# Wstęp do Informatyki - Wykład 12

Repozytoria danych - Git

# Repozytoria danych

- Ważnym narzędziem pracy każdego informatyka są repozytoria danych (inna nazwa - systemy kontroli wersji, VCS).

- Repozytorium danych służy do zarządzania zestawem plików, które mogą być modyfikowane przez wielu użytkowników jednocześnie.

- Dobre repozytorium powinno zapewnić spójność i bezpieczeństwo danych.

# Repozytoria danych

- Podstawowym zastosowaniem repozytoriów danych jest zarządzanie projektami, w których tworzone jest wiele plików tekstowych - projektami programistycznymi.

- Mogą one jednak być stosowane także do edycji dokumentów (LaTeX).

# Git

- Git jest wiodącym repozytorium danych, stosowanym powszechnie w projektach zespołowych.
- Git jest oprogramowaniem typu open-source, udostępnionym na licencji GPL
- Alternatywą dla Gita jest Mercurial.
- Wcześniej popularne były CVS oraz SVN.

# Git

- Repozytorium danych Git może zostać utworzone w dowolnym katalogu na dysku.

- Aby je utworzyć, należy z wewnątrz tego katalogu wykonać polecenie **`git init`**

- Od tej pory wskazany katalog staje się repozytorium, które będzie śledziło wszystkie zmiany w każdym katalogu i każdym pliku.

# Git

Spróbujmy dodać nowy plik do repozytorium:

```
echo “hello” > file1.txt
```

Sprawdźmy, czy repozytorium “wie” o nowym pliku:

```
git status
```

```
Untracked files:

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

      file1.txt

nothing added to commit but untracked files present (use "git add" to track)
```

# Git

- Nasz plik widnieje w tym momencie jako “untracked”, czyli nieśledzony.

- Oznacza to, że repozytorium widzi go jako plik nowo dodany do katalogu, który nie jest jeszcze w repozytorium.

- Dodajmy go do repozytorium.

# Git

- Dodawanie pliku oraz wykonywanie wszelkich zmian w repozytorium jest możliwe dzięki operacji `commit`

- Commit jest wprowadzeniem danej zmiany do repozytorium.

- Operacja ta jest dwuetapowa - w pierwszym kroku oznaczamy, które zmiany chcemy commitować (*staging changes*), a w drugim kroku dokonujemy samego commita.

# Git

- Dodanie pliku file1.txt do commita (ale nie commitowanie!):

```
git add file1.txt
```

- Sprawdźmy, jaki jest teraz wynik komendy `git status`

```
Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

    new file:   file1.txt
```

# Git

- Zatem wszystko gotowe, można commitować:

```
git commit -m “dodanie pliku”
```

- Opcja `-m` służy do podania komunikatu. Warto dobrze opisywać zmiany, których dokonujemy. Często commit wprowadza zmiany do wielu plików. Sprawdźmy, jaki jest teraz wynik komendy `git status`

```
On branch master

nothing to commit, working directory clean
```

# Git

- A teraz zmodyfikujmy zawartość naszego pliku:
  `echo “hello again” >> file1.txt`
- Co teraz pokaże `git status?`

```
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)

    modified:   file1.txt
```

# Git

- Repozytorium wykryło zmianę w pliku, ale zmiana ta jeszcze nie będzie commitowana (nie jest *staged*)
- Aby dodać zmianę do commita używamy `git add file1.txt`, aby cofnąć: `git checkout file1.txt`
- Możemy również sprawdzić, co zmieniło się w pliku od ostatniego commita poprzez `git diff file1.txt:`

```
@@ -1 +1,2 @@

 hello

+hello again
```

# Git

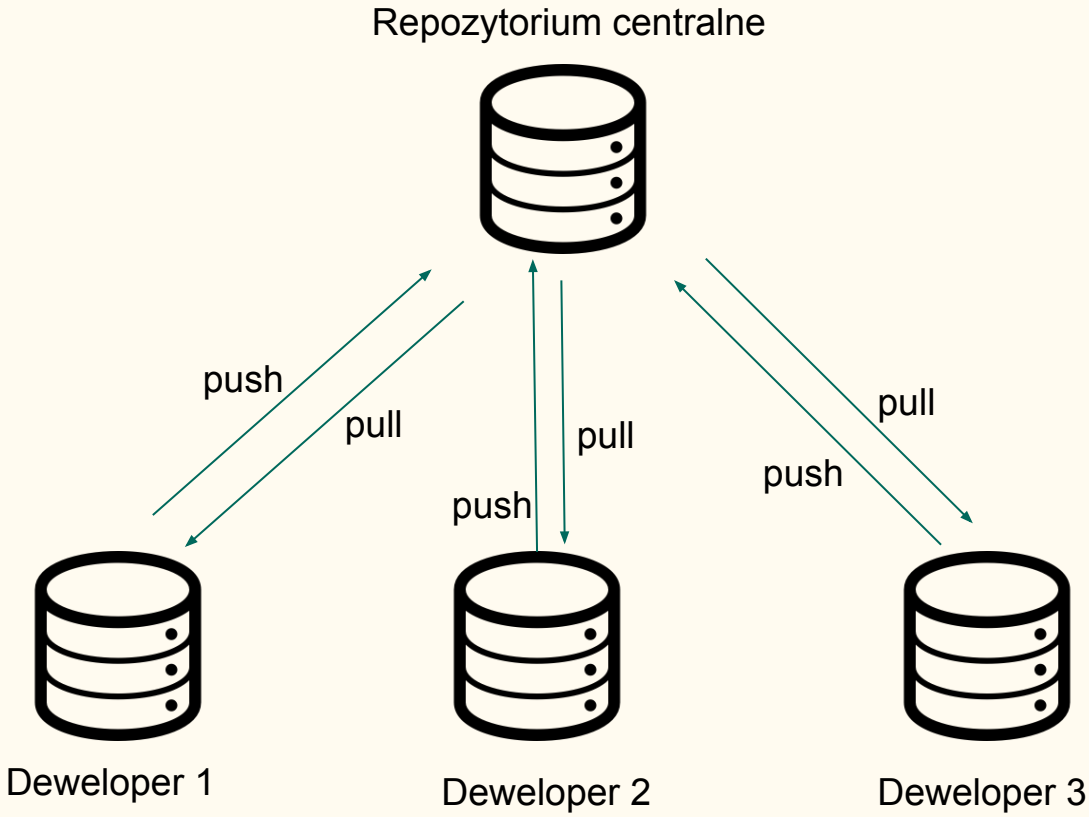
- Polecenie `git log` pokazuje listę commitów, które wykonano w danym repozytorium:

```
commit 588330c6b6d728f7fed2d7cea59a1e309291c625

Author: rjawor <rjawor@amu.edu.pl>

Date:   Tue Jan 2 22:36:54 2018 +0100

    dodanie pliku
```

# Git

- W ten sposób tworzymy i uaktualniamy nasze lokalne repozytorium.
- Co jednak w sytuacji, gdy chcemy pracować z innymi osobami?
- Git jest tzw. rozproszonym systemem kontroli wersji.

# Git


Repozytorium centralne
push
pull
push
pull
pull
push
Deweloper 1
Deweloper 2
Deweloper 3

# Git

- Każdy deweloper posiada swoją lokalną kopię centralnego repozytorium.

- Przed rozpoczęciem dnia pracy, deweloper wykonuje polecenie `git pull` i ściąga wszystkie nowe commity z centralnego repozytorium.

- Po zakończeniu pracy dokonuje operacji `git push`, wysyłając swoje nowe commity do centralnego repozytorium.

# Git

- Podczas operacji `git pull` Git automatycznie dokonuje tzw. scalenia zmian (commitów).
- Commity, które modyfikują różne pliki (niekonfliktujące), zostają scalone w sposób naturalny - powstaje commit, który wprowadza wszystkie te zmiany.
- Commity które modyfikują ten sam plik, ale w różnych miejscach, również są niekonfliktujące (jest to zasadnicza różnica w stosunku do starszych systemów, np. SVN).
- Problem pojawia się, gdy commity ściągane przez `git pull` modyfikują ten sam plik w tych samych liniach, co lokalne commity. Wówczas deweloper musi rozwiązać ten konflikt ręcznie i zrobić nowy commit.

# Git

- W ten sposób rozwiązuje się konflikty podczas `git pull`. Jak natomiast rozwiązuje się konflikty przy `git push`?

- Tu zasada jest prosta - Git nie pozwala wykonać operacji `git push`, jeśli nie wykonaliśmy wcześniej `git pull` i nie rozwiązaliśmy ewentualnych konfliktów.

# Git

- Dzięki swoim funkcjonalnościom Git zapewnia:
  - spójność danych (dzięki wymuszeniu rozwiązywania konfliktów)
  - bezpieczeństwo (każde lokalne repozytorium jest kopią zapasową)
  - łatwą współpracę wielu deweloperów (dzięki automatycznemu scalaniu zmian)

- Warto jak najlepiej poznać funkcje Git-a!

- Sugerowana dalsza lektura: `git branch` oraz `git merge`

# Dziękuję za uwagę!