**PODSTAWY GIT:**

**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’>

**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 –cached file.ext> zachowuje plik w katalogu roboczym jako nieśledzony,

- 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> !!