Kontrola wersji w git.

v.2.2

Coders Lab
SZKOŁA PROGRAMOWANIA

Co to znaczy „kontrola wersji”?

System kontroli wersji (ang. *version control system*, w skrócie *VCS*), to oprogramowanie służące do **śledzenia zmian** np. w kodzie źródłowym programu.

Istnieje mnóstwo systemów kontroli wersji (CVS, Subversion, Mercurial, Perforce i wiele innych).

Najbardziej rozpowszechnionym i popularnym VCS używanym przez branżę IT jest **git**.

Dlaczego używamy kontroli wersji?

Używanie systemu kontroli wersji jest bardzo ważne przy tworzeniu jakiegokolwiek projektu. Możemy dzięki niemu:

- **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).
- **Pomagają w organizacji wersji projektu** – dzięki możliwości nadawania **tagów** i cofania naszego kody do nich jesteśmy w stanie w łatwiejszy sposób trzymać wszystkie wersje swojego programu.
- **Cofanie wprowadzanych zmian –** jesteśmy w stanie cofnąć stan naszego kodu do dowolnego miejsca w przeszłości. Dzięki temu po wprowadzeniu zmian możemy je w bardzo łatwy sposób wycofać.
- **Trzymać backupy naszych projektów –** dzięki użyciu zewnętrznych repozytoriów będziemy mieli dostęp do naszego kodu praktycznie z każdego komputera. Nawet jeżeli nasz ulegnie awarii.

Co to jest GIT?

GIT jest najpopularniejszym systemem kontroli wersji. Stworzony w roku 2005 przez Linusa Torvalda (twórcę Linuxa) jako system kontroli do wspomagania projektów Open Source.

Jest typowym systemem rozproszonym. Oznacza to że pełna historia projektu znajduje się na każdym komputerze który posiada repozytorium z ty projektem.

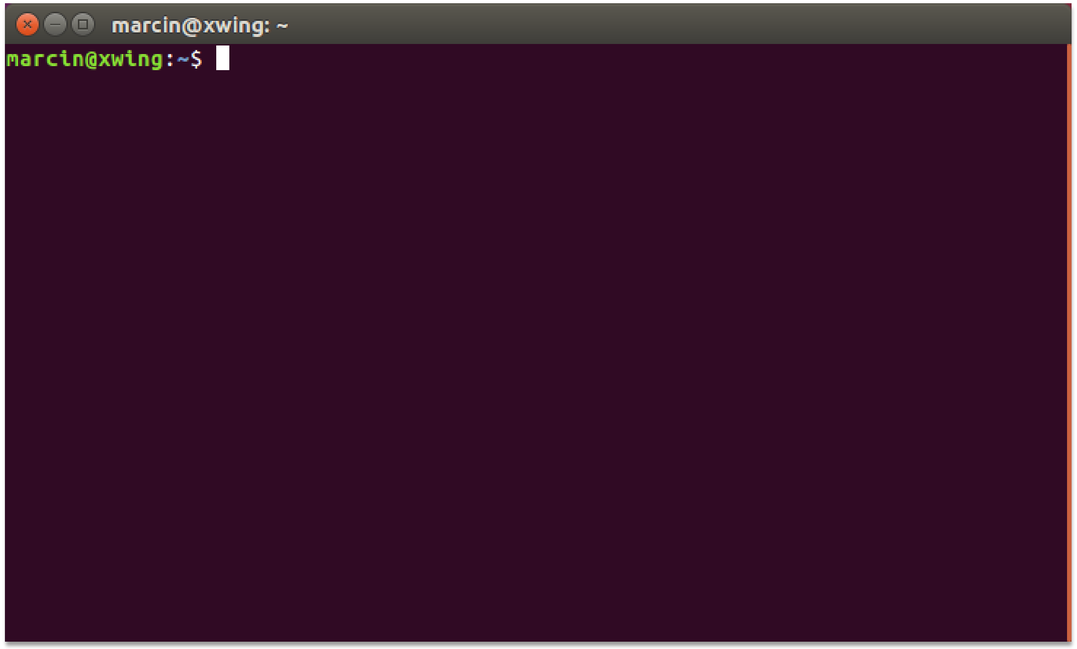
Git bardzo dobrze sobie radzi z wszelkimi plikami tekstowymi, gorzej z plikami binarnymi (zdjęciami, plikami pdf, itp).

Uruchamianie terminala

Jak otworzyć terminal?

Jeśli jesteś użytkownikiem Ubuntu:

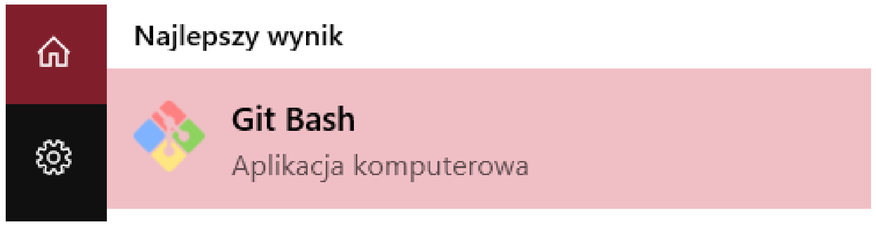
Otwórz terminal kombinacją klawiszy
Ctrl-Alt-T



Jak otworzyć terminal?

Jeśli jesteś użytkownikiem Windows:

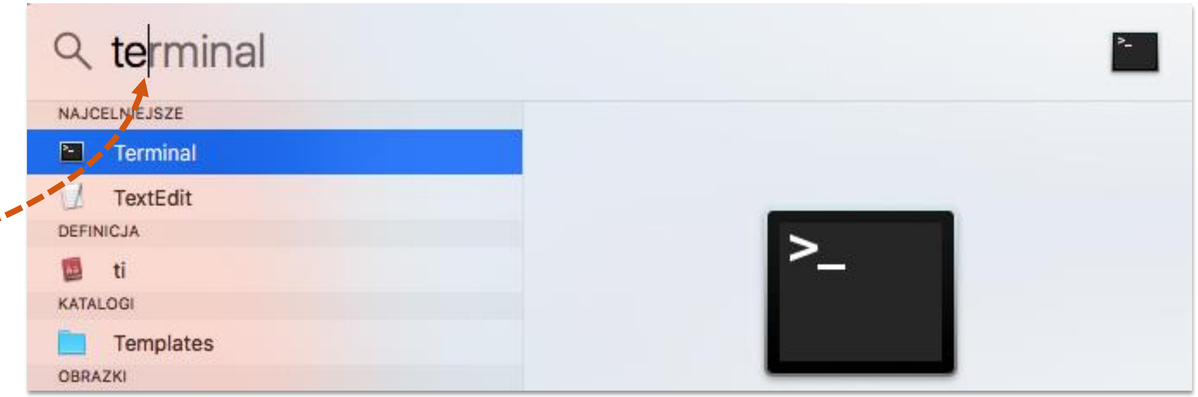
1. Jeśli masz zainstalowany git bash to wystarczy go otworzyć.
2. Jeśli nie to zainstaluj (jak go zainstalować jest opisane w następnym rozdziale).

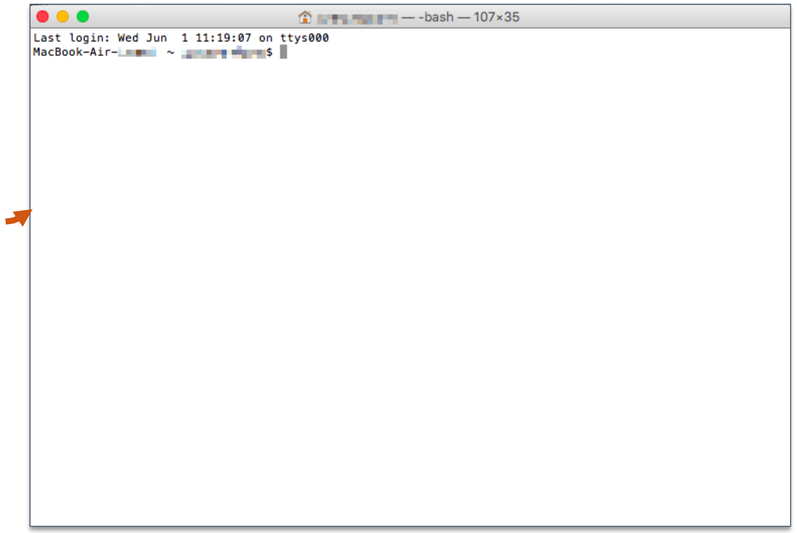


Jak otworzyć terminal?

Jeśli jesteś użytkownikiem MacOS:

- Wciśnij kombinację klawiszy **Cmd+Spacja**,
- W oknie **Spotlight** zacznij wpisywać „**terminal**”,
- Potwierdź klawiszem **Enter.**





Instalacja gita

Instalacja gita

Jeśli jesteś użytkownikiem Ubuntu:

- Otwórz terminal
  W terminalu wpisz następujące komendy:

  sudo apt-get update

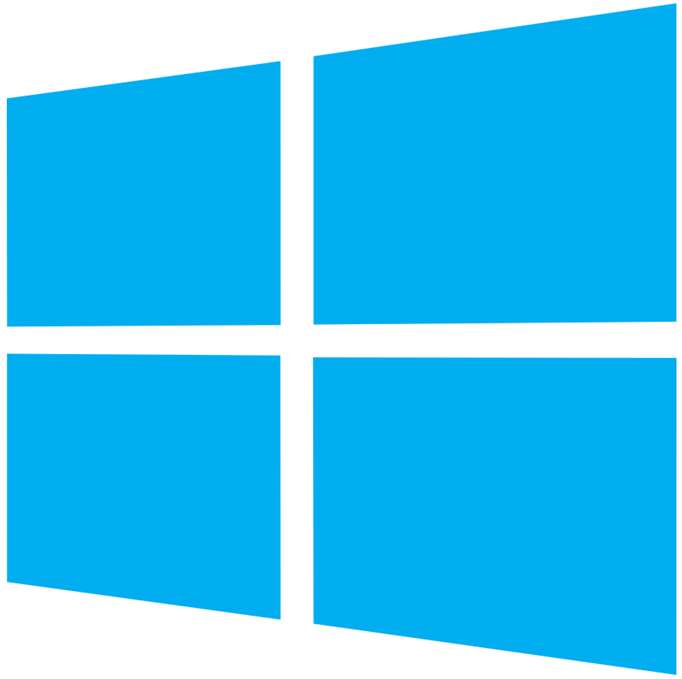
  sudo apt-get install git

Osoby korzystające z systemu operacyjnego dostarczonego przez nas mają już zainstalowany Git.

Instalacja gita

Jeśli jesteś użytkownikiem Windows:

- Wejdź na stronę https://git-scm.com/download/win *(program zacznie pobierać się automatycznie)*
- Uruchom pobrany program
- Podczas instalacji wybieraj proponowane opcje.



Instalacja gita

Jeśli jesteś użytkownikiem MacOS:

- Wejdź na stronę https://git-scm.com/download/mac *(program zacznie pobierać się automatycznie)*
- Uruchom pobrany program
- Podczas instalacji wybieraj proponowane opcje.

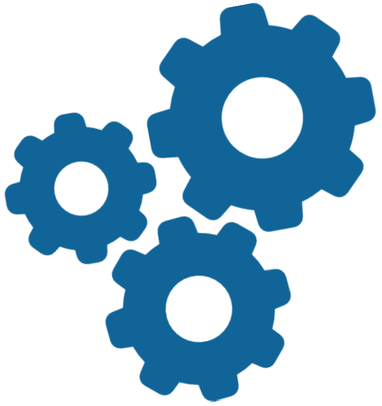
Konfiguracja gita

Podstawowa konfiguracja narzędzia:

Git potrzebuje Twoich danych, by dodawać je do informacji o zmianach w kodzie. W tym celu dodaj do konfiguracji Twoje imię i nazwisko oraz email.

Uruchom terminal *)
i wpisz następujące komendy:

```
git config --global user.name "Imię i nazwisko"
git config --global user.email "adres email"
```



```
marcin@xwing:~/workspace/sklep$ git config --global user.name "Jan Nowak"
marcin@xwing:~/workspace/sklep$ git config --global user.email "jan.nowak@coderslab.pl"
marcin@xwing:~/workspace/sklep$
```

Konfiguracja gita

Edytor komunikatów.

Czasami zdarza się, że musisz wpisać komentarz do zmian interaktywnie, przy użyciu edytora tekstu. Standardowym edytorem tekstu, używanym przez Gita jest **Vim** *). Jeśli chcesz zmienić edytor **), wykonaj instrukcje z kolejnych slajdów.

***)** Vim, to jeden ze starszych i podstawowych edytorów tekstów dla systemów UNIX i podobnych. Bardzo użyteczny, ale trudny w obsłudze, zwłaszcza dla początkujących. (https://pl.wikipedia.org/wiki/Vim)

****)** Zaufaj nam.

Konfiguracja gita

Ubuntu: otwórz terminal i wpisz:

```
git config --global core.editor nano
```

Nano jest wygodnym edytorem tekstu, uruchamianym w terminalu.

Więcej o Nano https://pl.wikipedia.org/wiki/Nano_(program)

Konfiguracja gita

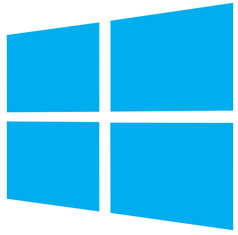
Mac: otwórz terminal i wpisz:

```
git config --global core.editor nano
```

Nano jest wygodnym edytorem tekstu, uruchamianym w terminalu.

Więcej o Nano https://pl.wikipedia.org/wiki/Nano_(program)

Konfiguracja gita



Windows: otwórz terminal

Wpisz komendy (po wpisaniu każdej wciśnij enter):

git config --global core.editor notepad

git config --global format.commitMessageColumns 72

Komendy te nie zwracają żadnych informacji (po wciśnięciu enter w terminalu pojawi się nowa linijka w której możesz wpisać następną komendę).

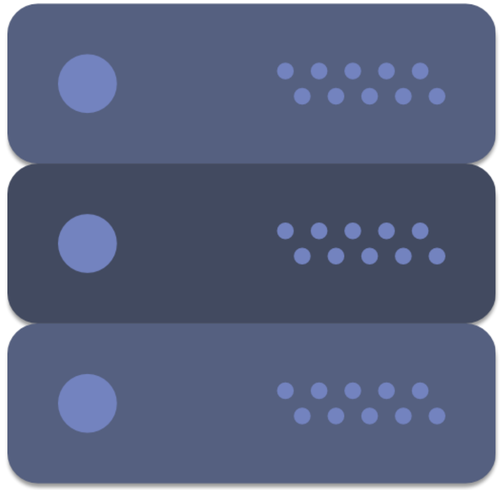
Jak działa kontrola wersji?

Jak działa kontrola wersji?

Wyobraź sobie system kontroli wersji jako bazę danych.

Gdy piszesz program, w każdym momencie możesz zapisać do tej bazy aktualny stan swojego kodu. Gdy później sprawdzisz ten stan (nazywajmy go od tego momentu *„wersją”*), oprogramowanie pokaże Ci, czym aktualna wersja różni się od poprzedniej.

Dzięki temu możesz prześledzić, co zmieniło się w Twoim kodzie.



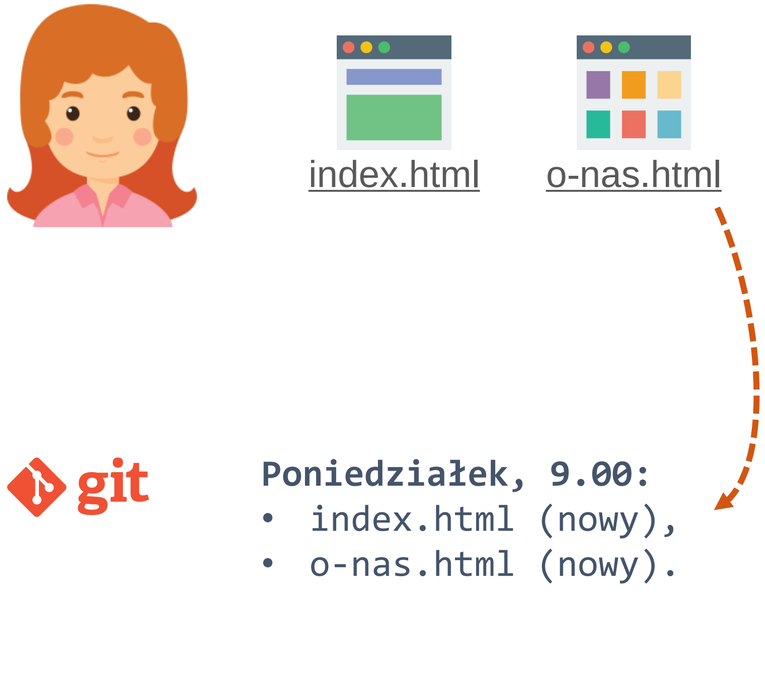
Jak działa kontrola wersji?

Poniedziałek, 9.00.

Agata tworzy nowe pliki:

- **index.html,**
- **o-nas.html.**

Umieszcza je w systemie kontroli wersji i opisuje zmianę.



Jak działa kontrola wersji?

Środa, 14.00.

Agata modyfikuje plik:
- **index.html.**

Agata tworzy dwa nowe pliki:
- **regulamin.html,**
- **smieszny-kotek.png.**

Umieszcza je w systemie kontroli wersji i opisuje zmianę.

index.html

regulamin.html

smieszny-kotek.png

git

Środa, 14.00:
- index.html (modyfikacja),
- regulamin.html (nowy),
- smieszny-kotek.png(nowy).

Poniedziałek, 9.00:
- index.html (nowy),
- o-nas.html (nowy).

Jak działa kontrola wersji?

Piątek, 16.55. (do weekendu zostało 5 minut)

Marcin modyfikuje plik:

- **regulamin.html.**

Z pliku **regulamin.html** usuwa większość treści i zmienia kompletnie wygląd.

Umieszcza je w systemie kontroli wersji i opisuje zmianę.

regulamin.html

git

Piątek, 16:55:
- regulamin.html (modyfikacja)

Środa, 14.00:
- index.html (modyfikacja),
- regulamin.html (nowy),
- smieszny-kotek.png(nowy).

Poniedziałek, 9.00:
- index.html (nowy),
- o-nas.html (nowy).

Jak działa kontrola wersji?

WTEM!

Okazuje się, że **Marcin** niepotrzebnie zmodyfikował plik **regulamin.html**!

Czy cała praca nad tym plikiem poszła na marne?!

Na szczęście nie! **Marcin** może wrócić do wersji sprzed zmiany, czyli wersji 2, ze środy, z godziny 14.00.

Weekend uratowany. Ufff…

regulamin.html

git

~~**Piątek, 16:55:**~~
~~• regulamin.html (modyfikacja)~~

Środa, 14.00:
- index.html (modyfikacja),
- regulamin.html (nowy),
- smieszny-kotek.png(nowy).

Poniedziałek, 9.00:
- index.html (nowy),
- o-nas.html (nowy).

Podstawowa składnia GIT

Podstawy korzystania z Git

Z Git-a możemy korzystać na dwa różne sposoby:

- Jako programu konsolowego,
- Przez dodatkowy program graficzny (trzeba go osobno doinstalować – istnieją dziesiątki różnych programów).

Podczas naszego kursu polecamy żebyście korzystali z komend konsolowych GIT-a. Dzięki temu poznacie to w jaki sposób on działa i co dokładnie można z nim zrobić.

Nauka komend GIT

Najłatwiej nauczyć się komend GITa korzystając z nich. Dlatego przejdź interaktywny tutorial który możesz znaleźć na stronie:

https://try.github.io/levels/1/challenges/1

Podczas robienia tego tutorialu zapisuj sobie co robią poszczególne komendy – te notatki bardzo Ci się przydadzą.

GIT CheatSheet

Możesz też skorzystać z gotowego zbioru najprzydatniejszych komend. Jest to tak zwany Git CheetSheet.

Możesz znaleźć wiele takich wersji np.:

- https://www.git-tower.com/blog/git-cheat-sheet/
- https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf
- http://www.cheat-sheets.org/saved-copy/git-cheat-sheet.pdf

Repozytoria

Repozytorium

Git przechowuje Twój kod źródłowy w tzw. **repozytorium.**

Są to zapisane kolejne zmiany w Twoim kodzie (od samego początku pracy), które nawarstwiają się od samego początku pracy nad projektem, aż do aktualnej wersji.

Repozytorium może być **lokalne** lub **zdalne**.

Repozytorium lokalne

Repozytorium lokalne (local repository) znajduje się na Twoim komputerze. Pracując nad swoim programem, możesz (a nawet powinieneś/powinnaś) umieszczać w nim wszystkie zmiany.

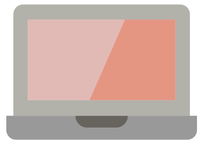
Lokalne repozytorium znajduje się w głównym katalogu Twojego projektu w ukrytym folderze o nazwie **.git**.

Praca lokalna

Marcin tworzy nowy projekt „Landing page”

1. **Marcin** tworzy **lokalne repozytorium** (na razie puste).

git init



(brak plików)

```
marcin@xwing:~/workspace/landing-page$ git init
Initialized empty Git repository in /home/marcin/workspace/landing-page/.git/
marcin@xwing:~/workspace/landing-page$
```

Praca lokalna

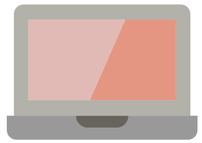
Marcin zaczyna kontrolować wersje pliku:

2. **Marcin** tworzy nowy plik:
- landing-page.html

Teraz **Marcin** musi poinformować git-a, że chce objąć ten plik kontrolą wersji:

git add landing-page.html

```
marcin@xwing:~/workspace/landing-page$ git add landing-page.html
marcin@xwing:~/workspace/landing-page$
```



(brak plików)

landing-page.html

Praca lokalna

Marcin umieszcza zmiany w systemie kontroli wersji.

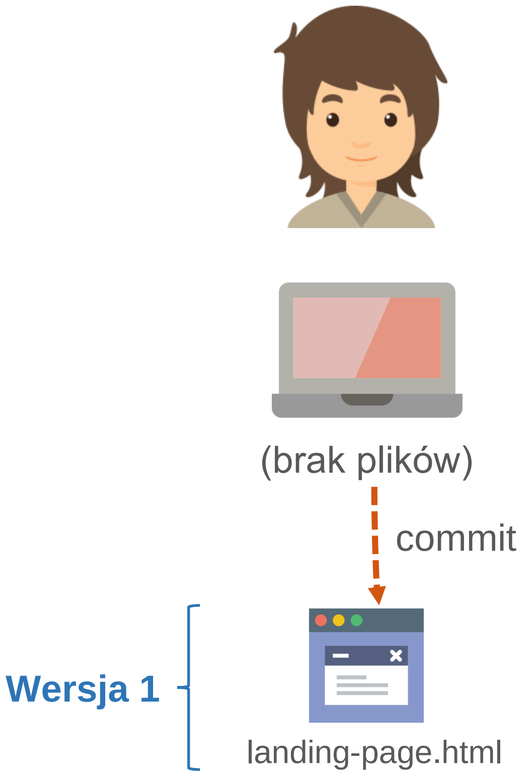
3. **Marcin** umieszcza zmiany w git:
- landing-page.html

Marcin musi pamiętać, aby opisać swoje zmiany!

git commit -m "Dodałem nagłówek i stopkę"

-m oznacza, że **Marcin** chce opisać zmiany w plikach. Informację tę podaje w cudzysłowach, zaraz po parametrze. Ta informacja jest obowiązkowa! Git nie pozwoli zapisać zmian bez opisu!

```
marcin@xwing:~/workspace/landing-page$ git commit -m "Dodałem nagłówek i stopkę"
[master (root-commit) d708c1d] Dodałem nagłówek i stopkę
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 landing-page.html
marcin@xwing:~/workspace/landing-page$
```



Repozytorium zdalne

Repozytorium zdalne (remote repository) znajduje się na zewnętrznym serwerze, np. **github.com**.

Repozytorium zdalnego używasz, gdy chcesz udostępnić swój kod innym (np. współpracownikom) lub przyłączyć się do istniejącego projektu.

Repozytorium zdalne vs lokalne

Jeśli chcesz dołączyć do istniejącego projektu musisz skopiować zdalne repozytorium na swój dysk. Ta czynność nazywa się **klonowaniem** (**clone**) repozytorium.

Powstaje wtedy lokalne repozytorium z aktualną wersją kodu.

Po wprowadzeniu zmian, umieszczasz je w lokalnym repozytorium (**commit**)**,** a potem wpychasz (**push**) na serwer zdalny. Od tej pory inni programiści widzą Twoje zmiany.

Praca na repozytoriach

Agata tworzy nowy projekt „Sklep internetowy”

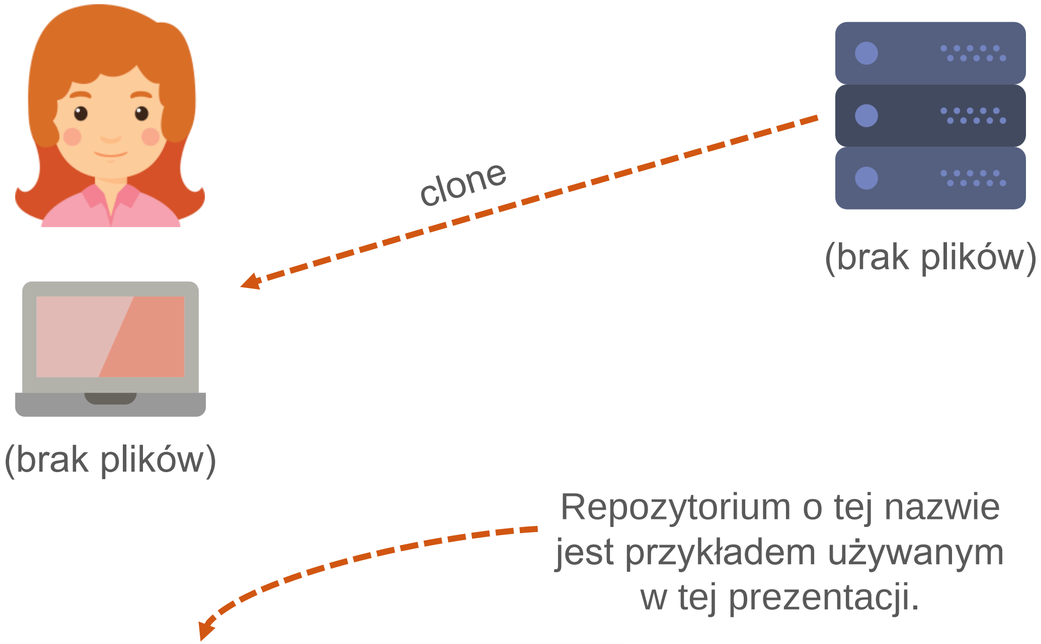
1. **Agata** tworzy **zdalne repozytorium** (na razie puste).

Praca na repozytoriach

Agata tworzy nowy projekt „Sklep internetowy”

2. Następnie **Agata klonuje** zdalne repozytorium do swojego laptopa, tworzy się **repozytorium lokalne**.

git clone <adres repozytorium>




```
agata@agata:~/workspace$ git clone https://github.com/marcin-barylka/sklep-internetowy.git
Cloning into 'sklep-internetowy'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
agata@agata:~/workspace$
```

Praca na repozytoriach

Agata pracuje nad projektem:

3. **Agata** pracuje nad nowymi plikami:

- produkt.html,
- koszyk.html.

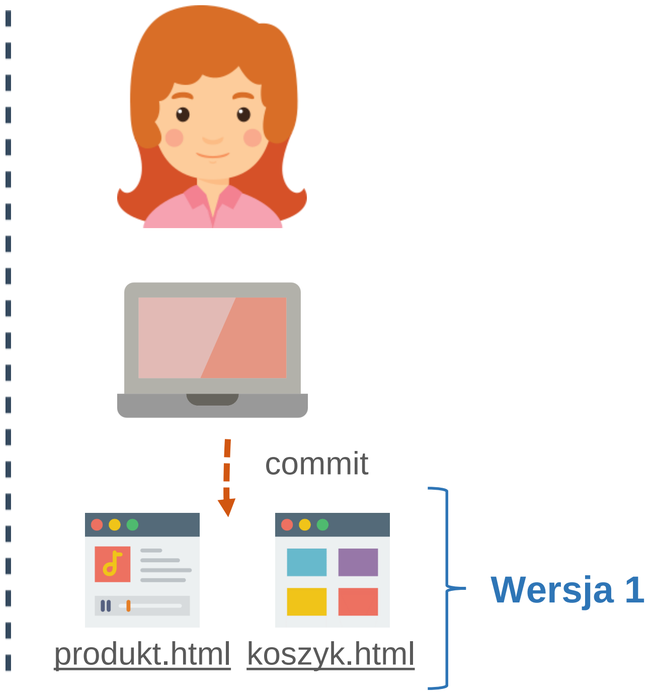
Agata dodaje je do kontroli wersji:

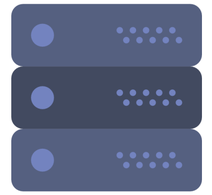
git add *.html

A potem zapisuje zmiany w Git:

git commit -m "Strony produktu i koszyka"

Te zmiany są widoczne tylko dla niej!





(brak plików)

Praca na repozytoriach

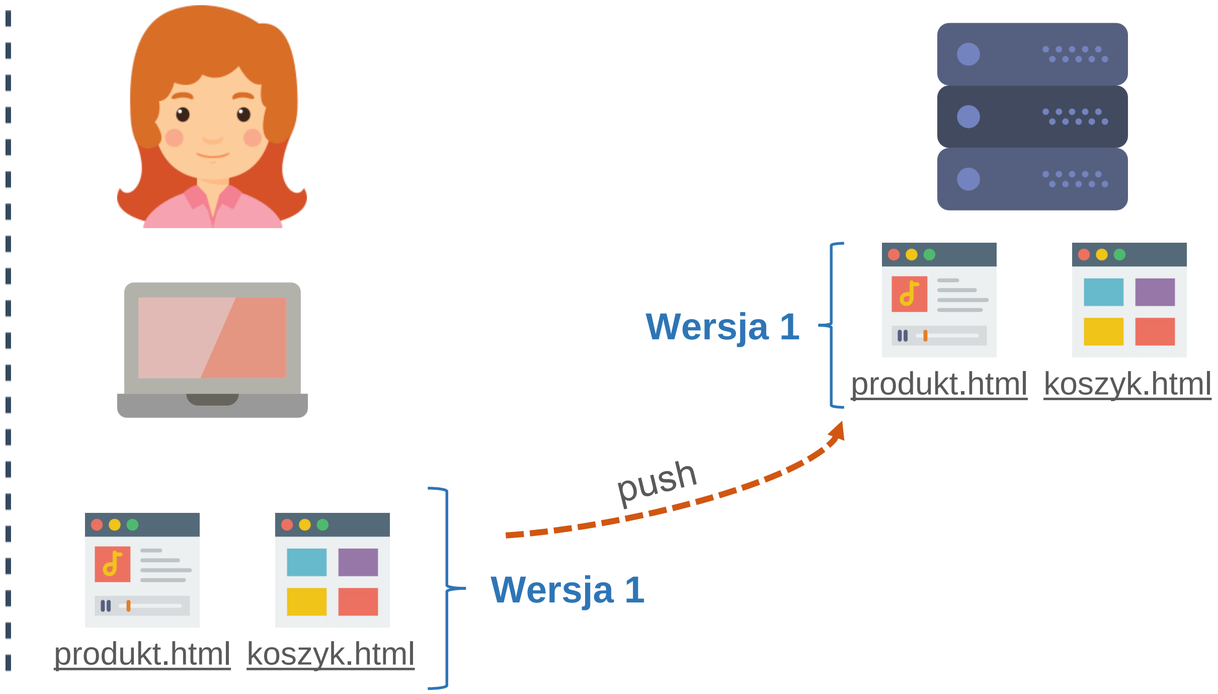
Agata aktualizuje zdalne repozytorium.

4. **Agata** wypycha (**push**) swoje zmiany do zdalnego repozytorium.

- produkt.html,
- koszyk.html.

git push

Od tej pory jej zmiany są widoczne dla reszty programistów w zespole.



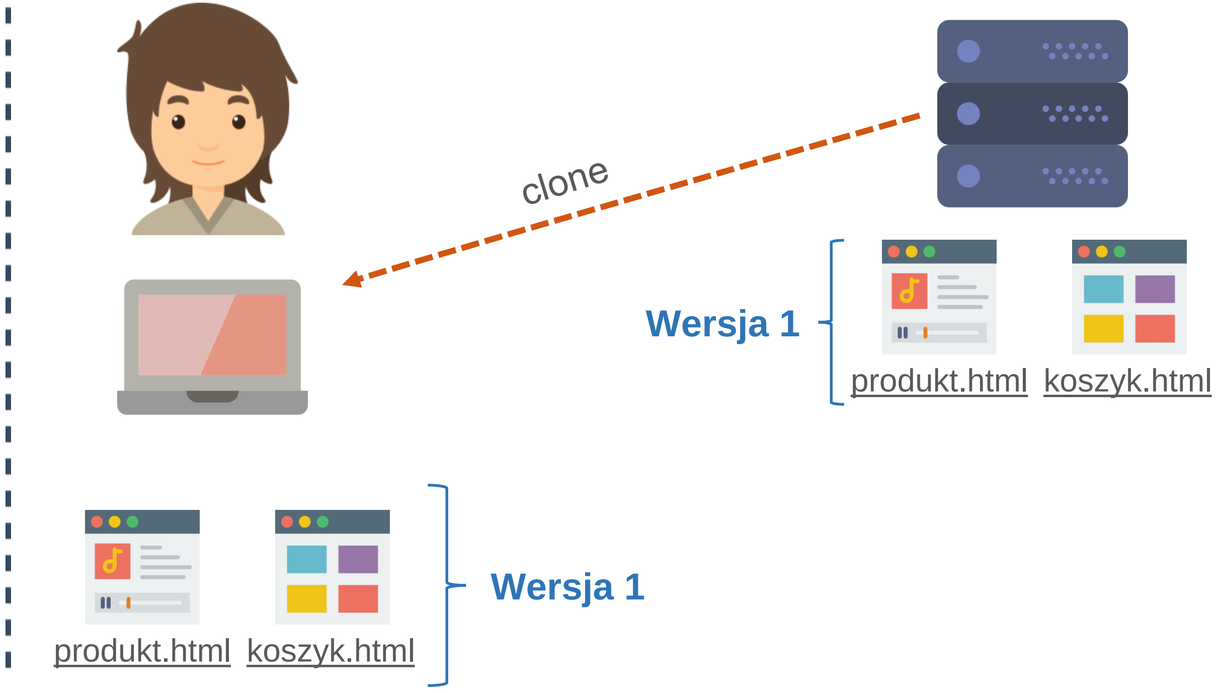
Praca zespołowa

Praca zespołowa

Marcin dołącza do projektu.

1. **Marcin klonuje** zdalne repozytorium do swojego laptopa, tworzy się **repozytorium lokalne**.

git clone <adres repozytorium>



Praca zespołowa

Marcin dołącza do projektu.

2. **Marcin** pracuje nad stroną produktu i stroną zamówienia:

- produkt.html,
- zamowienie.html

…ale w pewnym momencie musi odejść od komputera. Gdy wraca, chce sprawdzić, co już zmienił:

git status

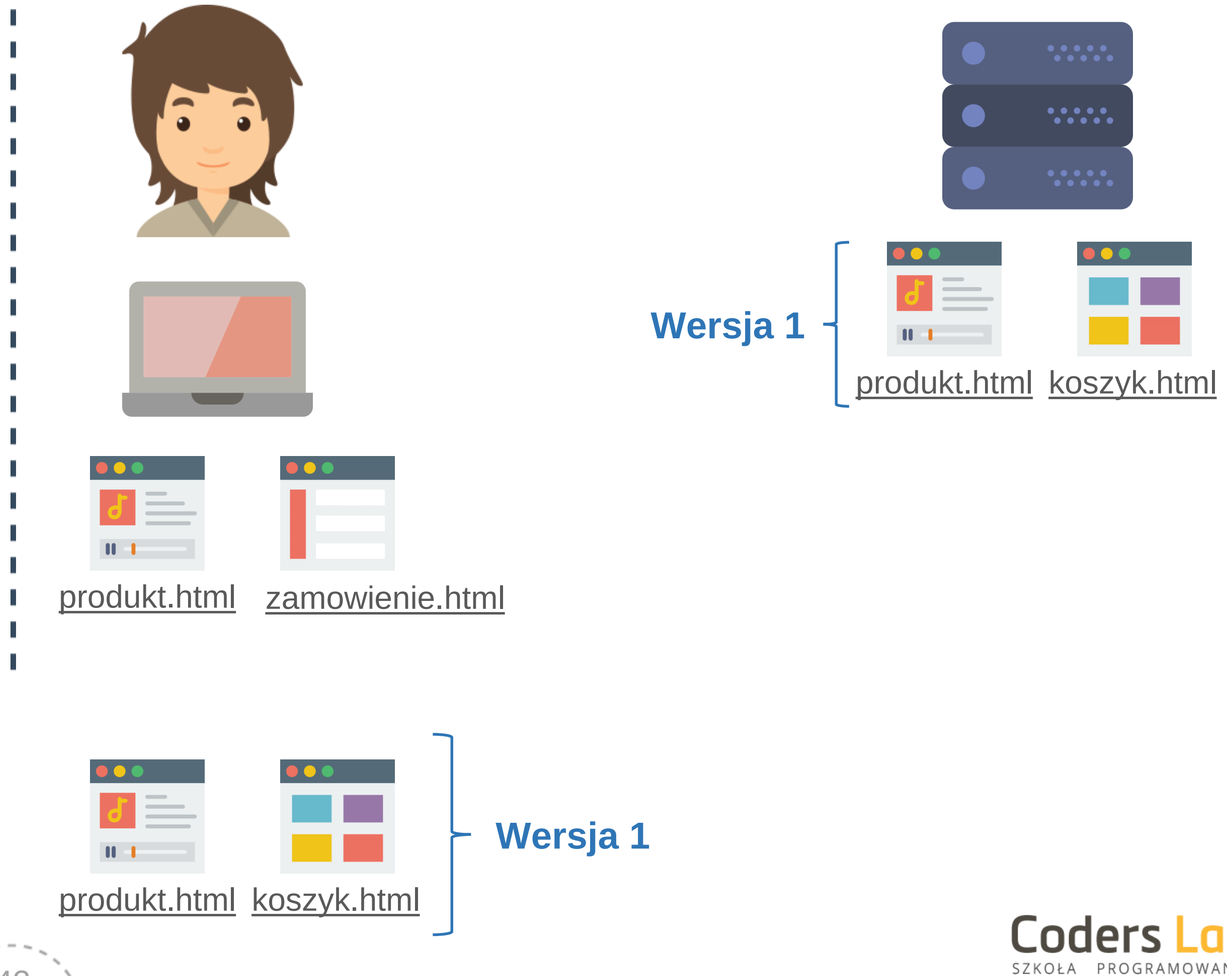
```
marcin@xwing:~/workspace/sklep$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        zmodyfikowany: produkt.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        zamowienie.html

no changes added to commit (use "git add" and/or "git commit -a")
marcin@xwing:~/workspace/sklep$
```



Praca zespołowa

Marcin dołącza do projektu.

3. **Marcin** dodaje zmiany do Gita:

- produkt.html,
- zamowienie.html

git add .

Uwaga: ta kropka jest tu nieprzypadkowo! Oznacza ona wszystkie zmienione lub dodane pliki z aktualnego folderu i wszystkich folderów poniżej.

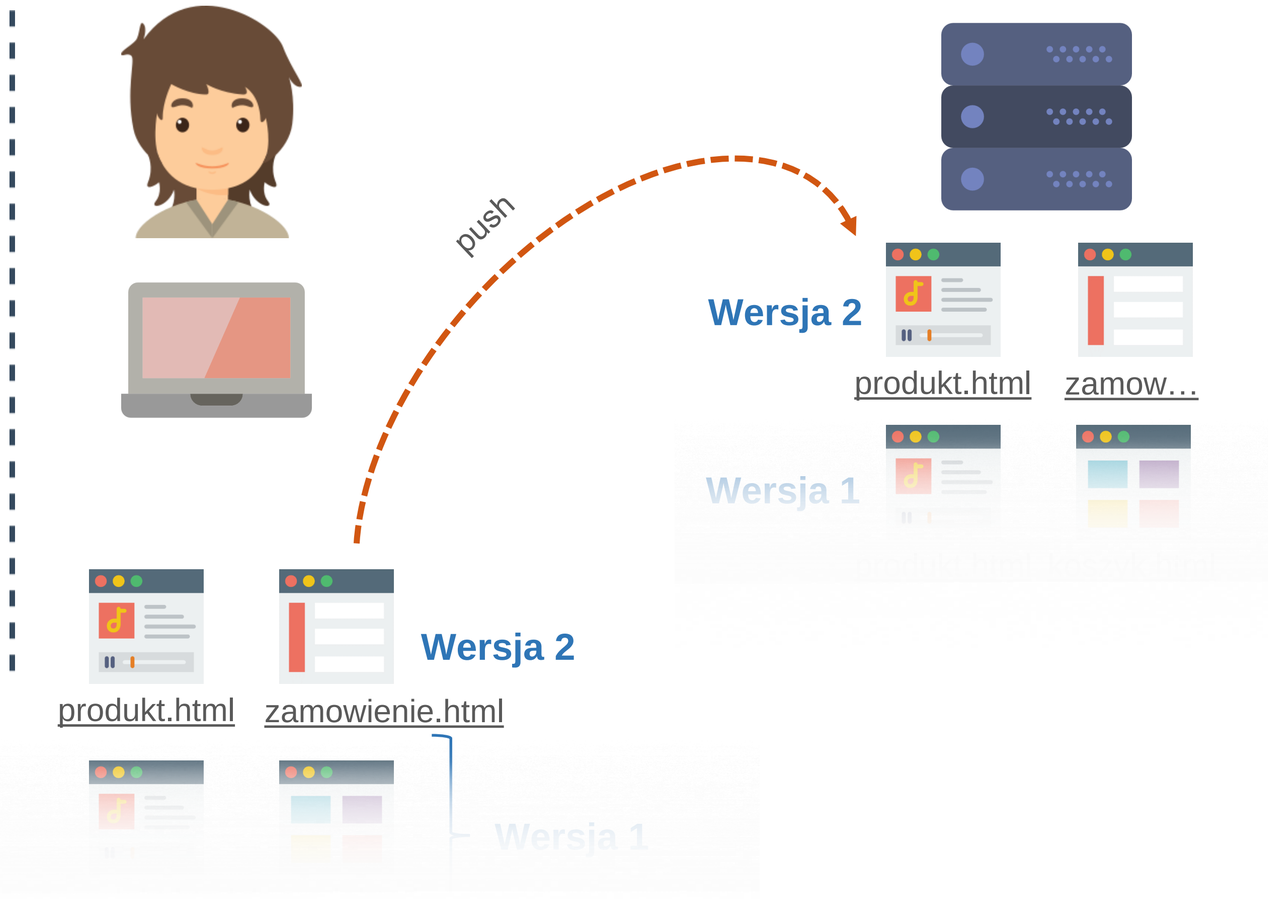
git commit -m "produkt i zamówienie"

Praca zespołowa

Marcin dołącza do projektu.

4. Marcin wypycha (**push**) swoje zmiany do **repozytorium zdalnego**.

git push



Praca zespołowa

Co się stanie, gdy Agata wróci do pracy nad zdalnym repozytorium?

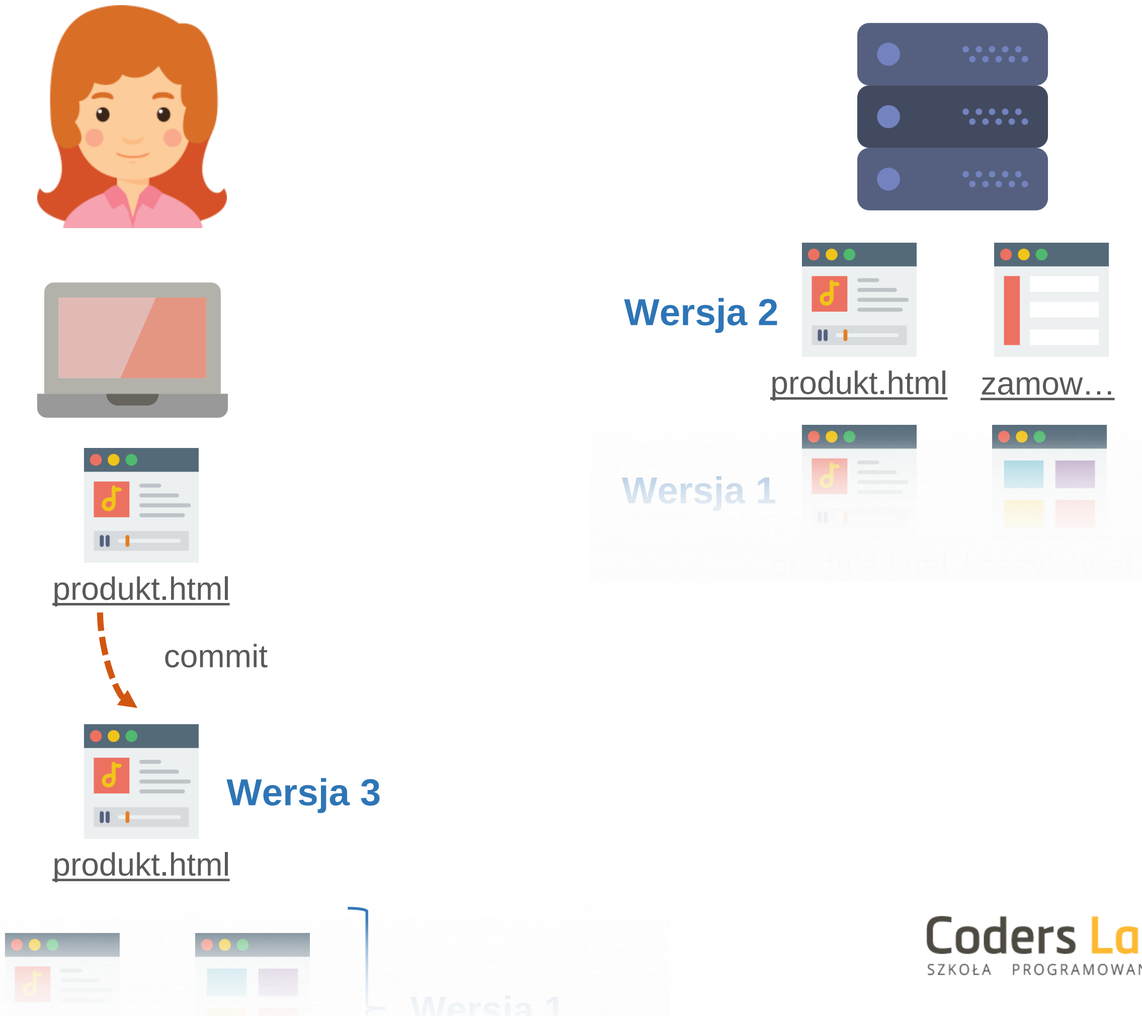
5. **Agata** ma u siebie **wersję 1**, **Marcin** dodał do zdalnego repozytorium, **wersję 2**.

Agata zmienia jeden plik:

- produkt.html.

Dodaje plik do kontroli wersji i zatwierdza zmiany:

git add produkt.html
git commit -m "praca nad produktem"



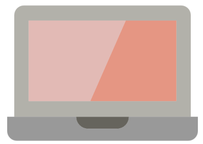
Praca zespołowa

WTEM!

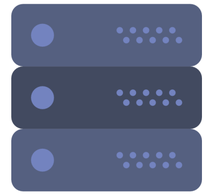
6. Gdy **Agata** próbuje wypchnąć plik (**git push**), dostaje komunikat, jak na ilustracji poniżej.

Git poinformował **Agatę**, że w zdalnym repozytorium znajdują się zmiany, które **Marcin** zrobił w międzyczasie. Poprawki te uniemożliwiają wrzucenie zmian, które zrobiła **Agata**.

Czy problem Agaty da się jakoś rozwiązać?



produkt.html



produkt.html

zamow…

Wersja 2

Wersja 1

Przeczytaj uważnie komunikat.
Czy wiesz, jak pomóc **Agacie**?

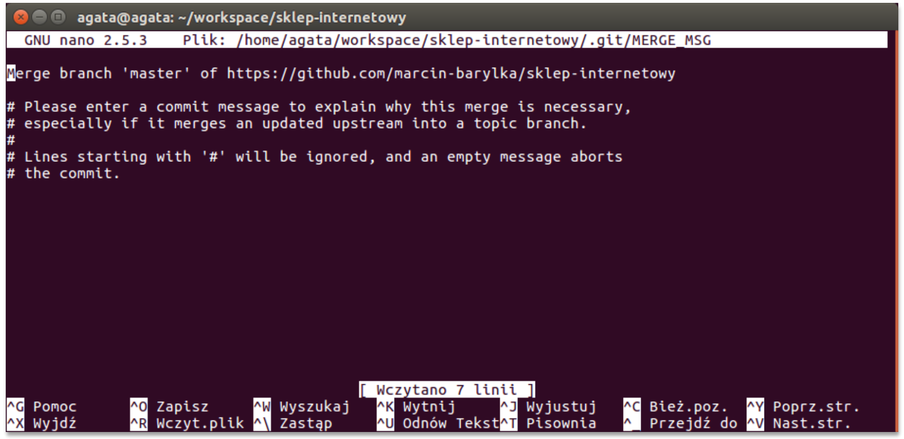
```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/marcin-barylka/sklep-internetowy.git'
podpowiedź: Updates were rejected because the remote contains work that you do
podpowiedź: not have locally. This is usually caused by another repository pushing
podpowiedź: to the same ref. You may want to first integrate the remote changes
podpowiedź: (e.g., 'git pull ...') before pushing again.
podpowiedź: See the 'Note about fast-forwards' in 'git push --help' for details.
agata@agata:~/workspace/sklep-internetowy$
```

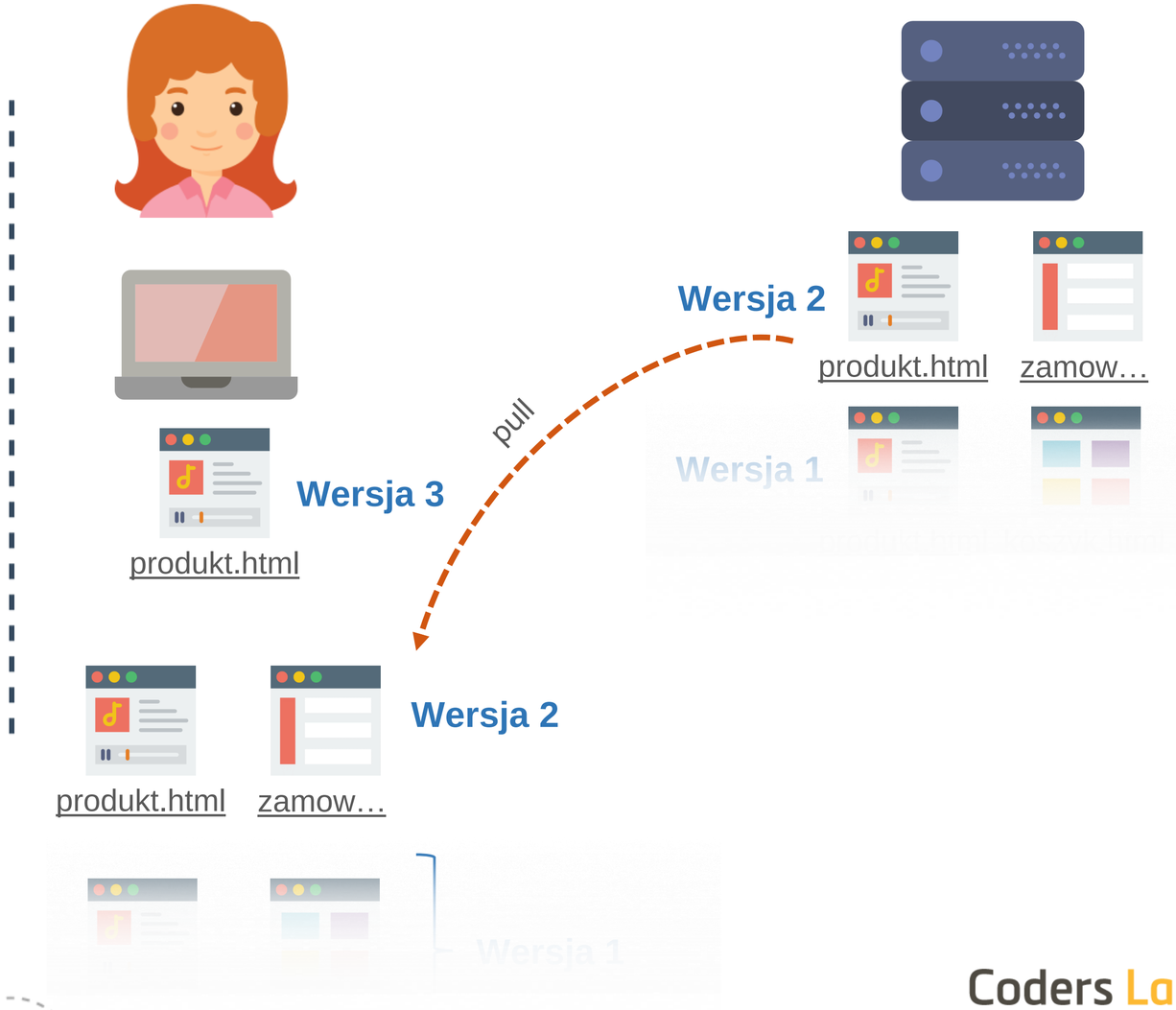
Praca zespołowa

Nic złego się nie stało!

7. **Agata** musi tylko zintegrować zmiany **Marcina** ze swoimi. Musi zatem wykonać instrukcję **pull**.

git pull
Git zaproponuje Agacie opis zmiany.
Wystarczy go zaakceptować.





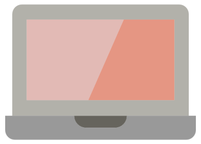

Konflikt!

Konflikt!

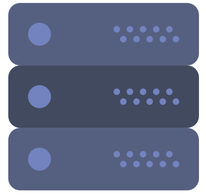
Agata pracuje nad plikiem **zamówienie.html**. Umieszcza plik w zdalnym repozytorium.

Kilka dni później, **Marcin** również pracuje nad tym samym plikiem. Również próbuje umieścić go w zdalnym repozytorium. Dostaje jednak znany nam komunikat o czekających zmianach w repozytorium zdalnym. Marcin próbuje ściągnąć te zmiany komendą **git pull.**

```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/marcin-barylka/sklep-internetowy.git'
podpowiedź: Updates were rejected because the remote contains work that you do
podpowiedź: not have locally. This is usually caused by another repository pushing
podpowiedź: to the same ref. You may want to first integrate the remote changes
podpowiedź: (e.g., 'git pull ...') before pushing again.
podpowiedź: See the 'Note about fast-forwards' in 'git push --help' for details.
agata@agata:~/workspace/sklep-internetowy$
```



zamowienie.html



zamow…

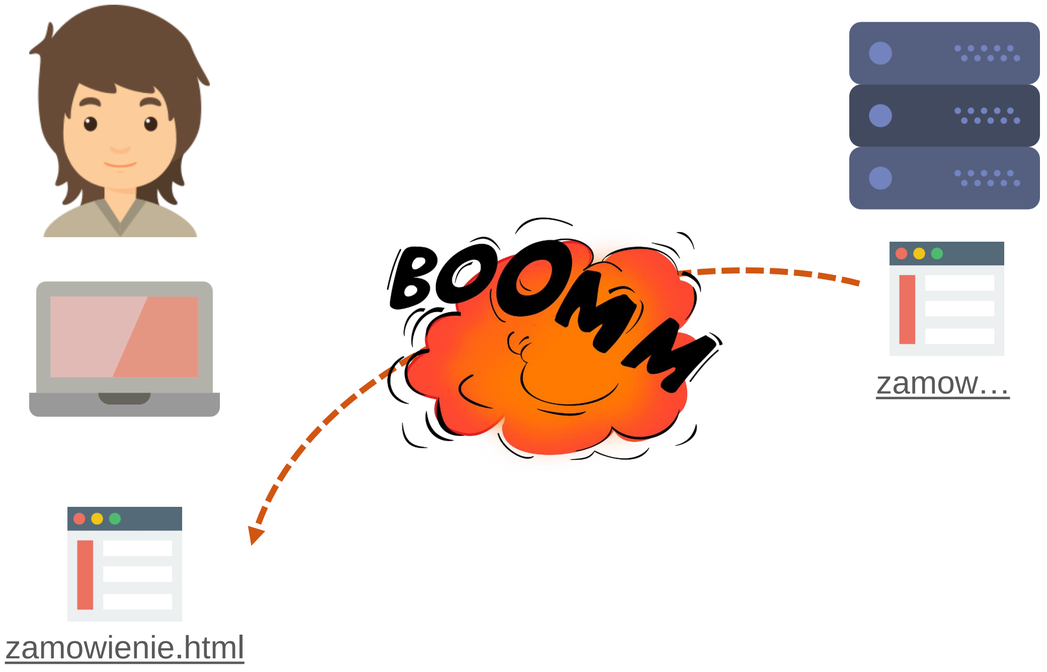
Konflikt!

WTEM!

```
marcin@xwing:~/workspace/sklep-internetowy$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/marcin-barylka/sklep-internetowy
   c9d6e9d..38d8078  master     -> origin/master
Auto-merging zamowienie.html
CONFLICT (zawartość): Merge conflict in zamowienie.html
Automatic merge failed; fix conflicts and then commit the result.
marcin@xwing:~/workspace/sklep-internetowy$
```

Git informuje Marcina o konflikcie!

Czy to oznacza, że **Marcin** pokłóci się teraz z **Agatą**?



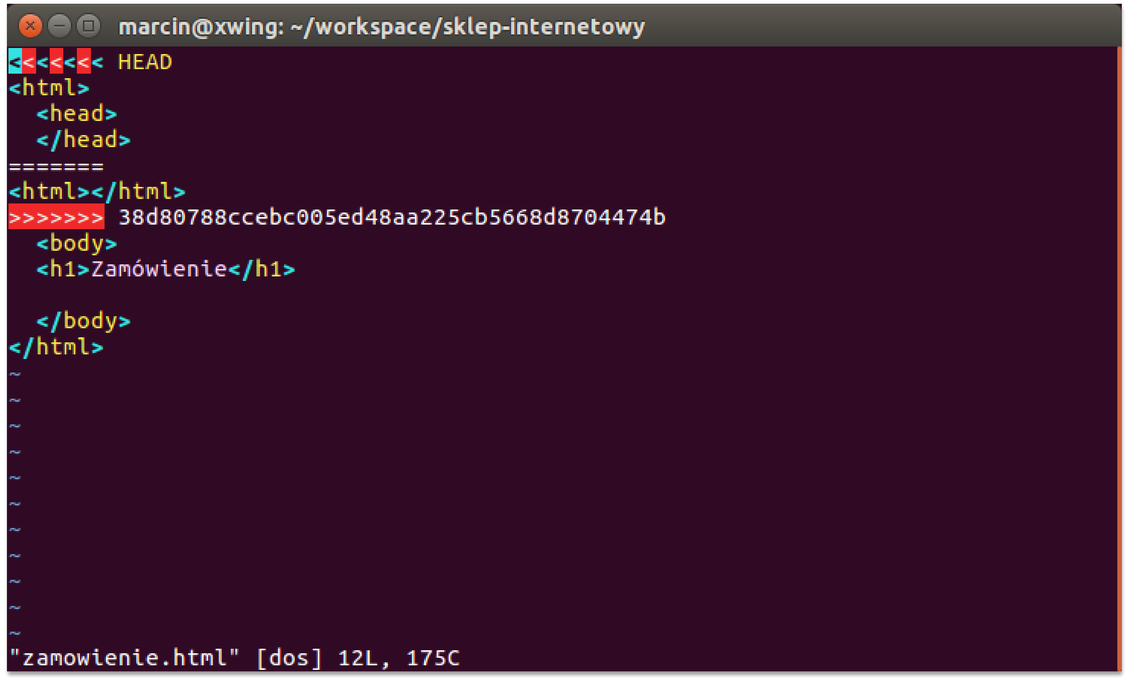
Konflikt!

Konflikt w kodzie, to nic złego!

Oznacza to jedynie, że Git nie potrafił połączyć automatycznie zmian **Marcina** i **Agaty**.

Marcin musi połączyć te zmiany ręcznie. Potem musi wykonać **commit** i **push**.

Wbrew pozorom, to duże ułatwienie dla programistów: mimo wszystko mogą pracować równocześnie na tym samym pliku. Git automatycznie przypilnuje, żeby programiści nie psuli sobie nawzajem kodu. Jeśli nie będzie potrafił poradzić sobie z integracją zmian, poprosi o to człowieka.

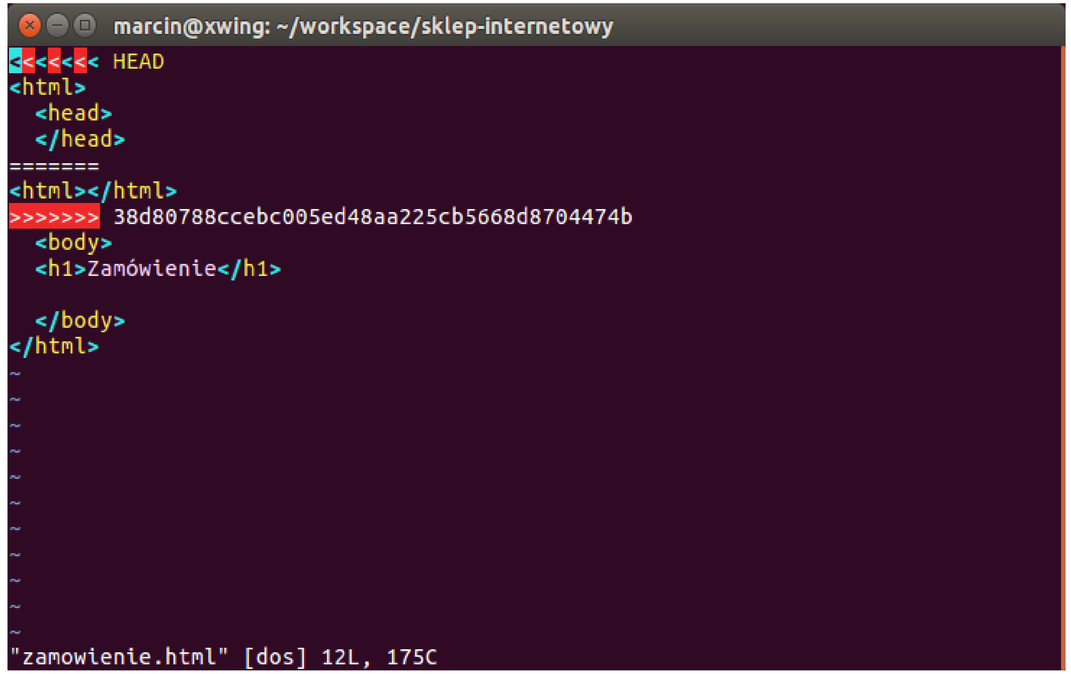


Konflikt!

HEAD to najświeższe zmiany z **lokalnego repozytorium** (czyli te, które zrobił przed chwilą **Marcin**)

Długa liczba, zaczynająca się od **38d807…**, to identyfikator najnowszej wersji ze zdalnego repozytorium (czyli te, które kilka dni temu zrobiła **Agata**).

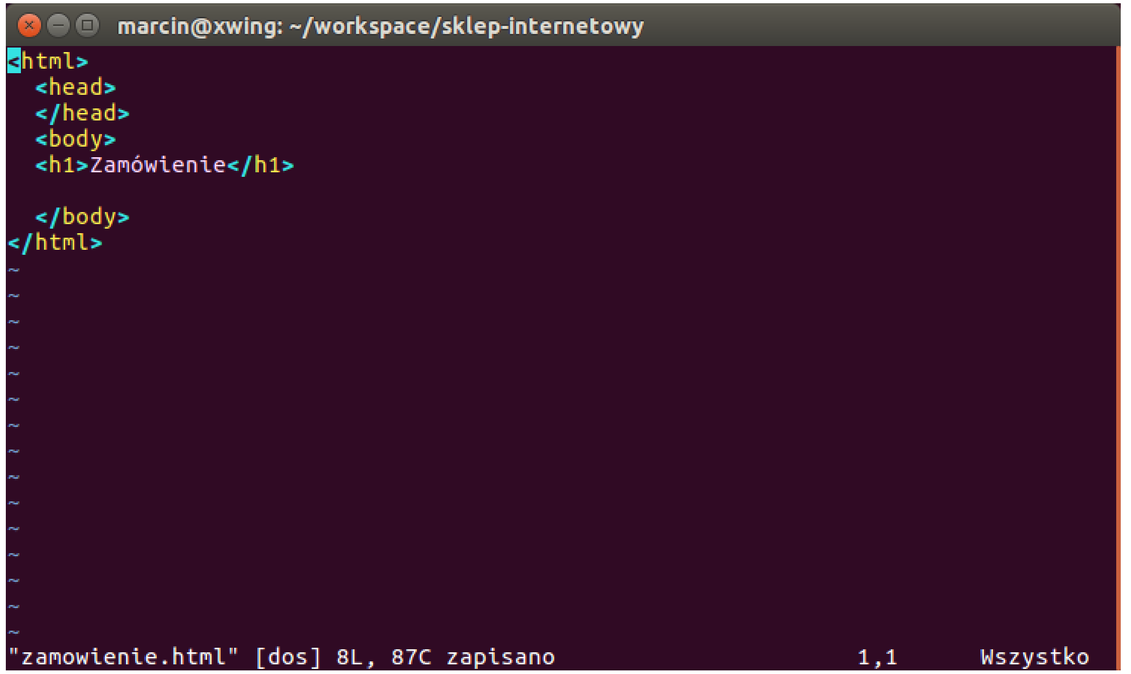
Zadaniem **Marcina** będzie połączenie tych dwóch wersji tak, by uwzględnić zmiany swoje i **Agaty**.



Konflikt!

Marcin rozwiązuje konflikt

1. Usuwa informacje o skonfliktowanych zmianach,
2. Łączy zmiany swoje i **Agaty**



Konflikt!

Marcin rozwiązuje konflikt

3. Dodaje zmiany do kontroli wersji (**git add**),
4. Wykonuje **git commit** z komentarzem o rozwiązaniu konfliktu,
5. Wypycha zmiany do repozytorium zewnętrznego.

```
marcin@xwing:~/workspace/sklep-internetowy$ git add zamowienie.html
marcin@xwing:~/workspace/sklep-internetowy$ git commit -m "Rozwiązanie konfliktu
"
[master b1de440] Rozwiązanie konfliktu
marcin@xwing:~/workspace/sklep-internetowy$ git push
```

Przydatne linki dla użytkowników Gita.

Przydatne linki

Strona główna projektu Git:
https://git-scm.com/

Oficjalny podręcznik do Gita (za darmo, w różnych formatach):
https://git-scm.com/book/en/v2

Ciekawy samouczek Gita, na stronie GitHub:
https://try.github.io/levels/1/challenges/1

Kolejny tutorial do Gita:
https://www.git-tower.com/learn/git/ebook/en/mac/introduction

Git - zadanie

- Spróbuj przejrzeć linki dostępne obok i wybierz przynajmniej dwa, które przerobisz obowiązkowo.

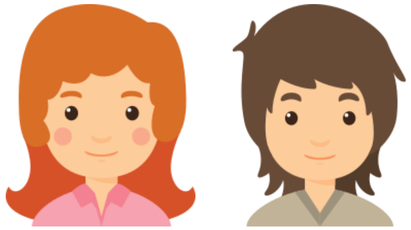
- http://gitreal.codeschool.com/levels/1
- http://www.git-tower.com/learn/git/videos/
- http://www.dataschool.io/git-and-github-videos-for-beginners/

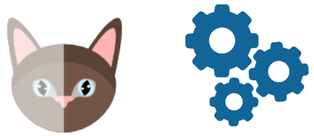
- Dla bardziej zaawansowanych http://pcottle.github.io/learnGitBranching/

Obowiązkowa informacja o prawach autorskich

Copyrights

Icons designed by **Freepik** from **www.flaticon.com**





Icons designed by **Madebyoliver** from **www.flaticon.com**