

GIT – rozproszony system kontroli wersji



System kontroli wersji

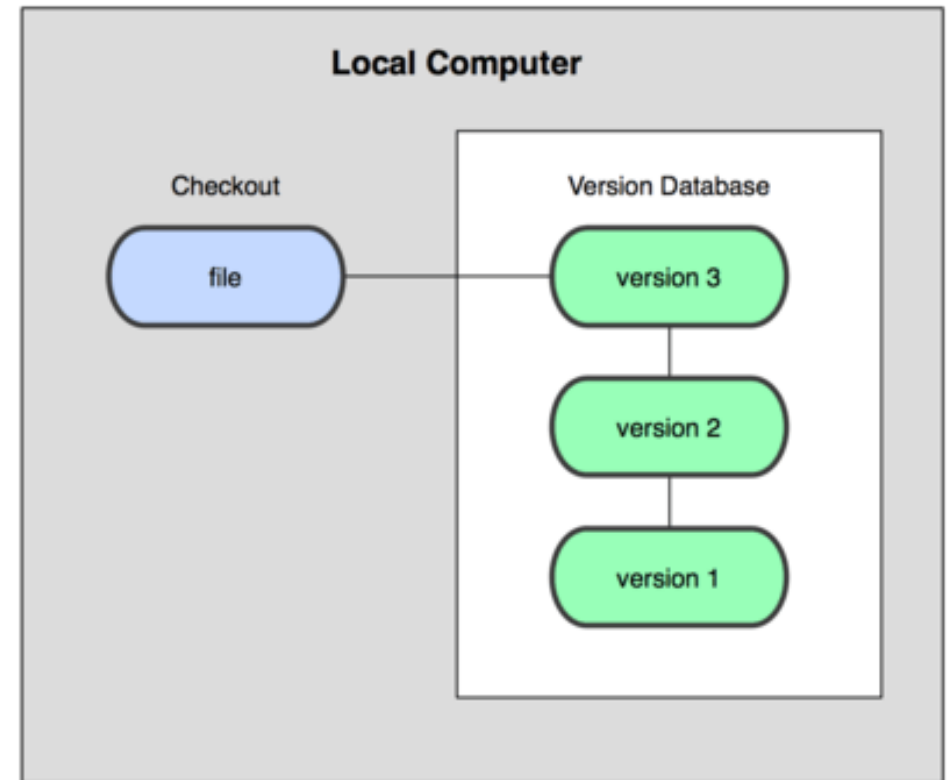
Oprogramowanie, które śledzi wszystkie zmiany dokonywane na plikach i umożliwia przywrócenie ich poprzedniej wersji

Pozwala na:

- Przywrócenie pliku do poprzedniej wersji
- Porównanie wprowadzonych zmian
- Uzyskanie informacji o tym kto i kiedy zmodyfikował część projektu
- W wypadku utraty danych wspiera ich względnie proste odzyskanie

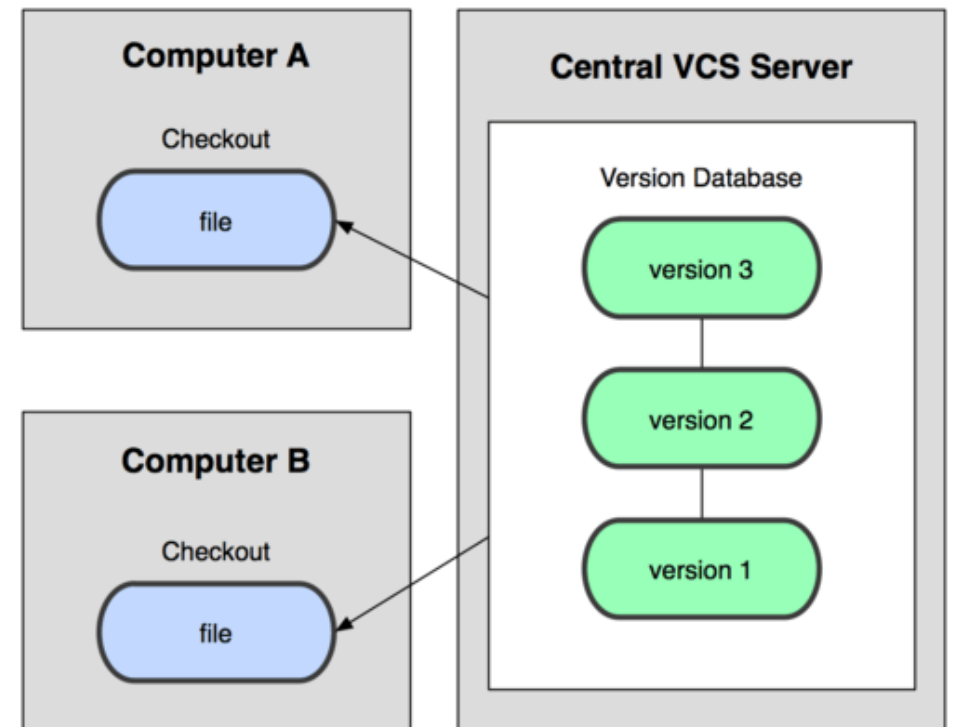
Typy systemów wersji: lokalne

- Zmiany różnicowe są zapisywane w lokalnej bazie danych
- Przykład: rcs



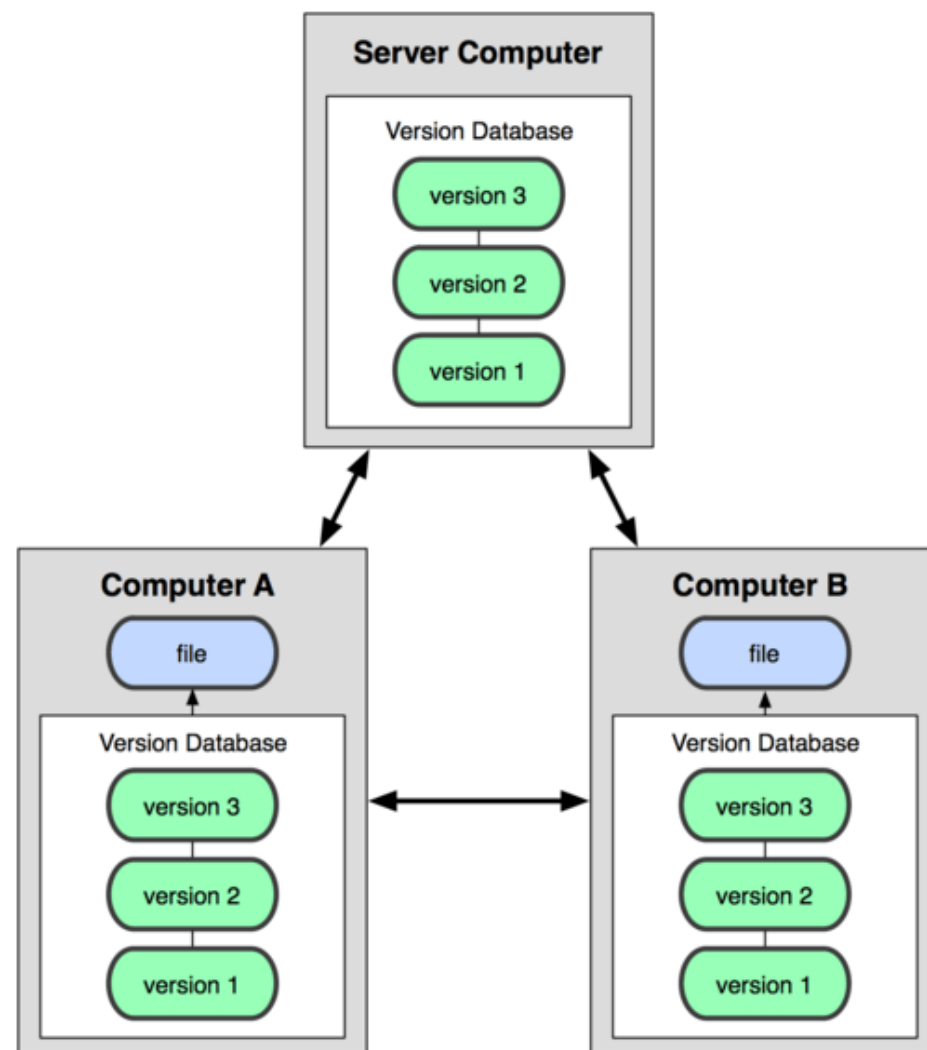
Typy systemów wersji: scentralizowane

- Serwer centralny zawierający wersjonowane pliki
- Klienci pobierający najnowsze wersje plików z serwera
- Przykłady: Subversion, Perforce
- Minus: awaria serwera – utrata całej historii



Typy systemów wersji: rozproszone

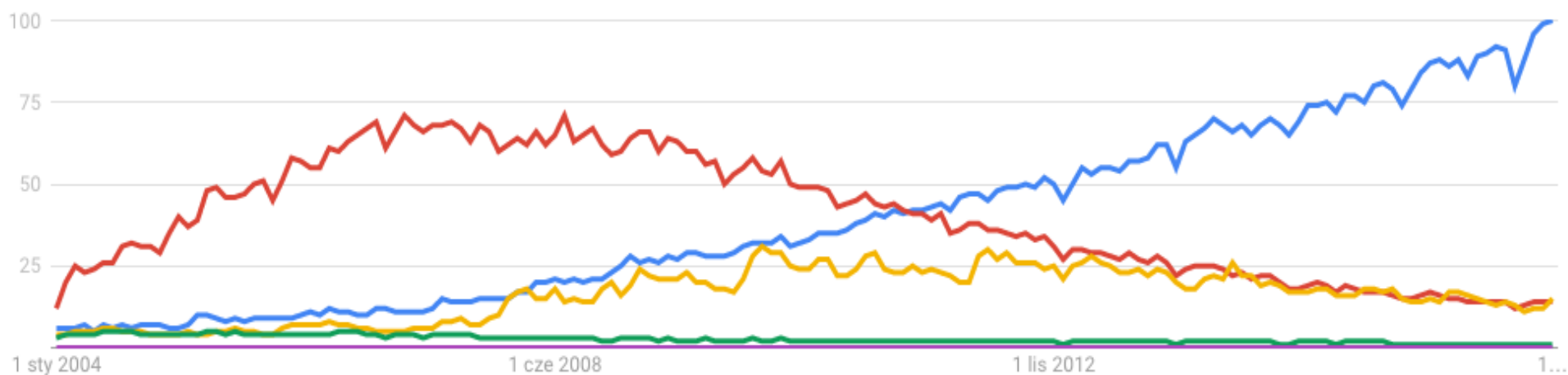
- Klienci kopiują całe repozytorium
- Łatwe przywracanie danych w wypadku awarii serwera
- Możliwość korzystania z kilku zdalnych repozytoriów
- Przykłady: Git, Mercurial



Git - historia

- Projekt kodu jądra Linuksa
- 2005 rok – cofnięto pozwolenie na nieodpłatne używanie systemu kontroli wersji Bitkeeper
- Linus Torvalds tworzy Git
- Zakłada, że system musi być:
 - Szybki
 - Mieć prostą konstrukcję
 - Silne wsparcie dla nieliniowego rozwoju
 - W pełni rozproszony
 - Wydajnie obsługiwać duże projekty

Google Trends



● Git

Oprogramowanie

● Subversion

Oprogramowanie

Oprogramowanie

● Mercurial

Wyszukiwane hasło

Wyszukiwane hasło

● Perforce

Wyszukiwane hasło

Wyszukiwane hasło

● Concurrent Version ...

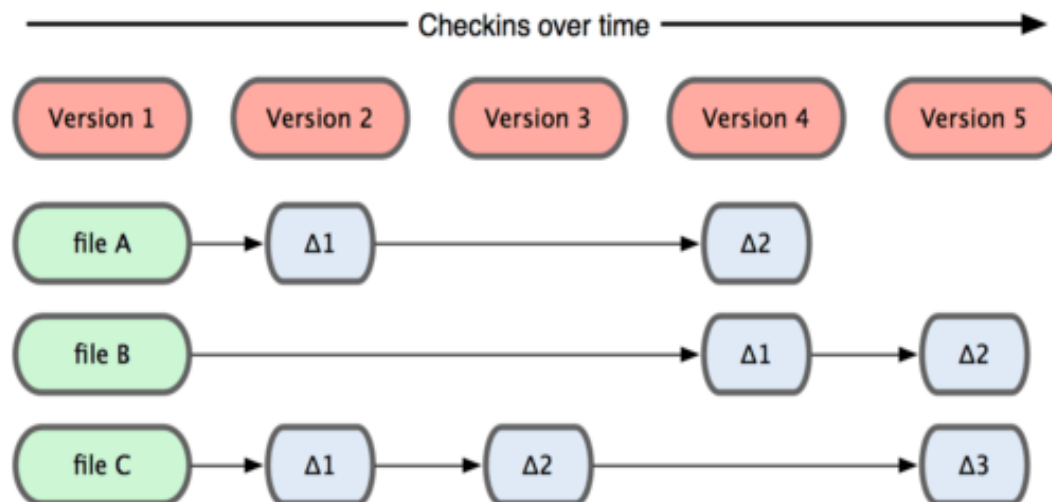
Wyszukiwane hasło

Wyszukiwane hasło

Przechowywanie danych w Git

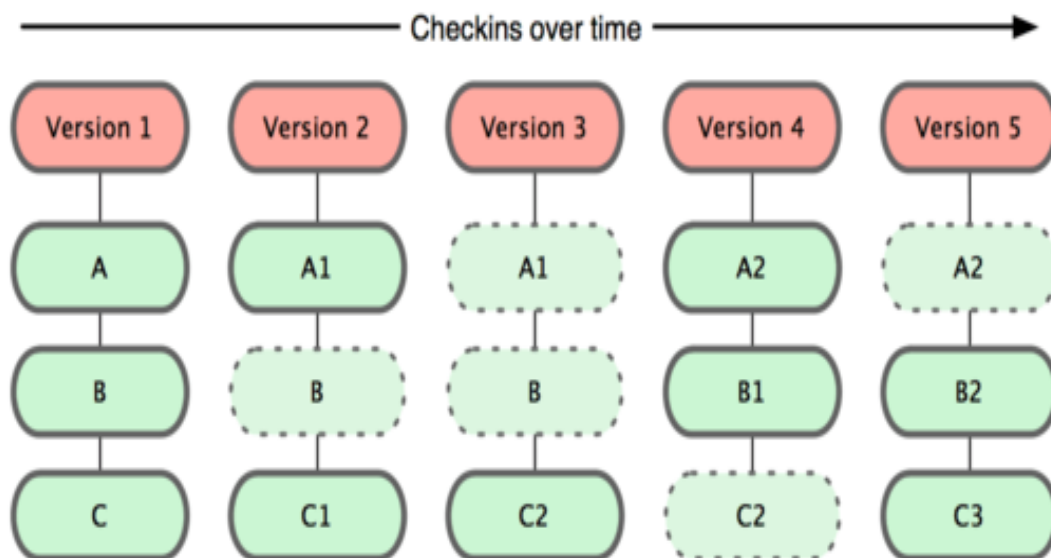
Tradycyjne podejście

Przechowywanie zmian dokonywanych na pliku



Rozwiązanie w Git

Przechowywane są „migawki” (snapshot) obrazu plików w danym Momencie. Git przechowuje referencję do migawki. Gdy plik nie został zmieniony zapisywana jest referencja do poprzedniej wersji pliku



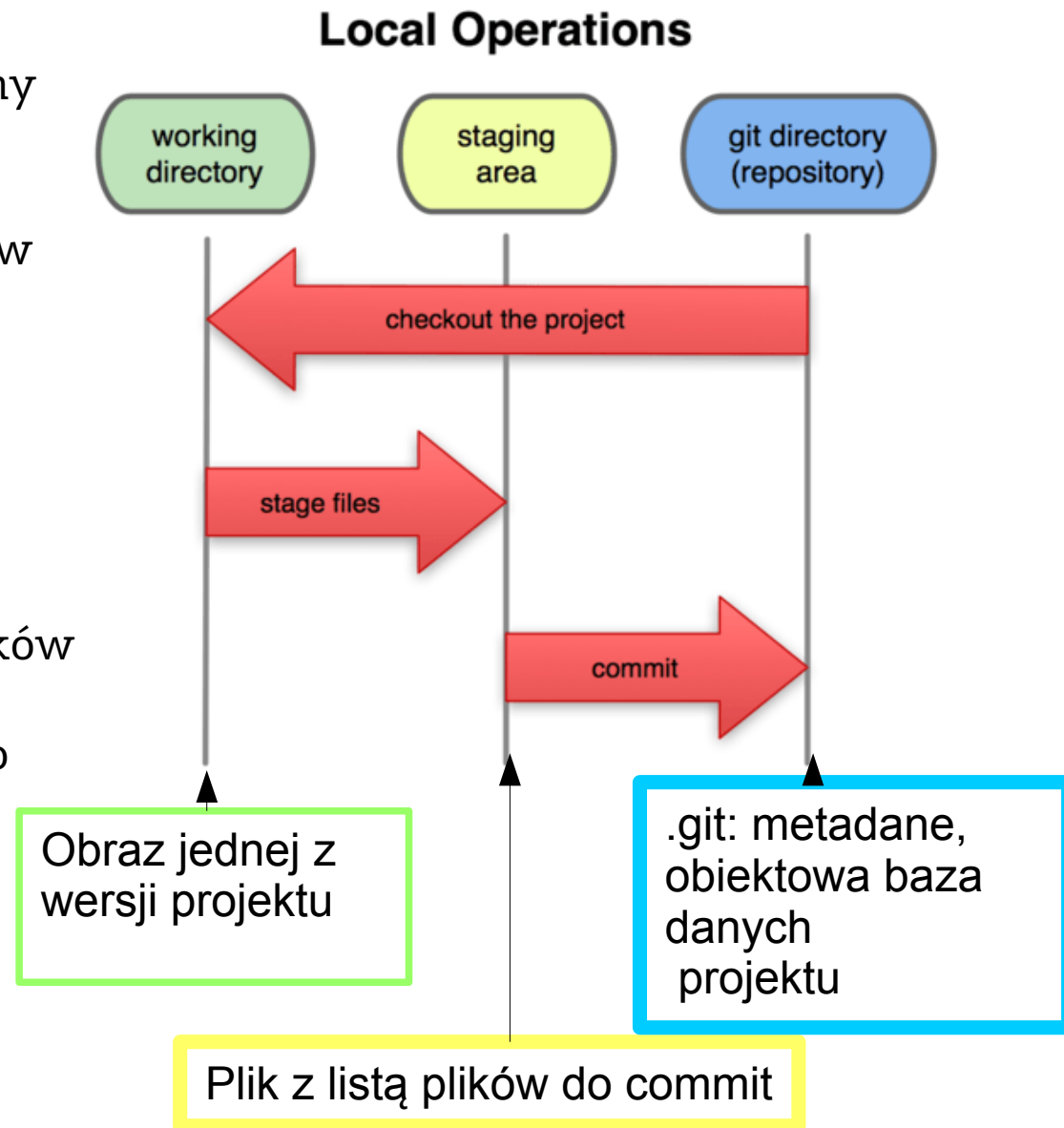
Inne istotne cechy

- Większość operacji jest lokalna
 - np. historia projektu (na podstawie lokalnej bazy)
 - Różnica pomiędzy wersjami projektu
 - Możliwa praca offline
- Wbudowane mechanizmy spójności danych
 - Dla każdego obiektu Git przed jego zapisem wyliczana jest suma kontrolna (SHA-1)
 - Do obiektu odwołujemy się na podstawie sumy kontrolnej

Stany pliku w Git

- **Zmodyfikowany (modified)** – zmieniony plik
- **Śledzony (tracked)** – zmodyfikowany plik, który został dodany do listy plików commitowanych w najbliższym czasie
- **Zatwierdzony (committed)** – plik który został dodany do lokalnej bazy

1. Modyfikacja pliku
2. Oznaczenie zmodyfikowanych plików jako śledzone (git add)
3. Zatwierdzenie/zapisanie plików do lokalnej bazy (git commit)



Tworzenie repozytorium

Importowanie istniejącego katalogu do Gita

```
$ git init
```

```
$ git add plik1.c
```

```
$ git add katalog
```

```
$ git commit
```

```
(git commit -m)
```

```
(git commit -a)
```

```
$ git status
```

```
$ git log
```

Sklonowanie istniejącego repozytorium z serwera

```
$ git clone [url] [katalog]
```

np.:

```
$ git clone  
git://github.com/schacon/g  
rit.git mygit
```

Protokoły transmisji:

- git://
- https://
- ssh
uzytkownik@serwer:/scie
zka.git

Ignorowanie plików

- Utworzenie pliku .gitignore

```
# komentarz – ta linia jest ignorowana
# żadnych plików .a
*.a
# ale uwzględniaj lib.a, pomimo ignorowania .a w linii powyżej
!lib.a
# ignoruj plik TODO w katalogu głównym, ale nie podkatalog/TODO
/TODO
# ignoruj wszystkie pliki znajdujące się w katalogu build/
build/
# ignoruj doc/notatki.txt, ale nie doc/server/arch.txt
doc/*.txt
```

Oglądanie zmian

- `Git status` – ogólne zmiany na poziomie plików
- `Git diff` – zmiany w kodzie nie wysłane do poczekalni
- `Git diff --staged` – zmiany w poczekalni

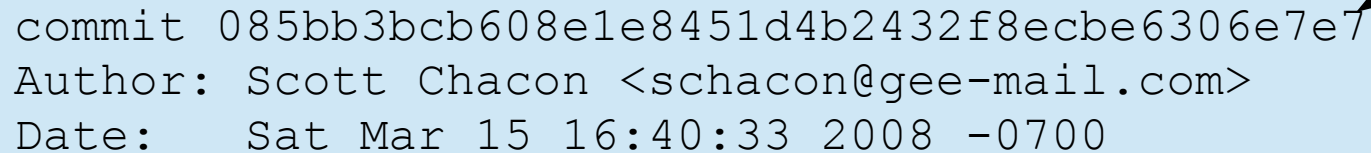
Usuwanie i przenoszenie plików

- `Git rm` – usunięcie ze zbioru plików śledzonych i z katalogu roboczego
- `Git rm -f` – wymuszanie usunięcia pliku śledzonego
- `Git rm --cached` – zachowuje plik w drzewie roboczym, wyłącza śledzenie
- `Git mv file_a file_b` – zmiana nazwy pliku
zapamiętana w poczekalni – równoważne sekwencji
(`git mv`, `git rm file_a`, `git add file_b`)

Przeglądanie historii

- git log
- Git log -p – dokładna historia zmian

SHA-1



```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700
```

```
    removed unnecessary test code
```

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
```

```
    first commit
```

Cofanie zmian

- Zmiana ostatniej rewizji:
 - **git commit - -amend** (bierze zawartość „poczekalni” i zatwierdza jako dodatkową zmianę).
- Usuwanie pliku z „poczekalni”
 - **git reset HEAD <file>**
- Cofanie zmian w zmodyfikowanym pliku
 - **git checkout - - <file>**

Praca ze zdalnym repozytorium

- Wyświetlanie zdalnych repozytoriów:
 - `git remote [-v]`

```
origin  git://github.com/schacon/ticgit.git
```

Domyślna nazwa

- Dodawanie zdalnych repozytoriów
 - `git remote add [skrót] [url]`

Praca ze dalnym repozytorium

- Pobieranie zmian z repozytorium zdalnego
 - **git fetch [nazwa-zdalnego-repozytorium]** (nie scala danych)
 - **git pull [remote name] [branch name]** (próbuje scalić pobrane dane z lokalnymi)
- Wypychanie danych na dalny serwer
 - **git push [remote name] [branch name]**
- Zmiana nazwy odnośnika do repozytorium
 - **git remote rename [old name] [new name]**
 - **git remote rm [name]** – usunięcie odnośnika

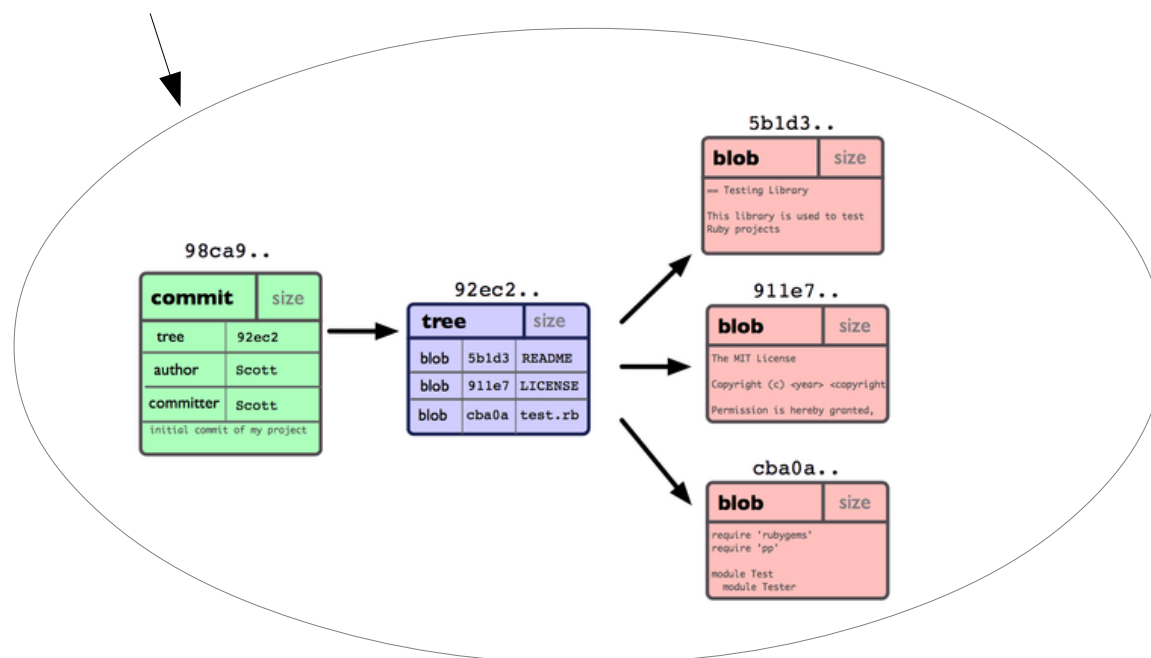
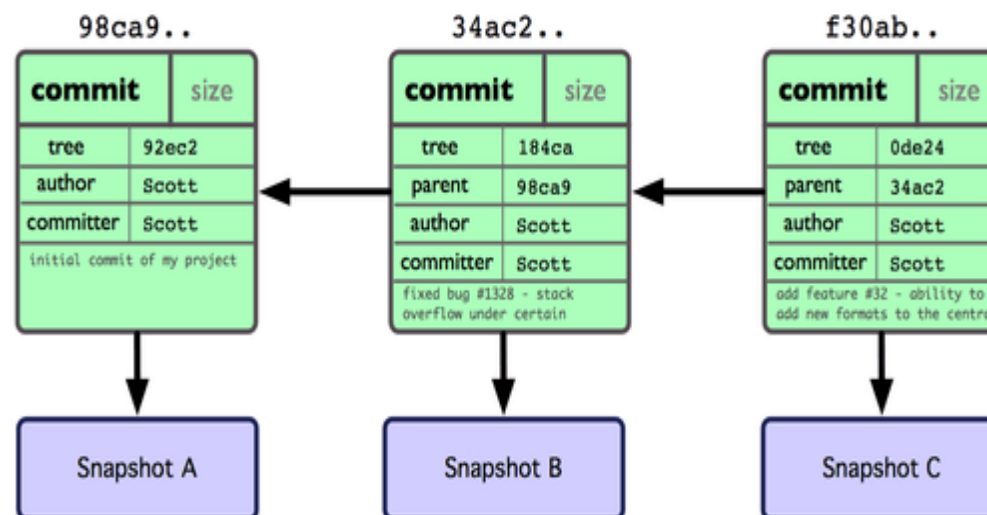
Gałęzie (Branches)

- Git pozwala na szybkie, lekkie tworzenie gałęzi kodu
- Proste, szybkie przełączanie się pomiędzy gałęziami
- Git zachęca do rozgałęziania i scalania projektu nawet kilkukrotnie w ciągu dnia

Przechowywanie zmian

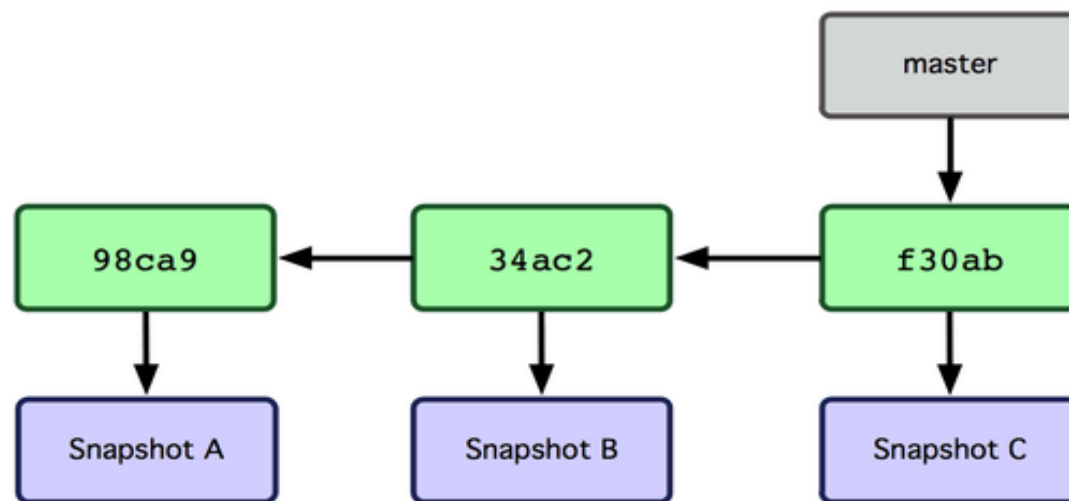
Commit zawiera:

- Migawkę zawartości
- Metadane
- 0 lub więcej wskaźników na commity rodziców



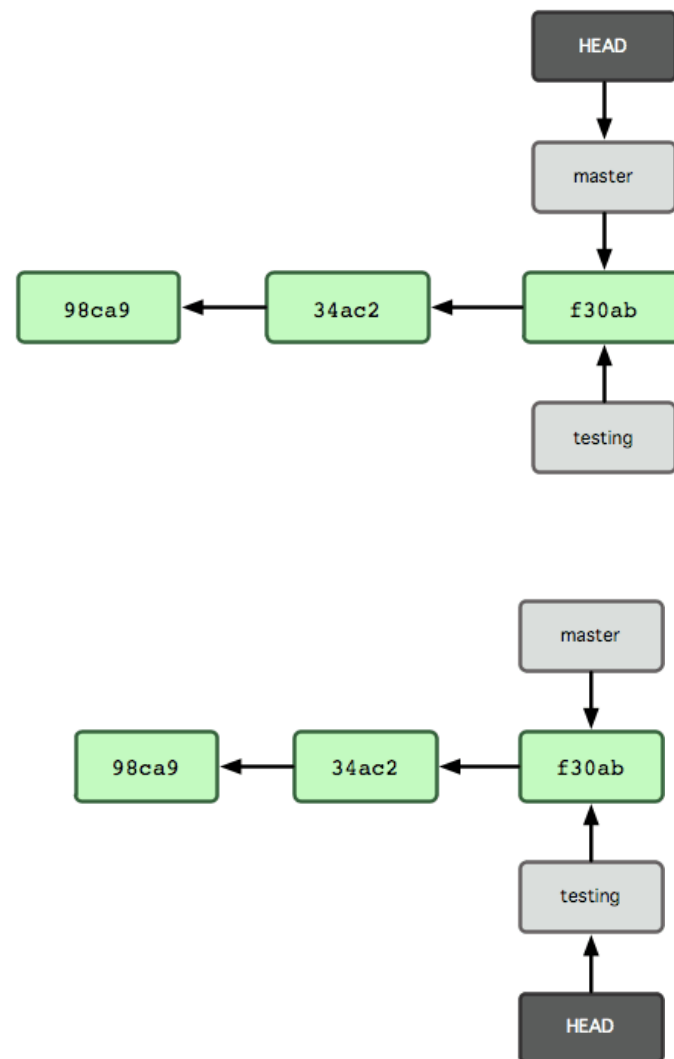
Gałąź

- Gałąź w gicie to przesuwalny wskaźnik na któryś z zestawów commitów (zmian)
- Domyślna gałąź to „master”
- Wskaźnik **HEAD** wskazuje na lokalną gałąź w której aktualnie się znajdujemy



Tworzenie nowej gałęzi

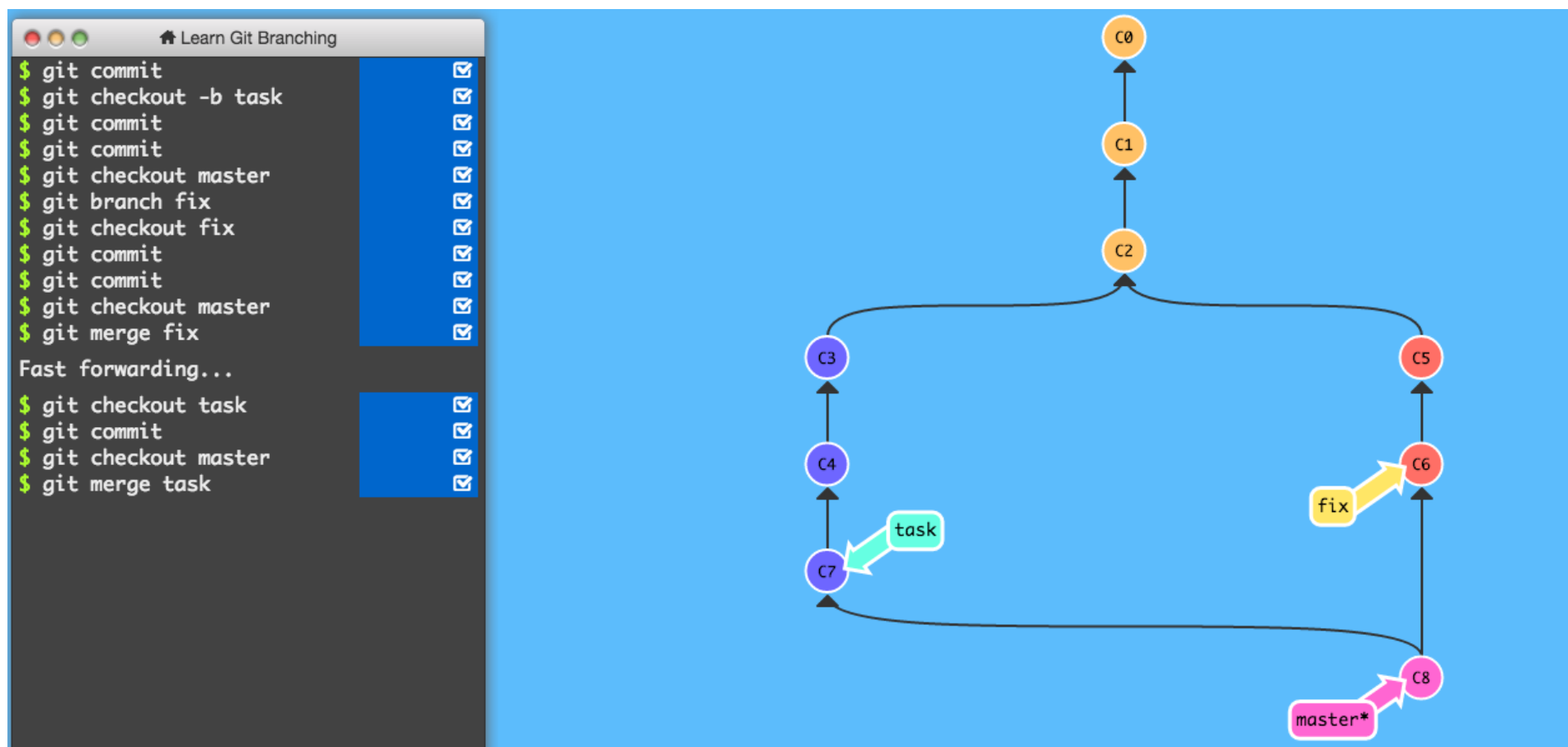
- Tworzenie nowej gałęzi:
 - `git branch [nazwa]`
- Przełączenie się na inną gałąź
 - `git checkout [nazwa]`
- Połączenie powyższych:
 - `git checkout -b [nazwa]`



Przykład

1. Praca nad projektem
2. Utworzenie gałęzi *task* dla zadania
3. Wykonanie pracy w gałęzi *task*
4. Przerwanie pracy inny problem z wyższym priorytetem
(musi być rozwiązany jak najszybciej)
5. Zmiana gałęzi na produkcyjną
6. Utworzenie gałęzi *fix*
7. Praca w gałęzi *fix*
8. Scalenie efektów pracy do gałęzi produkcyjnej (git merge)
9. Przełączenie się na gałąź *task* i dokończenie pracy
10. Scalenie *task* z gałęzią produkcyjną

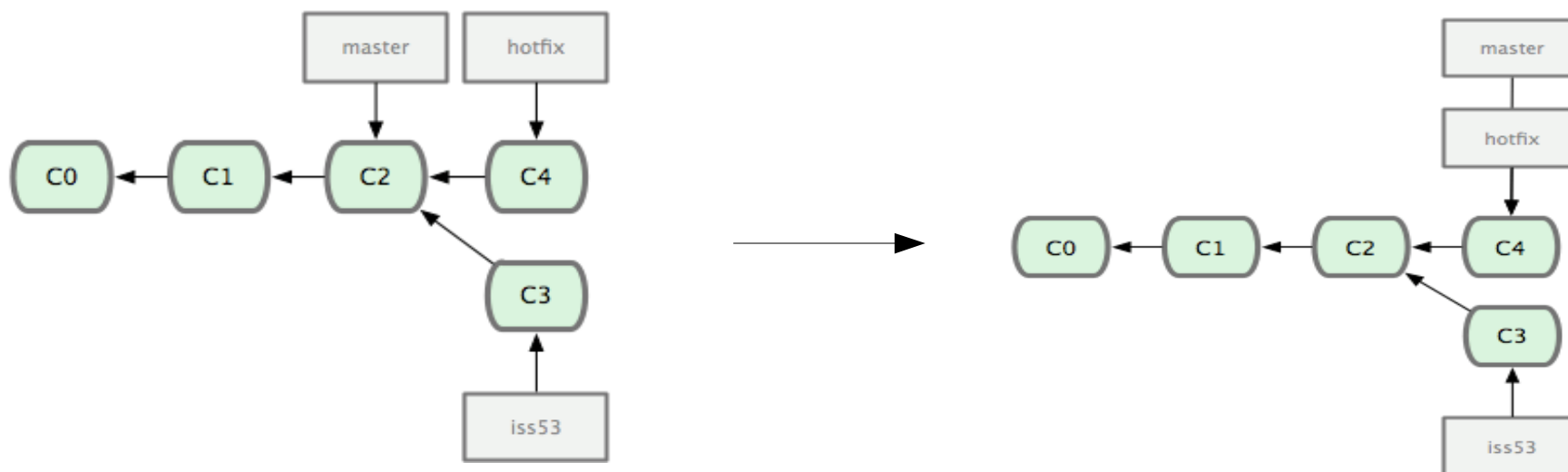
Przykład



<http://learngitbranching.js.org/>

Git merge: fast-forwarding

- Komenda służąca do scalania gałęzi:
 - Git merge [gałąź która scalamy do tej w której znajdujemy się obecnie]

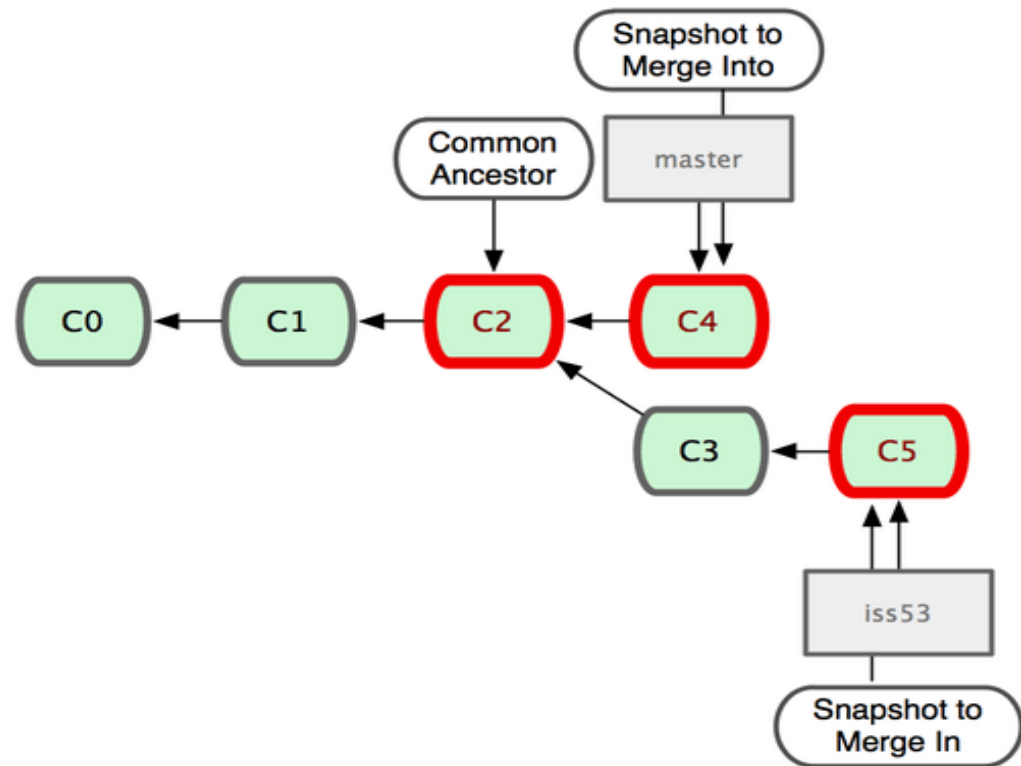


zestaw zmian wskazywany przez scalaną gałąź był bezpośrednim rodzicem aktualnego zestawu zmian,
Git przesuwa wskaźnik do przodu.

Git merge: three-way merge

Git tworzy nową migawkę, która jest wynikiem wspomnianego scalenia trójstronnego i automatycznie tworzy nowy zestaw zmian, wskazujący na ową migawkę. (merge commit)

Zmiana ta posiada więcej niż jednego rodzica.



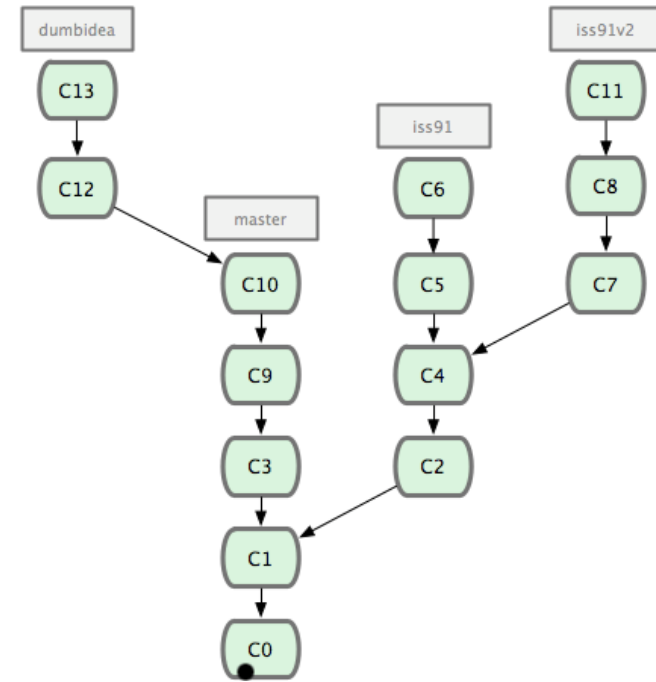
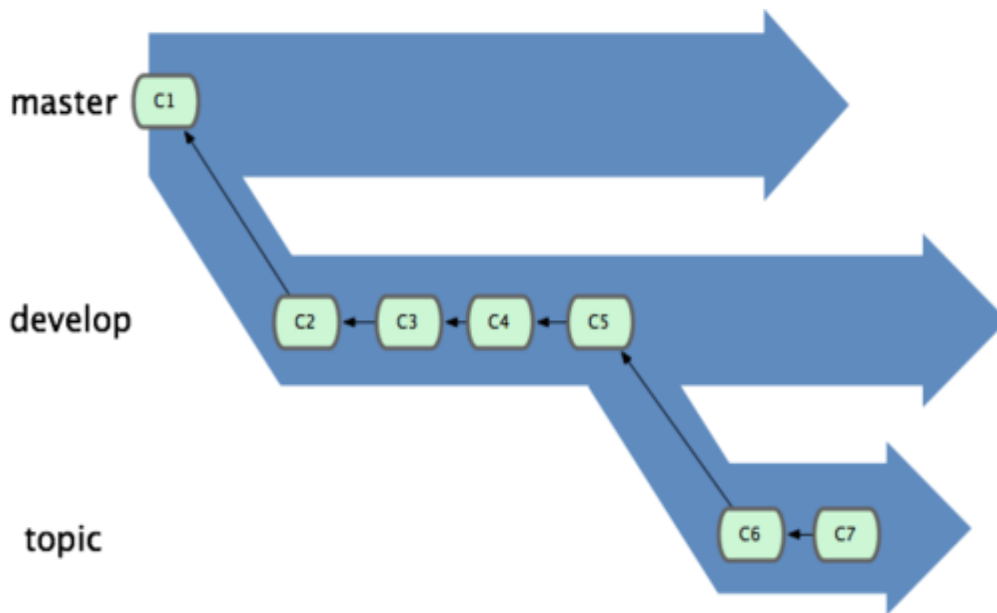
Co zrobić gdy pojawią się konflikty?

- Git wstrzymuje cały proces działania do czasu rozwiązania konfliktu
- Git status unmerged

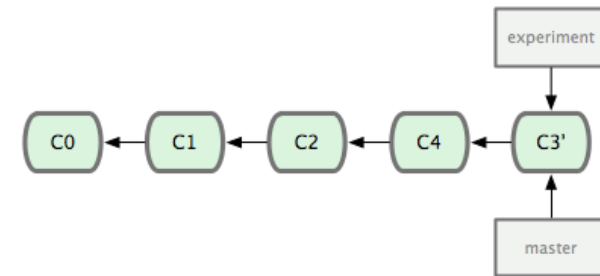
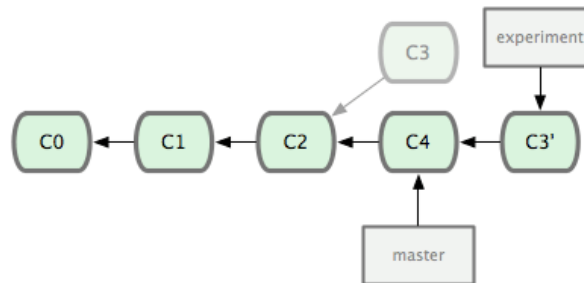
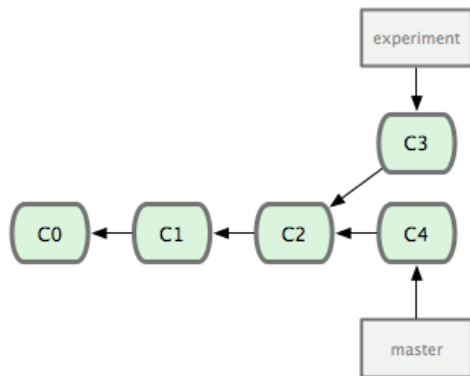
```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

- Ręcznie rozwiązujemy konflikt i usuwamy znaczniki
- Git add plik
- git mergetool

Sposoby pracy z gałęziami



Git rebase



\$ git checkout
experiment
\$ git rebase master

Git checkout master
Git merge

Polecenie to działa przesuwając się do ostatniego wspólnego przodka obu gałęzi (tej w której się znajdujesz oraz tej do której robisz zmianę bazy), pobierając różnice opisujące kolejne zmiany (ang. diffs) wprowadzane przez kolejne rewizje w gałęzi w której się znajdujesz, zapisując je w tymczasowych plikach, następnie resetuje bieżącą gałąź do tej samej rewizji do której wykonujesz operację zmiany bazy, po czym aplikuje po kolei zapisane zmiany.