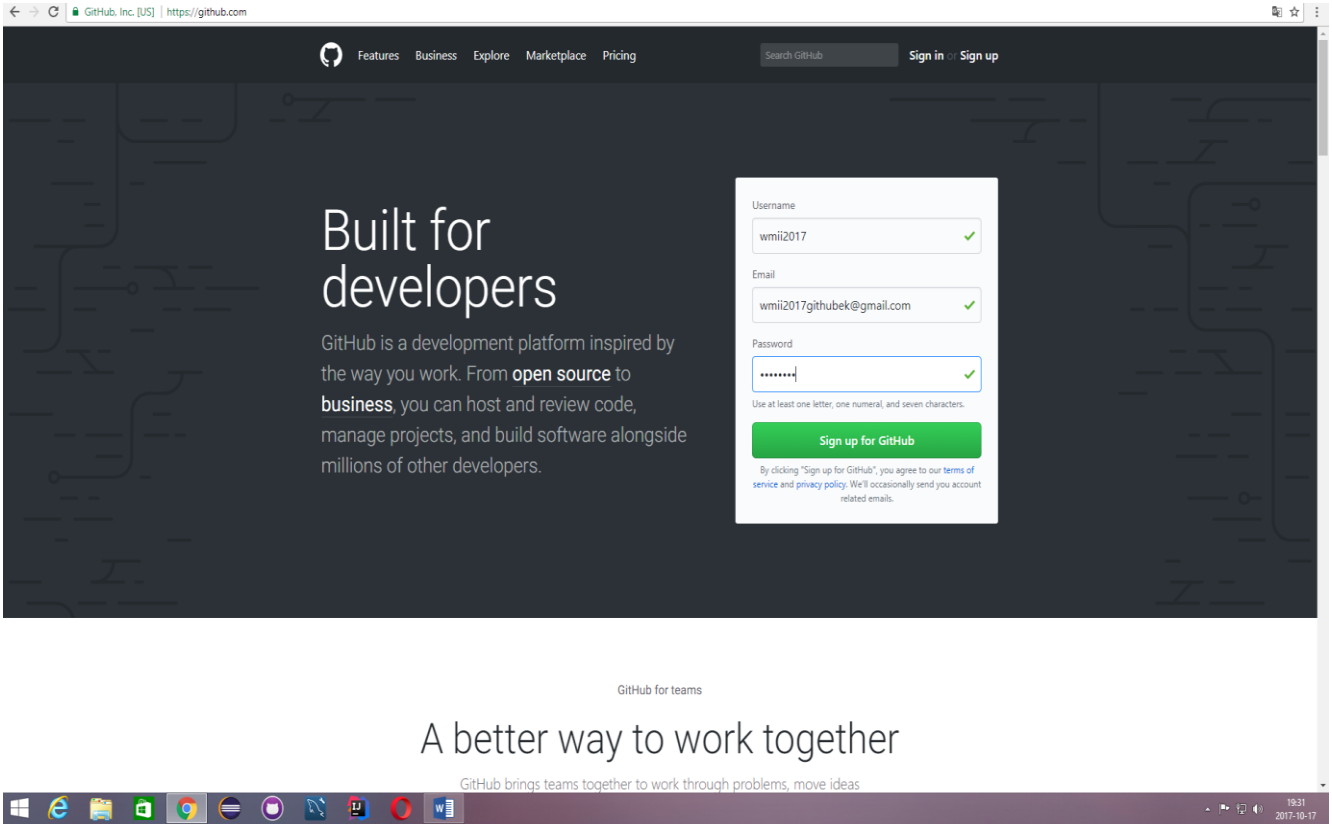
GitLab

Bitbucket

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

3. github.com

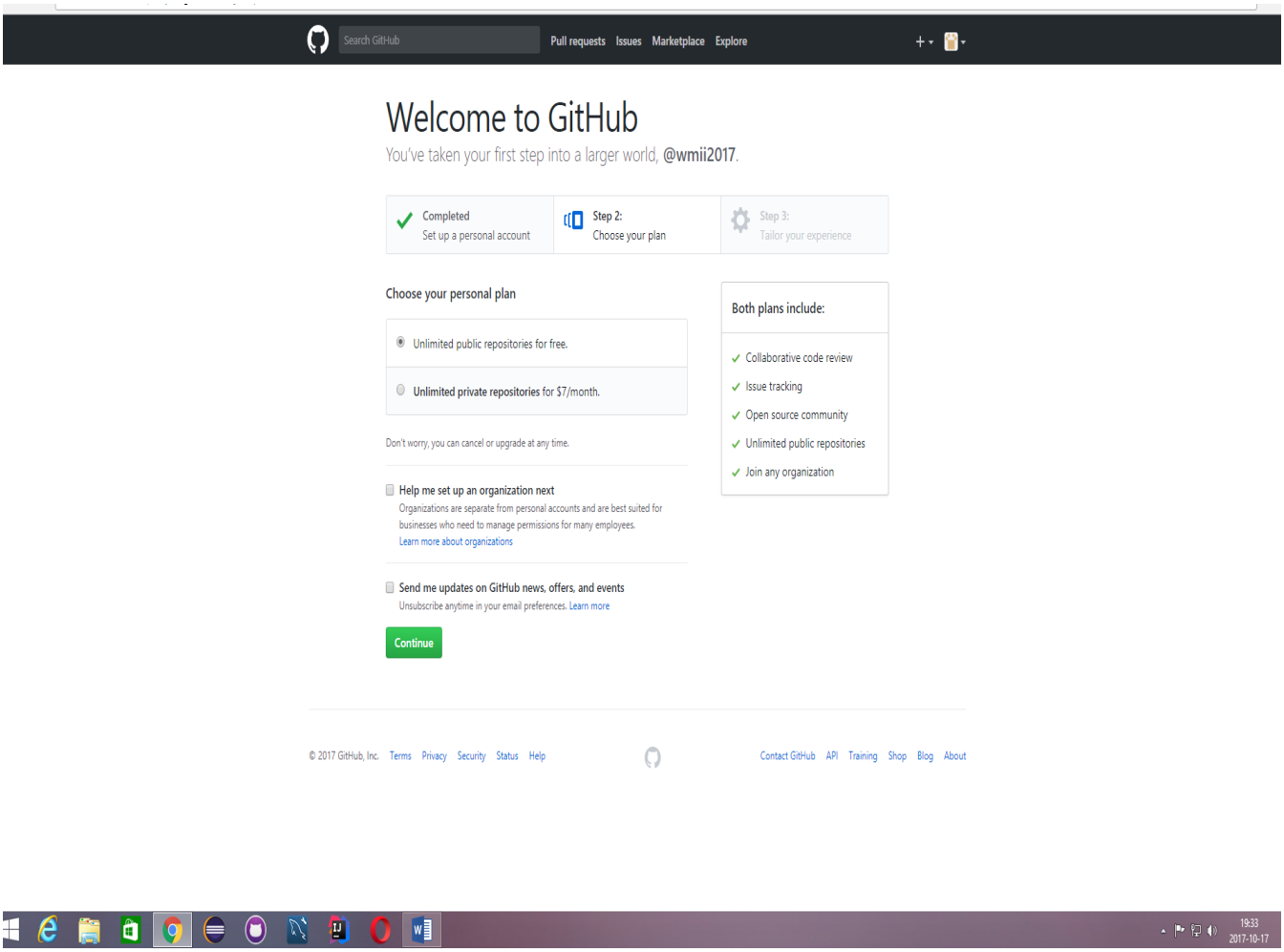
3.1) Zakładanie konta



wmii2017

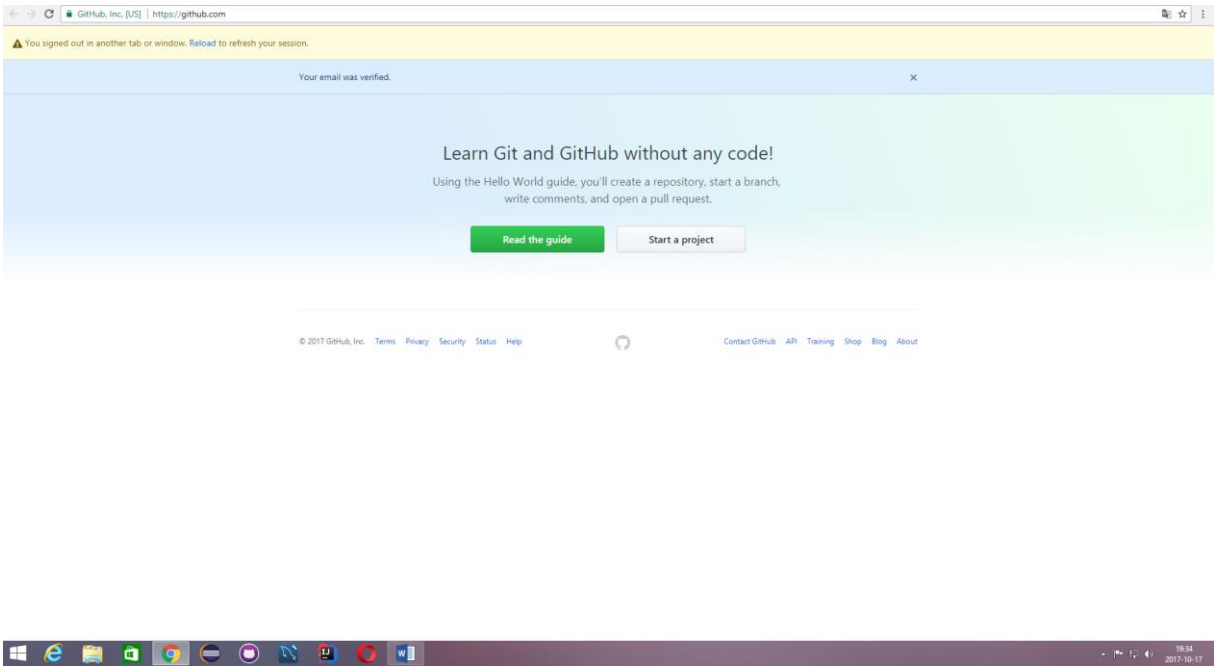
wmii2017githubek@gmail.com

<u>Uzupełniamy pola i klikamy w zielony przycisk.</u>



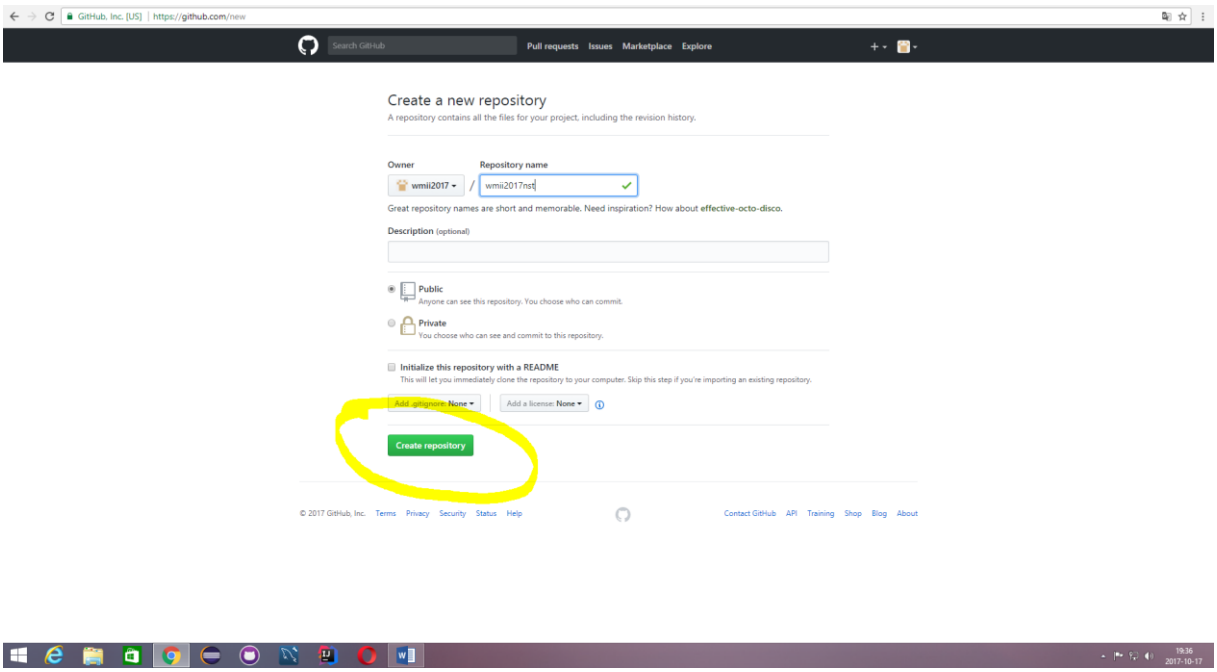
Otrzymujemy taki ekran.

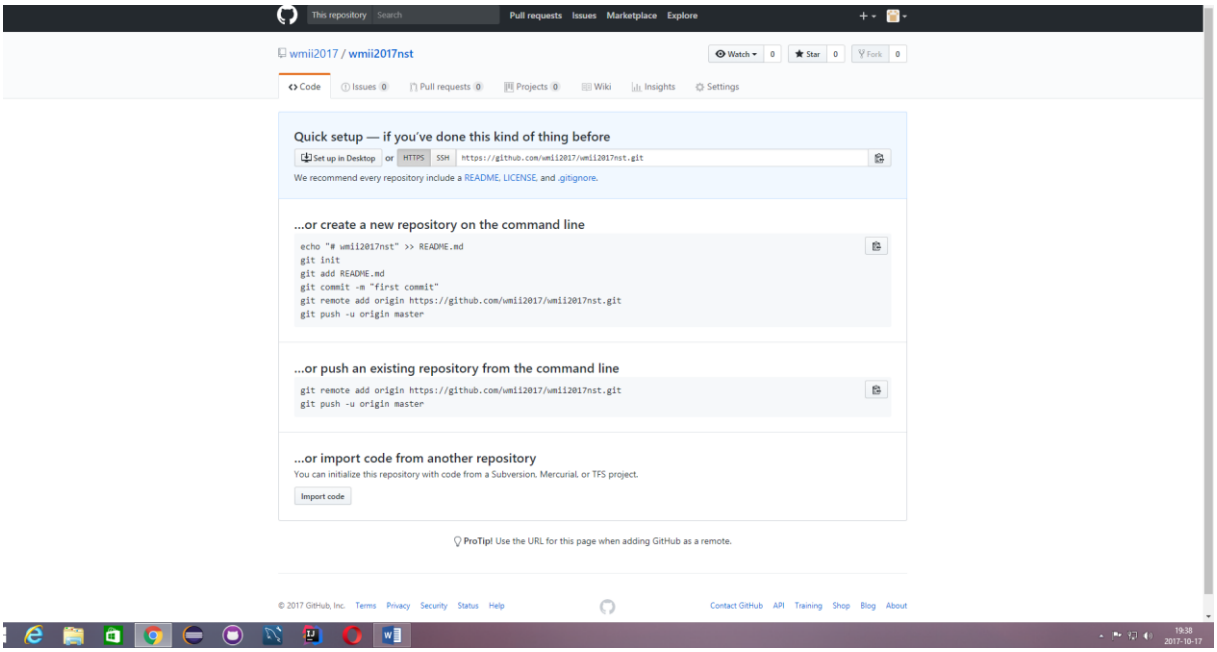
W opcji Choose your personal plan wybieramy Darmowe publiczne repozytorium i idziemy dalej.



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 ☺

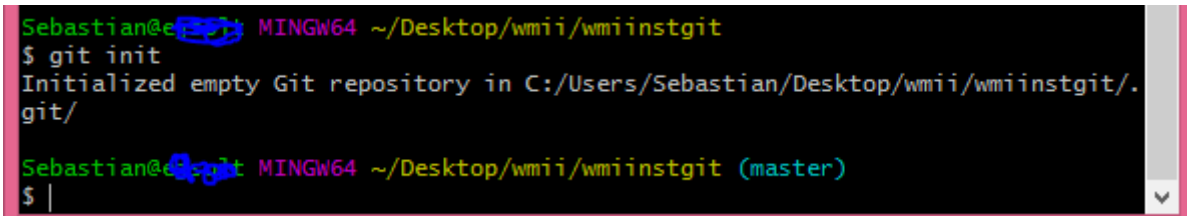
5.Tworzenie repozytorium

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 „wnętrznościach” ☺

Z listy wybieramy opcję Git Bash Here.

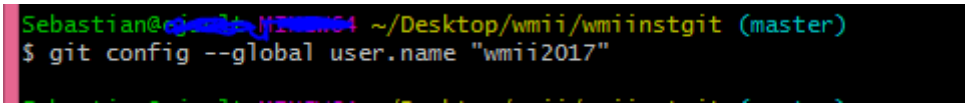
Otrzymaliśmy konsolę podobną do tej która jest domyślną systemową Windowsa.



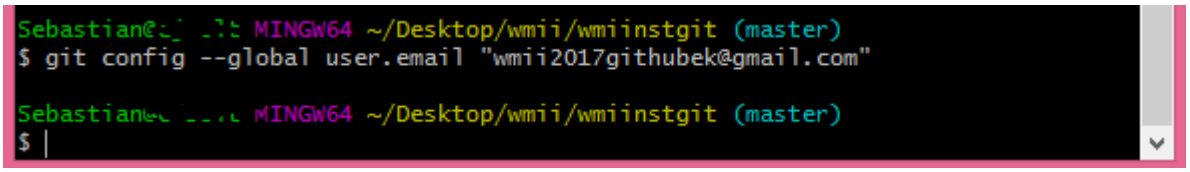
Wpisujemy git init – To polecenie stworzy nam nowy podkatalog zawierający wszystkie niezbędne pliki – szkielet repozytorium.

Następnie konfigurujemy ustawienia użytkownika :

Git config –global user.name „wmii2017”



Git config –global user.email wmii2017githubek@gmail.com



Ponownie wpisujemy komendę git init

Teraz musimy utworzyć klucz SSH.

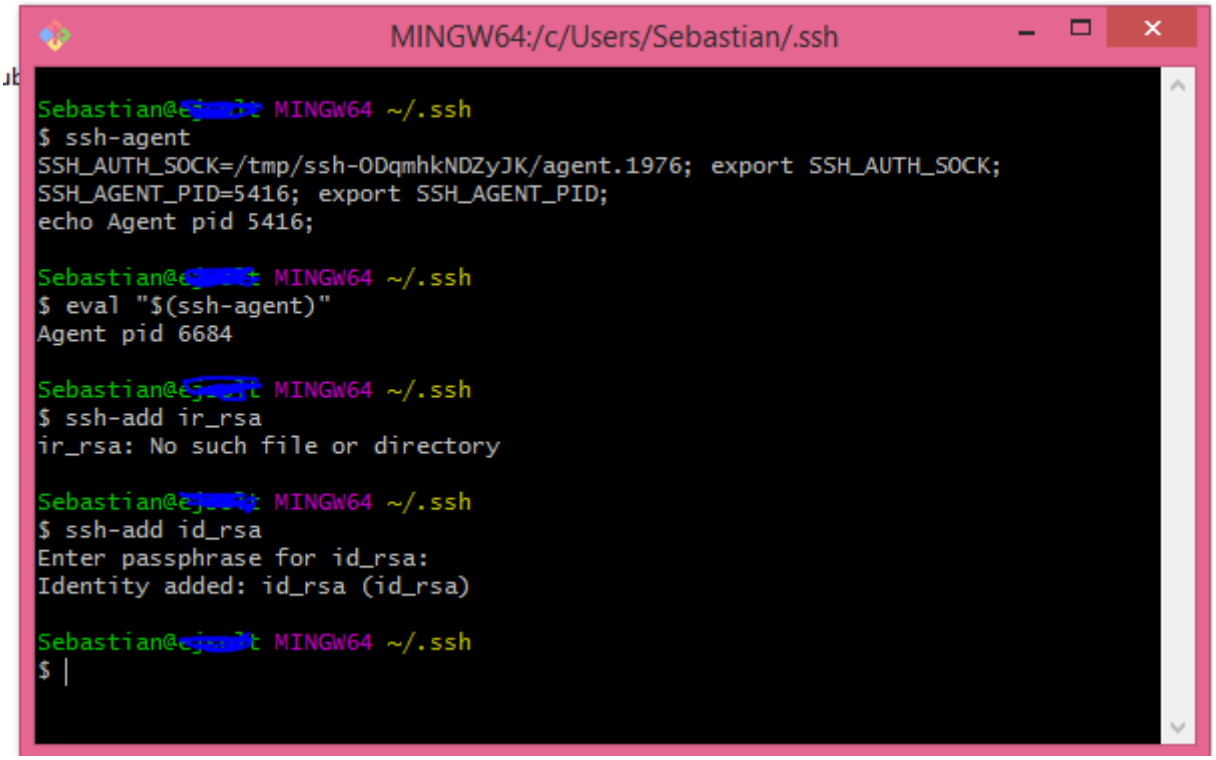
Czym one są?

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.

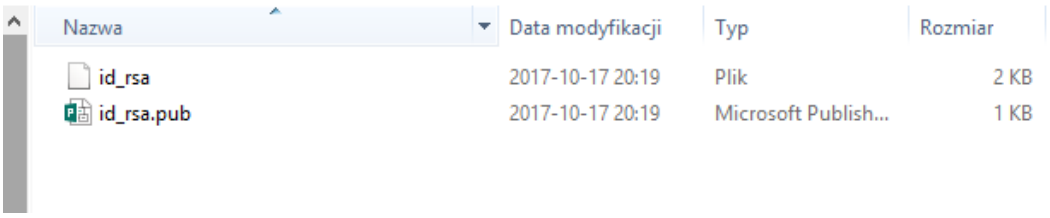
Wchodzimy w C:\Users\nazwa_uzytkownika i klikamy ponownie Git Bash Here.

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

Klikamy enter i potwierdzamy dwa razy naszym hasłem.



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

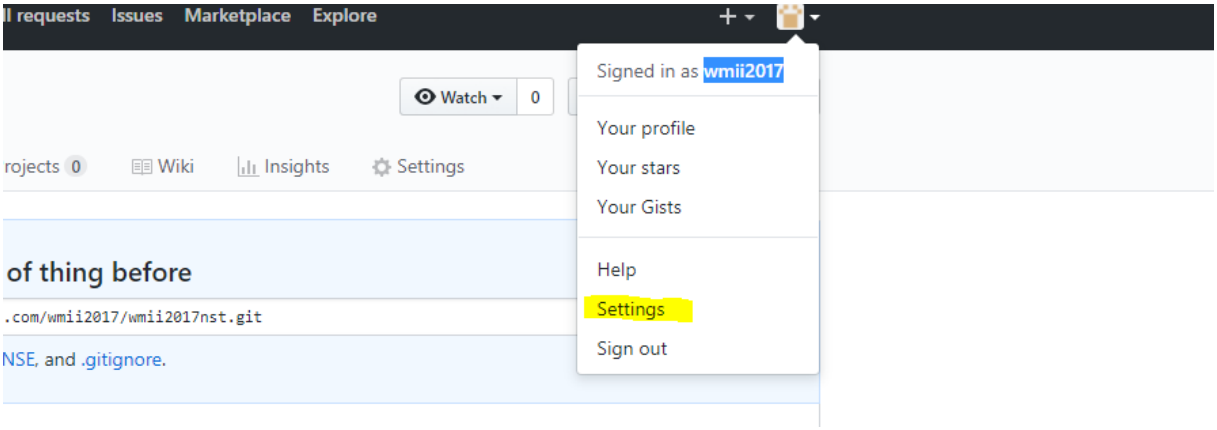


Otwieramy plik id_rsa.pub za pomocą dowolnego edytora tekstu i kopiujemy jego zawartość.

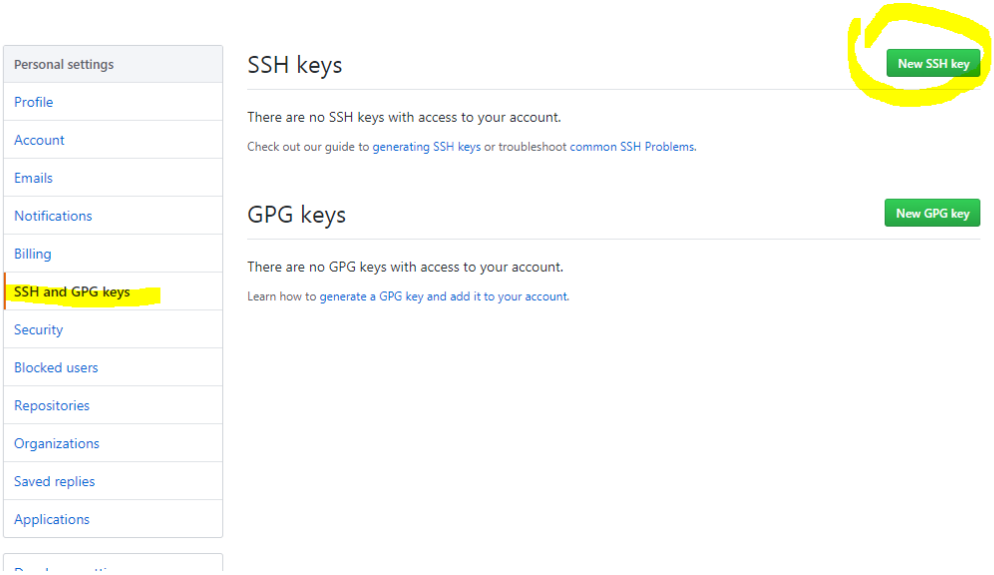
W moim przypadku klucz ma następująca wartość :

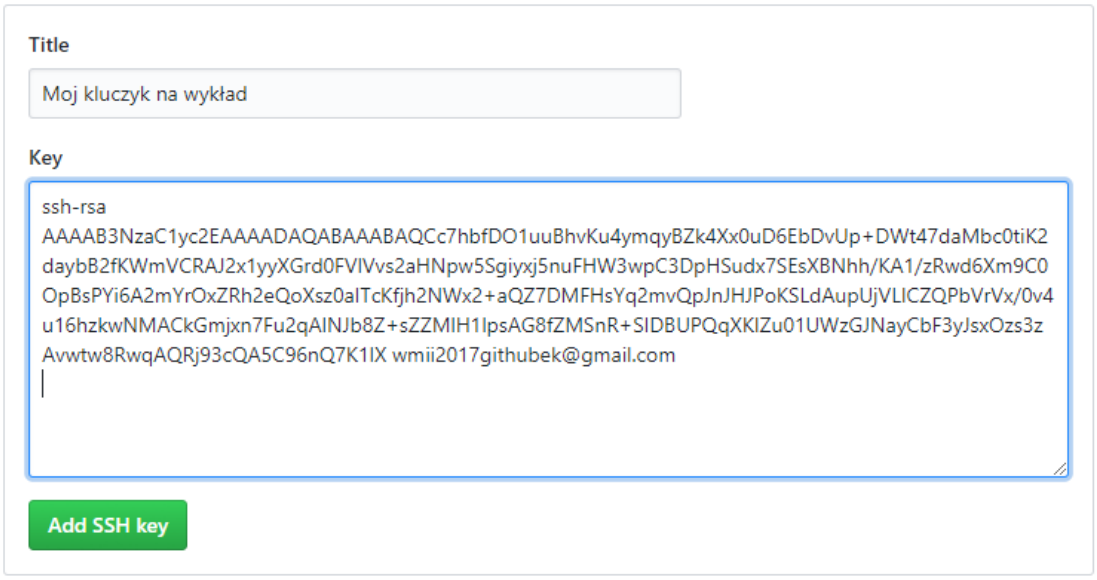
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCc7hbfDO1uuBhvKu4ymqyBZk4Xx0uD
6EbDvUp+DWt47daMbc0tiK2daybB2fKWmVCRAJ2x1yyXGrd0FVlVvs2aHNpw5S
giyxj5nuFHW3wpC3DpHSudx7SEsXBNhh/KA1/zRwd6Xm9C0OpBsPYi6A2mYrOx
ZRh2eQoXsz0aITcKfjh2NWx2+aQZ7DMFHsYq2mvQpJnJHJPoKSLdAupUjVLICZQP
bVrVx/0v4u16hzkwNMACkGmjxn7Fu2qAlNJb8Z+sZZMlH1lpsAG8fZMSnR+SIDB
UPQqXKIZu01UWzGJNayCbF3yJsxOzs3zAvwtw8RwqAQRj93cQA5C96nQ7K1lX
wmii2017githubek@gmail.com

Wracamy na stronę github.com i wybieramy zakładkę Settings.

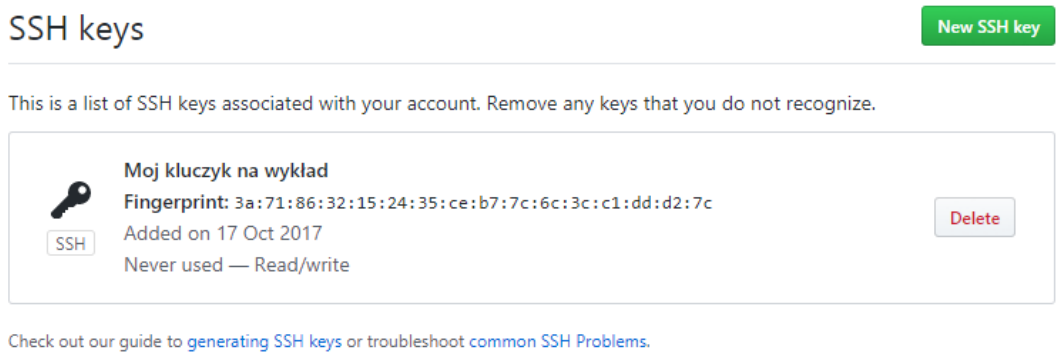


Wybieramy zakładkę SSH Key - > Add SSH key



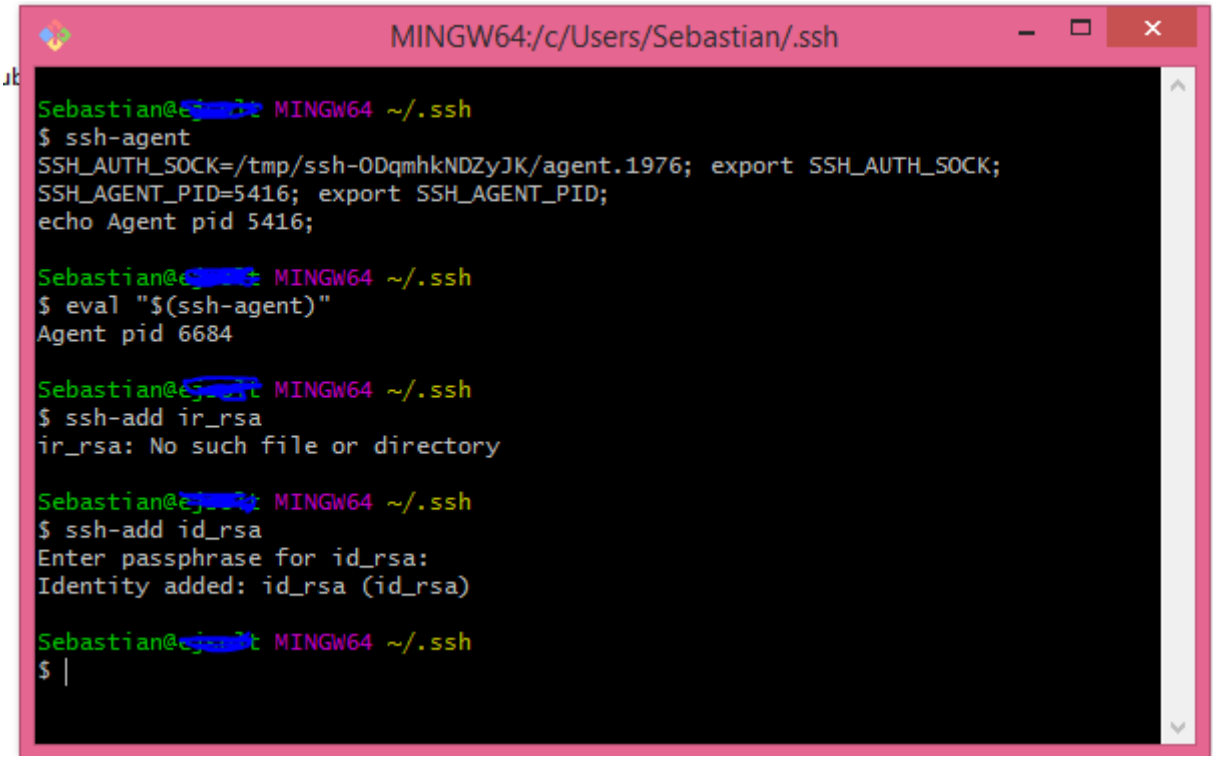


Tytuł uzupełniamy dowolnie , w Key wklejamy zawartość pliku id_rsa.pub



Jeśli wszystko pójdzie prawidłowo otrzymamy powyższy widok.

Wracamy do konsoli GitBash i wywołujemy poniższe polecenia :



W folderze .ssh :

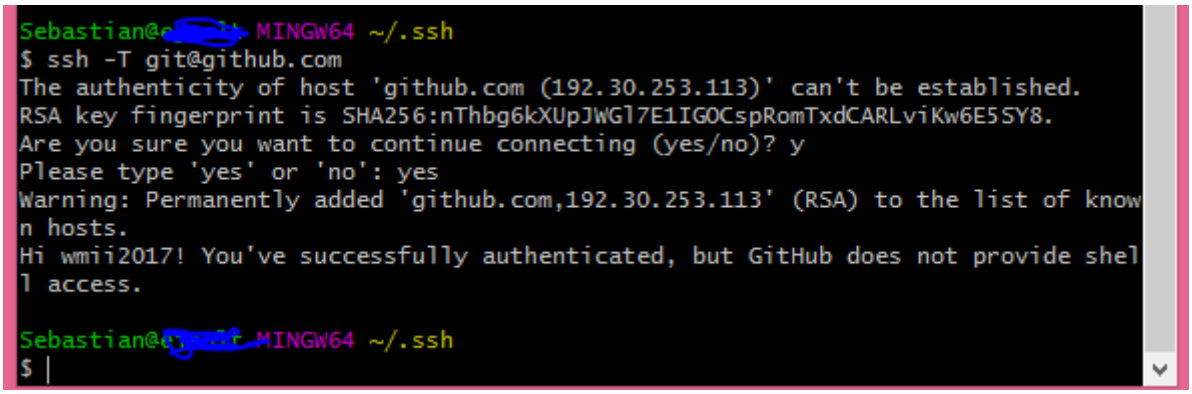
Ssh-agent

Eval ”$(ssh-agent)”

Ssh-add id_rsa

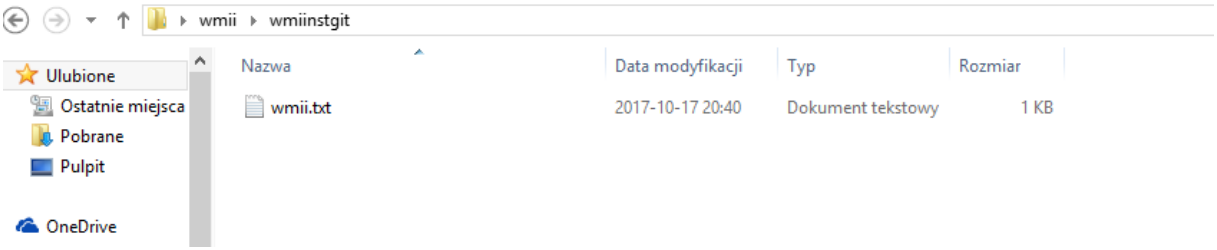
Potwierdzamy hasłem.

Ostatnim krokiem jest nasze uwierzytelnienie z GitHub.com

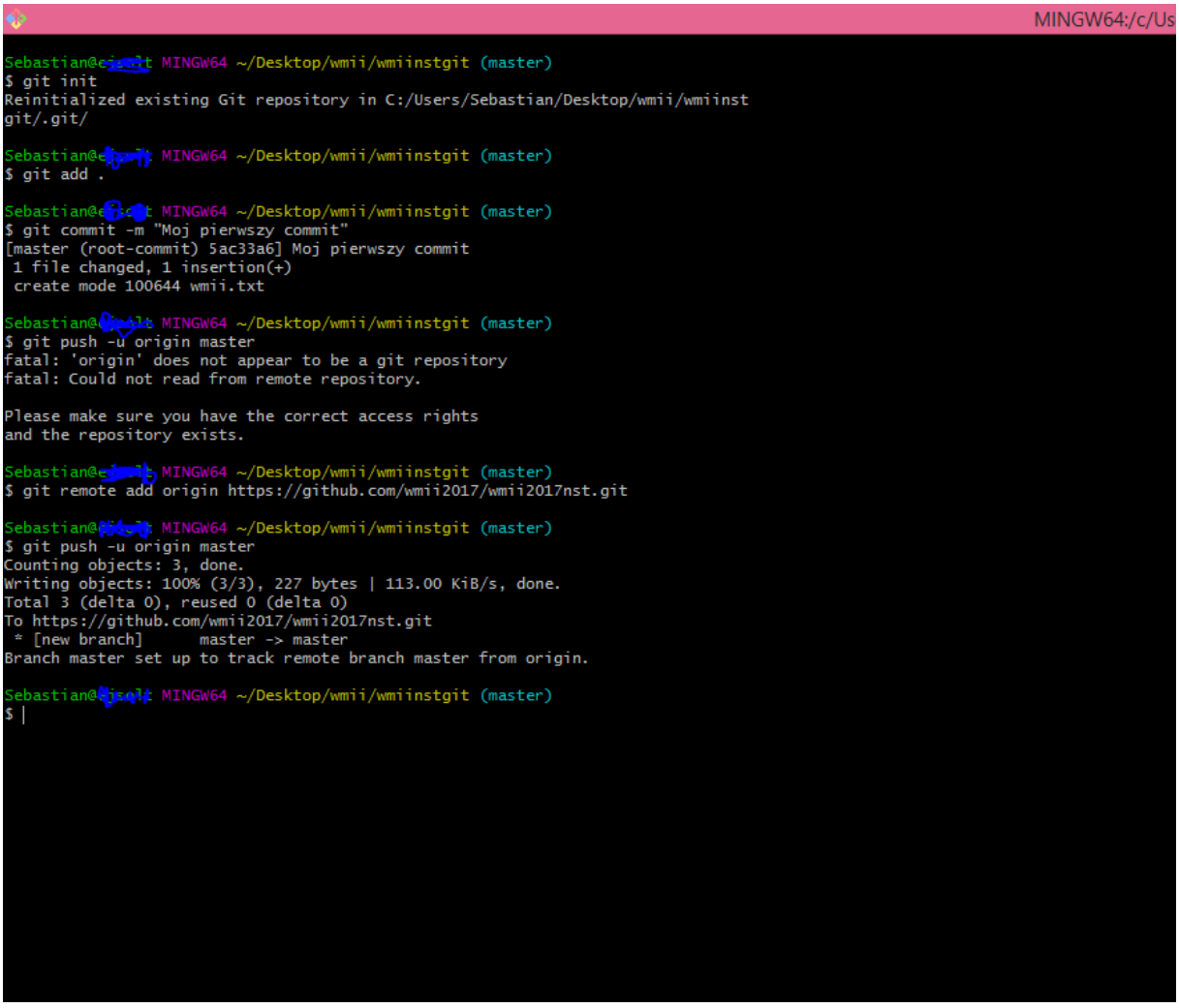


Wracamy do naszego repozytorium:

Tworzymy dowolny plik tekstowy:



Klikamy prawym i wybieramy Git Bash Here.



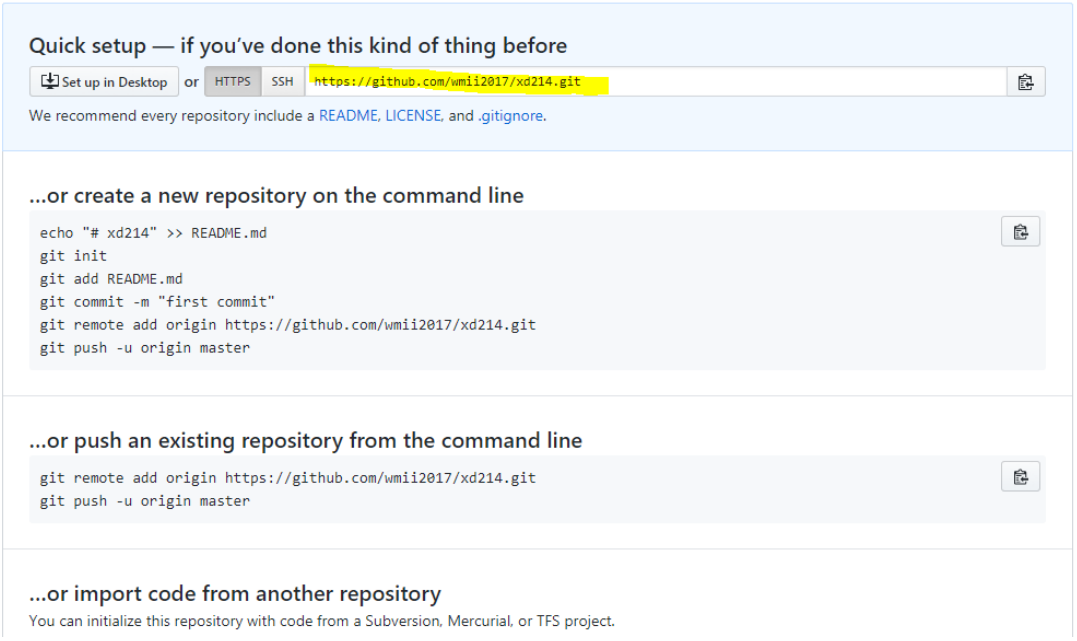
Po czym wywołujemy polecania :

Git init

Git add.

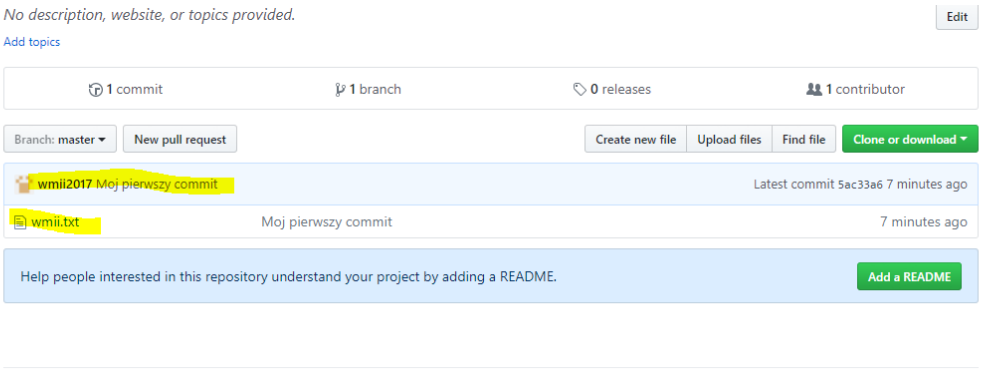
Git commit -m ”Mój pierwszy commit”

`git remote add origin Link_do_naszego_repozytorium ->`



Git push -u origin master

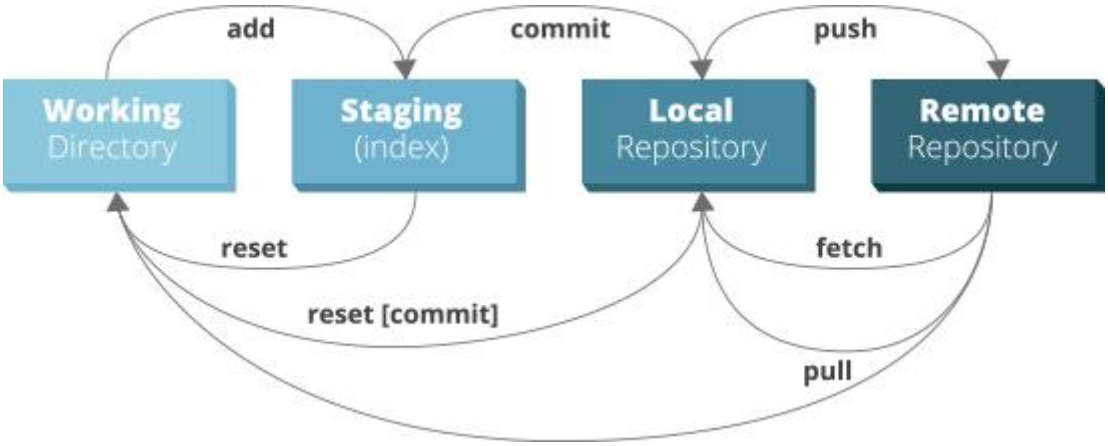
Po wejściu na GitHub.com i wybraniu naszego repozytorium otrzymamy następujący widok :



5.Git : Najważniejsze komendy – zrodlo : 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 branacha zdalnego
git push -f - Wysłanie zmian do zdalnego repozytorium ignorując konflikty, to znaczy, że jeśli wystapią 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 zmienonych 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 mozna 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 tworzysz plik zawierający listę wzorców do nich pasujących i nazywasz go `.gitignore`.

Źródła :

Wikipedia.org

GitHub.com

git-scm.com/