**PODSTAWY GIT:**

**0. Wstępne informacje:**

- <git help [polecenie]> - pozwala uzyskać pomoc na temat wszystkich poleceń lub innych spraw dotyczących Gita,

**1.Założenie repozytorium na 2 sposoby:**

- z poziomu katalogu projektu (w Bashu) wykonać polecenie <git init> *(co stworzy nowy katalog o nazwie .git wraz z niezbędnymi plikami)*

- sklonowanie istniejącego repozytorium z innego serwera (*np. z GitHuba – kopiowanie linka protokołu https!, SSH jest dodatkowym protokołem wymagającym klucza*) – poprzez polecenie <git clone ‘link’>

**1.A – Sklonowanie istniejącego repozytorium i założenie własnego na GitHub (from stackoverflow.com):**

1. **Create a new repository at**[**github.com**](https://github.com/)**.** (this is *your repository*)
   * Give it the same name as the *other repository*.
   * Don't initialize it with a README, .gitignore, or license.
2. **Clone the *other repository* to your local machine.** (if you haven't done so already)
   * git clone https://github.com/other-account/other-repository.git
3. **Rename the local repository's current '*origin*' to '*upstream*'.**
   * git remote rename origin upstream
4. **Give the local repository an '*origin*' that points to *your repository*.**
   * git remote add origin https://github.com/your-account/your-repository.git
5. **Push the local repository to *your repository* on github.**
   * git push origin master

**Now '*origin*' points to *your repository* & '*upstream*' points to the *other repository*.**

* Create a new branch for your changes with git checkout -b my-feature-branch.
* You can git commit as usual to *your repository*.
* Use git pull upstream master to pull changes from the *other repository* to your master branch.

**2. Rozpoczęcie kontroli wersji istniejących plików:**

- śledzenie plików rozpoczyna się od utworzenia początkowej rewizji,

- polecenia:

<git add file.ext> - *(dodaje pojedynczy plik lub ‘<git add .>’ dodaje wszystkie zmodyfikowane oraz nieśledzone pliki) =* w terminologii git jest to tzw. ‘stage’

<git add README> -

<git commit –m ‘initial project version’> - zatwierdzanie zmian od razu z komentarzem w cudzysłowiu,

<git reset HEAD file.ext> - cofa dodany plik *(lub wszystkie)* ze śledzenia,

<git status> - sprawdza status plików w repozytorium,

- pliki śledzone *(to te, które znalazły się w ostatniej migawce)* mogą być niezmodyfikowane, zmodyfikowane lub oczekiwać w poczekalni *(po dodaniu ich poleceniem add),*

- pliki nieśledzone – widziane przez Git *(czerwony kolor)* poprzez porównanie z ostatnią migawką, ale ich śledzenie rozpocznie się dopiero po wydaniu polecenia <git add>,

- utworzenie nowego pliku *(np. w repozytorium)* z poziomu wiersza poleceń poprzez edytor Windowsa <notepad file.ext> lub w macOS <vim file.ext>.

**3. Dodawanie zmodyfikowanych plików do poczekalni:**

- <git add file.ext> - wysyła zmodyfikowany plik do poczekalni *(był w trybie śledzenia z poprzedniej migawki)*,

- <git status> - pozwala sprawdzić aktualny stan plików w repozytorium,

- ewentualne dodatkowe zmiany w pliku wymagają ponownie polecenia <git add file.ext>, inaczej polecenie zatwierdzające uwzględni tylko pierwszą modyfikację,

- <git diff> - pokazuje wszystkie zmiany dokonane w plikach jeszcze przed wysłaniem do poczekalni *(w odróżnieniu od <git status>),*

- <git diff –staged> - pokazuje wszystkie zmiany przekazane do poczekalni, zanim zostaną zatwierdzone *(z ang. commited)*.

**4. Zatwierdzanie zmian:**

- w Windows warto wskazać nazwę edytora, który będzie użyty w trakcie commitu *(<git config –global core.editor notepad>, w macOS domyślny jest Vim),*

- <git commit> - zatwierdzenie zmian w repozytorium, po którym następuje uruchomienie edytora w celu podania charakterystyki wprowadzonych zmian,

- <git commit –v> - do komentarza trafią również zmodyfikowane wiersze poszczególnych plików,

- <git commit –m ”initial project version”> - zatwierdzanie zmian od razu z komentarzem w cudzysłowiu podwójnym!,

- pomijanie poczekalni poprzez <git commit –a – m ‘actualized READMI’> *(opcja ‘-a’ zastępuje add, ‘-m’ jest komentarzem do rewizji)* – obejmuje wszystkie pliki śledzone.

**5. Usuwanie i przenoszenie plików:**

- <git rm file.ext> - usuwa plik ze zbioru plików śledzonych oraz z katalogu roboczego, operacja ta wymaga jeszcze zatwierdzenia poleceniem ‘commit’,

- zwykła operacja usunięcia pliku z katalogu roboczego *(del w Windows lub rm w macOS)* skutkuje pokazaniem modyfikacji w sekcji zmienione ale nie zaktualizowane – poza poczekalnią; trwała zmiana wymaga wydania polecenia <git rm file.ext> co przesunie operację do poczekalni śledzenia plików,

- jeżeli nie chcemy aby dany plik trafił do repozytorium to polecenie <git rm – r –cached file.ext> zachowuje plik w katalogu roboczym jako nieśledzony, opcja ‘-r’ pozwala na ‘rekurencyjne’ usunięcie plików, tj. wszystkich w podanym (zamiast nazwy) pliku katalogu,

- pomimo istnienia polecenia ‘mv’ Git nie śledzi bezpośrednio operacji przenoszenia plików, natomiast polecenie <git mv file.ext file\_1.ext> pozwala zmienić nazwę pliku *(w ten sposób można uniknąć 3 poleceń: zwykłej zmiany nazwy, usuwania pliku ‘git rm’, oraz dodawania pliku ‘git add’)*.

**6. Podgląd historii rewizji:**

- <git log> - polecenie bez argumentów listuje zmiany zatwierdzone w tym repozytorium w odwrotnej kolejności chronologicznej *(najnowsze zmiany w pierwszej kolejności)*; podane są: suma kontrolna SHA-1, nazwisko oraz email autora, data zapisu oraz notka zmiany,

- ‘git log’ z opcją ‘-p’- <git log –p> - pokazuje różnice wprowadzone każdą rewizją *(po każdym wpisie pokazywany jest ‘diff’)*; dodatkowa opcja <git log –p -2> ogranicza zbiór do ostatnich 2 wpisów,

- <git log --stat> - z opcją ‘--stat’ pod każdym wpisem historii jest wyświetlana lista zmodyfikowanych plików, liczba zmienionych plików oraz dodanych i usuniętych wierszy *(‘+’, ‘-‘)*,

- <git log --pretty> - opcja ‘--pretty’ pozwala prezentować wpisy w różnych formatach poprzez dołączenie predefiniowanych wariantów*(np. ‘short’, ‘full’, ‘fuller’, ‘oneline’)*, np.: <git log –pretty=short>,

- <git log –pretty=format:”%h - %an, %ar : %s”> - to przykład własnego formatowania wpisów historii *(do znalezienia w dokumentacji Git)*, który można uzupełnić dodatkową opcją ‘—graph’ *(po znakach formatowania)* – co powinno utworzyć tzw. graf ASCII.

- oprócz wymienionych wyżej opcji formatowania polecenia ‘git log’ są jeszcze: --shortstat, --name-only, --name-status, --abbrev-commit, --relative-date.

- ograniczanie wyniku historii, oprócz wcześniej wspomnianego parametru ‘-n’, można zastosować ograniczenie w czasie np. <git log –since=2.weeks> lub podać datę bezwględną np. 2017-11-12 lub względną np. ‘2 years, 3 days, 1 min ago’,

- inne ograniczenia: ‘—after’, ‘—until’, ‘—before’, ‘—author’, ‘—committer’*(jedynie te rewizje, w których osoba zatwierdzająca zmiany pasuje do podanej),* np. <git log –pretty=”%h - %s” –author=”Leszek Tlałka”>,

- Wyświetlenie historii rewizji umożliwia fajne narzędzie okienkowe, uruchamiane z wiersza poleceń <gitk> !!

**7. Cofanie zmian:**

- <git commit –amend> - służy poprawieniu ostatniej rewizji *(korekta komentarza lub dodanie/usunięcie pliku, po wcześniejszych 2 poleceniach: <git commit –m ‘initial commit’> oraz <git add ‘forgotten file’>)*,

- usuwanie pliku z poczekalni: <git reset HEAD file.ext> pozwoli wycofać (unstage) plik z poczekalni – będzie wymieniany jako modified ale nie zgłoszony do stage,

- <git checkout -- file.ext> - pozwala cofnąć zmiany w zmodyfikowanym pliku do wersji, która była w ostatnim repozytorium *(b. przydatne jeśli np. w Visual Studio Code nie wiadomo jak cofnąć zmianę!).*

**8. Praca ze zdalnym repozytorium:**

- <git remote> - wyświetla listę skrótów nazw zdalnych repozytorium, które są śledzone (tracking) *(z poziomu katalogu roboczego w wierszu poleceń),*

- <git remote –v> - wyświetla pełny adres URL skrótu jw., tj. adres do serwera GitHub, może zostać wyświetlona większa liczba repozytoriów, jeżeli są takie wewnątrz katalogu roboczego,

- dodawanie zdalnych repozytoriów - <git remote add [skrót] [url]> , gdzie jako skrót można podać np. lt, a jako url adres https repozytorium na serwerze GitHuba,

- <git fetch lt> - pozwala pobrać zmiany w zdalnym repozytorium na serwerze dokonane przez kogoś innego lub np. własne z innego komputera *(powinno wyświetlać dane liczbowe oraz prawd. info nt zmian w repozytorium),*

- pobieranie i wciąganie zmian ze zdalnego repozytorium: <git fetch [nazwa zdalnego repo]> *(jako nazwa zdalnego może prawdop. działać zarówno skrót (lt), jak i ‘origin’ lub adres url)*, po którym można dokonać merge z własnym lokalnym repo o nazwie ‘master’ np. <git merge origin/master …files..> *(origin/master to lokalna gałęź reprezentująca stan gałęzi master zdalnego repozytorium),*

- ponowne odesłanie własnych zmian w repozytorium na serwer: <git push origin master> *(push master to origin)*,

- <git remote show origin> pokazuje kilka informacji o zdalnym repozytorium, w tym info na temat możliwości automatycznego scalania zmian w zdalnym repozytorium z gałęzią master w moim lokalnym repo – wystarczy <git pull>,

- zmiana nazwy i usuwanie zdalnego repozytorium np. <git remote rename lt licho>, a także <git remote rm licho>.

**9. Tagowanie (etykietowanie):**

- <git tag> - wyświetla w porządku alfabetycznym wszystkie etykiet/wersje repozytorium; z opcją <git tag –l ‘x\*’> prezentowane są wersje w zakresie wskazanym np. ‘3.4\*’,

- 2 rodzaje etykiet – lekkie i opisane; lekkie posiadają wyłącznie skrót rewizji, natomiast opisane są rozszerzone o autora, datę utworzenia i ewentualny komentarz,

- tworzenie taga opisanego - <git tag –a nazwa taga –m ‘Opis taga’> , ‘-m’ określa notkę, nazwa taga musi być oczywiście unikalna; zamiast parametru ‘-a’ – annotated, można użyć ‘-s’ – co podpisze etykietę prywatnym kluczem GPG,

- tag lekki nie wymaga opcji ‘-a, -s, -m’, jedynie <git tag nazwa taga>,

- <git tag describe –tags> - opcja ‘describe’ pokazuje najmłodszy znacznik *(tag, etykieta)*, <git show nazwa taga> prezentuje dane etykiety wraz z tagowaną rewizją,

- <git tag –v nazwa etykiety> - weryfikowanie etykiet wymaga klucza GPG do zweryfikowania podpisu, jeżeli się nie ma klucza to przyjdzie komunikat z błędem weryfikacji,

- etykietowanie historii - <git tag –a x.y suma kontrolna>, zamiast całej sumy można podać tylko jej część,

- współdzielenie etykiet - <git push origin nazwa etykiety> *(domyślnie, polecenie git push nie przesyła nadanych lokalnie etykiet do zdalnego repozytorium),* z opcją ‘—tags’ można wypchnąć wszystkie etykiety, których nie ma jeszcze na serwerze, jednocześnie.

**10. Aliasy i autouzupełnianie:**

- <git config –global alias.co checkout> - skrót ‘co’ do ‘checkout’,

- <git config –global alias.br branch> - skrót ‘br’ do ‘branch’,

- <git config –global alias.ci commit> - skrót ‘c’ do ‘commit’,

- <git config –global alias.st status> - skrót ‘st’ do ‘status’,

- <git config –global alias.fe fetch> - skrót ‘fe’ do ‘fetch’,

- każdy ze skrótów wymaga oddzielnego uruchomienia,

- <git config –global alias.unstage ‘reset HEAD fileA>‘ – równoważy polecenie <git reset HEAD fileA> z <git unstage fileA>,

- podobnie <git config –global alias.last ‘log –1 HEAD’>, pozwala łatwiej zobaczyć ostatnią rewizję,

- autouzupełnianie – po skopiowaniu z kodu źródłowego Gita skryptu ‘git-completion.bash’ do katalogu domowego *(w Mac)* należy dodać do .bashrc następującą linijkę: ‘source ~/.git-completion.bash.

**11. Gałęzie Gita:**