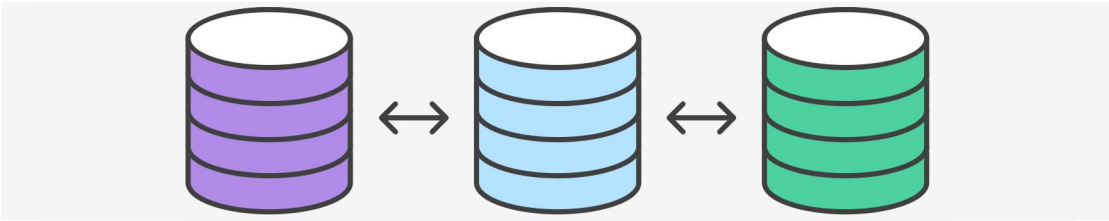


Porównywanie przepływów pracy

Git jest obecnie najczęściej używanym systemem kontroli wersji. Przepływ pracy Git to przepis lub zalecenie, jak używać Git do wykonywania pracy w spójny i produktywny sposób. Przepływy pracy Git zachęcają programistów i DevOpszespoły, aby efektywnie i konsekwentnie wykorzystywać Git. Git oferuje dużą elastyczność w zarządzaniu zmianami przez użytkowników. Biorąc pod uwagę, że Git koncentruje się na elastyczności, nie ma standardowego procesu interakcji z Git. Podczas pracy z zespołem nad projektem zarządzanym przez Git ważne jest, aby upewnić się, że wszyscy zgadzają się co do sposobu zastosowania przepływu zmian. Aby upewnić się, że zespół jest na tej samej stronie, należy opracować lub wybrać uzgodniony przepływ pracy Git. Istnieje kilka opublikowanych przepływów pracy Git, które mogą być odpowiednie dla Twojego zespołu. W tym miejscu omówimy niektóre z tych opcji przepływu pracy Git.

Różnorodność możliwych przepływów pracy może utrudnić określenie, od czego zacząć wdrażanie Git w miejscu pracy. Ta strona stanowi punkt wyjścia do przeglądu najczęstszych przepływów pracy Git dla zespołów programistycznych.

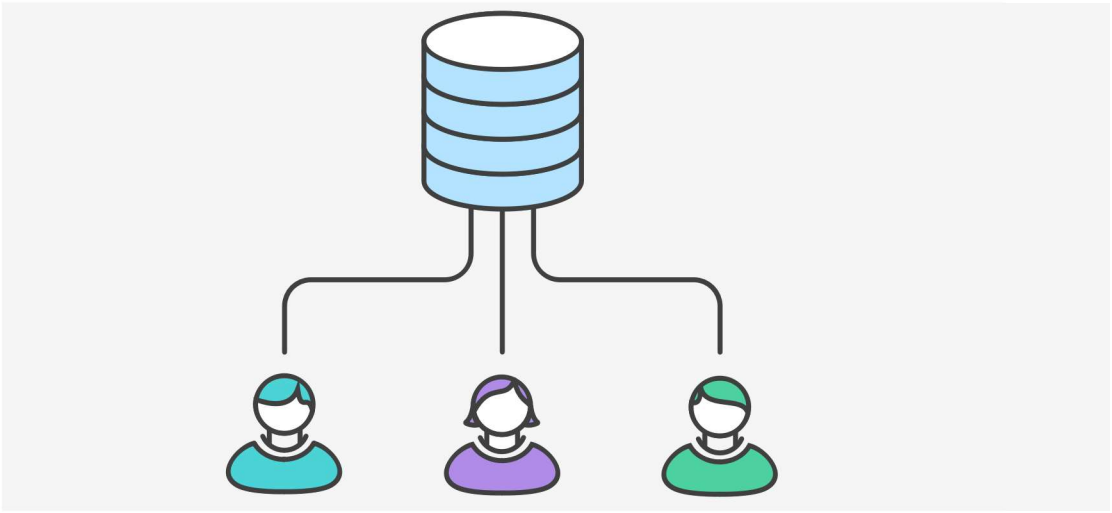
Czytając, pamiętaj, że te przepływy pracy mają być wytycznymi, a nie konkretnymi regułami. Chcemy pokazać Ci, co jest możliwe, abyś mógł łączyć i dopasowywać aspekty z różnych przepływów pracy, tak aby odpowiadały Twoim indywidualnym potrzebom.

Co to jest pomyślny przepływ pracy Git?

Oceniając przepływ pracy dla swojego zespołu, najważniejsze jest, aby wziąć pod uwagę kulturę swojego zespołu. Chcesz, aby przepływ pracy zwiększał efektywność Twojego zespołu i nie był obciążeniem ograniczającym produktywność. Oto kilka rzeczy, które należy wziąć pod uwagę podczas oceny przepływu pracy Git:

- Czy ten przepływ pracy jest skalowany wraz z wielkością zespołu?
- Czy łatwo jest cofnąć błędy i błędy w tym przepływie pracy?
- Czy ten przepływ pracy narzuca zespołowi nowe, niepotrzebne obciążenie poznawcze?

Scentralizowany przepływ pracy



Scentralizowany przepływ pracy to świetny przepływ pracy Git dla zespołów przechodzących z SVN. Podobnie jak Subversion, scentralizowany przepływ pracy wykorzystuje centralne repozytorium, które służy jako pojedynczy punkt wejścia dla wszystkich zmian w projekcie. Zamiast *trunk*, wywoływana jest domyślna gałąź programistyczna *main* i wszystkie zmiany są zatwierdzane w tej gałęzi. Ten przepływ pracy nie wymaga żadnych innych gałęzi poza *main*.

Przejście do rozproszonego systemu kontroli wersji może wydawać się trudnym zadaniem, ale nie musisz zmieniać istniejącego przepływu pracy, aby skorzystać z usługi Git. Twój zespół może rozwijać projekty dokładnie w taki sam sposób, jak w Subversion.

Jednak używanie Git do obsługi przepływu pracy programistycznej ma kilka zalet w porównaniu z SVN. Po pierwsze, daje każdemu deweloperowi własną lokalną kopię całego projektu. To izolowane środowisko pozwala każdemu programiście pracować niezależnie od wszystkich innych zmian w projekcie — mogą dodawać zatwierdzenia do swojego lokalnego repozytorium i całkowicie zapomnieć o wcześniejszych programach, dopóki nie będzie to dla nich wygodne.

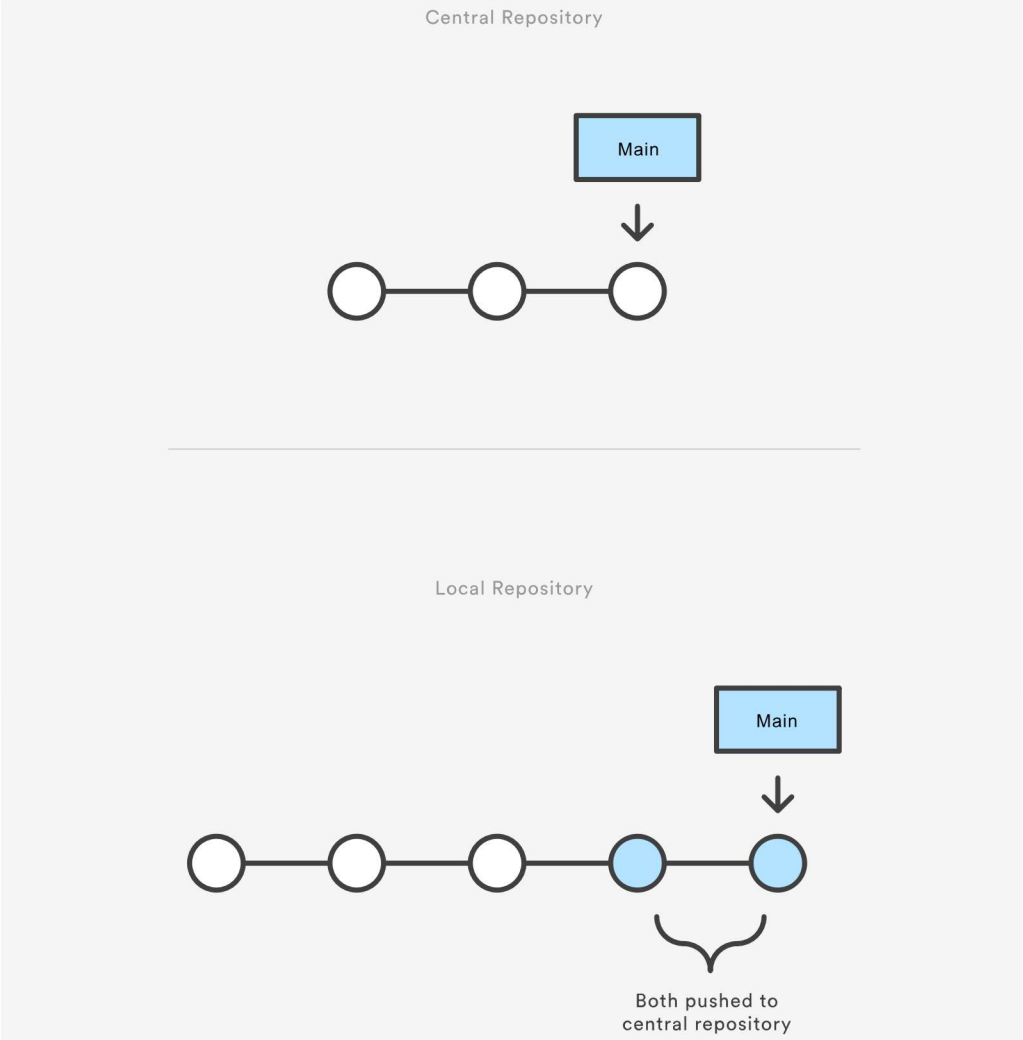
Po drugie, daje dostęp do solidnego modelu rozgałęziania i łączenia w Git. W przeciwieństwie do SVN, gałęzie Git są zaprojektowane jako bezpieczny mechanizm integracji kodu i udostępniania zmian między repozytoriami. Scentralizowany przepływ pracy jest podobny do innych przepływów pracy pod względem wykorzystania zdalnego repozytorium hostowanego po stronie serwera, które programiści wypychają i ściągają. W porównaniu z innymi przepływami pracy, scentralizowany przepływ pracy nie ma zdefiniowanych żądań ściągnięcia ani wzorców rozwidlenia. Scentralizowany przepływ pracy jest ogólnie lepiej dostosowany do zespołów migrujących z SVN do Git i mniejszych zespołów.

Jak to działa

Deweloperzy zaczynają od sklonowania centralnego repozytorium. We własnych lokalnych kopiach projektu edytują pliki i zatwierdzają zmiany, tak jak robiliby to w przypadku SVN; jednak te nowe zatwierdzenia są przechowywane lokalnie — są całkowicie odizolowane od centralnego repozytorium. Pozwala to programistom odroczyć synchronizację do momentu, gdy znajdą się w dogodnym punkcie przerwania.

Aby opublikować zmiany w oficjalnym projekcie, programiści „wpychają" swój lokalny *main*oddział do centralnego repozytorium. Jest to odpowiednik *svn commit*, z wyjątkiem tego, że dodaje wszystkie lokalne zatwierdzenia, które nie znajdują się jeszcze w *main*gałęzi centralnej.

Zainicjuj centralne repozytorium



Najpierw ktoś musi stworzyć centralne repozytorium na serwerze. Jeśli jest to nowy projekt, możesz zainicjować puste repozytorium. W przeciwnym razie będziesz musiał zaimportować istniejące repozytorium Git lub SVN.

Repozytoria centralne powinny zawsze być czystymi repozytoriami (nie powinny mieć katalogu roboczego), które można utworzyć w następujący sposób:

```
ssh user@host git init --bare /path/to/repo.git
```

Upewnij się, że używasz prawidłowej nazwy użytkownika SSH dla *user*domeny , domeny lub adresu IP serwera dla *host*, a także lokalizacji, w której chcesz przechowywać repozytorium */path/to/repo.git*. Zauważ, że *.git*rozszerzenie jest zwykle dołączane do nazwy repozytorium, aby wskazać, że jest to czyste repozytorium.

Hostowane centralne repozytoria

Centralne repozytoria są często tworzone przez zewnętrzne usługi hostingowe Git, takie jak Bitbucket Cloud lub Bitbucket Server . Omówiony powyżej proces inicjowania samego repozytorium jest obsługiwany przez usługę hostingową. Usługa hostingowa poda następnie adres centralnego repozytorium, do którego będzie można uzyskać dostęp z lokalnego repozytorium.

Sklonuj centralne repozytorium

Następnie każdy programista tworzy lokalną kopię całego projektu. Odbywa się to za pomocą git clonepolecenia:

```
git clone ssh://user@host/path/to/repo.git
```

Kiedy sklonujesz repozytorium, Git automatycznie dodaje skrót o nazwie *origin*, który wskazuje na repozytorium „nadrzędne", przy założeniu, że będziesz chciał wchodzić z nim w interakcje w dalszej części drogi.

Wprowadź zmiany i zatwierdź

Gdy repozytorium zostanie sklonowane lokalnie, programista może wprowadzać zmiany za pomocą standardowego procesu zatwierdzania Git: edycji, etapu i zatwierdzenia. Jeśli nie jesteś zaznajomiony z obszarem przemieszczania, jest to sposób na przygotowanie zatwierdzenia bez konieczności umieszczania każdej zmiany w katalogu roboczym. Pozwala to na tworzenie wysoce skoncentrowanych zatwierdzeń, nawet jeśli wprowadziłeś wiele lokalnych zmian.

```
git status # View the state of the repo
git add <some-file> # Stage a file
git commit # Commit a file</some-file>
```

Pamiętaj, że ponieważ te polecenia tworzą lokalne zatwierdzenia, John może powtarzać ten proces tyle razy, ile chce, nie martwiąc się o to, co dzieje się w centralnym repozytorium. Może to być bardzo przydatne w przypadku dużych funkcji, które należy podzielić na prostsze, bardziej atomowe fragmenty.

Prześlij nowe zatwierdzenia do centralnego repozytorium

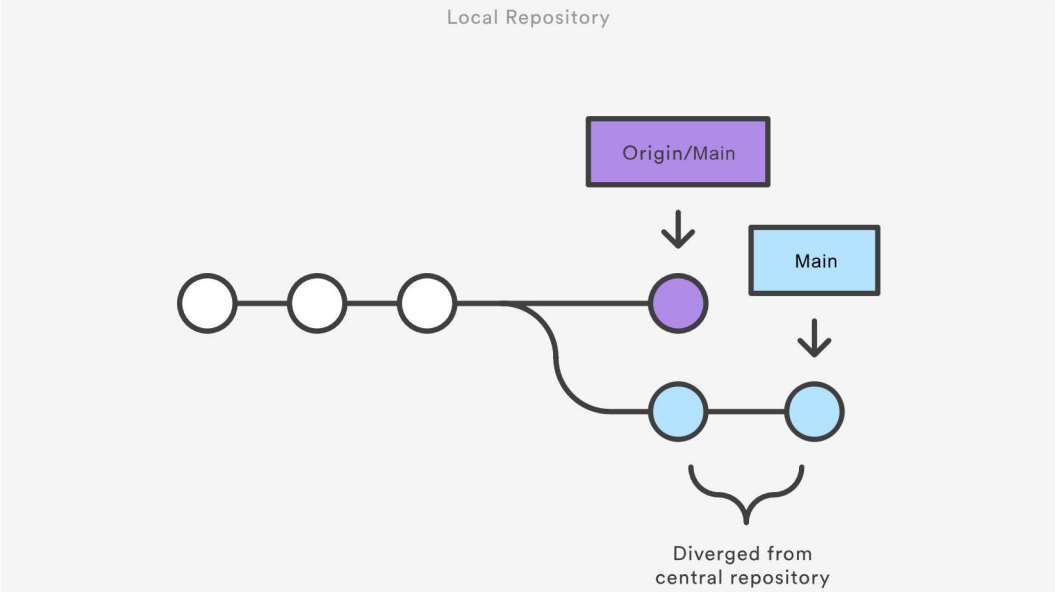
Gdy lokalne repozytorium zatwierdzi nowe zmiany. Te zmiany będą musiały zostać wprowadzone, aby udostępnić je innym programistom w projekcie.

```
git push origin main
```

To polecenie wypchnie nowe zatwierdzone zmiany do centralnego repozytorium. Podczas wypychania zmian do centralnego repozytorium możliwe jest, że aktualizacje od innego programisty zostały wcześniej przekazane, które zawierają kod, który jest w konflikcie z zamierzonymi aktualizacjami wypychanymi. Git wyświetli komunikat wskazujący na ten konflikt. W tej sytuacji *git pull*najpierw trzeba będzie wykonać. Ten scenariusz konfliktu zostanie rozwinięty w następnej sekcji.

Zarządzanie konfliktami

Centralne repozytorium reprezentuje oficjalny projekt, więc jego historię zaangażowania należy traktować jako świętą i niezmienną. Jeśli lokalne zatwierdzenia dewelopera odbiegają od centralnego repozytorium, Git odmówi wysłania ich zmian, ponieważ zastąpiłoby to oficjalne zatwierdzenia.



Zanim deweloper będzie mógł opublikować swoją funkcję, musi pobrać zaktualizowane centralne zatwierdzenia i ponownie oprzeć na nich swoje zmiany. To tak, jakby powiedzieć: „Chcę dodać moje zmiany do tego, co wszyscy już zrobili”. Rezultatem jest idealnie liniowa historia, podobnie jak w tradycyjnych przepływach pracy SVN.

Jeśli lokalne zmiany bezpośrednio kolidują z wcześniejszymi zatwierdzeniami, Git wstrzyma proces zmiany bazy i umożliwi ręczne rozwiązanie konfliktów. Zaletą Gita jest to, że używa tych samych poleceń *git status*i *git add*poleceń zarówno do generowania zatwierdzeń, jak i rozwiązywania konfliktów scalania. Ułatwia to nowym programistom zarządzanie własnymi połączeniami. Dodatkowo, jeśli wpadną w kłopoty, Git bardzo ułatwia przerwanie całej zmiany bazy i spróbuj ponownie (lub poszukaj pomocy).

Przykład

Weźmy ogólny przykład tego, jak typowy mały zespół współpracowałby przy użyciu tego przepływu pracy. Zobaczymy, jak dwóch programistów, John i Mary, może pracować nad osobnymi funkcjami i udostępniać swój wkład za pośrednictwem scentralizowanego repozytorium.

John pracuje nad swoim filmem

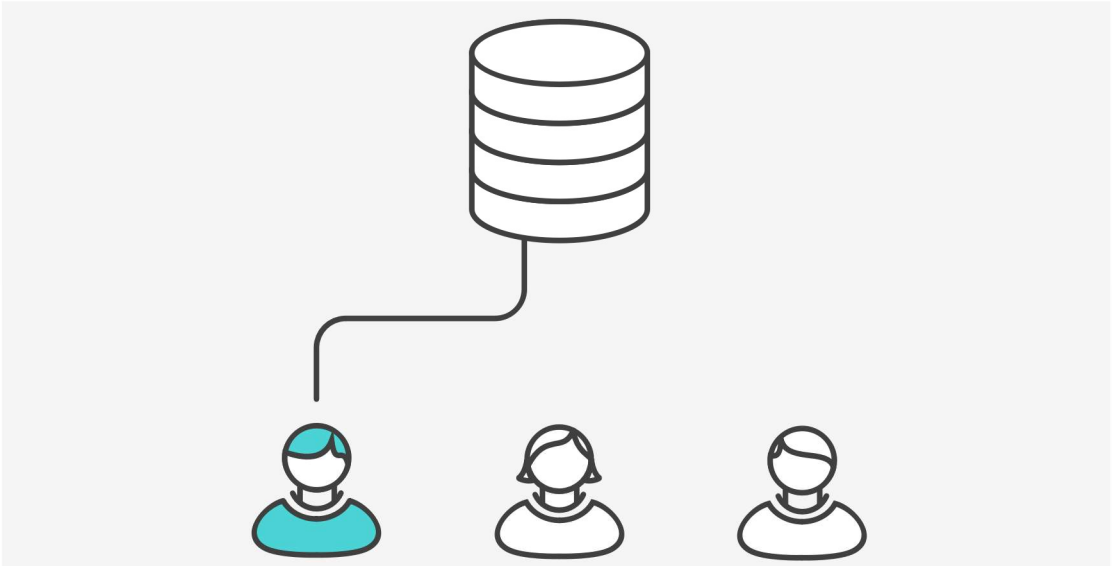
W swoim lokalnym repozytorium John może opracowywać funkcje przy użyciu standardowego procesu zatwierdzania Git: edycji, etapu i zatwierdzenia.

Pamiętaj, że ponieważ te polecenia tworzą lokalne zatwierdzenia, John może powtarzać ten proces tyle razy, ile chce, nie martwiąc się o to, co dzieje się w centralnym repozytorium.

Mary pracuje nad swoim filmem

Tymczasem Mary pracuje nad własną funkcją we własnym lokalnym repozytorium, korzystając z tego samego procesu edycji/etapu/zatwierdzania. Podobnie jak John, nie obchodzi ją, co dzieje się w centralnym repozytorium, i tak naprawdę nie obchodzi ją, co John robi w swoim lokalnym repozytorium, ponieważ wszystkie lokalne repozytoria są prywatne .

John publikuje swój artykuł

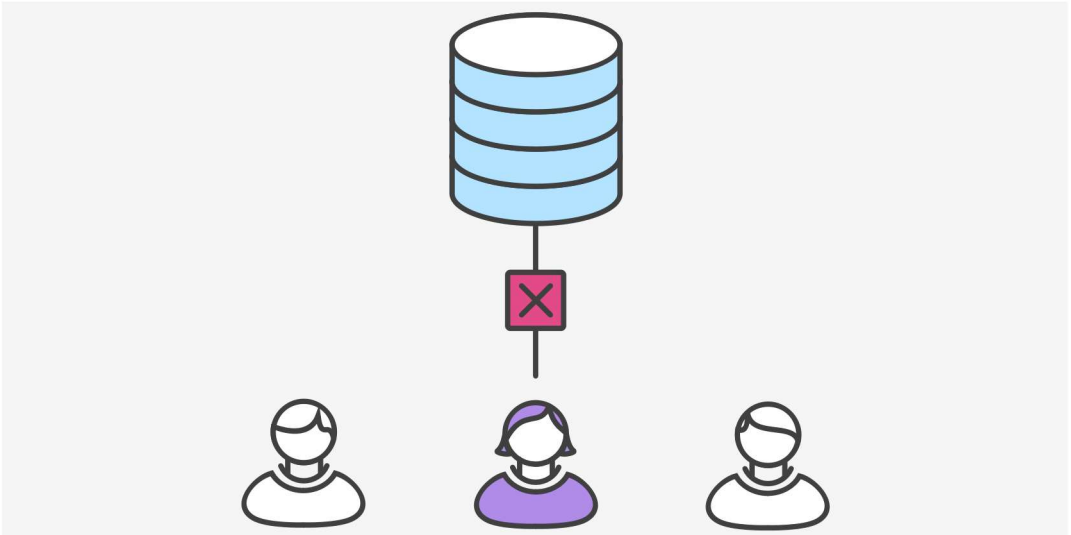


Gdy John zakończy swoją funkcję, powinien opublikować swoje lokalne zatwierdzenia w centralnym repozytorium, aby inni członkowie zespołu mieli do niego dostęp. Może to zrobić za pomocą git pushpolecenia, na przykład:

```
git push origin main
```

Pamiętaj, że *origin*jest to zdalne połączenie z centralnym repozytorium, które Git utworzył, gdy John je sklonował. Argument *main*mówi Gitowi, aby spróbował sprawić , by gałąź *origin*'s *main*wyglądała jak jego *main*gałąź lokalna. Ponieważ centralne repozytorium nie zostało zaktualizowane od czasu sklonowania go przez Johna, nie spowoduje to żadnych konfliktów, a wypychanie będzie działać zgodnie z oczekiwaniami.

Mary próbuje opublikować swój artykuł



Zobaczmy, co się stanie, jeśli Mary spróbuje przekazać swoją funkcję po tym, jak Jan pomyślnie opublikuje swoje zmiany w centralnym repozytorium. Może użyć dokładnie tego samego polecenia push:

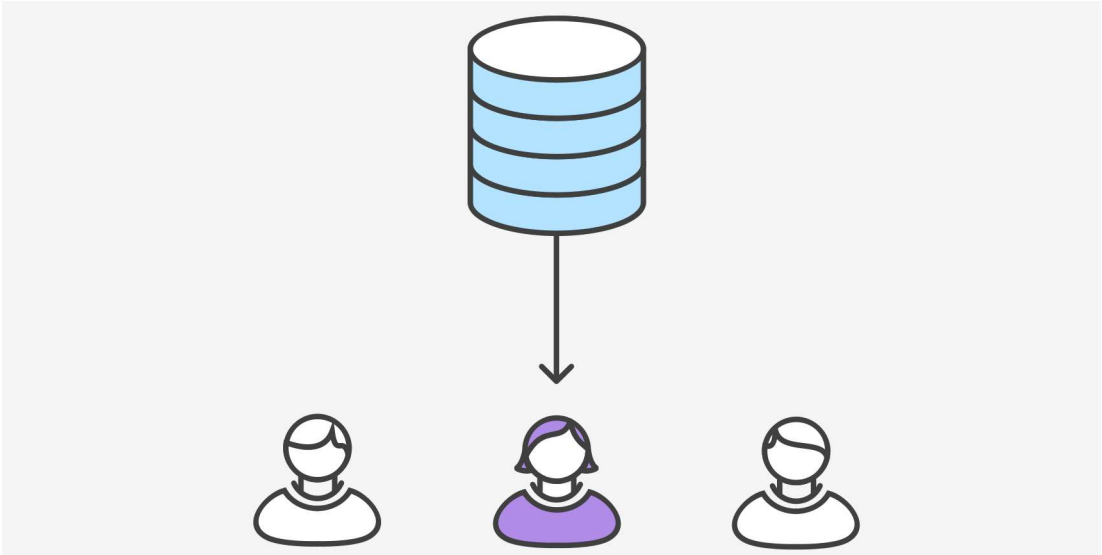
```
git push origin main
```

Ale ponieważ jej lokalna historia odbiega od centralnego repozytorium, Git odrzuci żądanie z dość szczegółowym komunikatem o błędzie:

```
error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')
hint: before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Uniemożliwia to Mary nadpisanie oficjalnych zobowiązań. Musi pobrać aktualizacje Johna do swojego repozytorium, zintegrować je z lokalnymi zmianami, a następnie spróbować ponownie.

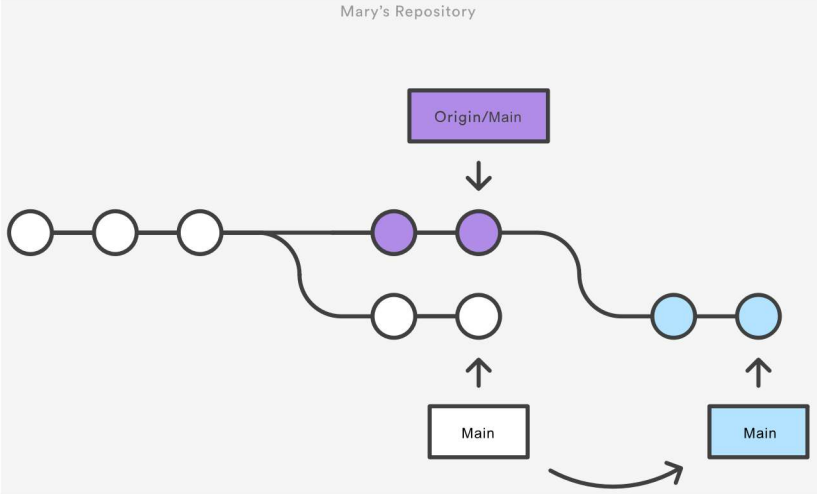
Mary ponownie bazuje na zatwierdzeniu (zatwierdzeniach) Johna



Mary może użyć git pulldo wprowadzenia zmian w swoim repozytorium. To polecenie jest trochę jak *svn update*— ściąga całą historię wcześniejszych zmian do lokalnego repozytorium Mary i próbuje zintegrować ją z jej lokalnymi zatwierdzeniami:

```
git pull --rebase origin main
```

Opcja `--rebase`mówi Gitowi, aby przeniósł wszystkie zatwierdzenia Mary na koniec *main*gałęzi po zsynchronizowaniu ich ze zmianami z centralnego repozytorium, jak pokazano poniżej:



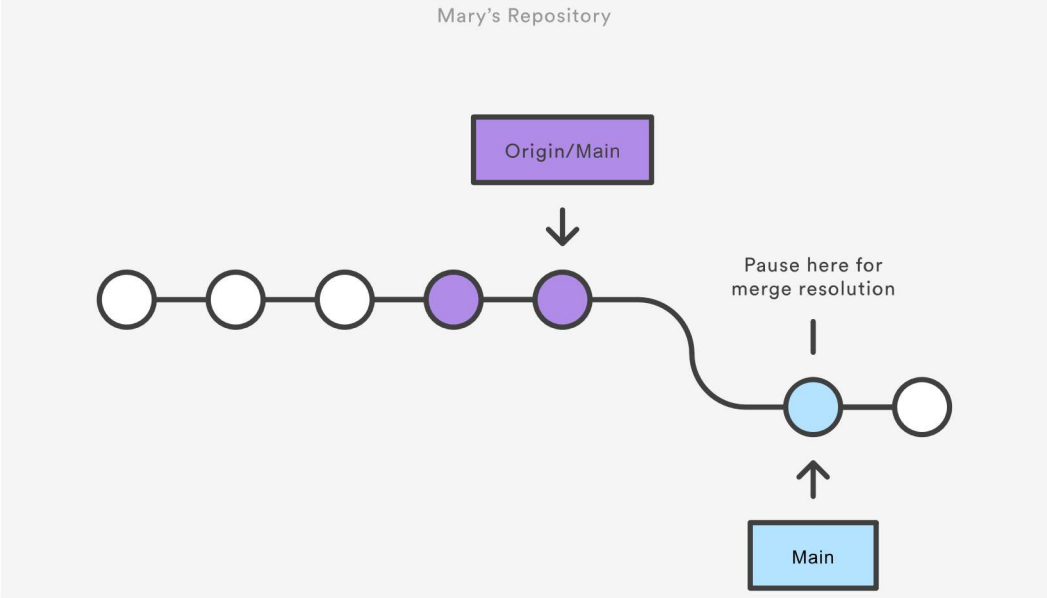
Ściągnięcie nadal działałoby, gdybyś zapomniał o tej opcji, ale za każdym razem, gdy ktoś musiałby zsynchronizować się z centralnym repozytorium, miałbyś niepotrzebne „zatwierdzenie scalania". W przypadku tego przepływu pracy zawsze lepiej jest zmienić bazę zamiast generować zatwierdzenie scalające.

Mary rozwiązuje konflikt scalania

Zmiana bazy polega na przeniesieniu każdego zatwierdzenia lokalnego do zaktualizowanej *main*gałęzi pojedynczo. Oznacza to, że łapiesz konflikty scalania na zasadzie zatwierdzania po zatwierdzeniu, zamiast rozwiązywać je wszystkie w jednym masowym zatwierdzeniu scalającym. Dzięki temu Twoje zatwierdzenia są tak skoncentrowane, jak to możliwe, i zapewnia czystą historię projektu. To z kolei znacznie ułatwia ustalenie, gdzie wprowadzono błędy i, jeśli to konieczne, cofnięcie zmian przy minimalnym wpływie na projekt.

Jeśli Mary i John pracują nad niepowiązanymi funkcjami, jest mało prawdopodobne, że proces zmiany bazy spowoduje konflikty. Ale jeśli tak się stanie, Git wstrzyma rebase przy bieżącym zatwierdzeniu i wyświetli następujący komunikat wraz z odpowiednimi instrukcjami:

```
CONFLICT (content): Merge conflict in <some-file>
```



Wspaniałą rzeczą w Git jest to, że każdy może rozwiązać własne konflikty scalania. W naszym przykładzie Mary po prostu uruchomiłaby a, git statusaby zobaczyć, gdzie jest problem. Skonfliktowane pliki pojawią się w sekcji Niescalone ścieżki:

```
# Unmerged paths:
#  (use "git reset HEAD <some-file>..." to unstage)
#  (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
# both modified: <some-file>
```

Następnie zmodyfikuje plik(i) zgodnie z własnymi upodobaniami. Gdy jest zadowolona z wyniku, może przygotować plik(i) w zwykły sposób i pozwolić git rebasezająć się resztą:

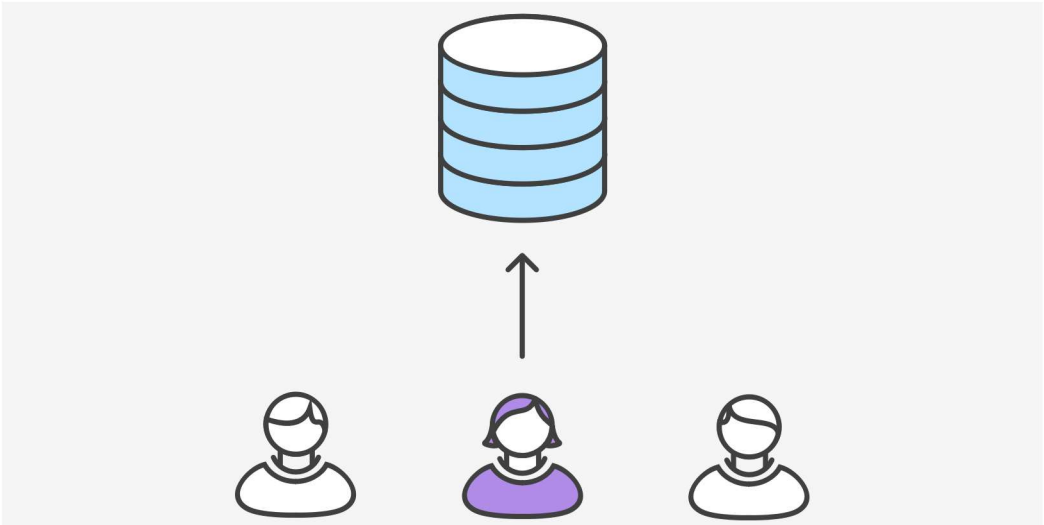
```
git add <some-file>
git rebase --continue
```

I to wszystko. Git przejdzie do następnego zatwierdzenia i powtórzy proces dla wszystkich innych zatwierdzeń, które generują konflikty.

Jeśli dojdziesz do tego punktu i zdasz sobie sprawę, a nie masz pojęcia, co się dzieje, nie panikuj. Po prostu wykonaj następujące polecenie, a wrócisz do miejsca, w którym zacząłeś:

```
git rebase --abort
```

Mary z powodzeniem publikuje swój artykuł



Po zakończeniu synchronizacji z centralnym repozytorium Mary będzie mogła pomyślnie opublikować swoje zmiany:

```
git push origin main
```

Gdzie iść stąd?

Jak widać, możliwe jest zreplikowanie tradycyjnego środowiska programistycznego Subversion przy użyciu tylko kilku poleceń Git. Jest to świetne do przenoszenia zespołów z SVN, ale nie wykorzystuje rozproszonego charakteru Git.

Scentralizowany przepływ pracy jest świetny dla małych zespołów. Opisany powyżej proces rozwiązywania konfliktów może stanowić wąskie gardło w miarę zwiększania się zespołu. Jeśli Twój zespół dobrze sobie radzi ze scentralizowanym przepływem pracy, ale chce usprawnić współpracę, zdecydowanie warto zbadać korzyści płynące z przepływu pracy gałęzi funkcji . Dedykując osobną gałąź każdej funkcji, możliwe jest zainicjowanie dogłębnych dyskusji na temat nowych dodatków przed włączeniem ich do oficjalnego projektu.

Inne popularne przepływy pracy

Scentralizowany przepływ pracy jest zasadniczo elementem konstrukcyjnym dla innych przepływów pracy Git. Najpopularniejsze przepływy pracy Git będą miały swego rodzaju scentralizowane repozytorium, z którego poszczególni programiści będą pchać i pobierać. Poniżej pokrótce omówimy kilka innych popularnych przepływów pracy Git. Te rozszerzone przepływy pracy oferują bardziej wyspecjalizowane wzorce w zakresie zarządzania gałęziami w celu opracowywania funkcji, poprawek i ewentualnego wydania.

Rozgałęzienie funkcji

Rozgałęzienie funkcji jest logicznym rozszerzeniem scentralizowanego przepływu pracy. Główną ideą przepływu pracy gałęzi funkcji jest to, że cały rozwój funkcji powinien odbywać się w dedykowanej gałęzi, a nie w *main*gałęzi. Ta enkapsulacja ułatwia wielu programistom pracę nad konkretną funkcją bez zakłócania głównej bazy kodu. Oznacza to również, że *main*gałąź nigdy nie powinna zawierać zepsutego kodu, co jest ogromną zaletą w środowiskach ciągłej integracji.

Przepływ pracy Gitflow

Przepływ pracy Gitflow został po raz pierwszy opublikowany w cenionym wpisie na blogu z 2010 roku autorstwa Vincenta Driessena z nvie . Gitflow Workflow definiuje ścisły model rozgałęziania zaprojektowany wokół wydania projektu. Ten przepływ pracy nie dodaje żadnych nowych koncepcji ani poleceń poza to, co jest wymagane dla przepływu pracy gałęzi funkcji. Zamiast tego przypisuje bardzo konkretne role do różnych gałęzi i określa, jak i kiedy powinny one ze sobą współdziałać.

Rozwidlenie przepływu pracy

Przepływ pracy rozwidlania różni się zasadniczo od innych przepływów pracy omówionych w tym samouczku. Zamiast używać pojedynczego repozytorium po stronie serwera, które działa jako „centralna" baza kodu, każdy programista otrzymuje repozytorium po stronie serwera. Oznacza to, że każdy współtwórca ma nie jedno, ale dwa repozytoria Git: prywatne lokalne i publiczne po stronie serwera.

Wytyczne

Nie ma jednego rozmiaru pasującego do wszystkich przepływów pracy Git. Jak wspomniano wcześniej, ważne jest, aby opracować przepływ pracy Git, który zwiększy produktywność zespołu. Oprócz kultury zespołowej przepływ pracy powinien również uzupełniać kulturę biznesową. Funkcje Git, takie jak gałęzie i tagi, powinny uzupełniać harmonogram wydań Twojej firmy. Jeśli Twój zespół korzysta z oprogramowania do zarządzania projektami śledzącymi zadania, możesz użyć gałęzi, które odpowiadają zadaniom w toku. Ponadto niektóre wytyczne, które należy wziąć pod uwagę przy podejmowaniu decyzji o przepływie pracy, to:

Oddziały krótkotrwałe

Im dłużej oddział żyje oddzielnie od oddziału produkcyjnego, tym większe ryzyko konfliktów scalania i wyzwań związanych z wdrożeniem. Krótkotrwałe gałęzie promują czystsze łączenia i wdrożenia.

Zminimalizuj i uprość powroty

Ważne jest, aby przepływ pracy pomagał aktywnie zapobiegać scalaniu, które trzeba będzie cofnąć. Przykładem jest przepływ pracy, który testuje gałąź przed umożliwieniem jej scalenia z *main*gałęzią. Jednak wypadki się zdarzają. Biorąc to pod uwagę, dobrze jest

mieć przepływ pracy, który pozwala na łatwe cofanie, które nie zakłóci przepływu dla innych członków zespołu.

Dopasuj harmonogram wydań

Przepływ pracy powinien uzupełniać cykl wydawania oprogramowania w Twojej firmie. Jeśli planujesz wypuszczać wiele razy dziennie, będziesz chciał, aby Twój *main*oddział był stabilny. Jeśli harmonogram wydań jest rzadszy, warto rozważyć użycie tagów Git do oznaczenia gałęzi do wersji.

Streszczenie

W tym dokumencie omówiliśmy przepływy pracy Git. Przyjrzeliśmy się dogłębnie scentralizowanemu przepływowi pracy z praktycznymi przykładami. Rozwijając scentralizowany przepływ pracy omówiliśmy dodatkowe wyspecjalizowane przepływy pracy. Niektóre kluczowe wnioski z tego dokumentu to:

- Nie ma jednego uniwersalnego przepływu pracy w Git
- Przepływ pracy powinien być prosty i zwiększać produktywność Twojego zespołu
- Twoje wymagania biznesowe powinny pomóc w kształtowaniu przepływu pracy w Git