Git to rozproszony i najpopularniejszy system kontroli wersji.

Git jest darmowy oraz open source. Git jest lubiany, szybki, wydajny, rozbudowany.

System kontroli wersji inaczej VSC(ang. Version control system)

VSC służy do zarządzania projektem w czasie. Tworzy historię projektu, pozwala na współpracę wielu osób i łącznie ich pracy. Jest narzędziem, które umożliwia tworzenie repozytoriów.

Każdy deweloper (czy użytkownik danego repozytorium gita) ma własne kompletne repozytorium na swoim komputerze(repozytorium lokalne). Najczęściej istnieje też repozytorium, które określa się jako główne, centralne, czy zdalne (bo znajduje się na serwerze, którym często jest platforma GitHub)

Co to repozytorium?

To historia stanów projektu, obejmująca wszystkie commity i stworzone w ich wyniku snapshoty oraz relację między nimi(kolejność). Repozytorium jest przechowywane w ukrytym katalogu .git

Commit – zapisanie w repozytorium informacji zawartych w commicie i utworzenie nowego snapshota. Każdy commit jest obiektem w bazie danych repozytorium git.

Co robi commit:

* Tworzy obiekt commita w bazie
* Tworzy informacje o nowym commicie w gałęzi, w której jesteśmy
* Ustawia HEAD na najnowszą zmianę

Opisy commitów:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeśli chcesz dodać dodatkowy tekst komentarza w nowej linii to w konsoli dajesz shift+enter

Commity dobrze jest robić w takich odstępach żeby przy opisywaniu commita nie przekroczyć 60 znaków

Co to snapshot?

obraz aktualnego stan projektu, każdy snapshot zawiera pełen obraz projektu a nie tylko listę zmian. Jeśli coś nie nie zmieniło to zawiera referencje do tej niezmienionej wersji.

Snapshot można traktować jako synonim commitu.

Trzy obszary repozytorium:

* Working directory(tree) – katalog roboczy – nasze pliki i folder. To tam jest plik .git
* Staging area(index) – obszar zawierający pliki dla których chcemy wykonać commit. Index po commicie nie jest pusty, zawiera wszystkie pliki projektu który zozstał dodany do commita, ale widzimy tego za pomocą git status, bo te pliki od razu po commicie są niezmienione. Czyli pliki w commicie i w indexie są identyczne.
* Git folder (git repository) – stricte repozytorium, katalog przechowujący repozytorium

Co to github?

platforma do zdalnego tworzenia i przechowywania repozytorium projektu, pełni rolę backupu

jak z poziomu konsoli otworzyć plik w edytorze kodu?

code index.html

inicjalizacja repozytorium

git init (domyślnie tworzone jest w folderze w którym wykonaliśmy polecenie)

Możemy stworzyć repo w folderze, gdzie są już pliki. Pliki te będą miały wtedy status początkowy nieśledzone

Możemy też podać ścieżkę, żeby nie tworzyć repo w aktualnym folderze, jeśli folderu ze ścieżki nie ma to zostanie on stworzony

-h przy poleceniu gita pokazuje nam dostępne parametry do użycia

Git clone – alternatywa dla git init, kopiujemy repo zamiast je tworzyć od zera. Klonowanie to tworzenie nowego repo, które jest identyczne

git status

wyświetla status katalogu roboczego, pokazuje nowe pliki i pliki w jakiś sposób zmodyfikowane

przypisanie nazwy użytkownika do gita

git config --global user.name "xyz"

przypisanie maila użytkownika do gita

git config --global user.email xyz

przypisanie maila użytkownika do gita

git config --global user.email xyz

git config --unset user.email – usuwa maila lub nazwę użytkownika z opcją --global robi to globalnie nie tylko do obecnego repo

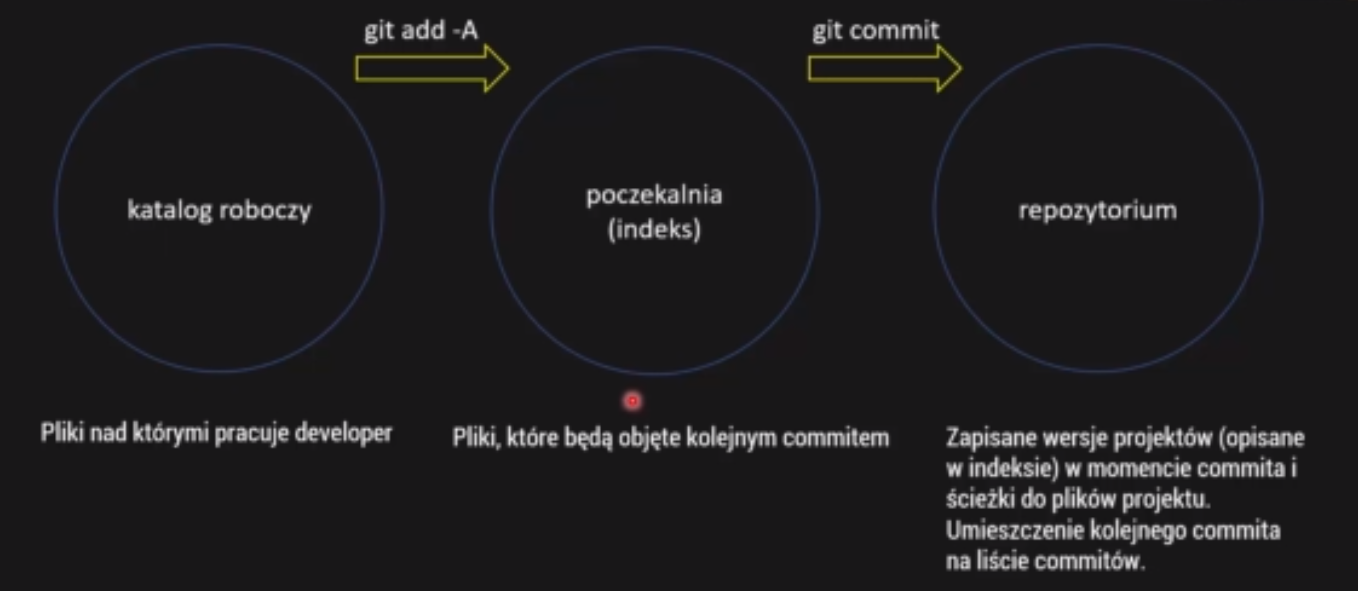
git config --global core.editor – jaki editor jest ustawiony globalnie dla git

git config --list – lista ustawień dla danego repozytorium

C:\Users\username\.gitconfig – plik z globalną konfiguracją

-h przy poleceniu gita pokazuje nam dostępne parametry do użycia

--help – włącza osobną stronę z dokumentacją konkretnej komendy przy której dodaliśmy tę opcję (np. git init --help)



Git add – dodajemy do staging area pliki z katalogu roboczego które mają być uwzględnione w kolejnym commicie.

Żeby dodać wszystkie zmiany za jedym razem możesz użyć git add --all lub -A lub .

Git commit -m „opis zmian w commicie”

Pierwszy commit dobrze nazwać initial commit

Git przechowuje dane jako obiekty posiadające swój hash(identyfikator obiektu). Identyfikator dotyczy commitów czy plików. Taka sama treść(i taki sam zapis) da taki sam identyfikatorm, nawet jeśli nazwa pliku będzie inna.

4 stany plików w repozytorium GIT:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Git checkout nazwa pliku – usuwa ostatnią zmianę ze stage area

Git diff – pokazuje różnice w linijkach kodu

Można bezpośrednio przenieść zmodyfikowany plik do commitu bez dodawania tej zmiany do stage area

Git commit -a --all – dodaje wszytkie śledzone pliki do commitu które zmodyfikowaliśmy

Git commit --amend – poprawki w ostatnim commicie. Zmiany w zawartości i message, nowe metadane też. Usuwa commita i na jego miejsce dodaje poprawionego.

Git commit --amend -m ‘msg’ – zmiana tylko komentarza

Git rm –cached nazwa-pliku – usuwa plik z indexu i przywraca jego poprzednią zawartość

Git log – historia commitów od najnowszego znajdującego się w repo

Jeśli chesz wyjść z widoku loga to naciśnij ‘q’

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Git log --stat – które pliki zmienił commit i w jaki sposób, wyświetla listę commitów jak git log

Git show – co się zmieniło w najnowszym commicie. Można też wskazać o jaki commit nam chodzi trzeba podać hasha commitu albo przynajmniej 7 pierwszych znaków hasha.

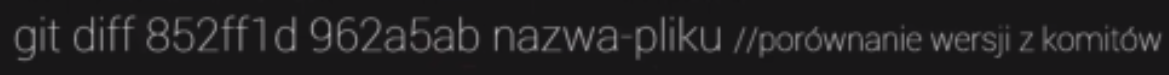
Obraz zawierający tekst

Opis wygenerowany automatycznie

Git diff – pokazuje domyślnie różnicę między plikami – wersją zmodyfikowaną w katalogu roboczym a wersją w indeksie

Git diff --cached – różnica między plikami w staging area a tymi z repo(HEAD – czyli ostatni commit w gałęzi). --staged robi dokładnie to samo.

Można też porównać pliki w różnych commitach:



Git rm plik

Obraz zawierający tekst

Opis wygenerowany automatycznie

Git rm --cached plik – usuwa pliki z indeksu, ale nie z katalogu roboczego(po tym poleceniu plik ma stan nieśledzony)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Git mv index.txt index.html – zmiana nazwy pliku. Jest to zadanie renamed w git , czyli usunięcie pliku i dodanie nowego

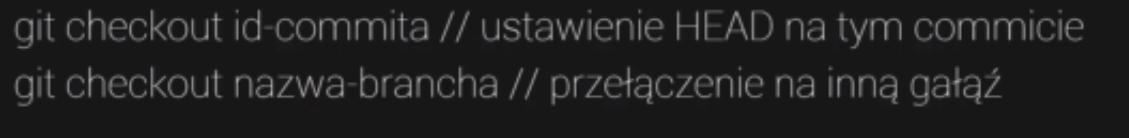
Obraz zawierający tekst

Opis wygenerowany automatycznie

Git checkout -- plik – zmiany w wersji roboczej są usuwane i przywracana jest(w katalogu roboczym) wersja, która znajduje się w indeksie. W praktyce jeślni nie mamy żadnej zmiany w staging area to wersja jaka była po ostanim commicie. Te -- muszą być po spacji bez tych kresek git może uznać, że kolejnym parametrem jest coś innego niż plik.

Git checkout HEAD – przywraca do katalogu roboczego wersje z ostaniego commita

Git checkout skrócony hash commita – przywrócenie indeksu z danego commita



Git reset / git reset HEAD – usuwa pliki ze staging area (pliki nadal są śledzone I są w katalogu roboczym). Przywracamy stan indeksu do stanu po ostanim commicie.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Warto zapamiętać:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Robiąc commity w gicie dobrą praktyką jest rozdzielanie commitów na te które dotyczą bezpośrednio projektu i na te które dotyczą np. tylko gita jak .gitignore

Plik .gitignore określa jakie pliki są ignorowane przez repozytorium(bo są tymczasowe)

.gitignore:

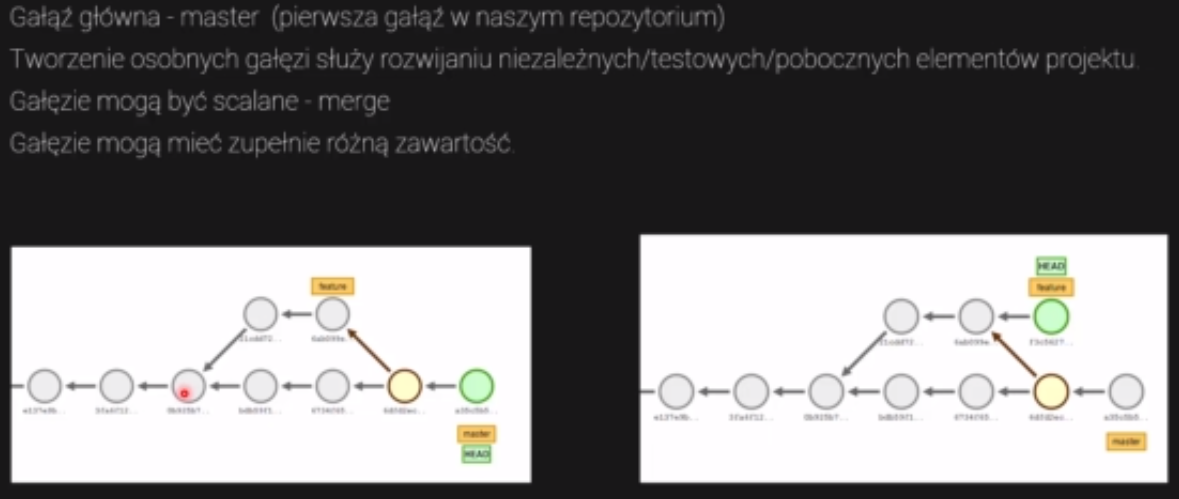
Obraz zawierający tekst

Opis wygenerowany automatycznie

Git branch – na jakim branchu(gałęzi) się znajdujemy

Początkowo jest tylko jeden branch(master). Po wpisaniu git branch dostaniemy info o istniejących gałęziach(by to zoaczyć trzeba zrobić przynajmniej jeden commit)

Branches:



Obraz zawierający tekst

Opis wygenerowany automatycznie

Tworzenie nowego brancha:

Git checkout -b nazwa-brancha

Przeskok na innego brancha:

Git checkout nazwa-brancha

Git remote add origin (link do repo) – łączy repo lokalne ze zdalnym na github

Git push – przekazywanie zmian z jednego repo(lokalnego) do innego repo(zdalnego)

Aktualizujemy repo które sklonowaliśmy (zdalne)

Git push -u origin master – opcja ‘u’ to set upstream czyli tworzy trwałe połączenie. Nie trzeba będzie za każdym razem dodawać origin master git już to zapamięta wystarczy git push

Git push – force - The --force option for git push allows you to override this rule: the commit history on the remote will be forcefully overwritten with your own local history. This is a rather dangerous process, because it's very easy to overwrite (and thereby lose) commits from your colleagues.

Git pull – pobieranie zmian z jednego repo(zdalnego) na inne(nasze lokalne)

Nasze sklonowane repo jest aktualizowane

Pull request – zgłoszenia osób zawierające poprawki. Muszą zostać zaakceptowane, żeby weszły w życie.

Fork w githubie to przeniesienie czyjegoś repo na swoją stronę na github takie klonowanie ale w obrębie githuba

Git pull – uaktualnia lokalne repo do stanu repo zdalnego

You Don't Care About the Local Changes

In this case, you just want to drop all the uncommitted local changes. Perhaps you modified a file to experiment, but you no longer need the modification. All you care about is being up to date with the upstream.

This means that you add one more step between fetching the remote changes and merging them. This step will reset the branch to its unmodified state, thus allowing git merge to work.

git fetch

git reset --hard HEAD

git merge origin/$CURRENT\_BRANCH

Git Pull Force – How to Overwrite Local Changes With Git bardzo przydatne =>

https://www.freecodecamp.org/news/git-pull-force-how-to-overwrite-local-changes-with-git/

Git rebase – jeśli historia lokalnego repo nie pokrywa się z historią repo na github.

Bierze końcówkę brancha, na którym jesteśmy i doczepia go do ostatniego commita na masterze. Nie tworzy commita.

Podobnie działa git merge master – ale stworzyłby commit z tym mergem

Często jak zrobisz repo z readme od razu a twoje repo lokalne nie zawiera tego readme.

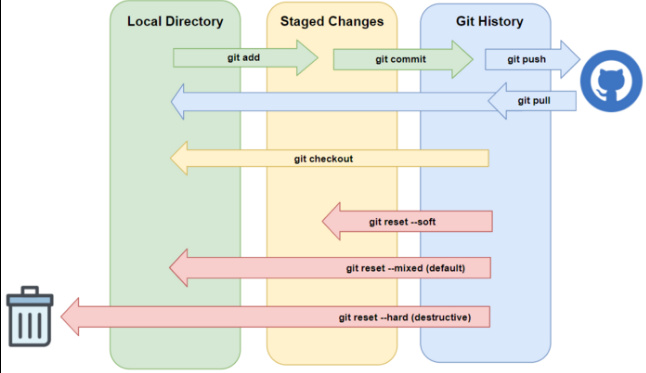
Nie używać git rebase gdy pracuje na branchu z innym programistą

git stash - temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-appfly them later on.

Pull requests are a mechanism for a developer to notify team members that they have completed a feature.

W komentarzu pull requesta powinienieś opisać co zrobiłeś, żeby inni wiedzieli co robi to co stworzyłeś.

Najważniejsze polecenia github:



Usuwanie commita z githuba:

<https://www.howtogeek.com/devops/how-to-remove-a-commit-from-github/>

branche usunięte w git hub są widoczne po wpisaniu git branch -a, żeby się ich pozbyć to:

git remote prune origin will remove all such stale branches. That's probably what you'd want in most cases, but if you want to just remove that particular remote-tracking branch, you should do:

git branch -d -r origin/coolbranch

(The -r is easy to forget...)

-r in this case will "List or delete (if used with -d) the remote-tracking branches." according to the Git documentation found here: <https://git-scm.com/docs/git-branch>

git prune:

<https://www.atlassian.com/git/tutorials/git-prune#:~:text=The%20git%20prune%20command%20is,is%20generally%20not%20executed%20directly>.

Usuwanie commita ze złego brancha i przenoszenie go na dobry branch => https://www.clearvision-cm.com/blog/what-to-do-when-you-commit-to-the-wrong-git-branch/#:~:text=Make%20sure%20you%20are%20on,commits%20you%20want%20to%20undo.&text=Then%20create%20a%20new%20branch,and%20commit%20your%20changes%20again.