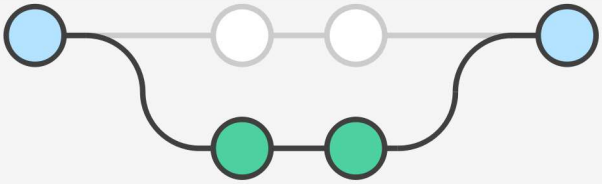


Porównanie merge i rebase

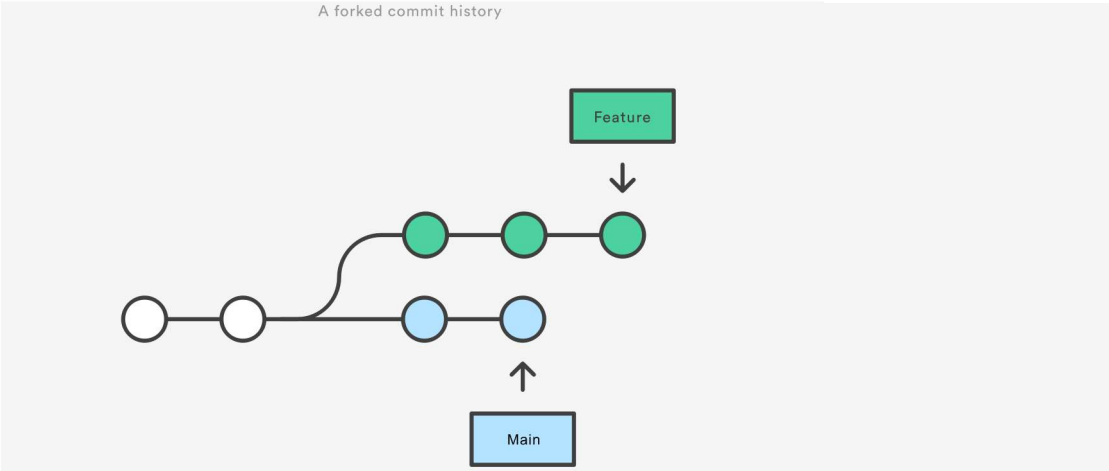
Przegląd koncepcji **Złota reguła zmiany bazy** **Przewodnik po przepływie pracy** **Podsumowanie**

Polecenie *git rebase* ma reputację magicznego rytuału Git, od którego osoby początkujące powinny trzymać się z daleka. Gdy jest jednak używane z zachowaniem ostrożności, znacznie ułatwia życie zespołowi programistycznemu. W tym artykule porównamy polecenie *git rebase* z powiązanym poleceniem *git merge* i wskażemy wszystkie potencjalne możliwości włączenia operacji zmiany bazy do typowego przepływu pracy Git.

Przegląd koncepcji

Pierwszą rzeczą, jaką trzeba wiedzieć na temat polecenia *git rebase*, jest fakt, że służy do rozwiązywania tego samego problemu, co polecenie *git merge*. Obydwa polecenia zaprojektowano tak, aby umożliwiały integrowanie zmian z jednej gałęzi z drugą gałęzią — po prostu robią to w bardzo odmienny sposób.

Zastanówmy się, co się stanie, gdy zaczniesz pracę nad nową funkcją w dedykowanej gałęzi, a następnie inny członek zespołu zaktualizuje gałąź *main* o nowe commity. W rezultacie powstanie podział historii — zjawisko znane każdemu, kto korzystał z systemu Git jako narzędzia do współpracy.



Przypuśćmy teraz, że nowe commity w gałęzi *main* są istotne dla funkcji, nad którą pracujesz. Aby włączyć nowe commity do swojej gałęzi *feature*, możesz użyć jednej z dwóch operacji: scalania lub zmiany bazy.

Operacja scalania

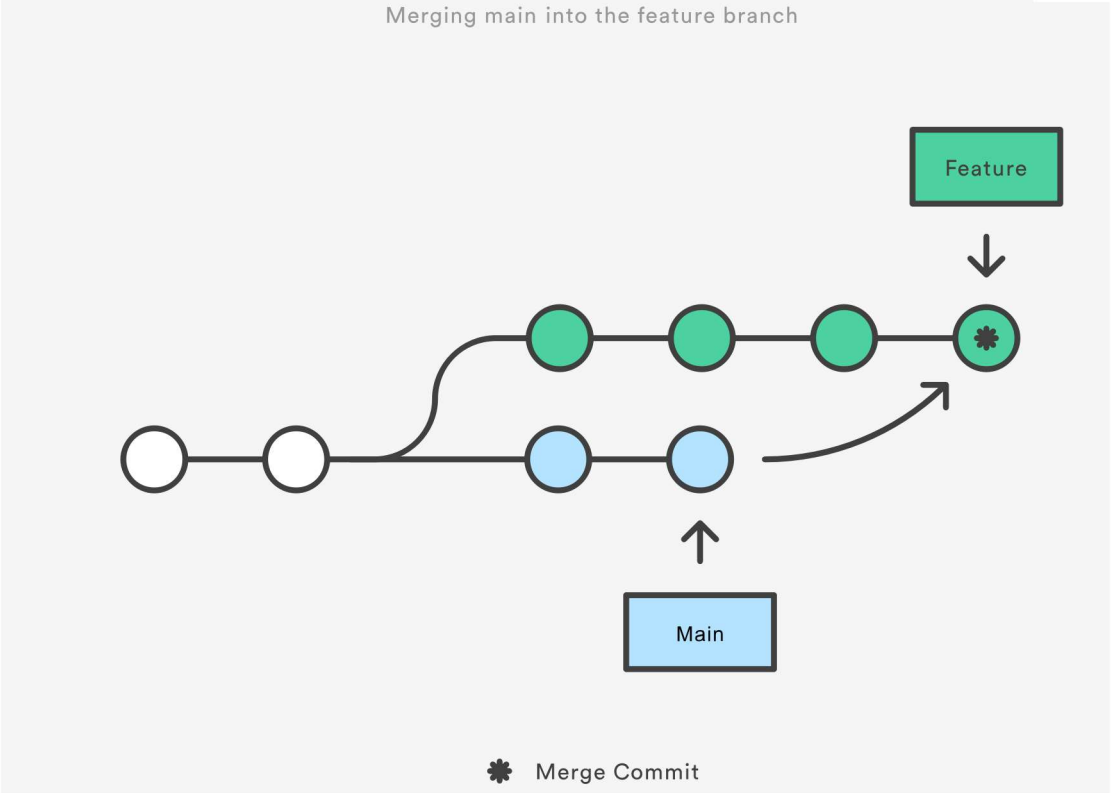
Najprostszym rozwiązaniem jest scalenie gałęzi *main* z gałęzią funkcji, wykonując polecenia podobne do następujących:

```
git checkout feature
git merge main
```

Można również zredukować je do postaci jednowierszowej:

```
git merge feature main
```

W efekcie w gałęzi *feature* powstanie „commit scalenia", który będzie łączył historie



obydwu gałęzi, dzięki czemu struktura gałęzi będzie wyglądała następująco:

Zaletą scalania jest niedestrukcyjny charakter operacji. Istniejące gałęzie w żaden sposób nie ulegają zmianie. Pozwala to uniknąć wszystkich potencjalnych (omówionych poniżej) zagrożeń, jakie wiążą się z operacją zmiany bazy.

Z drugiej strony oznacza to, że gałąź *feature* będzie powiększana o nieistotny commit scalenia za każdym razem, gdy będzie trzeba uwzględnić zmiany nadrzędne. Jeśli w gałęzi *main* są prowadzone bardzo intensywne prace, może to spowodować nieporządek

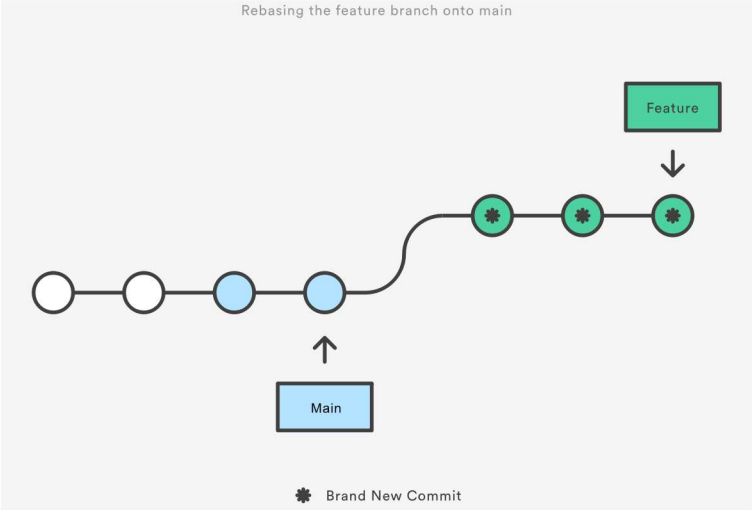
w historii gałęzi funkcji. Choć problem ten można zniwelować za pomocą zaawansowanych opcji polecenia *git log*, utrudni to innym programistom zrozumienie historii projektu.

Operacja zmiany bazy

Alternatywą dla scalania jest zmiana bazy gałęzi *feature* na gałąź *main* za pomocą następujących poleceń:

```
git checkout feature
git rebase main
```

Spowoduje to przesunięcie całej gałęzi *feature*, aby zaczynała się od końcówki gałęzi *main*, i pozwoli na skuteczne włączenie wszystkich nowych commitów do gałęzi *main*. Jednak zamiast tworzenia commita scalenia operacja zmiany bazy powoduje przepisanie historii projektu poprzez utworzenie zupełnie nowych commitów dla każdego commita w gałęzi pierwotnej.



Najważniejszą zaletą zmiany bazy jest uzyskanie znacznie bardziej przejrzystej historii projektu. Po pierwsze eliminuje ona tworzenie zbędnych commitów scalenia wymaganych w przypadku polecenia *git merge*. Po drugie, jak widać na powyższym diagramie, zmiana bazy pozwala również uzyskać idealnie liniową historię projektu. Można przejść od końcówki gałęzi *feature* do samego początku projektu bez żadnych podziałów. Ułatwia to poruszanie się po projekcie za pomocą poleceń, takich jak *git log*, *git bisect* i *gitk*.

Jednak taka nieskazitelna historia commitów niesie za sobą ryzyko dotyczące dwóch kwestii: bezpieczeństwa i możliwości śledzenia. Jeśli nie zastosujesz się do złotej reguły zmiany bazy, przepisanie historii projektu może być katastrofalne w skutkach dla przepływu pracy opartego na współpracy. Ponadto, choć nie jest to aż tak istotne, operacja zmiany bazy prowadzi do utraty kontekstu, jaki zapewniają commity scalenia. Nie da się sprawdzić, kiedy zmiany nadrzędne zostały włączone do gałęzi funkcji.

Interaktywna zmiana bazy

Interaktywna zmiana bazy umożliwia modyfikowanie commitów podczas przenoszenia ich do nowej gałęzi. Jest to rozwiązanie znacznie bardziej zaawansowane niż automatyczna zmiana bazy, ponieważ zapewnia pełną kontrolę nad historią commitów gałęzi. Zazwyczaj stosuje się je do oczyszczania nieuporządkowanej historii przed scaleniem gałęzi funkcji z gałęzią *main*.

Aby rozpocząć interaktywną zmianę bazy, należy przekazać opcję *i* do polecenia *git rebase*:

```
git checkout feature
git rebase -i main
```

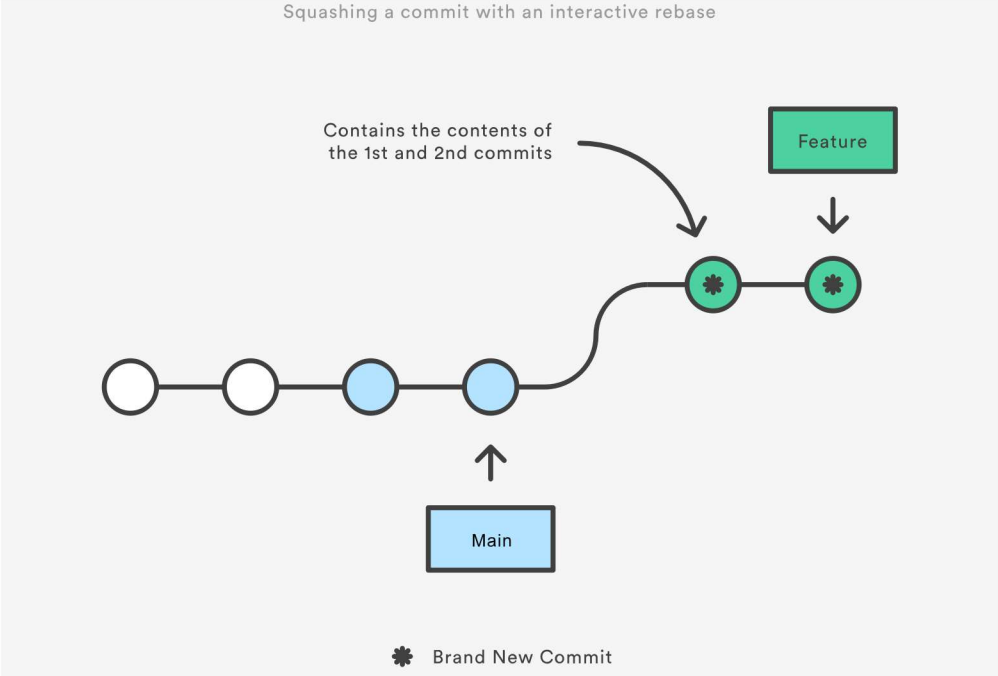
Spowoduje to otwarcie edytora tekstu, w którym zostanie wyświetlona lista wszystkich commitów do przeniesienia:

```
pick 33d5b7a Message for commit #1
pick 9480b3d Message for commit #2
pick 5c67e61 Message for commit #3
```

Ta lista określa dokładnie, jak będzie wyglądać gałąź po wykonaniu operacji zmiany bazy. Modyfikując polecenie *pick* i/lub zmieniając kolejność wpisów, można nadać historii gałęzi dowolny kształt. Przykładowo, jeśli drugi commit rozwiązuje niewielki problem w pierwszym commicie, można je połączyć, tworząc pojedynczy commit za pomocą polecenia *fixup*:

```
pick 33d5b7a Message for commit #1
fixup 9480b3d Message for commit #2
pick 5c67e61 Message for commit #3
```

Po zapisaniu i zamknięciu pliku Git wykona operację zmiany bazy według podanych instrukcji i otrzymamy historię projektu wyglądającą następująco:

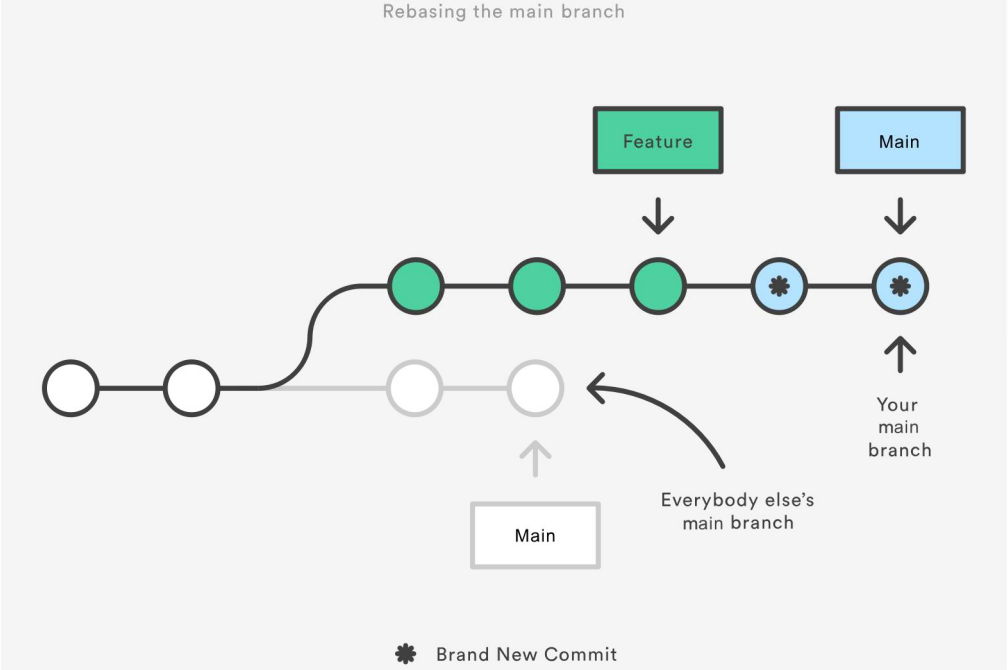


Wyeliminowanie mało istotnych commitów, takich jak ten, znacznie zwiększa przejrzystość historii funkcji. Takiego rezultatu nie da się uzyskać za pomocą polecenia *git merge*.

Złota reguła zmiany bazy

Gdy już zrozumiesz, na czym polega zmiana bazy, to najważniejsze będzie nauczenie się, kiedy nie należy jej wykonywać. Złota reguła dotycząca polecenia *git rebase* mówi o tym, aby nigdy nie stosować go w odniesieniu do gałęzi publicznych.

Zastanów się na przykład, co by się stało, gdybyśmy zmienili bazę gałęzi *main* na gałąź *feature*:



Operacja zmiany bazy spowoduje przeniesienie wszystkich commitów z gałęzi *main* na koniec gałęzi *feature*. Problem polega na tym, że to wszystko będzie miało miejsce wyłącznie w Twoim repozytorium. Wszyscy inni programiści wciąż będą pracowali na pierwotnej gałęzi *main*. Ponieważ operacja zmiany bazy powoduje utworzenie zupełnie nowych commitów, Git uzna, że historia Twojej gałęzi *main* różni się od historii wszystkich innych programistów.

Jedynym sposobem synchronizacji dwóch gałęzi *main* jest scalenie ich ze sobą, co prowadzi do powstania dodatkowego commita scalenia oraz dwóch zbiorów commitów zawierających te same zmiany (pierwotnych i pochodzących z Twojej gałęzi po wykonaniu operacji zmiany bazy). Nie trzeba dodawać, że jest to bardzo kłopotliwa sytuacja.

Dlatego zanim wykonasz polecenie *git rebase*, zawsze zadaj sobie pytanie: „Czy ktoś jeszcze obserwuje tę gałąź?". Jeśli odpowiedź jest twierdząca, cofnij dłonie z klawiatury i zastanów się nad niedestrukcyjnym sposobem wprowadzenia swoich zmian (np. poleceniem *git revert*). Jeśli odpowiedź brzmi „nie", możesz bezpiecznie przepisać historię.

Wymuszone wypychanie

Przy próbie wypchnięcia gałęzi *main*, na której wykonano operację zmiany bazy, z powrotem do repozytorium zdalnego, Git uniemożliwi to działanie z uwagi na konflikt ze zdalną gałęzią *main*. Można jednak wymusić wypchnięcie, przekazując flagę *--force* w następujący sposób:

```
# Be very careful with this command! git push --force
```

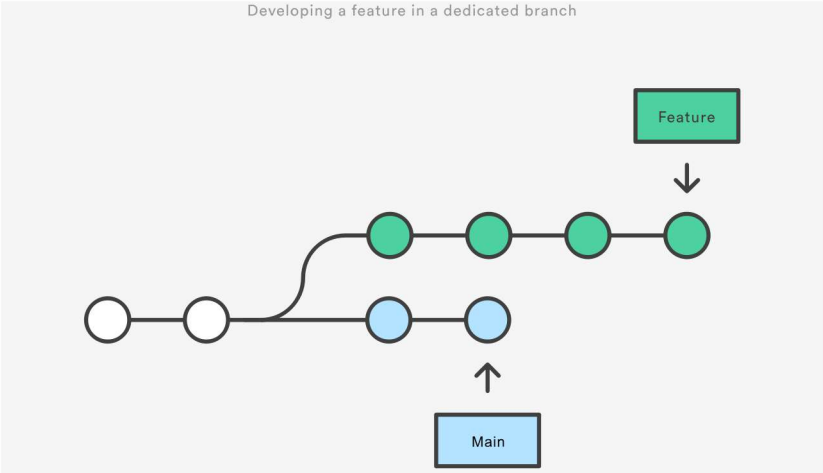
Spowoduje to zastąpienie zdalnej gałęzi *main* gałęzią po zmianie bazy z Twojego repozytorium, co może wprowadzić wiele zamieszania dla reszty zespołu. Zachowaj więc dużą ostrożność i korzystaj z tego polecenia tylko wtedy, gdy naprawdę wiesz, co robisz.

Jednym z niewielu momentów, gdy należy skorzystać z wymuszonego wypychania, jest sytuacja, w której przeprowadzono czyszczenie lokalne po uprzednim wypchnięciu prywatnej gałęzi funkcji do repozytorium zdalnego (np. w celu wykonania kopii zapasowej). To sytuacja, w której można by powiedzieć: „Ups, moim zamiarem nie było wypchnięcie pierwotnej wersji gałęzi funkcji. Zamiast niej miała być wypchnięta bieżąca". Tutaj również ważne jest, aby nikt nie pracował na commitach z pierwotnej wersji gałęzi funkcji.

Przewodnik po przepływie pracy

Operację zmiany bazy można włączyć do istniejącego przepływu pracy Git w takim zakresie, jaki odpowiada Twojemu zespołowi. W tej sekcji przyjrzymy się korzyściom, jakie może zapewnić zmiana bazy na różnych etapach prac programistycznych nad funkcją.

Pierwszym krokiem w dowolnym przepływie pracy wykorzystującym polecenie *git rebase* jest utworzenie dedykowanej gałęzi dla każdej funkcji. Zapewnia to strukturę gałęzi niezbędną do bezpiecznego korzystania z operacji zmiany bazy:



Czyszczenie lokalne

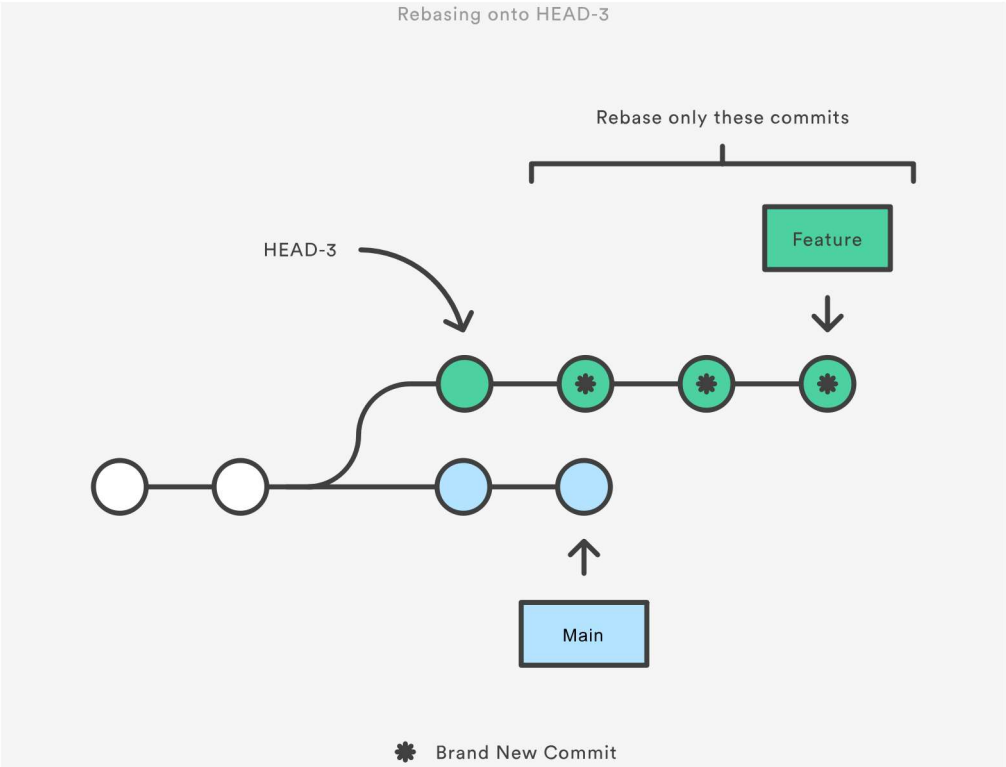
Jednym z najlepszych sposobów włączenia operacji zmiany bazy do przepływu pracy jest czyszczenie lokalnych funkcji w trakcie prac nad nimi. Okresowe wykonywanie interaktywnej zmiany bazy pozwala zyskać pewność, że każdy commit w funkcji jest ukierunkowany i znaczący. Dzięki temu można pisać kod bez obawy, że zostanie on podzielony na odizolowane commity — wszystko można naprawić później.

Podczas wywoływania polecenia *git rebase* są dostępne dwie opcje nowej bazy: gałąź nadrzędna funkcji (np. *main*) lub wcześniejszy commit w funkcji. W sekcji Interaktywna

zmiana bazy przedstawiono przykład pierwszej opcji. Druga opcja sprawdza się, gdy zachodzi potrzeba naprawienia tylko kilku ostatnich commitów. Przykładowo poniższe polecenie inicjuje interaktywną zmianę bazy w odniesieniu do wyłącznie 3 ostatnich commitów.

```
git checkout feature git rebase -i HEAD~3
```

Zdefiniowanie wskaźnika *HEAD~3* jako nowej bazy nie powoduje faktycznego przeniesienia gałęzi, a jedynie interaktywne przepisanie 3 commitów, które po nim następują. Należy pamiętać, że nie spowoduje to włączenia zmian nadrzędnych do gałęzi *feature*.



Aby przepisać całą funkcję przy użyciu tej metody, można użyć polecenia *git merge-base*, które pozwoli wyszukać oryginalną bazę gałęzi *feature*. Poniższe polecenie zwraca identyfikator commita oryginalnej bazy, który można następnie przekazać do polecenia *git rebase*:

```
git merge-base feature main
```

Takie zastosowanie interaktywnej zmiany bazy jest doskonałym sposobem na włączenie polecenia *git rebase* do przepływu pracy, ponieważ dotyczy wyłącznie gałęzi lokalnych. Jedyną rzeczą widoczną dla innych programistów będzie Twój gotowy produkt, czyli przejrzysta, łatwa do prześledzenia historia gałęzi funkcji.

Jednakże działa to tylko w przypadku prywatnych gałęzi funkcji. Jeśli współpracujesz z innymi programistami za pośrednictwem tej samej gałęzi funkcji, ta gałąź staje się publiczna i nie można przepisywać jej historii.

Polecenie *git merge* nie oferuje żadnej alternatywy dla czyszczenia lokalnych commitów za pomocą interaktywnej zmiany bazy.

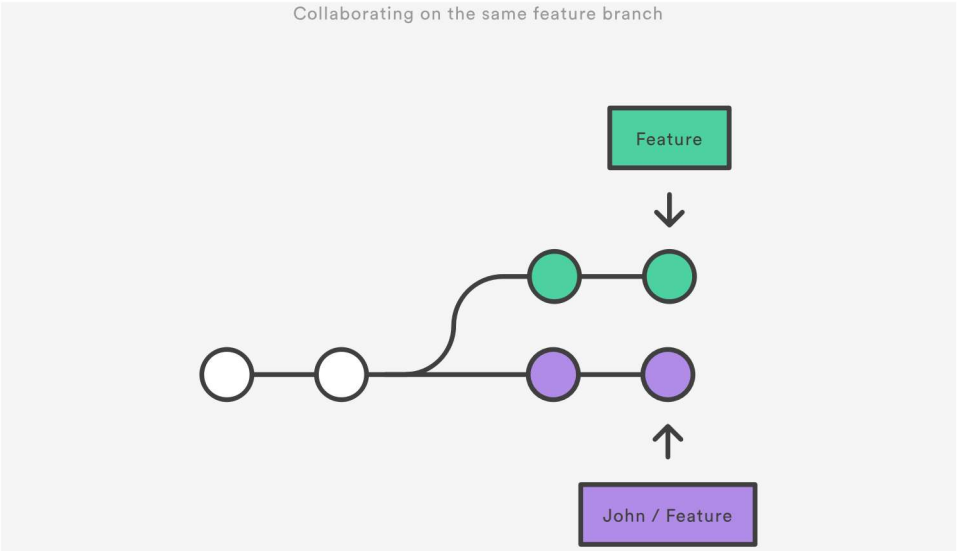
Włączanie zmian nadrzędnych do funkcji

W sekcji Przegląd koncepcji przedstawiono, w jaki sposób można uwzględnić w gałęzi funkcji nadrzędne zmiany z gałęzi *main*, używając polecenia *git merge* lub *git rebase*. Scalanie jest bezpieczną opcją, która pozwala zachować całą historię repozytorium, natomiast operacja zmiany bazy pozwala uzyskać liniową historię dzięki przeniesieniu gałęzi funkcji na koniec gałęzi *main*.

Takie użycie polecenia *git rebase* przypomina czyszczenie lokalne (i można je realizować równocześnie), jednak w procesie uwzględniane są te nadrzędne commity z gałęzi *main*.

Należy pamiętać, że zmiana bazy na gałąź zdalną zamiast gałęzi *main* jest jak najbardziej dopuszczalna. Może mieć to miejsce podczas współpracy nad tą samą funkcją z innym programistą, gdy zachodzi potrzeba włączenia jego lub jej zmian do swojego repozytorium.

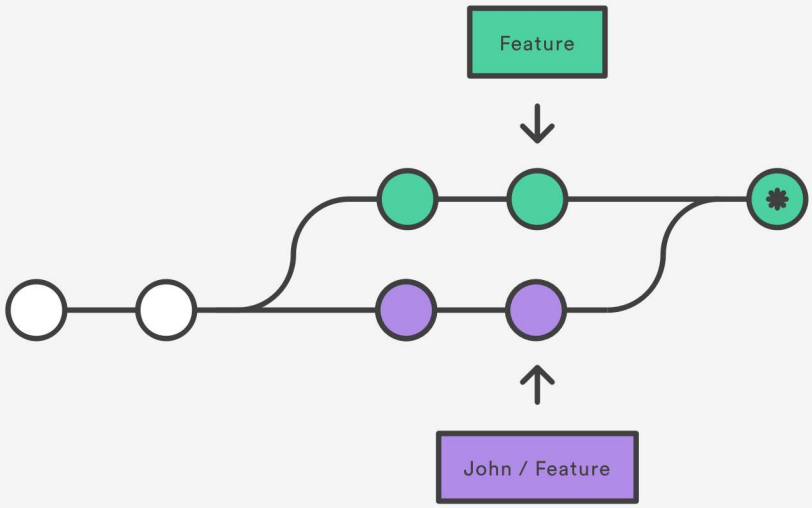
Załóżmy na przykład, że Ty oraz inny programista o imieniu John dodaliście commity do gałęzi *feature*. Po pobraniu zdalnej gałęzi *feature* z repozytorium Johna Twoje repozytorium może wyglądać następująco:



Ten podział można wyeliminować tak samo, jak przy włączaniu zmian nadrzędnych z gałęzi *main*: poprzez scalenie Twojej lokalnej gałęzi *feature* z gałęzią *john/feature* lub poprzez wykonanie zmiany bazy w celu dołączenia Twojej lokalnej gałęzi *feature* na koniec gałęzi *john/feature*.

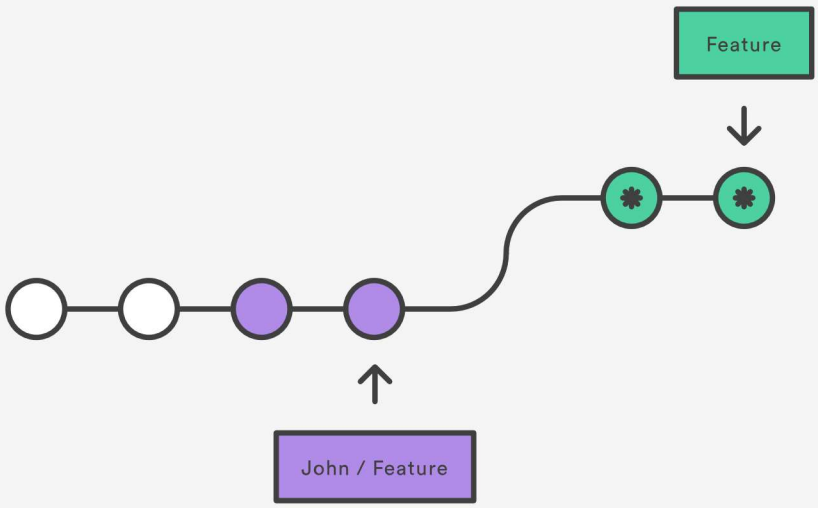
Merging vs. rebasing onto a remote branch

Merging



Merge Commit

Rebasing



Brand New Commits

Należy zauważyć, że taka zmiana bazy nie narusza złotej reguły zmiany bazy, ponieważ są przenoszone wyłącznie commity z lokalnej gałęzi *feature* — wszystko, co znajduje się przed nimi, pozostaje nietknięte. To jakby powiedzieć: „Dodaj moje zmiany do tego, co już zrobił John". W większości przypadków jest to bardziej intuicyjne niż synchronizacja z gałęzią zdalną za pośrednictwem commita scalenia.

Domyślnie polecenie *git pull* wykonuje operację scalania, jednak można wymusić, aby przeprowadzana była integracja gałęzi zdalnej za pomocą operacji zmiany bazy, przekazując do niego opcję *--rebase*.

Przegląd funkcji za pomocą pull requestu

Jeśli wykorzystujesz pull requesty w procesie przeglądu kodu, musisz unikać używania polecenia *git rebase* po utworzeniu pull requestu. Gdy tylko utworzysz pull request, inni programiści będą mieli wgląd w Twoje commity, co oznacza, że gałąź stanie się publiczna. Przepisanie jej historii uniemożliwi systemowi Git oraz innym członkom zespołu prześledzenie kolejnych commitów dodanych do funkcji.

Wszelkie zmiany pochodzące od innych programistów należy włączać za pomocą polecenia *git merge*, a nie *git rebase*.

W związku z tym zazwyczaj dobrze jest wyczyścić kod za pomocą interaktywnej zmiany bazy, zanim prześle się pull request.

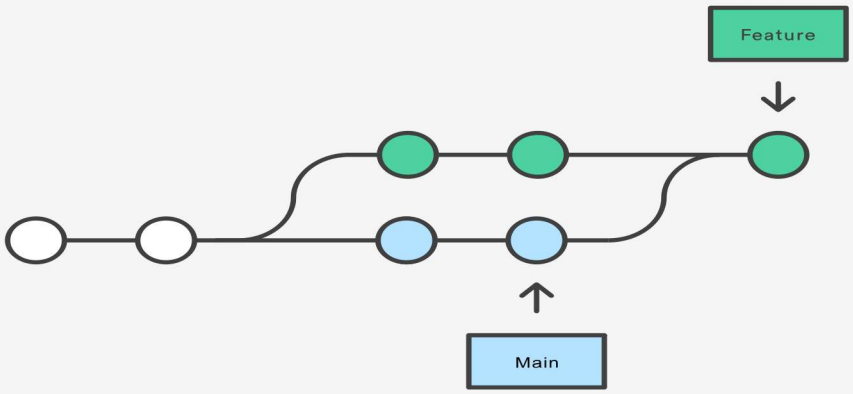
Włączanie zatwierdzonej funkcji

Gdy zespół zatwierdzi funkcję, można wykonać zmianę jej bazy na końcówkę gałęzi *main* przed użyciem polecenia *git merge* w celu włączenia funkcji do głównej bazy kodu.

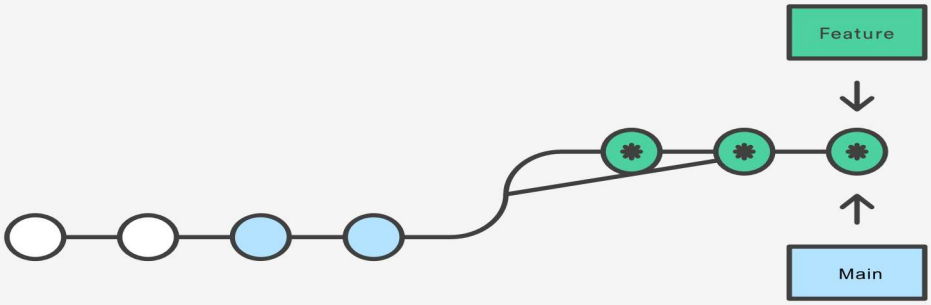
Ta sytuacja jest podobna do włączania zmian nadrzędnych do gałęzi funkcji, ale z uwagi na brak możliwości przepisania commitów w gałęzi *main* do włączenia funkcji trzeba ostatecznie użyć polecenia *git merge*. Wykonanie zmiany bazy przed scalaniem pozwala jednak zyskać pewność, że będzie to scalanie z przewinięciem, które umożliwi otrzymanie idealnie liniowej historii. Zapewnia również możliwość połączenia wszelkich kolejnych commitów dodanych w trakcie pull requestu.

Integrating a feature into main with and without a rebase

Initial State

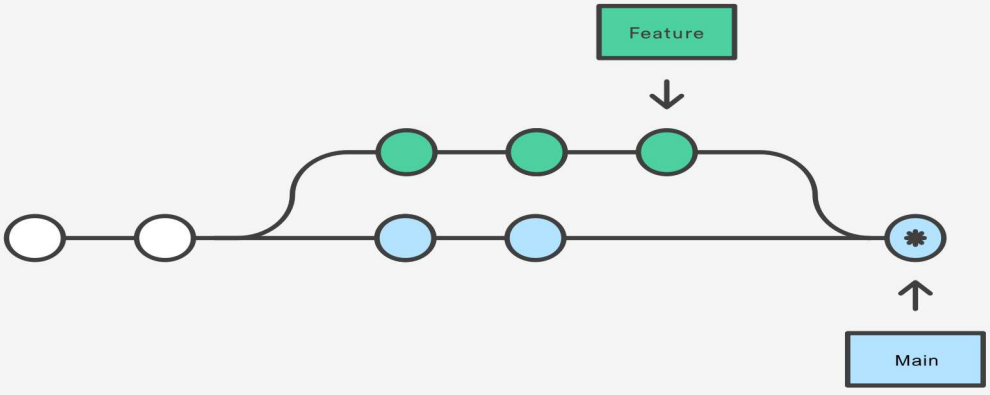


Rebase and Merge



✱ Brand New Commits

Merge without rebasing



✱ Merge Commit

Jeśli nie czujesz się pewnie, korzystając z polecenia *git rebase*, zawsze możesz wykonać operację zmiany bazy w gałęzi tymczasowej. Dzięki temu w razie przypadkowego namieszania w historii funkcji można przywrócić gałąź pierwotną i spróbować ponownie. Przykład:

```
git checkout feature
git checkout -b temporary-branch
git rebase -i main
# [Clean up the history]
git checkout main
git merge temporary-branch
```

Podsumowanie

To właściwie wszystko, co musisz wiedzieć, aby rozpocząć korzystanie z operacji zmiany bazy w gałęziach. Jeśli wolisz przejrzystą, liniową historię pozbawioną zbędnych commitów scalenia, podczas włączania zmian z innej gałęzi sięgnij po polecenie *git rebase*, zamiast *git merge*.

Jeśli jednak chcesz zachować kompletną historię projektu i uniknąć ryzyka przepisania publicznych commitów, możesz pozostać przy poleceniu *git merge*. Każda z opcji jest dobra, jednak teraz możesz przynajmniej uwzględnić zalety polecenia *git rebase*.