Version Control System

Spis treści

Czym jest system kontroli wersji?

Przygodę z **Git**em zacznijmy od wyjaśnienia, czym jest **VCS** (*Version Control System*). Zaznaczam, że chwilowo nie rozmawiamy o **Git**, tylko ogólnie o *Systemach Kontroli Wersji*.

Po co to?

Po co takie systemy w ogóle się stosuje? System taki pozwala zarządzać zbiorem plików i daje możliwość dostępu do poprzednich wersji zarządzanych plików. Inaczej mówiąc, system taki pozwala uchwycić zawartość plików w danym punkcie w czasie i pamięta później zawartość tych plików dla wspomnianych punktów w czasie. Oznacza to, że w dowolnym momencie możemy się cofnąć do każdego zapamiętanego punktu w czasie i zobaczyć jaką zawartość miał wtedy plik.

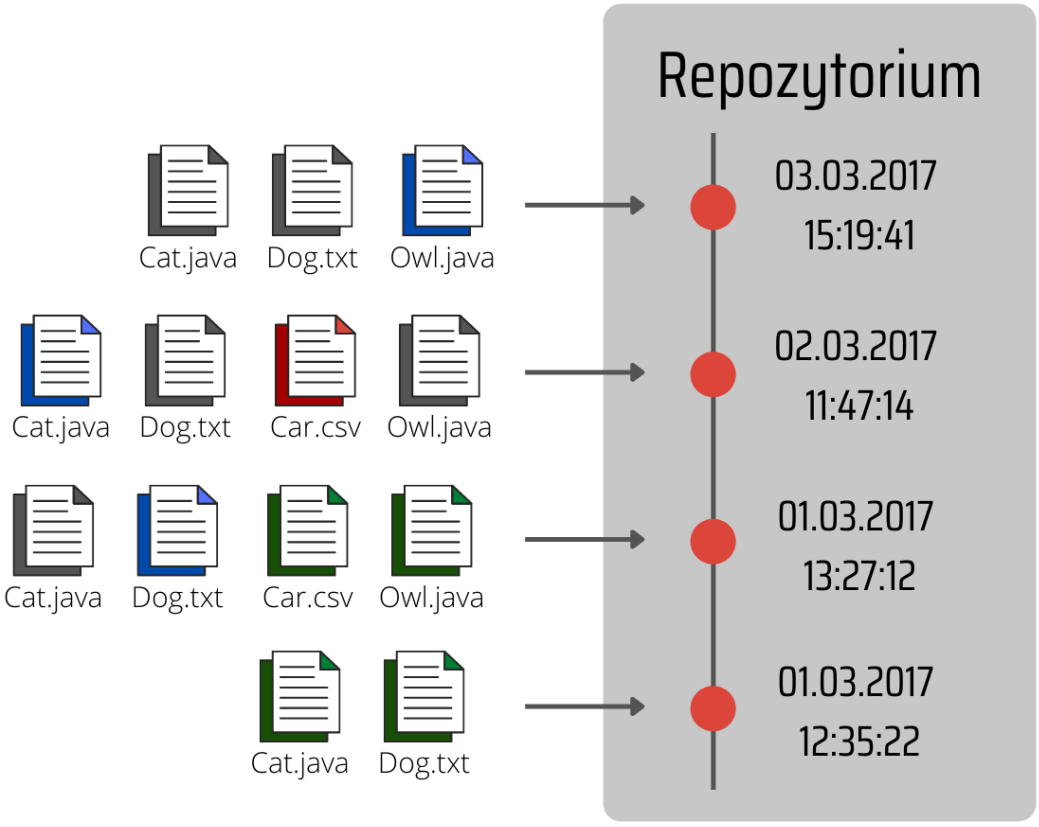
Systemy kontroli wersji są bardzo przydatnym rozwiązaniem, jeżeli chcemy śledzić historię zmian plików tekstowych. Patrząc na schemat, który jest poniżej, możesz zauważyć, że mając taki system, możemy zobaczyć, jaka była zawartość każdego z plików w danym punkcie czasowym. Możemy sprawdzić jakie zmiany zostały dokonane w każdym punkcie czasowym. Możemy również przywrócić konkretne wersje plików z danego punktu czasowego, czyli przywrócić ich zawartość w danym momencie w czasie. System taki zapisuje jednocześnie, kto dokonał zmian, więc ustalenie kto popsuł daną funkcjonalność, też jest ułatwione. 😈

Gdybyśmy pracowali nad jakimś projektem programistycznym w praktyce, w zespole złożonym z kilku osób, bardzo szybko pojawiłyby się dwa problemy:

- Każdy członek zespołu pracuje nad jakimś wycinkiem aplikacji. W jakiś sposób wypracowany kod trzeba pomiędzy członkami zespołu współdzielić. Najprostszym i jednocześnie trudnym do ogarnięcia sposobem jest wysyłanie wprowadzonych zmian mailem w postaci plików `feature-1.zip`. Oczywiste jest, że bardzo szybko doprowadziłoby to do problemów takich jak, np:
  - który z wysyłanych plików jest najnowszy,
  - jak połączyć ze sobą zmiany od kilku osób,
  - kto jest autorem danej zmiany,
  - w którym pliku `.zip` szukać zmian dotyczących konkretnej funkcjonalności.
  - i wiele, wiele innych.

- Chcemy dopytać autora danej linijki co miał na myśli, albo zwrócić mu uwagę, że popełnił błąd. Prościej jest móc sprawdzić to sobie samemu w jakimś jednym centralnym rejestrze niż biegać od biurka do biurka (albo przy pracy zdalnej dzwonić do każdego po kolei) i pytać: *Hej, czy to ty?*.

Miejsce, w którym przetrzymywane są te pliki, razem z zapisem wspomnianych punktów w czasie nazywane jest **repository** (*repozytorium*). Gdybyśmy mieli wyobrazić sobie w sposób abstrakcyjny, jak to wygląda, to możemy posłużyć się poniższą grafiką.



Obraz 1. VSC Schemat działania

Na powyższym przykładzie, możesz zauważyć, że dla kolejnych punktów w czasie zmienia się zawartość zbioru plików. Czerwone kropki oznaczają kolejne punkty na osi czasu. Na zielono oznaczone są pliki, które w danym punkcie w czasie są dodawane. Na niebiesko oznaczone są pliki, które są modyfikowane. Natomiast kolorem czerwonym oznaczono pliki, które są usuwane. Kolorem szarym zaznaczono pliki, które nie ulegają zmianie. Na dalszych etapach, punkty w czasie będą reprezentowały konkretną zmianę w projekcie, która ma cechy takie jak m.in.: swój unikalny identyfikator, czas dokonania tej zmiany, autora oraz zakres zmian.

Zwróć uwagę, że nigdzie nie wspominam, że technologia ta może być używana tylko z Javą. Na tym polega rzecz - systemy kontroli wersji mogą być używane z dowolnymi plikami. Oznacza to, że możemy ich używać niezależnie od języka programowania. Możemy ich też używać, nawet jak nie programujemy, po prostu chcemy śledzić historię zmian plików.

Systemy kontroli wersji sprawdzają się najlepiej, gdy będziemy chcieli śledzić historię zmian plików tekstowych, czyli np. plików `.txt`, albo `.java`. Możemy stosować te narzędzia również do zapisywania treści plików innych niż tekstowe, np. `.png`, `.jpg`, albo nawet `.docx` lub `.pdf`. Niestety w tym przypadku narzędzie to nie będzie już aż tak przydatne, gdyż nie będziemy w stanie porównać ze sobą zawartości

tych plików. Przekonasz się o tym później, jak przejdziemy do przykładów.

Zanim przejdziemy do omówienia **Git**a samego w sobie, wyjaśnijmy sobie jeszcze kilka kwestii.

Lokalne systemy kontroli wersji

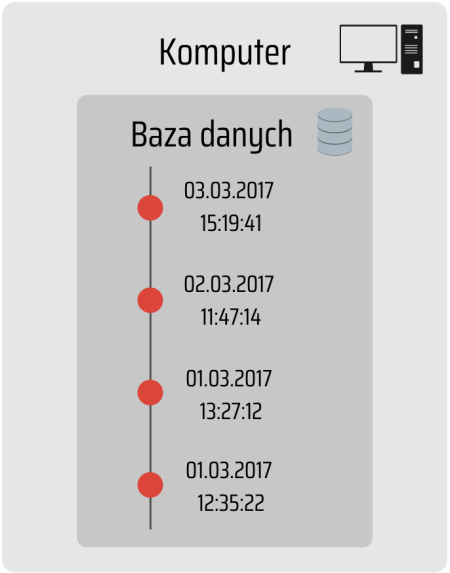
Zacznijmy sobie od wyjaśnienia, że możemy używać systemu kontroli wersji, który będzie tylko lokalny. Taki system kontroli wersji będzie ograniczony tylko do jednego komputera, więc jego przydatność w praktyce jest raczej mała. Warto natomiast wiedzieć, że tak naprawdę możemy stworzyć swój własny system kontroli wersji. Jeżeli pracujesz nad jakimś projektem, który wymaga tworzenia plików na komputerze, to możesz takie pliki później umieszczać w osobnych katalogach i nazywać te katalogi `v1`, `v2`, `v3-final`, `v3-final2`, `v3-ostateczny-final` i tak dalej. Możesz również pakować takie katalogi do plików `.zip` i też będzie to jakiś lokalny system kontroli wersji.

Wspomniany sposób zarządzania plikami jest dość popularny, gdy pracujemy nad jakimś projektem samodzielnie. Dużo osób stosuje takie podejście, bo jest ono proste. Często z racji tego, że projekt jest mały, wydaje nam się, że nie ma potrzeby stosować innych podejść, albo zwyczajnie takich podejść nie znamy. Podejście to ma natomiast dużo wad, przykładowo:

- łatwo pogubić się, która wersja pliku jest faktycznie najnowszą,
- możemy zacząć edytować inną wersję niż najnowsza, co wynika z poprzedniego podpunktu,
- można przez przypadek usunąć najnowszą wersję pliku.

Lokalne systemy kontroli wersji mogą być również oparte o bazę danych. Możemy wtedy zapisywać kolejne wersje plików razem z datą ich modyfikacji w bazie danych. Podejście takie jest o tyle ryzykowne, że cała historia projektu jest wtedy zapisana w jednym miejscu. Wprowadzamy wtedy tzw. **SPoF** (***S**ingle **P**oint **o**f **F**ailure*), czyli jeżeli zgubimy taki komputer, albo pies nam go zje, to straciliśmy cały projekt.

Jednym z bardziej popularnych systemów tego typu jest RCS (*Revision Control System*).



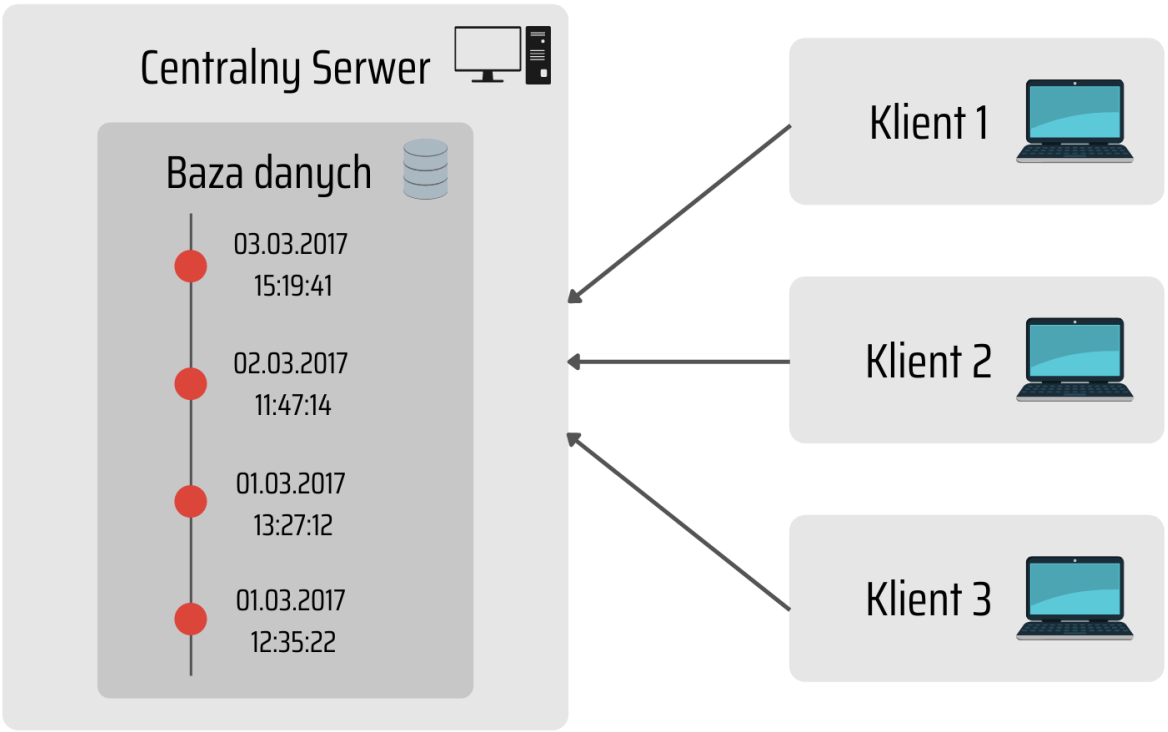
Obraz 2. RCS Schemat działania

Scentralizowane systemy kontroli wersji

Kolejny rodzaj systemów kontroli wersji to **CVCS** (*Centralized Version Control System - Scentralizowany System Kontroli Wersji*). Wprowadzają one możliwość dostępu do projektu z więcej niż jednego komputera. W takich rozwiązaniach mamy jeden centralny serwer (komputer, do którego możemy połączyć się przez sieć), który przechowuje informacje o zmianach w plikach. Do takiego serwera mogą łączyć się inne komputery (klienci) i pobrać informacje o zmianach.

Zaletą takiego podejścia jest możliwość pracy nad jednym projektem w kilka osób. Każdy członek zespołu może się wtedy łączyć do jednego centralnego miejsca przechowującego informacje o zmianach i dzięki temu, można wtedy pracować nad jednym projektem wspólnie. Podejście takie daje nam już możliwość weryfikacji, kto wprowadził dane zmiany. Co więcej, możemy również zastrzec dostęp do takiego projektu i chronić się przed dostępem osób nieuprawnionych. Jeżeli natomiast chodzi o wady, to nadal mamy jedno miejsce **SPoF**, którego awaria lub zniszczenie powoduje, że tracimy informacje o zmianach w projekcie. W wyniku zniszczenia takiego centralnego serwera możemy też potencjalnie utracić cały projekt. Jeżeli taki centralny serwer będzie miał awarię i nie będziemy mogli się z nim połączyć, to ani nie możemy pobrać informacji o zmianach, ani zapisać informacji o wprowadzanych zmianach.

Jednym z bardziej popularnych systemów tego typu są CVS (*Concurrent Versions System*) lub Subversion, inaczej znany jako **SVN**.



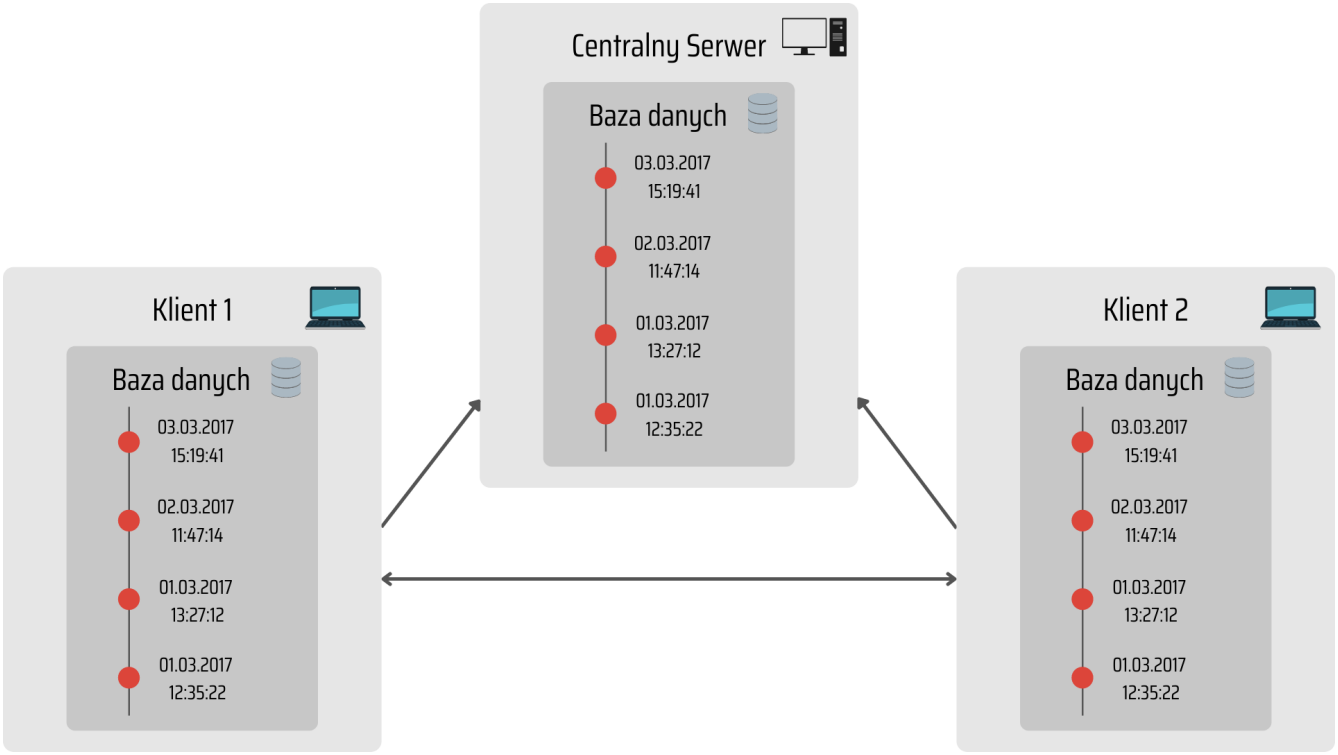
Obraz 3. CSVS Schemat działania

Rozproszone systemy kontroli wersji

Kolejnym rodzajem systemów kontroli wersji jest **DVCS** (*Distributed Version Control System - Rozproszony System Kontroli Wersji*). Ten rodzaj systemów kontroli wersji jest najbardziej złożony z wymienionych, ale pomaga obejść podstawowy problem poprzednich systemów kontroli wersji - **SPoF**. W przypadku systemów rozproszonych każdy klient ma wykonaną kopię całego repozytorium i

przechowuje ją u siebie. Czyli klienci nie pobierają tylko najnowszych zmian, tylko replikują cały stan historii zmian projektu i zapisują ją po swojej stronie. Podejście takie oznacza, że klienci mogą umówić się, który komputer jest głównym serwerem. Jeżeli taki główny serwer ulegnie awarii, nie zostanie utracona historia projektu, bo każdy z klientów ma ją odtworzoną po swojej stronie. W przypadku awarii serwera każdy z klientów może służyć za serwer, bo ma zapisaną u siebie cały rejestr zmian w projekcie.

Jednym z bardziej popularnych systemów tego typu są Mercurial albo właśnie Git, o którym będziemy rozmawiać w dalszych materiałach.



Obraz 4. DVCS Schemat działania