



Kurs Front End Developer
Systemy kontroli wersji - GIT

SYSTEMY KONTROLI WERSJI



git



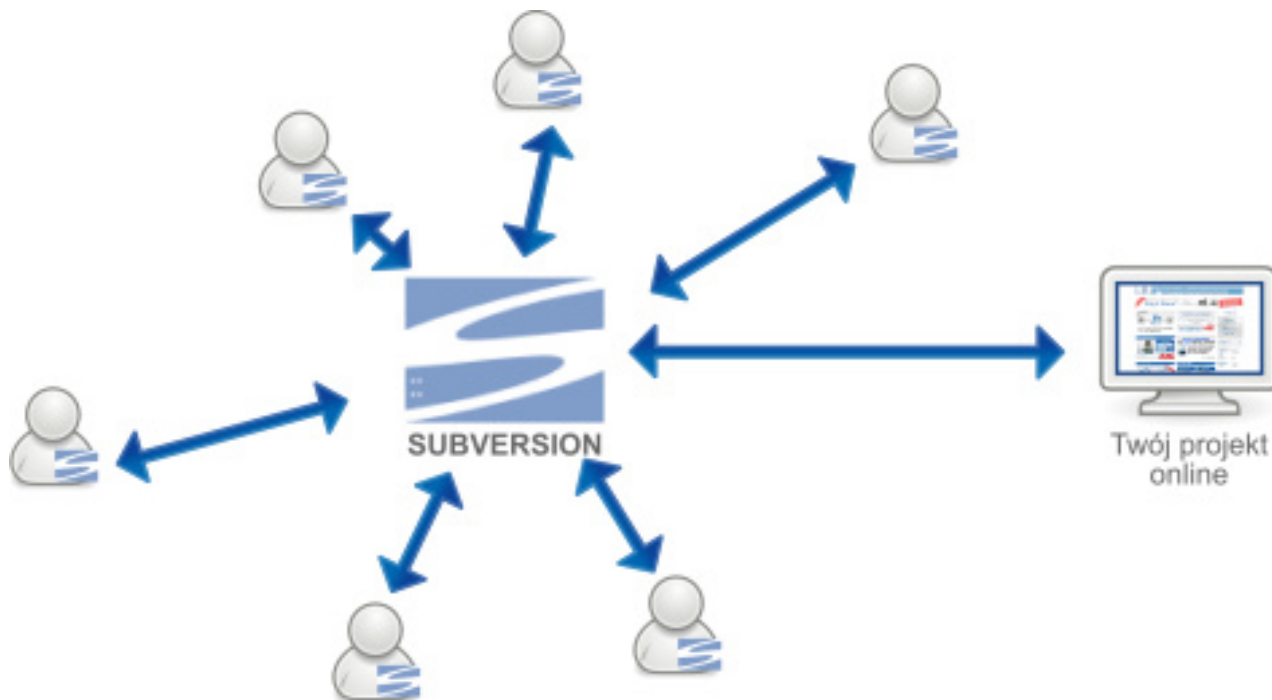
Systemy Kontroli Wersji to narzędzie pozwalające na zapisywanie i śledzenie zmian w kodzie projektu.

W każdym momencie programista może mieć wgląd w historie swojej pracy i w razie potrzeby do niej powrócić.

Ułatwiają także pracę nad projektem w kilkusobowych zespołach, pilnując, aby np. programiści nie nadpisywali sobie pracy – tzw. konflikty.

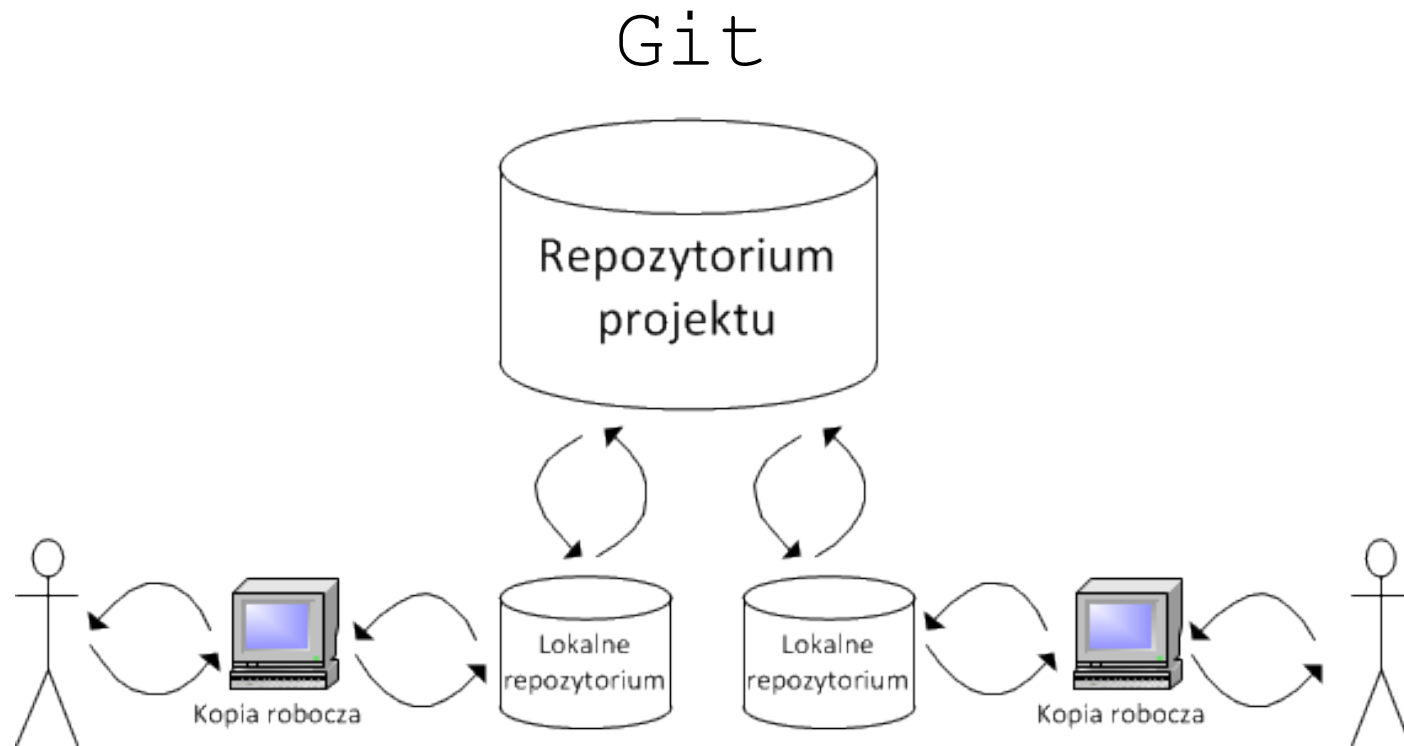
SCENTRALIZOWANE SYSTEMY KONTROLI WERSJI

Scentralizowane Systemy Kontroli Wersji - SVN



ROZPROSZONE SYSTEMY KONTROLI WERSJI

Rozproszone Systemy Kontroli Wersji - GIT



DLACZEGO GIT?



Git jest obecnie najpopularniejszym systemem kontroli wersji

Istnieje kilka darmowych, ogólniedostępnych repozytoriów np.

- GitHub – <https://github.com>
- BitBucket – <https://bitbucket.com>

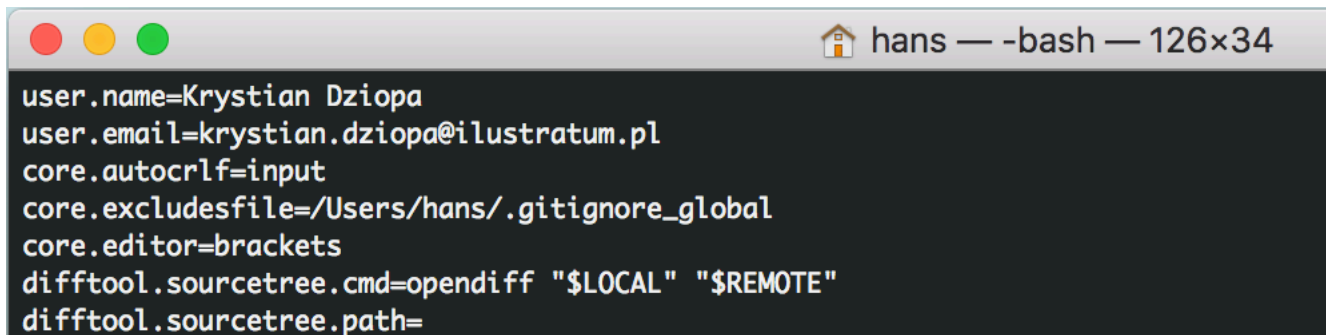
Łatwa instalacja i konfiguracja na najpopularniejszych systemach operacyjnych Windows, OSX, Linux

KONFIGURACJA GITA

Uruchamiamy Git Bash/Terminal i sprawdzamy konfigurację Git komendą

```
git config --list
```

Po wylistowaniu konfiguracji Git powinny być widoczne ustawienia `user.name` i `user.email` – jak poniżej:

A screenshot of a terminal window titled "hans — -bash — 126x34". The terminal displays the output of the command "git config --list", showing the following configuration: user.name=Krystian Dziopa, user.email=krystian.dziopa@ilustratum.pl, core.autocrlf=input, core.excludesfile=/Users/hans/.gitignore_global, core.editor=brackets, difftool.sourcetree.cmd=opendiff "\$LOCAL" "\$REMOTE", and difftool.sourcetree.path=.

```
user.name=Krystian Dziopa
user.email=krystian.dziopa@ilustratum.pl
core.autocrlf=input
core.excludesfile=/Users/hans/.gitignore_global
core.editor=brackets
difftool.sourcetree.cmd=opendiff "$LOCAL" "$REMOTE"
difftool.sourcetree.path=
```

W przypadku braku w/w ustawień, musimy ponownie skonfigurować Git

```
git config --global user.name "Jan Nowak"
git config --global user.email jannowak@example.com
```

**W miejsce imienia i nazwiska oraz adresu email wpisujemy własne dane.*

TERMINAL – KOMENDY

Najczęściej używane operacje podczas poruszania się po dysku komputera w terminalu:

`pwd` – pokazuje ścieżkę do aktualnego folderu

`cd nazwa-folderu` - przechodzi do folderu o podanej nazwie

`cd ..` - przechodzi do folderu powyżej

`cd ~` - przechodzi do folderu domowego

`ls` - listuje aktualną zawartość folderu

`ls -l` - listuje zawartość folderu z dodatkowymi informacjami

`ls -a` - listuje wszystkie pliki razem z ukrytymi plikami

`mkdir nazwa-folderu` - tworzy nowy katalog o podanej nazwie

`touch nazwa-pliku` - tworzy nowy plik o podanej nazwie

TERMINAL - KOMENDY

`rm nazwa-pliku` - usuń plik o podanej nazwie

`rm -r nazwa-folderu` - usuwa folder i całą jego zawartość – **ostrożnie!**

`cp ścieżka-do-pliku/folderu ścieżka_do_folderu` - kopiuje plik/folder do wskazanego folderu

`mv ścieżka-do-pliku/folderu ścieżka_do_folderu` - przenosi plik/folder do wskazanego folderu (używane także do zmiany nazwy pliku lub folderu)

`cat nazwa-pliku` - wyświetla w terminalu zawartość pliku jako tekst

WARSZTATY ☺ KOMEND W TERMINALU

1. Przejdź w terminalu na pulpit (`cd ~/Desktop`)
2. Utwórz na pulpicie folder `terminal` (`mkdir`)
3. W folderze `terminal` stwórz dwa foldery pierwszy i drugi
4. Przejdź do folderu `pierwszy` i stwórz w nim plik `index.html` (`touch`)
5. Sprawdź zawartość folderu (`ls`)
6. Skopiuj plik z folderu `pierwszy` do folderu `drugi` (`cp`)
7. Przejdź do folderu `drugi` i sprawdź, czy jest tam skopiowany plik
8. Usuń plik z folderu `pierwszy` (`rm`)
9. Usuń cały folder `pierwszy` (`rm -r`)
10. Zmień nazwę folderu z `drugi` na `pierwszy` (`mv`)

GitHub Pages 😊

Logujemy się na konto w serwisie GitHub i tworzymy nowe repozytorium New Repository.

Nadajemy nowemu repozytorium nazwę `username.github.io` (repozytorium GitHub Pages) oraz dodajemy jego opis.

* **username** to Twoja nazwa użytkownika GitHub

GitHub Pages == server WWW

Więcej informacji:

<https://pages.github.com/>



UŻYWAMY REPOZYTORIUM GIT 😊

Otwieramy terminal i przechodzimy na pulpit wydając polecenie

```
cd ~/Desktop
```

Następnie na pulpicie tworzymy folder o nazwie `repositories`

```
mkdir repositories
```

Przechodzimy do tego folderu

```
cd repositories
```

I klonujemy nasze repozytorium GitHub Pages z serwisu GitHub

```
git clone https://github.com/username/username.github.io.git
```

* *username* - to Twoja nazwa użytkownika GitHub

Przechodzimy do folderu z naszym repozytorium GitHub Pages

```
cd username.github.io
```

UŻYWAMY REPOZYTORIUM GIT 😊

Uruchamiamy edytor tekstu i otwieramy w nim nasz folder z projektem `username.github.io`

Następnie tworzymy plik `index.html` i dodajemy do niego treść:

- podstawową strukturę dokumentu HTML (!)
- i jeden paragraf (`p`)

Sprawdzamy status naszego repozytorium `git status`

Dodajemy plik(i) do śledzenia `git add *`

** przed `commitem` zawsze należy dodać pliki do śledzenia!*

Ponownie sprawdzamy status repozytorium `git status`

Robimy pierwszy commit z komentarzem

```
git commit -m "Dodano pierwszy paragraf na gałęzi master"
```

UŻYWAMY REPOZYTORIUM GIT 😊

Ponownie sprawdzamy status repozytorium `git status`

I wysyłamy pliki do zdalnego repozytorium na GitHub `git push`

Sprawdzamy repozytorium GitHub Pages pod adresem

<https://github.com/username/username.github.io>

Wysłany commit powinien być już widoczny 😊

Następnie sprawdzamy, czy działa już mechanizm GitHub Pages (czyli serwer WWW) pod adresem <https://username.github.io/>

Tworzymy nową gałąź `git checkout -b alfa-version`

Zostaniemy domyślnie przełączeni na tę gałąź.

I wysyłamy ją do zdalnego repozytorium `git push origin alfa-version`

UŻYWAMY REPOZYTORIUM GIT 😊

Sprawdzamy ponownie repozytorium GitHub Pages pod adresem

<https://github.com/username/username.github.io>

Wysłana gałąź `alfa-version` powinna być już widoczna 😊

Sprawdzamy stworzone gałęzie w repozytorium, oraz tą, na której znajdujemy się obecnie `git branch`

Będąc na gałęzi `alfa-version` modyfikujemy ponownie plik `index.html` dodając drugi paragraf (`p`)

Sprawdzamy różnicę we wprowadzonych zmianach od ostatniego commita `git diff`

Dodajemy plik `index.html` do śledzenia w nowej gałęzi `git add *`

UŻYWAMY REPOZYTORIUM GIT 😊

I robimy commit na gałęzi `alfa-version`, czyli

```
git commit -m "Dodano drugi paragraf na gałęzi alfa-version"
```

Porównujemy obie gałędzie `git diff master..alfa-version`

Przełączamy się do głównej gałęzi `git checkout master`

* w pliku `index.html` znikną zmiany dodane w commit na gałęzi `alfa-version` - sprawdź w edytorze 😊

Będąc na gałęzi `master` modyfikujemy ponownie plik `index.html` dodając trzeci paragraf (p)

Dodajemy plik `index.html` do śledzenia na tej gałęzi `git add *`

I robimy commit na gałęzi `master`, czyli

```
git commit -m "Dodano trzeci paragraf na gałęzi master"
```

UŻYWAMY REPOZYTORIUM GIT 😊

Porównujemy obie gałędzie `git diff master..alfa-version`

Łączymy gałąź `master` z gałęzią `alfa-version` robiąc merge
`git merge master alfa-version`

Podczas merge pojawia się konflikt, ponieważ na dwóch gałęziach nadpisaliśmy ten sam plik `index.html` i repozytorium GIT nie wie jak ma go automatycznie połączyć – otrzymamy komunikat:

```
Auto-merging index.html
```

```
CONFLICT (content): Merge conflict in index.html
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

Otwieramy edytor tekstu i usuwamy konflikt ustalając ręcznie ostateczną wersję pliku.

Po edycji pliku `index.html` dodajemy go do śledzenia na tej gałęzi `git add *`

UŻYWAMY REPOZYTORIUM GIT 😊

Ponownie dokonujemy commita poprawionego pliku

```
git commit -m "Merged master with alfa-version"
```

Wysyłamy gałąź master do repozytorium zdalnego `git push origin master`

Wysyłamy także gałąź alfa-version do repozytorium zdalnego

```
git push origin alfa-version
```

Historię commitów na każdej gałęzi możesz sprawdzić w Twoim repozytorium

GitHub Pages pod adresem

<https://github.com/username/username.github.io>

Ostateczną wersję projektu możesz sprawdzić w działaniu na swoim serwerze WWW

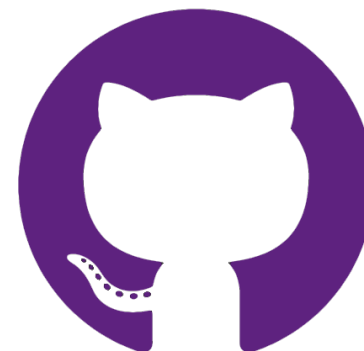
GitHub Pages pod adresem <https://username.github.io/>

GitHub Desktop

BEGIN NAVIGATION

Zainstaluj program GitHub Desktop. Posłuż on do wysyłania i pobierania kodu Twoich projektów z serwisu GitHub oraz do śledzenia zmian na poszczególnych etapach pracy poprzez aplikację okienkową

<https://desktop.github.com/>



Dodaj Twoje repozytorium GitHub Pages do programu GitHub Desktop

* **Podczas *dodawania istniejącego na dysku repozytorium*, lub klonowania już istniejącego z serwisu GitHub, pamiętaj, aby wskazać *Twój folder z repozytoriami na Pulpicie!* (`~/Desktop/repositories`)**

WARSZTATY ☺ Git Hub Pages

1. W folderze głównym Twojego repozytorium GitHub Pages stwórz folder `zadania-domowe`
2. W folderze `zadania-domowe` repozytorium GitHub Pages stwórz podfoldery `zadanie-1`, `zadanie-2`, `zadanie-3`
3. W każdym ze stworzonych w poprzednim punkcie podfolderze stwórz plik `index.html` z treścią `<p>Zadanie $</p>`, gdzie \$ to numer kolejnego zadania – ćwiczenie robimy w pełnej strukturze HTML5 w pliku `index.html`
4. Zrób `commit`, a potem zrób `push` do GitHub
5. Sprawdź w serwisie <http://github.com> w Twoim repozytorium GitHub Pages czy zmiany się wgrały?
6. Pobierz link do Twojej strony GitHub Pages w podfolderze z zadaniami np. <https://username.github.io/zadania-domowe/zadanie-1> wklej do przeglądarki i sprawdź, czy widzisz stworzone przez siebie strony?



Akademia 108

<https://akademia108.pl>