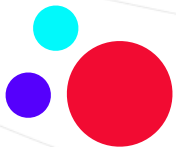


# **GIT**

# System Kontroli wersji

infoShare Academy



# HELLO

## Jakub Wojtach

Senior **full stack** developer





Zacznijmy ten dzień z przytupem!

**Imprezy**

**Podróże**

**VSCode**

**Webstorm**

**Koty**

**Psy**

**Piwo**

**Wino**



# Agenda

- Podstawy teorii systemów kontroli wersji
- Użycie podstawowych komend linii poleceń
- Podstawy konfiguracji GITa
- Podstawy wersjonowania
- Wykorzystanie mechanizmu gałęzi i scalanie zmian
- Rozwiązywanie konfliktu w GIT
- Zrozumienie .git i zastosowanie .gitignore
- GUI Git w VS:Code



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



systemów kontroli wersji



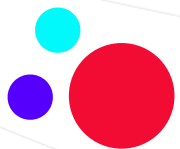
# Problem

- Jak pisać oprogramowanie w wiele osób?
- Jak wrócić do poprzedniej wersji kodu?
- Jak modyfikować te same pliki przez kilka osób?
- Jak pracować nad tym samym projektem na różnych maszynach?



# Systemy kontroli wersji





# Systemy kontroli wersji

**Distributed**(Rozproszone)

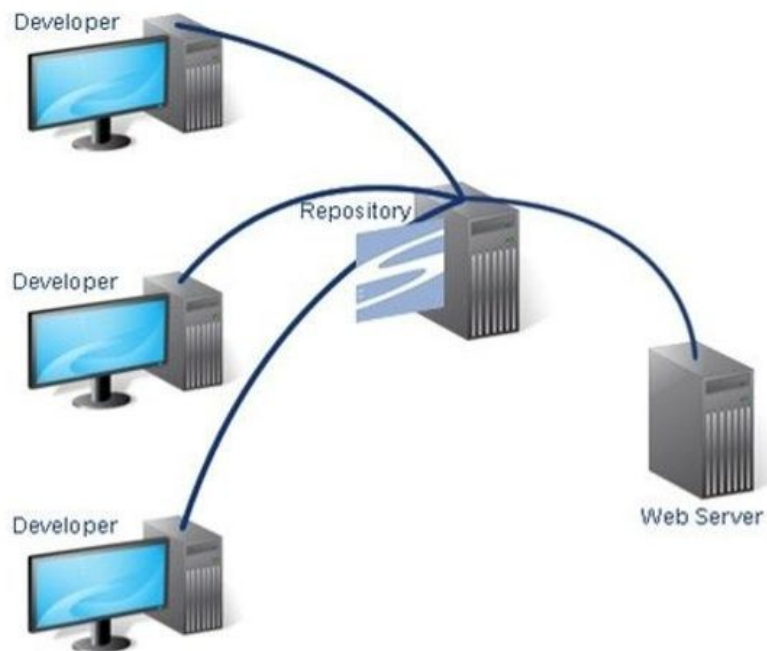


**Client-Server** (centralne)



## Client-server centralne

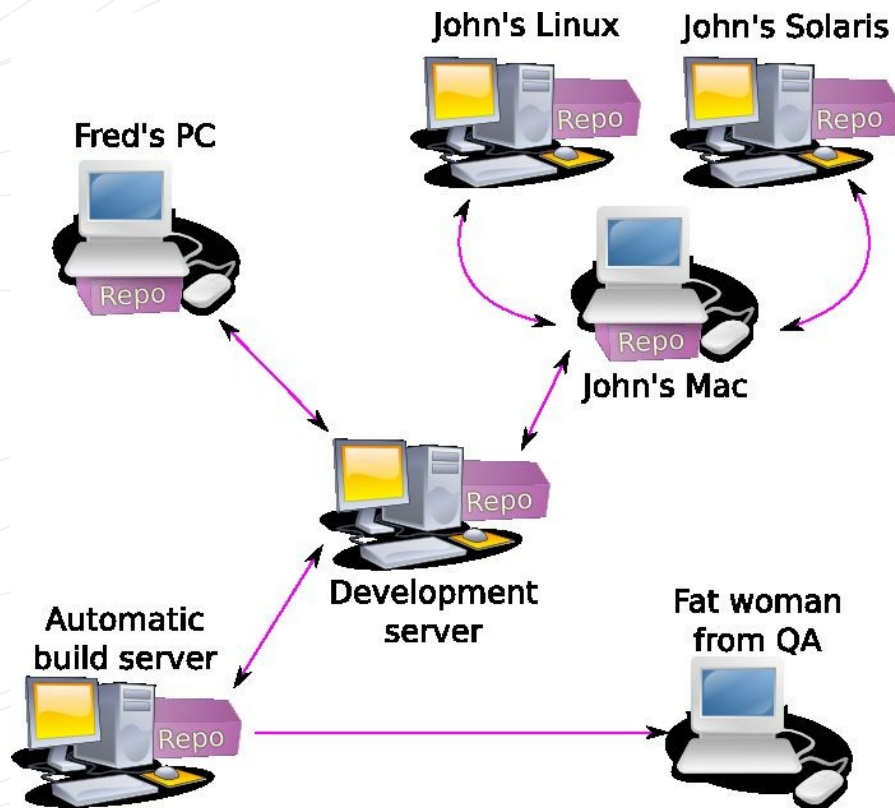
- Tylko jedno repozytorium na zewnętrznym serwerze
- Wersja synchronizowana w tym repozytorium
- Gdy ktoś zepsuje wersję w repo to mamy problem..



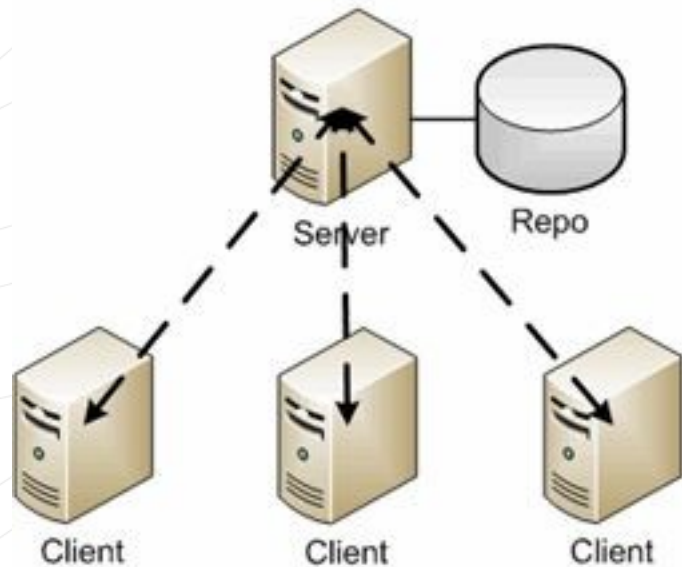


# **Distributed** rozproszone

- Każdy ma lokalną kopię repozytorium
- Jakakolwiek awaria jednego z "serwerów" nie grozi niczym poważnym – mamy tyle kopii ilu użytkowników

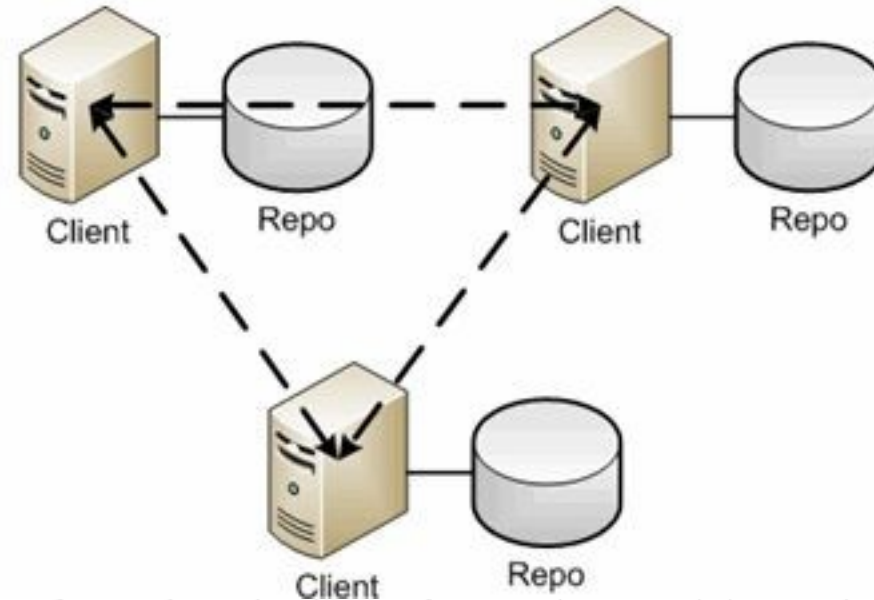


## Traditional



istnieje tylko jedno repozytorium

## Distributed



każdy klient ma lokalne repozytorium



# Zadanie samodzielne

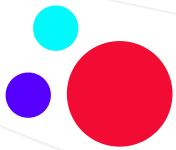


## Najważniejsze cechy

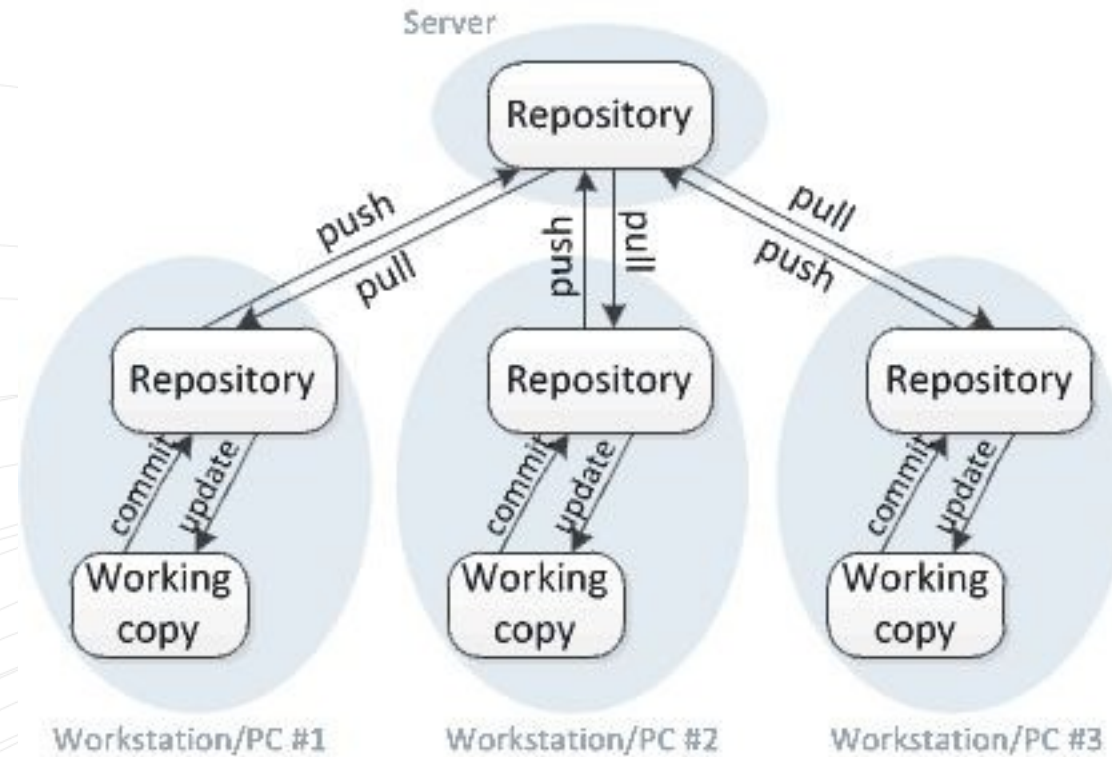
- Darmowy!
- Wydajność i prędkość działania
- Śledzenie zmian, a nie całych plików
- Rozgałęziony proces (łatwość łączenia zmian)
- Praca offline (lokalne repozytorium)
- Aktualny standard



- Od swoich narodzin w 2005 roku, Git ewoluował i ustabilizował się jako narzędzie łatwe w użyciu, jednocześnie zachowując wyżej wymienione cechy. Jest niewiarygodnie szybki, bardzo wydajny przy pracy z dużymi projektami i posiada niezwykły system gałęzi do nieliniowego rozwoju.



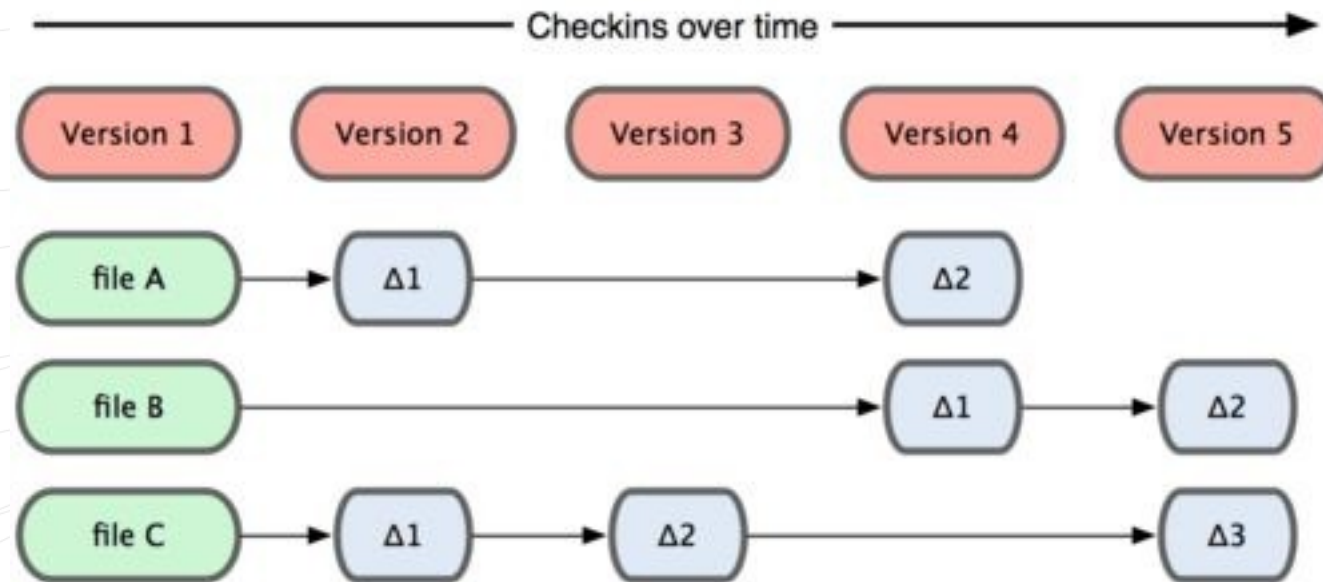
## Distributed version control



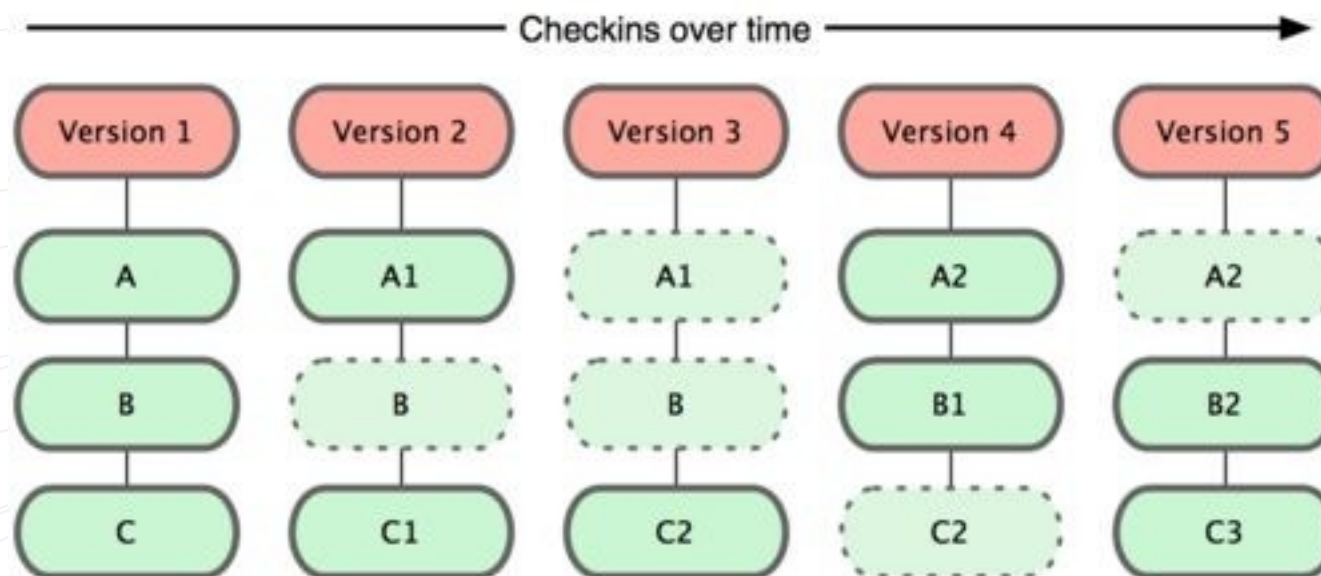


## Snapshots migawki

- GIT traktuje dane jak zestaw migawek (ang. snapshots) małego systemu plików
- Każdy commit tworzy obraz wszystkich plików i przechowuje ich referencje
- Jeśli plik nie zostanie zmieniony, git nie zapisuje go ponownie
- Zapisuje tylko referencję do poprzedniej wersji









komend linii poleceń



- pwd
- ls (-a, -l, -al)
- ll
- cd



# Praca na systemie plików

- mkdir
- touch
- echo
- rm
- cat

- find

- man

# Zadanie samodzielne





# Konfiguracja

W GIT istnieją 3 poziomy konfiguracji:

**project** – działa w obrębie aktualnego projektu (domyślna flaga)

**global** – działa dla wszystkich projektów danego użytkownika

**system** – działa dla wszystkich użytkowników/projektów

```
git config --global user.name "First name and lastname"  
git config --global user.email "example@example.com"  
git config --global credential.helper "cache --timeout=3600"  
git config --global core.editor gedit  
git config --global --list
```

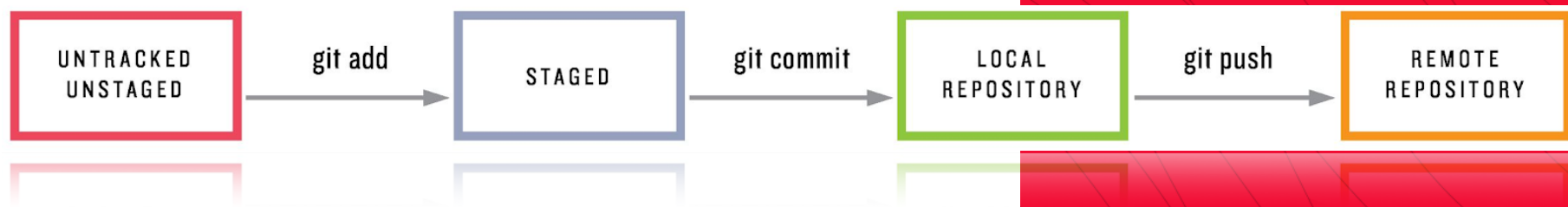


```
git config user.name "Specific project name"  
git config --global user.name "User name"  
git config --system user.name "All users name"  
git config -l --show-origin
```

# Zadanie samodzielne



## Podstawowe operacje



**infoShare**  
ACADEMY



# Operacje Inicjowanie repozytorium

- **Init** – Stwórz nowe lub odtwórz istniejące repozytorium

# Zadanie samodzielne

- **Help** – Dokumentacja gita



## Kilka definicji

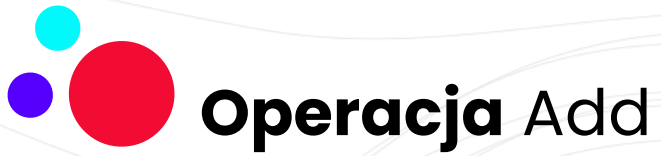
- **Working Directory**
- **Staging Area**
- **Repository**



## Operacje wersjonowanie

- **Add** – dodawanie plików do przechowalni (stage)
- **Restore** – cofnięcie zmian na lokalnym dysku
- **Commit** – zapisuje migawkę (snapshot) przechowalni (stage) jako nowy commit.
- **Revert** – cofnięcie zmian na repozytorium





## Operacja Add

- Dodaje pliki do śledzenia
- Pliki w poczekalni
- "changes to be committed"
- Zapisana wersja pliku z momentu wykonania operacji add

```
git add <parametr> <nazwa pliku lub .>
```



## Operacja Commit

- Zatwierdza zmiany z 'poczekalni'
- Zapisuje różnicę zawartości plików
- Stały i niezmienny
- Musi posiadać wiadomość
- Każdy commit jest unikalny (hash SHA1)



## Cechy dobrego commita

- Jeden commit powinien odpowiadać jednej zmianie
- Treść wiadomości powinna informować jakie zmiany commit zawiera
- Wiadomość powinna być jednoznaczna...
- ...ale jednocześnie nie za długa (np. *Fix typo in class name*)

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

# Zadanie samodzielne



## Operacje analizy wersji i historii plików

- **Diff** – pozwala porównać ze sobą pliki, commity, branche (liniowe wyświetlenie)
- **Log** – wypisuje historię zmian dokonanych na danym branchu w formacie HASH, AUTOR, DATA, COMMIT\_MESSAGE
- **Status** – wyświetla zmiany na branchu wraz z objaśnieniami



## Operacja status

- sprawdza aktualny stan plików
- pokazuje sytuację na aktualnym branchu

```
"nothing to commit, working directory clean" "untracked files:"  
"changes to be committed:" "changes not staged for commit:"
```





## Operacja log

- narzędzie do podglądu istniejących commit'ów
- dużo parametrów formatujących wynik, np.:

```
git log -2
```

```
git log --stat
```

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

- <https://git-scm.com/book/pl/v2/Podstawy-Gita-Podgl%C4%85d-historii-rewizji>

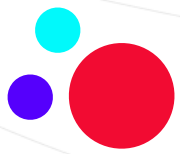


## Operacja diff

- informuje o szczegółach aktualnych zmian
- pokazuje zmiany przed wysłaniem do poczekalni
- wykorzystuje interfejs graficzny do porównania zmian

`git diff`

`git difftool`



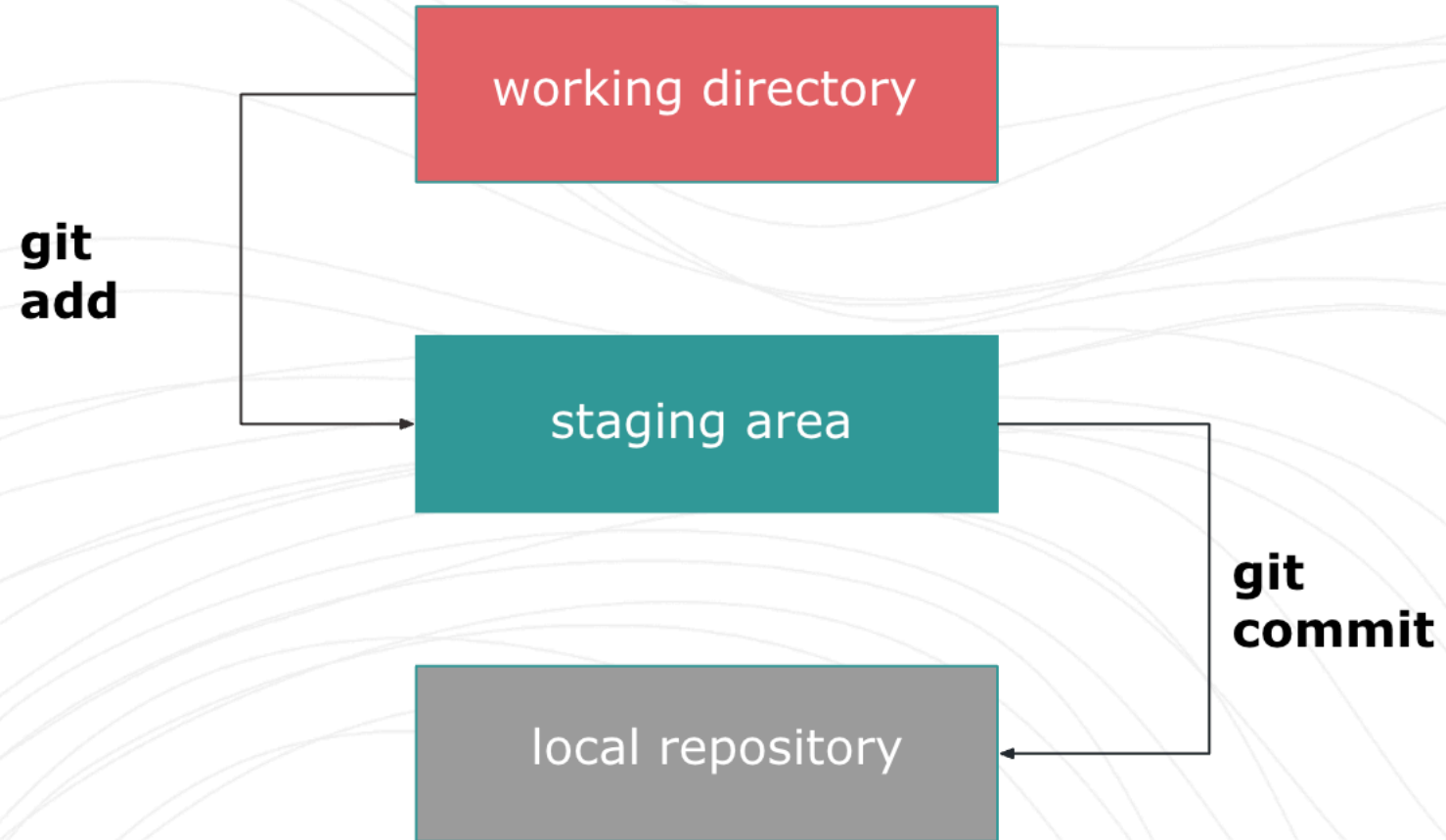
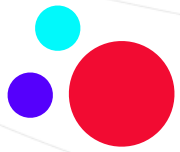
# Operacja status

```
$ git status
On branch feature/JZ3W-22
Your branch is up-to-date with 'origin/feature/JZ3W-22'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   com.infoshareacademy.wave-soft-web/src/main/resources/configuration.properties
        modified:   com.infoshareacademy.wave-soft-web/src/main/resources/scripts/setup-datasource.cli

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   com.infoshareacademy.wave-soft-web/src/main/java/Hellojava.java
        modified:   com.infoshareacademy.wave-soft-web/src/main/java/UserListServlet.java
        modified:   com.infoshareacademy.wave-soft-web/src/main/resources/META-INF/persistence.xml
        modified:   com.infoshareacademy.wave-soft-web/src/main/webapp/findPart.jsp
        modified:   com.infoshareacademy.wave-soft-web/src/main/webapp/userList.jsp
```



# Zadanie samodzielne



## Operacje modyfikacji historii

- **reset** – usuwa pliki z przechowalni (stage)
- **commit – -amend** – zmiana opisu ostatniego commita
- **revert** – tworzy commit wycofujący celem wycofania zmian wprowadzonych w danym commicie



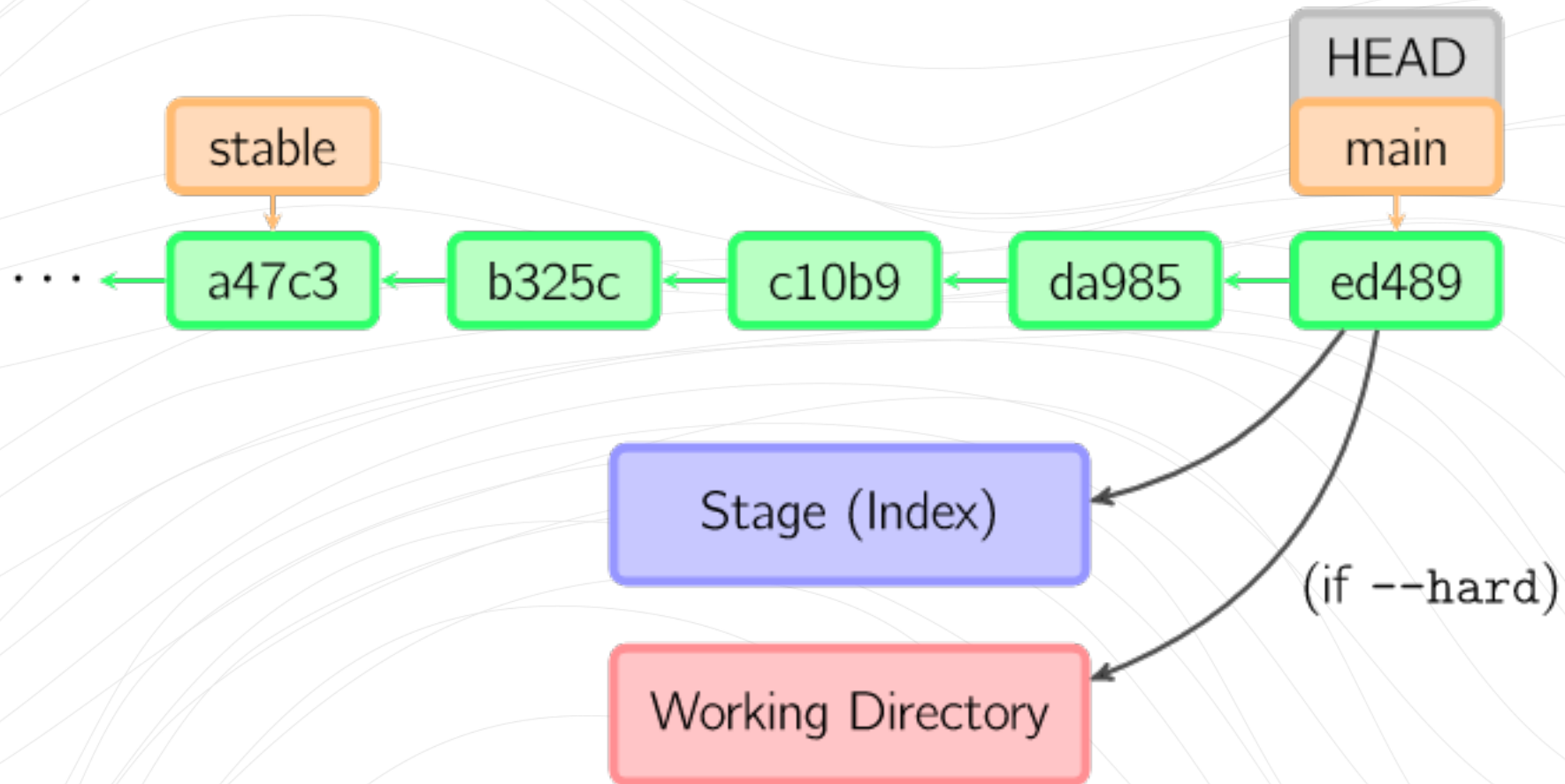


## Operacja Reset

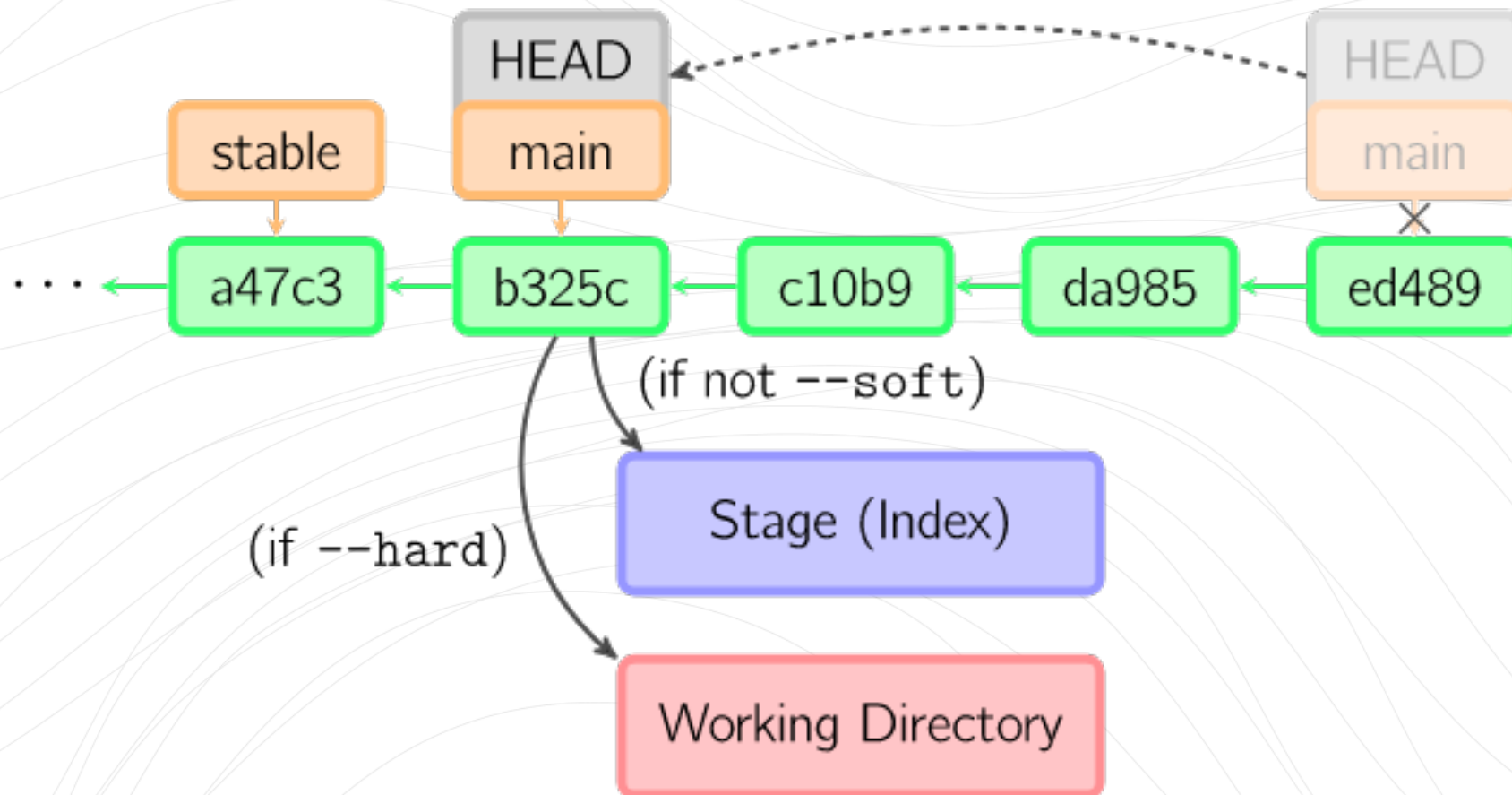
- Komenda reset przenosi aktualną gałąź (branch) do wskazanej pozycji oraz opcjonalnie aktualizuje przechowalnię (stage) oraz katalog roboczy (working directory).
- Jest również używana do kopiowania plików z katalogu git (history) do przechowalni (stage) bez naruszania katalogu roboczego (working directory).



`git reset`



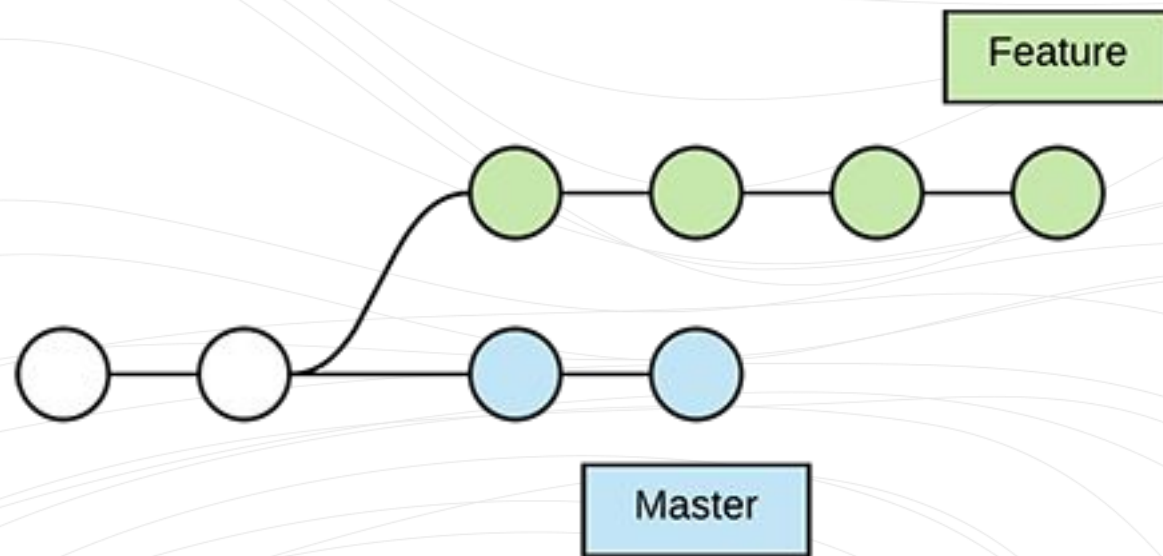
`git reset HEAD~3`



# Zadanie samodzielne



Gałęzie i scalanie zmian

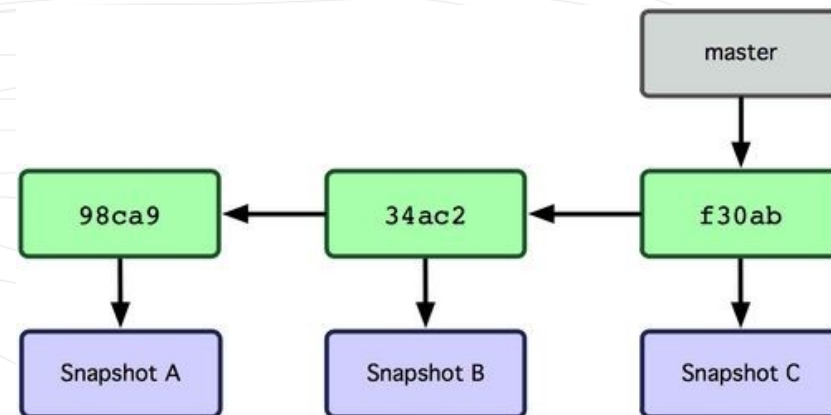
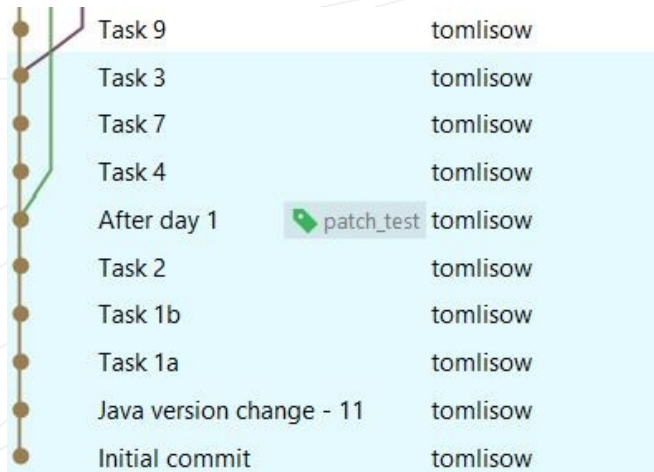




- Specjalny wskaźnik gałęzi
- Wskazuje **aktualną, lokalną** gałąź
- “Jestem na branchu”, czyli **HEAD** wskazuje branch, na którym **aktualnie** pracuję
- Stan kodu odpowiada commit’owi, na który wskazuje HEAD
- Można łatwo przełączać się między branchami
- **DETACHED HEAD** – wskazuje bezpośrednio na commit

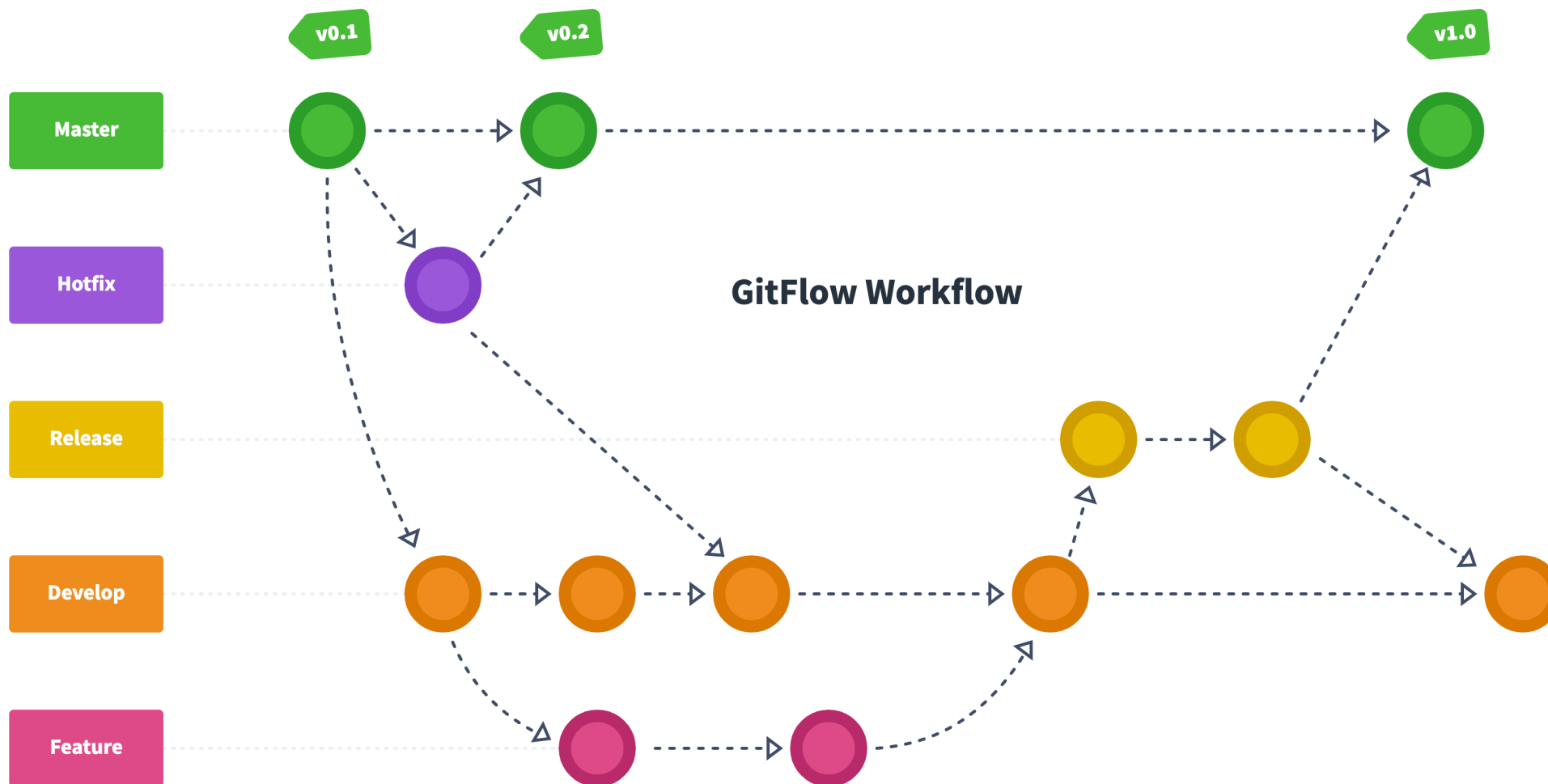
# Branch gałąź

- Wskaźnik na commit
- Pozwala na rozgałęzienie projektu
- Można go zawsze dodać/usunąć/zmienić
- **master/main** - pierwszy i domyślny branch





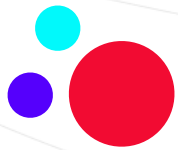
# Wykorzystanie w praktyce git flow



`git branch <nazwa brancha>`  
tworzy branch, ale nie zmienia HEADa

`git checkout -b <nazwa brancha>`  
przełącza na branch z parametru (zmienia HEAD)  
parametr `-b` tworzy branch, jeśli jeszcze nie istnieje

`git switch <nazwa brancha>`  
przełącza na branch z parametru (zmienia HEAD)  
parametr `-c` tworzy branch, jeśli jeszcze nie istnieje  
(*Ekwiwalent `git branch` -> `git switch`*)



**git checkout**

**git switch**

*# to switch from  
one branch to  
another.*

`git checkout <branchname>`

`git switch <branchname>`

*# creates a new  
branch and also  
switches to it.*

`git checkout -b <branchname>`

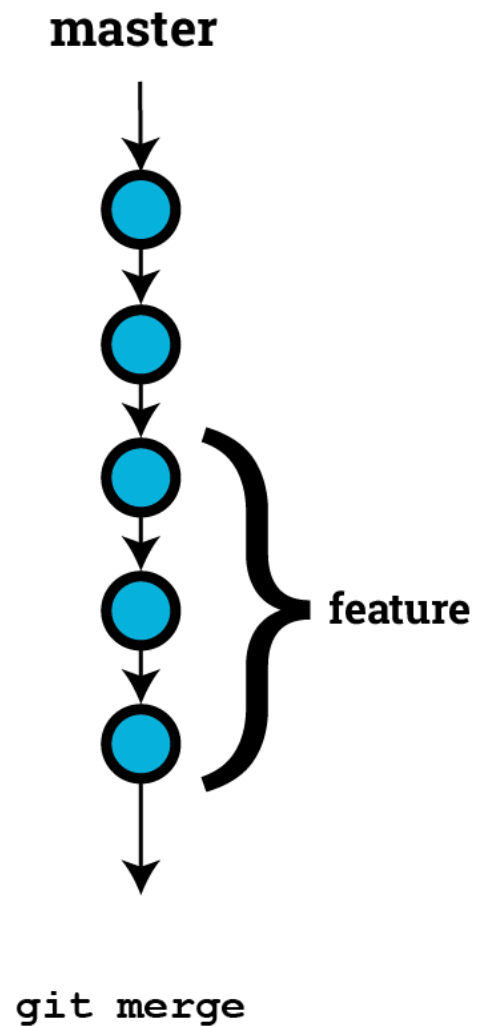
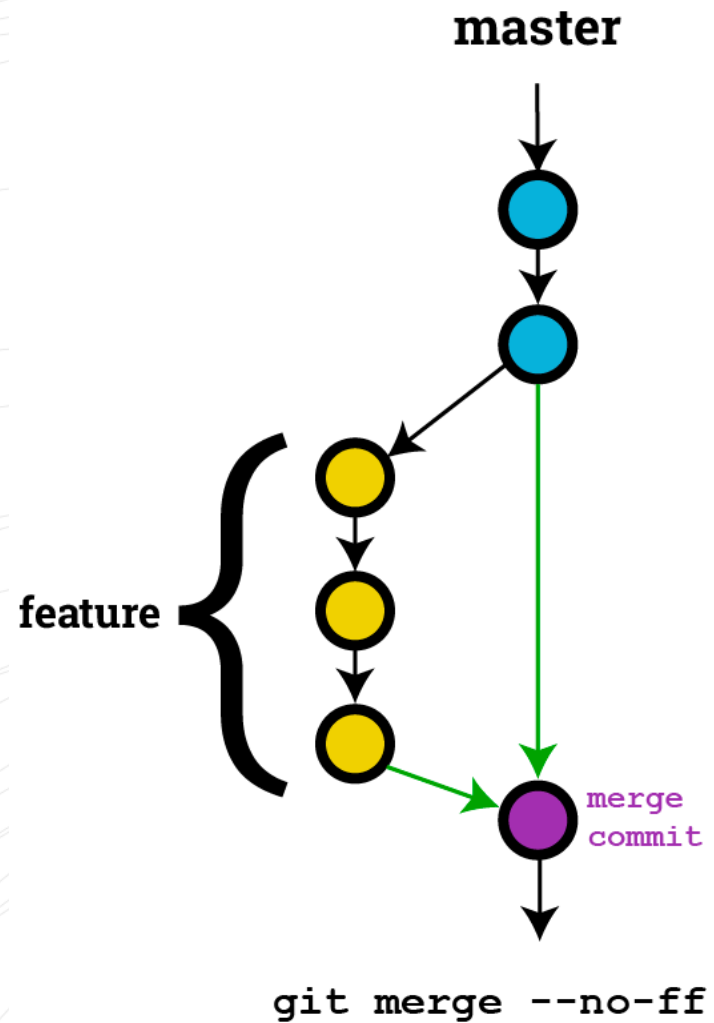
`git switch -c <branchname>`



- próba połączenia kodu z dwóch różnych branchy
- często wykonywany automatycznie (PULL)
- możliwe konflikty, wymagające ręcznego rozwiązania

`git merge <nazwa brancha, który chcemy włączyć>`

# Merge | Merge fast-forward



# **Zadanie** Samodzielnie

# Zadanie Wspólnie





```
objects
refs
config
description
HEAD

de11@DESKTOP-N961NR5 MINGW64 /e/tut_repo (master)
$ cmd
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. All rights reserved.

E:\tut_repo>cd .git
cd .git

E:\tut_repo\.git>tree .
tree .
Folder PATH listing
Volume serial number is CEEC-623D
E:\TUT_REPO\.GIT
.
├── hooks
├── info
├── objects
│   ├── info
│   ├── pack
│   └── refs
│       ├── heads
│       └── tags
└──
```



- mechanizm ukrywania plików/katalogów przed Gitem
- plik .gitignore zawiera listę wzorców
- warto ignorować dane związane z lokalnym środowiskiem lub skompilowany kod

np. \*. [oa] – ignoruje wszystko co kończy się literą 'o' lub 'a'

\*.java – ignoruje wszystkie pliki z rozszerzeniem .java



- puste linie, lub oznaczone '#' są pomijane
- znak '/' precyzuje, że chodzi o katalog
- możliwa negacja reguł poprzez znak '!'

Pełna dokumentacja:

<https://git-scm.com/docs/gitignore>

.gitignore

×

```
1  # Ignore file named `README.txt`  
2  README.txt  
3  
4  # Ignore folder named `output`  
5  output/  
6  
7  # Ignore all .java files  
8  *.java|
```

# Zadanie Indywidualne

# Pytania

# Dziękuję za uwagę

Jakub Wojtach