

Git



Kto mówi?

Mateusz Sosnowski

IT systems designer @ Logintegra Sp. z o. o.

| Co może być zmianą?

- Nowy plik dodany do projektu
- Usunięcie pliku z projektu
- Modyfikacja pliku (np. dopisanie kawałka kodu)
- Zmiana nazwy / przeniesienie pliku

| Repozytorium to ...

Repozytorium (łac. *repositorium*) – miejsce uporządkowanego przechowywania dokumentów, z których wszystkie przeznaczone są do udostępniania.

Określenie *repozytorium* odnosi się przede wszystkim do miejsca przechowywania, a nie udostępniania.

| System kontroli wersji to ...

System kontroli wersji (ang. *version/revision control system*) – oprogramowanie służące do śledzenia zmian głównie w kodzie źródłowym oraz pomocy programistom w łączeniu zmian dokonanych w plikach przez wiele osób w różnym czasie.

| Funkcje systemu kontroli wersji

- przechowywanie i kontrola dostępu do plików związanych z projektem,
- historia zmian,
- śledzenie modyfikacji zachodzących w poszczególnych plikach,
- pełna dokumentacja wprowadzanych zmian,
- możliwość tworzenia i śledzenia różnorodnych konfiguracji oprogramowania,
- udostępnianie kolejnych wersji poszczególnych plików.

| Funkcje związane z organizacją pracy zespołowej

- kontrola dostępu do plików dla różnych osób,
- synchronizacja zmian wprowadzanych przez różnych autorów,
- rozwiązywanie konfliktów pomiędzy kolidującymi ze sobą zmianami,
- praca w środowisku rozproszonym w sieci komputerowej,
- kontrola faz procesu rozwijania oprogramowania, na przykład wymuszanie faz testowania czy dokumentowania zmian.

| Systemy kontroli wersji



RCS



SCCS

10/24/2020

Systemy kontroli błędów i wersji

| Podział systemów kontroli wersji

Rozproszone



Scentralizowane



Lokalne

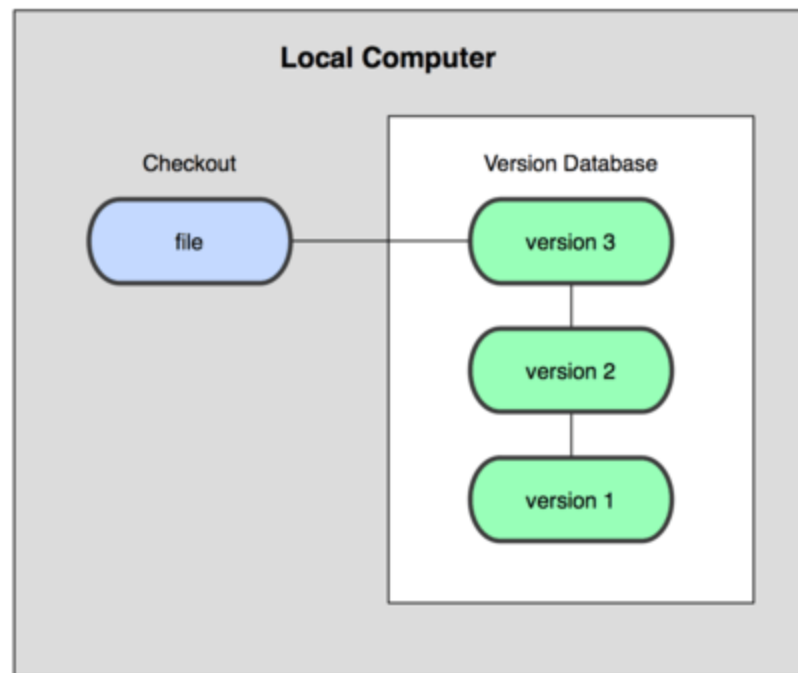
RCS

SCCS

| Lokalne systemy kontroli wersji

| Lokalne systemy kontroli wersji

Lokalne systemy kontroli wersji pozwalają na stworzenie repozytoriów danych tylko na lokalnym komputerze, czyli używać ich może tylko jedna osoba.



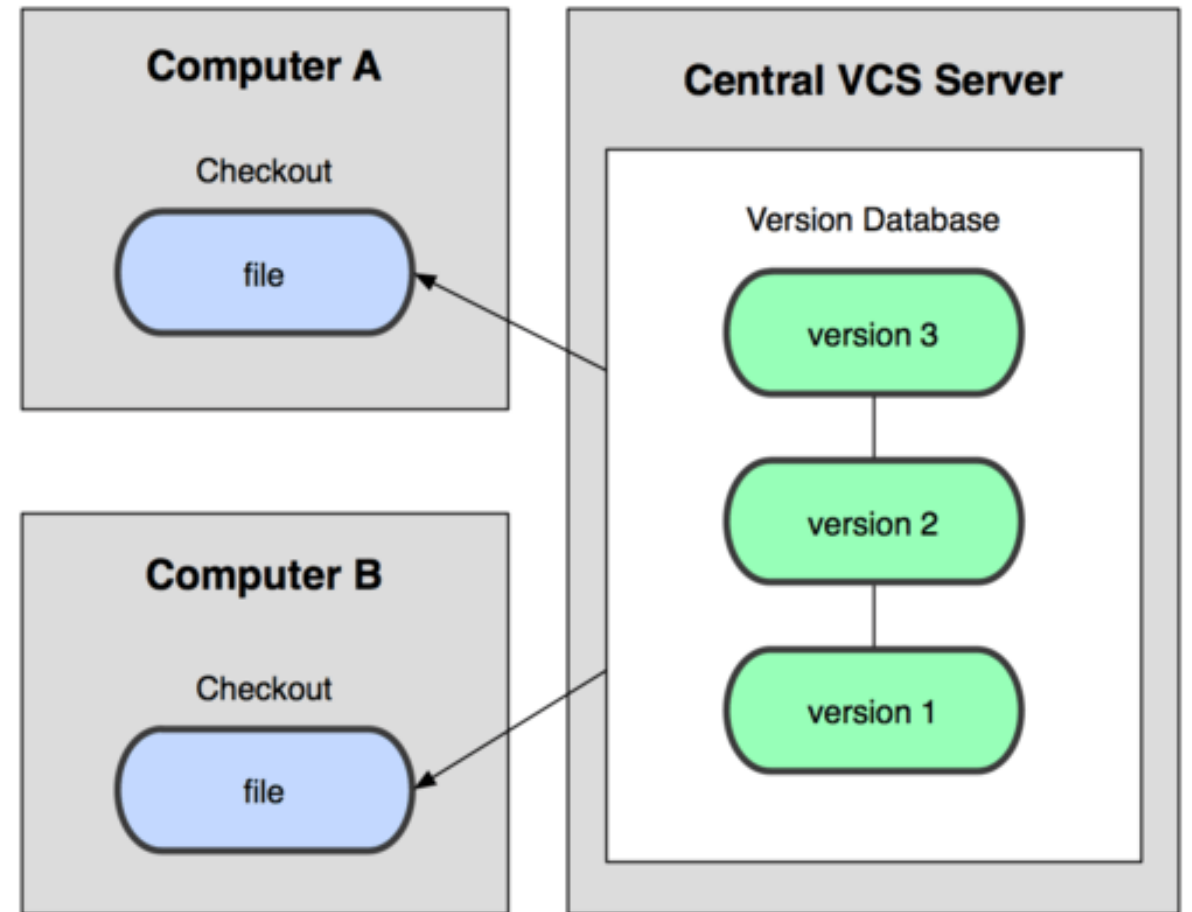
| Lokalne systemy kontroli wersji

Wiele osób decyduje się na kopiowanie plików do innego katalogu oznaczanego odpowiednią datą. Ta metoda jest często wybierana ze względu na łatwość wykonania. Jednakże, trzeba zauważyć, że jest to opcja podatna na błędy. Łatwo pomylić foldery, omyłkowo zmodyfikować pliki albo skopiować złe dane. Rozwiązaniem tego problemu, było stworzenie bazy danych, w której przechowuje się wszystkie zmiany, jakie się dokonywały na śledzonych plikach.

| Scentralizowane systemy kontroli wersji

| Scentralizowane systemy kontroli wersji

- Mamy tylko jedno repozytorium na serwerze
- Synchronizujemy wersje zawsze na serwerze
- Jeśli coś się zepsuje to mamy poważny problem



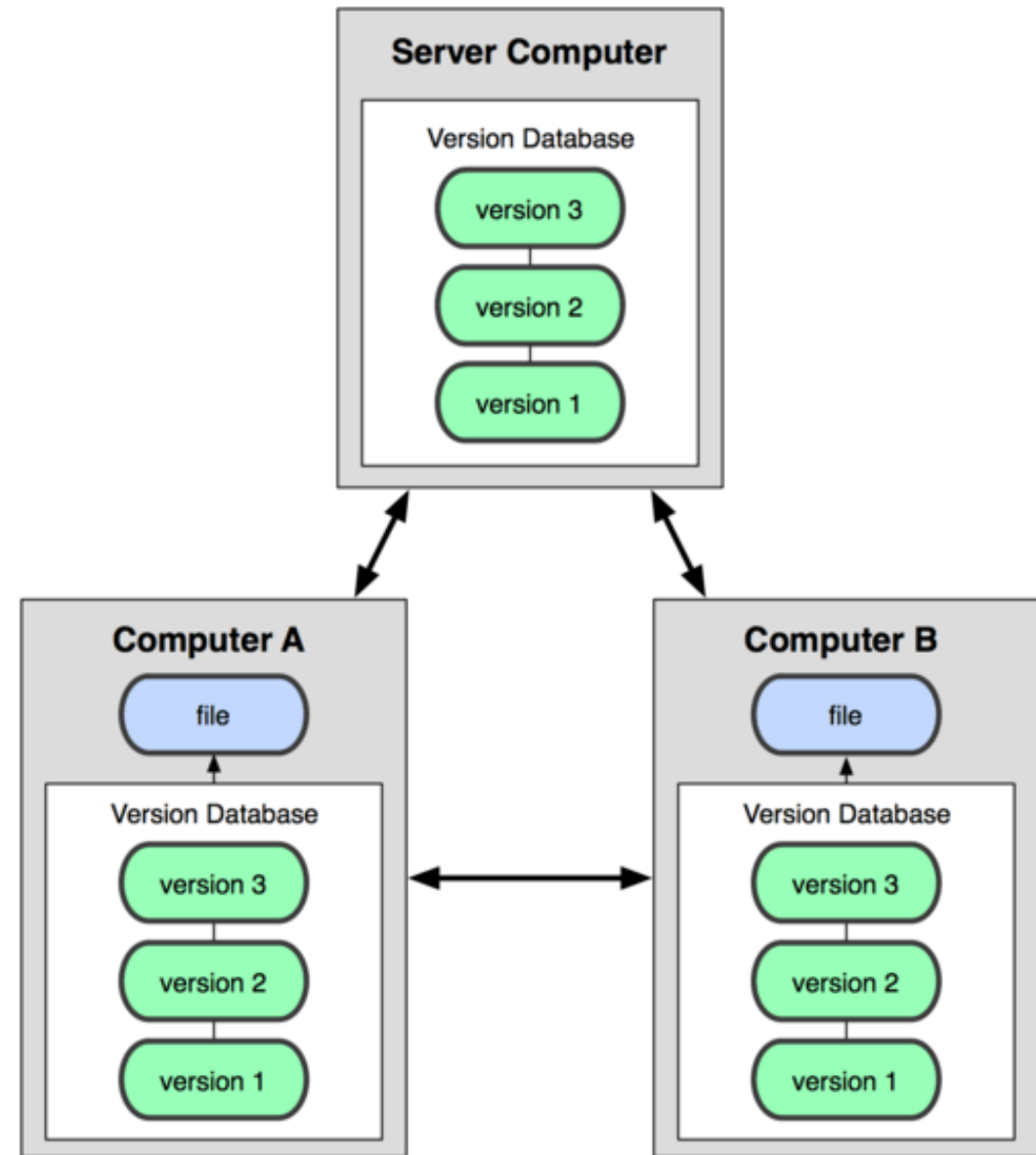
| Scentralizowane systemy kontroli wersji

Scentralizowane systemy kontroli wersji składają się z jednego serwera, gdzie zapisane są wszystkie śledzone pliki. Klienci mogą się z nim połączyć i uzyskać dostęp do najnowszych wersji. Dzięki temu rozwiązaniu, każdy może sprawdzić, co robią inni uczestnicy projektu. Dodatkowo, w porównaniu z lokalnymi bazami danych, system CVS jest dużo łatwiejszy w zarządzaniu.

| Rozproszone systemy kontroli wersji

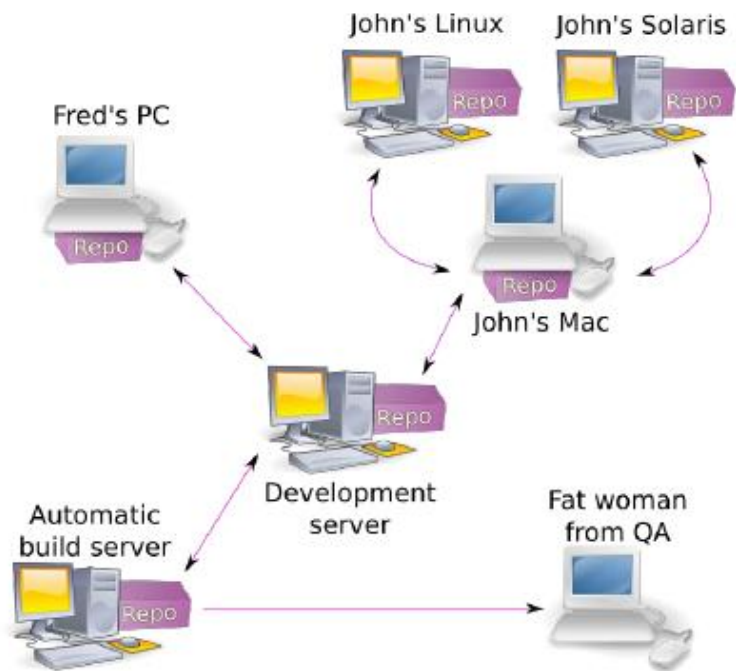
| Rozproszone systemy kontroli wersji

- Każdy ma swoje lokalne repozytorium
- Jeśli coś się zepsuje – mamy tyle repozytoriów ~ ilu developerów



Porównanie

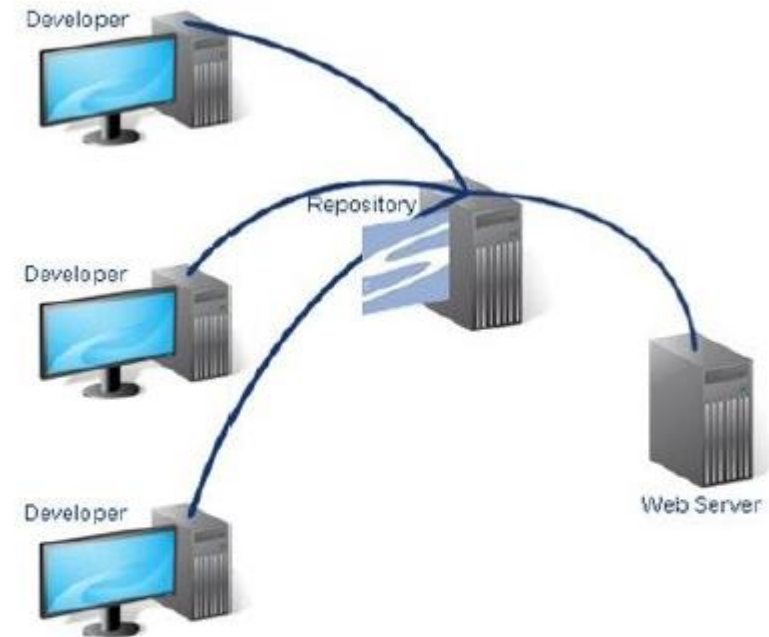
Rozproszone



Każdy ma swoje repozytorium.
Serwer też ma swoje repozytorium.

vs

Scentralizowane



Istnieje tylko jedno centralne repozytorium.
Tylko serwer ma repozytorium.
Reszta ma lokalne kopie plików.

GIT

System kontroli wersji



| Od czego to się zaczęło?

2005 r.

Jako narzędzie do wersjonowania kodu przy rozwijaniu Linuxa.



https://en.wikipedia.org/wiki/Linus_Torvalds

| Git - zalety

- Rozproszony
- Szybki
- Bezpieczny
- Elastyczny – wiele flow pracy
- Darmowy i rozwijany jako OSS
- Standard



Git config czyli podstawowa konfiguracja ustawień git



| Git config

Ustawienia obligatoryjne:

user.name

user.email

Do rozpoczęcia pracy z git konieczne jest ustawienie nazwy oraz email użytkownika. Plik konfiguracyjny .gitconfig po instalacji git znajduje się w katalogu domowym użytkownika (niezależnie od systemu).

git config --global user.name "Mateusz Sosnowski"

git config --global user.email "Mateusz.Sosnowski@logintegra.com"

Sprawdzenie ustawionych wartości w pliku konfiguracyjnym git:

```
matisosna@DESKTOP-S9S5N03:~$ git config user.name
Mateusz Sosnowski
matisosna@DESKTOP-S9S5N03:~$
```

```
matisosna@DESKTOP-S9S5N03:~$ git config user.email
mateusz.sosnowski@logintegra.com
matisosna@DESKTOP-S9S5N03:~$
```

Wyświetlenie wszystkich parametrów:

git config --list

| Git config

Poziomy według priorytetu:

--local

--global

--system

Dla polecenia git config można określić na jakim poziomie konfiguracji ma obowiązywać dane ustawienie.

--local

Domyślna opcja w przypadku braku podania poziomu w poleceniu git config. Wartości konfiguracji lokalnej są przechowywane w pliku, który znajduje się w pliku config katalogu .git w danym repozytorium.

--global

Konfiguracja stosowana dla użytkownika systemu operacyjnego. Globalne wartości konfiguracyjne przechowywane są w pliku .gitconfig w katalogu domowym użytkownika.

--system

Konfiguracja na poziomie systemu jest stosowana na całym komputerze i dotyczy wszystkich użytkowników systemu operacyjnego i wszystkich repozytoriów. Plik konfiguracyjny na poziomie systemu znajduje się w pliku gitconfig poza ścieżką katalogu głównego systemu.

| Git config

Ustawienia
opcjonalne:

core.editor

alias

Domyślnym edytorem jest vim, można zmienić editor np. na notepad:

```
git config --global core.editor notepad
```

Dla poleceń git możemy wprowadzić własne aliasy, np:

```
$ git config --global alias.co checkout  
$ git config --global alias.br branch  
$ git config --global alias.ci commit  
$ git config --global alias.st status
```

Po wprowadzeniu aliasów można stosować wprowadzone polecenia, a w pliku .gitconfig pojawią się wpisy, które będzie można edytować.

```
[alias]  
  co = checkout  
  br = branch  
  ci = commit  
  st = status
```

| Git config

Ustawienia opcjonalne:

color.ui

color.branch

color.status

color.decorate

W git można ustawić spersonalizowany motyw kolorów. Do włączenia kolorowania na wyjściu terminala git należy ustawić wartość: color.ui na `true`.

git config --global color.ui true

Możliwości:

git config --global color.banch.<slot>

<slot>: current, local, remote, upstream, plain

git config --global color.status.<slot>

<slot>: header, added or updated, changed, untracked, branch, nobranch, unmerged

git config --global color.decorate.<slot>

<slot>: branch, remoteBranch, tag, stash, HEAD

```
[color]
  ui = auto
[color "branch"]
  current = yellow reverse
  local = yellow
  remote = green
[color "diff"]
  meta = yellow bold
  frag = magenta bold
  old = red bold
  new = green bold
[color "status"]
  added = yellow
  changed = green
  untracked = cyan
```

Mechanizmy wewnętrzne w Git

- czyli jak to działa?



| Mechanizmy wewnętrzne w git

Komendy

Plumbing

Porcelain

W git możemy podzielić komendy na 2 grupy:

- Plumbing,
- Porcelain.

Komendy Plumbing

Git początkowo był tylko zestawem narzędzi do obsługi VCS, a nie pełnoprawnym systemem VCS. Dlatego ma garść komend, które wykonują niskopoziomowe czynności i zostały zaprojektowane do łączenia ich w łańcuchy komend. Komendy niskopoziomowe "plumbing" dają dostęp do wewnętrznych mechanizmów Gita i pomagają pokazać jak i dlaczego Git robi to co robi.

(git cat-file, git hash-object, git count-objects, ...)

Komendy Porcelain

Komendy bardziej przyjazne dla użytkownika: git add, git commit, git push, git pull, git branch, git checkout, git merge, git rebase.

| .git

Co kryje folder .git?

Po uruchomieniu **git init** w nowym lub istniejącym katalogu Git tworzy katalog .git, w którym jest umieszczone praktycznie wszystko czego używa Git.

description

używany przez program GitWeb

config

zawiera ustawienia konfiguracyjne

info

przechowuje globalny plik wykluczeń, który przechowuje ignorowane wzorce (.gitignore)

hooks

zawiera komendy uruchamiane po stronie klienta lub serwera

HEAD

wskazuje na gałąź na której się znajdujesz

index

przechowywane są tu informacje na temat przechowalni

objects

przechowuje całą zawartość bazy danych

refs

przechowuje wskaźniki do obiektów commitów (branches)

```
Name
----
hooks
info
objects
refs
config
description
HEAD
```

| Obiekty Gita

Git to ...

... system plików
zorientowanych na treść.

Git to tak na prawdę system plików zorientowany na treść.

Git u podstaw to baza danych, w której znajdują się dane i przypisane do niej klucze. Można zapisać dowolny rodzaj danych, a w odpowiedzi otrzymujemy klucz, dzięki któremu można się dostać do tych danych w każdej chwili.

Aby to zademonstrować można skorzystać w tym celu z komendy: **hash-object**, która pobiera jakieś dane, zapisuje je w katalogu .git i zwraca klucz pod którym te dane zostały zapisane.

Po zainicjowaniu nowego repozytorium Git inicjalizuje katalog **objects** oraz tworzy w nim dwa katalogi **pack** i **info**. Jednak nie ma w nich żadnych plików.

| Ćwiczenie

Obiekty Gita

1. Tworzymy nowy folder.
2. Inicjalizujemy repozytorium Git.
3. Sprawdzamy zawartość katalogu z repozytorium.
4. Zapisujemy dane w bazie danych Gita.

```
matisosna@DESKTOP-S9S5N03:~/repo$ echo 'test content' | git hash-object -w --stdin  
d670460b4b4aece5915caf5c68d12f560a9fe3e4  
matisosna@DESKTOP-S9S5N03:~/repo$
```

5. W odpowiedzi dostajemy 40 znakową sumę kontrolną. Sprawdzamy zawartość folderu z repozytorium.
6. Przechodzimy do folderu: **cd ./git/objects**. Co się tam znajduje?

```
matisosna@DESKTOP-S9S5N03:~/repo/.git/objects$ tree  
.  
├── d6  
│   └── 70460b4b4aece5915caf5c68d12f560a9fe3e4  
├── info  
└── pack  
  
3 directories, 1 file
```

7. Pobieramy dane z Gita: **git cat-file <SHA1> -p**.

```
matisosna@DESKTOP-S9S5N03:~/repo/.git/objects$ git cat-file -p d670460b  
test content  
matisosna@DESKTOP-S9S5N03:~/repo/.git/objects$
```

| Ćwiczenie

Obiekty Gita

8. Dodajemy nowy plik w głównym katalogu repozytorium.

```
matisosna@DESKTOP-S9S5N03:~/repo$ echo "This is a text string" > test.txt
matisosna@DESKTOP-S9S5N03:~/repo$ ls
test.txt
matisosna@DESKTOP-S9S5N03:~/repo$ _
```

9. Zapisujemy zawartość w bazie danych: **git hash-object -w test.txt**.

```
matisosna@DESKTOP-S9S5N03:~/repo$ git hash-object -w test.txt
dcbdc0a43751cf318ec35f9580d37435db82f1f5
matisosna@DESKTOP-S9S5N03:~/repo$ _
```

10. Wprowadzamy zmianę w tym pliku i zapisujemy ponownie.

11. Sprawdzamy co jest w bazie danych: **tree .git/objects**.

```
matisosna@DESKTOP-S9S5N03:~/repo$ tree .git/objects/
.git/objects/
├── c1
│   └── a7a2a7a45ca7f4fe15048b790ed098d443c495
├── d6
│   └── 70460b4b4aece5915caf5c68d12f560a9fe3e4
├── dc
│   └── bdc0a43751cf318ec35f9580d37435db82f1f5
├── info
└── pack

5 directories, 3 files
```

12. Baza danych ma teraz 2 wersje pliku.

13. Cofamy zawartość pliku do pierwszej wersji: **git cat-file -p <SHA1> > test.txt**.

14. Wracamy do drugiej wersji pliku.

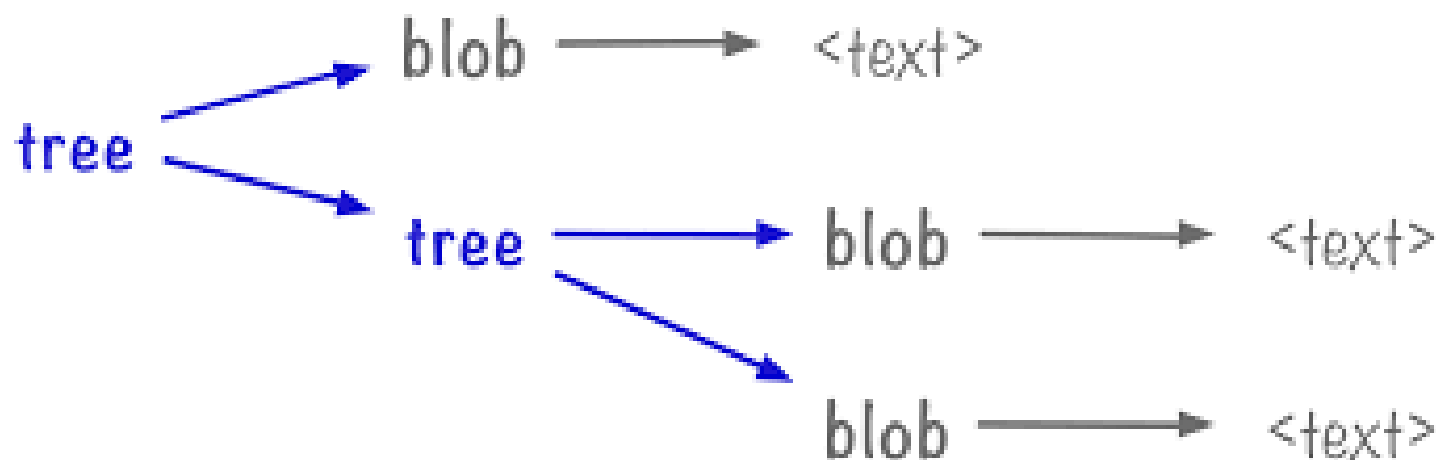
Systemy kontroli błędów i wersji

| Tree and blob

Tree - rozwiązuje problem przechowywania nazwy pliku, pozwala przechować grupę plików razem.

W Git cała zawartość jest przechowywana jako obiekty **tree** i obiekty **blob**, a **tree** odpowiadają pozycjom katalogu UNIX, a obiekty **blob** odpowiadają zawartościom plików.

- **blob** — object blob służy do przechowywania zawartości pliku.
- **tree** — object tree zawiera rerefencję do obiektów blob lub poddrzew.
- **commit** — object commit zawiera referencję do obiektu tree oraz trochę innych informacji (author, committer etc.)
- **tag** — tag to inna referencja wskazująca na obiekt commit, po prostu w celu uzyskania łatwiejszej referencji do obiektu commit



| Tree and blob

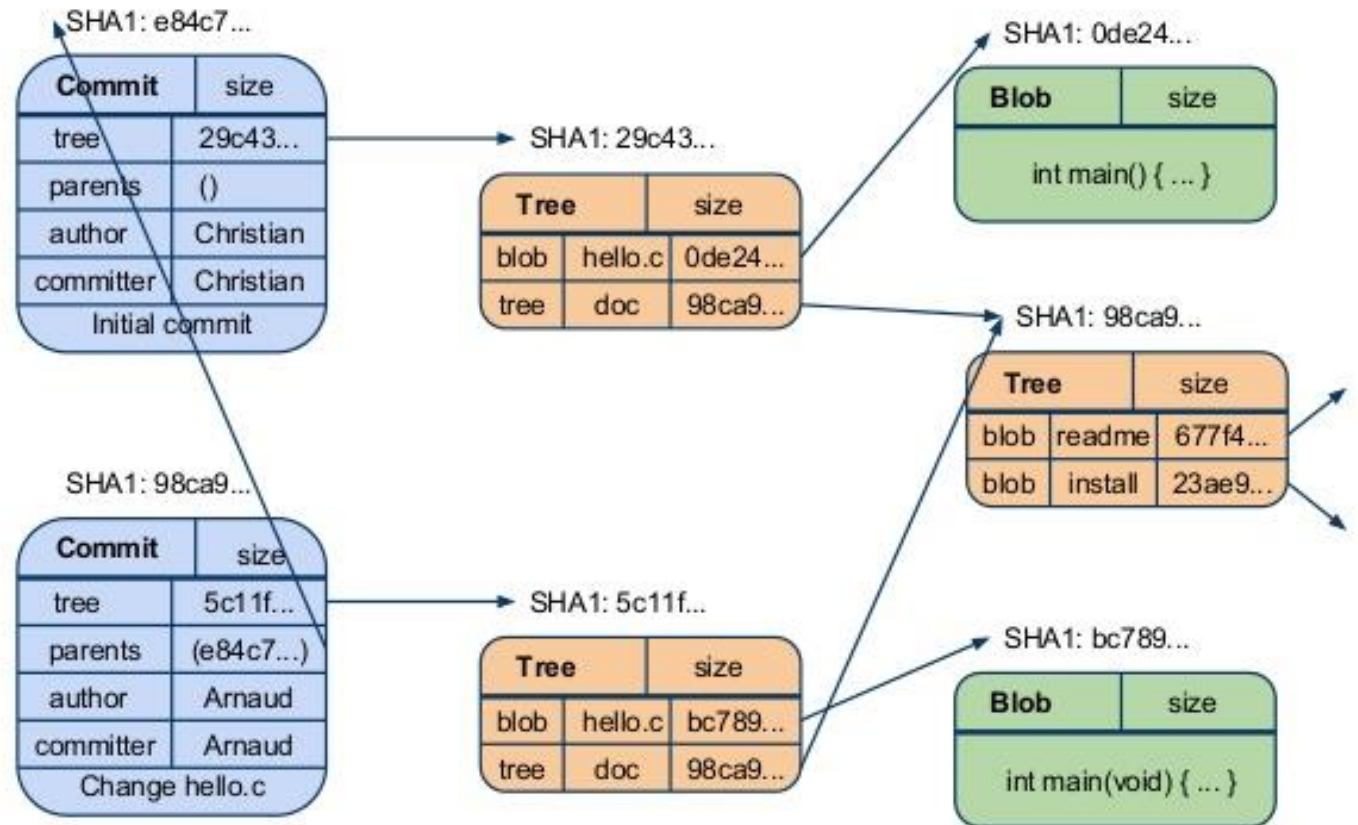
Commit:

- tree
- parents
- author
- committer

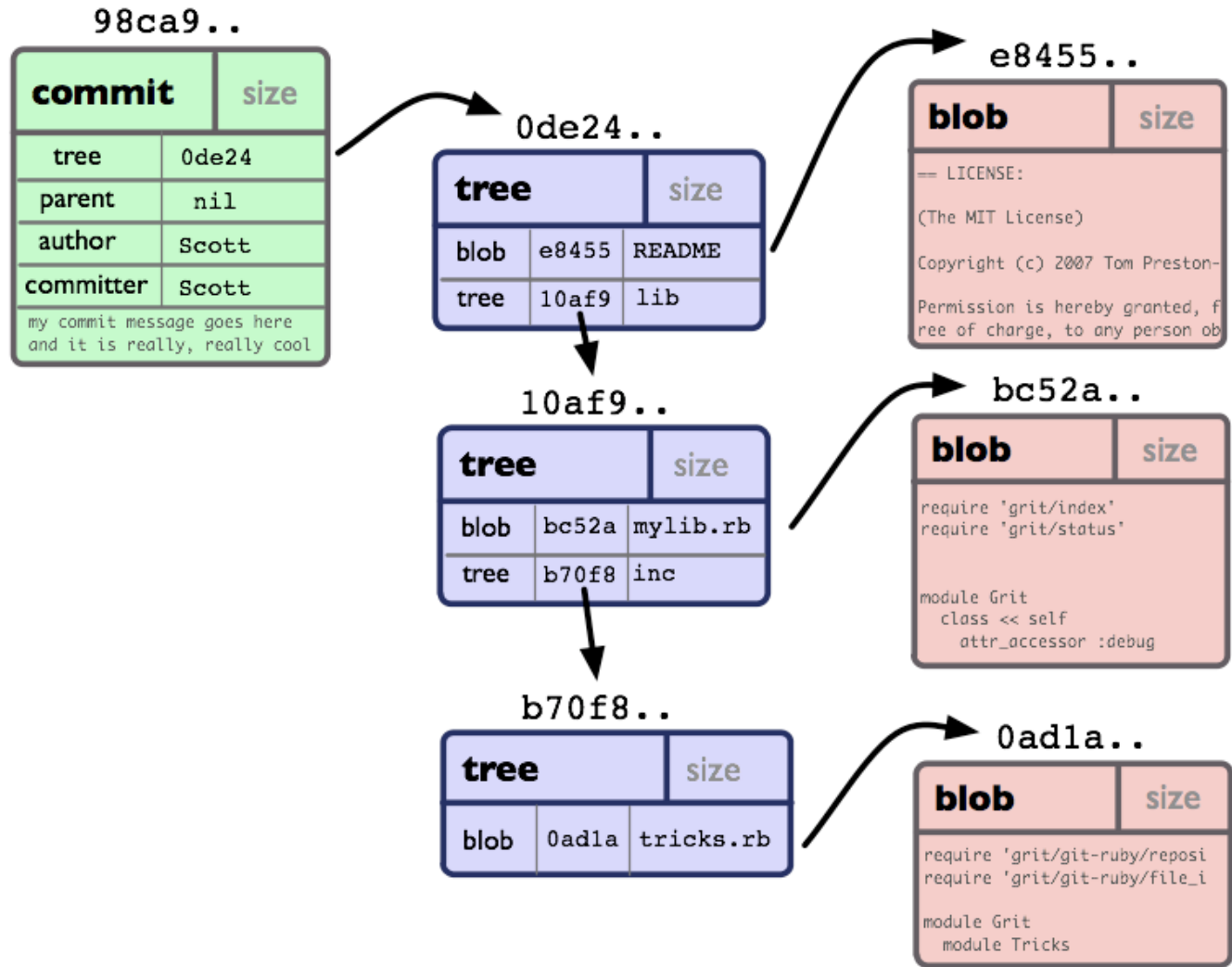
Tree:


- tree
- blob

Git Objects Relations



| Tree and blob





If you want to master Git, don't worry
about learning the commands.
Instead, learn the model.

| Co tak na prawdę śledzi git?

Git śledzi zmiany w zawartości plików.

**Ale co z pustymi katalogami?
Sprawdźmy to!**

**Ale co z białymi znakami?
Sprawdźmy to!**



- W Git katalogi istnieją tylko niejawnie, tylko poprzez ich zawartość.



- Różnica w zawartości plików co do białych znaków wygeneruje inny kod SHA1.

Ćwiczenie

W dowolnym repozytorium korzystamy z poniższych komend

```
$ git write-tree  
dc6b8ea09fb7573a335c5fb953b49b85bb6ca985
```

```
$ find .git/objects -type f  
.git/objects/a3/7f3f668f09c61b7c12e857328f587c311e5d1d  
.git/objects/b1/3311e04762c322493e8562e6ce145a899ce570  
.git/objects/ce/289881a996b911f167be82c87cbfa5c6560653  
.git/objects/dc/6b8ea09fb7573a335c5fb953b49b85bb6ca985
```

```
$ git cat-file -p dc6b8ea09fb7573a335c5fb953b49b85bb6ca985  
100644 blob b13311e04762c322493e8562e6ce145a899ce570    animals.txt
```

```
$ git cat-file -t a37f3f668f09c61b7c12e857328f587c311e5d1d  
blob
```

```
$ git cat-file -t dc6b8ea09fb7573a335c5fb953b49b85bb6ca985  
tree
```

Useful commands

```
find .git/objects -type f  
list the files in .git/objects
```

```
git cat-file -p <object ID>  
pretty-print the contents of an object in the Git object store
```

```
git cat-file -t <object ID>  
show the type of an object in the Git object store
```

```
git update-index --add <path>  
add a file to the Git index
```

```
git ls-files  
list the files that are in the current index
```

```
git write-tree  
write the current index to a new tree
```

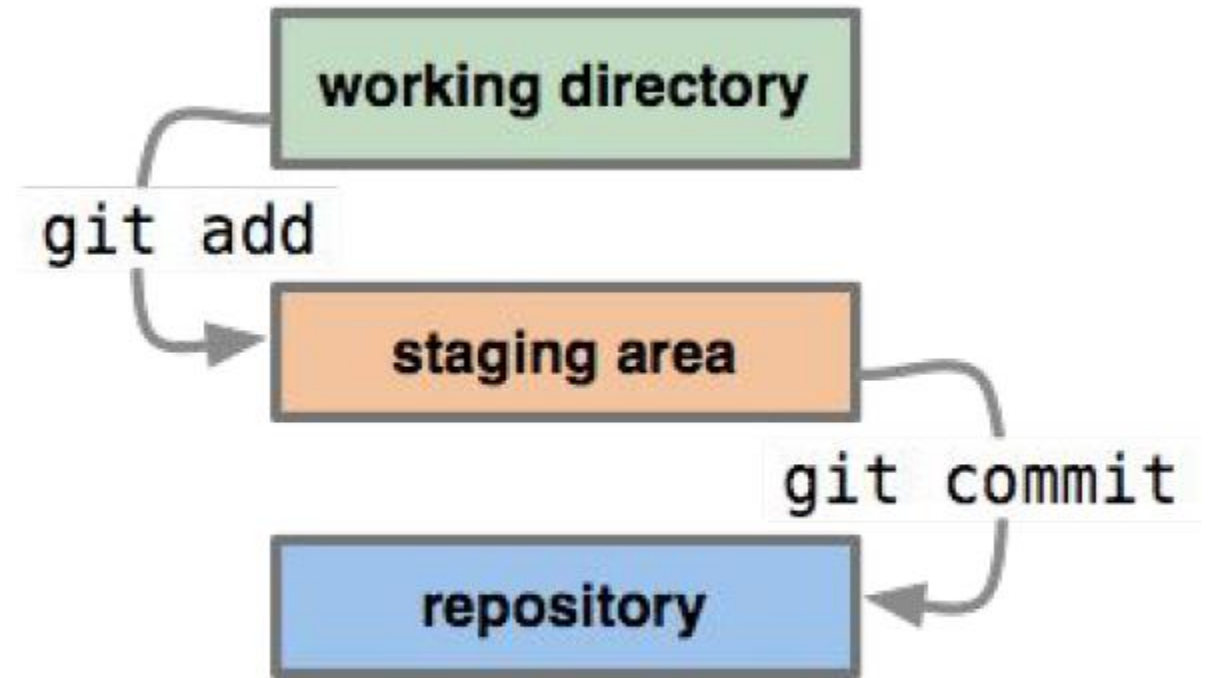
Commit oraz podstawowe operacje



| Commit

git commit

- Zapisanie różnicy zawartości plików.
- Commity są stałe i niezmiennie.
- Commit odpowiada konkretnym zmianom.
- Commit ma wiadomość



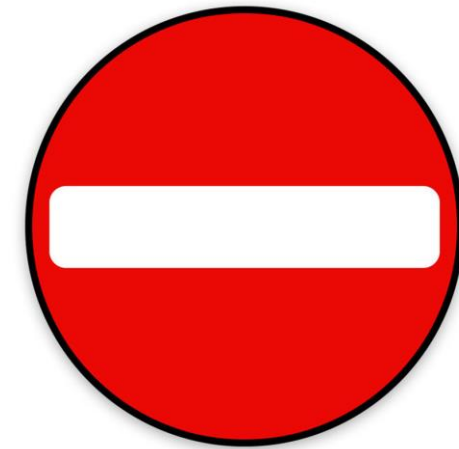
Zanim zaczniemy commitować

Nie chcemy wszystkiego commitować

Prawda?

Czego nie warto trzymać na repozytorium?

- Wynik kompilacji kodu
- Personalne ustawienia IDE
- Pliki generowane (nie zawsze)



.gitignore

Ignoruj, nie śledź

.gitignore

Specjalny plik w bazowym katalogu repozytorium.

Zawiera spis rzeczy których nie chcemy śledzić/wersjonować.

◆ .gitignore x

```
1  # Created by https://www.gitignore.io/api/webstorm
2
3  # OSX specific files
4  **/.DS_Store
5
6  ### WebStorm ###
7  # Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm,
8  # Reference: https://intellij-support.jetbrains.com/hc/
9
10 # User-specific stuff:
11 .idea/workspace.xml
12 .idea/tasks.xml
13 .idea/dictionaries
14 .idea/vcs.xml
15 .idea/jsLibraryMappings.xml
16
17 # Sensitive or high-churn files:
18 .idea/dataSources.ids
19 .idea/dataSources.xml
```

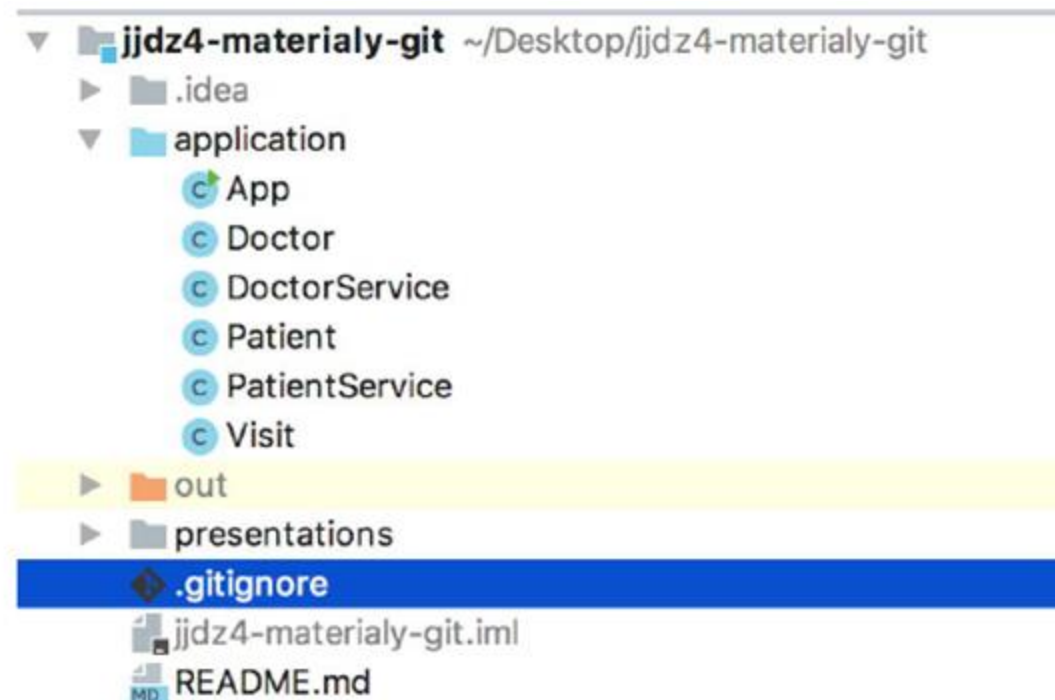
.gitignore

Ignoruj, nie śledź

.gitignore

Musi być w **bazowym katalogu** repozytorium.

Wszystkie jego ścieżki muszą być **relatywne** do jego położenia.



.gitignore

Jak go napisać?

Polecam zacząć od
wygenerowania pliku przez
<https://www.gitignore.io/>

 **gitignore.io**

Create useful .gitignore files for your project

× Java

× Maven

× IntelliJ

Create

[Source Code](#)

[Command Line Docs](#)

[Watch Video Tutorial](#)

Ćwiczenie

commitowanie

Najpierw sprawdź czy masz poprawne ustawienia gita:

`cat ~/.gitconfig` (niezależnie od system operacyjnego jest to katalog domowy usera)

powinno dać (oczywiście z waszymi danymi):

```
[user]
```

```
name = Mateusz Sosnowski
```

```
email = msosnowski@wsb.gda.pl
```

```
[core]
```

```
autocrlf = true
```

Operacje które właśnie wykonaliśmy



Co zawiera commit?

- Ma autora
- Każdy commit ma swojego rodzica
- Każdy commit ma swój klucz SHA1
- Klucz SHA1 jest unikalny

Dobry Commit

Zawartość i wiadomość

- Z wiadomości da się jednoznacznie określić czego dotyczy
- Wiadomość nie jest zbyt długa
- Wiadomość nie jest zbyt szczegółowa
- Można go powiązać z zadaniem/wymaganiem

```
task-5 Now users list  
contains user email |
```


*" Use the body to explain what
and why vs how*

Czyli - nikogo nie interesuje w wiadomości commita
że dodaliście klasę ``UserUtil`` w Javie lub ``user-list`` w
CSS.

Interesuje ich **po co** to zrobiliście.

Zły commit

The Hall of Shame

Nie róbcie tego
w domu

“bug fix”

“more work”

“wtf”

“oopsie”

“minor changes”

“Work on feature blah blah”

“Fix BUG-9284”

“Change X constant to be 10”

“super long commit message goes here, something
like 100 words and lots of characters woohoo!”

<https://www.codelord.net/2015/03/16/bad-commit-messages-hall-of-shame/>

WIADOMOŚCI COMMITÓW POWRACAJĄ !

Zaglądasz do historii dużego projektu.
Duuuuzo commitów.

Co chciałbyś zobaczyć?

Sprawdzanie historii w git



Git log

Listuje zmiany
zatwierdzone w
repozytorium.

10/24/2020

```
$ git log -p -2
```

- p pokazuje różnice
- 2 ogranicza do 2 wpisów

```
$ git log --stat
```

- stat skrócone statystyki każdej z zatwierdzonych zmian

```
$ git log --pretty=oneline
```

- pretty inny format niż domyślny
- oneline wyświetla każdą zmianę w pojedynczej linii

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : changed the version number
085bb3b - Scott Chacon, 6 years ago : removed unnecessary test
a11bef0 - Scott Chacon, 6 years ago : first commit
```

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
* 5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | 5a09431 add timeout protection to grit
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
* 11d191e Merge branch 'defunkt' into local
```

Git log Filtry

Filtr po autorze

```
git log --author="Mateusz Sosnowski"
```

```
git log --after="1-7-2017"
```

Filtr po dacie

```
git log --before="1-7-2017"
```

```
git log --before="a week ago" --oneline --pretty="%cn committed %h on %cd"
```

output

```
Dan Abramov committed 17aa4d468 on Wed Nov 8 22:59:38 2017 +0000
Dan Abramov committed 2437e2c3d on Wed Nov 8 22:54:10 2017 +0000
GitHub committed c83596df6 on Wed Nov 8 22:37:11 2017 +0000
Dan Abramov committed e88f29204 on Wed Nov 8 21:23:46 2017 +0000
GitHub committed 8a0285fb4 on Wed Nov 8 19:59:26 2017 +0000
Dan Abramov committed a2ec77136 on Wed Nov 8 18:48:19 2017 +0000
GitHub committed c932885e7 on Wed Nov 8 00:02:07 2017 +0000
GitHub committed 94f44aeba on Tue Nov 7 18:09:33 2017 +0000
GitHub committed 05f3ecc3e on Tue Nov 7 16:52:48 2017 +0000
GitHub committed de48ad164 on Tue Nov 7 16:16:46 2017 +0000
```

Git log

Statystyki

```
git shortlog
```

Powyższe polecenie pokaże Nam ile kto zrobił commitów wraz ze wszystkimi commit message. Dane są posortowane alfabetycznie po nazwie autora.

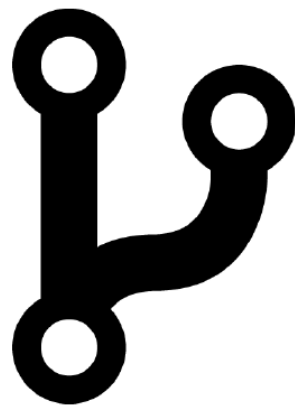
```
git shortlog -n
```

Teraz mamy posortowane informacje według liczby commitów.

```
git shortlog -n -s
```

Teraz widzimy już tylko autora wraz z liczbą jego commitów.

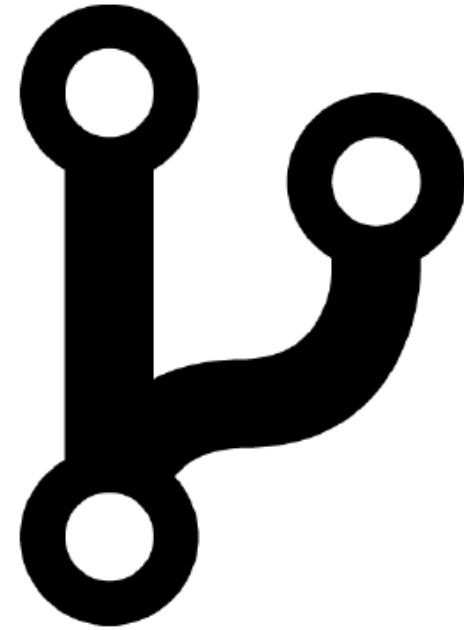
| Branche



Branch (gałąź)

Branch to **wskaźnik** na commit.

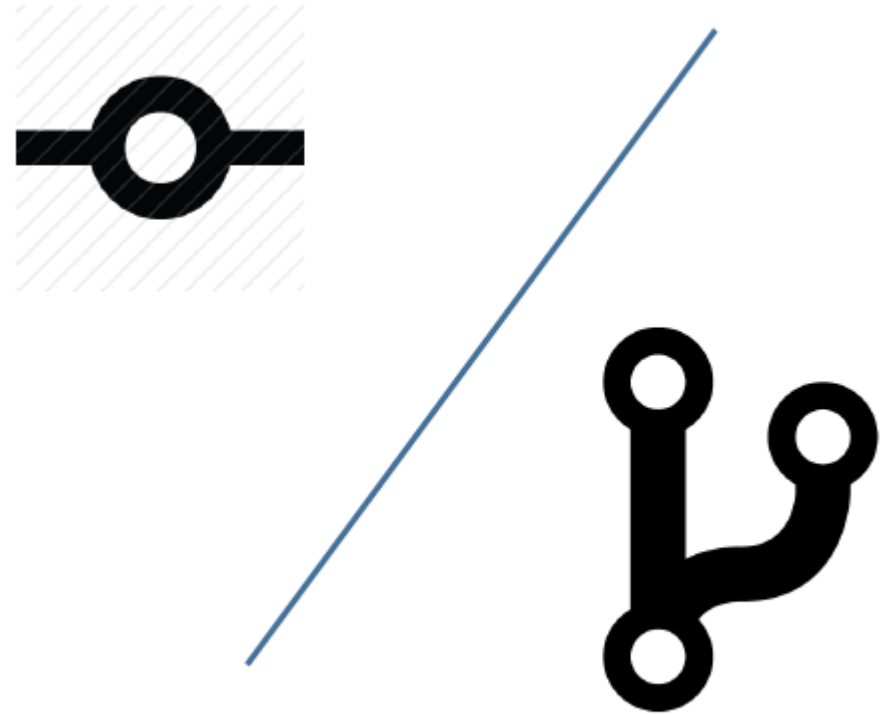
Można go zawsze
dodać/usunąć/zmienić.



Commity a branche

Commity są przechowywane (persistent).

Branche są płynne, to wskaźniki na commity.



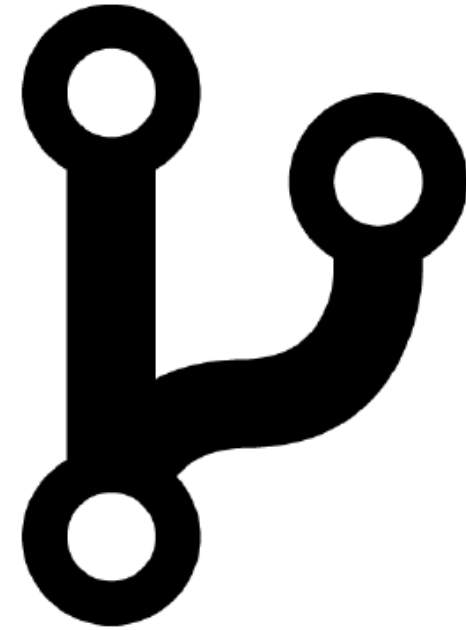
Commit a branch

- Problem pracy równoległej
- Problem nadpisywania sobie zmian
- Problem nieukończonych rzeczy

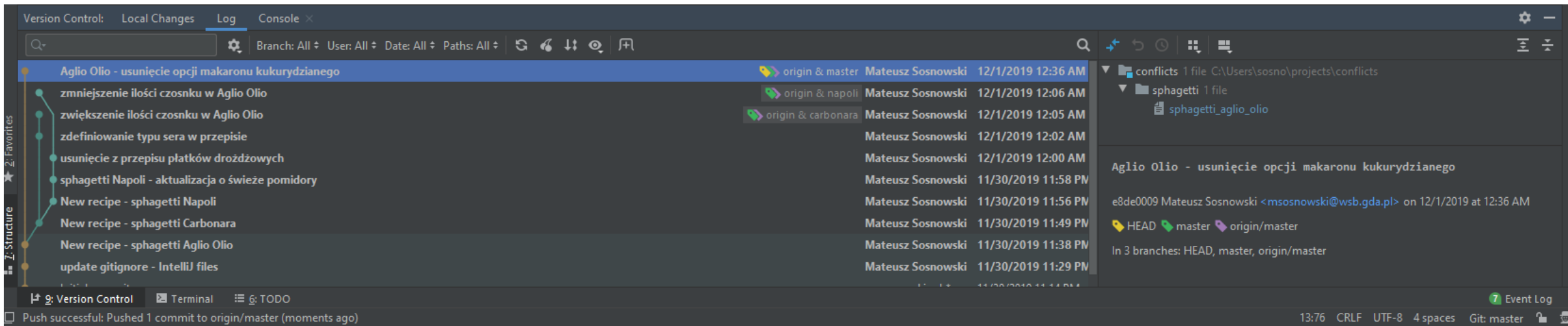


Co oznacza, że "jestem na branchu"

- Aktualnie stan twojego kodu "odpowiada" commitowi na którym jest branch
- Możesz się przełączać pomiędzy branchami



A konkretnie, to obiekt HEAD wskazuje na brancha



Ćwiczenie

IntelliJ - Version control - log

Tworzymy repozytorium.

Tworzymy przynajmniej 2 nowe branche.

W IntelliJ przełączamy się pomiędzy branchami i obserwujemy znacznik HEAD.