infoShare ACADEMY

GIT

System Kontroli wersji

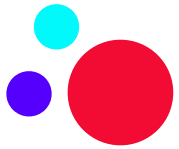
infoShare Academy

HELLO

Jakub Wojtach

Senior **full stack** developer

infoShare ACADEMY



Rozgrzewka i agenda

Zacznijmy ten dzień z przytupem!

Rozgrzewka

Woodstock

Opener

Rozgrzewka

All inclusive 5*

Camper I namioty

Rozgrzewka

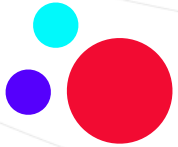
Impreza w klubie

Domówka

Rozgrzewka

Dobra książka

Polecany film

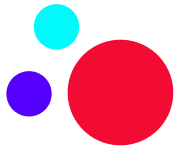


Agenda

- Podstawy teorii zdalnych repozytoriów
- Wysyłanie i pobieranie zmian z repozytorium zdalnego
- Wykorzystanie rebase
- Kolaboracja na GitHub i wykorzystanie Pull Request
- Flow w pracy zespołowej

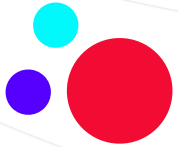
Współpraca

- Zadajemy pytania w dowolnym momencie - bezpośrednio **DM**
- Krótkie przerwy *(5 min)* co godzinę



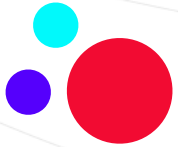
Podstawy teorii

zdalnych repozytoriów



Popularne serwisy GitHub

- GitHub to usługa hostingu umożliwiająca zarządzanie repozytoriami Git.
- Przy jego pomocy jesteś w stanie udostępnić swój kod w jednym miejscu dla wszystkich.
- Dzięki temu – w tym samym czasie – zapewniona jest możliwość aktywnej współpracy z pozostałymi członkami projektu.
- GitHub w przeciwieństwie do samego Gita działa w oparciu o chmurę.
- Każdy członek projektu może bez względu na szerokość geograficzną i sprzęt, na którym działa uzyskać zdalny dostęp do repozytorium Git (warunkiem jest dostęp do sieci).

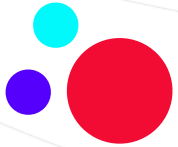


Popularne serwisy BitBucket

- Hostingowy serwis internetowy przeznaczony dla projektów programistycznych wykorzystujących system kontroli wersji Git oraz Mercurial, którego obecnym właścicielem jest firma **Atlassian**.
- Serwis umożliwia bezpłatne wykorzystanie usługi wraz z dodatkowymi płatnymi planami.
- Jest obecnie jednym z najpopularniejszych tego typu serwisów, z którego korzystają m.in. firmy Ford, PayPal, czy Starbucks.
- W kwietniu 2019 r. Atlassian ogłosił, że Bitbucket dotarł do 10 milionów zarejestrowanych użytkowników i ponad 28 milionów repozytoriów.

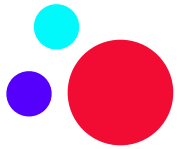
Popularne serwisy GitLab

- Hostingowy serwis internetowy przeznaczony dla projektów programistycznych oparty o system kontroli wersji Git oraz otwartoźródłowe oprogramowanie, stworzone przez Dmitrija Zaporozhets.
- Serwis umożliwia bezpłatne wykorzystanie usługi lub zainstalowanie samodzielnie zarządzanego oprogramowania wraz z opcjonalnymi, płatnymi planami.
- Usługa jest jedną z najpopularniejszych tego typu na rynku, z której korzystają takie firmy, jak IBM, Sony, NASA, Oracle, GNOME Foundation, NVIDIA, czy SpaceX[5].
- Usługa oprócz repozytoriów opartych na systemie kontroli wersji oferuje także platformę dla metodyki DevOps[6] oraz CI/CD.
- Backend GitLab został stworzony w języku Ruby (Ruby on Rails), a wygląd serwisu w języku JavaScript (Vue.js).



Dlaczego

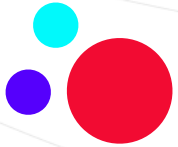
- Aby móc współpracować w jakimkolwiek projekcie opartym na git, musisz nauczyć się zarządzać zdalnymi repozytoriami.
- Zdalne repozytorium to wersja twojego projektu utrzymywana na serwerze dostępnym poprzez Internet lub inną sieć.
- Możesz mieć ich kilka, z których każde może być tylko do odczytu lub zarówno odczytu jak i zapisu.
- Współpraca w grupie zakłada zarządzanie zdalnymi repozytoriami oraz wypychanie zmian na zewnątrz i pobieranie ich w celu współdzielenia pracy/kodu.
- Zarządzanie zdalnymi repozytoriami obejmuje umiejętność dodawania zdalnych repozytoriów, usuwania ich jeśli nie są dłużej poprawne, zarządzania zdalnymi gałęziami oraz definiowania je jako śledzone lub nie, i inne.



Wysyłanie i pobieranie zmian

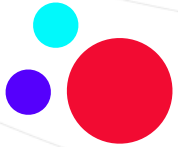
Z repozytorium zdalnego

infoShareAcademy.com



Pojęcia

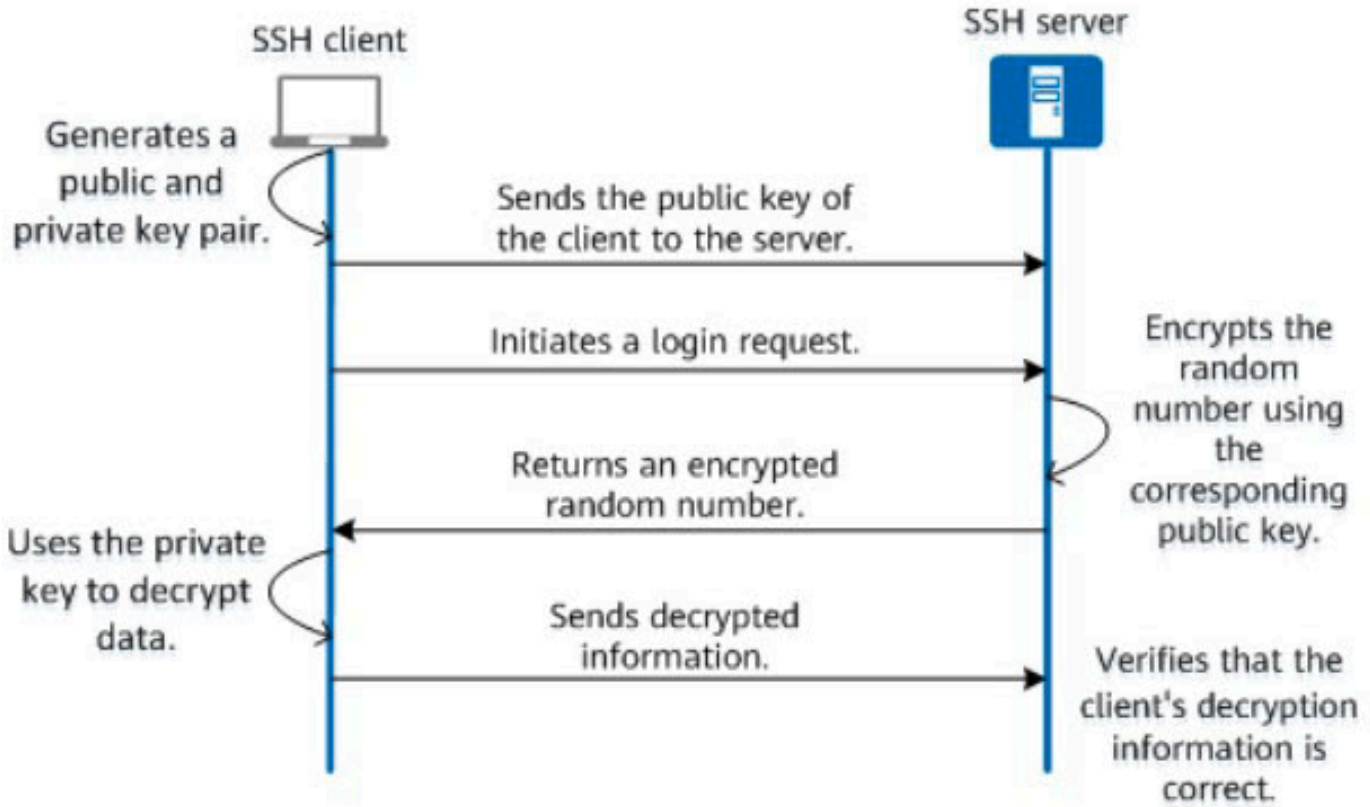
- **Tracking branch** - Checkout lokalnego brancha z remote brancha automatycznie tworzy coś, co nazywa się remote branch. Remote branch to local branche, które mają bezpośredni związek z remote branchem. Jeśli jesteś na tracking branchu i wpiszesz git push, Git automatycznie wie, do którego serwera i brancha pushować. Również uruchomienie git pull będąc na jednej z tych branchy pobiera wszystkie zdalne referencje, a następnie automatycznie łączy się w odpowiednią remote branch.
- **Remote** - połączenie do innych repozytoriów, wygodne nazwy wykorzystywane do odwołanie się do niezbyt wygodnego URL (coś jak DNS)
- **Origin** - alias w twoim systemie do konkretnego remote repo. Używając push **git push origin branchname** wypychamy do niego. Może nazywać się inaczej, jest to najczęstsza nazwa, domyślny przy użyciu git remote.



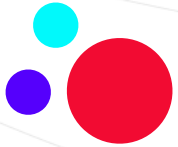
SSH

- **SSH –** secured shell, zaszyfrowane połączenie, używa portu 22 do uzyskania i autentykacji połączenia. Autentykacja kluczem publicznym urządzenia w oparciu o uprzednio wygenerowany kod SSH przypisany do konta.
- Nie trzeba wpisywać hasła, bo logujemy się w oparciu o klucz
- Można mieć wiele kluczy ssh na jednym urządzeniu - dodatkowa konfiguracja
- Jest trochę bardziej skomplikowany w konfiguracji niż HTTPS

SSH



SSH Public/Private Key based Login Process



HTTPS

- **SSH –** bezpieczny HTTP szyfrujący dane przesyłane pomiędzy klientem i serwerem. HTTPS wykorzystuje port 443, autentykacja Public/Private key.
- Trzeba wpisywać hasło za każdym razem
- Można zabezpieczyć 2FA, ale wtedy logowanie kluczem zamiast hasła

Klonowanie repozytorium

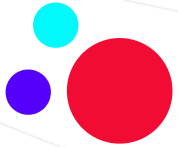
- Jeżeli chcesz uzyskać kopię istniejącego już repozytorium Gita — na przykład projektu, w którym chciałbyś zacząć się udzielać i wprowadzać własne zmiany — polecenie, którego potrzebujesz to clone.
- Po wykonaniu polecenia git clone zostanie pobrana każda rewizja, każdego pliku w historii projektu. W praktyce nawet jeśli dysk serwera zostanie uszkodzony, możesz użyć któregokolwiek z dostępnych klonów aby przywrócić serwer do stanu w jakim był w momencie klonowania.
- Repozytorium klonujesz używając polecenia git clone [URL]. Na przykład jeśli chcesz sklonować bibliotekę Rubiego do Gita o nazwie Grit, możesz to zrobić wywołując:

```
git clone https://github.com/libgit2/libgit2
```

infoShare ACADEMY

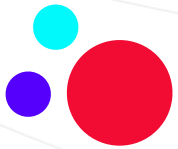
Zadanie
Indywidualne

infoShareAcademy.com



Remote

- Aby zobaczyć obecnie skonfigurowane serwery możesz uruchomić polecenie git remote.
- Pokazuje ono skrócone nazwy wszystkich określonych przez ciebie serwerów. Jeśli sklonowałeś swoje repozytorium, powinieneś przynajmniej zobaczyć origin (źródło) - nazwa domyślna którą Git nadaje serwerowi z którego klonujesz projekt.
- Zobaczcie teraz swój remote na uprzednio pobranym repozytorium



Wysyłanie zmian

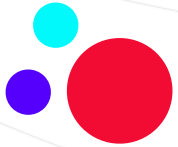
- Jeśli doszedłeś z projektem do tego przyjemnego momentu, kiedy możesz i chcesz już podzielić się swoją pracą z innymi, wystarczy, że wypchniesz swoje zmiany na zewnątrz.
- Służące do tego polecenie jest proste git push [nazwa-zdalnego-repo] [nazwa-gałęzi].
- Jeśli chcesz wypchnąć gałąź główną master na oryginalny serwer źródłowy origin (ponownie, klonowanie ustawia obie te nazwy - master i origin - domyślnie i automatycznie), możesz uruchomić następujące polecenie:

```
$ git push origin master
```

infoShare ACADEMY

Zadanie
Indywidualne

infoShareAcademy.com



Pobieranie zmian pull

- **Git pull** służy do ściągania i pobierania zawartości ze zdalnego repozytorium oraz natychmiastowej aktualizacji lokalnego repozytorium zgodnie z tą zawartością.
- Scalanie zdalnych zmian nadrzędnych z lokalnym repozytorium jest powszechnym zadaniem w kolaboracyjnych przepływach pracy opartych na systemie Git.
- Polecenie **git pull** jest właściwie kombinacją dwóch innych poleceń, **git fetch**, a następnie **git merge**. W pierwszym kroku operacji **git pull** wykonane zostanie polecenie **git fetch** w zakresie ograniczonym do lokalnej gałęzi wskazywanej przez wskaźnik **HEAD**. Po pobraniu zawartości polecenie git pull spowoduje przejście do przepływu pracy **scalania**. Zostanie utworzony nowy commit scalenia, a wskaźnik **HEAD** zostanie zaktualizowany tak, aby wskazywał na **nowy commit**.

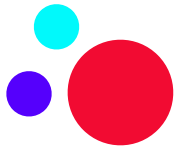
Pobieranie zmian fetch

- Polecenie **git fetch** pobiera commity, pliki i refy ze zdalnego repozytorium do twojego lokalnego repo.
- Pobieranie jest tym, co robisz, gdy chcesz zobaczyć, nad czym wszyscy inni pracowali.
- Git izoluje pobraną zawartość od istniejącej zawartości lokalnej; nie ma absolutnie żadnego wpływu na twoją lokalną pracę nad rozwojem.
- Pobrana zawartość musi być jawnie sprawdzona za pomocą polecenia git checkout. To sprawia, że **fetch** jest bezpiecznym sposobem na przeglądanie commitów przed zintegrowaniem ich z lokalnym repozytorium.
- Często wykorzystywany do pobrania zmian o nowych commitach i **branchach**

infoShare ACADEMY

Zadanie
Indywidualne

infoShareAcademy.com



Wykorzystanie rebase

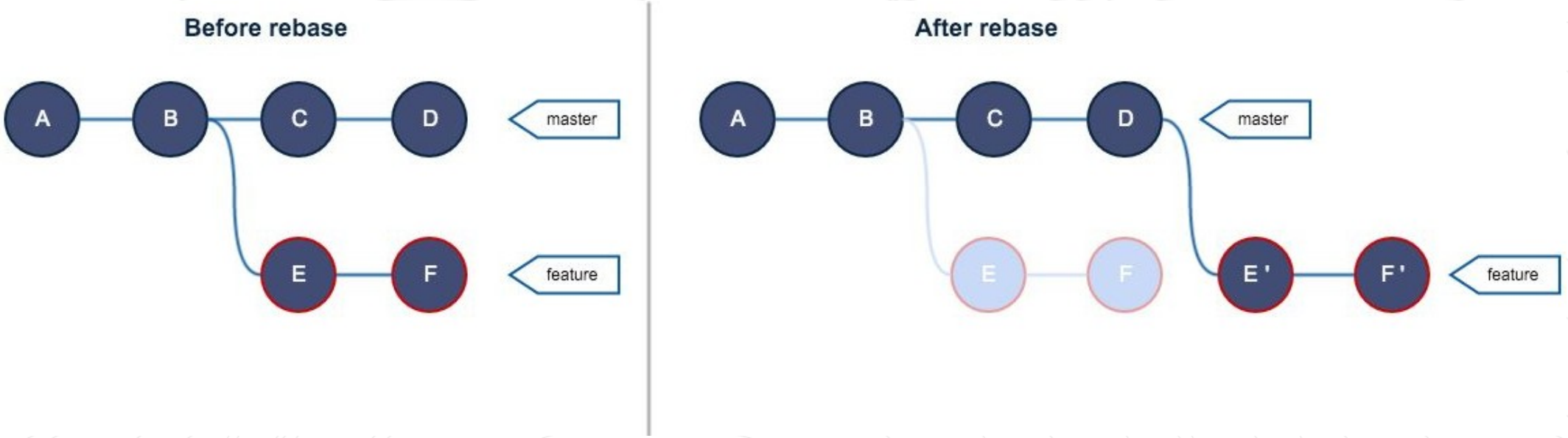
Czyli ładniejsza historia

infoShareAcademy.com

infoShare ACADEMY

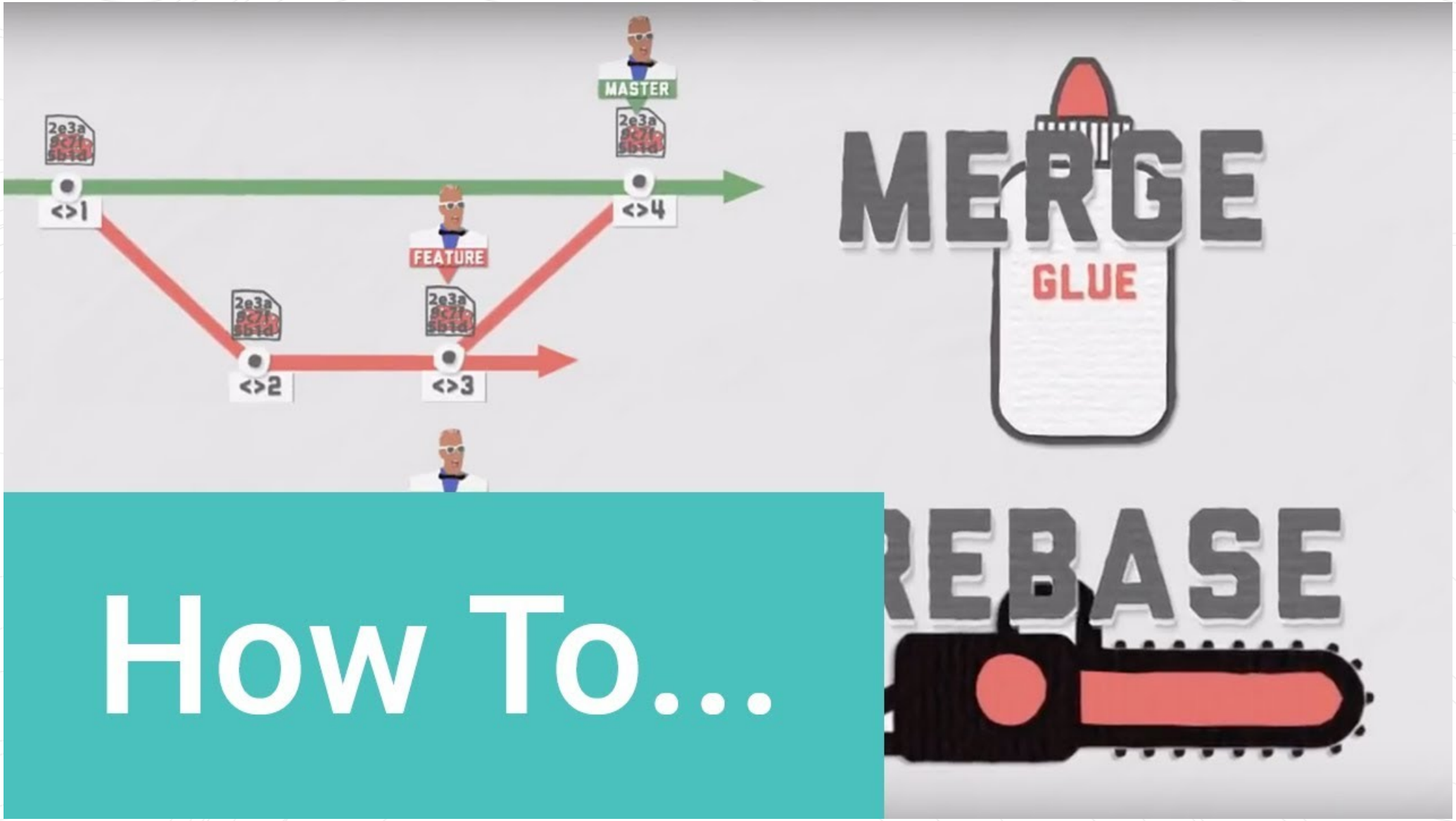
Rebase

- Wypłaszcza historię commitów
- Nanosi zmiany w kolejności commitów
- Możliwość zmiany historii -> stosować **tylko lokalnie!!**

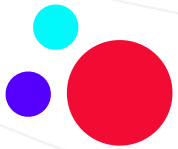


infoShare ACADEMY

Rebase & Merge



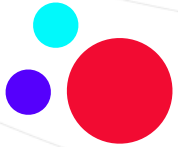
infoShare ACADEMY



Rebase scenariusz 1

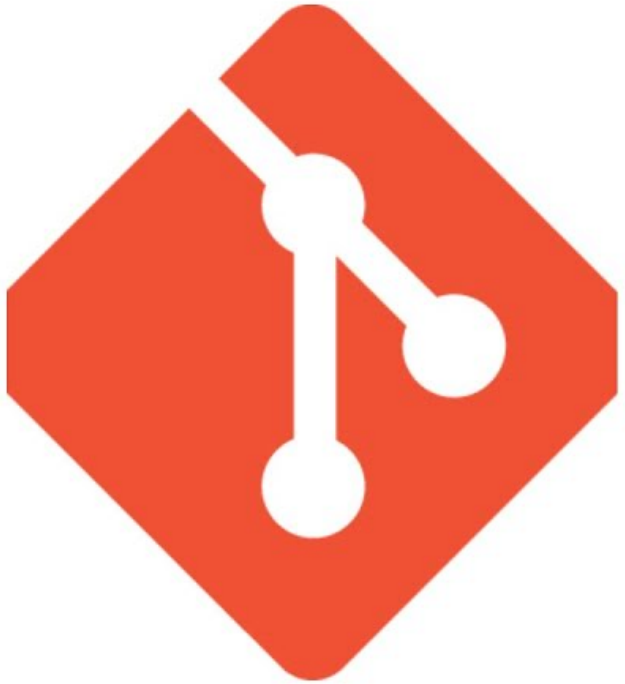
- Unikamy 3 way merge (różny stan na masterze i własnym branchu)
- Rebase gałęzi master do własnej gałęzi
- Po udanym **rebase** wykonujemy merge fast-forward do gałęzi master
- Dzięki temu uzyskujemy liniową historię
- Cały proces opisany w linku:

https://medium.com/@catalinaturlea/clean-git-history-a-step-by-step-guide-eefc0ad8696d



Rebase scenariusz 2

- Tworzymy 3 commity, wypychamy je do repo (nasz branch) (dowolne wiadomości)
- Używamy **git rebase -i HEAD~3**
- Squashujemy commity
- Tworzymy ładną wiadomość do commita
- Wypychamy zmiany z fortem

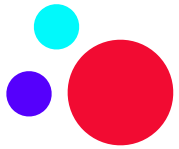


Resolve merge conflict during Git Rebase

infoShare ACADEMY

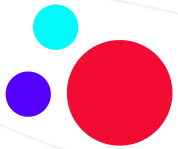
Zadanie
Indywidualne

infoShareAcademy.com



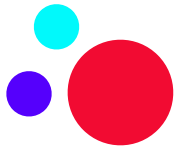
Kolaboracja na GitHub

infoShareAcademy.com



Strategie

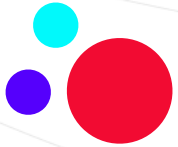
- https://github.com/jakubwojtach/demo-repo/pull/1



Flow

W pracy zespołowej

infoShareAcademy.com

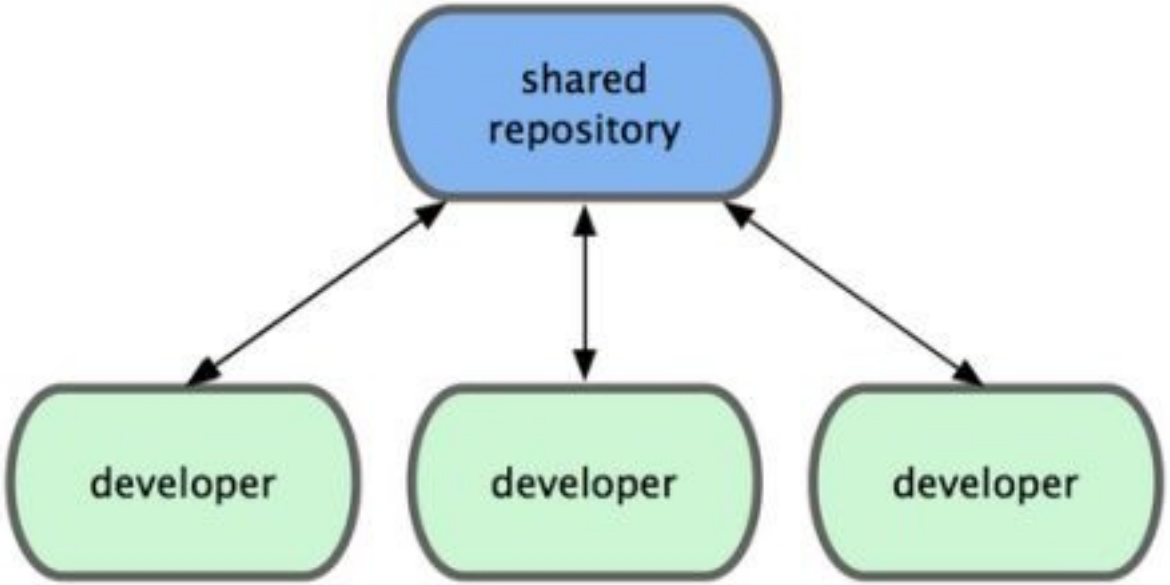


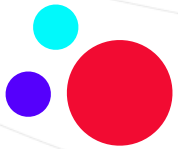
Workflow

- GIT nie narzuca sposobu pracy - to tylko narzędzie
- Istnieje kilka popularnych praktyk pracy z GIT
  - Centralized workflow
  - Feature branch workflow
  - Git-flow workflow

Centralized workflow

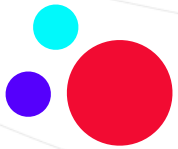
- Praca na jednym branchu (podobnie jak SVN)
- Brak podziału na kod rozwojowy i produkcyjny
- Trudna koordynacja przy wielu osobach
- **KONFLIKTY!**





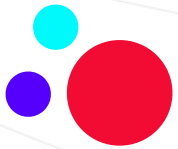
Feature-branch workflow

- Naturalny podział zadań
- Każdy feature/bug ma swojego brancha
- Rozdzielenie kodu rozwojowego dla ‘ficzerów’
- Wiele osób = wiele branchy
- Weryfikacja kodu przed włączeniem do mastera
- Brak podziału na kod rozwojowy i produkcyjny



GIT-flow

- Rozwinięcie koncepcji ‘feature-branch’
- Oddzielne branche na releasy
- Osobny branch na kod produkcyjny (master)
- Osobny branch na kod rozwojowy (develop)
- Pełna informacja jaki kod w jakiej fazie projektu
- Nazwy branchy wg konwencji -> np. nr zadania w JIRA
- Obecny standard pracy z GIT



GIT-flow

- **MASTER** - główna gałąź produkcyjna

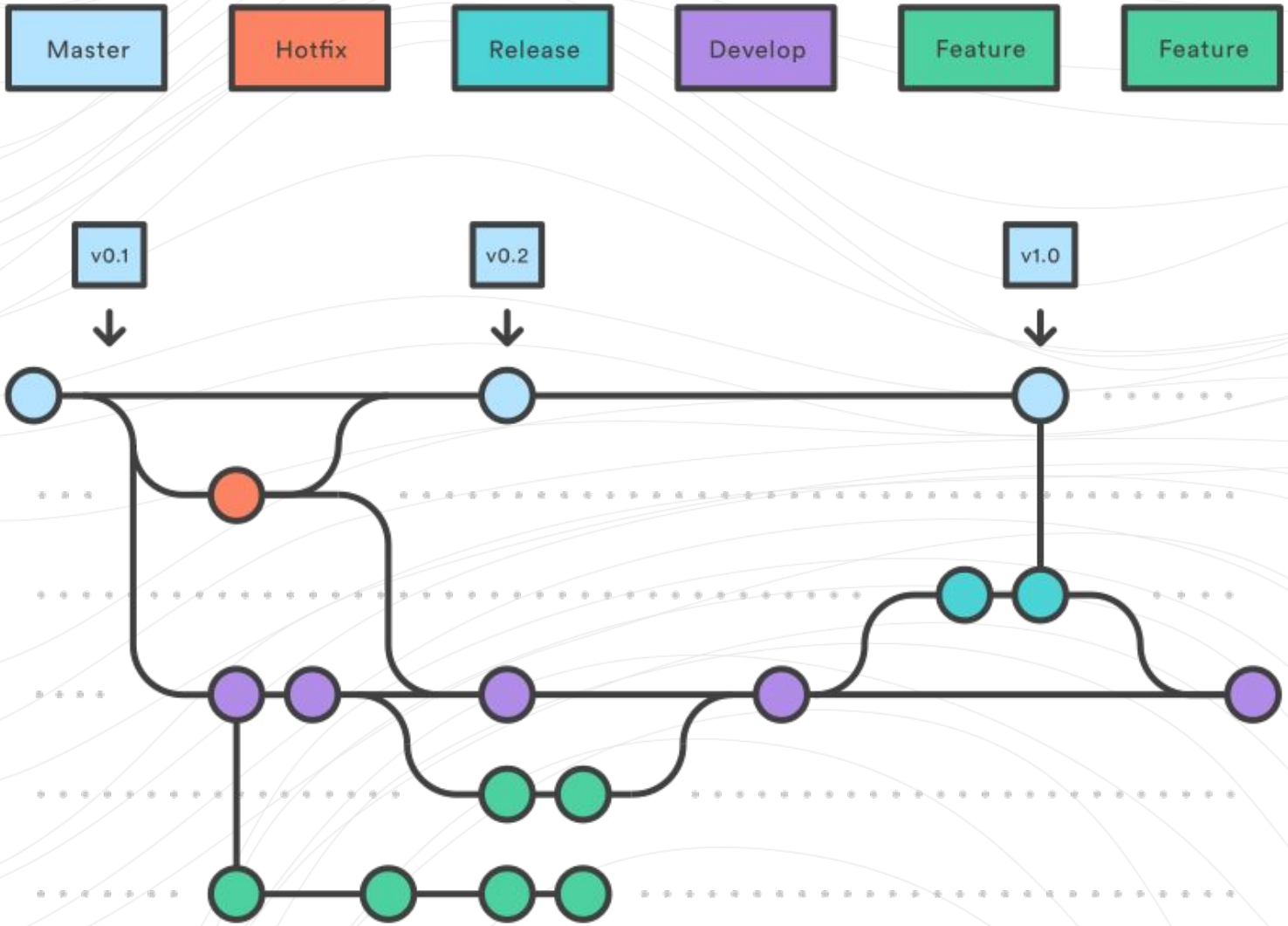
kod z tej gałęzi znajduje się na serwerze produkcyjnym i korzystają z niego klienci

- **DEVELOP** - główna gałąź rozwojowa

tutaj przygotowany kod do kolejnych wydań produkcyjnych wersja w develop powinna być w pełni działająca (możliwe błędy)

- **FEATURE_BRANCH** - branch dla konkretnej funkcjonalności po zakończeniu prac zostanie włączony do develop

GIT-flow



Pytania

infoShareAcademy.com

Dziękuję za uwagę

Jakub Wojtach