Git - podstawy

Błażej Kowalczyk

Koło Naukowe Robotyków KoNaR

7 listopada 2014

Po co nam kontrola wersji?

- Cofanie zmian
- Kopia zapasowa
- Współpraca

Centralne systemy kontroli wersji (SVN)

- Konieczność uruchamiania usługi
- Obciążenie serwera
- Dostępność serwera
- Powolne działanie wielu operacji
- Tylko jedna kopia zapasowa uwzględniająca historię i gałęzie
- Każda zatwierdzona zmiana jest widoczna przez wszystkich

Rozproszone systemy kontroli wersji (Git)

- Może działać bez żadnego serwera
- Błyskawiczne operacje na historii i gałęziach
- Każdy posiada pełną kopię zapasową
- Rozdzielenie kontrolowania zmian i dzielenia się nimi

Instalacja

- Linux

```
yum install git
apt-get install git
cokolwiek install git
```

- Windows - MSysGit
  `http://msysgit.github.io`

Podstawowa konfiguracja

- Dane użytkownika

```
git config --global user.name "Jan Kowalski"
git config --global user.email jan.kowalski@gmail.com
```

- Edytor

```
git config --global core.editor notepad
```

Klucze SSH

Generujemy parę kluczy poleceniem ssh-keygen

```
[gt@localhost ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/gt/.ssh/id_rsa):
Created directory '/home/gt/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gt/.ssh/id_rsa.
Your public key has been saved in /home/gt/.ssh/id_rsa.pub.
The key fingerprint is:
4e:32:99:d2:cb:97:1b:32:ff:28:0b:48:05:1e:df:56 gt@localhost
[gittest@localhost ~]$
```

Dodajemy wpis do konfiguracji SSH (opcjonalne)

```
echo StrictHostKeyChecking no >> ~/.ssh/config
```

Lokalne repozytorium

Lokalne repozytorium git zapewnia nam niemal wszystkie możliwości kontroli wersji (oprócz synchronizacji pracy z innymi programistami oraz, co oczywiste, kopii zapasowej). Wszystko co potrzebne dla działania takiego repozytorium znajduje się w katalogu naszego projektu. Takie repozytorium jest świetnym rozwiązaniem dla jednoosobowych projektów.

Tworzymy nowe repozytorium w katalogu ‘repo‘

```
mkdir repo
cd repo
git init
```

Dodajemy pierwszą zmianę do repozytorium

```
echo Tekst >> plik
git add plik
git commit -m "pierwszy commit"
```

Ignorowanie plików

W większości przypadków przydatne może być dodanie do repozytorium pliku .gitignore. Określa on pliki, które mają nie być automatycznie dodawane do repozytorium, np. pliki tymczasowe, wyniki kompilacji, niektóre pliki środowiska programistycznego. Linie zaczynajace się od # to komentarze. Linie zaczynajace się od ! to wyjątki do poprzednich reguł

.gitignore

```
# żadnych plików .a
*.a
# ale uwzględniaj lib.a
!lib.a
# ignoruj plik TODO, ale nie podkatalog /TODO
/TODO
# ignoruj wszystkie pliki znajdujące się w katalogu build/
build/
```

Serwer gita

Nie istnieje de facto coś takiego jak serwer gita. Zamiast tego git zazwyczaj wykorzystuje transfer plików przez protokół SSH. Oznacza to, że możemy uruchomić zdalne repozytorium gita na każdym serwerze na którym posiadamy konto z dostępem SSH, bez konieczności instalowania bądź uruchamiania żadnych dodatkowych usług.
Inną opcją jest skorzystanie z istniejących w sieci serwisów oferujących hosting gita, np:

- `http://bitbucket.org` - nieograniczona ilość prywatnych repozytoriów. W wersji darmowej maks. 5 użytkowników na repo. Wersja unlimited po dodaniu adresu @pwr.wroc.pl
- `http://gitlab.com` - nieograniczona ilość prywatnych repozytoriów, z nieograniczoną liczbą użytkowników. Rozbudowany system zarządzania uprawnieniami.

Większość takich serwisów poza samym repozytorium oferują wbudowaną wiki i issue tracker dla naszych projektów.

Serwer z dostępem SSH

1. Wysyłamy swój klucz publiczny

```
scp .ssh/id_rsa.pub user@server.com:klucz.pub
```

2. Dodajemy klucz (na serwerze)

```
[user@localhost ~]$ ssh user@server.com
[user@server ~]$ cat klucz.pub >> .ssh/authorized_keys
```

```
[user@server ~]$ chmod 700 .ssh
[user@server ~]$ chmod 600 .ssh/*
```

3. Inicjujemy repozytorium 'bare'

```
git init --bare repo.git
```

Repozytorium 'bare' to takie, które nie posiada katalogu roboczego - służy tylko jako repozytorium zdalne. Zwyczaj każe nadawać im nazwy typu nazwa.git

Konfiguracja zdalnego repozytorium

Zarówno w przypadku własnoręcznie konfigurowanego serwer, jak i serwisów hostingowych, praca ze zdalnym repozytorium wygląda podobnie:

- Pobieramy do nowej lokalizacji

```
git clone user@server.com
```

- lub wysyłamy istniejące repozytorium lokalne

```
cd repo
git remote add origin user@server.com:repo.git
git push --set-upstream origin master
```

Schemat pracy

1. Wprowadzamy zmiany
2. Dodajemy pliki do śledzenia

```
git add (pliki)
```

3. Zatwierdzamy zmiany - jak najczęściej

```
git commit
```

4. Pobieramy zmiany z serwera - zawsze przed push!

```
git pull
```

5. Wysyłamy zmiany na serwer

```
git push
```

Przydatne komendy

```
git add .  #dodaj  wszytskie pliki do śledzenia
```

```
git commit -a  # zatwierdź wszystkie zmienione pliki
```

```
git commit --ammend  #zmień ostatni commit
```

```
git stash    # schowaj zmiiany od ostatniego commita
```

```
git stash apply # przywróć schowane zmiany
```

```
git reset --hard HEAD #usuń niezatwierdzone zmiany
```

```
git reset  HEAD~n #cofnij całe repozytorium o n zmian
```

Gałęzie

- Szybkie przełączanie między wersjami
- Łatwy powrót do działającej wersji (bez wyrzucania zmian)
- Możliwość równoległego rozwoju różnych funkcji, pomysłów itd.

Kiedy warto tworzyć nową gałąź?

- Zawsze

Tworzenie gałęzi

- Nowa gałąź

```
git branch nowa
git checkout nowa
```

lub

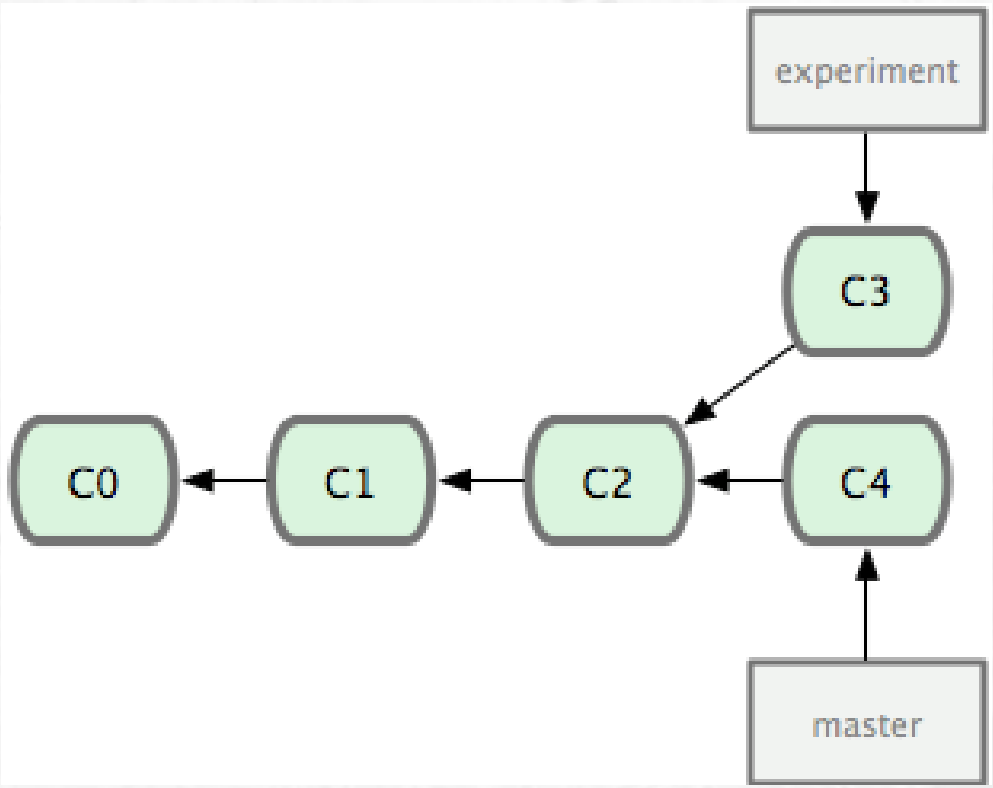
```
git checkout -b nowa
```

- Przenieś niezatwierdzone zmiany do nowej gałęzi

```
git stash
git stash branch nowa
```

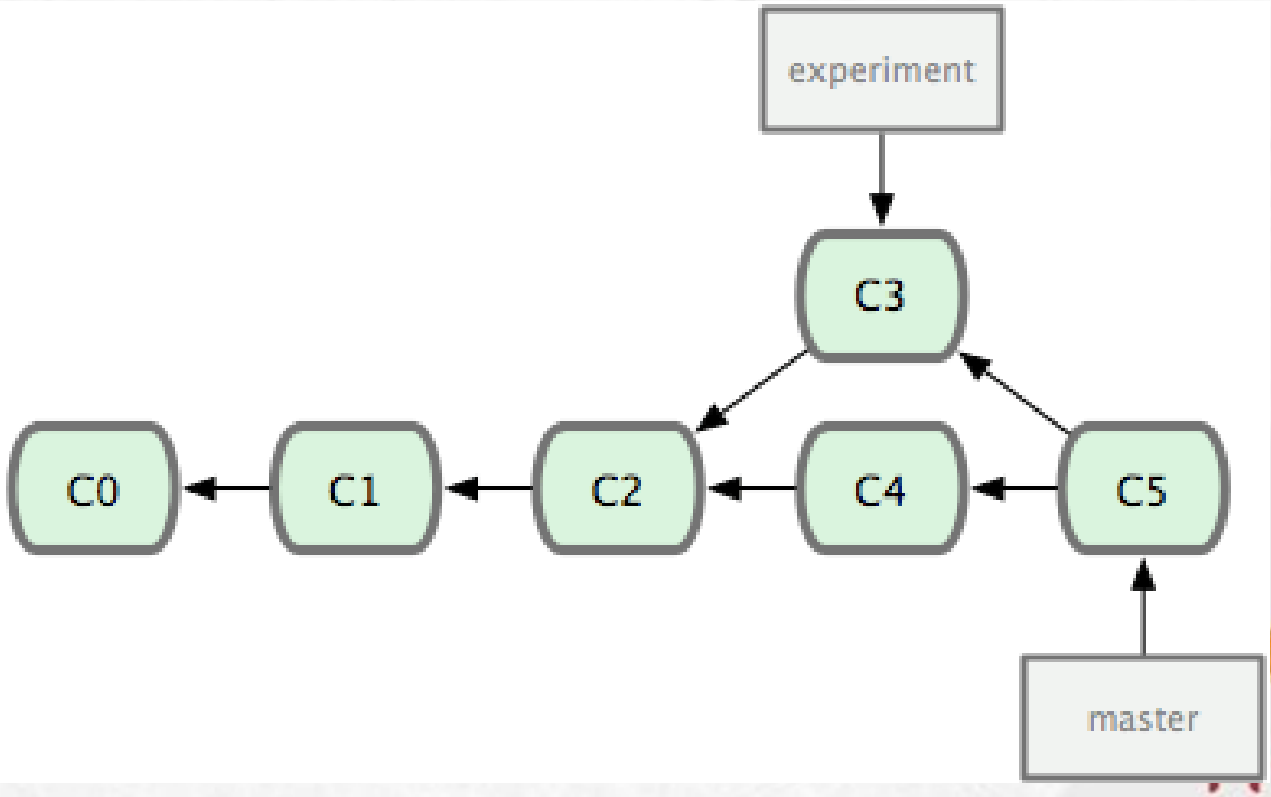
Merge

- Tworzy nowy punkt (commit) łączący gałęzie
- Nie narusza historii zmian
- ale czyni ją mniej czytelną



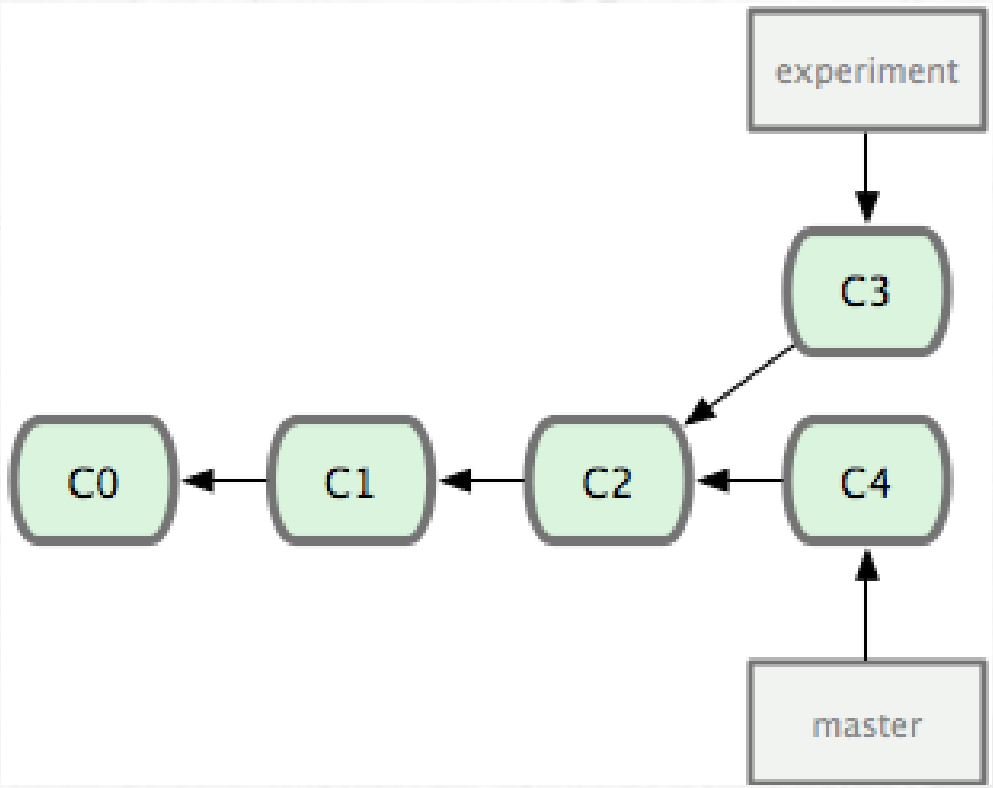
Merge

- Tworzy nowy punkt (commit) łączący gałęzie
- Nie narusza historii zmian
- ale czyni ją mniej czytelną



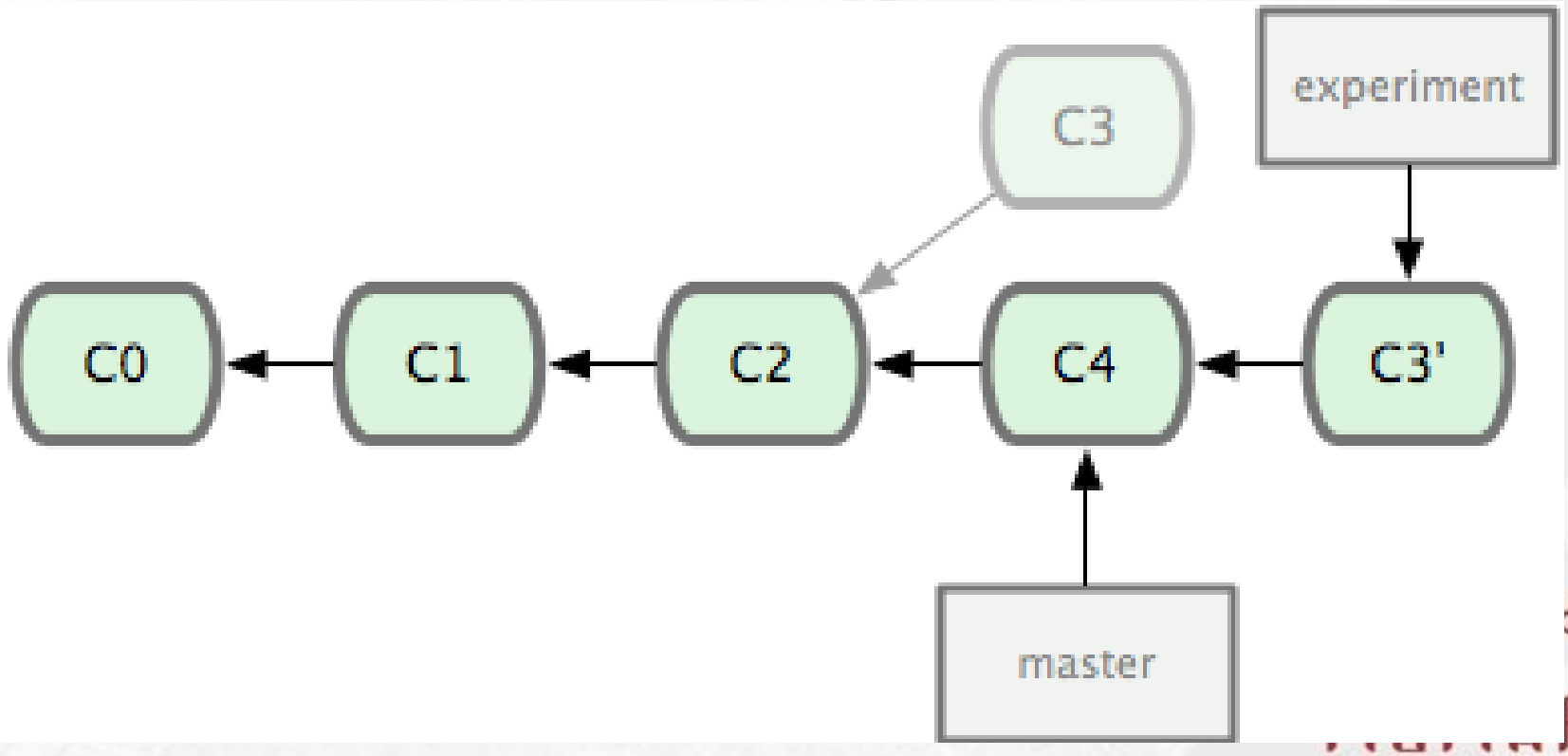
Rebase

- Przenosi zmiany i nanosi je od nowa
- Tworzy czytelną liniową historię zmian
- Nie można stosować do gałęzi, które były udostępnione



Rebase

- Przenosi zmiany i nanosi je od nowa
- Tworzy czytelną liniową historię zmian
- Nie można stosować do gałęzi, które były udostępnione



Rebase

- Przenosi zmiany i nanosi je od nowa
- Tworzy czytelną liniową historię zmian
- Nie można stosować do gałęzi, które były udostępnione