GIT

systemy kontroli wersji

Cześć!

Jestem Michał Ignaciuk

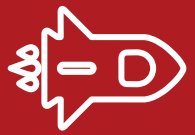
Java Web Applications Developer

michal.ignaciuk@gmail.com

Czym będziemy się zajmować?

- Systemy kontroli wersji - historia GIT
- Zasada działania GIT
- Podstawowe operacje
- Branche
- GIT workflow

Systemy kontroli wersji



Od Worda do GITa

Systemy kontroli wersji
po co nam system kontroli wersji

- Śledzenie zmian w projekcie,
- Możliwość cofania się do dowolnego momentu w historii,
- Możliwość współpracy wielu osób na tych samych plikach,
- Możliwość sprawdzenia kto i kiedy wprowadził zmiany,
- Backup danych.

Systemy kontroli wersji
różne rodzaje systemów kontroli wersji

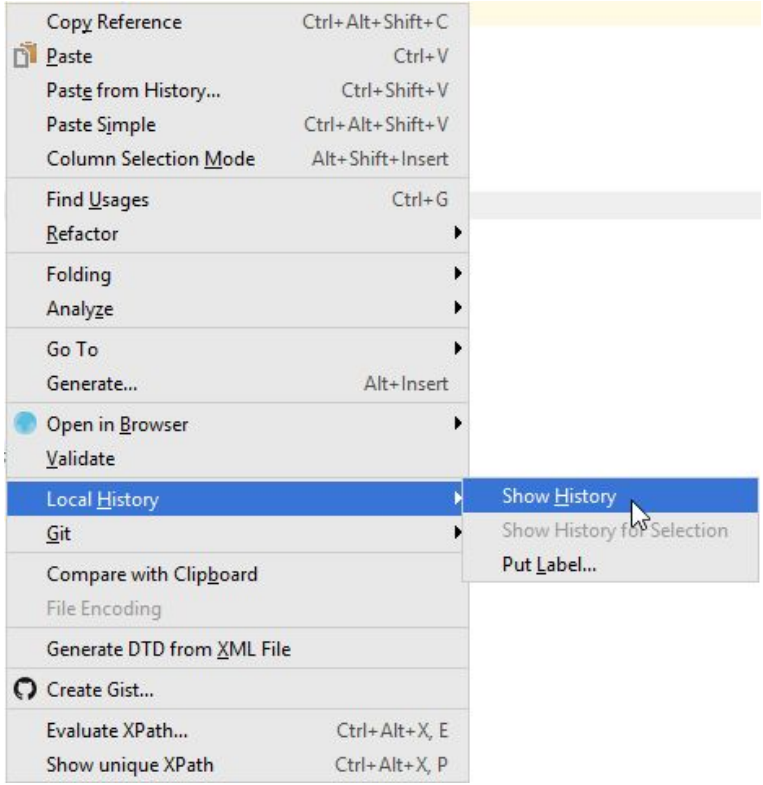
- Lokalne
  - Działają tylko na lokalnej maszynie,
  - np. wbudowane w Idea "Local History".

- Scentralizowane
  - Działają w architekturze klient-serwer,
  - np. Subversion (SVN) albo Concurrent Versions System (CVS).

- Rozproszone
  - Działają w architekturze peer-to-peer (P2P),
  - np. GIT albo Mercurial

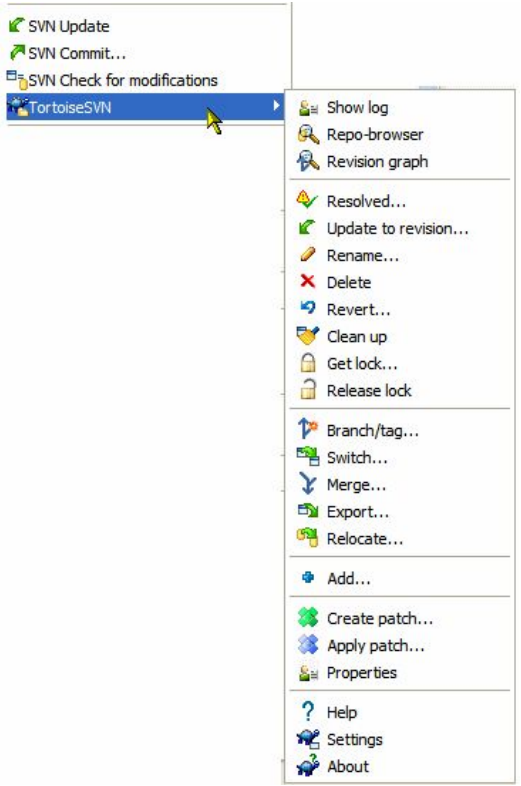
Systemy kontroli wersji
systemy lokalne

- Dane nie są nigdzie backupowane - możliwa utrata danych,
- Nie wspiera pracy grupowej,
- Przydatne jeśli np. przez przypadek usuniemy plik albo chcemy cofnąć lokalne zmiany.



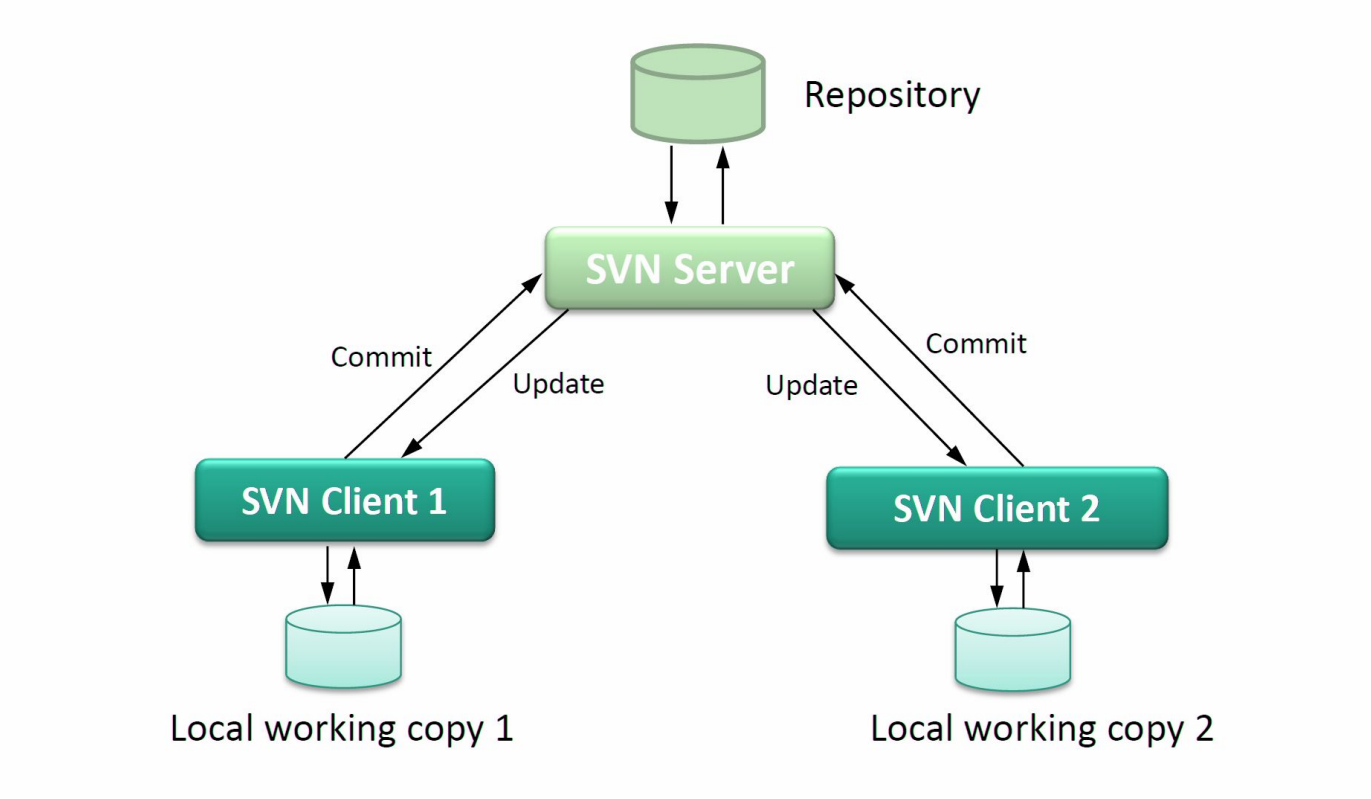
Systemy kontroli wersji
systemy scentralizowane

- Wszystkie dane są przechowywane na serwerze - mniejsze (!) prawdopodobieństwo utraty danych,
- Lokalna praca tylko na określonej wersji,
- Historia przechowywana tylko w repozytorium na serwerze.



Systemy kontroli wersji

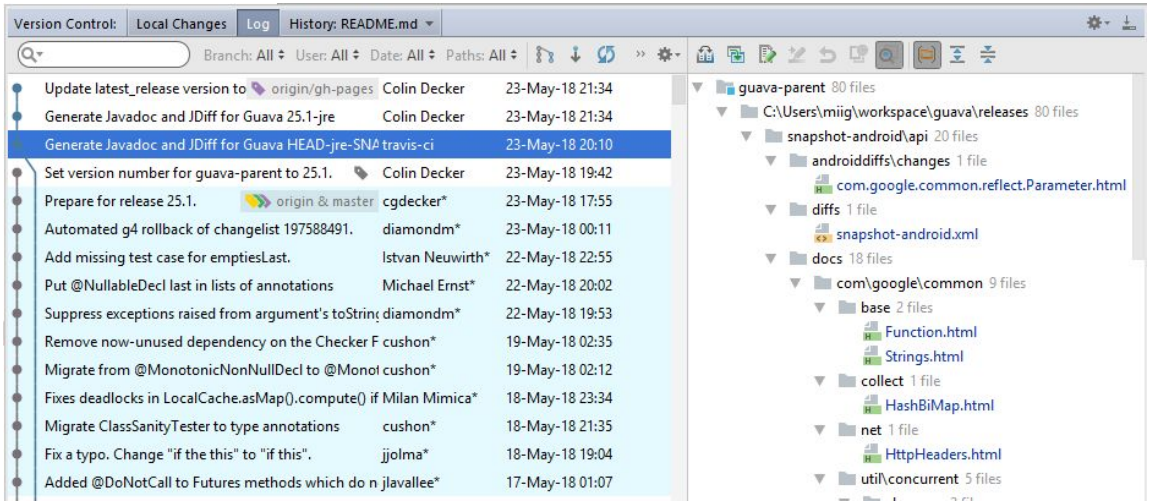
systemy scentralizowane - architektura SVN



źródło: https://commons.wikimedia.org/wiki/File:SVN_Server_Client_Structure.png

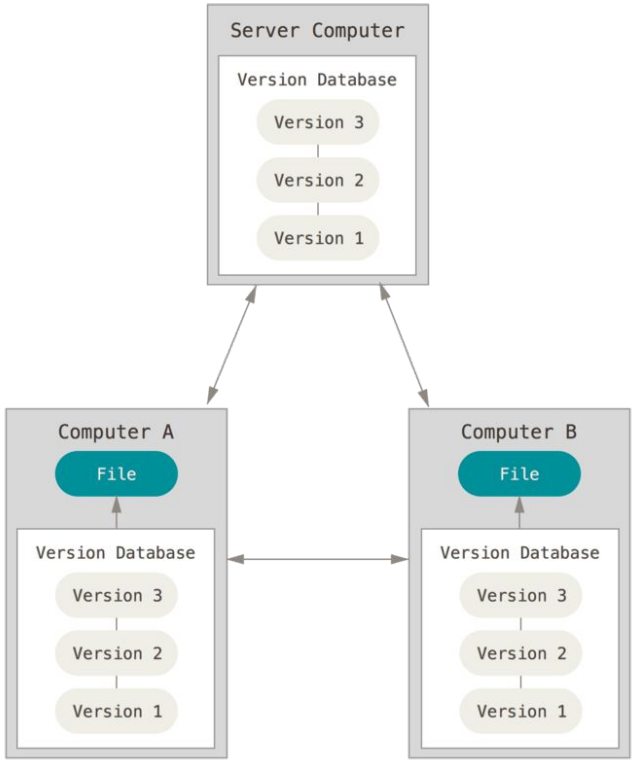
Systemy kontroli wersji
systemy rozproszone

- Każdy klient/serwer posiada pełną kopię repozytorium (!) - możliwość odtworzenia z dowolnej kopii,
- Zmiany między różnymi repozytoriami mogą być dowolnie synchronizowane,
- Szybsze działanie - wszystkie dane są trzymane lokalnie.



Systemy kontroli wersji

systemy scentralizowane - architektura GIT



źródło: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Systemy kontroli wersji
historia GITa

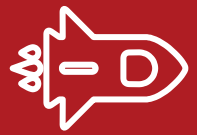
- Kiedy BitKeeper przestał być darmowy Linus Torvalds (ten od Linuxa) postanowił napisać swój VCS
  - https://youtu.be/4XpnKHJAok8?t=56
- Drugą przyczyną była nienawiść do CSVa i SVNa ;)
- Powstał na przestrzeni kilku miesięcy w 2005 roku.

Systemy kontroli wersji
cechy GITa

- Rozproszony (!),
- Wydajny,
- Wiarygodny (Reliable),

- Wsparcie wielu protokołów - np.: HTTPS i SSH,
- Możliwość pracy offline (!),
- Doskonałe wsparcie gałęzi (branch),
- Kryptograficznie gwarantowana integralność historii.

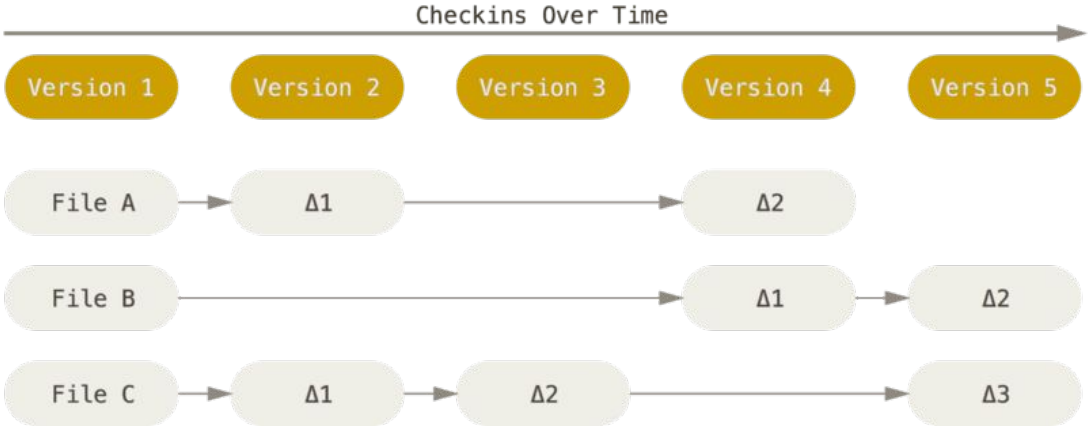
Zasada działania GIT

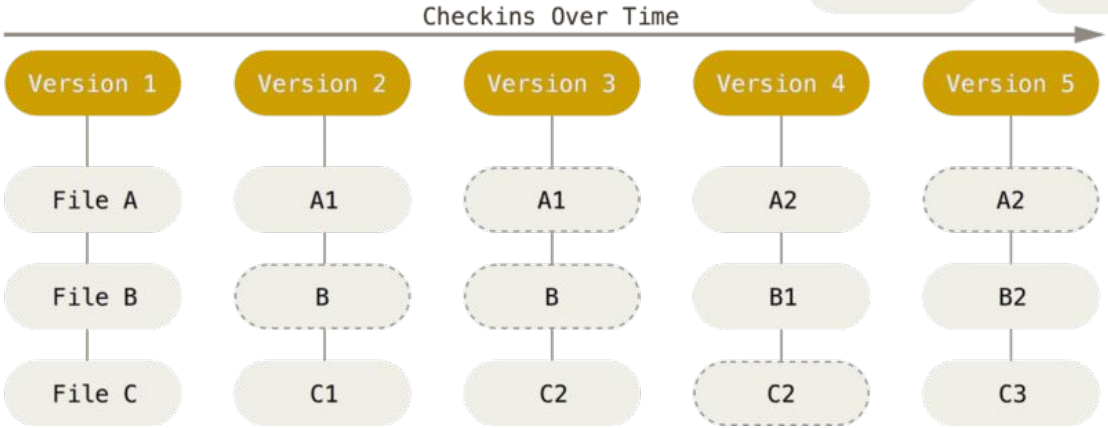


Zasada działania GIT

Snapshots, Not Differences

- Główna różnica między Git i innymi VCS (Subversion i inne) to sposób, w jaki Git traktuje swoje dane.





- Git traktuje dane bardziej jako serię migawek miniaturowego systemu plików.

Zasada działania GIT
lokalne operacje

- Większość operacji w GIT jest lokalna,
  - commit, revert, reset, add, init, log, branch, checkout
- Nie wymagają połączenia z siecią = są szybkie (!),
- Umożliwia to pracę bez dostępu do sieci.

Zasada działania GIT
integralność

- GIT zapisuje sumę kontrolną dla danych przed zapisaniem i następnie używa tej sumy jako identyfikatora zapisanych danych,
- To oznacza, że nie można zmienić zawartości żadnego pliku lub katalogu bez wiedzy GITa,
- Ta funkcjonalność jest wbudowana w GITa na najniższych poziomach i jest integralną częścią filozofii GITa,
- Git wykorzystuje do takiego sprawdzania funkcję SHA-1 - jest to 40-znakowy ciąg obliczany na podstawie zawartości plików, np.:

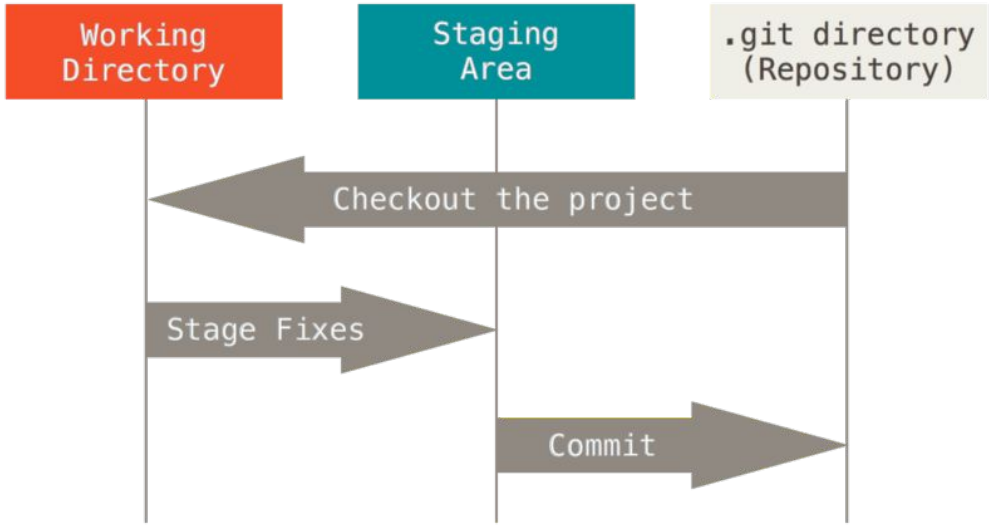
  24b9da6552252987aa493b52f8696cd6d3b00373

Zasada działania GIT
trzy stany

- "Zacommitowane" oznacza, że dane są bezpiecznie przechowywane w lokalnej bazie danych,

- "Zmodyfikowane" oznacza, że plik został zmieniony, ale nie został jeszcze zacommitowany do bazy danych.

- "Staged" oznacza, że zaznaczyłeś zmodyfikowany plik w bieżącej wersji, aby go zacommitować.



Zasada działania GIT
struktura repozytorium

- Repozytorium znajduje się w ukrytym katalogu /.git
- W pliku .gitignore
- Staging Area znajduje się w pliku index
- Dane znajdują się w katalogu /objects

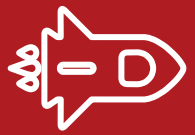
```
C:\isa (master -> origin)
λ ls -la
total 20
drwxr-xr-x 1 miig 1049089  0 May 25 21:49 ./
drwxr-xr-x 1 miig 1049089  0 May 25 22:27 ../
drwxr-xr-x 1 miig 1049089  0 May 25 21:38 .git/
-rw-r--r-- 1 miig 1049089 12 May 25 23:41 .gitignore
-rw-r--r-- 1 miig 1049089 30 May 25 20:19 READ.ME
-rw-r--r-- 1 miig 1049089 15 May 25 20:18 READ.ME.old
drwxr-xr-x 1 miig 1049089  0 May 25 21:49 help/
-rw-r--r-- 1 miig 1049089 37 May 25 21:36 index.html

C:\isa (master -> origin)
λ ls -la .git
total 22
drwxr-xr-x 1 miig 1049089   0 May 25 21:38 ./
drwxr-xr-x 1 miig 1049089   0 May 25 21:49 ../
-rw-r--r-- 1 miig 1049089  15 May 25 21:38 COMMIT_EDITMSG
-rw-r--r-- 1 miig 1049089  23 May 25 17:57 HEAD
-rw-r--r-- 1 miig 1049089 189 May 25 20:36 config
-rw-r--r-- 1 miig 1049089  73 May 25 17:57 description
-rw-r--r-- 1 miig 1049089 299 May 25 21:45 gitk.cache
drwxr-xr-x 1 miig 1049089   0 May 25 17:57 hooks/
-rw-r--r-- 1 miig 1049089 297 May 25 21:38 index
drwxr-xr-x 1 miig 1049089   0 May 25 17:57 info/
drwxr-xr-x 1 miig 1049089   0 May 25 18:01 logs/
drwxr-xr-x 1 miig 1049089   0 May 25 21:38 objects/
drwxr-xr-x 1 miig 1049089   0 May 25 17:57 refs/
```

Zasada działania GIT
podstawowy tryb pracy z repozytorium

- Modyfikacja plików w "Working directory",
- "Stageowanie" wybranych zmian które chcemy zacommitować,
- Zacommitowanie zmian, które zostają zapisane w bazie GITa,
- "Wypchnięcie" zmian do zdalnego repozytorium.

Podstawowe operacje



clone, commit, pull, push, revert, reset, add, init

Podstawowe operacje
konfiguracja GITa

- Podstawowa konfiguracja GITa:
  - >> git config --global user.name "Jan Kowalski"
  - >> git config --global user.email "jan.kowalski@example.com"
  - >> git config --global credential.helper store
  - >> git config --global core.editor nano
- W razie problemów:
  - >> git config --help

Podstawowe operacje
zadanie

- Skonfiguruj lokalnego GITa
  - user.name
  - user.email
  - credential.helper "cache --timeout=3600"
  - core.editor

Podstawowe operacje
inicjalizacja lokalnego repozytorium

- Aby zainicjować lokalne repozytorium, w dowolnym katalogu, używamy:
  - >> git init
- Aby dodać plik do "Staging Area" używamy:
  - >> git add README.md albo >> git add .
- Aby sprawdzić co znajduje się w "Staging Area":
  - >> git status
- Aby zacommitować zmiany:
  - >> git commit -m "Nasz pierwszy commit"
- Aby zobaczyć lokalną historię zmian:
  - >> git log
- Aby zobaczyć historię zmian na pliku:
  - >> git log -p README.md
- Aby uzyskać pomoc:
  - >> git help albo >> git init --help

Podstawowe operacje
zadanie

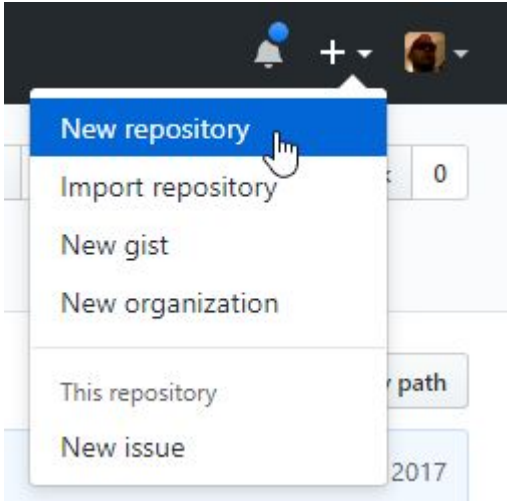
- Zainicjuj lokalne repozytorium (np. w katalogu ~/isa_demo)
- Dodaj w nim plik README.MD o dowolnej treści
- Dodaj ten plik do "Staging Area"
- Sprawdź co znajduje się w "Staging Area"
- Zacommituj z dowolnym komentarzem
- Sprawdź log repozytorium
- Zmień plik README.MD i zacommituj zmiany

Podstawowe operacje
repozytorium zdalne

- Aby połączyć lokalne repozytorium ze zdalnym:
  - >> git remote add **origin** https://github.com/michalig/demo.git

- Aby "wypchnąć" lokalne zmiany do zdalnego repozytorium:
  - >> git push -u **origin** master

Podstawowe operacje
zadanie



- Stwórz publiczne repozytorium na bitbucket
- Połącz lokalne repozytorium z nowo utworzonym
- "Wypchnij" wcześniej wprowadzone zmiany do zdalnego repozytorium

Podstawowe operacje
klonowanie zdalnego repozytorium

- Aby sklonować repozytorium:
  - >> git clone https://github.com/michalig/demo.git
- Klonowanie domyślnie ściąga całą historię,
- Zasięg klonowanej historii można ograniczyć parametrem
  - >> git clone --depth=X

Podstawowe operacje
zadanie

- Sklonuj swoje zdalne repozytorium do folderu ~/isa_remote
- Sprawdź historię tego repozytorium
- Czy wszystko się zgadza?

Podstawowe operacje
cofanie zmian

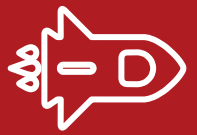
- Aby usunąć plik ze "Staging Area" (nie chcemy go commitować)
  - >> git reset README.MD
- Aby wycofać wszystkie commity po danym commicie ale pozostawić zmiany z tych commitów "Staging Area"
  - >> git reset --soft [ID]
- Aby wycofać wszystkie commity po danym commicie (zmiany z tych commitów są usuwane ze "Staging Area"
  - >> git reset --mixed [ID]
- Aby odrzucić zmiany całkowicie (!)
  - >> git reset --hard [ID]
- Aby odwrócić zmiany z danego commita (już "wypchniętego")
  - >> git revert [ID]

Podstawowe operacje
zadanie

- Wykonaj zmiany w lokalnym repozytorium, zacommituj i wypchnij do zdalnego
- Spróbuj cofnąć zmiany używając polecenia git reset --hard [ID]
- Sprawdź log i status repozytorium
- Spróbuj "wypchnąć" zmiany

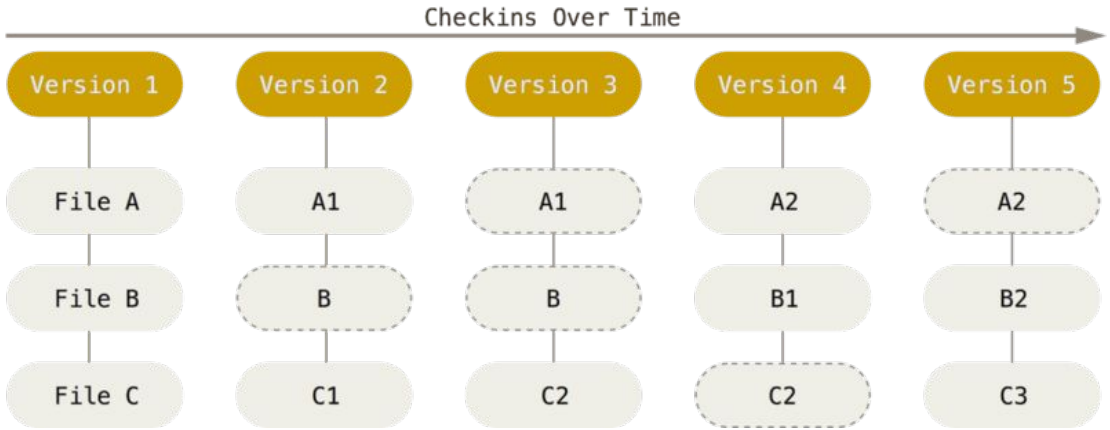
- Uaktualnij lokalne repozytorium (git pull)
- Cofnij ostatnie zmiany przy użyciu polecenia git revert [ID]
- Sprawdź log i status repozytorium
- Spróbuj "wypchnąć" zmiany
- Sprawdź status, log i zawartość zmienionego pliku

Branching



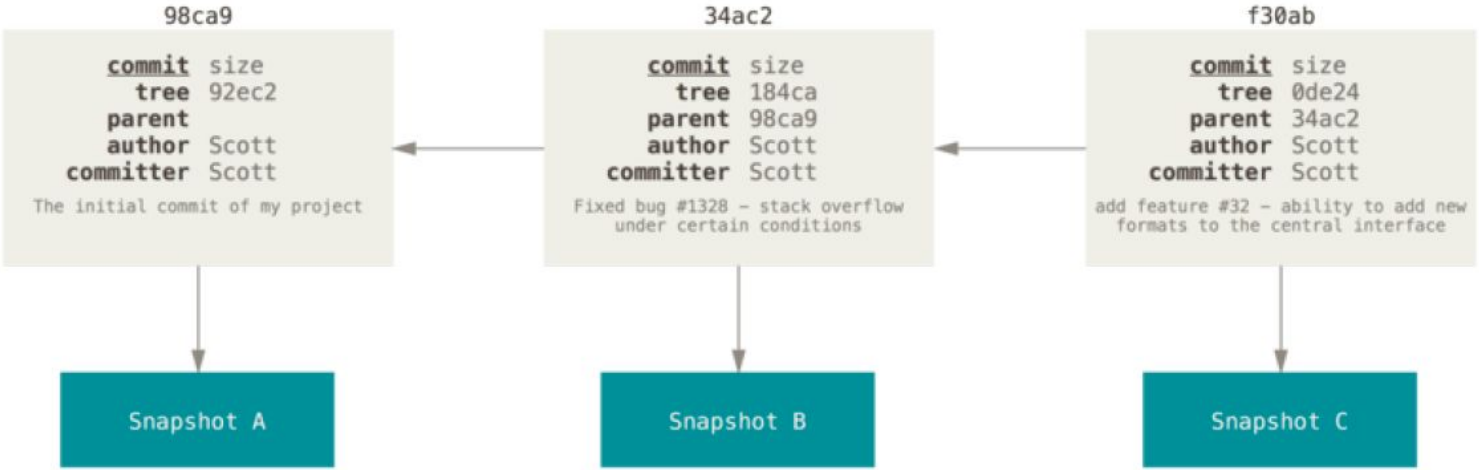
Branching
gałęzie

- Rewizje (wersje) tworzą punkty w historii repozytorium,
- Każda rewizja jest tworzona przez commity
- Każdy commit zawiera snapshot - drzewo plików i katalogów w projekcie



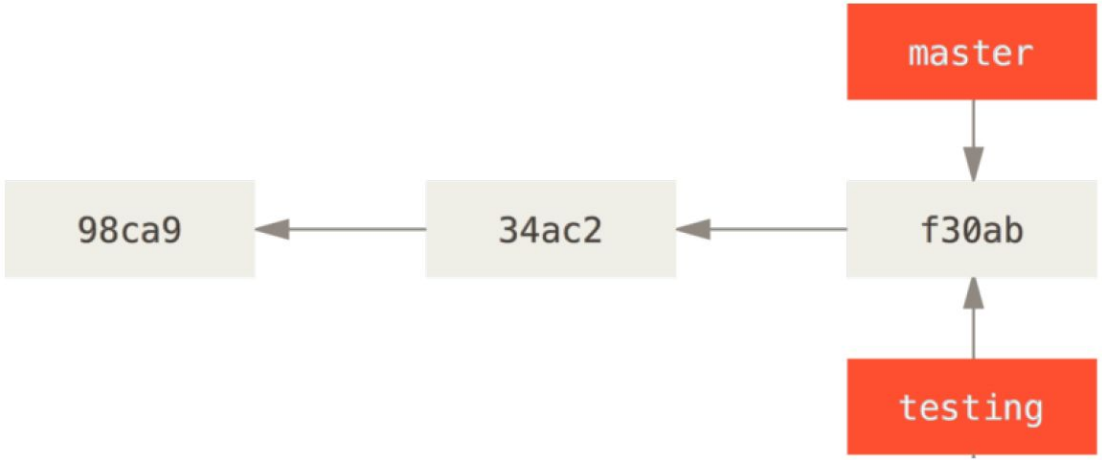
Branching
gałęzie

- Commity są powiązane poprzez id i w ten sposób tworzą następujące po sobie punkty na osi czasu,
- Gałąź to ruchomy wskaźnik na commit,
- Gałąź jest wyznaczana przez pojedynczy, wskazywany commit.



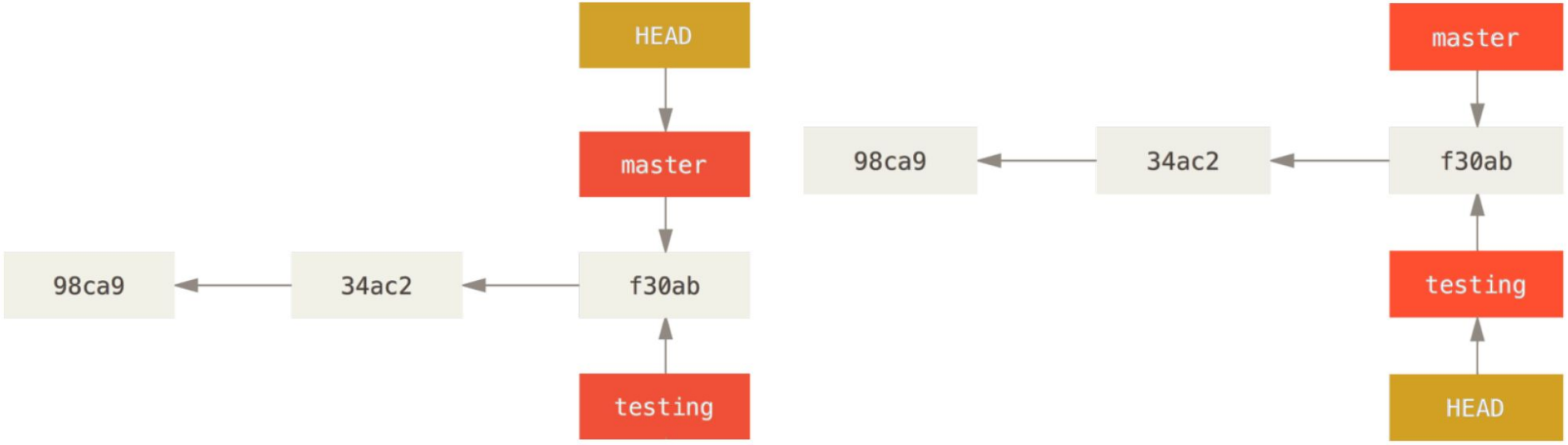
Branching
wskaźniki

- Domyślnie w repozytorium jest jedna gałąź - master,
- Wskazuje ona najnowszy commit na osi,
- Utworzenie nowej gałęzi to utworzenie nowego, nazwanego wskaźnika na dowolny commit,
- Każda gałąź ma swoją nazwę i jest reprezentowana przez plik w katalogu .git,
- Taki plik zawiera id najnowszego commita
- Takie pliki nazywają się heads - głowy gałęzi.



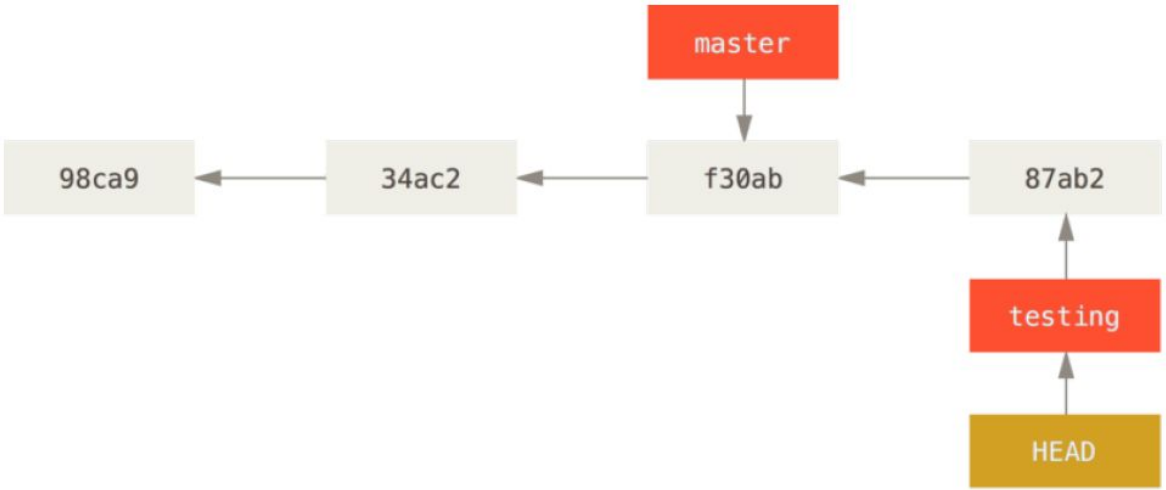
Branching
obiekt HEAD

- Git posiada też specjalny wskaźnik - HEAD, który mówi na której gałęzi aktualnie jesteśmy,
- Zawiera on nazwę wskaźnika reprezentującego gałąź,
- Zmiana gałęzi powoduje zmianę wskaźnika HEAD.



Branching
commit na branchu

- head danej gałęzi wskazuje na najnowszy commit,
- HEAD wciąż wskazuje na aktualną gałąź (nie został zaktualizowany).



Branching
detached HEAD

- W GIT można się cofać do określonego commita,
- HEAD wówczas wskazuje na ten commit - plik HEAD zawiera id commita,
- Taki stan określa się jako detached HEAD,
- Konsola i WebStorm ostrzegają, jeśli taka sytuacja wystąpi,
- Nie powinno się commitować w takiej sytuacji - commit trafia do gałęzi bez nazwy.

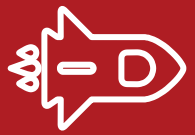
Branching
podstawowe operacja

- Aby utworzyć nowy branch:
  - >> git branch {nazwa}
- Aby przełączyć się na inny branch:
  - >> git checkout {nazwa}
- Można też stworzyć branch i od razu się na niego przełączyć:
  - >> git checkout -b {nazwa}
- Aby zaktualizować lokalne repozytorium:
  - >> git fetch
- Aby zobaczyć listę branchy:
  - >> git branch -a albo >> git branch

Branching
zadanie

- Sklonuj repozytorium
  - https://github.com/infoshareacademy/jjdz5-materialy-git.git
- Stwórz własny branch o nazwie {imie}_{nazwisko}
- Dodaj plik o nazwie {imie}.{nazwisko}.txt o dowolnej treści,
- Zacommituj i wypchnij zmiany do zdalnego repozytorium,
- Spróbuj ściągnąć wszystkie branche ze zdalnego repozytorium,
- Sprawdź listę wszystkich branchy.

GIT workflow



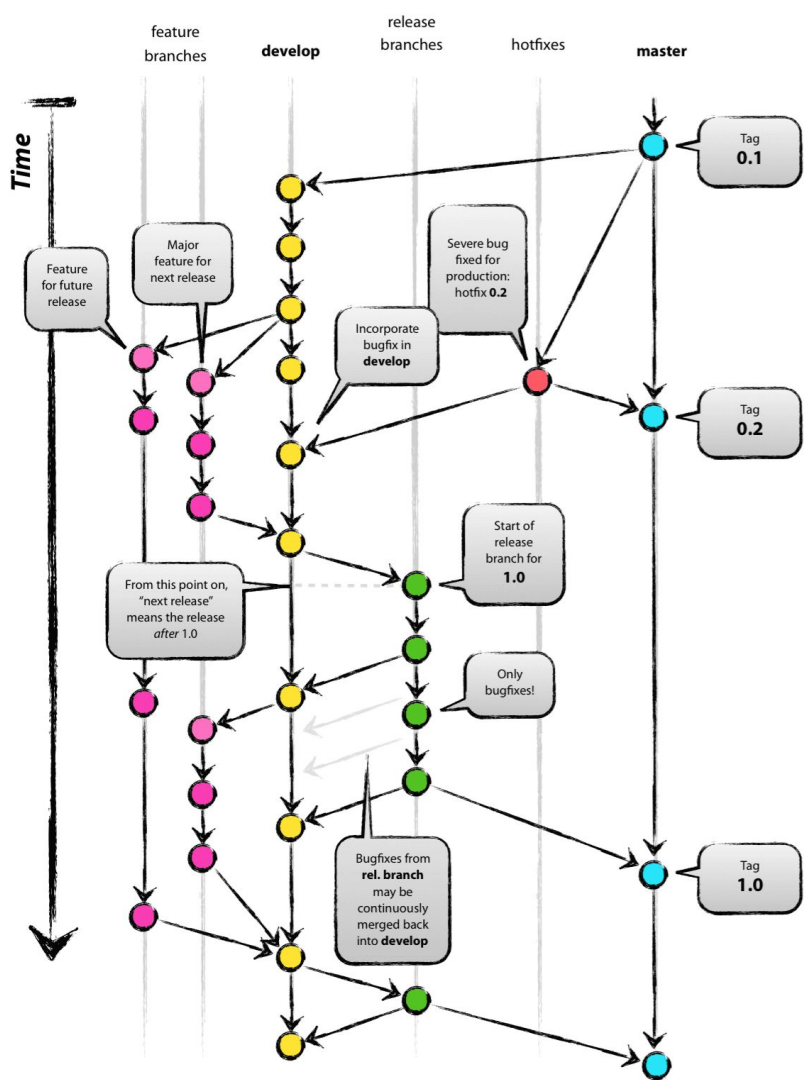
GIT workflow
wstęp

```
λ git flow init

Which branch should be used for bringing forth production releases?
   - another_one
   - develop
   - master
Branch name for production releases: [master]

Which branch should be used for integration of the "next release"?
   - another_one
   - develop
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] v_
```



GIT workflow
zasady

- Pracujemy na feature branchach,
  - Jedno zadanie/funkcja - jeden branch,
- Branche odpowiednio nazywamy, np.:
  - feature/FZ-54_some_new_awesome_feature
  - hotfix/FZ-54_some_new_awesome_feature
- Commity odpowiednio komentujemy, np.:
  - FZ-54 new awesome feature

Dzieki!!

Pytania?

michal.ignaciuk@gmail.com