1. Tworzymy puste repozytorium o nazwie cw02
2. Sprawdzamy listę dostępnych gałęzi (branch).
3. Tworzymy plik main.cpp, dodajemy do repozytorium z zatwierdzamy zmiany.
4. Sprawdzamy listę dostępnych gałęzi (branch) i historię zatwierdzeń z pokazywaniem etykiet (log --decorate).
5. Tworzymy nową gałąź o nazwie hello (branch), przełączamy się do niej (checkout). Można użyć jednego polecenia zamiast dwóch innych (checkout -b).
6. Sprawdzamy listę dostępnych gałęzi i historię zatwierdzeń z pokazywaniem etykiet.
7. W pliku main.cpp piszemy program „Hello World". Tworzymy plik README z naszym imieniem i nazwiskiem. Wszystkie zmiany dodajemy i zatwierdzamy.

```
#include <iostream>
<<<<<<< HEAD
// Cały program, który niewiele co robi ...

int main() {
std::cout<<"Hello World"<<std::endl;
return 0;
}
```

8. Sprawdzamy historię zatwierdzeń z pokazywaniem etykiet.
9. Przełączamy się na gałąź master, sprawdzamy zawartość katalogu roboczego i historię zatwierdzeń z pokazywaniem etykiet.
10 Scalamy zmiany wykonane w gałęzi hello do gałęzi master. Najprostszy typ scalania – Fast-forward. Następuje tylko przeniesienie wskaźnika gałęzi.
11. Oglądamy zawartość katalogu roboczego i historię zmian z pokazywaniem etykiet.
12. Usuwamy gałąź hello (branch -d). Tworzymy nową gałąź witaj. *Prościej wystarczy zmienić nazwę.*
13. Będąc w gałęzi master dodajemy w pierwszej linijce pliku main.cpp informacje o prawach autorskich: „Copyright by ...(imie)". Zmiany zatwierdzamy.
14. Przełączamy się na gałąź witaj.
15. Zmieniamy treść wyświetlanego tekstu w pliku main.cpp na „Witaj ZIMO". Dodatkowo usuwamy plik README (git rm), bo jest nam już niepotrzebny. Wszystkie zmiany zatwierdzamy (tym razem w gałęzi witaj).
16. Oglądamy historię zmian z pokazywaniem etykiet.

17. Zmieniamy gałąź na `master` i znowu oglądamy historię zmian.
18. Scalamy gałęzie.Jeśli nie zrobiliśmy błędu powinno się wykonać scalanie trójstronne. Jeśli nie ma zmian w tym samym miejscu tego samego pliku to git sam automatycznie wprowadzi zmiany do plików i połączy odpowiednie gałęzie (Auto-merging main.cpp). Powstanie nowe zatwierdzenie będące połączeniem obu gałęzi – zmiana scalająca (ang. merge commit).
19. Sprawdzamy co się usunęło, co się zmieniło, jak wygląda historia. Można użyć dodatkowo opcji `--graph`.

20. Tworzymy sobie nową gałąź dotyczącą dokumentacji (`doc`).
21. W gałęzi `master` w pliku `main.cpp`, nad funkcją `main()` dodajemy linijkę komentarza opisującą działanie programu.
22. Zatwierdzamy zmiany w gałęzi `master`.
23. Przechodzimy do gałęzi `doc`.
Nad funkcją main() piszemy dokumentację funkcji dla programu dtxag np.:

```
/// @brief Główna funkcja programu, wyświetla komunikat ...
///
/// @return zawsze zwraca wartość 0.
```

24. Zmiany zapisujemy i zatwierdzamy.
25. Przechodzimy z powrotem do gałęzi `master`. Scalamy `master` z `doc`.
Mamy konflikt!! Bardzo dobrze, tak miało być :) Trzeba go ręcznie rozwiązać.
26. Polecenie git status pokazuje w jakich plikach jest konflikt:

```
UU main.cpp
```

27. Ręcznie rozwiązujemy konflikt – wybieramy to co jest właściwe.
28. Poleceniem git add oznaczamy, że w pliku rozwiązano konflikt. Patrzymy jeszcze raz na status.
29. Zatwierdzamy zmiany z domyślną (bez parametru `-m`) lub naszą własną wiadomością.
30. Oglądamy historię zmian w gałęzi `master`, najlepiej z opcją – `graph`