

GitHub

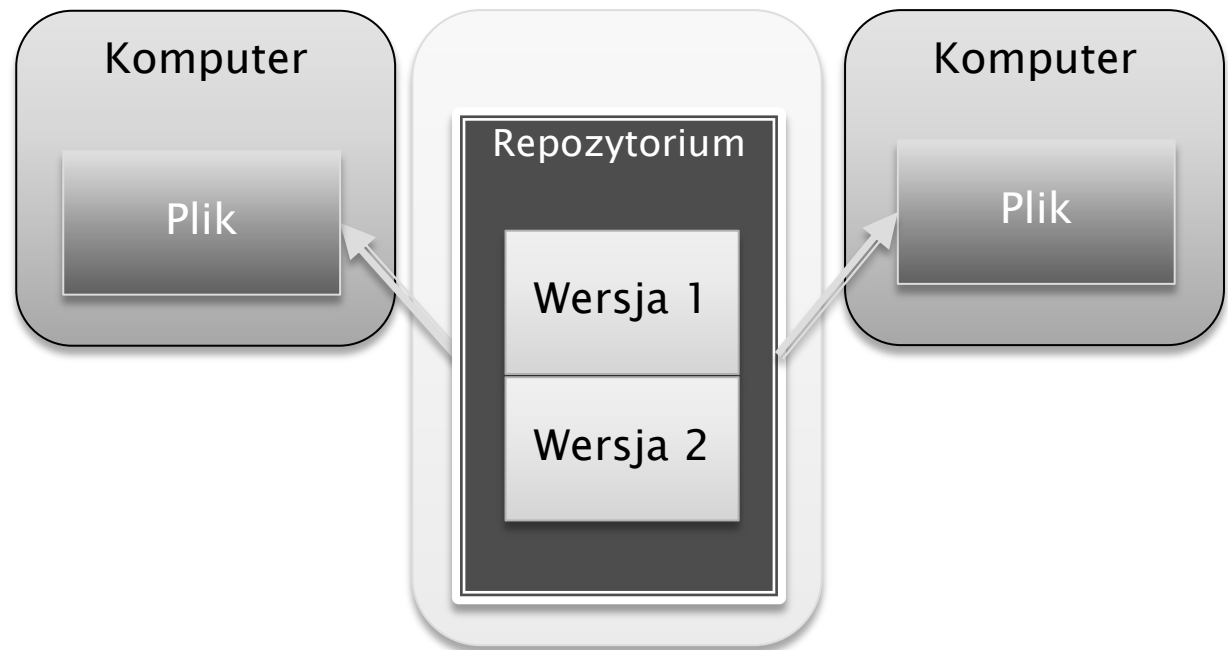
Czym jest git?

- ▶ Git to rozproszony system kontroli wersji
- ▶ Pozwala m.in. na:
 - śledzenie zmian w plikach
 - łatwy powrót do wcześniejszej wersji pliku
 - sprawną zdalną współpracę
- ▶ Projekt opiera się na
 - Szybkości
 - Prostocie
 - Wsparcie dla nieliniowego wytwarzania oprogramowania
 - Całkowite rozproszenie
 - Efektywnym przechowywaniu dużych projektów (np. kernel Linuksa)

Modele pracy

- ❑ Model Scentralizowany

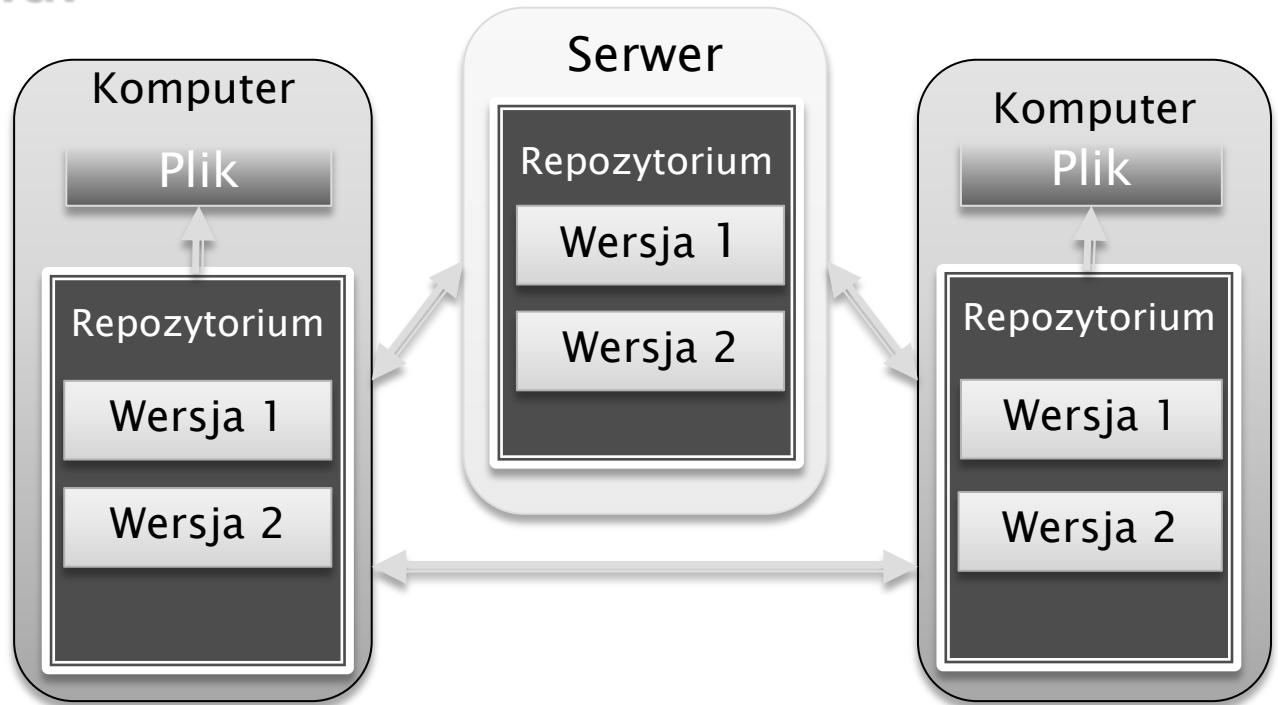
- ❑ SVN
- ❑ CVS
- ❑ Perforce



- Używana jako miejsce przechowywania kopii zapasowej (Backupu)
- Brak prywatnych branch'y
- Do wykonania większości standardowych operacji wymaga połączenia do serwera.

Modele pracy c.d.

- Model Rozproszony
 - GIT
 - Mercurial
 - Bazaar



- Możliwość pracy na wielu zdalnych repozytoriach.
- Wsparcie dla lokalnych/prywatnych branch'y.
- Nie potrzebuje dostępu do serwera dla większości operacji
- Pełna historia zmian dostępna lokalnie.

Możliwości GIT'a

- ▶ Rozproszony model pracy
- ▶ Wsparcie dla nieliniowego programowania (branche)
- ▶ Wydajny w przypadku dużych projektów
- ▶ Publikacja repozytorium (git://, http(s)://, ssh://)
- ▶ Adresowanie przez zawartość (SHA-1)
- ▶ Praca lokalna bez połączenia z repozytorium
 - Większość operacji nie wymaga połączenia z serwerem.

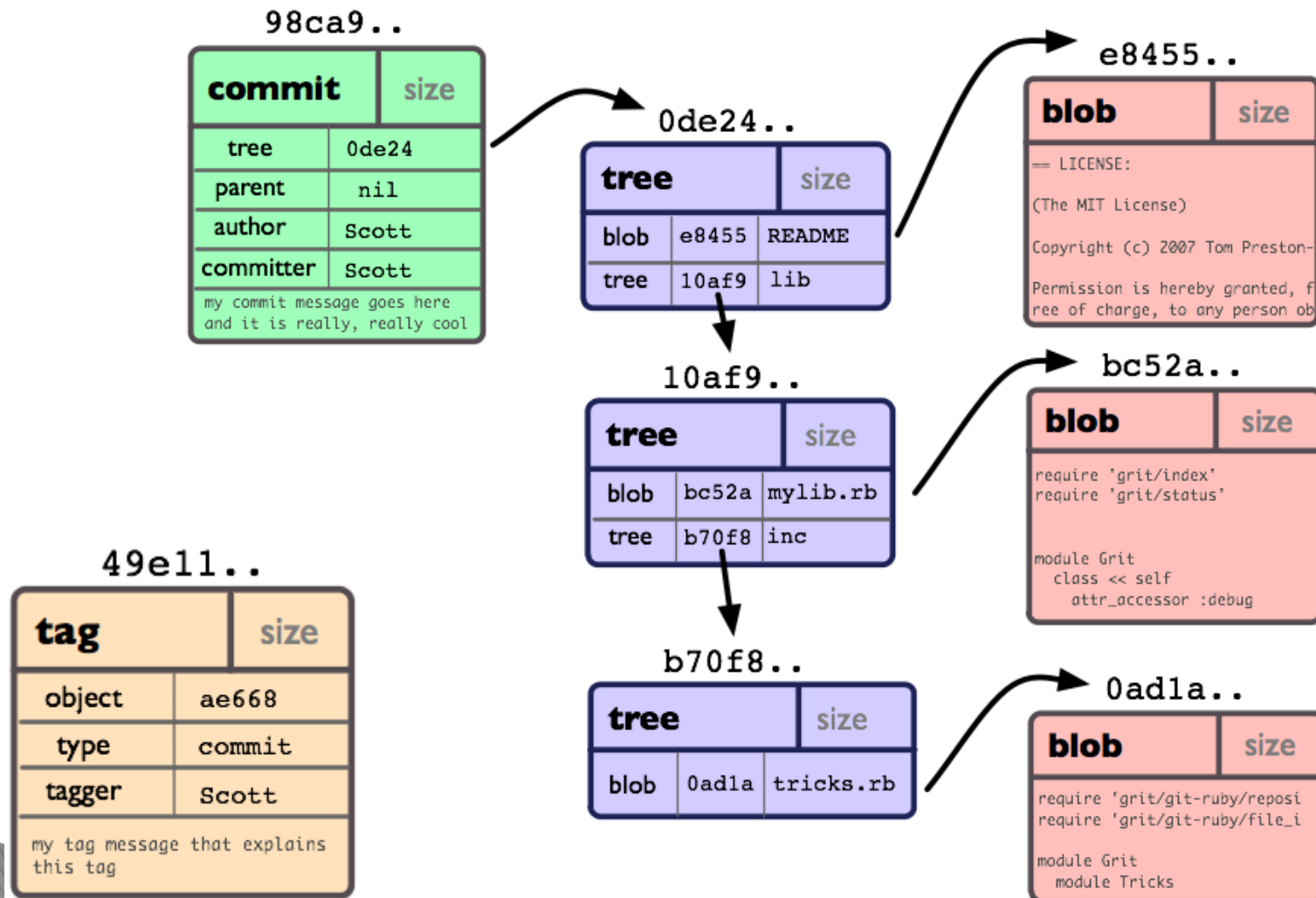
Terminologia

- ❑ Branch – równoległa gałąź projektu rozwijana oddzielnie od głównej.
- ❑ Tag – marker konkretnej wersji (rewizja w SVN'ie) projektu.
- ❑ Working Dir – katalog roboczy na którym pracujemy
- ❑ Index – rodzaj „cache”, czyli miejsca gdzie trzymane są zmiany do commita
- ❑ Master Branch – główny branch z którym łączymy (merge) nasze zmiany przed wysłaniem do zdalnego repozytorium.

Obiekty GIT'a

- Commit – wskazuje na tree oraz ojca, zawiera przykładowo takie informacje jak autor, data i treść wiadomości.
- Tree – reprezentuje stan pojedynczego katalogu (lista obiektów blob oraz zagnieżdżonych obiektów tree)
- Blob – zawiera zawartość pliku bez żadnej dodatkowej struktury
- Tag – wskazuje na konkretny commit oraz zawiera opis taga.

Obiekty GIT'a cd.



Struktura zmian w GIT'ie

1. Przechowywanie zawartości projektu jako „snapshot’ów”.
2. Kompresowanie zawartości projektu.

Zmiany w czasie



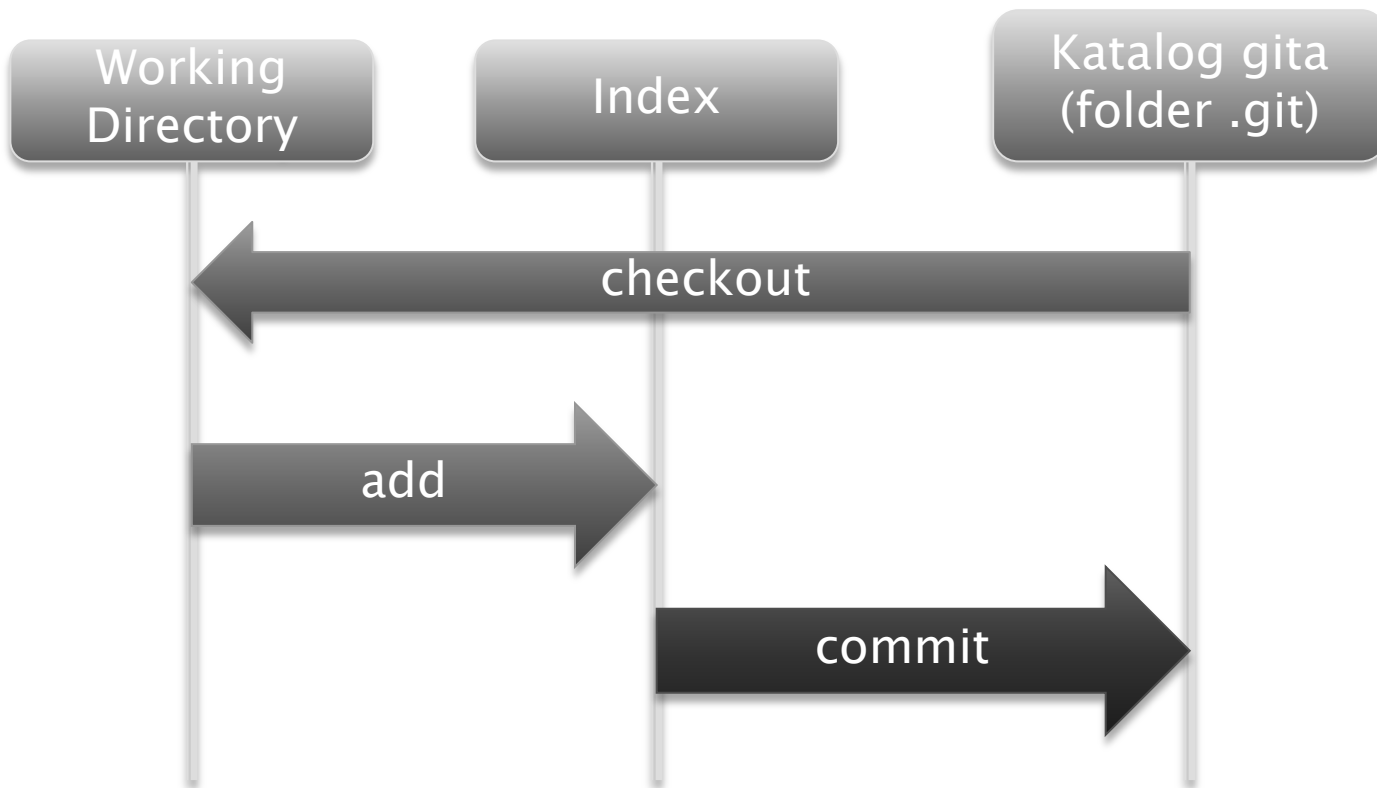
Operacje zdalne

► Przykładowymi operacjami zdalnymi są:

- git clone – pobiera zdalne repozytorium do podanego folderu
- git fetch – pobiera obiekty i wskaźniki z innego repozytorium
- git pull – pobiera i integruje obiekty i wskaźniki z innego repozytorium
- git push – aktualizuje zdalne repozytorium o wskaźniki i powiązane obiekty.

Operacje lokalne

- ▶ Większość operacji wykonywanych jest na lokalnym repozytorium.



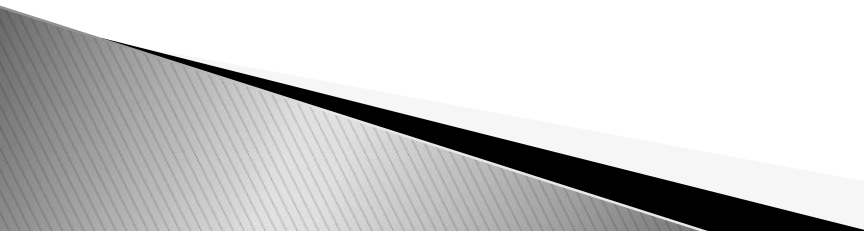
Podstawowe operacje

- `git init` – stworzenie nowego repozytorium
- `git add` – dodanie zawartości pliku do Index'u
- `git rm` – usuwa plik z indexu
(plik zniknie z working directory po commit'ie)
- `git mv` – przenosi plik
- `git status` – pokazuje status katalogu roboczego i poczekalni
- `git config` – pobiera i ustawia opcje globalne GIT'a lub tylko repozytorium

Podstawowe operacje cd.

- `git commit` – zapisuje zmiany do repozytorium lokalnego
- `git log` – wyświetla logi z commit'ów
- `git show` – wyświetla obiekt
- `git fetch` – pobiera zmiany z repozytorium zdalnego
- `git pull` – wywołuje polecenia `fetch` i `merge`
- `git push` – wysyła zmiany do zdalnego repozytorium

Podstawowe operacje cd.

- `git branch` – do zarządzania branch'ami
 - `git checkout` – przełączanie się między branch'ami
 - `git merge` – łączy podane branch'e
 - `git rebase` – zmienia punkt startu dla branch'a
 - `git reset` – przywraca stan katalogu roboczego
 - `git stash` – zapisuje/odczytuje zmiany z przestrzeni tymczasowej (rodzaj schowka)
 - `git gc` – porządkowanie i optymalizacja repozytorium
- 

Prezentacja wykorzystania

- ▶ Stworzenie zdalnego repozytorium :

- > git init

- ▶ Dodanie nowych plików:

- ▶ > git add .

- ▶ > git add readme.txt

- ▶ Struktura projektu. (Plik .git/config)

- ▶ > git config

Prezentacja wykorzystania cd.

- ▶ Pierwszy commit:
 - ▶ > git commit -m „Treść wiadomości”
- ▶ Historia :
 - ▶ > git log

Prezentacja wykorzystania

- ▶ Stworzenie tag'a:
 - ▶ > git tag v1.00
 - ▶ > git tag v2.00 -m „Wersja druga”
- ▶ Wypisanie tagów:
 - ▶ > git tag
- ▶ Wysłanie do zdalnego repozytorium tagów:
 - ▶ > git push -tags
- ▶ Usunięcie taga:
 - ▶ > git tag -d v2.00

Prezentacja wykorzystania

- ▶ Tworzenie brancha:
 - ▶ > git branch myBranch
- ▶ Wyświetlenie branchy:
 - ▶ > git branch
- ▶ Przełączanie branchy:
 - ▶ > git checkout myBranch
- ▶ Usuwanie brancha:
 - ▶ > git branch -d myBranch

Prezentacja wykorzystania

- ▶ Łączenie (merge) branchy:
 - ▶ > git merge branchName
- ▶ W przypadku konfliktów po poprawkach w łatwy sposób można kontynuować pracę:
 - ▶ > git commit -a -m „Merge branchy”

Prezentacja wykorzystania

- ▶ Historia zmian:
 - ▶ > git log
 - ▶ > git log --after=22.06.2014.19:20
 - ▶ > git log <nazwa_pliku>
- ▶ Wyszukiwanie odpowiedzialnej osoby:
 - ▶ > git blame <nazwa_pliku>
 - ▶ > git blame -L 12,3 <nazwa_pliku>

Podsumowanie

- ❑ Tworzenie i łączenie (merge) branch'y jest szybkie i łatwe.
- ❑ Możliwość wymiany kodu pomiędzy zdalnymi repozytoriami. (Udostępnienie bezpośrednio dla kogoś swojego prywatnego branch'a)
- ❑ Łatwe prototypowanie. (Prywatne branche)
- ❑ Można w łatwy sposób dostosować repozytoria do swojego procesu wytwarzania kodu.
- ❑ Możliwość automatycznego migrowania repozytorium SVN do GIT'a.
- ❑ Projekt OpenSource