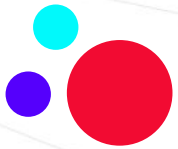
infoShare ACADEMY

GIT - Part 1

HELLO

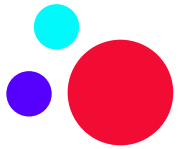
Błażej Laskowski

InfoShare ACADEMY



Agenda

1. Systemy Kontroli Wersji
2. Podstawowe komendy
3. Branching
4. .gitignore i .gitkeep
5. Dobre praktyki

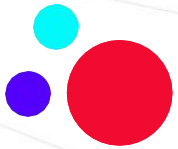


Systemy kontroli wersji

infoShareAcademy.com

Problemy przy pracy w projektach

- Projekty ciągle ewoluują
- Ludzie popełniają błędy - jak w łatwy sposób cofnąć zmiany?
- Co zrobić by członkowie zespołu nie przeszkadzali sobie wzajemnie?
- Jak uniezależnić się od jednej maszyny?
- Jak w prosty sposób udostępnić projekt nowej osobie w zespole?



System kontroli wersji

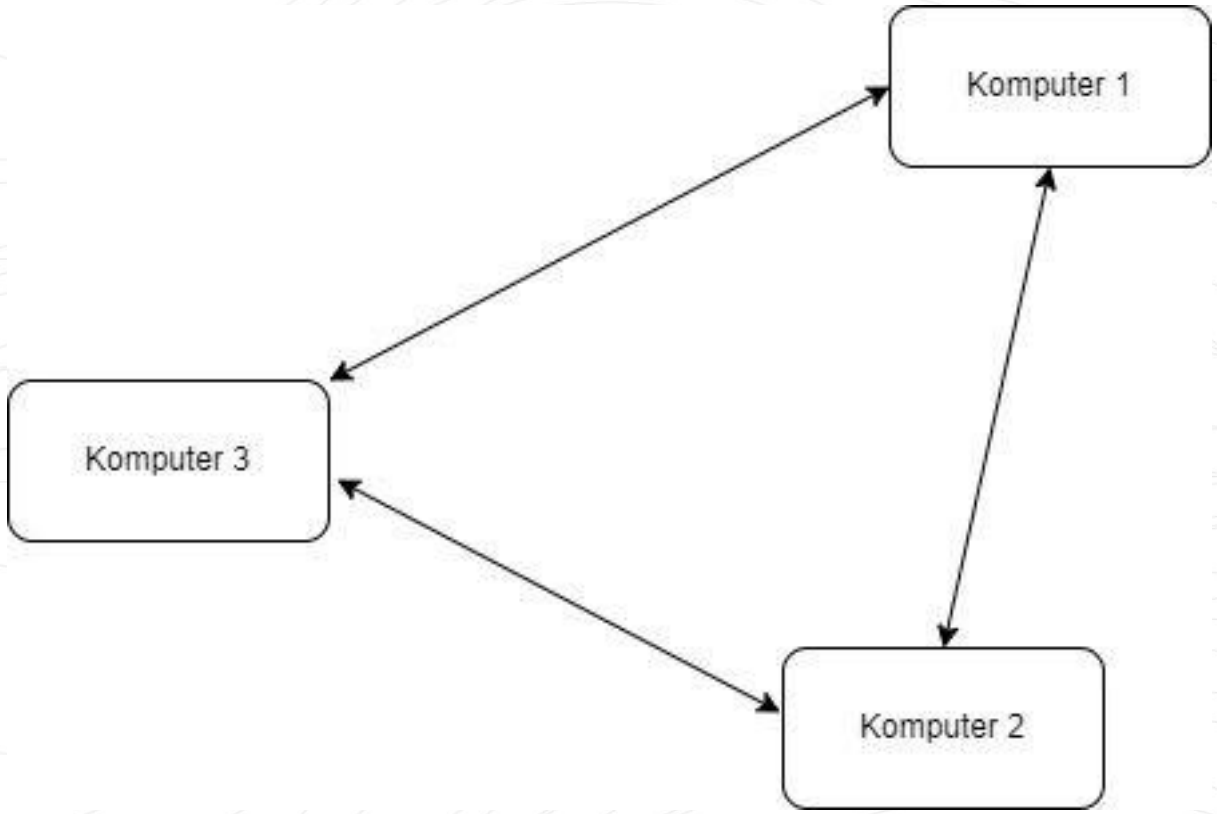
Systemem kontroli wersji nazywamy system służący do zarządzania zmianami w dokumentach, programach, stronach internetowych oraz innych zbiorach danych.

Istnieje wiele różnych systemów kontroli wersji m.in. Git, SVN, Perforce.

GIT

Git jest systemem kontroli wersji udostępnionym na licencji open-source(kod systemu jest dostępny publicznie oraz bezpłatnie i każdy może go zmieniać na swoje potrzeby oraz proponować swoje zmiany w głównym katalogu).

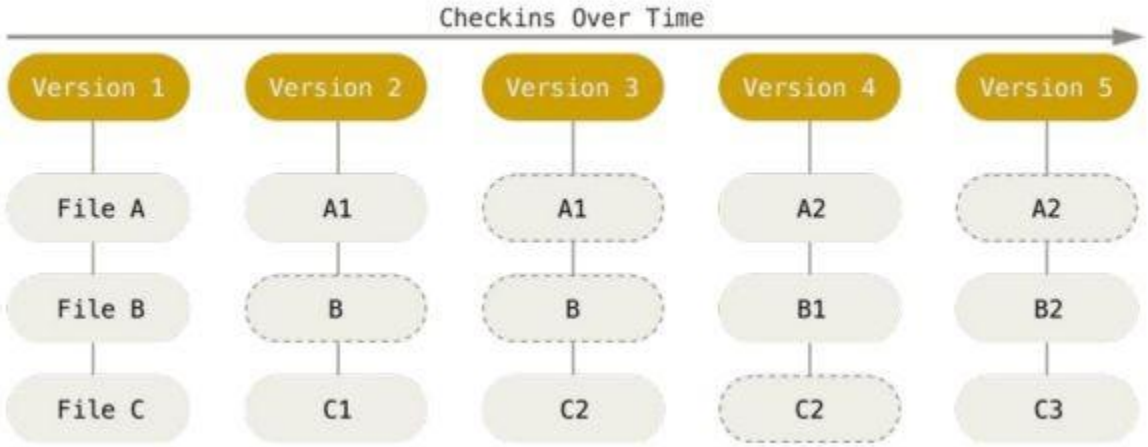
GIT

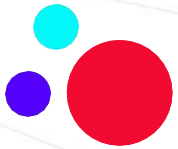


Główną zaletą Git-a jest jego rozproszona architektura – kopia repozytorium danych znajduje na każdym komputerze, na które zostało ściągnięte a nie na centralnym serwerze, dzięki czemu użytkownicy mają dostęp nawet pomimo awarii serwera. Zazwyczaj jednak wszyscy synchronizują się tylko z jednym konkretnym hostem.

GIT

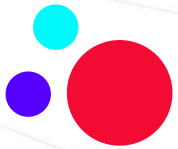
- Kopia lokalna repozytorium znajduje się w podfolderze .git - w tym podfolderze znajdują się wszystkie migawki określonego obszaru plików
- Każdy commit tworzy migawkę
- Prawie wszystkie operacje są lokalne (poza m.in. pull i push)
- Migawki zapisują kompletny stan śledzonego przez system obszaru w systemie plików
- Jeśli plik od ostatniej migawki nie został zmodyfikowany zamiast duplikowania pliku zapisywana jest referencja(wskaźnik) do jego obszaru w poprzedniej migawce





Commit zawiera informacje o:

- jakie pliki zostały zmienione
- jak pliki zostały zmienione
- autorze
- komentarz o zmianach
- wskaźnik na poprzedni commit

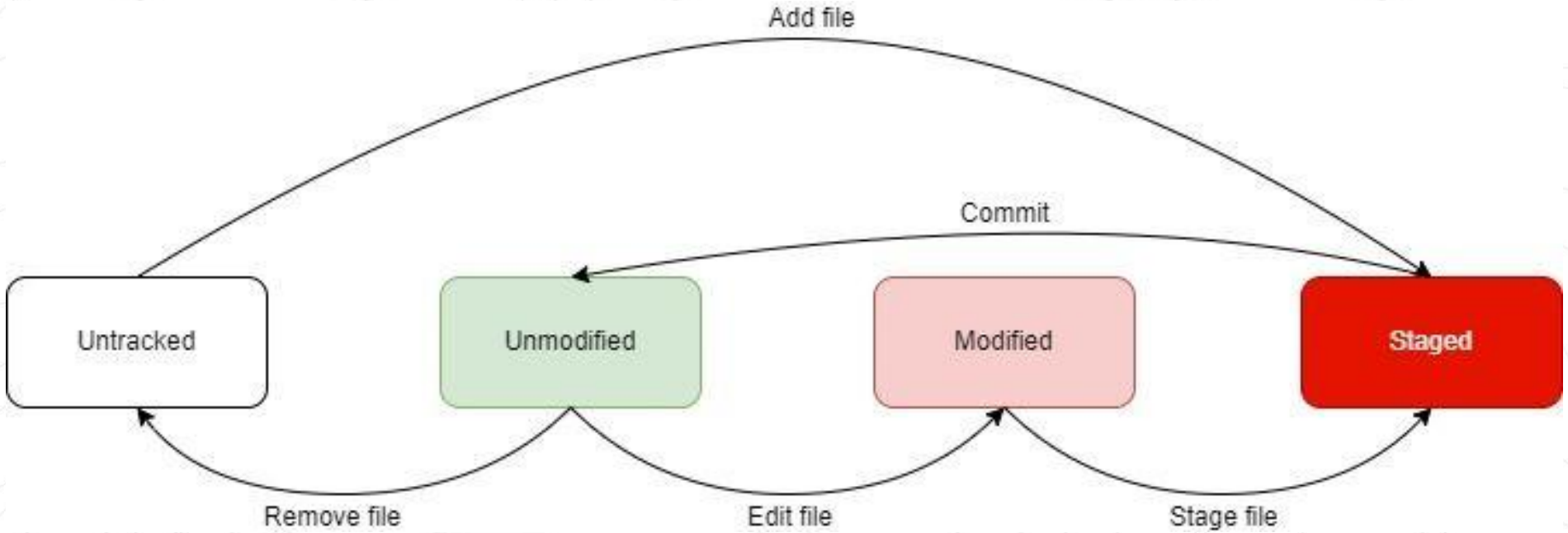


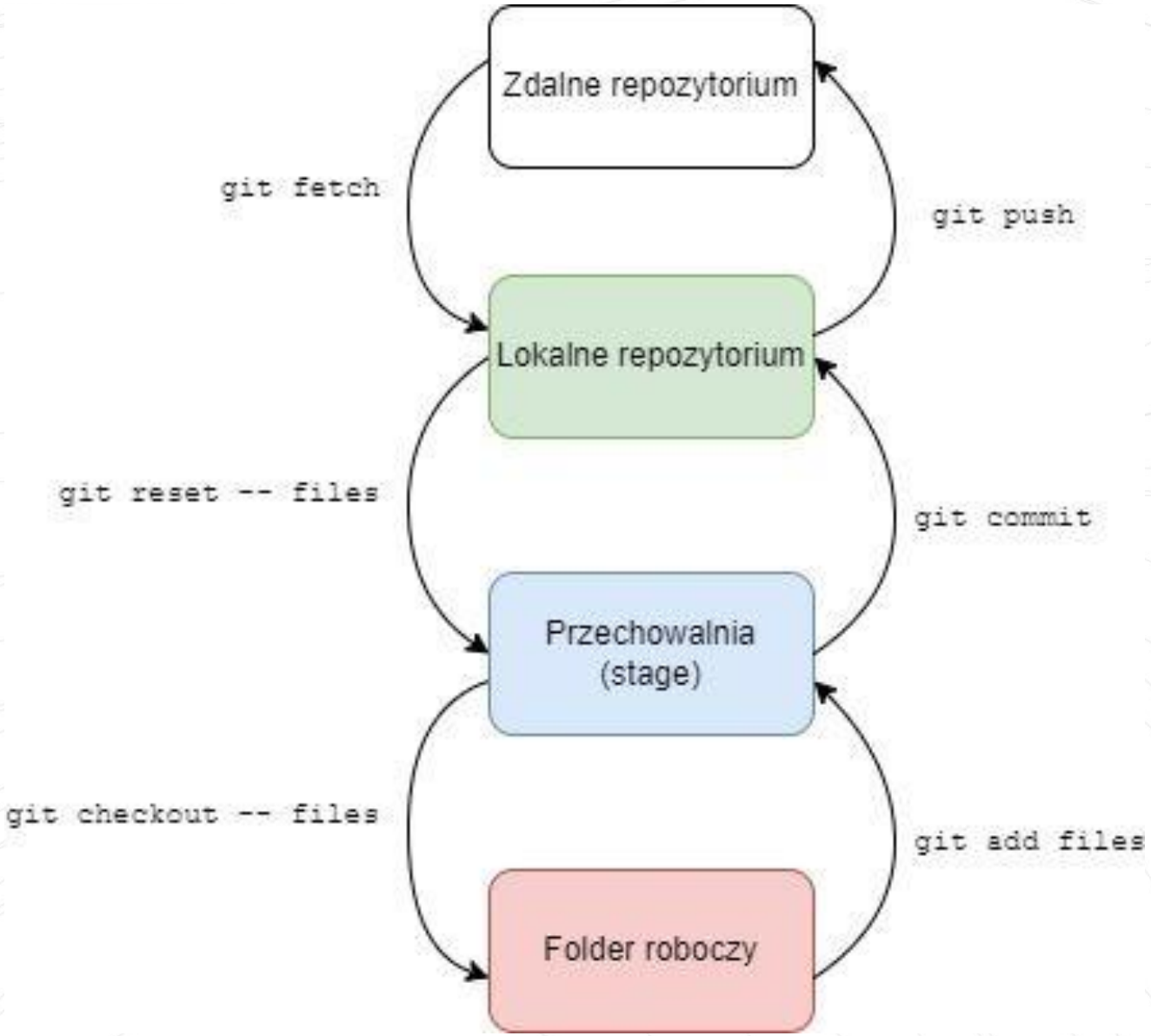
Repozytorium

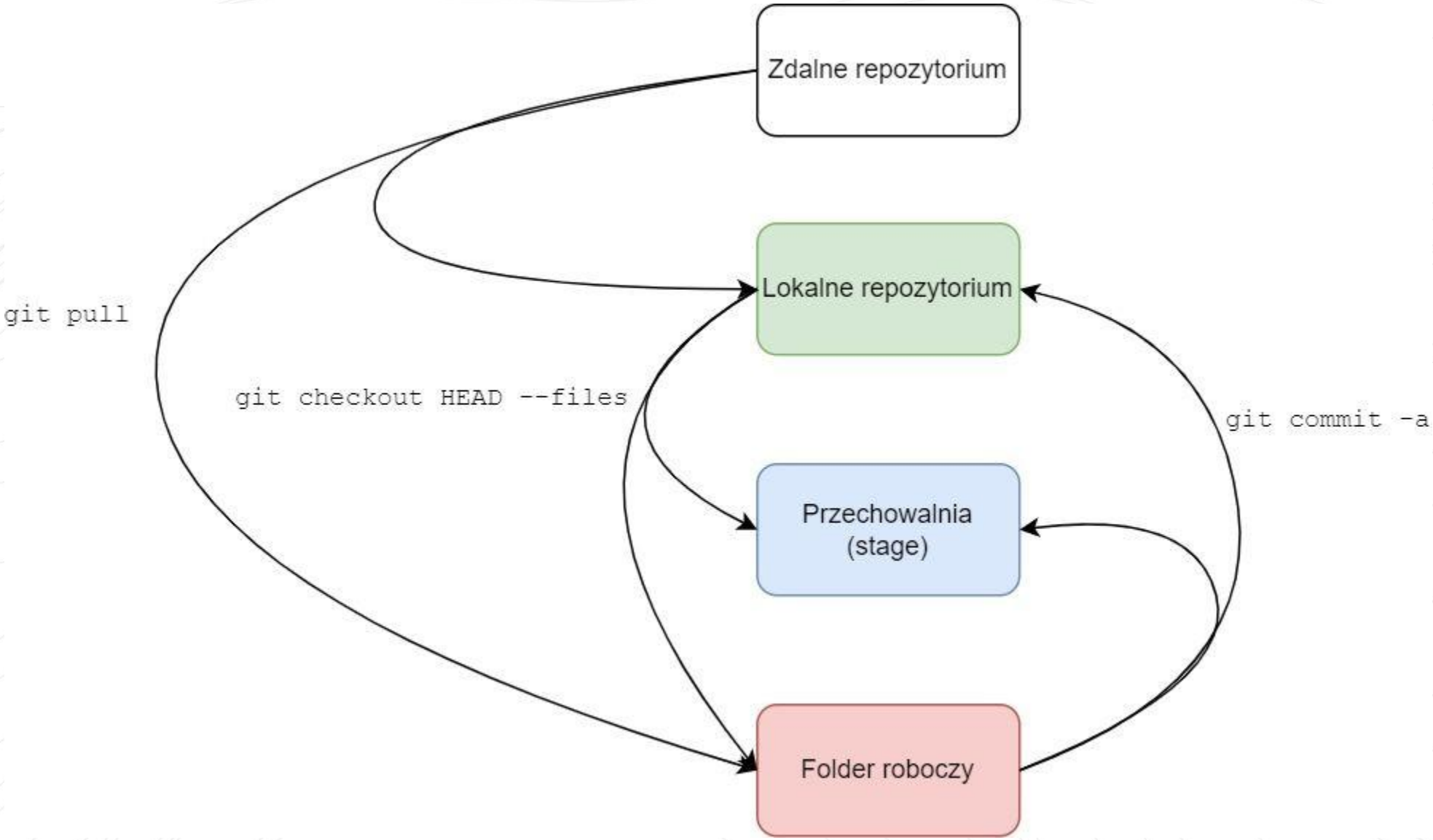
Miejsce uporządkowanego przechowywania dokumentów, z których wszystkie przeznaczone są do udostępniania.

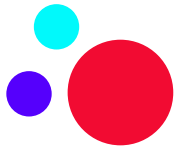
Mówiąc repozytorium w kontekście systemu GIT zazwyczaj mówimy nie tylko o miejscu w którym przechowujemy pliki ale o zbiorze informacji o zmianach w dokumentach.

Stany pliku w GIT




Zdalne repozytorium
git fetch
git push
Lokalne repozytorium
git reset -- files
git commit
Przechowalnia
(stage)
git checkout -- files
git add files
Folder roboczy


Zdalne repozytorium
Lokalne repozytorium
Przechowalnia
(stage)
Folder roboczy
git pull
git checkout HEAD --files
git commit -a



Podstawowe komendy

infoShareAcademy.com

Git clone

Pobiera repozytorium z określonego miejsca oraz kopiuje je do folderu na twoim komputerze.

```
git clone <adres do zdalnego
repozytorium>
```

Git init

Tworzy nowe repozytorium z folderu z którego komenda została wykonana.

```
git
init
```

Git add

Kopiuje pliki z folderu roboczego w ich aktualnym stanie do przechowalni (Stage).

```
git add * (doda wszystkie
pliki)
```

Git commit

Zapisuje przechowalnie(stage) jako nowy commit i dodaje go do naszego lokalnego repozytorium.

```
git commit -m <treść
komentarza>
```

Git push

Przesyła wszystkie commity z naszej lokalnej kopii repozytorium do repozytorium znajdującego się na zdalnym serwerze(z którego sklonowaliśmy repozytorium).

```
git
push
```

Git pull

Pobiera i kopiuje zmiany z repozytorium znajdującego się na zdalnym serwerze i stosuje te zmiany w naszym katalogu roboczym.

```
git
pull
```

Git revert

Cofa zmiany ze wskazanego commita i zapisuje zmiany w repozytorium jako nowy commit.

```
git revert <identyfikator
commita>
```

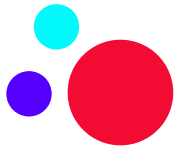
Git reset

Cofa aktualną gałąź do wybranego commita oraz w zależności od użytej flagi stosuje zmiany w katalogu roboczym i/lub w przechowalni.

Flaga --hard skopiuje i zastosuje zmiany do wybranego commita w katalogu roboczym co oznacza, że usunie wszelkie zmiany zastosowane po wskazanym commicie, natomiast flaga --soft spowoduje że zmiany nie pojawią się w przechowalni co oznacza że pozostawione tam zmiany będą gotowe do zatwierdzenia.

```
git reset --hard <identyfikator
commita>
```

When you messed up a commit and try to undo your changes

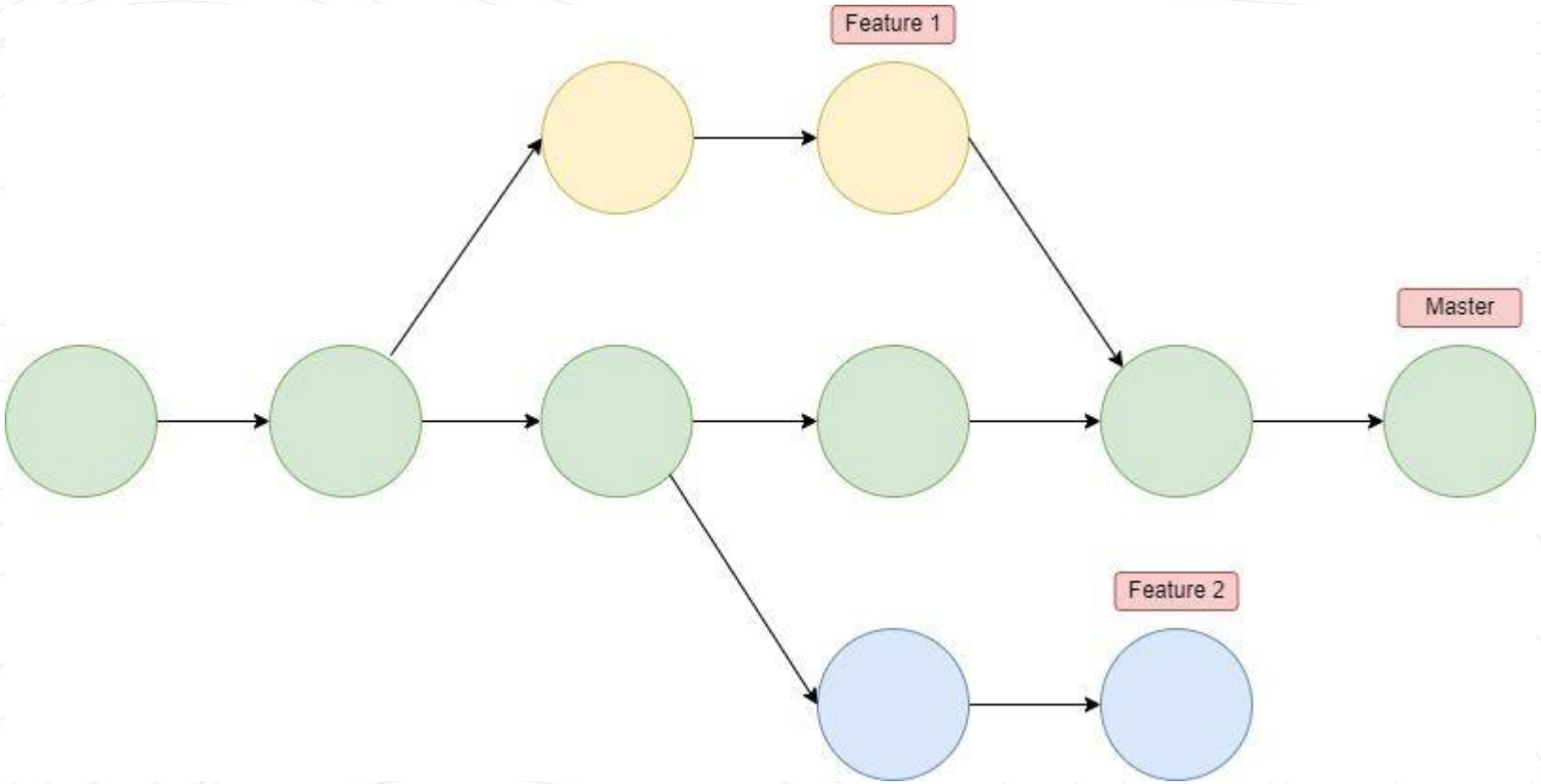


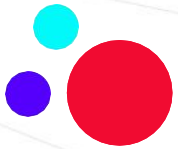
Branching

infoShareAcademy.com

Branche

- Wskaźnik na commit, który zawiera informację o obecnym statusie plików
- Pozwala tworzyć alternatywne ścieżki zmian w projekcie
- Może zawsze być utworzony/usunięty/zmodyfikowany
- Każde repozytorium posiada co najmniej jeden branch (pierwszy domyślny branch z konwencji to master/main)





HEAD

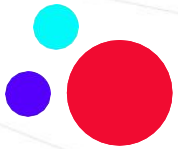
Wskaźnik na gałąź na której obecnie się znajdujemy.

infoShare ACADEMY

Git branch

Komenda pozwala podejrzeć gałęzie znajdujące się w naszym lokalnym repozytorium oraz tworzyć nowe gałęzie. Nowo utworzona gałąź będzie wskazywać na commit na który wskazuje nasz HEAD.

```
git branch <nazwa nowego
brancha>
```



Git checkout

Przełącza HEAD na wybraną gałąź lub commit oraz kopiuje pliki z repozytorium do przechowalni lub katalogu roboczego. Używając flagi -b dodatkowo tworzy nową gałąź.

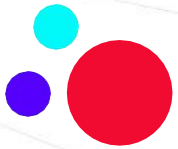
```
git checkout <nazwa istniejącego
brancha>  git checkout -b <nazwa nowego
brancha>
```

DETACHED HEAD

Mówimy, że wskaźnik HEAD jest odłączony

w momencie gdy nie wskazuje na żadną gałąź

a na konkretny commit.

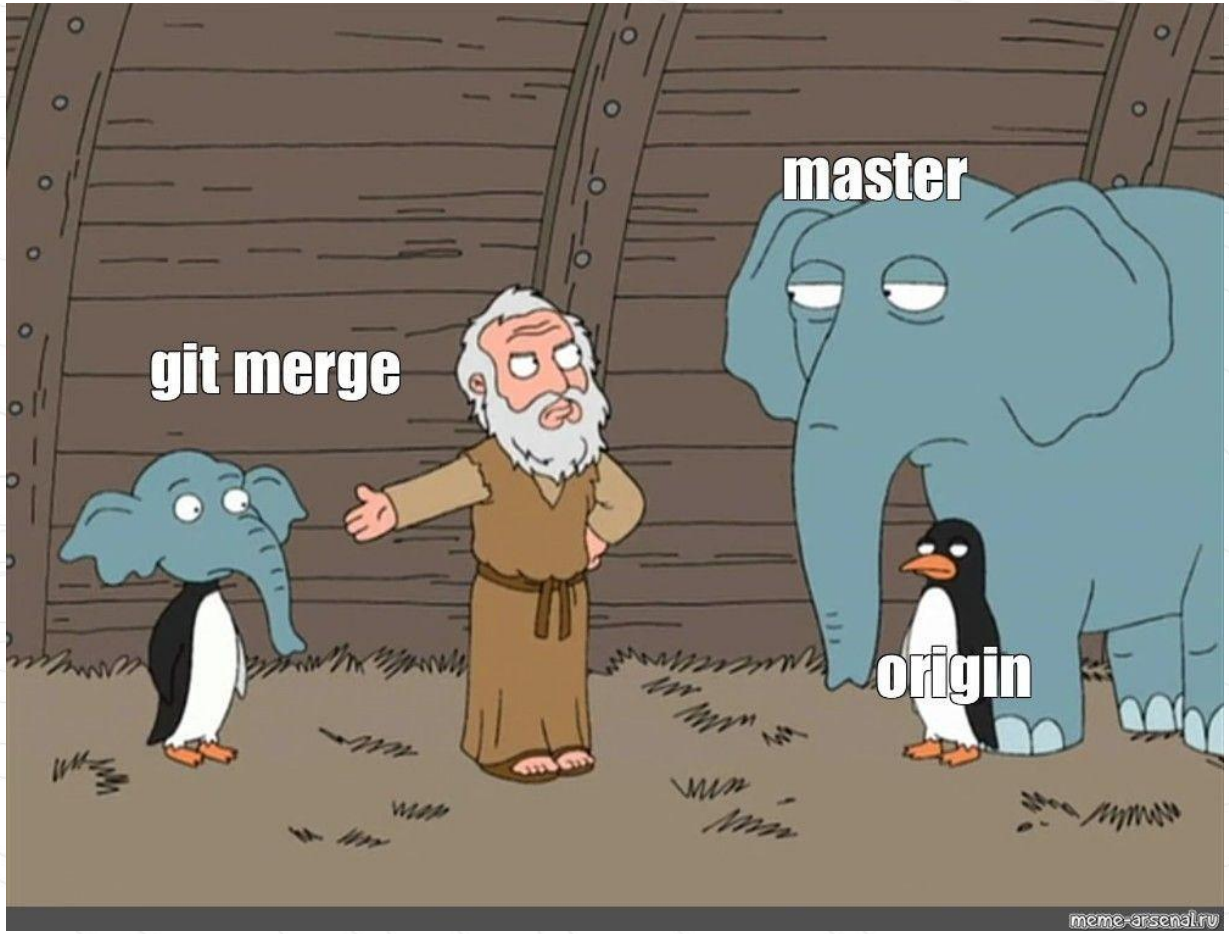


Commity wprowadzone na odłączonym HEAD przepadną jeśli przełączymy na HEAD na inną gałąź lub commit dlatego przedtem należy utworzyć nową gałąź która będzie wskazywać na ostatni wprowadzony commit.

Git merge

Łączy dwie gałęzie w jedną. Merge zawsze tworzy nowego commita zawierającego zmiany między gałęziami.

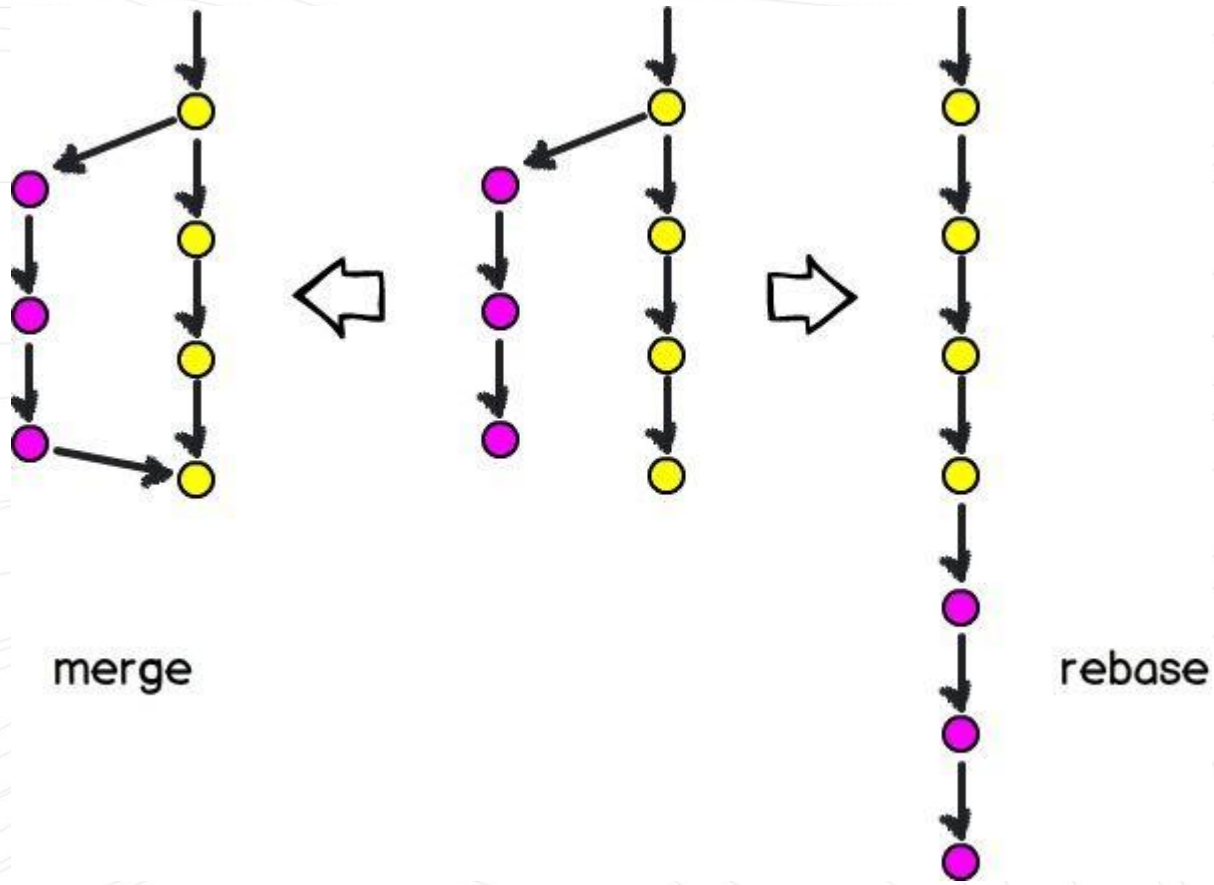
```
git merge <gałąź którą chcemy połączyć>
```

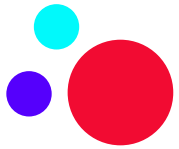


Git rebase

Łączy dwie gałęzie w jedną, jednak commity nie są w kolejności chronologicznej. Tylko commity z aktualnej gałęzi są kopiowane na koniec innej gałęzi przy zachowanej kolejności ich wprowadzenia.

```
git rebase <gałąź którą chcemy
połączyć>
```


merge
rebase



.gitignore i .gitkeep

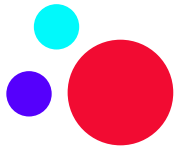
infoShareAcademy.com

Plik .gitignore

Jest to plik, który zawiera informacje, które pliki znajdujące się w folderze roboczym repozytorium mają być pomijane przy zatwierdzaniu i przesyłaniu zmian.

Plik .gitkeep

Pusty plik który stosujemy gdy chcemy wersjonować w repozytorium pusty folder lub chcemy wersjonować folder ale z jakiegoś powodu nie chcemy wersjonować jego zawartości.



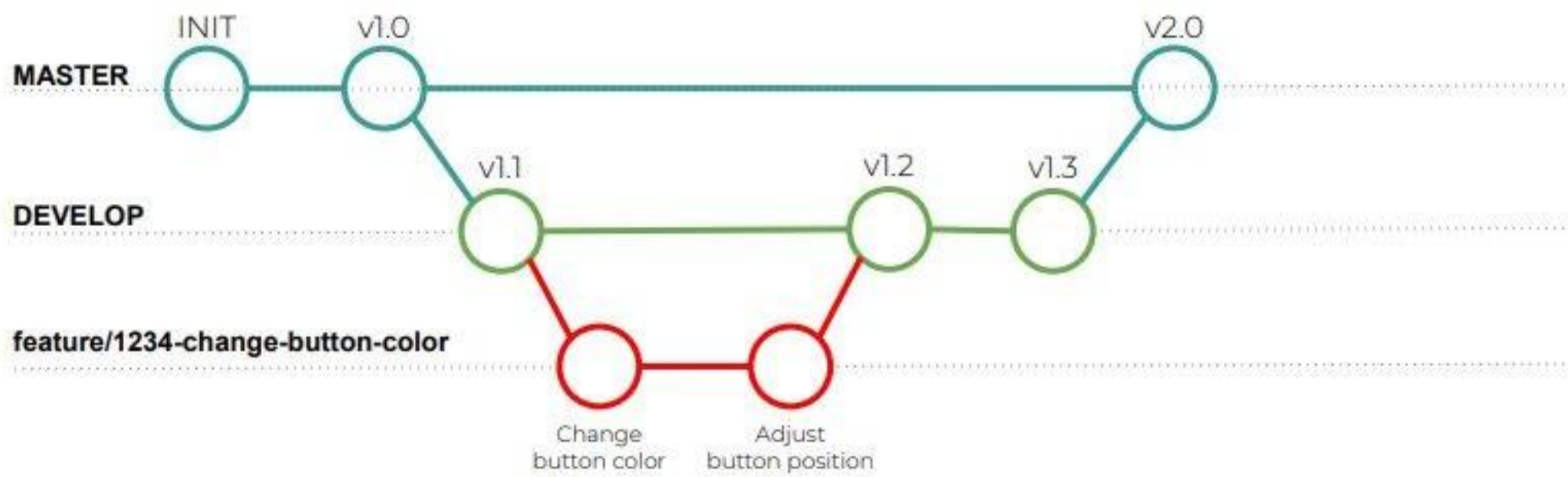
Dobre praktyki

infoShareAcademy.com

Dobre praktyki

- Opis commita powinien w jasny sposób przedstawiać co zostało zmienione i dlatego, w komentarzu zawsze powinien znajdować się identyfikator zadania jakiego dotyczyły wprowadzone zmiany
- Dla każdego nowego zadania tworzymy osobny branch zazwyczaj z gałęzi develop/dev
- Staramy się pisać zwięźle informacje na temat zmian - nikomu nie chce się czytać kilku akapitów dotyczących jednej zmiany w kodzie programu
- Staramy się nie pisać komentarzy typu “build fix”, “added unit tests”, “fixed typo” itp.

GIT flow



A co z okienkowymi klientami GIT-a?

Zadanie

1. Stwórz na githubie publiczne repozytorium
2. Sklonuj repozytorium i stwórz w katalogu roboczym nową solucję w Visual Studio
3. Scommituj i wypchnij do gałęzi master nową solucję (pamiętaj o .gitignore)
4. Stwórz nową gałąź o nazwie develop oraz new-amazing-feature
5. Na gałęzi new-amazing-feature commituj zmianę w solucji (może to być stworzenie nowej klasy)
6. Wykonaj merge z new-amazing-feature do develop


ONE DOES NOT SIMPLY
UNDERSTAND GIT
imgflip.com

Dziękuję za uwagę!