**System kontroli wersji GIT**

**Wstęp**

We wstępie tego szkolenia, chciałbym omówić, czym jest system kontroli wersji, po co z niego korzystać i jakie niesie ze sobą korzyści. Dopiero po uzbrojeniu się w tak solidną wiedzę, przejdziemy do pierwszych zagadnień związanych z tytułowym systemem pracy - GIT.   
  
Tak więc: system kontroli wersji to oprogramowanie służące do śledzenia zmian, np. w kodzie produkcyjnym aplikacji. Dodatkowo umożliwia nam on wprowadzanie i integrowanie wielu zmian jednocześnie, z poziomu maszyn różnych programistów.

Istnieje wiele kontroli wersji, które możemy pogrupować na:

* lokalne - pozwalają przechowywać dane jedynie na lokalnym komputerze
* scentralizowane - oparte na architekturze klient-serwer
* rozproszone - oparte na architekturze peer-to-peer

W dalszej części szkolenia wyjaśnię dokładniej, czym się różnią między sobą dane rodzaje systemów, także czytaj dalej!

**Dlaczego używać systemu kontroli wersji?**

Używanie systemu kontroli wersji jest bardzo ważne przy tworzeniu jakiegokolwiek projektu. Zalety jakie się wiążą z takimi systemami to:

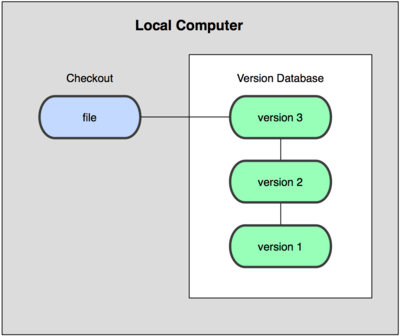
|  |  |
| --- | --- |
| **Możemy współpracować w wiele osób**.  Systemy kontroli wersji wspomagają łatwą pracę na  jednym projekcie przez wiele osób.  Rozwiązują konflikty (czyli sytuacje w których kilka osób pracuje na jednym pliku). | **Systemy pomagają w organizacji wersji projektu**  dzięki możliwości nadawania tagów i cofania naszego kodu do nich jesteśmy w stanie w łatwiejszy sposób trzymać wszystkie wersje swojego programu. |
| **Możemy cofać wprowadzone zmiany** Jesteśmy  w stanie cofnąć stan naszego kodu do  dowolnego miejsca w przeszłości. Dzięki  temu po wprowadzeniu zmian możemy je w  bardzo łatwy sposób wycofać. | **Możemy trzymać backupy naszych projektów**  Dzięki użyciu zewnętrznych repozytoriów  będziemy mieli dostęp do naszego kodu  praktycznie z każdego komputera. Nawet  jeżeli nasz ulegnie awarii. |

**Co tak naprawdę oznaczają te różne rodzaje systemów kontroli wersji?**

1. Lokalny system kontroli wersji

Polega na lokalnym przechowywaniu różnych wersji projektu. Idea jest dość prosta i usprawnia bardzo powszechny nawyk robienia lokalnych kopii zapasowych zasobu. Otóż nieraz zdarza się, że użytkownik zachowuje różne wersje swojego repozytorium, kopiując go w różne miejsce na swoim dysku twardym. Ci sprytniejsi, mają również w zwyczaju rozróżnianie kopii przez zmianę ich nazwy, dodając, np. datę modyfikacji. W takim rozwiązaniu jednak bardzo łatwo o błędy i niedopatrzenia. Zwiększa się bowiem ryzyko wprowadzenia zmian w niechcianej kopii. Przyznaj, jak wiele razy zdarzyło Ci się zapomnieć, w którym znajdujesz się w katalogu i zmodyfikować niechcianą wersję przy pracy z wieloma kopiami pliku/projektu.

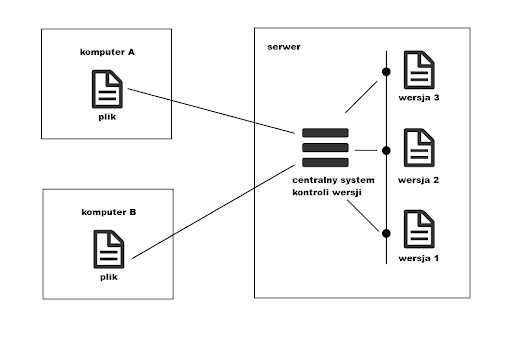
Powyższy problem stanowił główną przesłankę do tego, aby zbudować lokalny system kontroli wersji. System ten wykorzystuje lokalną bazę danych, w której przechowywane są wszystkie zmiany wprowadzane wewnątrz repozytorium (*repozytorium to inaczej projekt*).   
Przykłady: system rcs, który wykorzystywany jest również na wielu współczesnych komputerach.



*Źródło: medium.com*

1. Scentralizowany system kontroli wersji

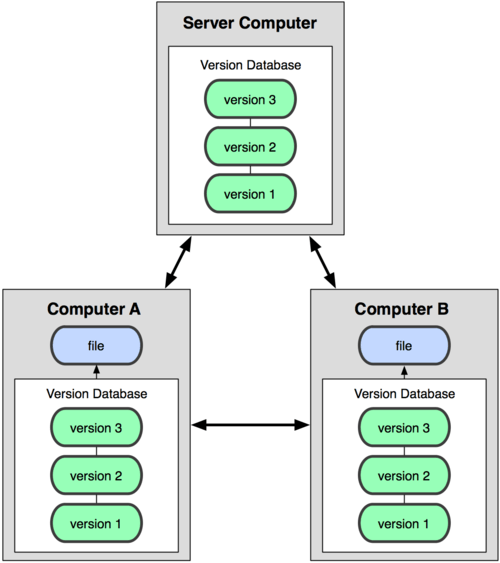
Przykłady: SVM (Subversion) czy Perforce.   
Ten rodzaj systemu powstał, aby zapewnić pracę nad jednym projektem z poziomu oddzielnych komputerów i użytkowników. Tak więc w danym momencie, wszyscy programiści mogą pracować nad jednym projektem, co zapewnia dużą transparentność i przejrzystość wprowadzanych zmian. Deweloperzy bezpośrednio aplikują swoje zmiany do scentralizowanego systemu, o czym od razu są informowani pozostali kontrybutorzy. Z takim rozwiązaniem wiąże się jednak duża wada - awaria serwera powoduje całkowity zator w pracy i brak możliwości rozwijania projektu. Co gorsza, gdyby storage informacji systemu uległ formatowaniu, a programiści nie stworzyliby kopii zapasowych repozytorium, to niemożliwe by było przywrócenie wcześniejszych wersji projektu (na serwerze głównym). Wszystkie dane zostałyby utracone. Dlatego obecnie dużo rozsądniejszym rozwiązaniem może być wykorzystanie rozproszonego systemu kontroli wersji, o którym więcej poniżej.



*Źródło: Blog programistyczny rzemieślnika*

1. Rozproszony system kontroli wersji

W tym rozwiązaniu, użytkownicy mają dostęp do repozytoriów nie tylko w oryginalnej wersji na serwerze, ale również w postaci kopii lokalnej. Co za tym idzie, gdy, np. jeden z serwerów ulegnie awarii, to możliwe jest przywrócenie uszkodzonych danych, pobierając je z “lokalnego schowka” każdego użytkownika. W oparciu o omawiane rozwiązanie pracują takie systemy kontroli wersji jak: GIT, SVK, BitKeeper.



*Źródło: ispro*

**Historia GIT-a**

Jak już zapewne zauważyłeś po tytule szkolenia, my zajmiemy się omówieniem jednego z rozproszonych systemów kontroli wersji - GIT. Obecnie jest on najczęściej wykorzystywanym systemem przez deweloperów i stanowi niezbędną bazę wiedzy do opanowania, aby zacząć działać jako profesjonalny programista. Ucząc się dowolnego nowego zagadnienia, warto spojrzeć na nie z szerszego pola widzenia. Dlatego w szkoleniu tym przedstawię również, skąd narodził pomysł stworzenia GIT-a i kto jest za niego odpowiedzialny.

Stworzenie GIT-a jest związane z Linuxem i jego twórcą - Linus Torvalds-em. Otóż niegdyś kod źródłowy jądra Linuxa był przechowywany przy użyciu popularnego, ale płatnego systemu kontroli wersji BitKeeper SCM. Fakt ten mocno wadził twórcy Unixowego systemu, ponieważ przeczył jego ideologii, że stworzony przez niego system operacyjny powinien być ogólnodostępny i darmowy. Stąd Torvalds postanowił rozpocząć prace nad własnym systemem kontroli wersji - GIT-em.

Cele, jakie Linus postawił przed nowym systemem to:

1. Szybkość.
2. Prosta konstrukcja.
3. Silne wsparcie dla programowania nieliniowego – jednoczesna praca na wielu gałęziach (branchach).
4. Całkowicie rozproszony.
5. Potrafi bardzo efektywnie obsługiwać duże projekty jak np. jądro Linuxa.

W formie ciekawostki, dodam jeszcze, że jak sam twórca przyznaje, GIT jest nazwą, która może znaczyć cokolwiek w zależności od Twojego nastroju, np.

* Jest to losowa kombinacja 3 liter, która jest wymawialna i aktualnie nie używana jako jakieś polecenie w systemie UNIX.
* “**G**lobal **I**nformation **T**racker” – jeśli jesteś w dobrym nastroju i to Ci pasuje. Anioły śpiewają, a światło nagle wypełnia pokój.
* “**G**oddamn **I**diotic **T**ruckload of sh\*t” – kiedy nie działa.

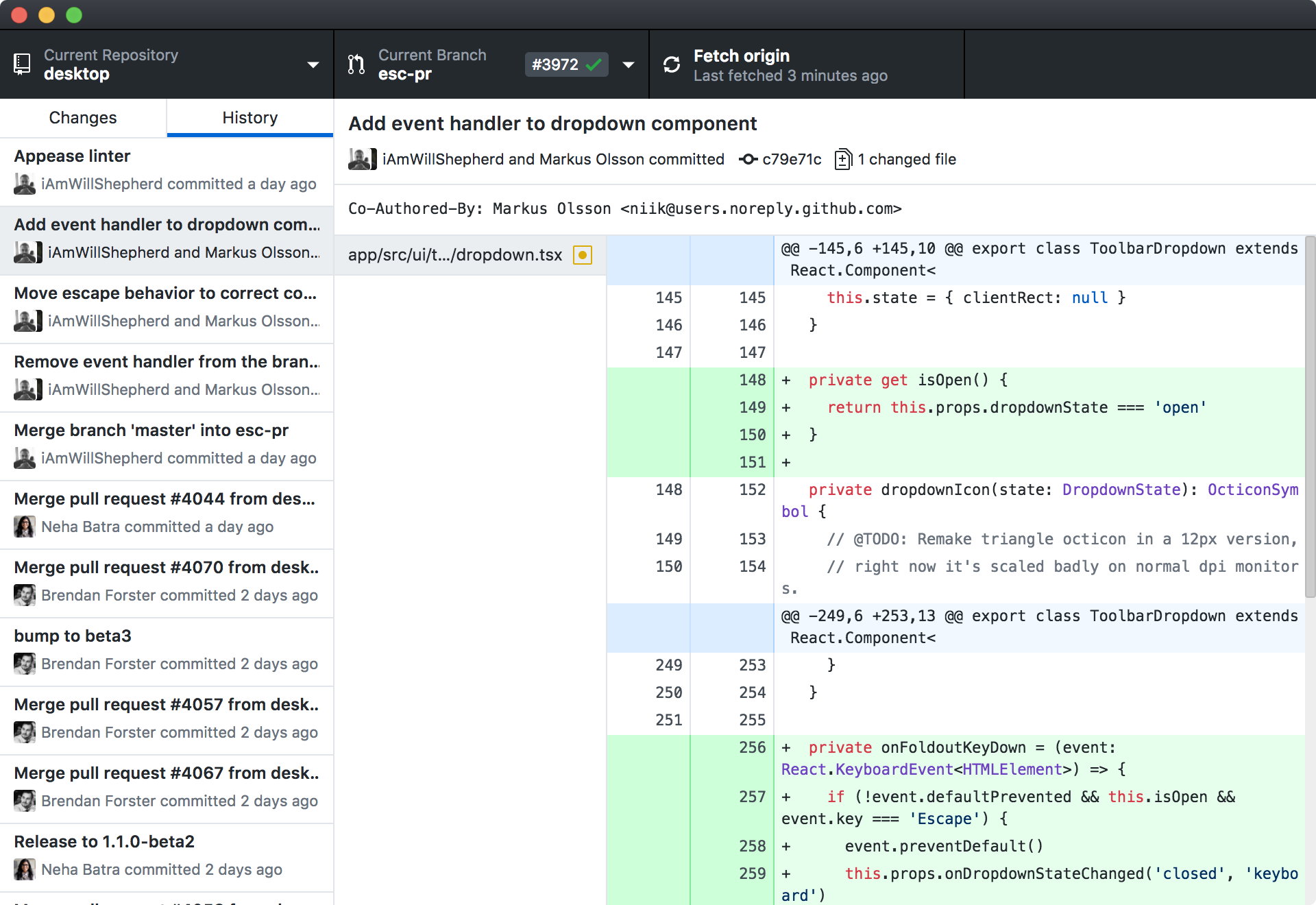
**GIT a GitHub**

Poznaliśmy już krótką historię GIT-a oraz jego cechy, jakimi się wyróżnia. Teraz nadeszła pora, aby raz na zawsze rozwiać wątpliwości, jaka jest różnica między GitHub-em a GIT-em. Jako mentor, zauważyłem, że sporo początkujących osób myli ze sobą te pojęcia i niepoprawnie używa ich naprzemiennie. Otóż zapamiętaj:

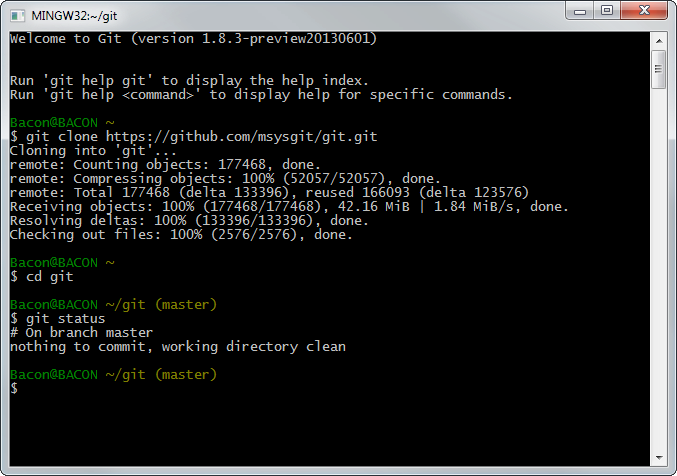
**GitHub to serwis do hostowania repozytoriów (projektów) GIT-a.**Zastąpił on kiedyś używany i niezbyt lubiany Sourceforge. Serwis ten był pełny reklam, a UX (user-expierence) pozostawiał wiele do życzenia. GitHub przyjął się tak mocno, że obecnie jest na nim zarejestrowanych około 56 milionów użytkowników i ilość ta nieustannie wzrasta! To właśnie z nim jest często zintegrowany system GIT i dzięki temu możemy w łatwy sposób upubliczniać projekty.

**Jak obsługiwać GIT-a?**

Omawiany system kontroli wersji możemy przede wszystkim obsługiwać przez CLI (Command Line Interface). Tak więc po zainstalowaniu odpowiedniej paczki, będziemy w stanie w pełni zarządzać naszymi repozytoriami z poziomu terminala. Oczywiście będzie to związane z koniecznością poznania wielu nowych poleceń, ale spokojnie - sky is the limit. Trochę praktyki, a wykorzystanie odpowiednich komend w zależności od potrzeby, wejdzie Ci w nawyk. Oczywiście proces managementu projektów możemy również przeprowadzać przez GUI, np. GitHub Desktop. Wtedy praca głównie będzie się opierała na przeklikiwaniu przez odpowiednie opcje. I choć jest to na pewno bardziej intuicyjne i łatwiejsze do nauki, to nie polecam tego rozwiązania. Dobry programista nie powinien bać się używać terminala, a wszystkie podstawowe polecenia GIT-a powinien mieć w małym palcu. Dlatego w szkoleniu będę Ci prezentował workflow w oparciu o konsolę. To wszystko dla Twojego dobra i odpowiednio efektywnego rozwoju :)



*GitHub Desktop*



*Terminal i GIT*

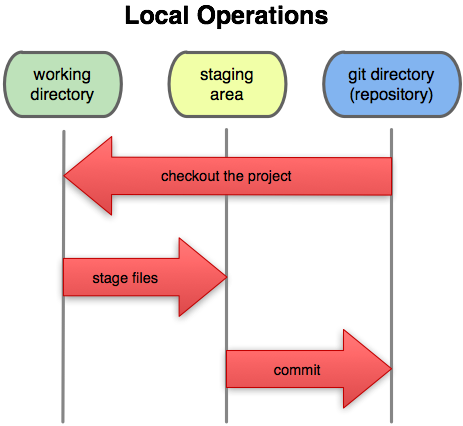
**Podstawy GIT-a**

Jak działa omawiany system kontroli wersji?

Zacznijmy od zrozumienia podstawowych pojęć związanych z GIT-em oraz tego, jak wygląda proces pracy z plikami od strony technicznej.

Na wstępie, muszę zaznaczyć, że GIT jako rozproszony system kontroli wersji podchodzi w zupełnie odmienny sposób do przechowywania plików niż jak to robi Subversion czy Perforce (przykłady centralnych systemów kontroli wersji). Tak więc, poznając omawiany system kontroli wersji, zapomnij o pozostałych. Rozważania zaczniemy od, tzw. trzech stanów przechowywania danych. Bądź uważny, ponieważ jest to jeden z ważniejszych aspektów do zrozumienia w pracy z GIT-em.

Otóż. tak jak wspomnieliśmy na początku szkolenia, GIT przechowuje informacje w sposób rozproszony - główne repozytorium znajduje się na serwerze, a każdy z programistów pracujących na nim, ma dostęp do lokalnej “migawki” takiego projektu. O zaletach tego rozwiązania już wspominałem, ale przypomnę Ci tylko, że takie podejście chroni nas przed trudnościami w pracy i niemożliwością operowania na projekcie, w momencie, gdy nie mamy dostępu do internetu czy serwer główny ulegnie awarii.



Jednak z rozwiązaniem tym wiąże się specyficzny way-of-working, do którego musimy przywyknąć. Chcąc bowiem zaaplikować zmiany przeprowadzane u nas lokalnie na komputerze do głównego serwera, konieczne jest przejście przez, tzw. trzy stany plików systemu GIT-a. Od **working directory** (lokalnego miejsca pracy) przez **staging area** (wspomniane pośrednie miejsce zapisu; umożliwia rozproszenie systemu) i ostatecznie do **repository** (główne miejsce, w którym przechowywany jest projekt).

1. Working directory - katalog roboczy Twojego projektu. W nim przechowywane są wszystkie metadane (np. informacje powiązane z GIT-em). To właśnie ten katalog jest pobierany podczas klonowanie repozytorium z serwisu na inny komputer
2. Staging area (nazywana również indeksem) - odzwierciedla ją prosty plik, który zawiera informacje o wprowadzanych zmianach. Obszar ten cechuje się tym, że dodawane są do niego ostatnio wprowadzone zmiany (tzw. commit-y). Wyobraź sobie sytuację, że w piątek pracujesz nad zmianami do projektu. Musisz zrealizować 2 feature-y. Nagle wybija 16sta. Pierwszy feature skończyłeś, ale do zamknięcia drugiego brakuje kilku rzeczy do zakodowania. Dodatkowo w poniedziałek chcesz jeszcze raz sprawdzić stworzone zmiany w projekcie, dlatego czujesz, że nie jesteś jeszcze gotowy do uploadowania zmian na główny serwer (do repository). Wówczas najrozsądniejszym rozwiązaniem będzie dodanie zmian przeprowadzonych w pierwszym featurze do obszaru stage, a pozostałe modyfikacje pozostawić lokalnie.
3. Repository - główne miejsce przechowywania zmian. Jest ono reprezentowane przez katalog .git. Katalog ten umożliwia nam tworzenie “migawki” projektu i przechowywanie go w sposób rozproszony.

Podstawowy workflow pracy z GIT-em wygląda następująco:

1. Dokonujemy zmian wewnątrz katalogu roboczego.
2. Zmodyfikowane pliki oznaczamy jako śledzone (przenosząc je do statusu stage).
3. Dokonujemy zatwierdzenia (commit) wszystkich śledzonych zmian - zawartość plików z przechowalni jest przenoszona do repozytorium.
4. Tak utworzono repozytorium możemy pushować (uploadować) na główny serwer zintegrowany z GIT-em.

Zdaję sobie sprawę, że jest tutaj sporo nowej wiedzy. Dlatego podsumujmy wszystkie wyżej wymienione zagadnienia w treściwym słowniczku:

1. Commit - zatwierdzenie zmian i przeniesienie ich ze stanu staged do commited.
2. Stage - nazywany również indeksem; jeden ze stanów plików - między stanem working directory a repository.
3. .git directory - miejsce lokalnego przechowywania całego repozytorium; ma ono postać ukrytego katalogu plików

**Instalacja GIT-a**

Pracę z omawianym systemem kontroli wersji zaczniemy od instalacji odpowiedniego pakietu. W zależności od platformy, na jakiej działasz, operacja będzie przebiegała w różny sposób, dlatego zapoznaj się z poniższymi instrukcjami:

**Instalacja na Windows**

Najłatwiejszym i najskuteczniejszym sposobem na instalację GIT-a, będzie skorzystanie z instalatora. Pobrać go możesz stąd: <https://git-scm.com/downloads>. Proces instalacji jest dość złożony, jednak niech CIę to nie przestraszy. “Przeklikaj się” przez wszystkie opcje, pozostawiając je domyślnie ustawione. Dzięki temu prosto przebrniesz przez proces instalacji i zapewnisz sobie prawidłową konfigurację.

**Instalacja na Linux**

W celu instalacji GIT-a na systemach Unixowych (dystrybucja Debian), to w terminalu:

1. Wykonaj polecenia:

$ sudo apt-get update

$ sudo apt-get install git

2. Sprawdź, czy GIT został poprawnie zainstalowany:

$ git --version

Jeżeli ujrzysz komunikat podobny do poniższego...

git version 2.9.2

… to znaczy że wszystko przebiegło zgodnie z oczekiwaniami!

**Instalacja na MAC OS**

1. Paczkę z oprogramowaniem pobierz stąd: [git-osx-in „staller - Browse Files at SourceForge.net](https://sourceforge.net/projects/git-osx-installer/files/)

2. Przejdź przez proces instalacji zgodnie z instrukcjami

3. Sprawdź, czy GIT został poprawnie zainstalowany:

$ git --version

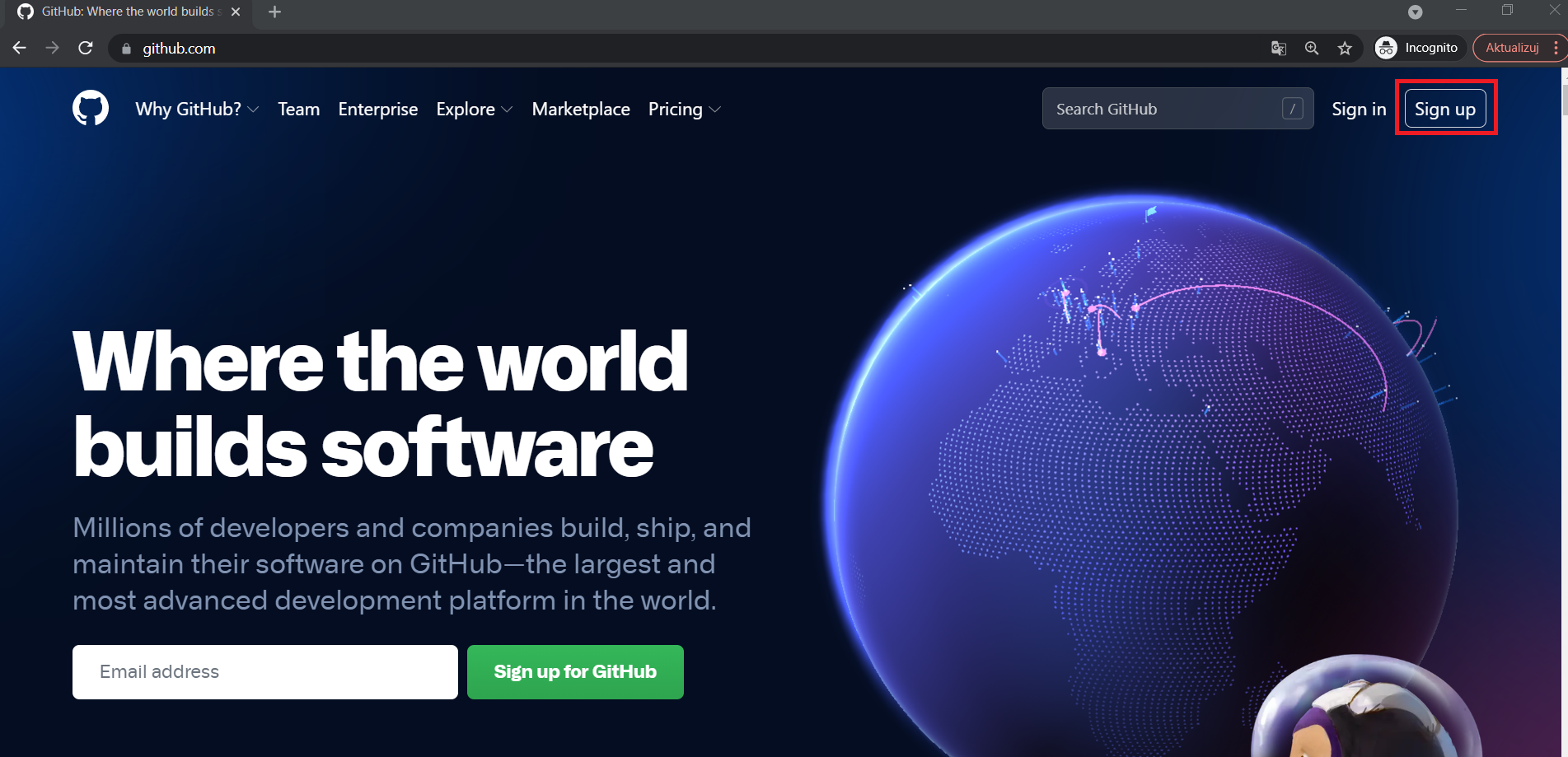
Jeżeli ujrzysz komunikat podobny do poniższego...

git version 2.9.2

**Utworzenie konta i pierwszego repozytorium na Github**

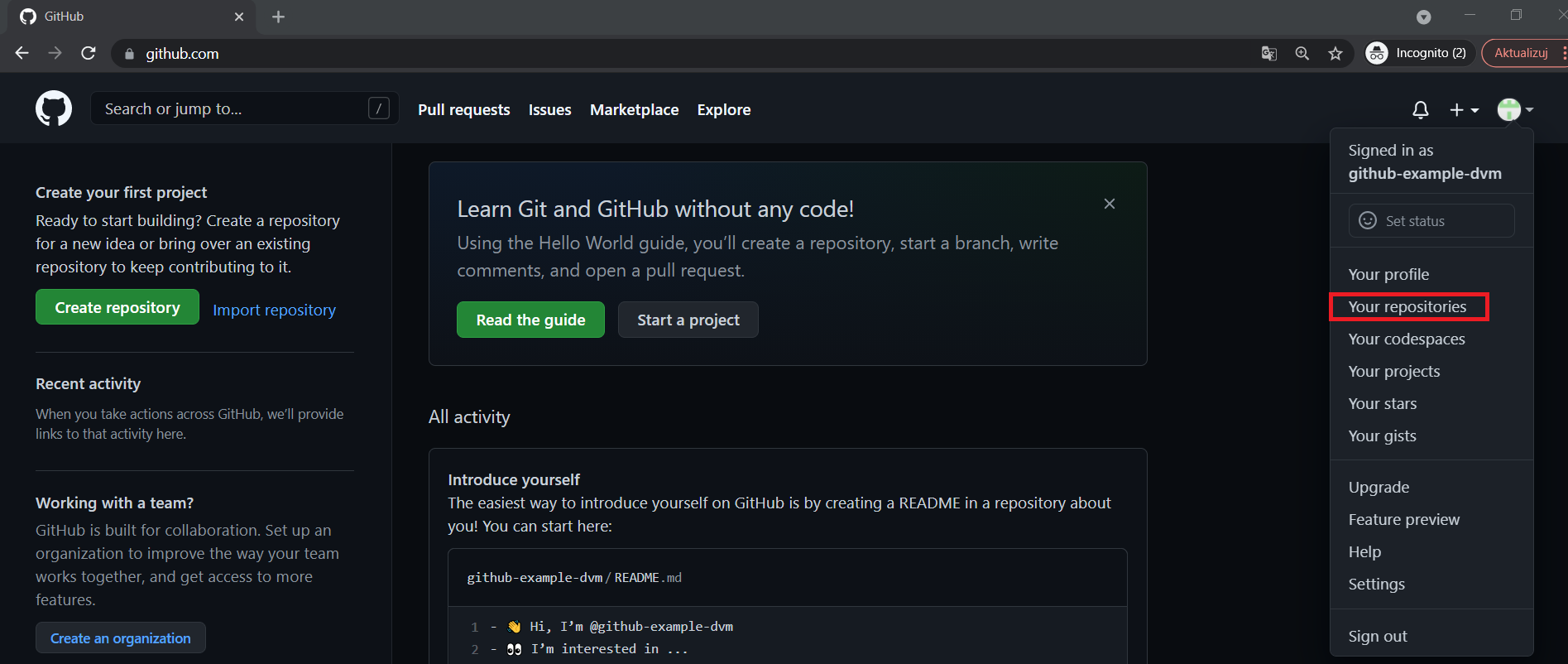
Na zakończenie, zajmiemy się utworzeniem własnego konta na GH i przypiszemy do niego pierwsze repozytorium. Dzięki temu, przygotujemy miejsce do uploadowania swoich lokalnych zmian. Wówczas zostanie nam tylko zapoznanie się z podstawowymi poleceniami GIT-a i jego workflow (o czym więcej w następnych szkoleniach).

Przejdź zatem na poniższą stronę:



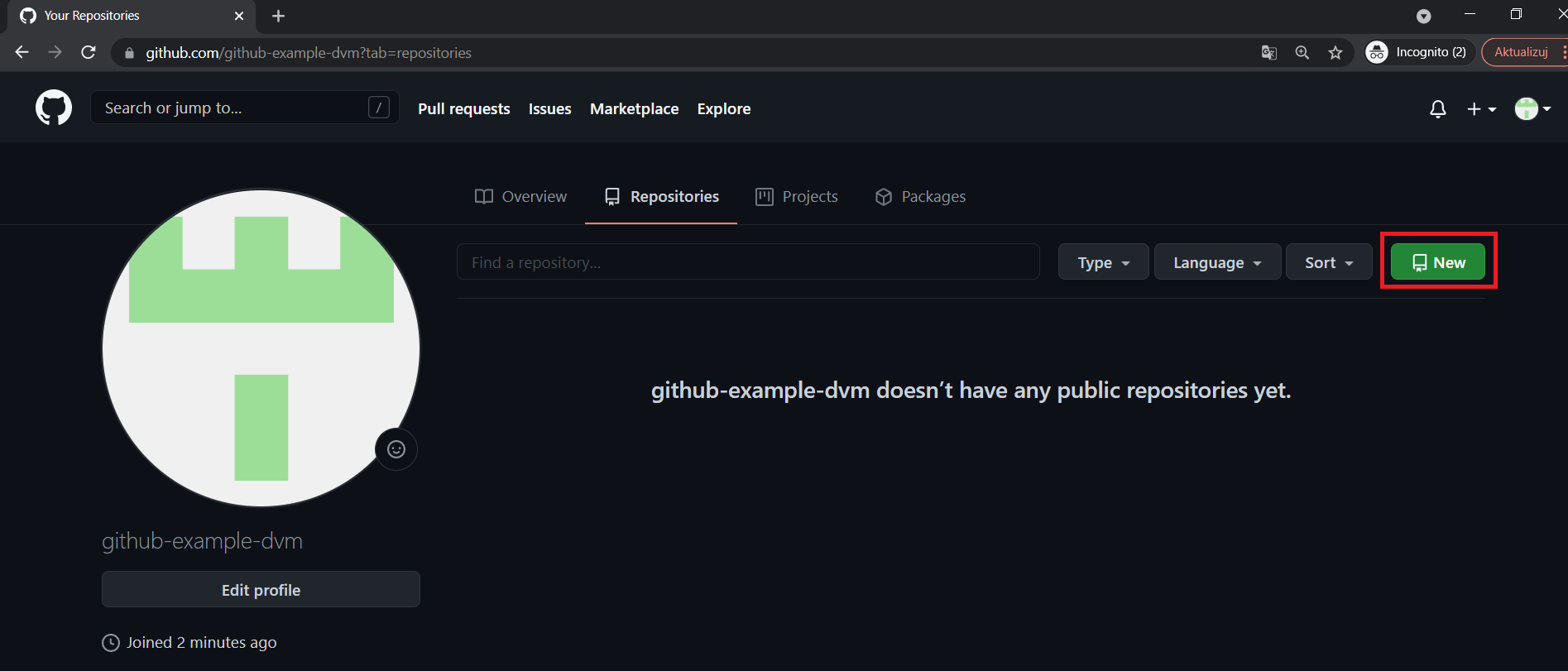
I kliknij w znajdujący się na niej przycisk Sign Up. Następnie przejdź przez proces rejestracji.

Po poprawnym założeniu konta i zalogowaniu się na nie, w prawym górnym roku będziesz miał dostęp do awatara swojego profilu. Kliknij w tę ikonę, a ujrzysz następujące pola do wyboru:



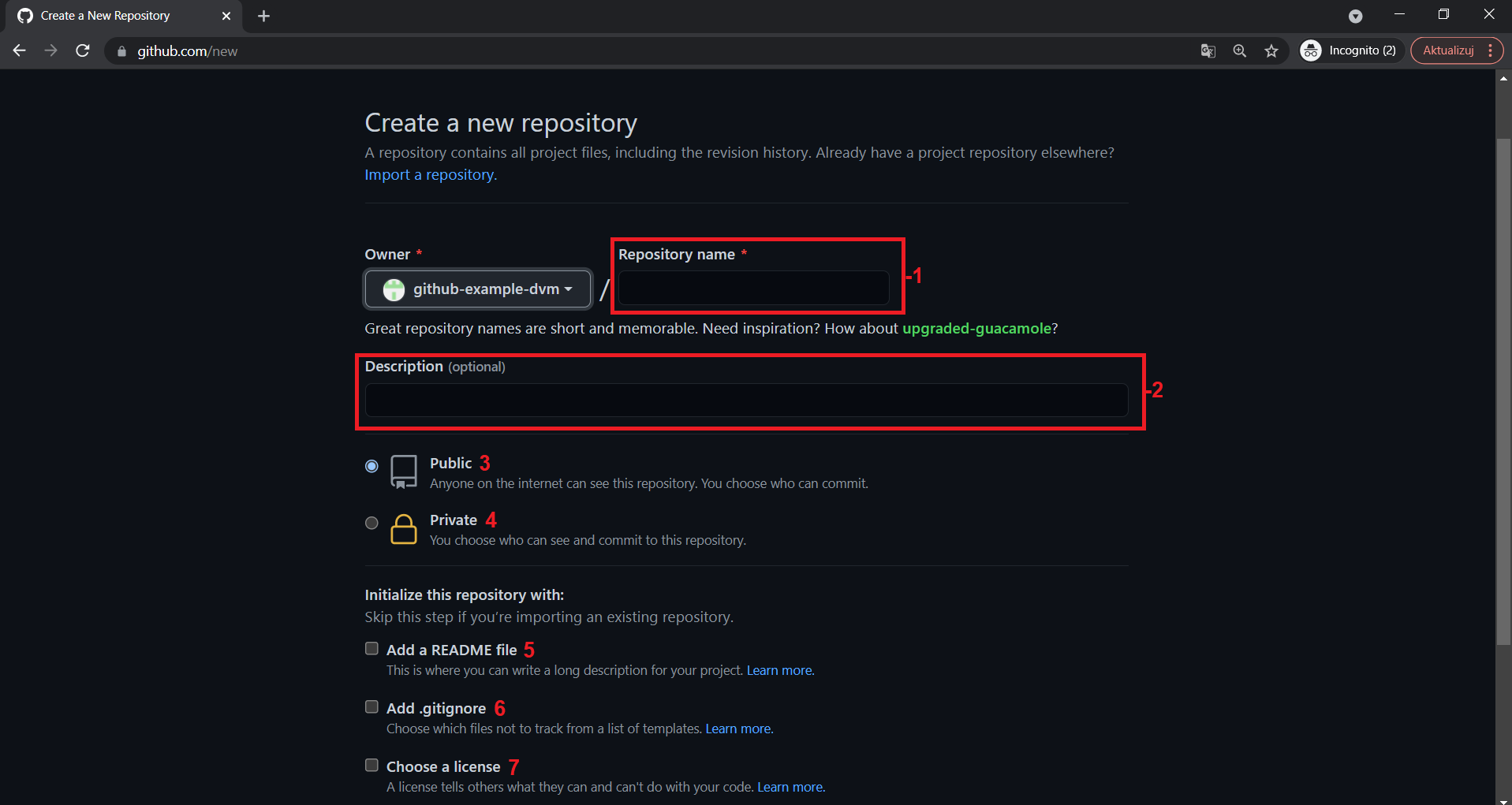
Nas najbardziej interesuje zakładka Your repositories, która umożliwia Ci dostęp do listy utworzonych projektów.

Po kliknięciu w nią i zostaniu przekierowanym do nowego odnośnika, ujrzysz:



To miejsce będzie się wypełniało listą kolejnych przesyłanych przez Ciebie projektów. Aby utworzyć swoje pierwsze repozytroium (póki co puste), kilknij w zaznaczony przycisk New.

Zostaniesz wówczas przekierowany do strony, na której będziesz mógł podać informacje o tworzonym repozytorium.



1 - to nazwa repozytorium, do którego będziemy dodawali pliki, np. Tetris-Game-C++

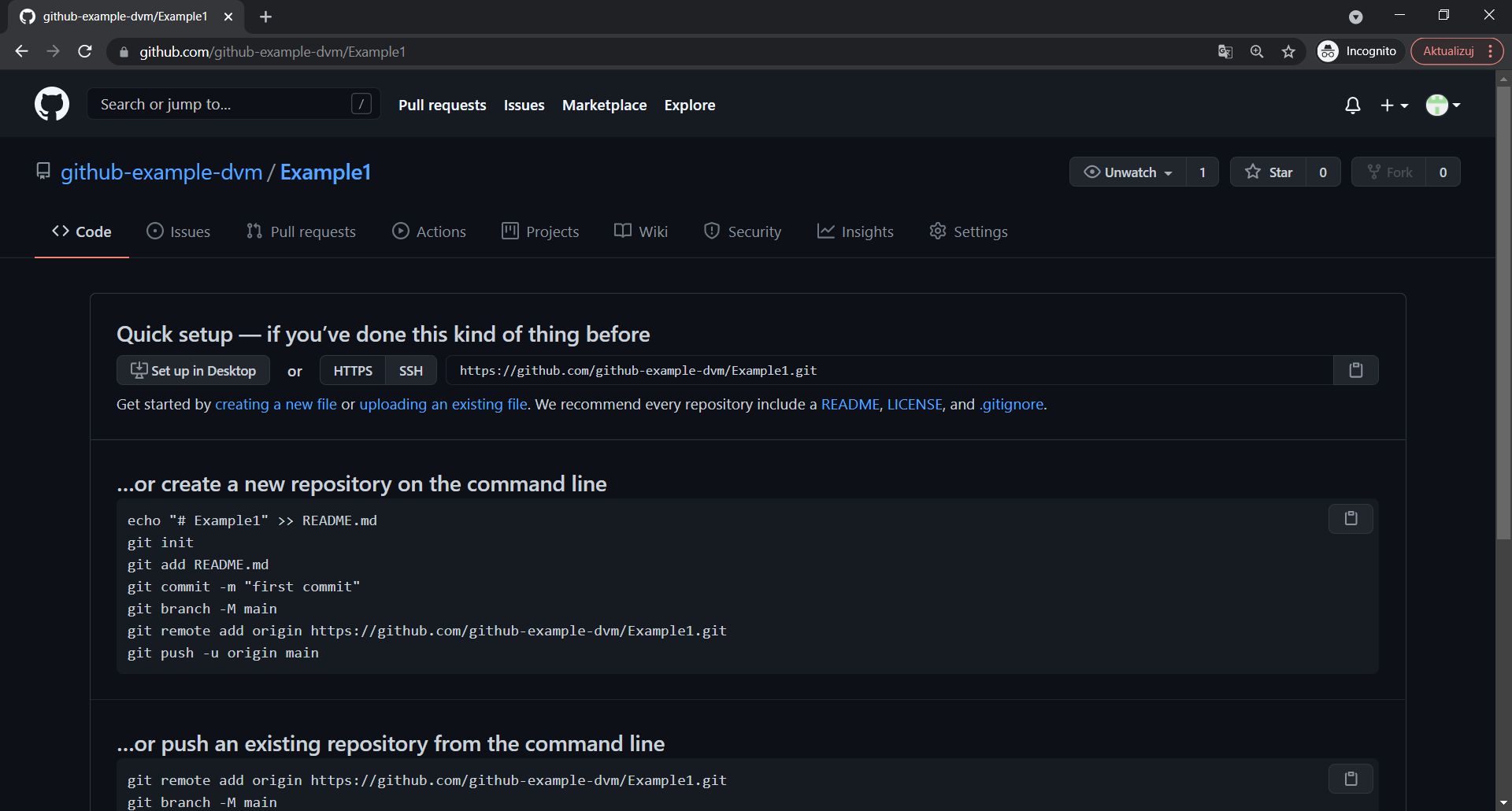
2 - wartość opcjonalna; umożliwia dodanie krótkiego opisu do udostępnianego repozytorium   
  
3, 4 - to opcje pozwalające określić widoczność Naszego repozytorium; przy wybraniu opcji Public - projekt będzie ogólnodostępny i możliwy do wyszukania przez dowolnego użytkownika, private natomiast zrobi projekt widoczny tylko dla nas i wybranych kontrybutorów

5 - opcja określająca, czy w naszym repozytorium ma się również znaleźć plik README; jest to plik, w którym zazwyczaj programiści krótko opisują swój projekt, informują, jakie wykorzystali technologie oraz jak poprawnie uruchomić ich program

6 - czasami nie będziemy chcieli udostępniać wszystkich plików z naszego projektu; wówczas, aby zignorować niektóre z nich i nie uploadować ich na GH, możemy utworzyć plik .gitignore, w którym wylistujemy nazwy zasobów do pominięcia; więcej o tym powiem w dalszych szkoleniach

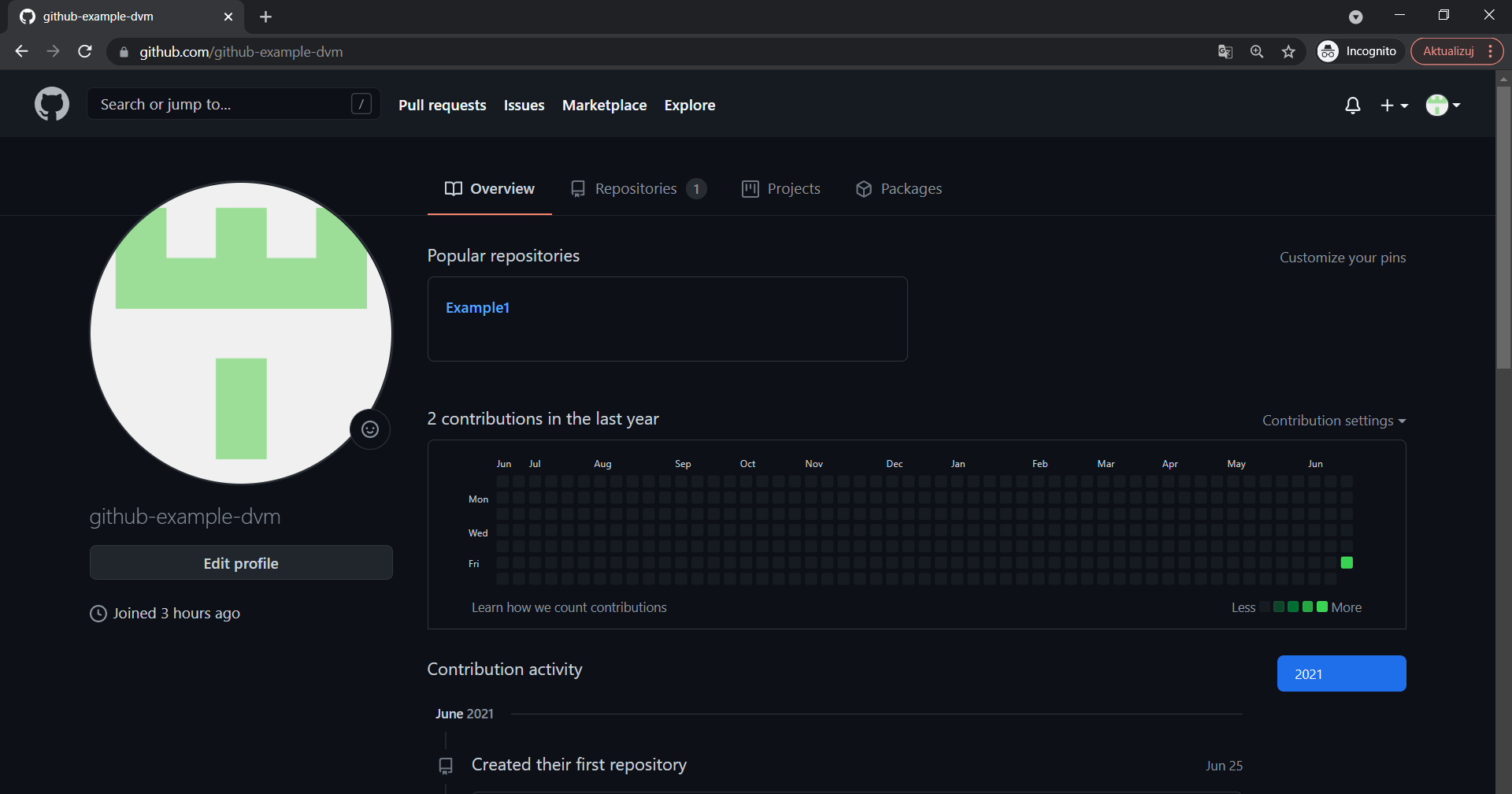
7 - na koniec możemy określić, na bazie jakiej licencji chcemy udostępnić nasz projekt; GH bazuje głównie na repozytoriach open-sourcowych, ale możemy to doprecyzować, wybierając, np. licencję MIT, GNU, Apache itd.

Gdy wybierzesz wszystkie odpowiadające Ci opcje, kliknij przycisk Create Repository. Twoje repozytorium zostanie utworzone, a GH powinien przenieść Cię do następującej strony:



Ukazane instrukcje przedstawiają Ci, jakie polecenia możesz wykonać, aby udostępnić swój pierwszy projekt do repozytorium. Na tę chwilę zignoruj te wskazówki Quick Setup-u, pokażę Ci w dalszych szkoleniach, jak udostępniać pierwsze pliki na GH.

Gdy teraz wrócisz do zakładki repositories, na liście ujrzysz odnośnik do właśnie utworzonego projektu Example1:



Repozytorium Example1 będziemy przy użyciu systemu kontroli wersji GIT wypełniali kolejnymi plikami. Będzie to jednak zadanie na następne lekcje, w trakcie których poznamy już konkretny workflow pracy z GIT-em. Musimy jeszcze tylko zintegrować GIT-a z utworzonym kontem na GH.

**Podstawowa konfiguracja narzędzia**

Po zainstalowaniu GIT-a i utworzeniu konta na serwisie, uruchom terminal (lub jeżeli korzystasz z Windowsa, to CMD/PowerShella) i przejdź przez ten krótki proces konfiguracji swojego konta. Będzie to ustawienie potrzebnych informacji, aby narzędzie mogło dodawać informacje o zmianach w kodzie. Sprecyzujemy tutaj nazwę konta oraz email, który ma być wykorzystywany do późniejszego publikowania zmian.

**Skonfigurowanie nazwy użytkownika:**

git config --global user.name “github-example-dvm”

**Skonfigurowanie email-a użytkownika:**

git config --global user.email “[kacper@devs-mentoring.pl](mailto:kacper@devs-mentoring.pl)”

**UWAGA:**  
Pamiętaj, aby podane dane były zgodne z tymi, które podałeś przy tworzeniu konta w serwisie.

**Sprawdzanie ustawień**

Jeżeli będziesz chciał sprawdzić bieżące ustawienia, to wykonaj polecenie git config --list. Wówczas wyświetli się pełna konfiguracja, np:

$ git config --list

user.name=github-example-dvm

user.email=kacper@devs-mentoring.pl

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

…

Można również sprawdzić jaka jest rzeczywista wartość zmiennej o konkretnej nazwie za pomocą polecenia git config {zmienna}:

$ git config user.name

github-example-dvm