



GIT



UTWÓRZ NOWE REPOZYTORIUM

git init - tworzy nowe repozytorium. Utwórz katalog, wejdź do niego i wykonaj **git init**, aby utworzyć nowe repozytorium



POBIERZ REPOZYTORIUM

git clone /path/to/repository (pobierz repozytorium)
utwórz kopię roboczą lokalnego repozytorium

korzystając ze zdalnego serwera użyj polecenia:

git clone username@host:/path/to/repository

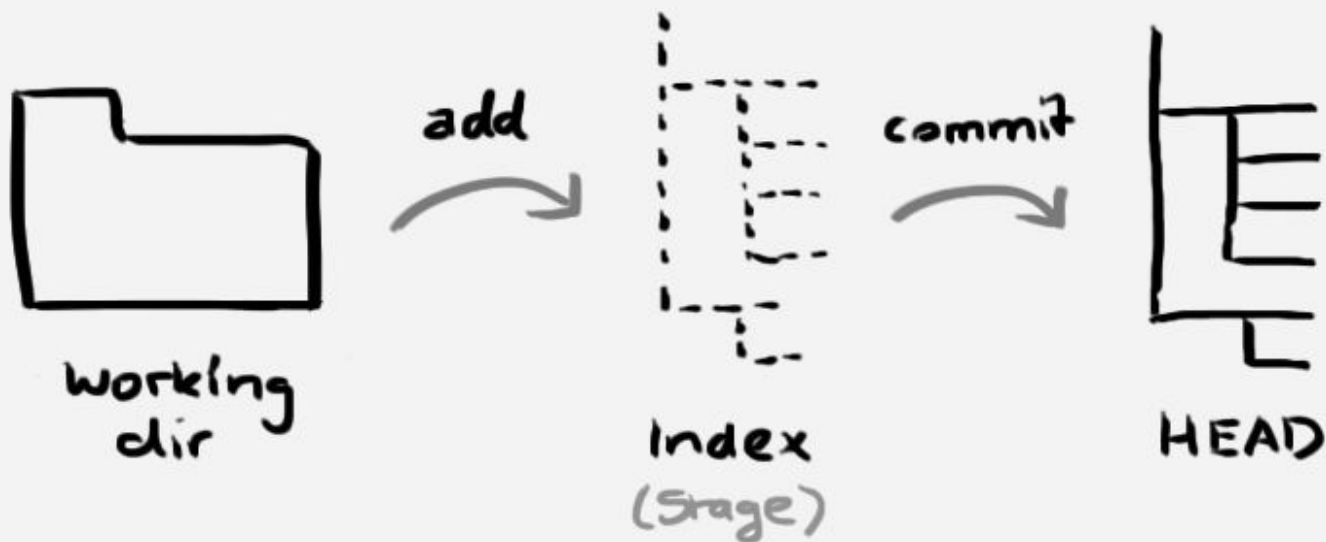


PRZEPŁYW PRACY

Twoje lokalne repozytorium składa się z trzech “drzew” zarządzanych przez git.

1. **Katalog Roboczy**, który przechowuje bieżące pliki.
2. **Index (stage)**, które działa jak poczekalnia
3. **HEAD**, które wskazują na ostatni utworzony commit.

PRZEPŁYW PRACY





ADD & COMMIT

Możesz zaproponować zmiany (dodać je do **Index/stage**) używając:

git add <filename>

git add * dodaje wszystko

Aby rzeczywiście zatwierdzić te zmiany użyj:

git commit -m "Commit message"

teraz plik jest zatwierdzony w **HEAD**, ale nie w zdalnym repozytorium.



WYSYŁANIE ZMIAN

Teraz Twoje zmiany są w **HEAD** twojej kopii roboczej, aby wysłać je do zdalnego repozytorium, wykonaj **git push origin master**

Zmień *master* na dowolną gałąź, której zmiany wysyłasz.

Jeśli nie sklonowałeś istniejącego repozytorium i chcesz połączyć je ze zdalnym serwerem, musisz go dodać poprzez **git remote add origin <server>**

Teraz masz możliwość wysyłania zmian na wskazany serwer.



BRANCHING

Gałęzie są używane do rozwijania funkcjonalności odizolowanych od siebie.

Gałąź *master* jest domyślną gałęzią kiedy tworzysz repozytorium. Używaj innych gałęzi do rozwoju projektu, a kiedy skończysz scalaj je z powrotem z gałęzią główną.

BRANCHING





BRANCHING

Aby utworzyć nową gałąź o nazwie `feature_x` i przełączyć się na nią wykonaj **git checkout -b feature_x**

przełącz się z powrotem na master **git checkout master**

i usuń gałąź **git branch -d feature_x**

gałąź nie jest dostępna dla innych dopóki nie wyślesz jej do zdalnego repozytorium **git push origin <branch>**



AKTUALIZACJA I SCALANIE

aby zaktualizować lokalne repozytorium do ostatniego commita, wykonaj **git pull**

w swoim katalogu aby pobrać (*fetch*) i scalić (*merge*) zdalne zmiany.

aby scalić inną gałąź z gałęzią aktywną (np. master), użyj **git merge <branch>** w obu przypadkach git próbuje scalić zmiany automatycznie. Niestety nie zawsze jest to możliwe i powoduje *konflikty*.



SCALANIE - przykład

Jesteś na branchu *praca*, jednak musisz naprawić usterkę.

Jeśli z mastera przeczuciłeś się na brancha np. *usterka*, wprowadziłeś konieczne zmiany, które przeszły testy i chcesz scalić z gałęzią master, wykonaj polecenie **git merge**.

Przerzuć się na mastera: **git checkout master**

I scal branch z masterem **git merge usterka**.

Usuń niepotrzebny branch **git branch -d usterka**.



cd

Teraz możesz wrócić do wcześniejszego brancha na którym pracowałeś, jednak zmiany które wprowadziłeś w *usterka*, nie są uwzględnione w plikach gałęzi *praca*.

Jeśli ich potrzebujesz możesz scalić mastera z branchem *praca* używając **git merge master** lub możesz poczekać na wszystkie zmiany z gałęzi *praca* i przenieś je wszystkie na mastera. Na masterze użyć

git checkout master

git merge praca)



KONFLIKTY

Podczas scalania mogą wystąpić konflikty

#CONFLICT (content): Merge conflict in index.html

#Automatic merge failed; fix conflicts and then commit the result.



KONFLIKTY

Git nie zatwierdził automatycznie zmiany scalającej. Jeżeli chcesz zobaczyć, które pliki nie zostały scalone wykonaj **git status**

```
[master*]$ git status
```

```
index.html: needs merge
```

```
# On branch master
```

```
# 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)
```

```
#
```

```
# unmerged:  index.html
```



KONFLIKTY

To co spowodowało konflikty jest tutaj pod **unmerged**

Wchodząc do pliku wszystko co było w master będzie pod <<<**HEAD** oraz powyżej =====


A wersja z gałęzi *praca* to wszystko poniżej do >>>>> **praca:**
index.html

Teraz dodaj pliki **git add, git commit**



KONFLIKTY

Możesz je ręcznie edytować poprzez edycje plików wskazanych przez git.
Po zmianie musisz oznaczyć je jako scalone poprzez **git add <filename>**
Przed scaleniem zmian, możesz je obejrzeć używając **git diff**
<source_branch> <target_branch>



git diff pokazuje co zmieniłeś (dokładnie linijki i zawartość pliku) ale nie wysłałeś jeszcze do poczekalni. **git diff --cached** aby zobaczyć zmiany wysłane już do poczekalni

git commit -a (skrót od commitowania + dodawania pliku do poczekalni, dot. każdego zmienionego pliku, który jest już śledzony)

git commit -a -m (daję od razu możliwość wpisania treści commitu)



LOG

Możesz przeglądać historię repozytorium w najprostszej formie używając **git log**

Możesz dodawać dużo parametrów, aby uzyskać to co potrzebujesz.
Aby zobaczyć commity konkretnego autora: **git log --author=bob**

aby zobaczyć bardziej zwarty rezultat, gdzie commit jest pojedynczą linią:
git log --pretty=oneline



LOG

Aby zobaczyć drzewo w ASCII art wszystkich gałęzi opatrzone ich nazwami oraz nazwami tagów: **git log --graph --oneline --decorate --all**

Zobacz tylko te pliki które zostały zmienione **git log --name-status**

Aby zobaczyć wszystkie z możliwych parametrów, których możesz użyć sprawdź

git log --help



WYCOFAJ LOKALNE ZMIANY

Jeśli coś pójdzie nie tak, możesz wycofać lokalne zmiany poprzez **git checkout -- <filename>** zastępując zmiany w katalogu roboczym ostatnią zawartością **HEAD**.

Zmiany które zostały dodane do **index/stage**, tak jak nowe pliki zostaną już zachowane.



WYCOFAJ LOKALNE ZMIANY

Jeśli zamiast tego chcesz porzucić wszystkie lokalne zmiany i commity pobierz ostatni historię z serwera i ustaw na nią swoją gałąź lokalną

git fetch origin

git reset --hard origin/master

git checkout HEAD -- files kopiuje *files* z ostatniego commitu do obu: **stage(index)** oraz do katalogu roboczego(**working directory**)



COFANIE ZMIAN -- amend

```
git commit -m "initial commit"
```

```
git add forgotten_file
```

```
git commit --amend
```

Druga operacja commit zastąpi wynik pierwszej

Jeżeli nic nie zmieniłeś po pierwszym git commit, to polecenie **git commit --amend** (popraw) nic nie zmieni, ale będziesz miał możliwość modyfikacji notki.



USUWANIE PLIKU Z POCZEKALNI

```
$ git add .  
$ git status  
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
#       modified:   README.txt  
#       modified:   benchmarks.rb
```

Aby usunąć plik z poczekalni użyj **git reset HEAD <file>**



USUWANIE PLIKÓW

Aby usunąć plik z Gita, należy go najpierw wyrzucić ze zbioru plików śledzonych, a następnie zatwierdzić zmiany. Służy do tego polecenie **git rm**, które dodatkowo usuwa plik z katalogu roboczego. Nie zobaczysz go już zatem w sekcji plików nieśledzonych przy następnej okazji.

Po użyciu pierwszego **git rm <filename>** plik ląduje poza poczekalnią. W dalszej kolejności **git rm** doda do poczekalni opcję usunięcia pliku

git rm --cached <filename> jeśli chcesz trzymać plik na dysku, ale nie chcesz, żeby Git go śledził



ZMIANA NAZWY PLIKU W REPO

git mv file_from file_to pozwala na zmianę nazwy pliku w repozytorium