



Git



Sztuka Wytwarzania Oprogramowania

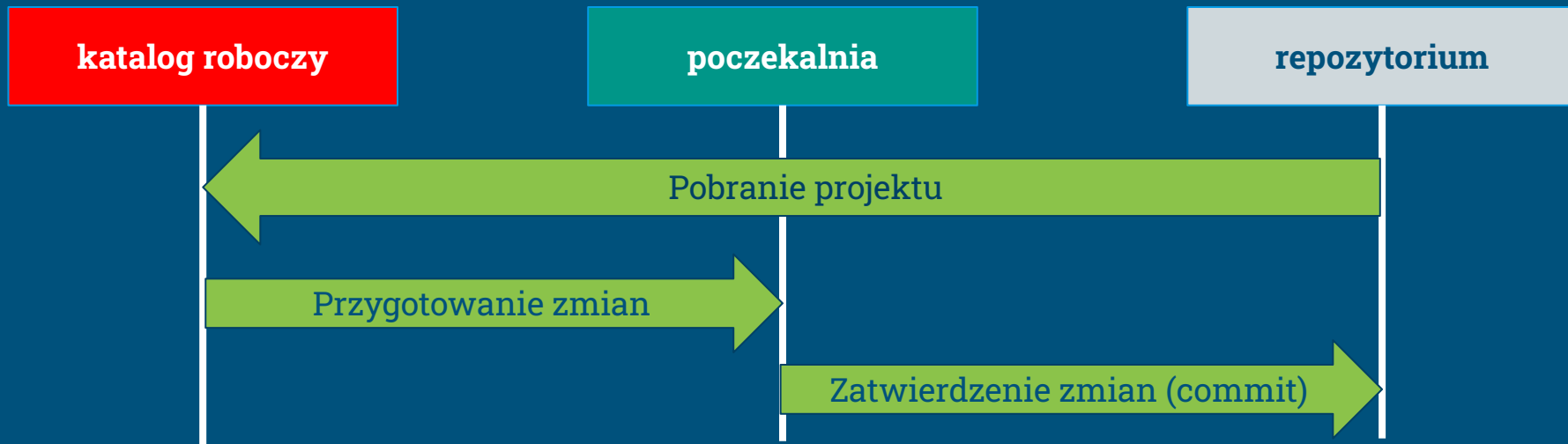


Git a inne systemy kontroli wersji

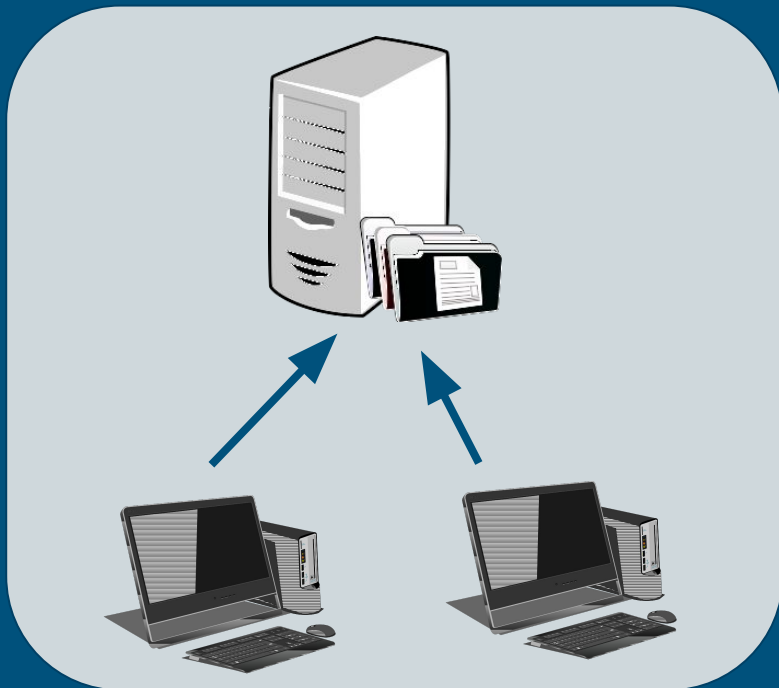
- Systemy kontroli wersji pomagają śledzić i zarządzać zmianami w kodzie źródłowym oprogramowania
- Tradycyjnie systemy śledziły zmiany na poziomie pojedynczych plików
- Innowacyjność gita polega na traktowaniu repozytorium jako “systemu plików”
 - git robi migawkę wszystkich plików naraz, a nie każdego z osobna

Podstawowe pojęcia

- składnica (repository)
- poczekalnia (staging)
- katalog roboczy (working directory)



Scentralizowana architektura (nie git)



- Repozytorium na centralnym serwerze
- Stacje robocze posiadają tylko wersję kodu, na której aktualnie pracują
- Większość operacji w oparciu o stan centralnego serwera
- Awaria serwera praktycznie uniemożliwia pracę (praca “wiecznie online”)
- Uszkodzenie serwera powoduje utratę danych

Rozproszona natura git

- brak centralnego serwera
- każda stacja robocza ma* pełną kopię repozytorium z całą historią zmian
 - może pełnić funkcję „serwera”
 - można z niej odtworzyć repozytorium
- większość operacji odbywa się lokalnie
- praca offline — zmiany „wysyłamy” do lokalnego repozytorium
- do katalogu roboczego można dodać dowolną liczbę zdalnych repozytoriów
 - tradycyjnie główne zdalne repozytorium nazywa się origin

Reprezentacja danych w git

- składnica przechowuje różne obiekty
 - commity
 - drzewa
 - pliki
- każdy obiekt ma niepowtarzalny identyfikator (hash)
- obiekty są dodawane do repozytorium ale w praktyce nie są usuwane
 - bardzo ciężko jest „zgubić” jakiekolwiek dane
- fizycznie commity opisują stan wirtualnego systemu plików
- **konceptyjnie myślimy o commicie jako o różnicy względem poprzedniego**

Reprezentacja danych w git - commit

- meta-dane commita
- identyfikator obiektu poprzednika (może być więcej niż jeden)
- identyfikator obiektu drzewa

```
tree beb30397e7dfb1e1e39224aeda442a7fa9cdf677
parent 9c2d7173337b340b0ff81df8afbdaa7592087bea
author Witold Wysota <witold.wysota@pw.edu.pl> 1678694127 +0100
committer Witold Wysota <witold.wysota@pw.edu.pl> 1678694127 +0100

Print "Hello world!"
```

- commity w repozytorium tworzą graf acykliczny
- gałąź (branch) jest wskaźnikiem na commit
 - wskaźniki to nazwy, które mogą być przesuwane na inne obiekty

Podstawowe operacje - add

- Dodanie bieżącej wersji pliku do poczekalni
- Rozpoczyna śledzenie pliku

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

```
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file:   main.cpp
```

- `git add -p` pozwala wybiórczo dodawać zmiany z plików

Podstawowe operacje - add

- Ten sam plik można mieć w trzech różnych wersjach niezależnie od siebie
 - repozytorium
 - poczekalnia
 - kopia robocza

Na gałęzi main

Zmiany do złożenia:

(użyj „git restore --staged <plik>...”, aby wycofać)

zmieniono: main.cpp

Zmiany nie przygotowane do złożenia:

(użyj „git add <plik>...”, żeby zmienić, co zostanie złożone)

(użyj „git restore <plik>...”, aby odrzucić zmiany w katalogu roboczym)

zmieniono: main.cpp

Podstawowe operacje - commit

- składa zmiany z poczekalni w repozytorium
- wymaga podania opisu zmiany
 - pierwsza linijka opisu może być traktowana jako tytuł zmiany
 - kolejne linie (najczęściej poprzedzone pustą linią) opisują szczegóły
- zmianę można poprawiać dodając opcję `--amend` (przydatne też `--no-edit`)

```
commit 81ea865a882e787fd2c6df76f52e215180a2a01f (HEAD -> main)
Author: Witold Wysota <witold.wysota@pw.edu.pl>
Date:   Sun Mar 12 23:08:02 2023 +0100

    main.cpp file added to the project

    - empty main() function
    - returns int
```

Podstawowe operacje - stash

- Odkłada (chomikuje) zmiany “na półkę” i przywraca czysty stan katalogu roboczego
- Kolejne tego typu operacje odkładane są na listę
- Zachomikowaną zmianę można przywrócić, podejrzeć albo usunąć
 - `git stash pop`
 - `git stash show`
 - `git stash drop`
- Ta część składnicy jest osobna od głównego repozytorium i jego stanu

Podstawowe operacje - status

- W dowolnej chwili pozwala podejrzeć stan katalogu roboczego
- Jak się pogubimy, warto zapytać o status — git często podpowiada, jakie mamy możliwości

```
trwa interaktywne przestawianie na 033247c
Last command done (1 command done):
  pick c2823ab Return 1
Brak pozostałych poleceń.
Przestawiasz właśnie gałąź „other” na „033247c”.
(napraw konflikty i wykonaj „git rebase --continue”)
(użyj „git rebase --skip” aby pominąć tę łatkę)
(użyj „git rebase --abort”, aby wybrać pierwotną gałąź)

Niescalone ścieżki
(użyj „git restore --staged <plik>...”, aby wycofać)
(użyj „git add <plik>...” aby zaznaczyć rozwiązanie)
    oba zmienione:      main.cpp

brak zmian dodanych do zapisu (użyj „git add” i/lub „git commit -a”)
```

Podstawowe operacje - push

- Przesyła lokalną gałąź do zdalnego repozytorium
- Operacja udaje się jedynie wtedy, gdy repozytoria mają wspólną historię

Podstawowe operacje - pull

- Składa się z dwóch operacji
 - `git fetch` — pobiera obiekty ze zdalnego repozytorium
 - `git merge` lub `git rebase` — uzgadniają historię pomiędzy zdalną i lokalną gałęzią

Git w praktyce (w projekcie wieloosobowym)

- Rozwój danej funkcji oprogramowania realizowany na dedykowanej gałęzi
- Autorzy składają zmiany w swoich lokalnych repozytoriach
- Co jakiś czas (i na koniec pracy) synchronizują zmiany pomiędzy sobą
 - Synchronizacja polega na uzgodnieniu historii zmian
 - Może powodować kolizje (ja zmieniłem plik x i ty zmieniłeś plik x, która wersja obowiązuje?)
- Po zakończonej pracy należy zintegrować zmiany z główną (lub inną) gałęzią repozytorium
- Istnieją dwie strategie integracji
 - merge
 - rebase

Git merge

- łatwiejsza strategia do przeprowadzenia
- konflikty rozwiązywane jednokrotnie pomiędzy końcowymi wersjami kodu
- w składnicy dodawany jest sztuczny obiekt, którego rodzicami są łączone commity
- nie zachowuje historii zmian

Git merge

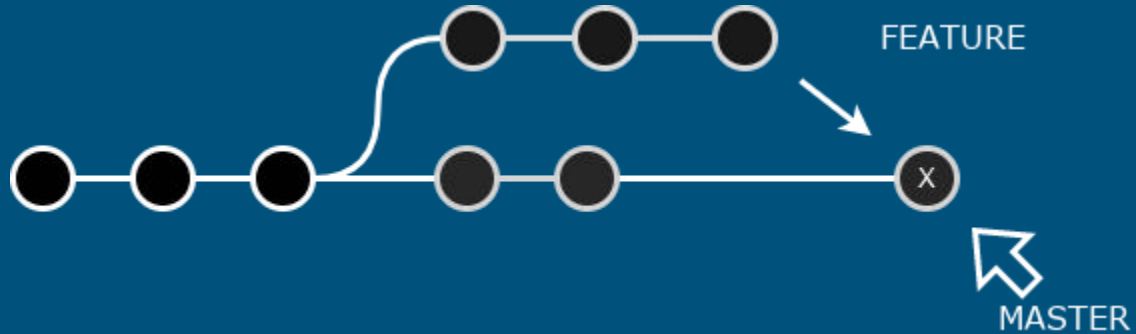
```
wysota@dell:/swo/git$ git log --all --oneline --graph
* c422d79 (HEAD -> m2) m2.cpp added
| * eb306ee (m1) m1.cpp added
|/
* 81ea865 (main) main.cpp file added to the project
```

```
wysota@dell:/swo/git$ git log --all --oneline --graph
* 7c29ba3 (HEAD -> m2) Merge branch 'm1' into m2
| \
| * eb306ee (m1) m1.cpp added
* | c422d79 m2.cpp added
|/
* 81ea865 (main) main.cpp file added to the project
```

```
commit 7c29ba377cc17db2b8bed81d59e4720f97a7f21c (HEAD -> m2)
Merge: c422d79 eb306ee
Author: Witold Wysota <witold.wysota@pw.edu.pl>
Date: Sun Mar 12 23:49:10 2023 +0100
```

Merge branch 'm1' into m2

Polityka "merge"



Git rebase

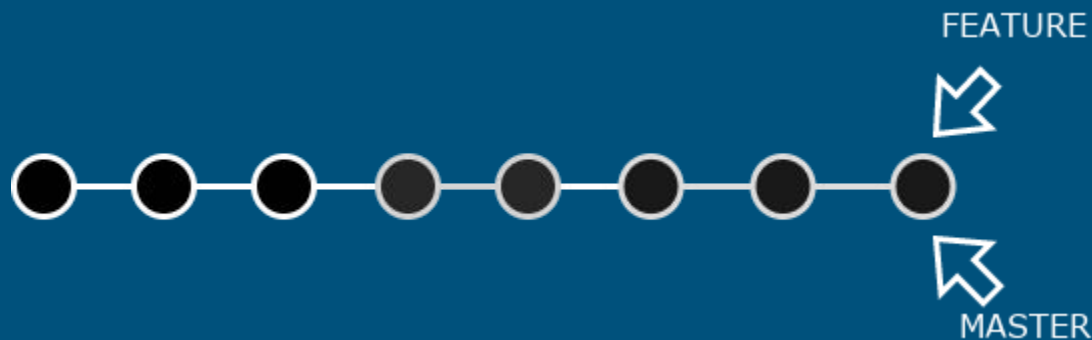
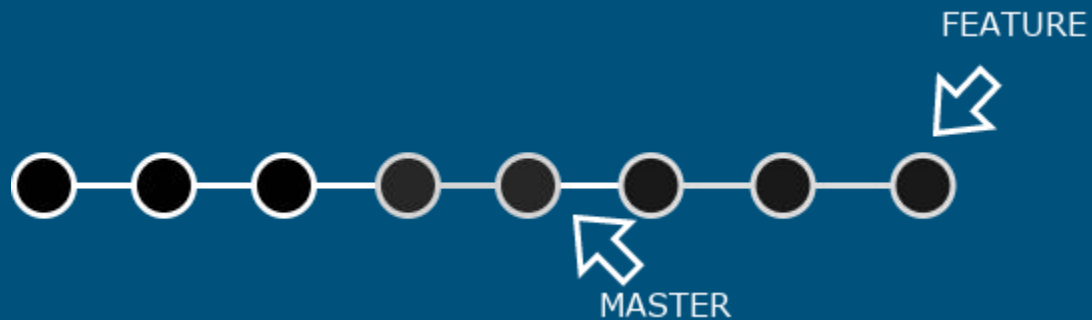
- trudniejsza strategia, ale zachowuje liniowość historii
- zmiany nakładane są commit po commicie
 - przy każdym kolejnym commicie mogą wystąpić konflikty, które trzeba rozwiązać
- efektywnie przepinamy całą naszą gałąź na koniec gałęzi źródłowej

Git rebase

```
wysota@dell:/swo/git$ git log --all --oneline --graph
* 85fdf81 (HEAD -> r2) r2.cpp added
| * 5a20196 (r1) r1.cpp added
|/
* 81ea865 (main) main.cpp file added to the project
```

```
wysota@dell:/swo/git$ git checkout r1
Switched to branch 'r1'
wysota@dell:/swo/git$ git rebase r2
Successfully rebased and updated refs/heads/r1.
wysota@dell:/swo/git$ git log --all --oneline --graph
* 4b33283 (HEAD -> r1) r1.cpp added
* 85fdf81 (r2) r2.cpp added
* 81ea865 (main) main.cpp file added to the project
```

Polityka “rebase”



Narzędzia git

Git posiada wiele wbudowanych narzędzie do wykonywania różnych operacji

- `cherry-pick` - nakłada zestaw commitów na gałąź
- `bisect` - pomaga znaleźć commit z błędem
- `diff` - pokazuje różnice pomiędzy dwiema gałęziami
- `grep` - wyszukuje wzorce w repozytorium
- `annotate/blame` - pokazuje kto i kiedy zmienił daną linię pliku
- `apply` - nakłada łatkę bez tworzenia commita

git bisect

```
git bisect start bad_commit good_commit
```

```
git bisect good
```

```
git bisect bad
```

```
git bisect reset
```

Literatura

- Git Pro Book (Scott Chacon, Ben Straub) - <https://git-scm.com/book/en/v2>
- Atlassian git tutorials - <https://www.atlassian.com/git/tutorials>