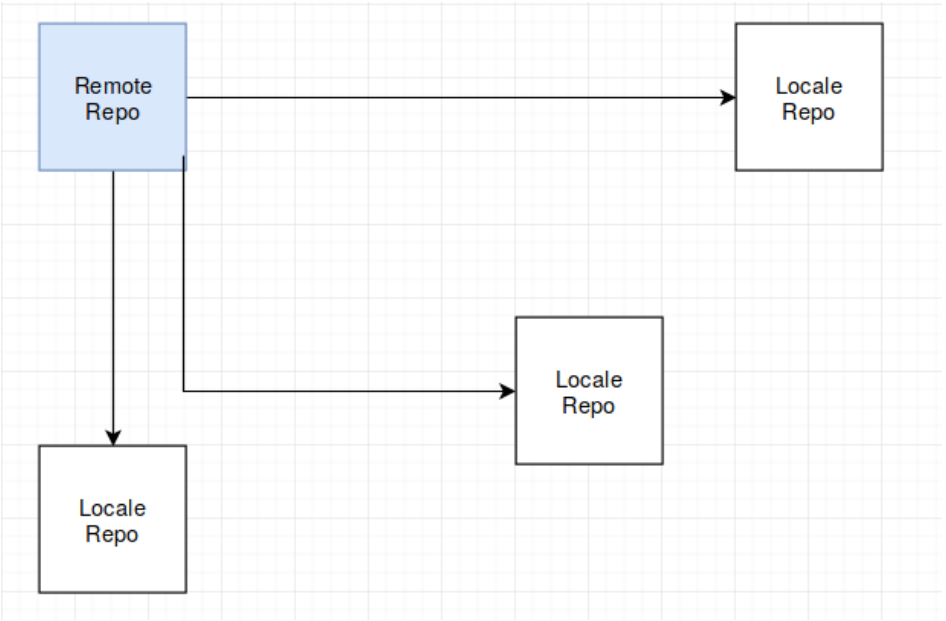
GIT

Rozproszony system kontroli wersji

Ogólnie



- GIT nie posiada scentralizowanego repozytorium, każdy użytkownik przechowuje jego dokładną kopię wraz z historią na swoim komputerze
- W każdym repozytorium znajdują się gałęzie (branch), a każda gałąź składa się z serii commitów
- Istnieje umowna główna gałąź zwana master

git clone

Komenda służy do kopiowania repozytorium

Komendy

git status

Komenda, która przedstawia stan naszego repozytorium w kontekście aktualnego brancha, od ostatniego commita. Używając tej komendy możemy zobaczyć, które nowe pliki dodaliśmy, jakie zmodyfikowaliśmy lub usuneliśmy

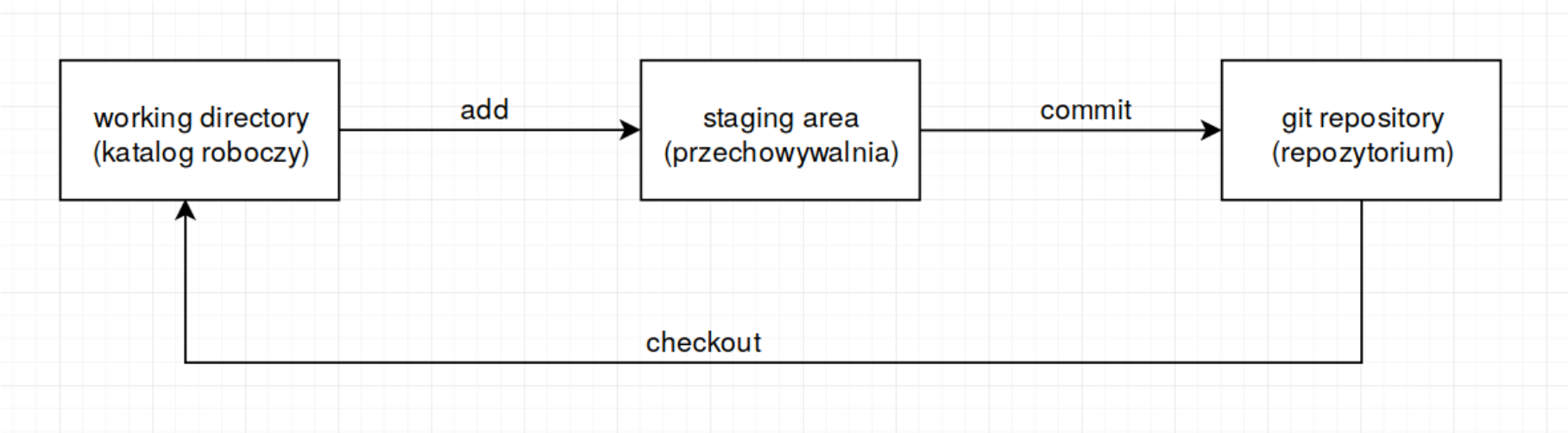
git log

Dzięki tej komendzie, możemy zobaczyć historię commitów w ramach wybranego brancha

git branch

Wyświetla nam listę branchy w lokalnym repozytorium oraz wskazuje, na którym aktualnie się znajdujemy

Przepływ



- Katalog roboczy – obraz wersji projektu, na której aktualnie pracujemy
- Przechowywalnia – plik zawierający informację o plikach, których będzie dotyczył najbliższy commit
- Repozytorium – katalog zawierający metadane oraz bazę danych naszego projektu. Stanowi on jeden z najważniejszych elementów naszego repozytorium

Komendy

git add

Dodaje do staging area pojedynczy plik. Aby usunąć plik ze staging area, używamy komendy będącą odwrotnością add – *git reset*

git add .

Dodaje do staging area wszystkie pliki znajdujące się w katalogu roboczym, które zmieniliśmy lub dodaliśmy. Aby nie dodawać wszystkiego, możemy wykorzystać plik *.gitignore*

git commit -m „tytuł”

Komenda tworzy commita na podstawie plików znajdujących się w staging area

git commit -am „tytuł”

Komenda łączy dwie komendy *git add* i *git commit*. Dodajemy do commita tytuł oraz dodaje automatycznie do staging area wszystkie zmodyfikowane pliki (pomija nowo utworzone)

Komendy

git push

Wysyłanie naszego lokalnego repozytorium do zdalnego repozytorium

git checkout nazwaGalezi

Komenda pozwalająca przełączyć się na inny, dostępny branch

git checkout -b „nowaGalaz”

Tworzy nowy branch o podanej nazwie oraz automatycznie
się na niego przełącza. Nowy branch ma taką samą historię zmian co branch,
na którym go tworzymy

git checkout -

Komenda przydatna gdy głównie pracujemy między dwoma branchami.
Pozwala się nam przełączać między aktualnym, a wcześniejszym

Komendy – cofanie zmian

git checkout plik.txt

Komenda służy do cofnięcia zmian dla wskazanego pliku (nie brancha) do stanu z ostatniego commita na branchu, na którym się znajdujemy

git checkout <hash>

Znając identyfikator hash danego commita, możemy cofnąć się do niego przy pomocy powyższej komendy. Znajdziemy się w trybie przeglądania gdyż żadne zmiany w przywróconym commicie, nie będą mogły być wprowadzane do repozytorium. Aby temu zaradzić należy stworzyć nowego brancha git checkout -b „nowaGalaz”

git checkout <hash> plik.txt

Komenda umożliwia nam cofnięcie zmian dla danego pliku w ramach wybranego commita w ramach branchu, na którym się znajdujemy

git revert <hash>

Znając hash wybranego commita możemy przywrócić zmiany i zapisać je jako nowy commit. Operacja nie niszczy nam historii zmian – zachowujemy liniowość

Komendy – usuwanie

git clean

Usuwa pliki, które nie zostały dodane do staging area

git rm plik.txt

Komenda działa podobnie jak klasyczny *rm* przy czym tutaj dodatkowo są wprowadzane zmiany w staging area. Możemy usunąć plik fizycznie oraz z przechowywalni. Całość zatwierdzamy komendą *git commit*

Przypadek bilokacji

Tworzymy plik f1.txt

touch f1.txt

Dodajemy go do staging area

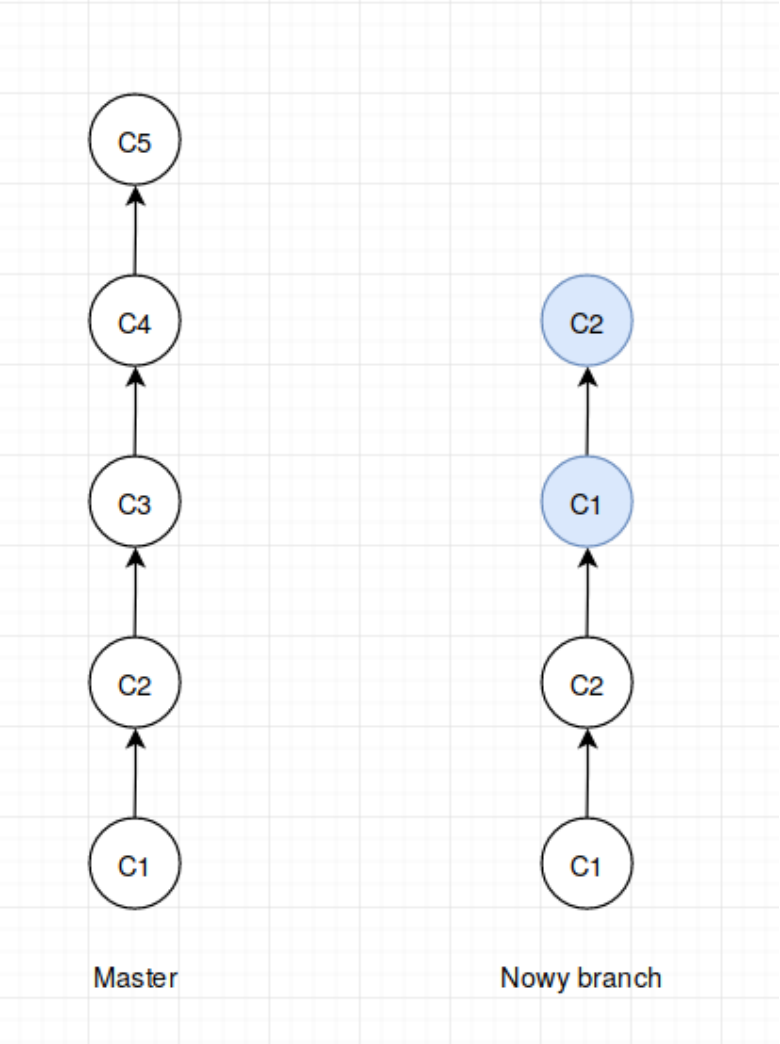
git add f1.txt

Dokonujemy zmian w pliku (np. wykorzystując edytor vim)

vim f1.txt

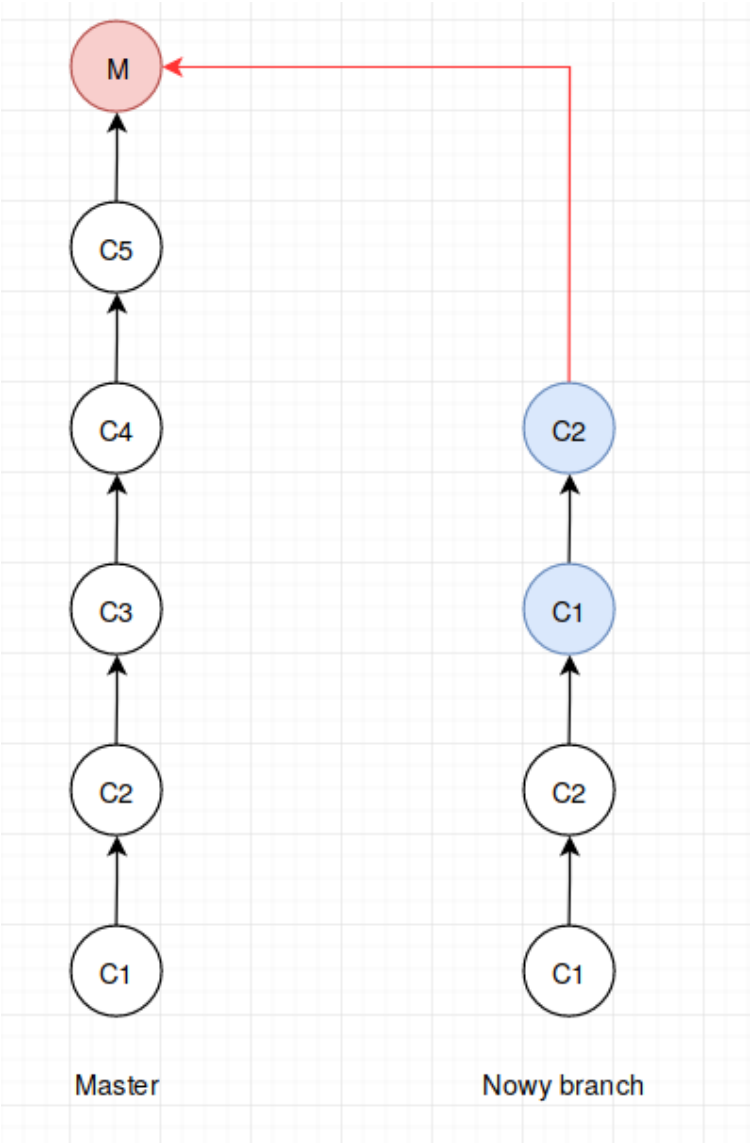
Po wykonaniu polecenia git status widzimy, że plik f1.txt znajduje się jednocześnie w katalogu roboczym oraz przechowalni – jest to poprawne zachowanie, gdyż w staging area znajduje się wersja pliku przed modyfikacją.
Jeżeli w tym momencie zacommitujemy zmiany, w repozytorium nie zapiszą się zmiany po edycji

Merge



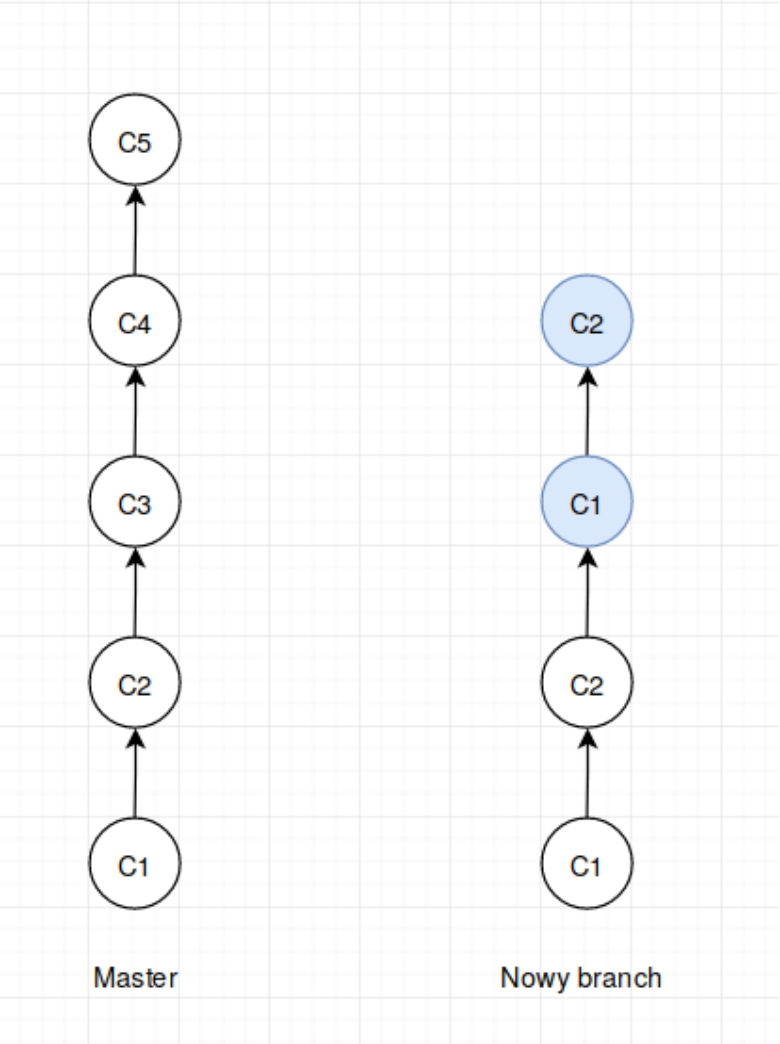
C<n> - kolejny commit

Merge



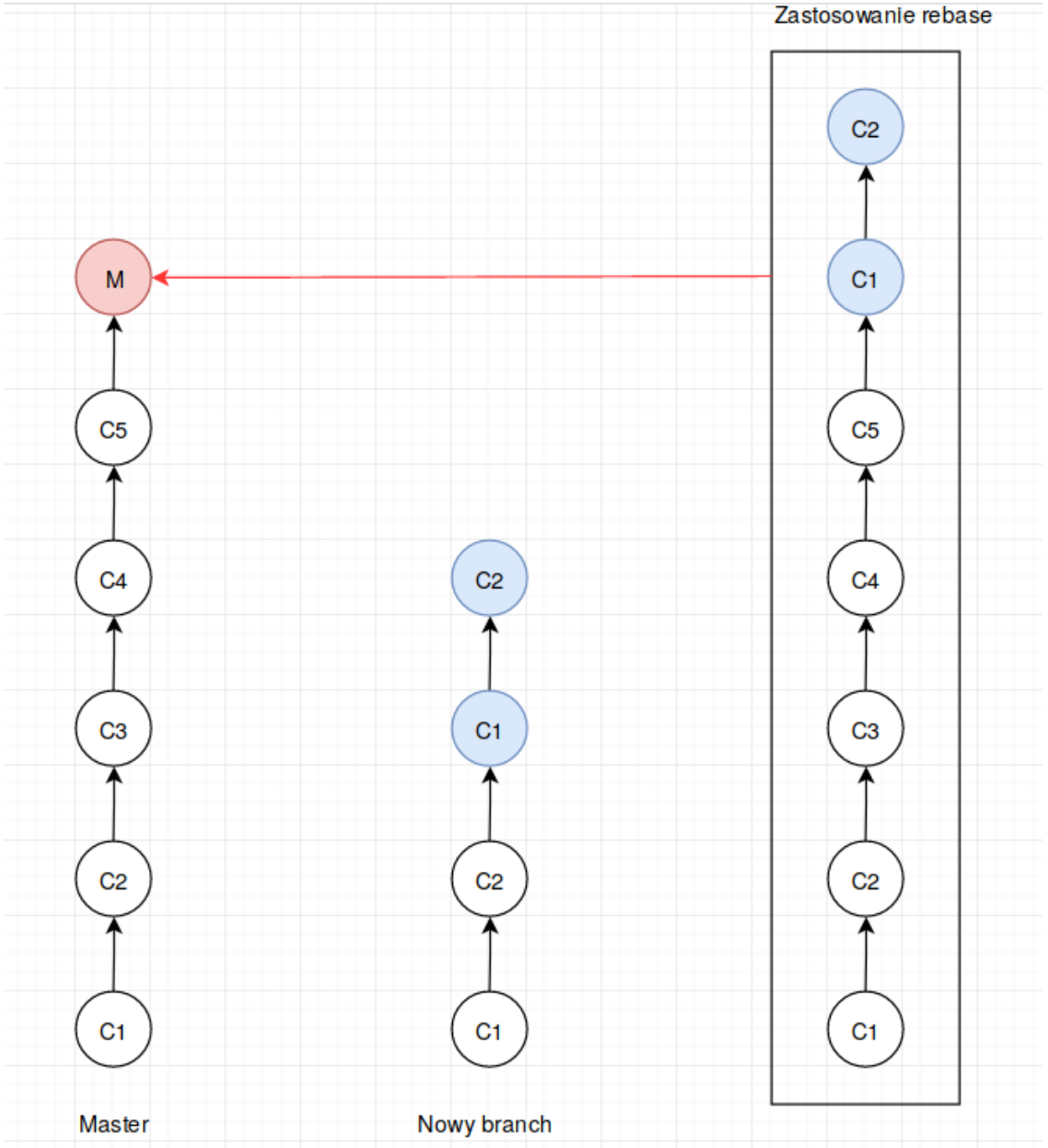
M - merge commit
C<n> - kolejny commit

Rebase



C<n> - kolejny commit

Rebase



M - merge commit
C<n> - kolejny commit

Merge vs Rebase

- Komendy *git merge* i *git rebase* służą do łączenia dwóch branchy w jeden
- Megre jest najprostszą metodą, tworząc *merge commit* , w którym scala commity z innego brancha i dołącza do nowego – problem z liniowością historii zmian
- Rebase pobiera wszystkie zmiany jakie zostały zacommitowane w innej gałęzi i dołącza do naszej. W ten sposób mamy wszystkie zmiany z obu gałęzi w jednym miejscu (zachowujemy liniowość zmian)