KURS GIT DLA POCZĄTKUJĄCYCH – NOTATKI

(tam gdzie znajduje się - - należy wpisać 2 minusy bez spacji; cudzysłów wpisujemy zawsze górny)

**cd nazwa katalogu** === change directory

**ls** ===lista plików które znajdują się w tej ścieżce

**pwd** ===pokazuje w jakiej ścieżce się znajdujemy

**git init** === inicjuje stworzenie nowego repozytorium

**/.git/** === folder w którym tworzone będzie repozytorium, często jest ukryty

**git status**

po przekopiowaniu plików do folderu, w którym znajduje się repozytorium pojawia się w konsoli komunikat o tych plikach – nie są one jeszcze śledzone

**git add nazwa pliku** === przeniesie pliki do indexu (staging area), zaczną być śledzone

**git rm - -cached nazwa pliku** === aby cofnąć tę zmianę

**git add .** === doda do indexu wszystkie pliki z katalogu, dodaje pliki zarówno dotąd nie śledzone, jak też zmiany w plikach śledzonych (nowa wersja traktowana jest jako nowy plik)

**git commit –m „ „** === wewnątrz (cudzysłów górny ”)podajemy wiadomość, która będzie widoczna dla innych użytkowników i w ten sposób zapisujemy plik z komentarzem w repozytorium, zapisuje w repozytorium tylko pliki przygotowane do złożenia (czyli wcześniej konieczna komenda git add lub git commit –a)

**git restore nazwa pliku** === aby odrzucić zmiany w katalogu roboczym, aż do ostatniego commita

**git log** === pokaże informacje o commicie,w tym jego id

**git config** === aby skonfigurować nazwę użytkownika lub e-mail

**git diff nazwa pliku** === aby zobaczyć z poziomu konsoli jakie zmiany zostały naniesione w tym pliku

na czerwono wyświetla się co usunięto, na zielono co dodano

**git commit –a** === doda jednocześnie pliki do sekcji przygotowane do złożenia i je scommituje

**czyli git commit – a – m „nazwa commita”** === spowoduje, że do repozytorium zostanie dodane wszystko (gdy chcemy dodać tylko niektóre zmienione pliki należy używać git add, git commit –m), użycie parametru –a tworzy jakby całą paczkę do dodania i commituje ją zbiorczo

**cat ./nazwa pliku** === uruchamiamy plik, który znajduje się w obecnej ścieżce, w ten sposób można podejrzeć jego zawartość

**git restore – - staged nazwa pliku** === aby wycofać zmiany, które zostały dodane do złożenia komendą git add (- - staged jako parametr, którym informujemy gita, że ten dokument znajduje się w sekcji stagingowej czyli w sekcji zmian gotowych do złożenia) = w ten sposób wycofamy zmiany do sekcji dodanych do złożenia, żeby wycofać je całkowicie należy następnie użyć komendy git restore nazwa pliku (czyli odrzucić zmiany w katalogu roboczym)

Jak wykasować zmiany wprowadzone w ramach konkretnego commitu

Należy znać id commitu (git log)

**git revert id commitu** === komenda robi rzecz przeciwną niż tę wykonaną w commicie, którego id wkleiliśmy

w edytorze można zmienić opis wycofywania commita

żeby zamknąć edytor i zapisać te zmiany należy napisać **:wq!**

Przywracanie repozytorium do stanu z danego commita

**git reset id commita**, do którego chcemy zresetować nasze repozytorium (w ten sposób usuwamy również historię logów)

wszystkie zmiany, które były wykonane po tym commicie zostały wrzucone do sekcji nieprzygotowane do złożenia, aby je usunąć z danego pliku używamy komendy git restore nazwa pliku.

Bezpieczniej usuwać zmiany komendą git revert, ponieważ w ten sposób nie usuwamy historii commitów i samych commitów – można się cofnąć do konkretnych commitów. Korzystając z git reset usuwamy historię commitów i nie możemy już cofnąć się do poszczególnych zmian.

Ignorowanie plików i folderów, których nie chcemy przenosić do repozytorium

W tym celu tworzymy nowy plik, z którego git będzie zczytywał informację, co ma ignorować

**touch .gitignore**

Po otwarciu tego pliku wpisujemy w nim nazwę pliku czy folderu, który chcemy ignorować.

Plik ten dodajemy do repozytorium (git add, git commit –m)

Gałęzie

**git branch nazwa ===** tworzenie gałęzi

**git branch (bez podania nazwy)** === pokazuje wszystkie stworzone gałęzie

**git checkout nazwa gałęzi** === aby przejść do innej gałęzi

Przy tworzeniu nowego brancha należy zrócić uwagę, na jakim aktualnie się znajdujemy. Gdybyśmy tworzyli branch będąc na masterze to nowa gałąź będzie kopią mastera. Gdy stworzymy nową gałąź znajdując się na innej tonowa gałąź będzie kopią tej innej.

Łączenie zmian miedzy branchami

Gdy chcemy do gałęzi dodać to, co jest na innej należy znaleźć się na tej gałęzi, do której będzie się dołączać kolejną (np. na masterze, gdy chcemy zmiany dołączyć do mastera)

**git merge nazwa gałęzi przyłączanej**

Aby usunąć zbędną gałąź

**git branch –d nazwa usuwanego brancha**

Konflikt przy scalaniu zmian na branchach – gdy w tym samym pliku, który chcemy scalić dokonano innych zmian na 1 z branchy niż na 2 branchu.

Należy porównać oba pliki na obu branchach i zobaczyć czym się różnią.

Po naprawieniu konfliktów (czyli ręcznym dokonaniu zmian w modyfikowanym pliku) wykonujemy git commit, aby zapisać zmiany.

**git stash ===** zapisywanie zmian do schowka (przydatne, gdy pracujemy nad jakimiś zmianami w1 gałęzi, daliśmy już komendę git add, ale potrzebujemy przejść na kolejną gałąź – np. jakieś szybkie zmiany na tej kolejnej gałęzi a nie chcemy przez przypadek wgrać tych zmian z 1 gałęzi, które jeszcze nie są scommitowane na drugą gałąź – normalnie po przejściu na kolejną gałąź zmiany są widoczne i możliwe do scommitowania, więc należy uważać by nie wgrać ich na złą gałąź przez przypadek – gdy zmiany przenieśliśmy do schowka komendą git stash nie są one widoczne do scommitowania ani na pierwotnej gałęzi, ani na tej na którą się przełączyliśmy)

**git stash apply ===** aby przywrócić zmiany zapisane w schowku (gdy już będziemy pracować na branchu na którym rzeczywiście chcemy wgrać te zmiany), one nadal pozostaną w schowku – można korzystać z tej komendy wielokrotnie

**git stash pop** === pobierze zmiany ze schowka i automatycznie wyczyści schowek

**git remote add origin link do git huba** === dodanie informacji o repozytorium zdalnym

**git remote rm origin** === usunięcie informacji o repozytorium zdalnym (gdy np. wcześniej podaliśmy link do repozytorium https a teraz używamy kluczy SSH link do repozytorium jest inny, czyli trzeba będzie usunąć informację o repozytorium zdalnym https a następnie komendą git remote add origin link do git huba wprowadzamy link do git huba jaki widnieje w instrukcji przy użyciu SSH)

**git push –u origin main (jeśli zmieniliśmy nazwę głównej gałęzi na main) albo master ===** aby w repozytorium zdalnym stworzyć branch master odpowiadającą gałęzi master w naszym repozytorium lokalnym

po odświeżeniu git huba wszystko powinno być widoczne

Gdy utworzymy nową gałąź i w niej stworzymy zmiany, które następnie wypchniemy na git huba to do połączenia tych zmian z masterem potrzebujemy pull request.

Pull request to prośba o to, by móc zmergować nasze zmiany do mastera, inni członkowie zespołu mogą skomentować te zmiany, zaakceptować lub zażądać poprawek. Gdy pull request zostanie wykonany, zmiany zostaną zmergowane do głównej gałęzi. Zmiany te na razie znajdą się w naszym zdalnym repozytorium. Nasze lokalne repozytorium (branch master) nadal jest w wersji bez zmian.

Aby zmiany zostały naniesione także w naszym lokalnym repozytorium należy wykonać ( pierwszy sposób) komendę

**git fetch origin master** === pobiera informację o aktualnej kopii repozytorium zdalnego, a następnie

git status === aby zorientować się czym się różnią repozytoria

**git merge** === powoduje połączenie zmian pobranych z repozytorium zdalnego z tym, co znajduje się w repozytorium lokalnym

(2 sposób) komenda

**git pull origin master** = zastępuje komendy git fetch i git merge

Powyższe komendy pobierały zmiany dla pojedynczego wskazanego brancha.

**git pull - -all** === pobiera zmiany dla całego repozytorium

analogicznie można wypchnąć wszystkie zmiany z repozytorium lokalnego do repozytorium zdalnego wykonując **git push - - All**

**cd ..** === przechodzimy folder wyżej

**mkdir nazwa folderu** === tworzy nowy folder

**git clone adres repozytorium** === klonujemy czyjeś repozytorium z githuba na nasz dysk

Tortoise GIT === rozszerzenie Windows dla obsługi gita, panel użytkownika

Intellij IDEA === zintegrowane środowisko programistyczne dla Javy (IDE for Java)

Pycharm = zintegrowane środowisko programistyczne dla Pythona ( IDE for Python)