

Wprowadzenie



git



Git & GitHub Workshops

ZSE Kielce 27.11.2018

Pawel Osobinski @accenture.com

accenture > operations

Ktoś z was zna te marki ?

poniedziałek, 26 listopada 2018

22:13



Czym jest git?



Git – rozproszony system kontroli wersji. Stworzył go Linus Torvalds jako narzędzie wspomagające rozwój jądra Linux. Git stanowi wolne oprogramowanie i został opublikowany na licencji GNU GPL w wersji 2. Pierwsza wersja narzędzia Git została wydana 7 kwietnia 2005 roku, by zastąpić poprzednio używany w rozwoju Linuksa, niebędący wolnym oprogramowaniem, system kontroli wersji BitKeeper.

Najważniejsze cechy

Dobre wsparcie dla rozgałęzionego procesu tworzenia oprogramowania: jest dostępnych kilka algorytmów łączenia zmian z dwóch gałęzi, a także możliwość dodawania własnych algorytmów.

Praca off-line: każdy programista posiada własną kopię repozytorium, do której może zapisywać zmiany bez połączenia z siecią; następnie zmiany mogą być wymieniane między lokalnymi repozytoriami.

Wsparcie dla istniejących protokołów sieciowych: dane można wymieniać przez HTTP(S), FTP, rsync, SSH.

Efektywna praca z dużymi projektami: system Git według zapewnień Torvaldsa, a także według testów fundacji Mozilla, jest o rzędy wielkości szybszy niż niektóre konkurencyjne rozwiązania.

Każda rewizja to obraz całego projektu: w przeciwieństwie do innych systemów kontroli wersji, Git nie zapamiętuje zmian między kolejnymi rewizjami, lecz kompletne obrazy. Z jednej strony wymaga to nieco więcej pracy aby porównać dwie rewizje, z drugiej jednak pozwala np. na automatyczną obsługę zmian nazw plików.

[https://pl.wikipedia.org/wiki/Git_\(oprogramowanie\)](https://pl.wikipedia.org/wiki/Git_(oprogramowanie))

GIT 101

Instalacja

Windows:

<https://git-scm.com/download/win>

(jest wersja portable)

Linux:

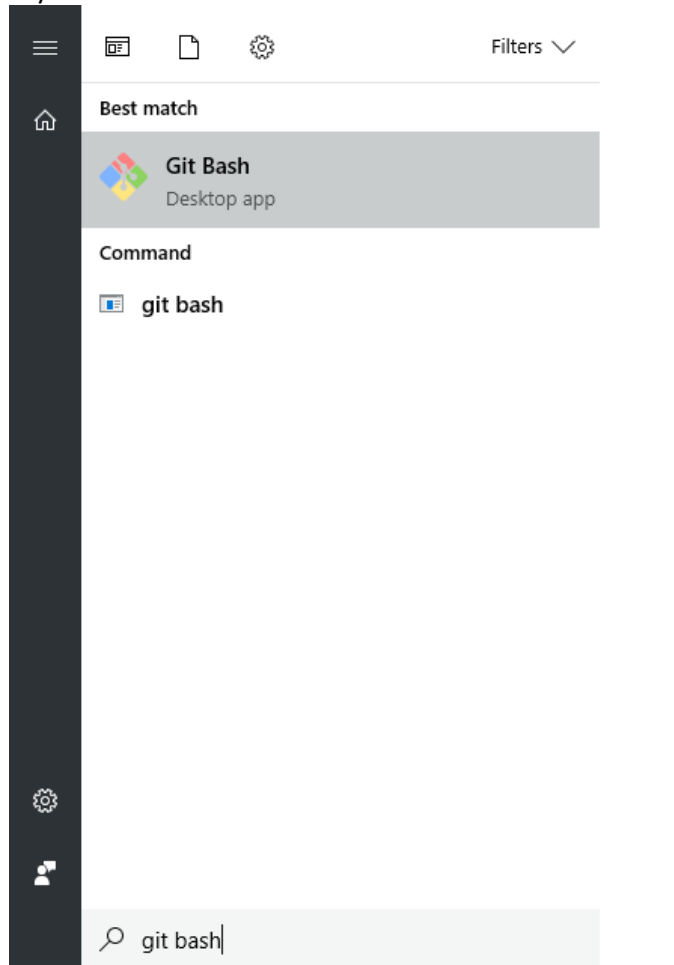
<https://git-scm.com/download/linux>

Wymagana podstawowa znajomość Linuksa (komendy z poziomi konsoli)

Prezentacja w formie dokumentu PDF

Podstawowa konfiguracja

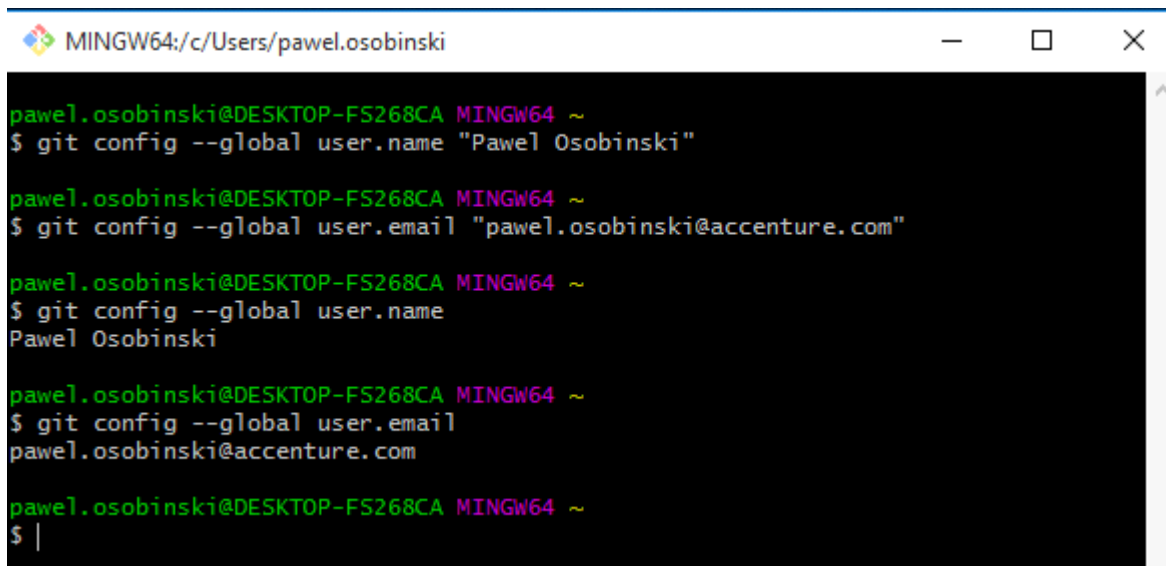
Wybierz Git Bash z menu start



Po uruchomieniu konsoli ustaw nazwę użytkownika oraz email (łatwiej będzie określić, kto wprowadził zmiany)

```
$ git config --global user.name "Imie Nazwisko"
```

```
$ git config --global user.email "(twoj_email)@ezse.pl"
```

A screenshot of a Windows terminal window titled "MINGW64:/c/Users/pawel.osobinski". The terminal shows a series of git configuration commands and their outputs. The prompt is "pawel.osobinski@DESKTOP-F5268CA MINGW64 ~". The commands and outputs are: 1. "\$ git config --global user.name 'Paweł Osobinski'" followed by a blank line. 2. "\$ git config --global user.email 'pawel.osobinski@accenture.com'" followed by a blank line. 3. "\$ git config --global user.name" followed by the output "Paweł Osobinski". 4. "\$ git config --global user.email" followed by the output "pawel.osobinski@accenture.com". 5. The prompt "\$ |" is shown at the end.

```
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~  
$ git config --global user.name "Paweł Osobinski"  
  
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~  
$ git config --global user.email "pawel.osobinski@accenture.com"  
  
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~  
$ git config --global user.name  
Paweł Osobinski  
  
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~  
$ git config --global user.email  
pawel.osobinski@accenture.com  
  
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~  
$ |
```

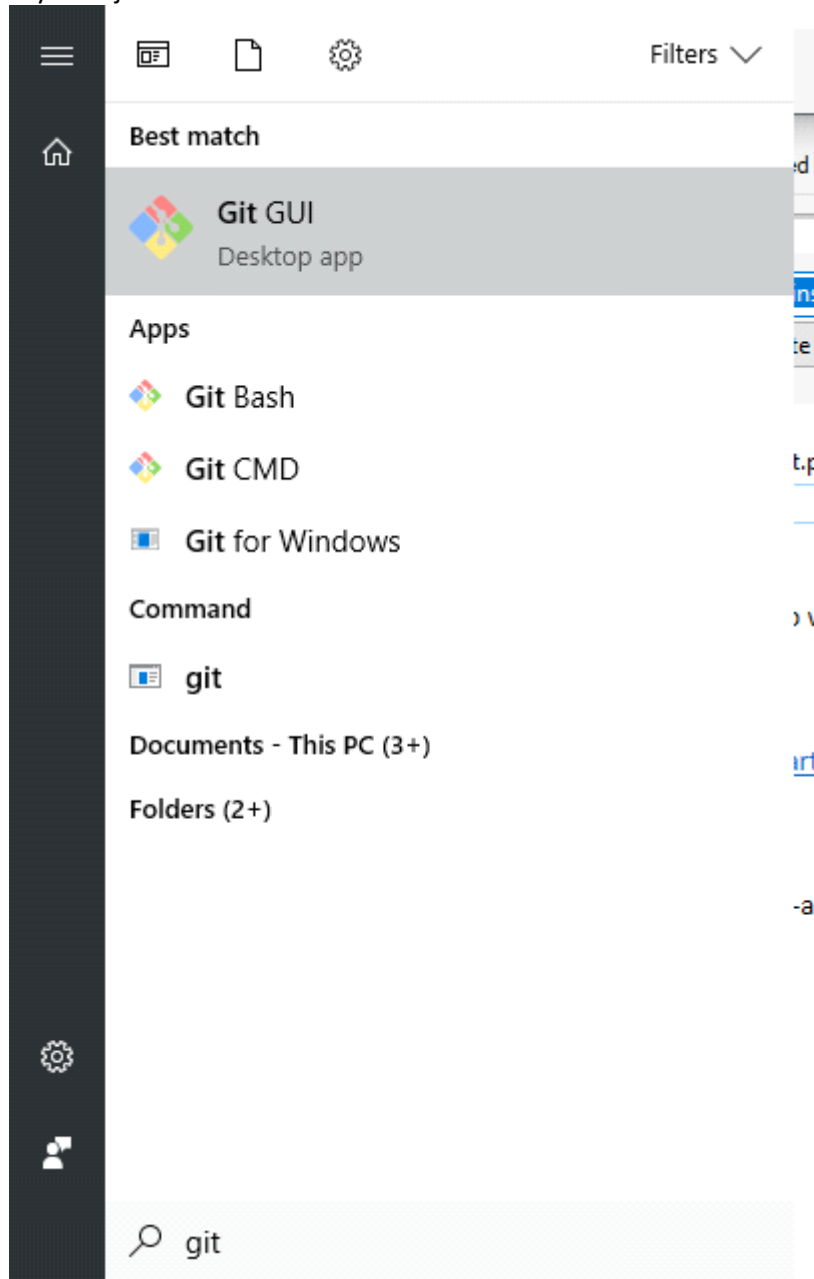
<https://help.github.com/articles/setting-your-commit-email-address-in-git/>
<https://help.github.com/articles/setting-your-username-in-git/>

Generowanie klucza SSH

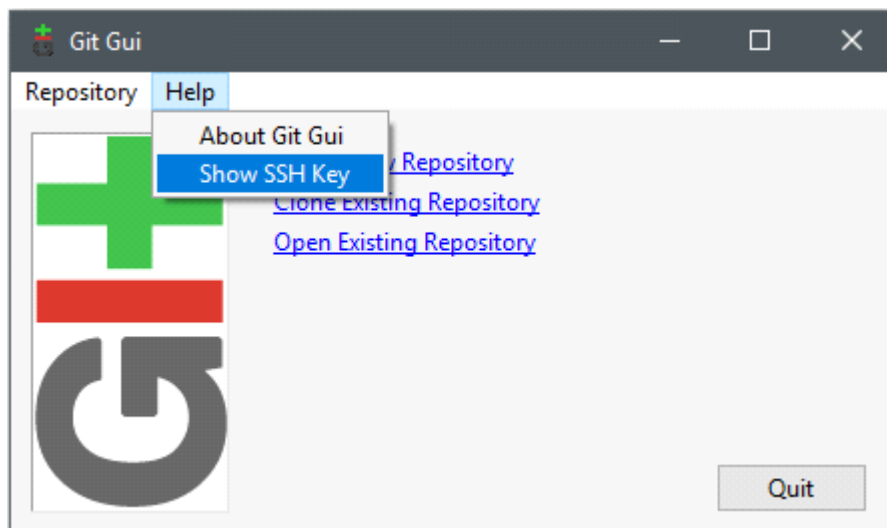
Metoda 1:

Generowania klucza z GIT-a

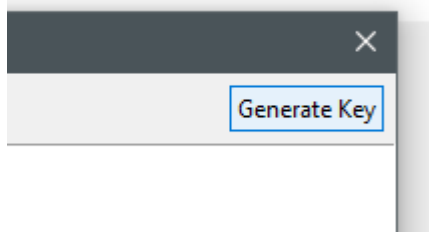
Wyszukaj Git GUI



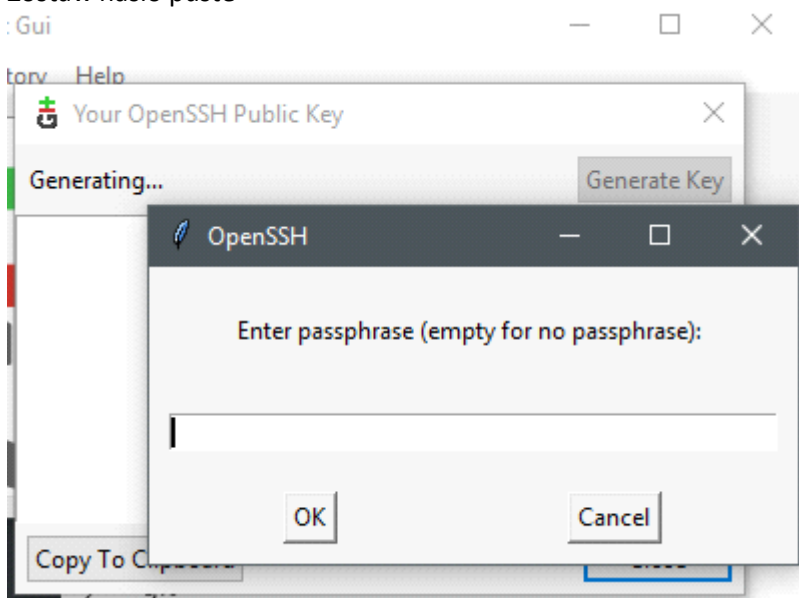
Z opcji wybierz Help -> Show SSH Key



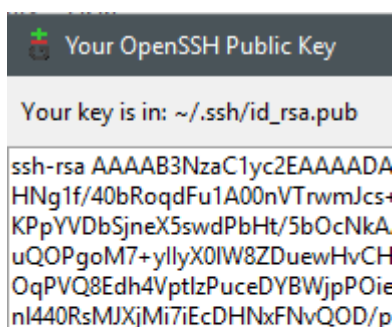
Wygeneruj nowy klucz

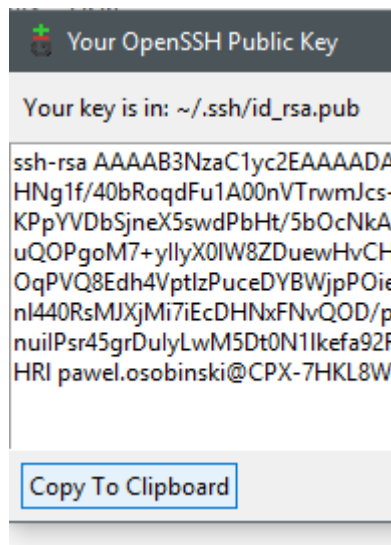


Zostaw hasło puste



Nowy klucz publiczny powinien pojawić się w oknie, skopiuj go do schowka



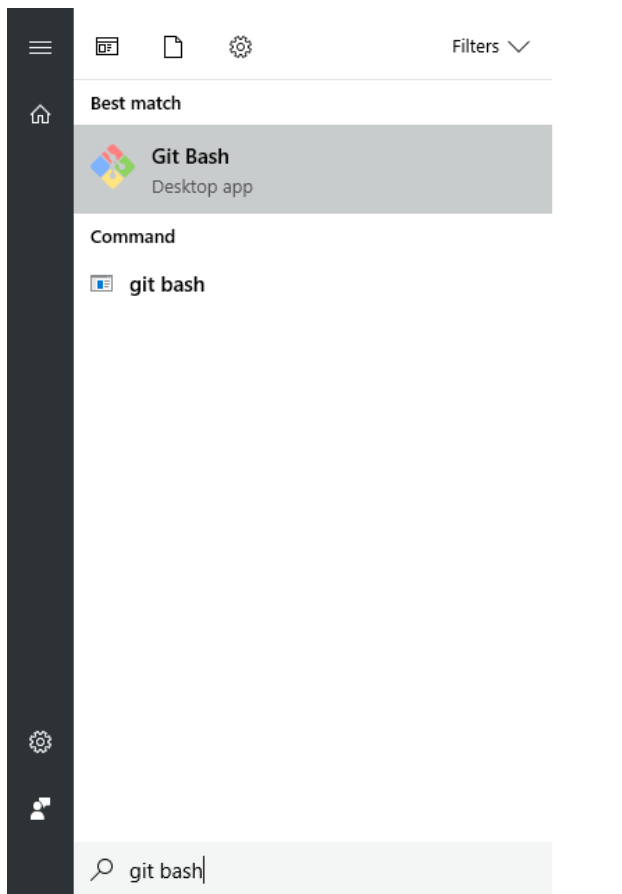


Lokalizacja plików do autoryzacji w systemie:
C:/Users/(twoj uzytkownik)/.ssh

Z poziomu git bash:
/c/Users/(twoj uzytkownik)/.ssh

id_rsa (klucz prywatny)
id_rsa.pub (klucz publiczny)

Metoda 2:
Wybierz Git Bash z menu start



Wpisz ssh-keygen w konsoli, następnie potwierdzaj kolejne elementy (enter)

```
MINGW64:/c/Users/pawel.osobinski
pawel.osobinski@DESKTOP-FS268CA MINGW64 ~
$ ssh-keygen.exe
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/pawel.osobinski/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/pawel.osobinski/.ssh/id_rsa.
Your public key has been saved in /c/Users/pawel.osobinski/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Q0iaQYL5goT84QiiX9ePr7suHjuwB9mPthtyYB8aqEo pawel.osobinski@DESKTOP-FS268CA
The key's randomart image is:
+---[RSA 2048]-----+
|+oo  ..          |
|0=.o..          |
|0+=.. ..        |
|=Eo+o ...       |
|oo =o+  So      |
|. +.+.. . .     |
|  .++o .        |
|  .o^+. .        |
|  +=*=+         |
+----[SHA256]-----+
pawel.osobinski@DESKTOP-FS268CA MINGW64 ~
$ ls .ssh/
id_rsa id_rsa.pub
pawel.osobinski@DESKTOP-FS268CA MINGW64 ~
$ |
```

<https://help.github.com/articles/connecting-to-github-with-ssh/>

Instalacja i Generowanie klucza ssh puttygen

Instalacja:

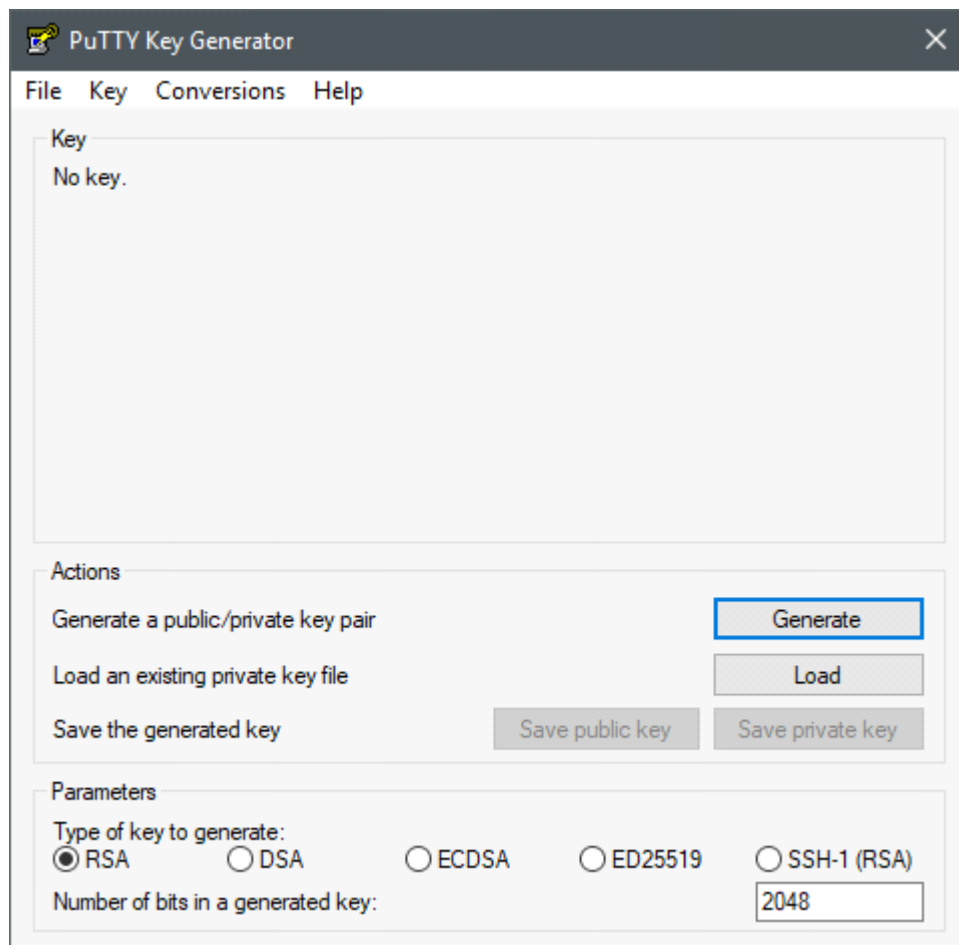
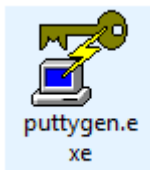
Windows:

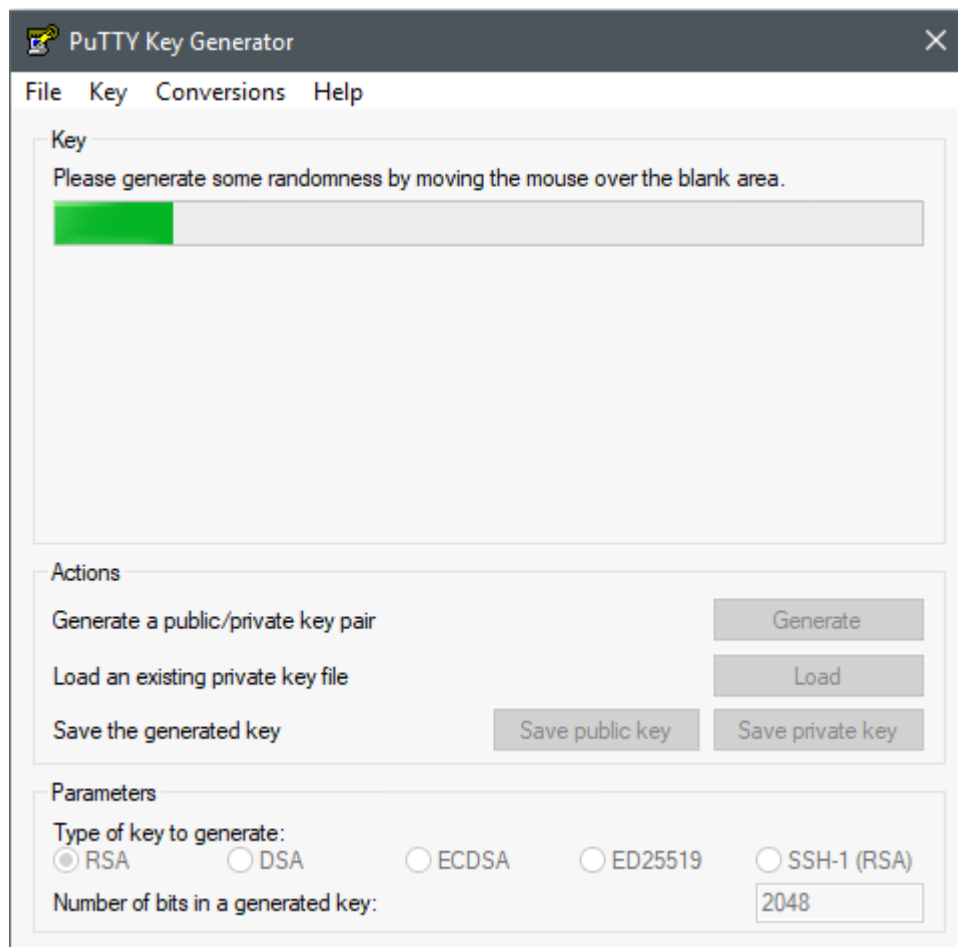
<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

puttygen.exe


Linux:

Pakiet ssh zawierający ssh-keygen





Zmieniamy comment:

 PuTTY Key Generator

File Key Conversions Help

Key

Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAQEAuA2zVjWhXYJE6teHR/vIkVY7AOCVtgMgb1Ti
ryTuGKcet
+HhPkkE7RCa1nkiNlihQVrIB63aqMLVLzvBUR0zapaNFJ5mSn2yHYccojmzzPn/3OVD
e7XDi7ix0hikERzWVzVK2vHMiibFm0RT48H92pwdd1iN5JFNsYAXmvHrEvjyuV/P9pZ
```

Key fingerprint: ssh-rsa 2048 c5:4a:f9:3a:45:da:af:72:66:ee:88:8e:98:c3:5c:52

Key comment: pawel.osobinski@ezse.pl

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair Generate

Load an existing private key file Load

Save the generated key Save public key Save private key

Parameters

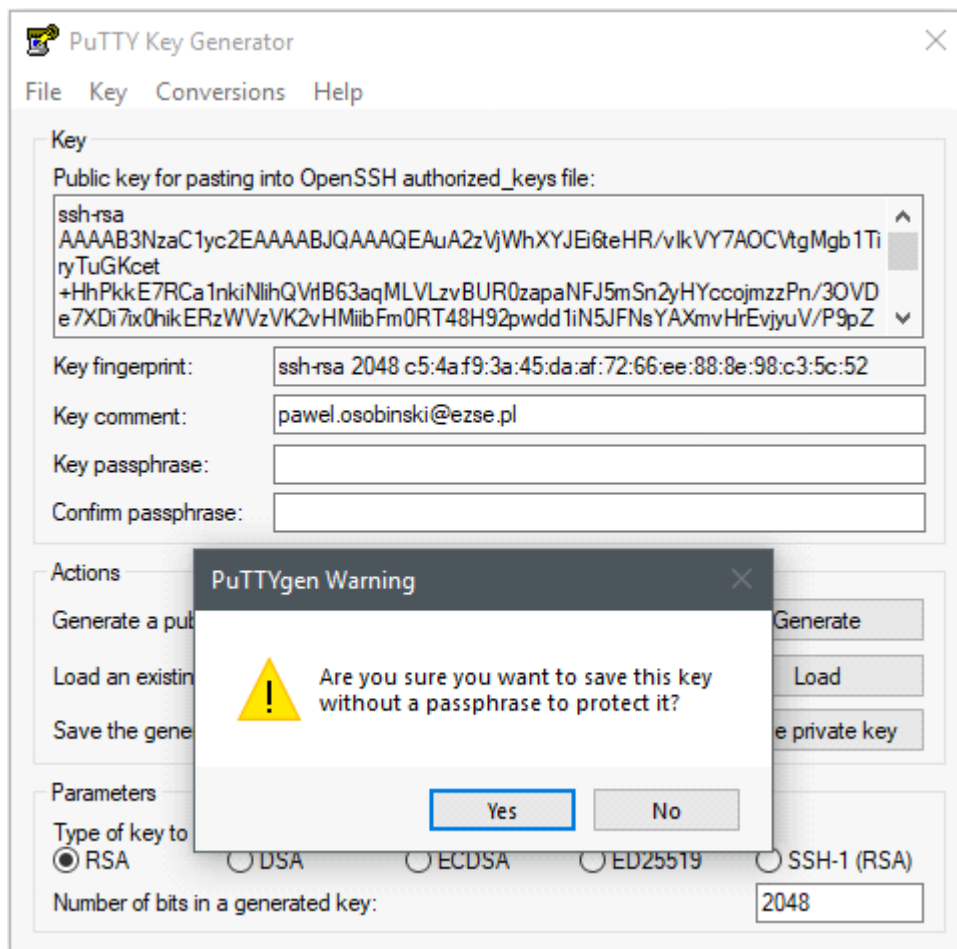
Type of key to generate:

☒ RSA ☐ DSA ☐ ECDSA ☐ ED25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048


File name: pawel-osobinski-zse-git.pub


Save as type: All Files (*.*)




File name:

Save as type:

 pawel-osobinski-zse-git.ppk

 pawel-osobinski-zse-git.pub

 puttygen.exe

Polecam ED25519, w jego wypadku wpis do GitHub zaczynałby się od:
ssh-ed25519 (KLUCZ)

Żeby klucze działały musimy zmienić im nazwy i skopiować do katalogu .ssh np:

c:\Users\" + user + "\.ssh\id_rsa(.pub)

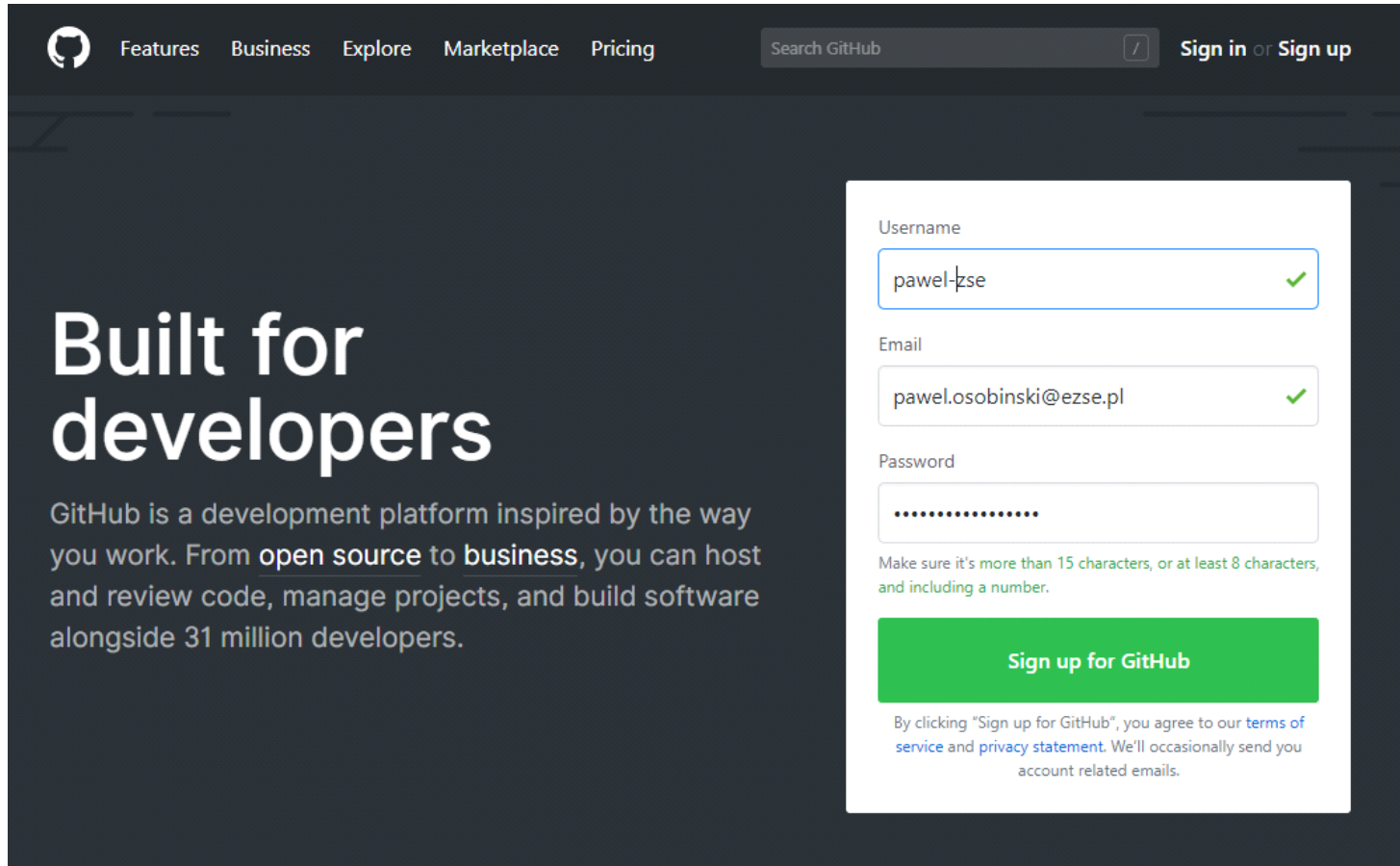
c:\Users\" + user + "\.ssh\id_dsa(.pub)

c:\Users\" + user + "\.ssh\id_ecdsa(.pub)

c:\Users\" + user + "\.ssh\id_ed25519(.pub)

GitHub 101

Rejestracja konta



The image shows the GitHub sign-up page. On the left, there's a dark blue section with the GitHub logo and navigation links: Features, Business, Explore, Marketplace, Pricing. Below these is a search bar labeled 'Search GitHub' and links for 'Sign in' or 'Sign up'. The main heading is 'Built for developers'. Below it, a paragraph describes GitHub as a development platform. On the right, there's a white sign-up form with fields for Username, Email, and Password. The Username field contains 'pawel-zse' and the Email field contains 'pawel.osobinski@ezse.pl', both with green checkmarks. The Password field is masked with dots. Below the password field is a note about password requirements. At the bottom of the form is a green 'Sign up for GitHub' button and a disclaimer about terms of service and privacy statement.

Features Business Explore Marketplace Pricing

Search GitHub

Sign in or Sign up

Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 31 million developers.

Username

pawel-zse ✓

Email

pawel.osobinski@ezse.pl ✓

Password

.....

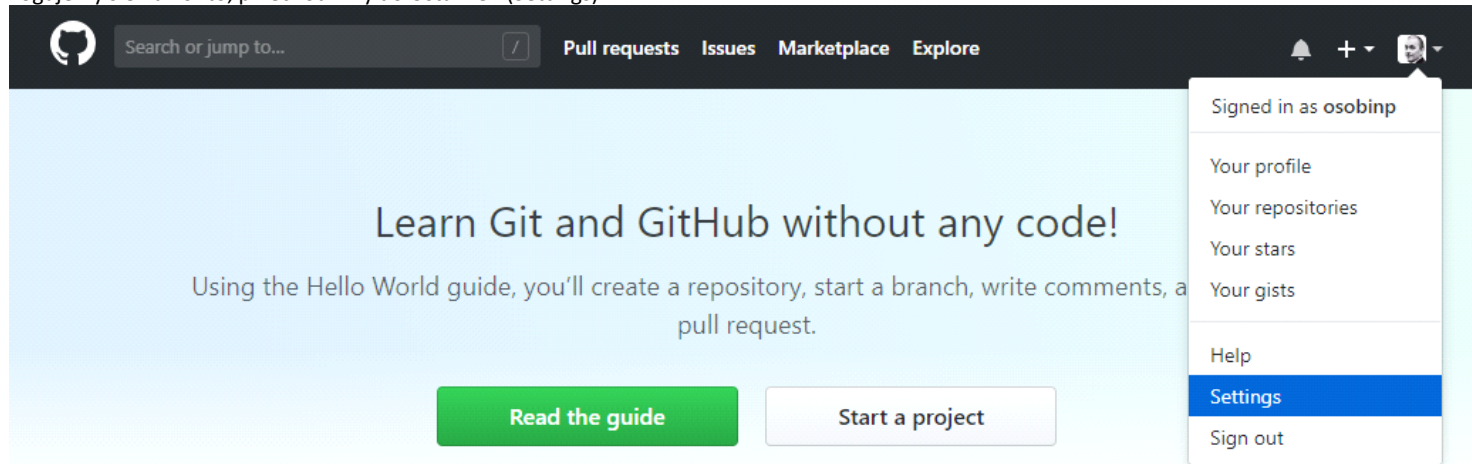
Make sure it's more than 15 characters, or at least 8 characters, and including a number.

Sign up for GitHub

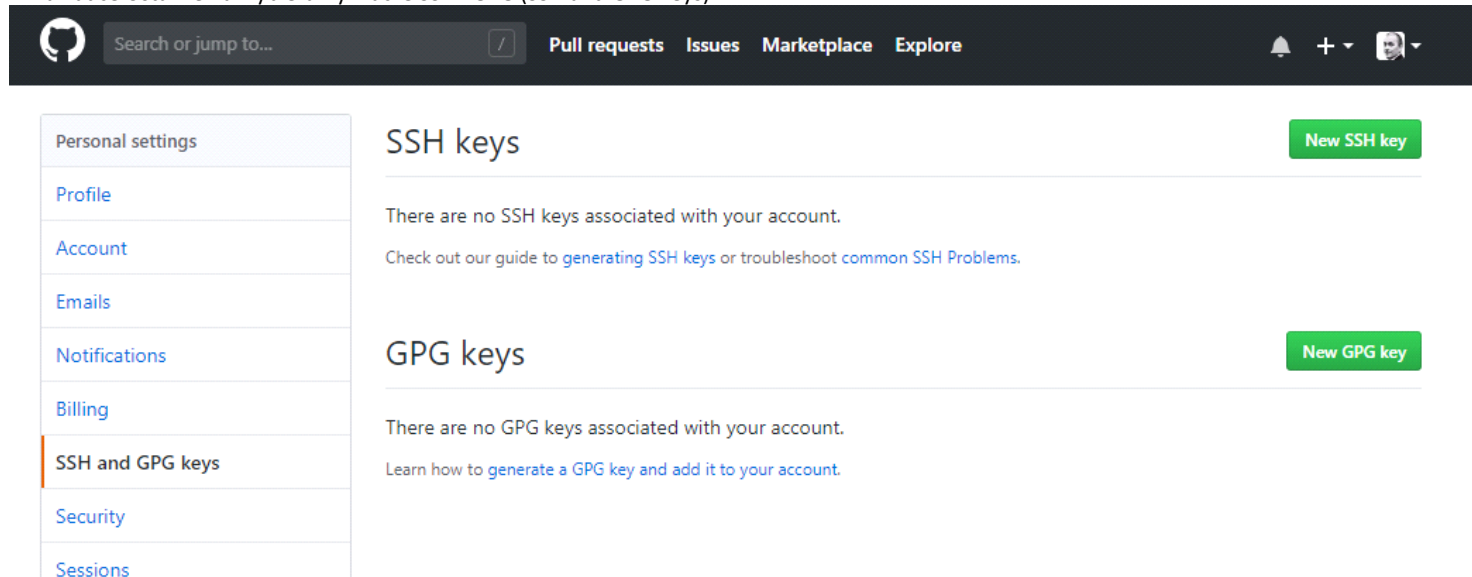
By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails.

Dodanie klucza do GitHub-a

Logujemy się na konto, przechodzimy do Ustawień (Settings)




W zakładce Ustawienia wybieramy Klucze SSH i GPG (SSH and GPG keys)






Podajemy tytuł klucza (który będzie nam jednoznacznie umożliwiał identyfikację do czego jest owy klucz)

W zakładce Klucz (Key) podajemy wygenerowany klucz pamiętając by klucz zaczynał się od:

- ssh-rsa (w przypadku klucza RSA)
- ssh-ed25519 (w przypadku klucza ED25519)
- ecdsa-sha2-nistp256 (w przypadku klucza ECDSA) itd.



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



[Personal settings](#)
[Profile](#)
[Account](#)
[Emails](#)
[Notifications](#)
[Billing](#)
[SSH and GPG keys](#)
[Security](#)
[Sessions](#)
[Blocked users](#)
[Repositories](#)
[Organizations](#)

SSH keys / Add new

Title


zse-git-2018-klucz

Key




ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAAQEAuA2zVjWhXYEi6teHR/vlkVY7AOCVtgMqb1T
iryTuGKcet+HhPkkE7RCa1nkiNlihQvrlB63aqMLVLzvBUR0zapaNFJ5mSn2yHYc
cojmzzPn/3OVD7x0hikERzWVzVK2vHMiiBfM0RT48H92pwwd1iN5JFNsYA
XmvHrEviyuV/P9pZDwv8tIKxU2HDMtnD5iEN+rGnFY++JS2mygg/ChgadvAXRbXp
Pkc0Kar441iM3dzTDkrYJOprLJmYsSZ0oYrB3QLzefOfHtJ3f2gQWY9PQmpZBdwr
1PR315brgcDczwU47+ABWivBooCnaXfNmO3OFTTsj919pSNVxw==

Add SSH key

Nowy klucz powinien być widoczny w sekcji kluczy



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)




[Personal settings](#)
[Profile](#)
[Account](#)
[Emails](#)
[Notifications](#)
[Billing](#)
[SSH and GPG keys](#)

SSH keys

New SSH key

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



zse-git-2018-klucz

c5:4a:f9:3a:45:da:af:72:66:ee:88:8e:98:c3:5c:52

Added on Nov 17, 2018

Never used — Read/write

Delete

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

Git_Workshops_ZSE2018 Page 20

Nowy Projekt - hello-world

Repozytorium jest zwykle używane do organizowania pojedynczego projektu.

Repozytoria mogą zawierać foldery i pliki, obrazy, filmy, arkusze kalkulacyjne i zestawy danych - wszystko, czego potrzebuje Twój projekt.

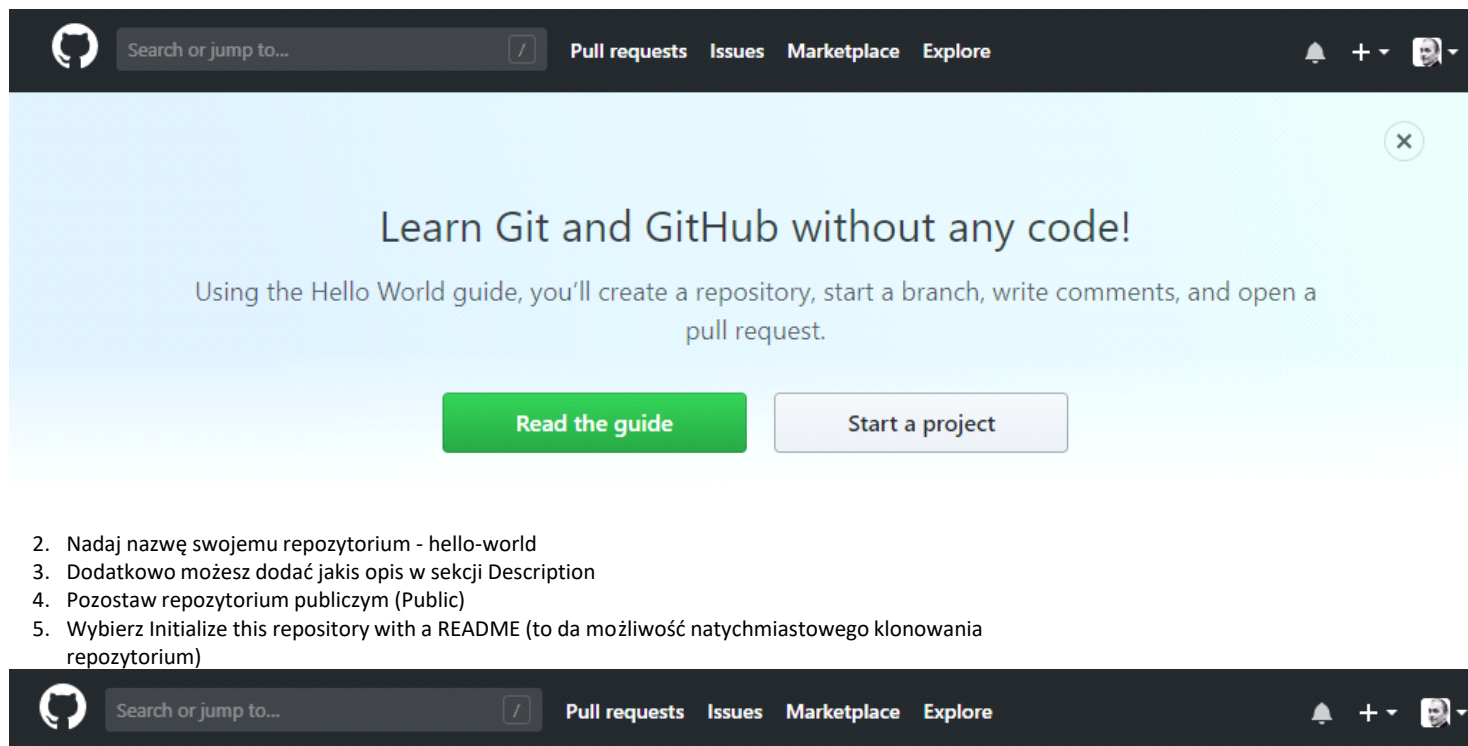
Zalecamy dołączenie pliku README lub pliku zawierającego informacje o projekcie.

GitHub ułatwia dodawanie go w tym samym czasie, gdy tworzysz swoje nowe repozytorium.

Oferuje również inne typowe opcje, takie jak plik licencji.

Twoje repozytorium Hello World może być miejscem, w którym przechowujesz pomysły, zasoby, a nawet udostępniasz i omawiasz rzeczy z innymi.

1. Tworzenie repozytorium: w prawym górnym rogu, koło avatara, kliknij przycisk + a później wybierz New repository (Nowe repozytorium)




2. Nadaj nazwę swojemu repozytorium - hello-world
3. Dodatkowo możesz dodać jakiś opis w sekcji Description
4. Pozostaw repozytorium publicznym (Public)
5. Wybierz Initialize this repository with a README (to da możliwość natychmiastowego klonowania repozytorium)

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name

 osobinp / hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **reimagined-eureka**.

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

— you choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

6. Kliknij przycisk stworzenia repozytorium (Create repository)

The screenshot shows the GitHub interface for a newly created repository named 'hello-world' by user 'osobinp'. The repository is empty, with no description, website, or topics provided. It has 1 commit, 1 branch (master), 0 releases, and 1 contributor. The 'Clone or download' button is highlighted in green. Below the repository name, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Code' tab is selected, showing the 'Initial commit' by 'osobinp' with a 'README.md' file. The content of the README.md file is 'hello-world'.

Gratulacje ! Pierwsze repozytorium stworzone :)

The screenshot shows the 'Clone or download' dropdown menu. It contains the following options: 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The 'Clone or download' option is selected, showing a modal with the following content: 'Clone with HTTPS', 'Use SSH', 'Use Git or checkout with SVN using the web URL.', and the URL 'https://github.com/osobinp/hello-world.git'. There are also buttons for 'Open in Desktop' and 'Download ZIP'.

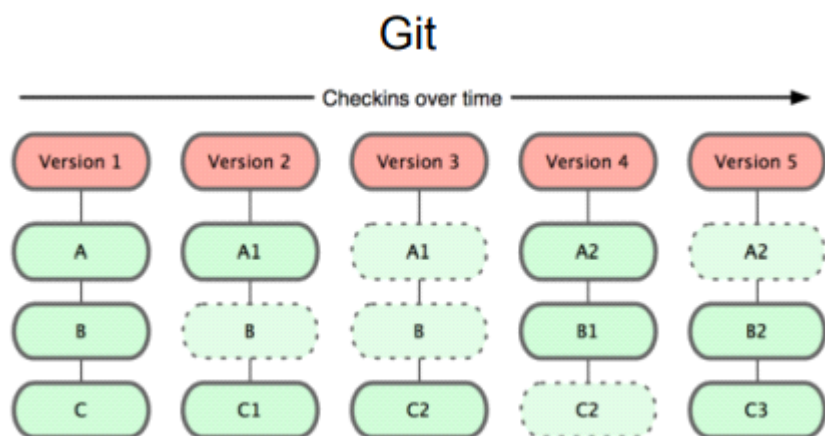
<https://guides.github.com/activities/hello-world/>

GIT 201

Podstawowy koncept Git-a

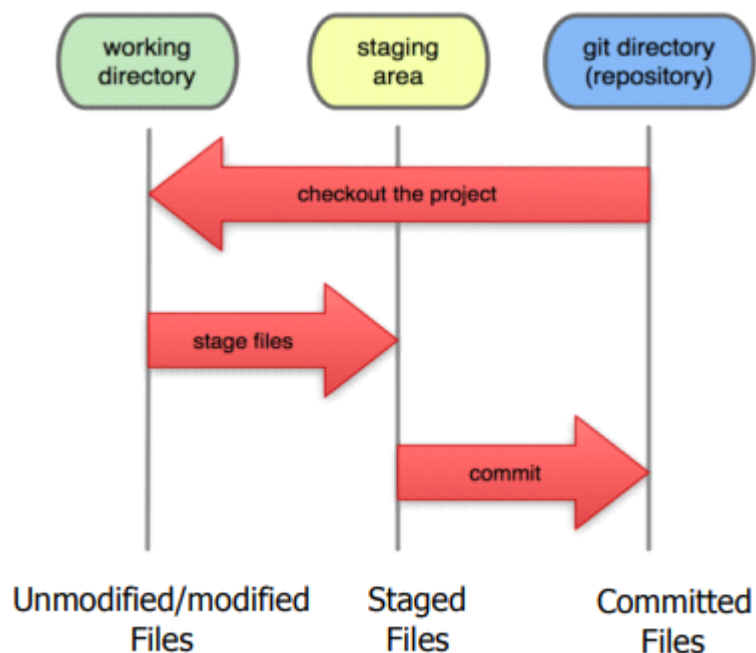
Git utrzymuje stan całego projektu w tzw. Migawkach (Snapshots)

- Każda wersja (checkin) sprawdzająca kod (dane) ma kopie każdego pliku
- Niektóre pliki mogą się zmienić w danej wersji, inne nie
- Większa redundancja, szybsze działanie



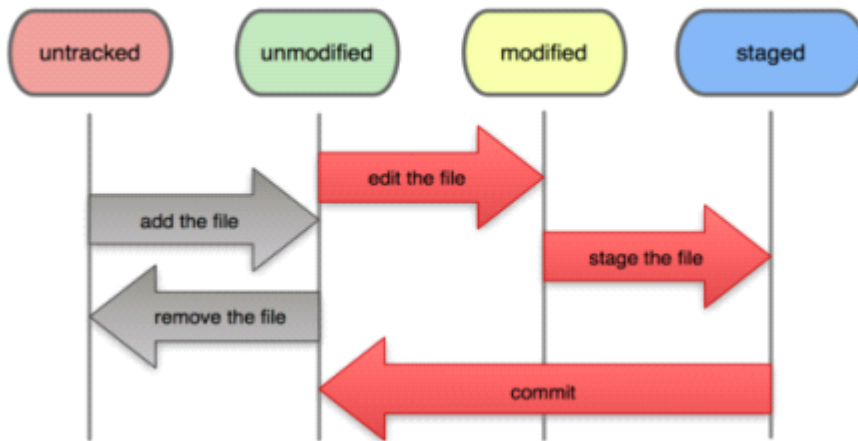
Pliki w lokalnej kopii repozytorium git mogą być w następujących stanach:

- Pliki mogą być zawierdzone (committed)
- Kopia na której pracujemy (working copy) - pobrane i zmodyfikowane pliki ale jeszcze nie zatwierdzone (committed)
- Stan pomiędzy czyli (faza przygotowawcza) - staging. Pliki znajdujące się w tym stanie są gotowe do zatwierdzenia. Commit zapisuje snapshot wszystkich plików do zatwierdzenia



Cykl życia plików w repozytorium:

File Status Lifecycle



Sumy kontrolne:

W przypadku innych systemów kontroli wersji (np. Subversion) każda modyfikacja centralnego repozytorium inkrementuje (zwiększa) wersję całkowitą repozytorium

- W przypadku Git-a każdy użytkownik ma swoją własną kopię repozytorium i wprowadza zmiany (commituje) zmiany w lokalnej kopii nim je wyśle (push) do serwera centralnego
- Git generuje unikatowy hash (40 znakowy łańcuch w szesnastkowym systemie liczbowym (hex)) dla każdej zatwierdzonej zmiany.
- Git odwołuje się częściej do zmian po przez commit ID niż przez numer wersji
- Zazwyczaj widzimy to przez 7 pierwszych znaków np:
1677c89 Zmiana pierwszej linii pliku README.md
25a12c9 Dodanie nowej linii do pliku inventory
0e41fa2 Initial commit (kiedy tworzy się repozytorium)

Tworzenie repozytoriów (dwa podstawowe scenariusze):

1. Tworzenie lokalnego repozytorium Git-a:

- W swoim katalogu domowym tworzymy nowy katalog w którym będziemy trzymać repozytrium
`$ mkdir test-repo`
`$ cd test-repo/`
- Inicjujemy stworzenie repozytorium
`$ git init`
 - To spowoduje stworzenie katalogu .git zawierającego plik konfiguracyjny oraz dodatkowe pliki wymagane do utrzymania wersji
 - Od tej chwili w katalogu test-repo możesz commitować zmiany w plikach
- Stwórz plik który dodasz do repozytorium (np. test-file)
`$ touch test-file`
- Dodaj plik do repozytorium
`$ git add test-file`
- Zatwierdź zmiany w repozytorium (commit) wraz z komentarzem, jeśli nie dodasz -m "komentarz" pojawi się edytor wraz z prośbą o wpisanie powodu zmiany
`$ git commit -m "nowy plik dodany"`

2. Klonowanie repozytorium zdalnego do katalogu lokalnego

- `$ git clone REPO_URL (repo_katalog)`
 - Komenda spowoduje stworzenie lokalnego katalogu repozytorium o nazwie zdefiniowanej w repozytorium lub nazwy którą sami wybraliśmy (repo_katalog), w środku stworzy się także lokalna kopia plików z repozytorium oraz katalog .git

<https://courses.cs.washington.edu/courses/cse403/13au/lectures/git.ppt.pdf>

Podstawowe komendy



NIM ZACZNIEMY!

Podstawowym edytorem z linii poleceń git bash jest VIM

Jeśli jeszcze nie wiesz jak korzystać z VIM-a to :

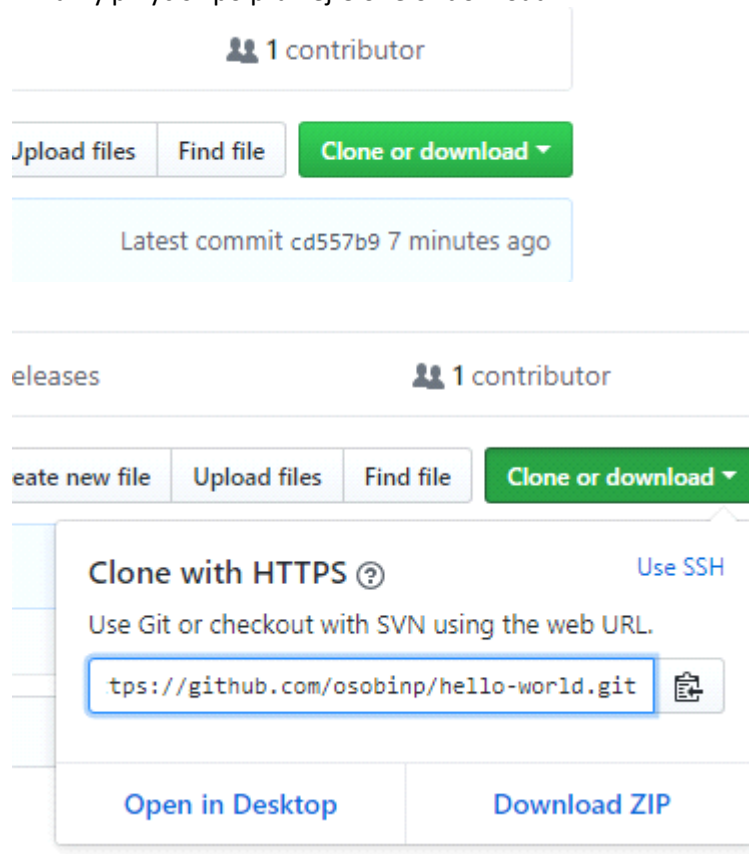
- Bardzo źle, ale poprawisz swój błąd !
- Jeśli jesteś fanem tekstu pisanego:
<http://www.arturpyszczyk.pl/vimtutorial.html>
- Jeśli jesteś graczem :)
<https://vim-adventures.com/>
- Możesz zmienić podstawowy edytor z VIM na np. Nano, Notepad++
\$ git config --global core.editor nano

Komenda	Opis
<code>git clone <i>url</i> [<i>dir</i>]</code>	Tworzy kopie repozytorium ze zdalnego źródła
<code>git add <i>plik</i></code>	Dodaje plik do fazy przygotowawczej (staging)
<code>git commit</code>	Zapisuje snapshot z fazy stage
<code>git status (-s)</code>	Wyświetla status plików z twojego katalogu roboczego oraz staging, -s short
<code>git diff</code>	Pokazuje różnicę między tym co zostało przeniesionego do stage w odróżnieniu do tego co zostało zmodyfikowane ale nie wpisane do stage
<code>git help [<i>command</i>]</code>	Pomoc dla poszczególnych komend

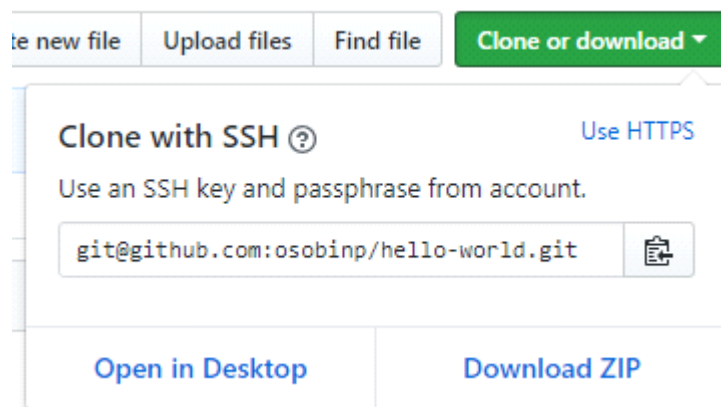
<code>git pull</code>	Pobierz zawartość ze zdalnego repozytorium i postaraj się połączyć z repozytorium lokalnym
<code>git push</code>	Wyślij swoje zmiany oraz pliki do zdalnego repozytorium
	Inne (o tym później): <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>

Klonowanie repozytorium hello-world

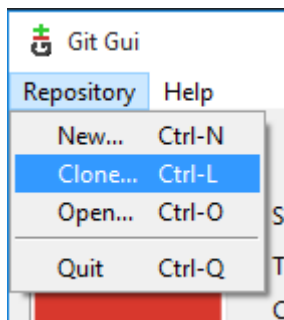
Wchodzimy na swoje repozytorium hello-world na Github
Klikamy przycisk po prawej Clone or download



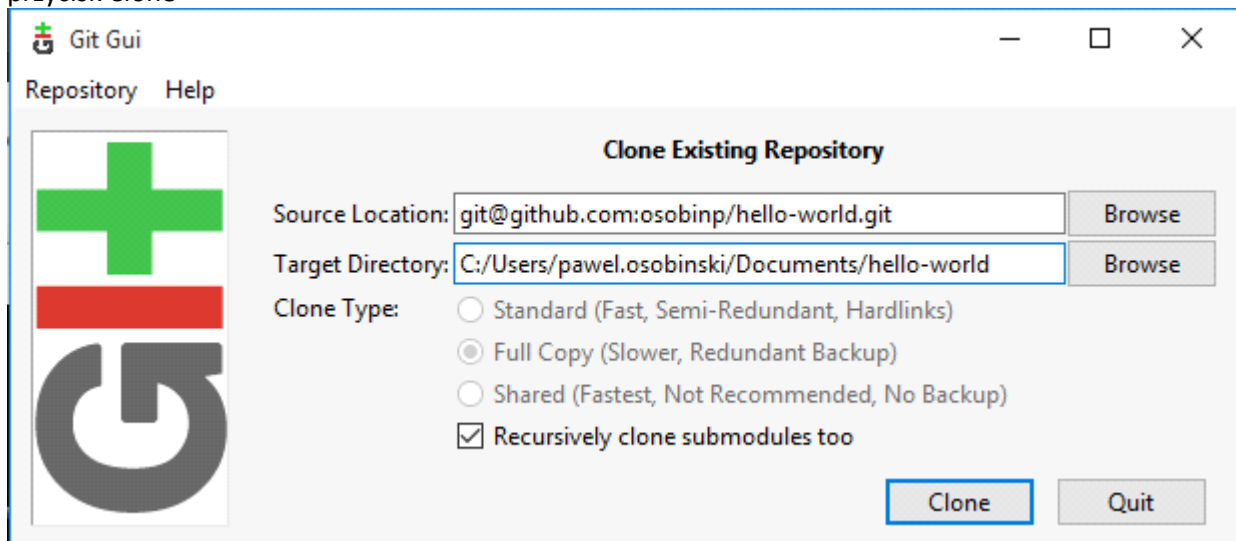
Wybierz opcje klonowania przez SSH (Use SSH)



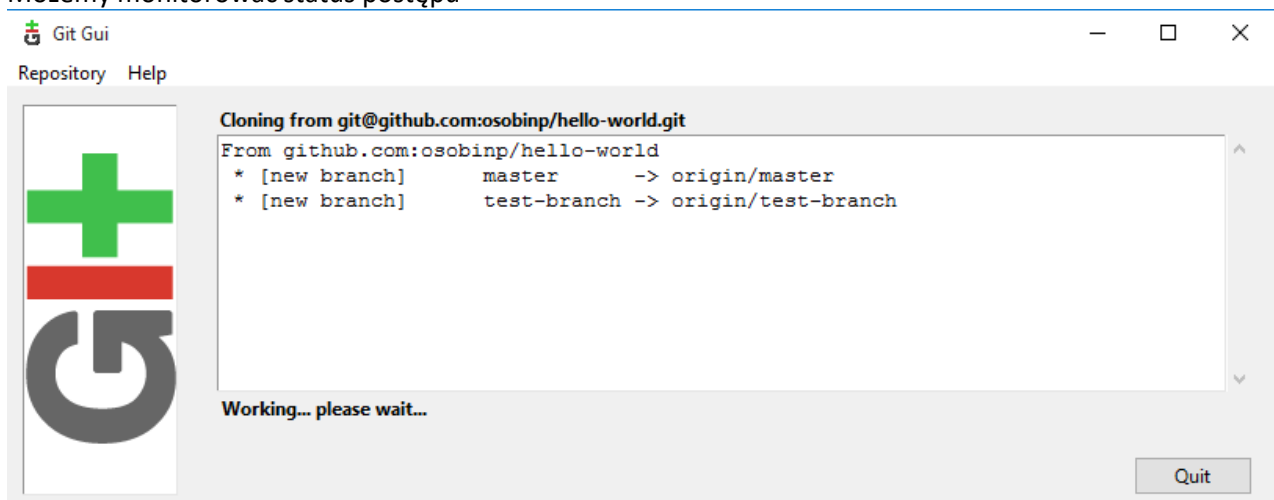
Otwieramy GIT GUI z menu wybieramy Repozytorium (Repository) > Klonowanie (Clone)



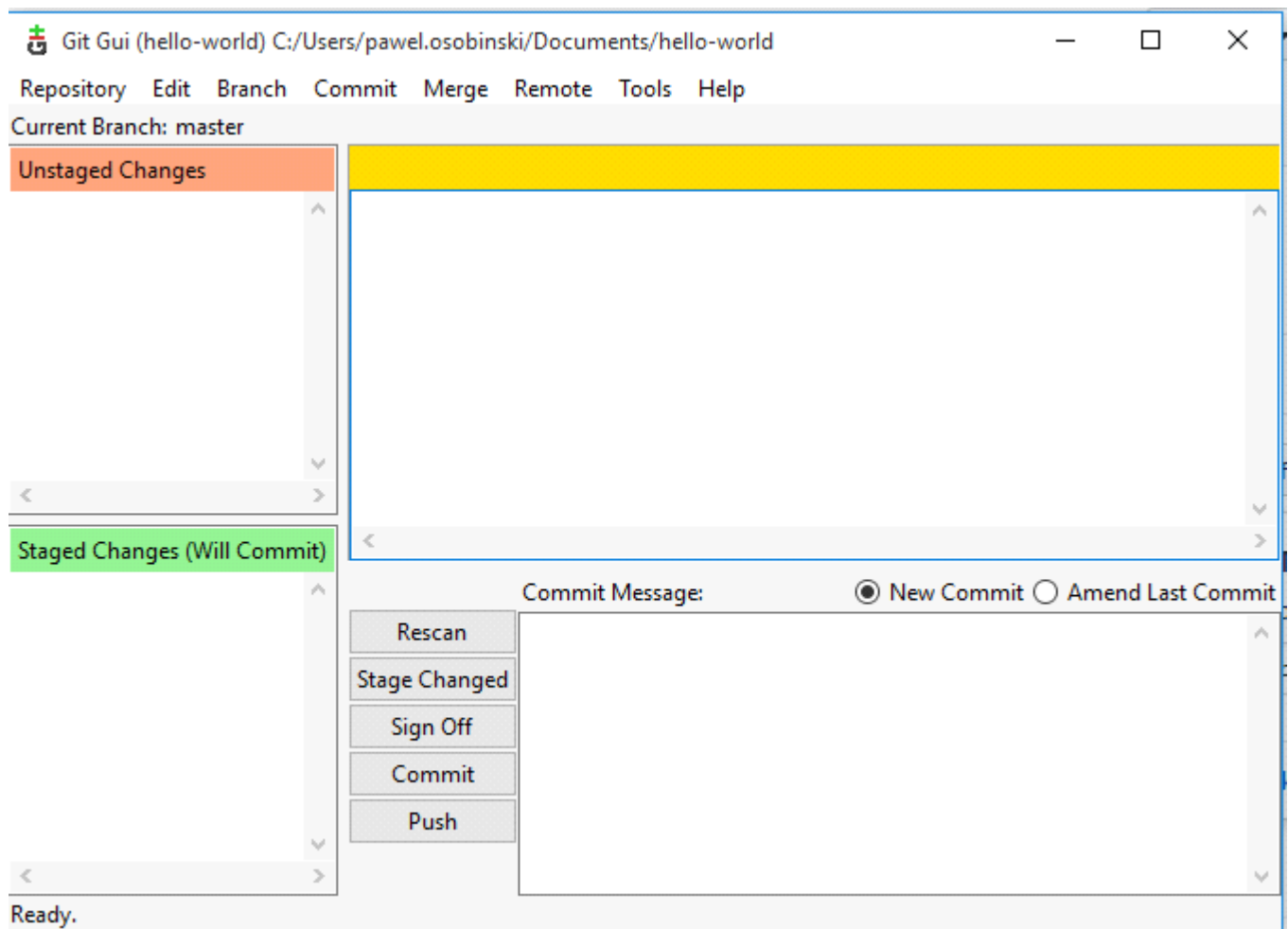
Wpisujemy lokalizację źródłową repozytorium oraz katalog docelowy. Gdy wprowadzimy dane klikamy przycisk Clone



Możemy monitorować status postępu



Po stworzeniu repozytorium zobaczymy ekran nie zawierający (jeszcze żadnych plików)



Jednak jeśli wejdziemy do stworzonego przez nas projektu zobaczymy plik README oraz katalog .git

```
MINGW64:/c/Users/pawel.osobinski/Documents/hello-world

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ pwd
/c/Users/pawel.osobinski/Documents/hello-world

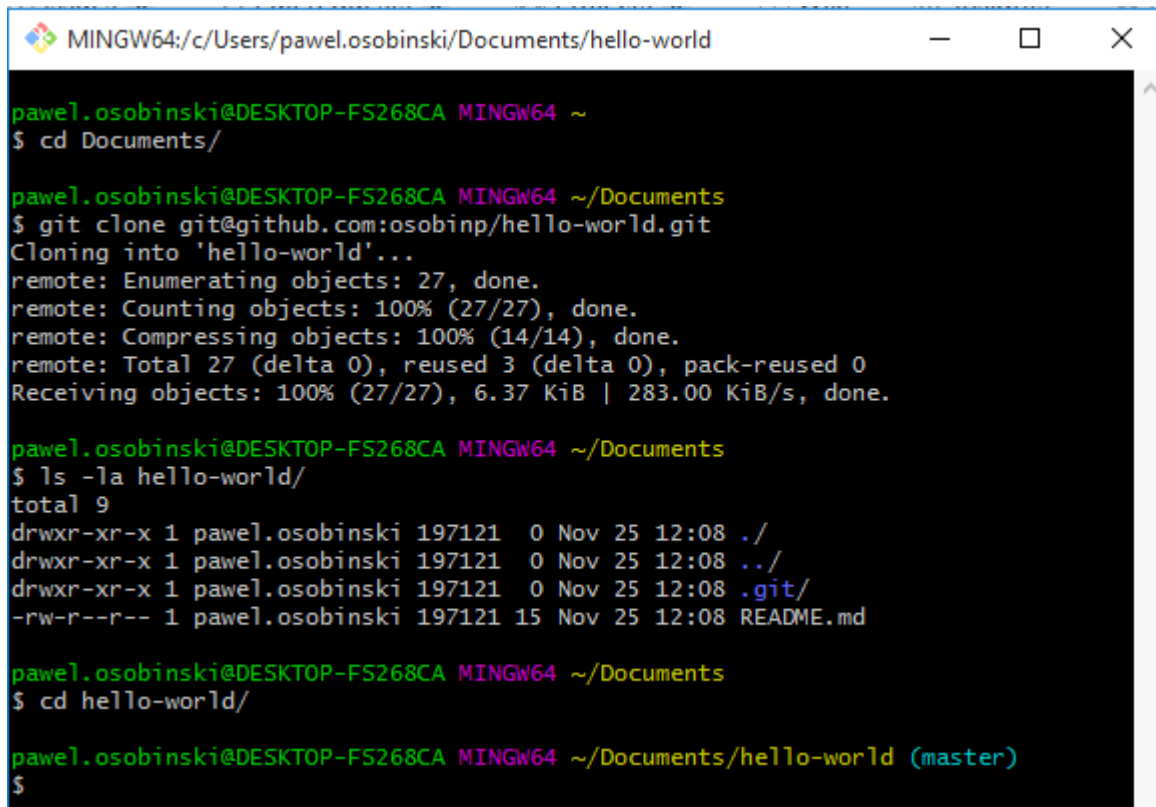
pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ ls -la
total 9
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 11:46 ./
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 11:46 ../
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 11:46 .git/
-rw-r--r-- 1 pawel.osobinski 197121 15 Nov 25 11:46 README.md

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ cat README.md
# hello world

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ |
```

Alternatywnie:
\$ cd Documents/

```
$ git clone git@github.com:(twój_użytkownik)/hello-world.git
```



```
MINGW64:/c/Users/pawel.osobinski/Documents/hello-world

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~
$ cd Documents/

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents
$ git clone git@github.com:osobinp/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 27 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (27/27), 6.37 KiB | 283.00 KiB/s, done.

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents
$ ls -la hello-world/
total 9
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 12:08 ./
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 12:08 ../
drwxr-xr-x 1 pawel.osobinski 197121  0 Nov 25 12:08 .git/
-rw-r--r-- 1 pawel.osobinski 197121 15 Nov 25 12:08 README.md

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents
$ cd hello-world/

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$
```

Pamiętaj o ustawieniu user.name i user.email

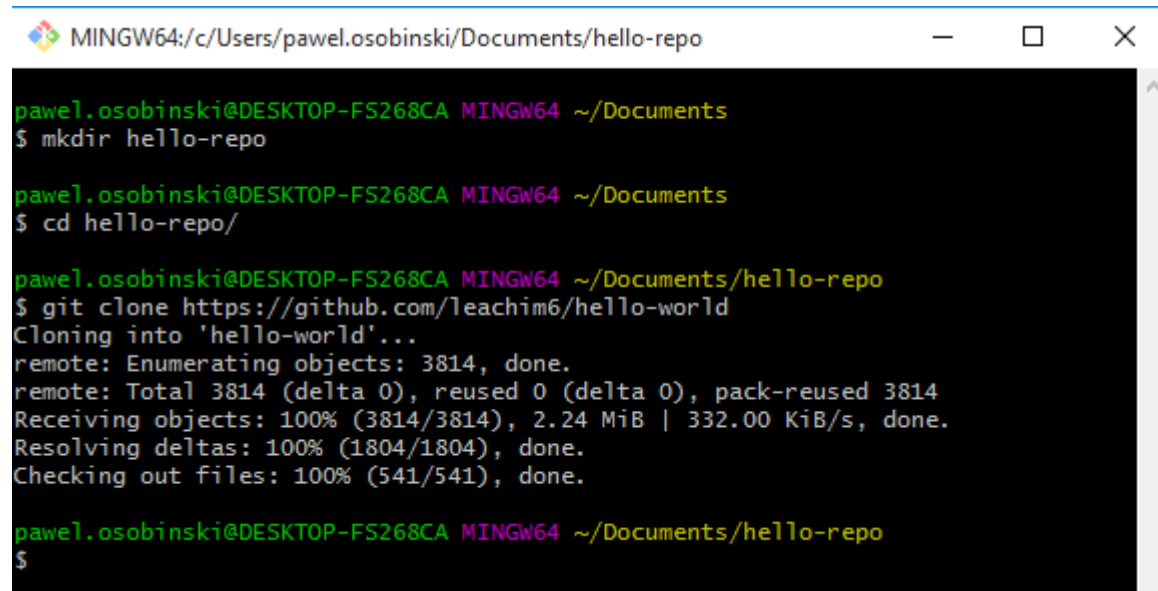
Dodatkowe repozytorium

Ściągnij dodatkowe repozytorium:

<https://github.com/leachim6/hello-world>

<https://github.com/leachim6/hello-world.git>

Użyj innego katalogu (np. hello-repo) żeby ściągnąć sklonowane repozytorium



```
MINGW64:/c/Users/pawel.osobinski/Documents/hello-repo

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents
$ mkdir hello-repo

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents
$ cd hello-repo/

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-repo
$ git clone https://github.com/leachim6/hello-world
Cloning into 'hello-world'...
remote: Enumerating objects: 3814, done.
remote: Total 3814 (delta 0), reused 0 (delta 0), pack-reused 3814
Receiving objects: 100% (3814/3814), 2.24 MiB | 332.00 KiB/s, done.
Resolving deltas: 100% (1804/1804), done.
Checking out files: 100% (541/541), done.

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-repo
$
```

Zadania do wykonania

Zadanie 1

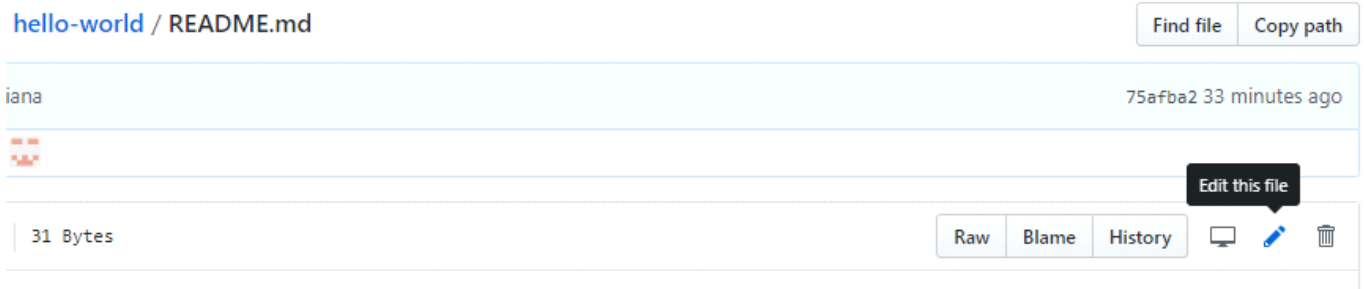
1. Dodatkowe repozytorium, które sklonowałeś zawiera zbiór Hello World w każdym znanym języku komputerowym (nie tylko programowania)
2. Wybierz sobie jeden z języków w którym stworzono hello world (najlepiej wybierz inny od swoich kolegów :))
3. Skopiuj plik (pliki - dla bardziej ambitnych ;)) z jednego lub więcej katalogów do swojego projektu (hello-world)
4. Sprawdź status repozytorium
5. Dodaj plik/pliki do repozytorium
6. Sprawdź status repozytorium ponownie
7. Stwórz nową migawkę swojego projektu i zatwierdź zmiany (commit)
8. Wyślij swoją zmianę do GitHub-a

Zadanie 2

1. Pobierz najświeższą wersję swojego repozytorium (hello-world) z GitHub-a
2. Zmodyfikuj plik hello-world oraz README.md
3. Zamień frazę hello world na swój login
4. Sprawdź status oraz różnicę między plikami
5. Zastage-uj oba pliki (dodaj oba do repozytorium w nowej wersji)
6. Stwórz nową migawkę swojego projektu i zatwierdź zmiany (commit)
7. Wyślij swoją zmianę do GitHub-a

Zadanie 3

1. Zmodyfikuj plik README.md z poziomu GitHub-a



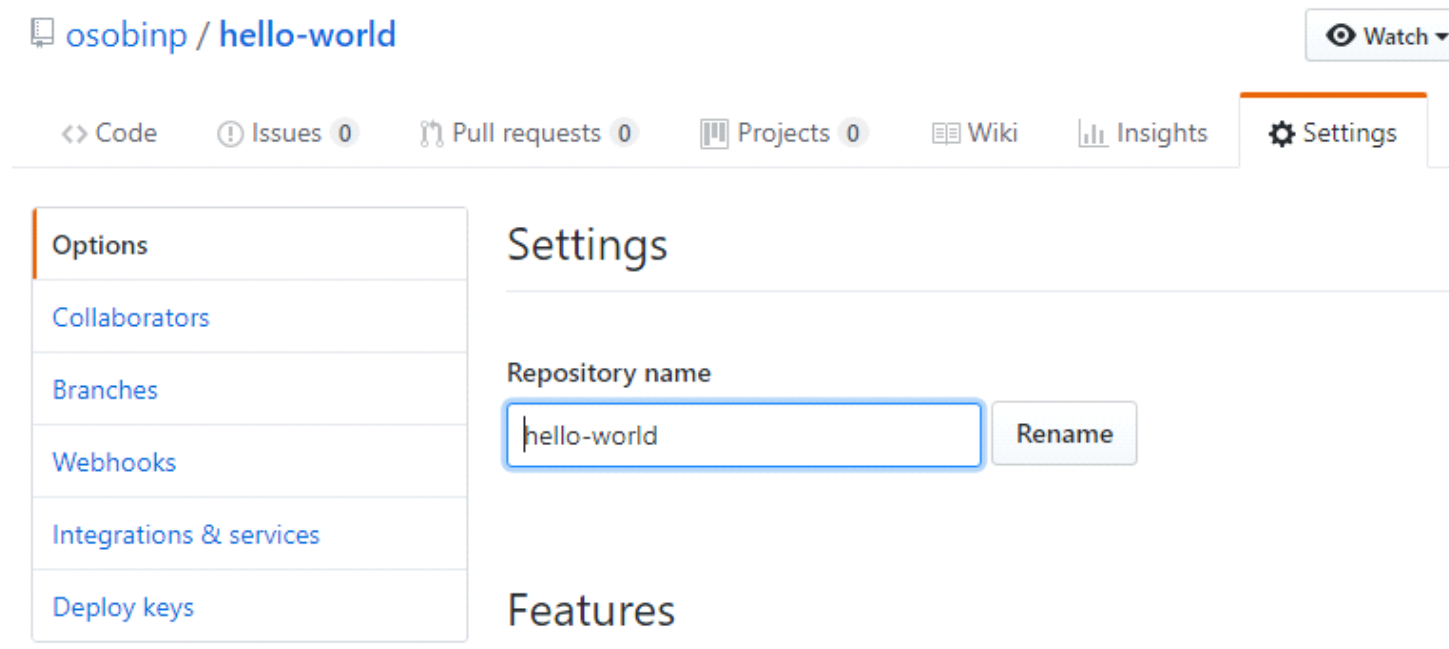
2. Dodaj datę na końcu pliku
3. Zapisz plik i zatwierdź zmianę
4. Zerknij na opcje History oraz Blame (do czego mogą się przydać ?)

GitHub 201

Współpraca przy projektach

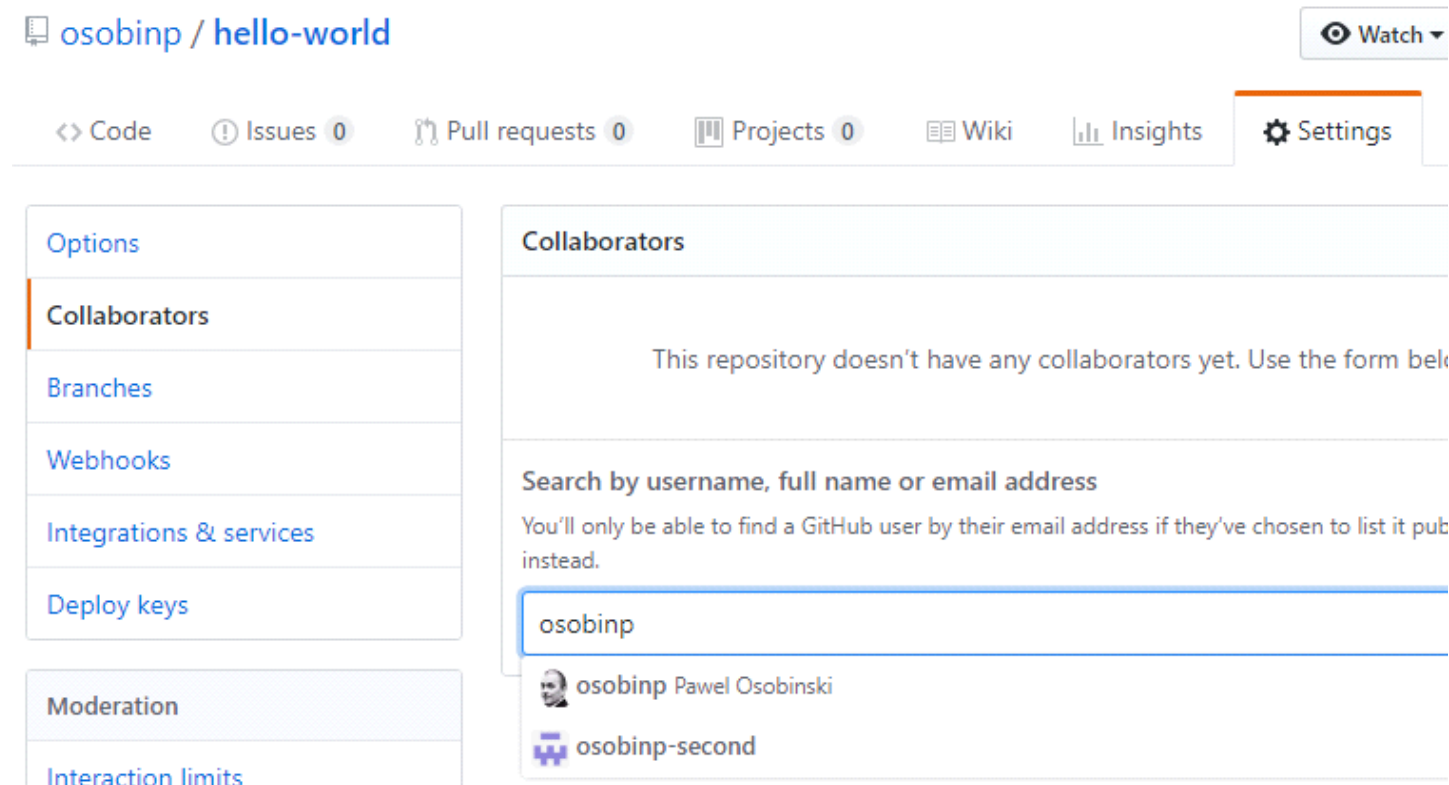
W realnym świecie częściej pracuje się w projektach w których bierze udział więcej niż jedna osoba :)

Aby dodać dodatkowych developerów do swojego projektu wejdź w Ustawienia (Settings)



The screenshot shows the GitHub repository settings page for 'osobinp / hello-world'. The 'Settings' tab is selected in the top navigation bar. On the left sidebar, the 'Options' menu is expanded, showing 'Collaborators', 'Branches', 'Webhooks', 'Integrations & services', and 'Deploy keys'. The main content area is titled 'Settings' and contains a 'Repository name' section with a text input field containing 'hello-world' and a 'Rename' button. Below this is a 'Features' section.

Wybierz opcje Collaborators (współpracownicy). Po prawej stronie pojawi się pole wyszukiwania - znając loginy swoich kolegów możesz ich dodawać do swoich projektów i tworzyć je razem z nimi



The screenshot shows the GitHub repository collaborators page for 'osobinp / hello-world'. The 'Collaborators' tab is selected in the top navigation bar. On the left sidebar, the 'Collaborators' menu item is highlighted. The main content area is titled 'Collaborators' and contains a message: 'This repository doesn't have any collaborators yet. Use the form below to add collaborators.' Below this is a search bar with the placeholder text 'Search by username, full name or email address'. A search input field contains the text 'osobinp'. Below the search bar, a list of users is displayed, including 'osobinp Pawel Osobinski' and 'osobinp-second'.


Gdy wybierzesz współtworzących projekt - każdy z nich otrzyma zaproszenie (można je też wysłać

samemu), które musi zaakceptować by móc tworzyć projekt.

0 requests 0 Projects 0 Wiki Insights Settings

Collaborators

Push access to the repository

 Awaiting osobinp-second's response


Copy invite link ▼

Cancel invite

Aby wysłać link do zaproszenia kliknij Copy invite link

Collaborators

Push access

 Awaiting osobinp-second's response

Copy invite link ▼


Search by username, full name or email address

You'll only be able to find a GitHub user by the name or email address instead.


Copy invite link



osobinp-second's shareable invitation link.

`https://github.com/osobinp/hello-world/invitations/1`



Każdy ma możliwość zaakceptowania udziału w projekcie lub też odrzucenia propozycji


 [osobinp](#) / [hello-world](#)



osobinp invited you to collaborate

Accept invitation

Decline

 Owners of hello-world will be able to see:


- Your public profile information
- [Certain activity](#) within this repository
- Country of request origin
- Your access level for this repository
- Your IP address

Is this user sending spam or malicious content?

[Block osobinp](#)

Po zaakceptowaniu zaproszenia mamy możliwość dodawania/usuwania/modyfikowania projektu

You now have push access to the osobinp/hello-world repository.

 [osobinp](#) / [hello-world](#)

Zadania do wykonania

1. Połączcie się 3-4 osobowe zespoły (jeśli to możliwe)
2. Wybierzcie jeden wspólny projekt
3. Właściciel projektu dodaje zespół do projektu
4. Każdy developer akceptuje zaproszenie i klonuje repozytorium do swojego katalogu
5. Każdy developer dodaje swój plik hello-world

Jeśli dostaniecie komunikat:

```
pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ git push
To github.com:osobinp-second/hello-world.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:osobinp-second/hello-world.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/Documents/hello-world (master)
$ |
```

Dodajcie inny plik, konfliktami oraz gałęziami zajmiemy się za chwilę

GIT 301

Gałęzie rozwojowe (branch) i scalanie (merge)

Rozgałęzienie jest sposobem pracy w różnych wersjach repozytorium naraz.

Domyślnie twoje repozytorium ma jedną gałąź o nazwie master, która jest uważana za gałąź ostateczną. Używamy gałęzi rozwojowych do eksperymentowania i wprowadzania zmian przed wprowadzeniem ich do master-a.

Kiedy tworzysz gałąź z gałęzi głównej, robisz kopię lub migawkę master-a, z aktualnego punktu w czasie. Jeśli ktoś inny wprowadził zmiany w gałęzi głównej podczas pracy swojej gałęzi, możesz pobrać te aktualizacje.

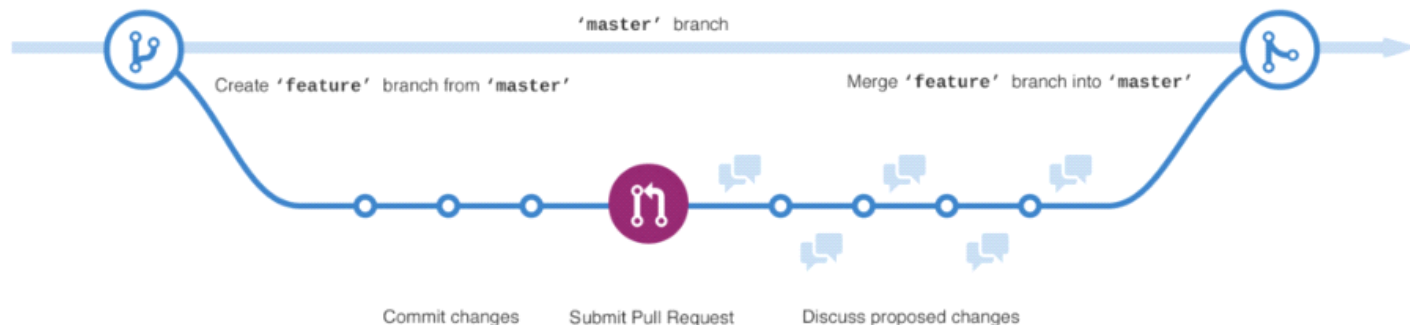


Diagram przedstawia:

- Główną gałąź rozwojową (master)
- Nową gałąź nazwaną feature
- Drogę jaką feature przechodzi do momentu złączenia z master-em

Na GitHub programiści, twórcy i projektanci używają gałęzi do wprowadzania poprawek, dodawania nowej funkcjonalności oddzielnie od gałęzi produkcyjnej. Kiedy zmiana jest gotowa następuje jej złączenie z gałęzią główną (master-em).

Przykładowy scenariusz:

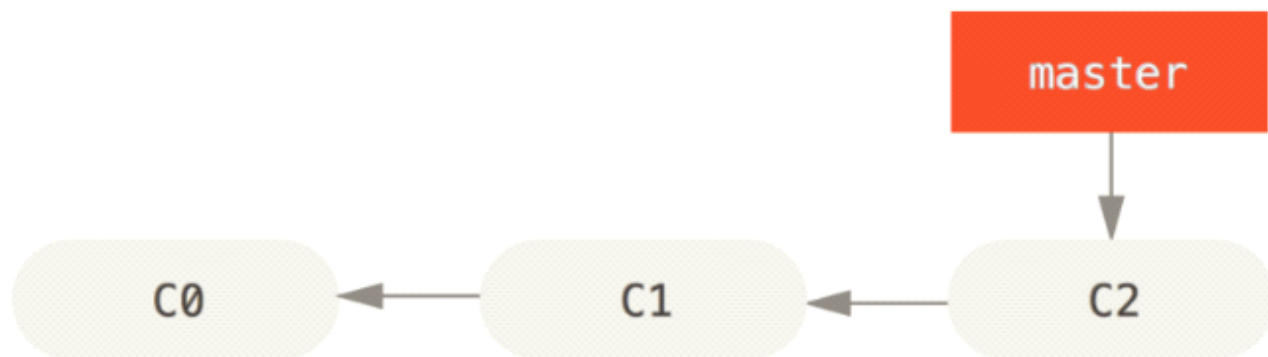
Pracujemy nad projektem otrzymałeś wiadomość że pojawił się problem w aplikacji i po chwili pojawił się kolejny, trzeba wydać hotfix (na issue #53 w systemie ticket-owym) oraz hotfix na drugi problem.

Trzeba zrobić następujące rzeczy:

1. Przejść do produkcyjnej gałęzi (master)
2. Stworzyć nową gałąź i dodać hotfix
3. Po przetestowaniu, scalić (merge) i wypromować zmianę do produkcji (push)
4. Wrócić do gałęzi produkcyjnej i kontynuować pracę

Jak to zrobić ?

Zakładam, że już kilka commitów zostało wprowadzonych do aplikacji w gałęzi master:



Założmy że chcemy nazwać nową gałąź iss53 (od numeru problemu z którym walczymy).

Aby stworzyć nową gałąź i jednocześnie do niej przejść używamy git checkout o przełączniku -b

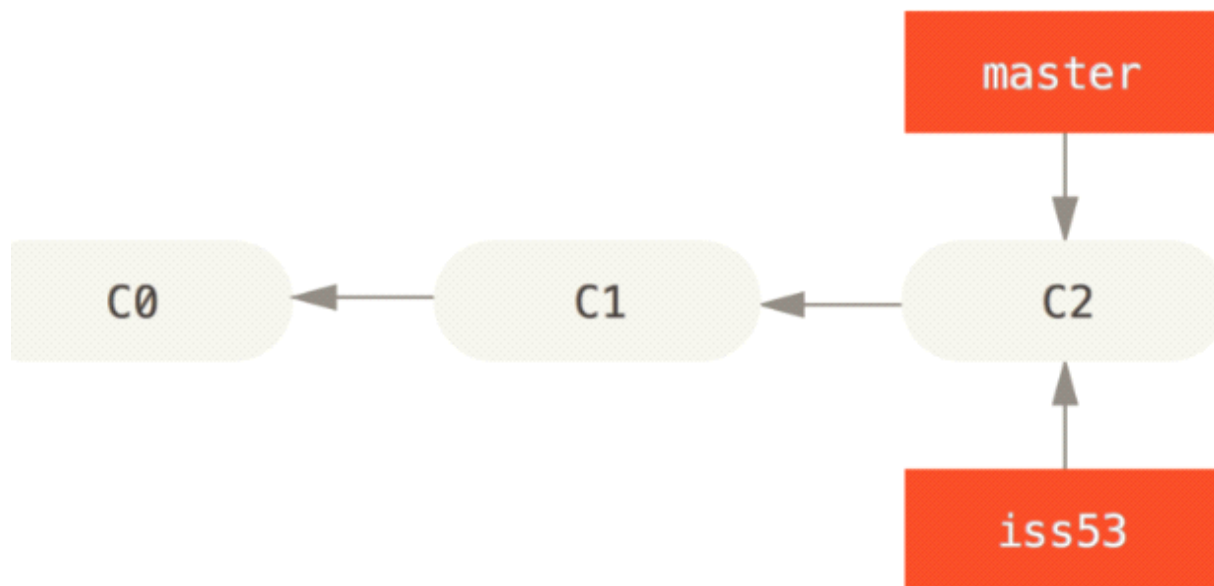
```
$ git checkout -b iss53
```

Switched to a new branch "iss53"

Ten sam efekt można osiągnąć używając kombinacji:

```
$ git branch iss53
```

```
$ git checkout iss53
```



Wprowadzamy modyfikacje które mają poprawić działanie naszej aplikacji (np. Hello-world :))

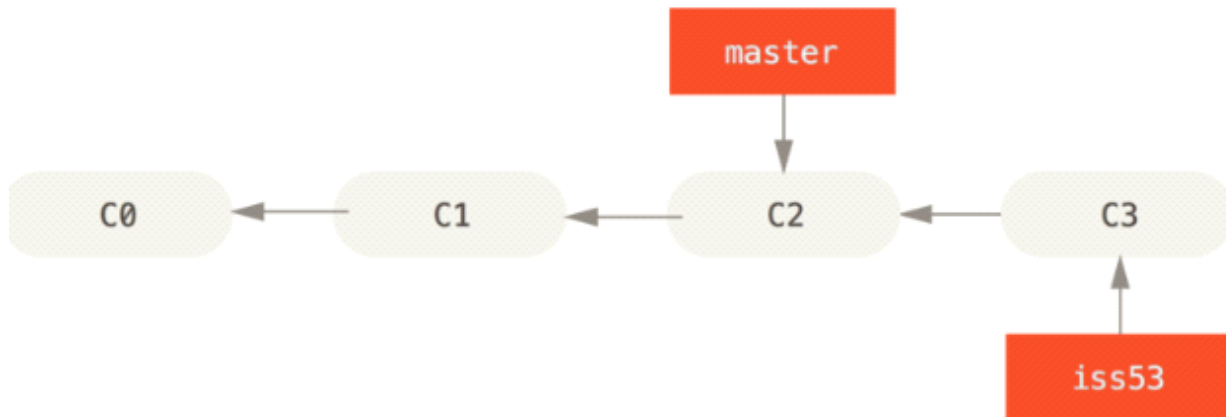
Po wprowadzeniu zmian zatwierdzamy zmiany commit-em.

```
$ vim c.c
```

```
$ git commit -a -m 'dodany hotfix - issue53'
```

```
[iss53 55ed6d2] dodany fix do iss53
```

```
1 file changed, 1 insertion(+)
```



W tej chwili dostajesz telefon, że pojawił się inny problem w aplikacji i nim też trzeba się zająć jak najszybciej (w korporacji ASAP :)). W świecie Git-a nie musisz dodawać swoich zmian do już znanego problemu, nie musisz też cofać już wprowadzonych zmian. Możesz ponownie przejść do gałęzi master. Pamiętaj, że jeśli masz jakieś zmiany nie zacommitowane które konfliktują z gałęzią do której się podłączasz (master) git nie pozwoli Ci przełączyć się między gałęziami. Najlepiej mieć czystą "konto" przez przejściem - tzn. mieć wszystkie zmiany zacommitowane. Istnieją metody jak omijać ten problem (stashing i cleaning).

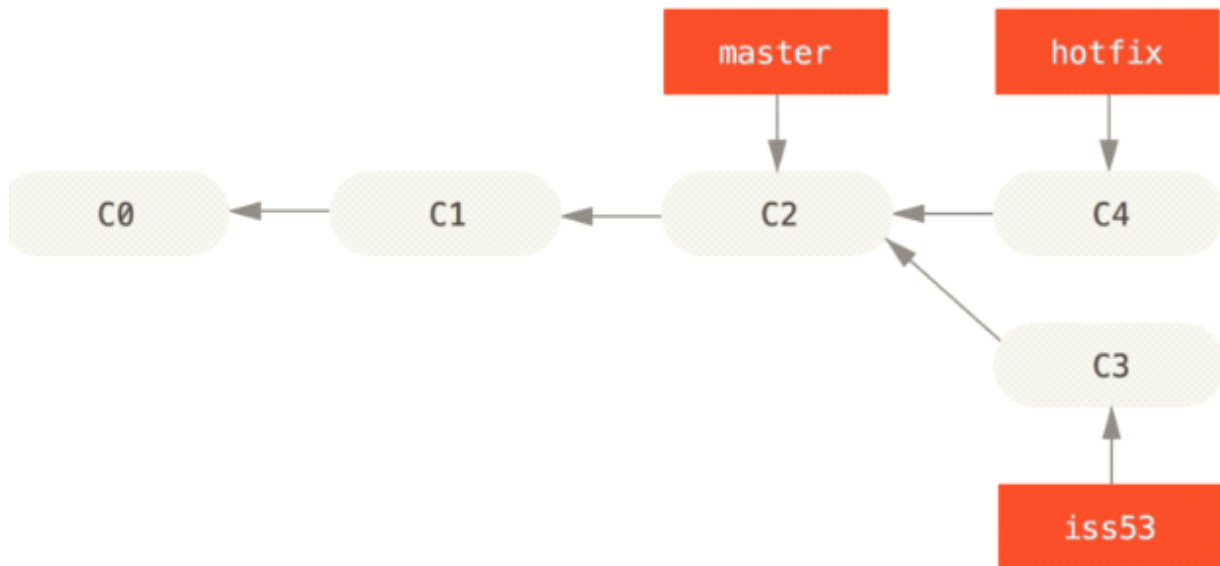
Na razie przyjmujemy, że wszystkie zmiany w obecnej gałęzi zostały zacommitowane, co da możliwość przejścia do master-a:

```
$ git checkout master
```

W ten chwili twoja przestrzeń robocza wraca do stanu z przez iss53, możesz skupić się na hotfix-ie. To ważny element do zapamiętania, gdy zmieniasz gałąź - git resetuje przestrzeń roboczą do ostatniego stanu w którym ta przestrzeń był zacommitowana. Git dodaje usuwa i modyfikuje pliki automatycznie, aby zapewnić, że kopia robocza jest tak sama jak podczas ostatniego zatwierdzenia.

Teraz można stworzyć nową gałąź do hotfix-a. Wprowadzić zmiany w aplikacji i zacommitować zmianę.

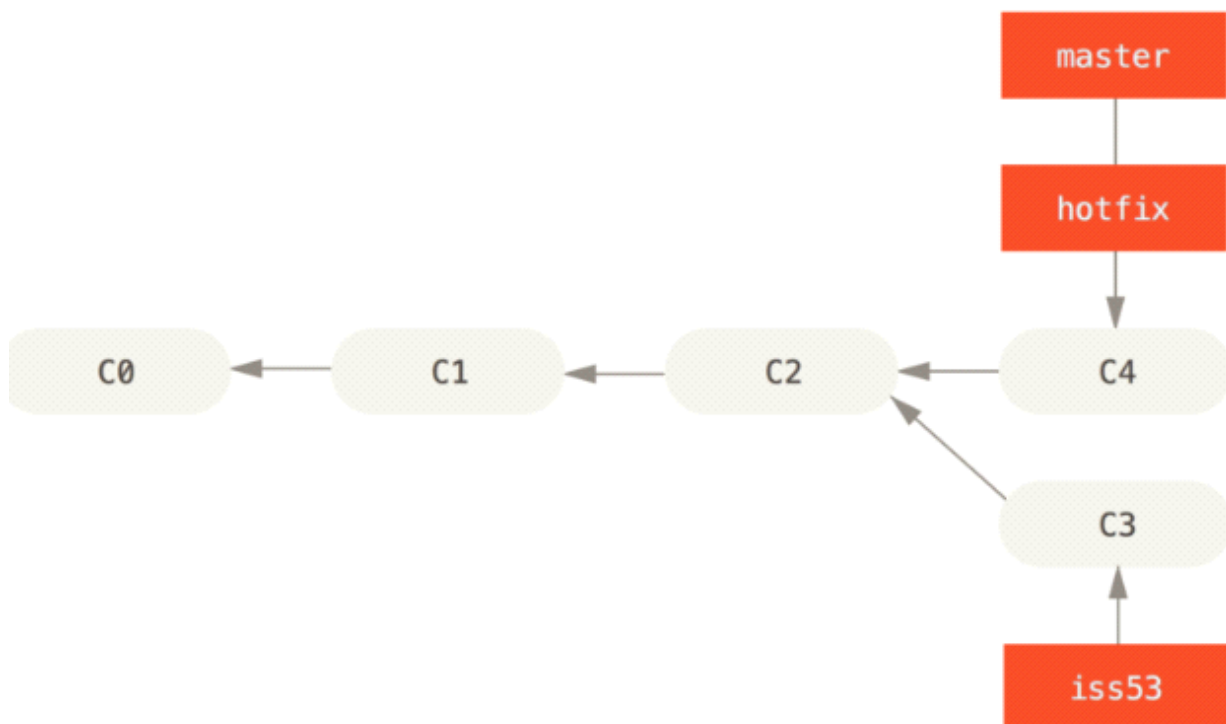
```
$ git checkout -b hotfix
$ vim hello-world.c
$ git commit -a -m "naprawiony problem asap - hotfix"
[hotfix 1fb7853] fixed the broken email address
1 file changed, 2 insertions(+)
```



Gdy już mamy wprowadzone zmiany w aplikacji czas scalić (merge) zmiany z gałęzi hotfix do gałęzi master. Używamy komendy `git merge`

```

$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 hello-world.c | 2 ++
 1 file changed, 2 insertions(+)
  
```



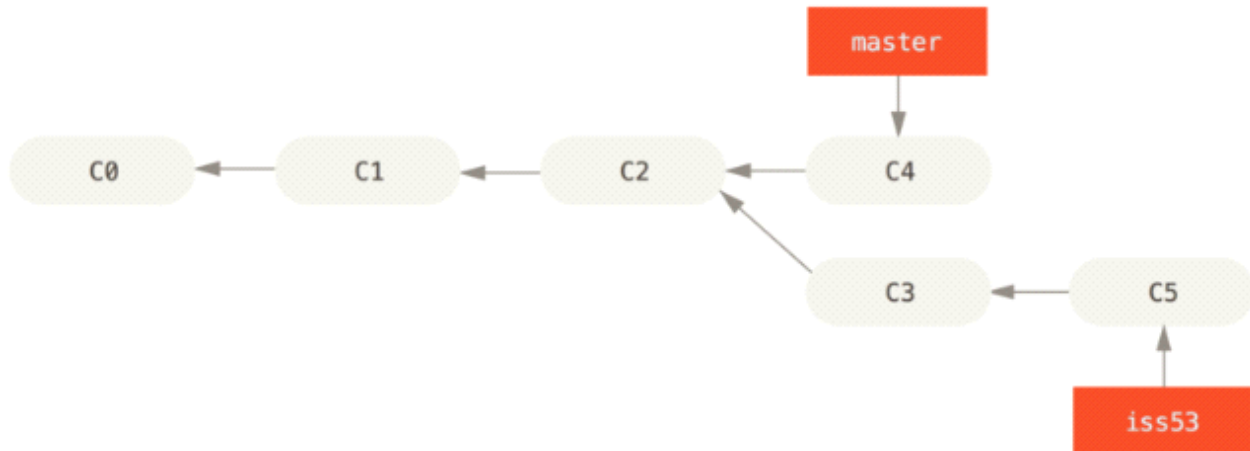
Gdy już super ważny fix zostanie wprowadzony, jesteś gotowy by przejść do poprzedniej pracy (którym było łatanie iss53). Wpierw jednak usuń gałąź hotfix ponieważ już jej nie potrzebujesz - ponieważ master wskazuje teraz na ten sam punkt w czasie do hotfix (ten sam commit). Usuwanie gałęzi wygląda

następująco:

```
$ git branch -d hotfix  
Deleted branch hotfix (3a0874c).
```

Teraz możesz wrócić do iss53:

```
$ git checkout iss53  
$ vim c.c  
$ git commit -a -m "skonczony task - iss53"
```



Warto tutaj zauważyć, że praca wykonana na gałęzi poprawek (hotfix) nie jest zawarta w plikach z gałęzi iss53.

Jeśli musisz je dodać możesz scalić gałąź master z gałęzią iss53. Aby to zrobić wykonaj w swojej gałęzi:

```
$ git merge master
```

Możesz też poczekać ze zmianą i wprowadzić je w trakcie scalania gałęzi iss53 z masterem.

Jeśli nie spowoduje ona ... konfliktu :)

Jesli

WAŻNE:

Gdy korzystamy z centralnego serwera repozytorium pamiętaj o `$ git pull` przed wprowadzaniem zmian - będziesz miał zawsze najświeższe repozytorium

<https://guides.github.com/activities/hello-world/>

<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

Scalanie (merge) - konflikty



```
$ git merge master
Auto-merging c/c.c
CONFLICT (content): Merge conflict in c/c.c
Automatic merge failed; fix conflicts and then commit the result.
```

Możemy zobaczyć różnicę w pliku elementy różniące będą zaznaczone <<< MARKERAMI >>>>

```
pawel.osobinski@DESKTOP-F5268CA MINGW64 ~/hello-world-copy/c (iss53)
$ git merge master
Auto-merging c/c.c
CONFLICT (content): Merge conflict in c/c.c
Automatic merge failed; fix conflicts and then commit the result.

pawel.osobinski@DESKTOP-F5268CA MINGW64 ~/hello-world-copy/c (iss53|MERGING)
$ git status
On branch iss53
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   c.c

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

pawel.osobinski@DESKTOP-F5268CA MINGW64 ~/hello-world-copy/c (iss53|MERGING)
$ cat c.c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    <<<<<< HEAD
    printf("Dodatkowy wpis z iss53\n");
    =====
    printf("Kompletnie inny komentarz\n");
    >>>>>> master
    return 0;
}
```

W każdej chwili możemy sprawdzić które pliki są skonfliktowane status git:

```
$ git status
On branch iss53
```

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

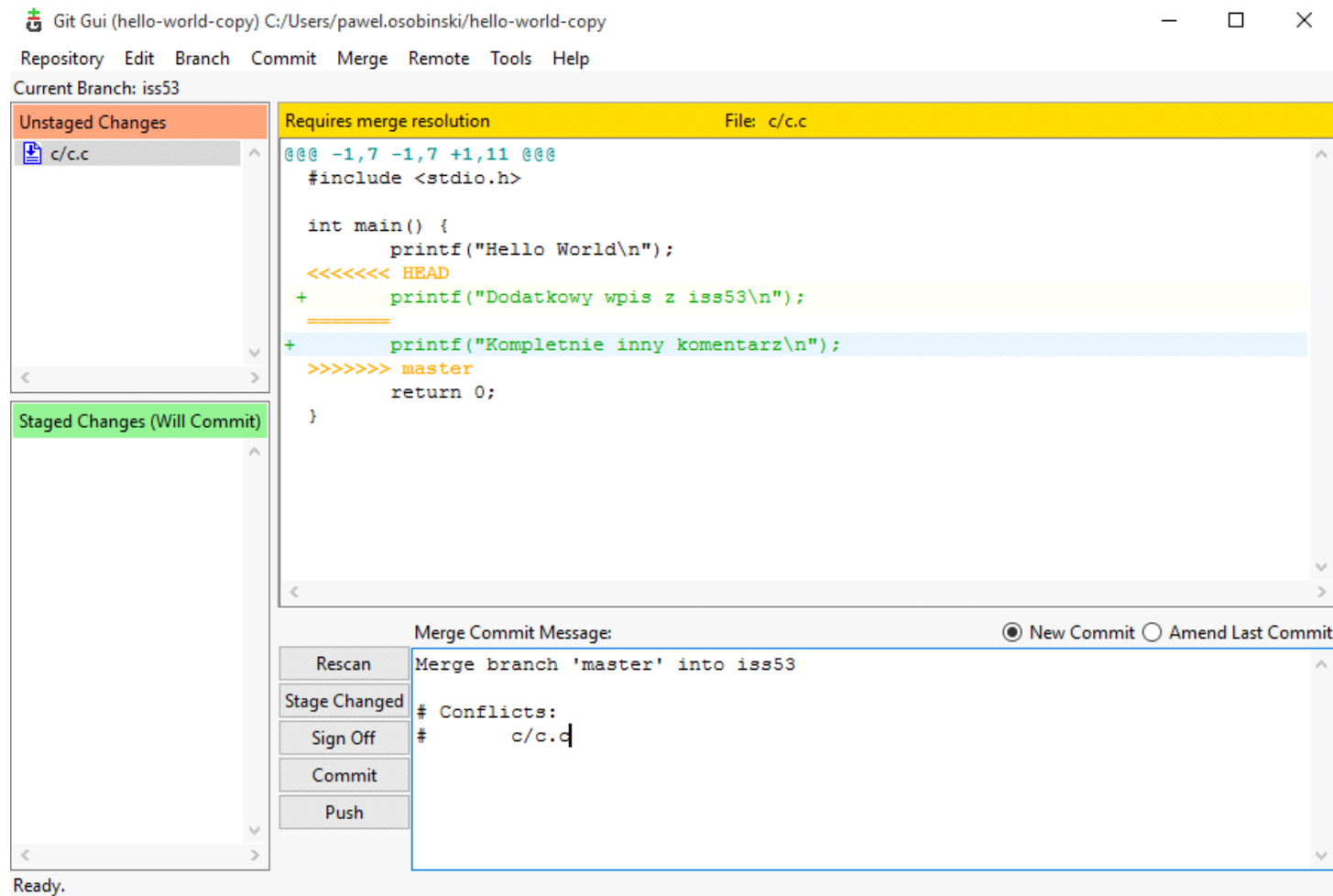
Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: c.c

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

Widać że ten plik został zmodyfikowany, z poziomu GIT GUI



Metoda 1:

Edycja bezpośrednia (vim)

Następnie dodajemy plik do git-a (potwierdzając tym sposobem, że zmiany zostały poprawione)

Po rozwiązaniu konfliktu postępujemy tak jak ze zwykłym scalaniem

```

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (iss53|MERGING)
$ vim c.c

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (iss53|MERGING)
$ git add c.c

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (iss53|MERGING)
$ git commit -a
[iss53 cbc5bd9] oMerge branch 'master' into iss53

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (iss53)
$ cat c.c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    printf("Dodatkowy wpis z iss53\n");
    printf("Kompletnie inny komentarz\n");
    return 0;
}

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (iss53)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (master)
$ git merge iss53
Updating 5a1827a..cbc5bd9
Fast-forward
 c/c.c | 1 +
 1 file changed, 1 insertion(+)

pawel.osobinski@DESKTOP-FS268CA MINGW64 ~/hello-world-copy/c (master)
$ cat c.c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    printf("Dodatkowy wpis z iss53\n");
    printf("Kompletnie inny komentarz\n");
    return 0;
}

```

Motoda 2 (vimdiff) :

\$ git mergetool

```

#include <stdio.h>

int main() {
    printf("Hello World\n");
    printf("Dodatkowy wpis z iss53\n");
    return 0;
}

~
~
~

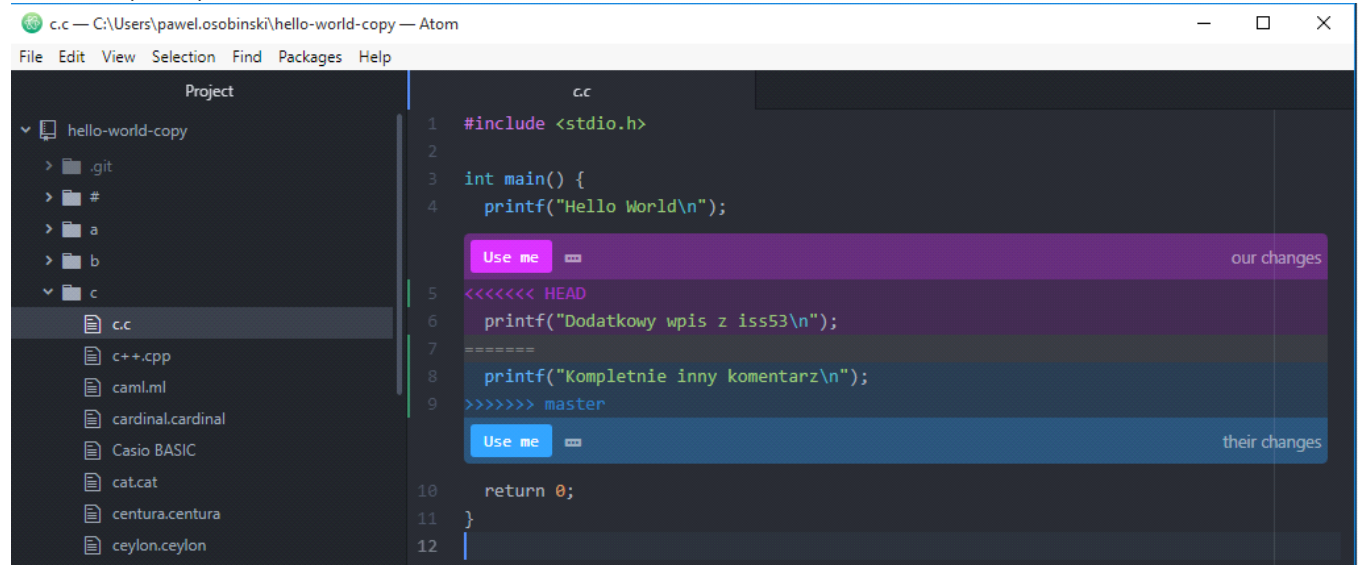
<4.c [dos] (17:51 26/11/2018)1,1 A11 | <4.c [dos] (17:51 26/11/2018)1,1 A11 | ./c/c_REMOTE_1784.c [dos] (17:51 26/11/2018)1,1 A11
#include <stdio.h>

int main() {
    printf("Hello World\n");
    <<<<<< HEAD
    printf("Dodatkowy wpis z iss53\n");
    =====
    printf("Kompletnie inny komentarz\n");
    >>>>>> master
    return 0;
}

~
~
~

```


Metoda 3 (Atom)



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World\n");
5     <<<<<< HEAD
6     printf("Dodatkowy wpis z iss53\n");
7     =====
8     printf("Kompletnie inny komentarz\n");
9     >>>>>> master
10    return 0;
11 }
12
```

Później Git add i commit z Atoma i merge z konsoli :(

<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

Podstawowe Komendy 2.0

Komenda	Opis
<code>git checkout -b <i>branch</i></code>	Tworzy kopie repozytorium (gałąź rozwojową)
<code>git checkout <i>branch</i></code>	Umożliwia przechodzenie między gałęziami
<code>git commit -a -m '<i>comment</i>'</code>	Zapisuje snapshot z fazy stage (wraz z komentarzem)
<code>git branch</code>	Lista gałęzi w projekcie
<code>git pull</code>	Pobierz zawartość ze zdalnego repozytorium i postaraj się połączyć z repozytorium lokalnym
<code>git push -u origin <i>branch</i></code>	Wyślij swoje zmiany oraz pliki do zdalnego repozytorium do konkretnej gałęzi

Zadanie do wykonania

Zadanie 1:

Praca w zespole:

- a) Każdy developer tworzy swoją gałąź rozwojową do wspólnego repozytorium (nazwa = login)
- b) Wprowadza zmiany w swoich plikach/pliku, które później trafiają do repozytorium (w ramach nowej gałęzi)
- c) Commitujecie swoją gałąź na serwer Github-a
- d) Wprowadza zmianę w pliku właściciela oraz pozostałych członków zespołu w repozytorium (dodając swój login np) - tutaj też nowa gałąź
- e) Właściciel pobiera najświeższą wersję z repozytorium i rozwiązuje potencjalne konflikty we własnym pliku
- f) Każdy developer potwarza krok (e) dla swojego pliku

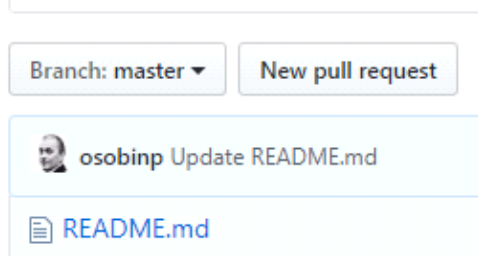
GitHub 301

poniedziałek, 26 listopada 2018

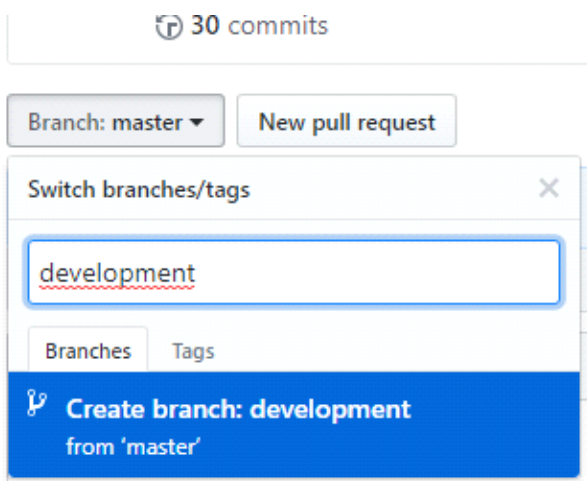
19:20

Tworzenie nowej gałęzi i konflikty

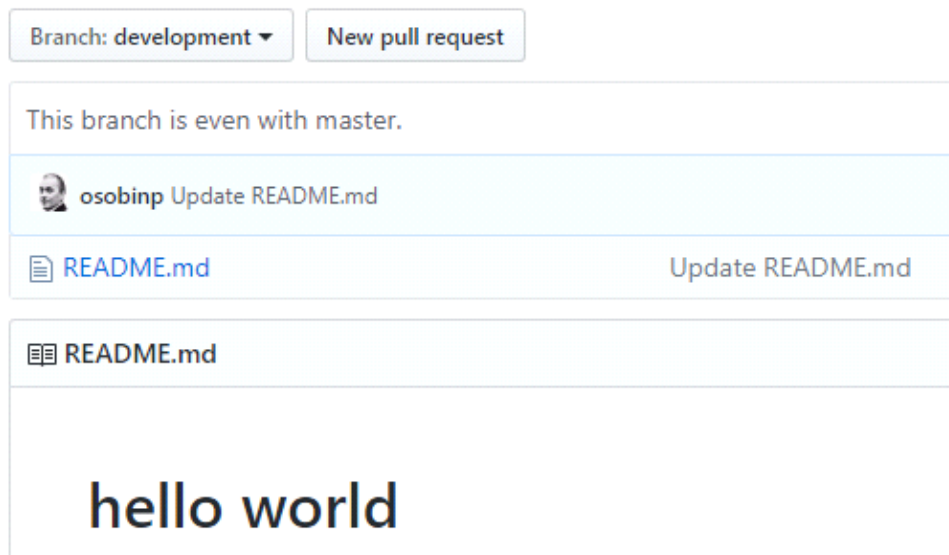
Aby stworzyć nową gałąź z poziomu GitHub-a , wybierz repozytorium następnie kliknij na przycisk Branch: master




W opcjach wpisz nazwę nowej gałęzi





Po wpisaniu nowej nazwy zostaniesz od razu przeniesiony do nowej gałęzi



Edytujemy README.md (lub inny plik)

hello-world / README.md  or cancel

 Edit file

 Preview changes



```
1 # hello world
2 # by Paweł
```


Zatwierdzamy zmianę w nowej gałęzi


Commit changes

aktualizacje README

W nowej gałęzi

p.osobinski@gmail.com  

☒  Commit directly to the `development` branch.

☐  Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes


Cancel


Pull Request


To serce współpracy na GitHub-ie. Kiedy otwierasz Pull Request proponujesz zmianę i wysyłasz prośbę o sprawdzenie / weryfikację czy twój wkład jest poprawny i gotowy do scalenia z kodem głównym (masterem).


Pull Request może pokazać różnicę między branch-em głównym i stworzonymi przez nas zmianami. Ta metoda jest często wykorzystywana w projektach w których strefy czasowe i praca zmianowa ma duże znaczenie lub gdy istnieje nadrzędny programista / architekt, który wymaga weryfikacji kodu. Pull Request-y mogą być tworzone we własnych repozytoriach - tym sposobem można nauczyć się kolaboracji w większych projektach.


Z opcji repozytorium wybieramy Pull requests

 osobinp / hello-world

 Code

 Issues 0

 Pull requests 0



Nowy request (przycisk po prawej stronie okna)

New pull request

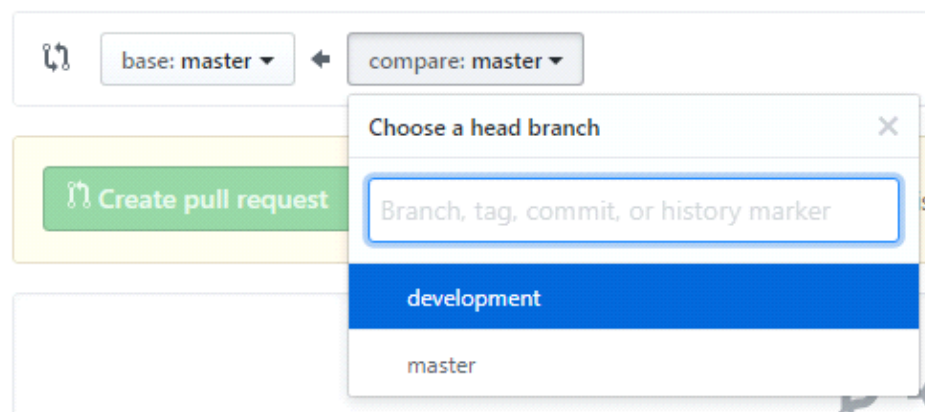
Assignee ▾

Sort ▾

Chcemy wprowadzić zmiany z development branch do master-a , dlatego porównujemy te dwie gałęzie

Compare changes

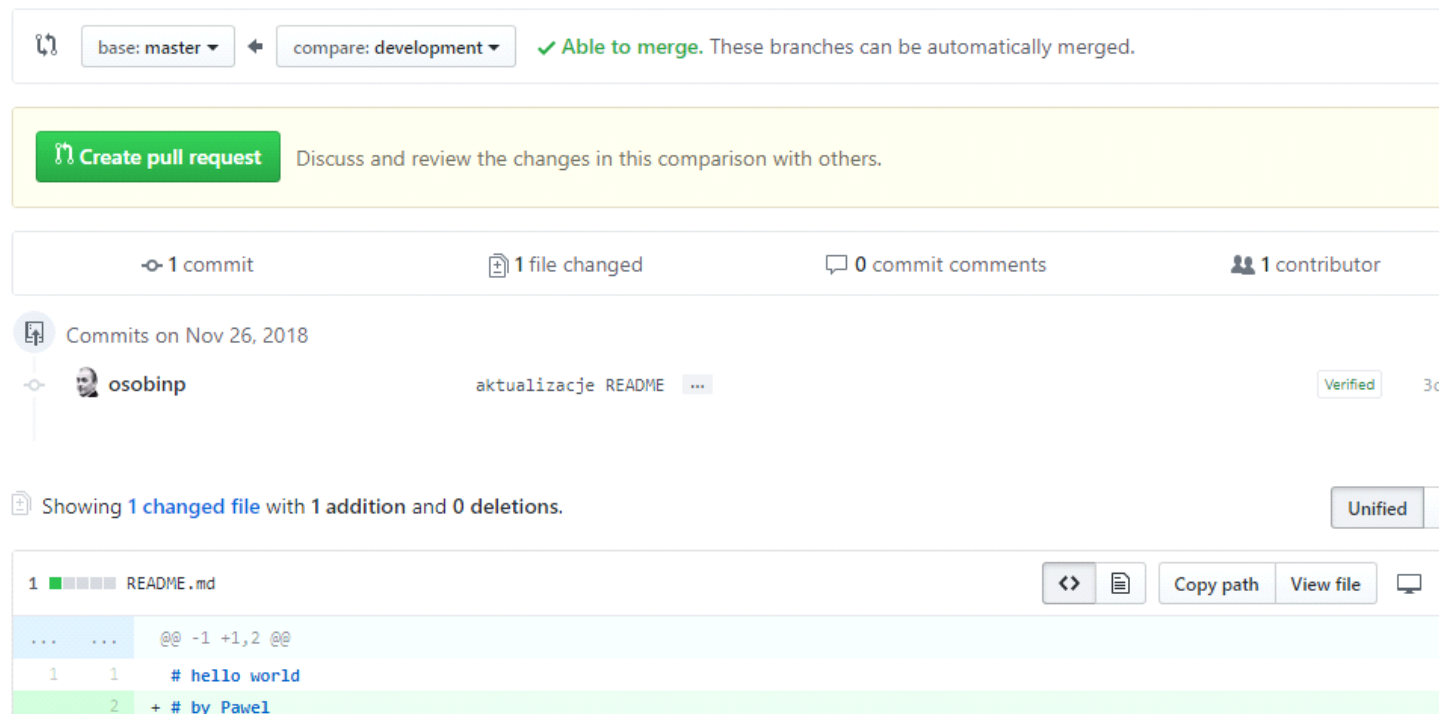
Compare changes across branches, commits, tags, and more below. If you need



Jeśli dodaliśmy coś to nie powinien wystąpić konflikt i możemy stworzyć nowy request. Klikamy Create pull request - w przypadku konfliktu patrz poniżej.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



base: master ← compare: development ✓ Able to merge. These branches can be automatically merged.

Create pull request Discuss and review the changes in this comparison with others.

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Nov 26, 2018

osobinp aktualizacje README Verified

Showing 1 changed file with 1 addition and 0 deletions. Unified

```
1 README.md
... @@ -1,2 @@
1 1 # hello world
2 + # by Pawel
```

Dodajemy nazwę oraz opis zmiany jaką dokonaliśmy


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master

compare: development

✓ Able to merge. These branches can be automatically merged.




aktualizacje README

WritePreview

AA B i “ < > ↻ ⋮ ⋮ ⋮ @ 📎 ↶

Merge do master-a

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

 Styling with Markdown is supported

Create pull request

Tworzymy nowy request, który za chwilę zostanie scalony z gałęzią główną (master). Jeśli nie ma konfliktów klikamy Merge pull request.

Open


osobinp wants to merge 1 commit into master from development

Conversation 0

Commits 1

Checks 0

Files changed 1




osobinp commented just now

Owner + 🧑🏻 ...


Merge do master-a


🔗

 aktualizacje README ...


Verified3c0e893

Add more commits by pushing to the **development** branch on **osobinp/hello-world**.



 Continuous integration has not been set up

Several apps are available to automatically catch bugs and enforce style.

 This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Nasza zmiana zostaje dodana do gałęzi głównej (master). Możemy usunąć gałąź development (Delete branch)

aktualizacje README

Merged osobinp merged 1 commit into master from development just now

Conversation 0 Commits 1 Checks 0 Files changed 1

osobinp commented just now Owner + 😊 ...

Merge do master-a

aktualizacje README Verified 3c0e893

osobinp merged commit 4ceb356 into master just now Revert

Pull request successfully merged and closed Delete branch

You're all set—the development branch can be safely deleted.

Jeśli pojawi się konflikt musimy go rozwiązać, aby to zrobić klikamy Resolve conflicts

Add more commits by pushing to the testing branch on osobinp/hello-world.

This branch has conflicts that must be resolved Resolve conflicts

Use the [web editor](#) or the [command line](#) to resolve conflicts.

Conflicting files

README.md

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Uwzględniamy bądź też odrzucamy zmiany wprowadzone między wartościami w pliku. Gdy już poprawimy plik klikamy Mark as resolved (zaznacz jako rozwiązany)

README.md 1 conflict Prev ^ Next v ⚙️ Mark as resolved

```
1 # hello world
2 <<<<<< testing
3 # by Pawel - second
4 =====
5 # by Pawel - first
6 >>>>>> master
7
```

Po tej operacji kończymy scalenie klikając Commit merge



nd `master` and committing changes ➔ `testing`

Commit merge

README.md

✓ Resolved

```
1 # hello world
2
3 # by Pawel - second
4 # by Pawel - first
5
```

Zakończenie



Q & A