



NOVEMBER 25, 2020

GIT OD PODSTAW





CZYM JEST

**INSTALACJA
KONFIGURACJA**

**PODSTAWOWE
KOMENDY**

**ZDALNE
REPOZYTORIUM**





CZYM JEST GIT



**Rozproszony system
kontroli wersji**



**Służy do śledzenia
zmian w plikach**



**Narzędzie do
zarządzania
plikami projektu**





GIT od innych systemów VCS

(Version Control System)



przede wszystkim różni się rodzajem przechowywania informacji o plikach. Korzysta on z migawek (nie nadpisuje pliku a tworzy migawkę i przechowuje do niej jedynie referencję)



INSTALACJA

Linux



```
yum install git-core
```

V



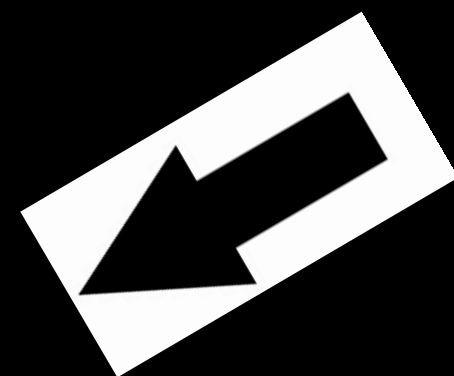
```
apt-get install git
```

Windows

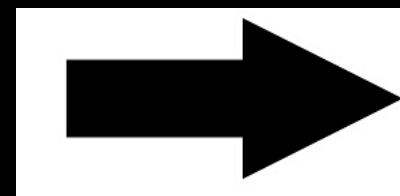


Instalator dostępny
jest na stronie:
[https://git-for-
windows.github.io](https://git-for-windows.github.io)

KONFIGURACJA



```
$ git help init
```



W przypadku gdy brakuje internetu a zapomnieliśmy o czymś istotnym bardzo przydatne staje się polecenie **git help**, które przyjmuje jako parametr komendę gita np:

Polecenie jest również przydatne bez parametru ponieważ wyświetla wtedy listę najczęściej używanych komend gita

KONFIGURACJA

Podstawowa konfiguracja git polega na ustawieniu kilku zmiennych. Najłatwiej to zrobić poprzez polecenie **git config** które pozwala na konfigurację z trzech poziomów **systemu**, **użytkownika** oraz **repozytorium**, jednak dla większości potrzeb wystarczy nam konfiguracja na poziomie użytkownika (globalna).

Aby prawidłowo korzystać z gita warto podać przynajmniej swoją nazwę użytkownika oraz adres e-mail

```
$ git config --global user.name "Kamil Nowak"  
$ git config --global user.email "kamil@example.com"
```

TRZY PRZESTRZENIE DLA PLIKÓW



WORKING DIRECTORY

Miejsce pracy, w którym dokonujemy zmian na plikach.



STAGING AREA

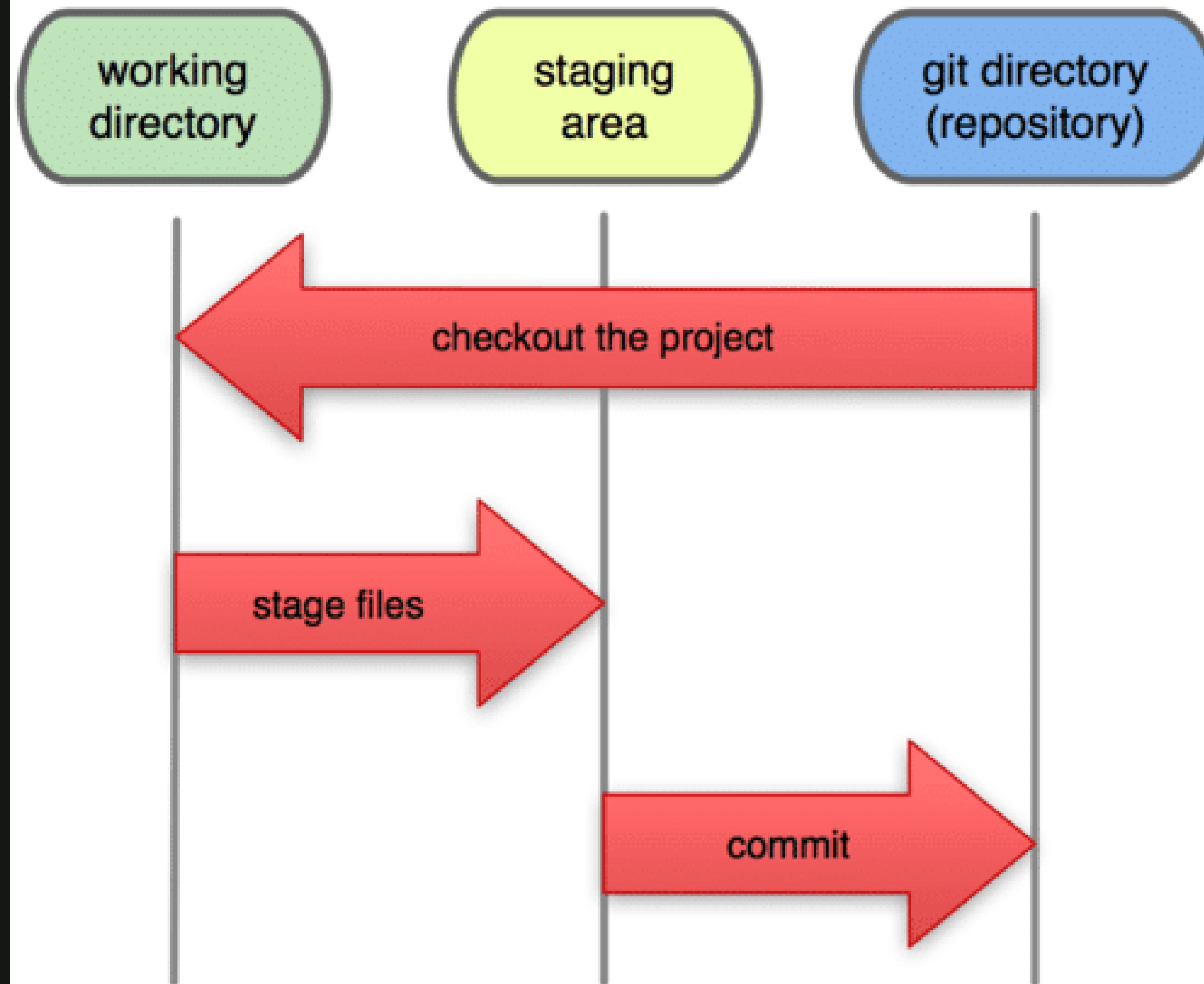
Pliki których zmiany zostały zaakceptowane i które są gotowe, aby je dodać do repozytorium (commit)



GIT DIRECTORY

Miejsce w którym git przechowuje metadane dotyczące projektu oraz pliki które zostały zaakceptowane do repozytorium (committed)

Local Operations





NOWE REPOZYTORIUM LOKALNIE

\$ git init

Tworzenie repozytorium



POBRANIE ISTNIEJĄCEGO REPOZYTORIUM KLONOWANIE

\$ git clone https://example.com/user/repo.git (https)

\$ git clone git@example.com:user/repo.git (ssh)



Jednym z podstawowych poleceń jest *git status* które służy do sprawdzenia stanu plików



Przykładowe wywołanie:

```
$ git status
```

```
On branch master  
nothing to commit, working tree clean
```



Git add

dzięki temu poleceniu możemy dodawać plik lub pliki do przestrzeni staging

- \$ **git add README.md** dodajemy plik README.md
- \$ **git add .** dodajemy wszystkie pliki z aktualnego katalogu
- \$ **git add '*.cpp'** dodajemy wszystkie pliki z rozszerzeniem .cpp
- \$ **git add -A** dodajemy wszystkie pliki z katalogu roboczego

ZATWIERDZANIE ZMIAN

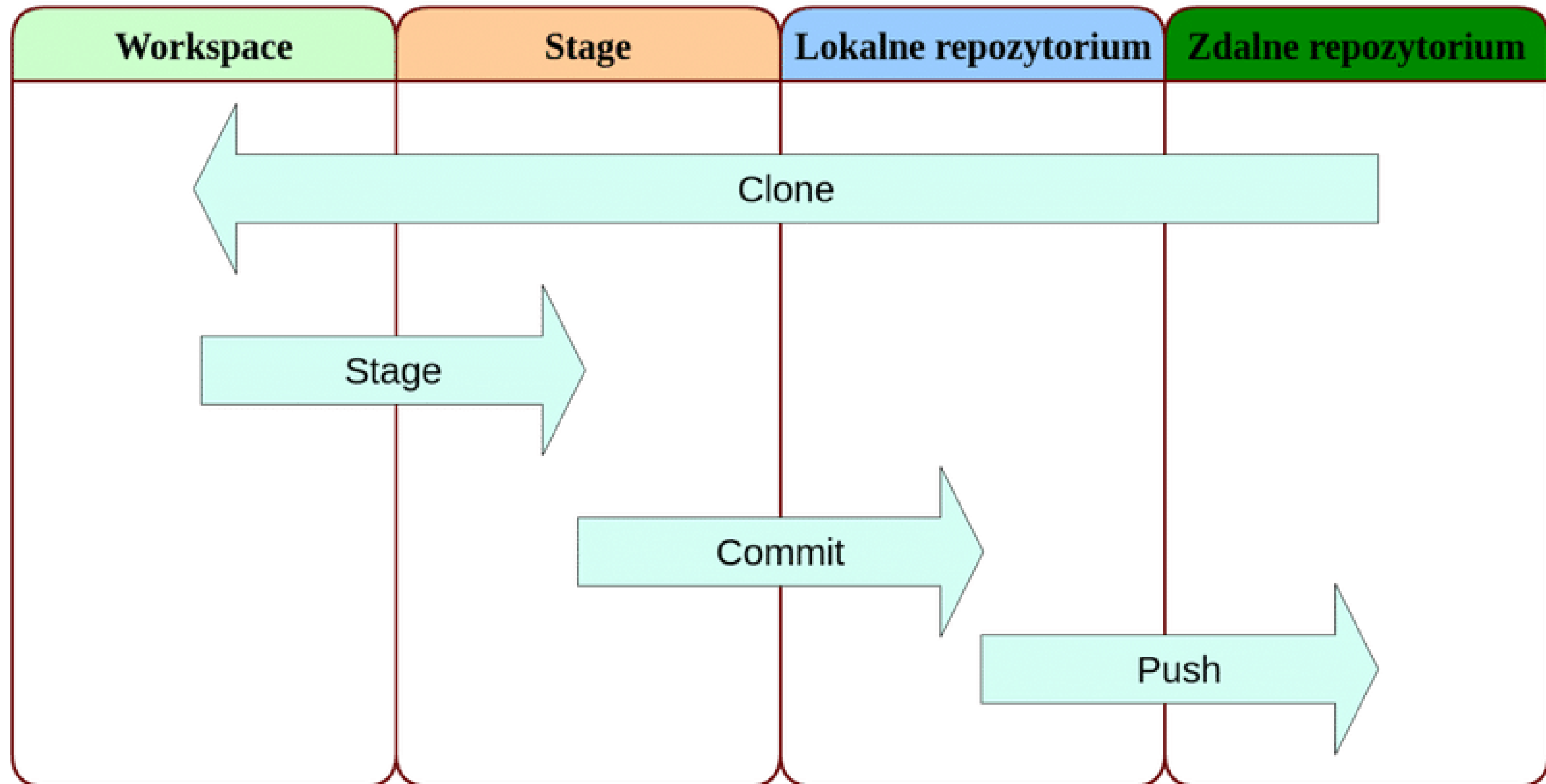
Zmiany znajdujące się w przestrzeni staging mogą zostać zatwierdzone do repozytorium przez polecenie **git commit**, które wymaga określenia opisu zmian (commit message). Istnieje możliwość wyboru edytora tekstu, z którego git będzie korzystać będzie przez zmienną środowiskową \$EDITOR. Można również nadpisać domyślną wartość przed wykonaniem polecenia:

```
$ EDITOR=vim git commit
```

```
$ git commit [-m "opis zmian, etc."] [-a]
```

```
$ git commit -m "Add README.md"
```

Stany GIT



HISTORIA ZMIAN

1. Wyświetlanie historii zmian commitów

\$ git log [--oneline --decorate --graph]

2. Wyświetlanie modyfikowanych plików od ostatniego commita

\$ git status

3. Wyświetlanie zmian w ostatnim commitcie

\$ git show

4. Wyświetlanie zmian w plikach

\$ git diff [--cached | HEAD]



COFANIE ZMIAN

5. Porzucanie zmian w pliku (przed stworzeniem commita)

```
$ git checkout <nazwa_pliku>
```

6. Usunięcie nowo utworzonych plików zachowując zmodyfikowane

```
$ git clean -fd
```

7. Porzucenie zmian (przed commitem) w trybie interaktywnym

```
$ git checkout -p
```



PRACA ZE ZDALNYM REPOZYTORIUM

WYKORZYSTUJĄC POLECENIE GIT
REMOTE MOŻEMY ZOBACZYĆ OBECNIE
SKONFIGUROWANE SERWERY A
DODAJĄC PARAMETR -v WYŚWIETLAMY
ADRES URL PRZYPISANY DO DANEGO
SKRÓTU

origin git@example.com:user/repo.git
(fetch)

\$ git remote -v

origin git@example.com:user/repo.git
(push)

JEŻELI Utworzyliśmy repozytorium za pomocą git init to aby mieć możliwość wysyłania naszych commitów do świata zewnętrznego musimy podpiąć zdalne repozytoria



```
$ git remote add origin git@example.com:user/frepo.git (ssh)
```

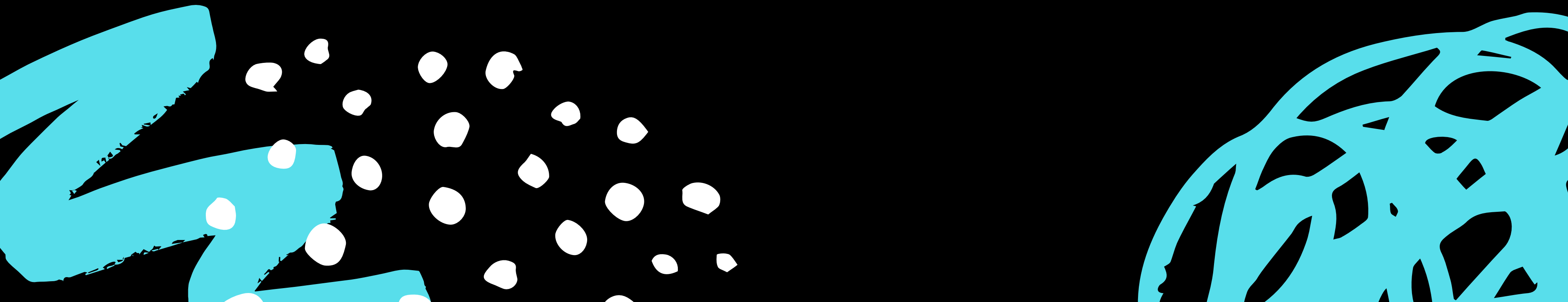


Możemy również nadawać zdalnym repozytoriom inne skróty niż origin

```
$ git remote add backup git@example.com:user/repo.git (ssh)
```

```
$ git fetch [nazwa-  
zdalnego -repozytorium]
```

Polecenie to sięga do zdalnego projektu i pobiera z niego wszystkie dane, których jeszcze nie masz. Po tej operacji, powinieneś mieć już odnośniki do wszystkich zdalnych gałęzi, które możesz teraz scalić z własnymi plikami lub sprawdzić ich zawartość. Tak więc, `git fetch` pobierze każdą nową pracę jaka została wypchnięta na oryginalny serwer od momentu sklonowania go przez ciebie (lub ostatniego pobrania zmian).



PRACA Z REMOTE

WRESZCIE ABY WYSŁAĆ SWOJE ZMIANY DO
ZDALNEGO REPOZYTORIUM KORZYSTAMY
Z GIT PUSH

\$ git push origin master

NATOMIAST ABY POBRAĆ ZMIANY INNYCH OSÓB LUB NA
INNYM URZĄDZENIU UŻYWAMY GIT PULL

\$ git pull



BRANCHE W SKRÓCIE

Branch(ang) - gałąź, służy do pracy nad różnymi wersjami projektu jednocześnie, główna gałąź to tak zwany master.

Gałąź w Gicie została zaimplementowana jako lekki, przesuwalny wskaźnik na miejsce w historii.




Branches

What we Expect

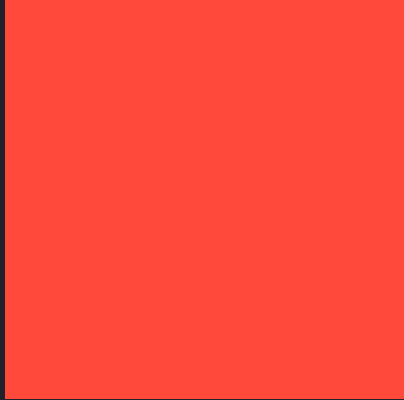


Lista branchy
\$ git branch [-r | -a]

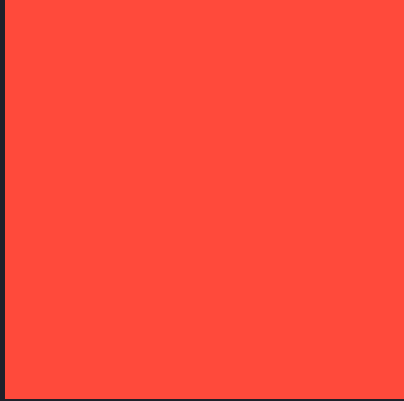


Tworzenie nowego
brancha i przejście na
niego


\$ git checkout -b
<nazwa_brancha>



lub
\$ git branch
<nazwa_brancha>



\$ git checkout
<nazwa_brancha>



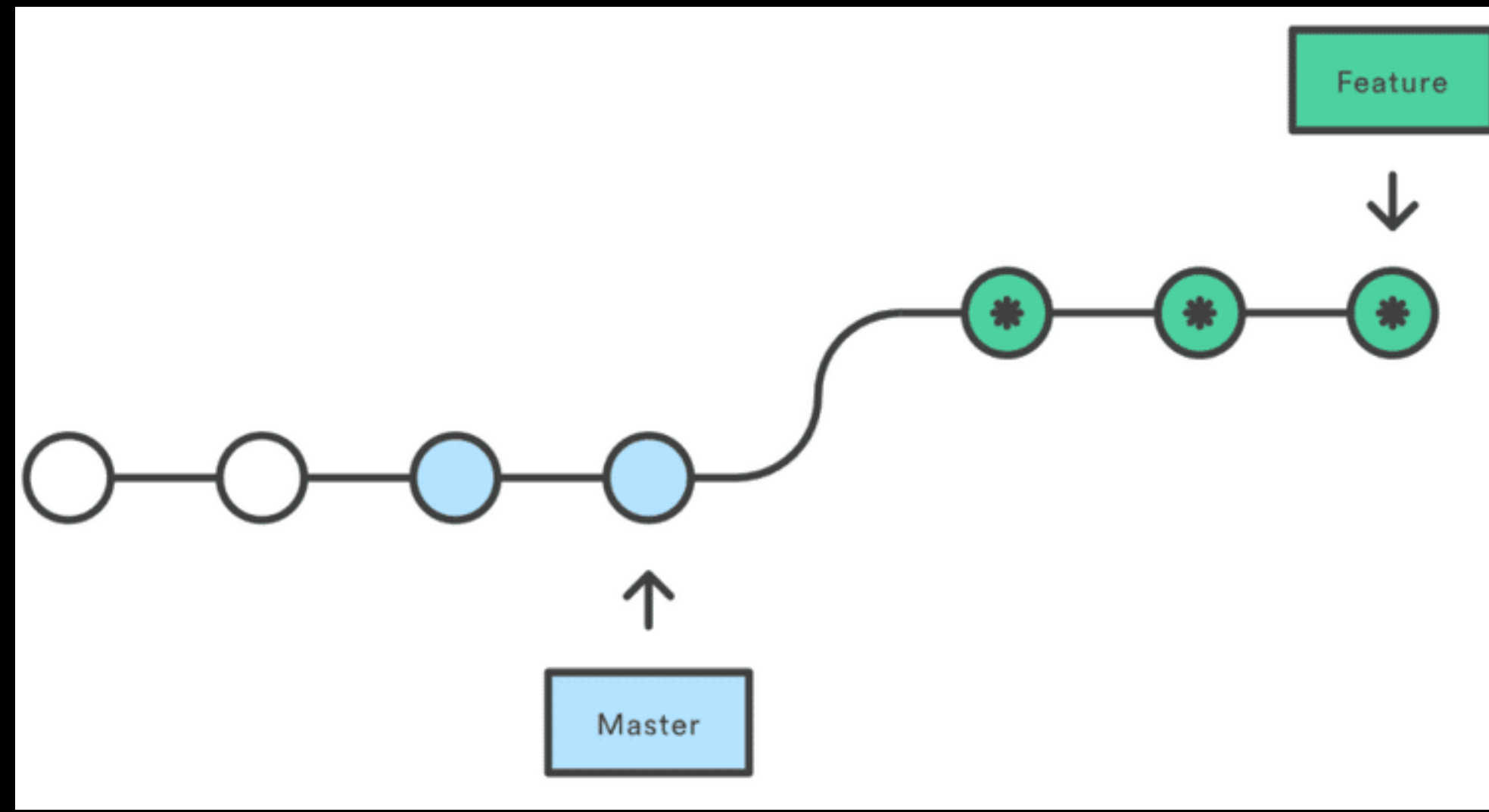
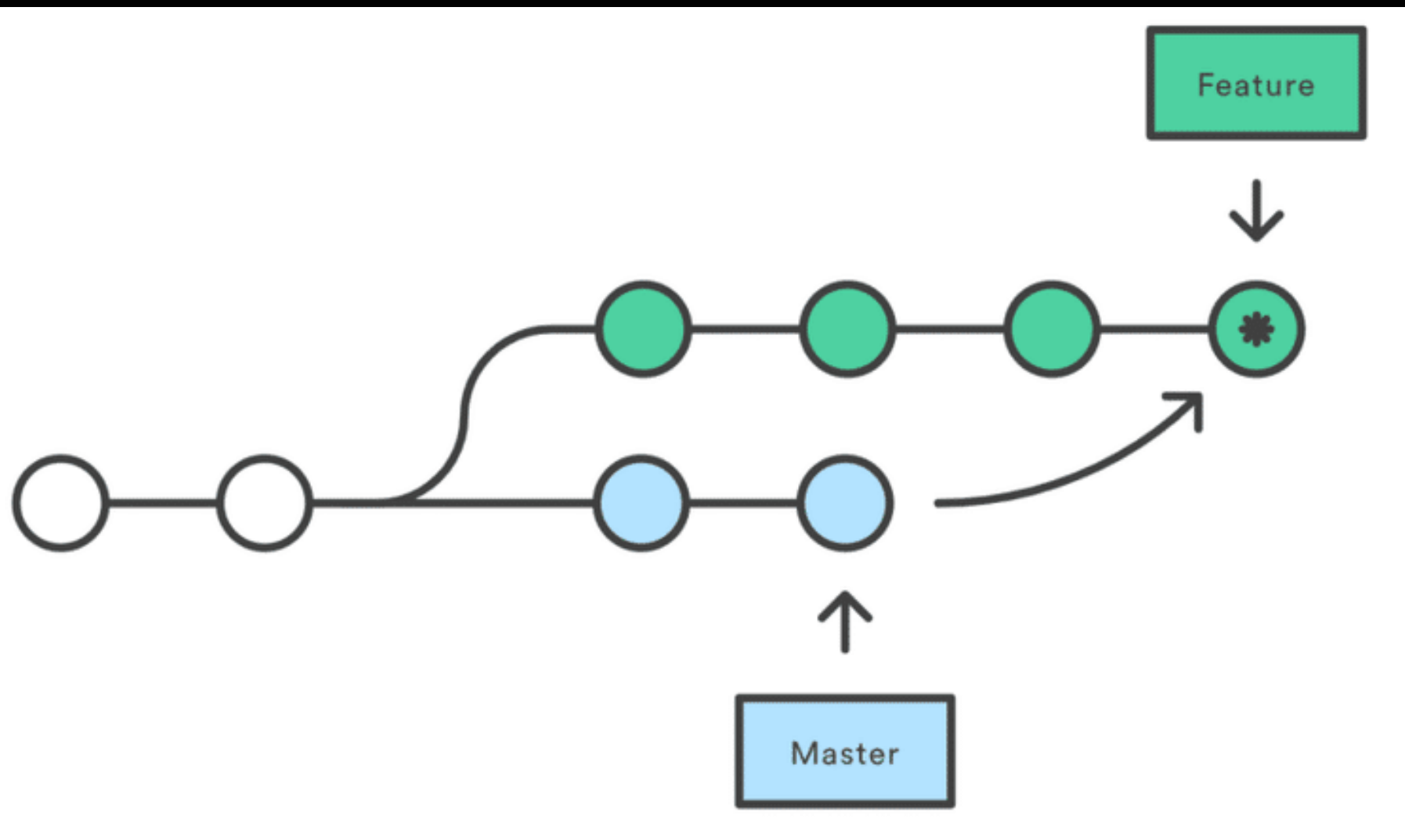
Merge, rebase i cherry-pick, czyli scalanie zmian między branchami

git rebase polega na przeniesieniu danego commita z jednego brancha, na drugi. Niestety, tracimy przez to część historii, bo powstaje złudzenie, że dany commit zawsze był w tej gałęzi.

git merge powoduje scalenie dwóch gałęzi, co jest lepiej widoczne w historii i zapewnia lepszą kontrolę na rozwoju projektu.

Samo użycie git merge może spowodować, że git sam scali zmiany (czyli nastąpi fast-forward). Dlatego lepiej w takim przypadku wykonać polecenie **git merge --no-ff**, czyli merge bez fast-forward, który wymusi na gicie stworzenie oddzielnego commita, opisującego co, skąd zostało zmergowane.

MERGE VS. REBASE



Dziękujemy za uwagę!