Zadanie: Zaawansowane operacje w Git

Cel:

Nauka zaawansowanych operacji w Git, takich jak klonowanie repozytorium, dodawanie plików, commitowanie zmian (w tym za pomocą edytora Vim), pushowanie do zdalnego repozytorium, stashowanie i popowanie zmian, oraz praca z plikami i katalogami w konsoli.

Instrukcje:

1. **Klonowanie repozytorium (5 pkt):**
    - Skopiuj poniższe repozytorium na swój lokalny komputer:
    - **git clone https://github.com/przykladowe-repozytorium.git**
2. **Dodawanie nowego pliku (5 pkt):**
    - Przejdź do sklonowanego repozytorium:
    - **cd przykladowe-repozytorium**
    - Utwórz nowy plik o nazwie nowy_plik.txt:
    - **touch nowy_plik.txt**
    - Dodaj plik do śledzenia przez Git:
    - **git add nowy_plik.txt**
3. **Commitowanie zmian (5 pkt):**
    - Zatwierdź zmiany z odpowiednim komunikatem:
    - **git commit -m "Dodano nowy plik nowy_plik.txt"**
4. **Commitowanie zmian za pomocą edytora Vim (5 pkt):**
    - Wprowadź zmiany w pliku nowy_plik.txt:
    - **echo "Dodano nową linię tekstu" >> nowy_plik.txt**
    - Dodaj zmodyfikowany plik do śledzenia przez Git:
    - **git add nowy_plik.txt**
    - Zatwierdź zmiany za pomocą edytora Vim:
    - **git commit**
        - Po uruchomieniu edytora Vim, naciśnij **i**, aby wejść w tryb wprowadzania tekstu.
        - Wpisz wiadomość commit, np. **“Zaktualizowano plik nowy_plik.txt”**.
        - Naciśnij **Esc**, aby wyjść z trybu wprowadzania tekstu.
        - Wpisz **:wq**, aby zapisać i wyjść z edytora Vim.

5. **Pushowanie zmian (5 pkt):**
    - Wyślij zmiany do zdalnego repozytorium:
    - **git push origin main**
6. **Modyfikowanie pliku (5 pkt):**
    - Otwórz plik nowy_plik.txt w edytorze tekstu i dopisz dowolny tekst.
    - Zapisz zmiany i zamknij edytor.
    - Dodaj zmodyfikowany plik do śledzenia przez Git:
    - **git add nowy_plik.txt**
    - Zatwierdź zmiany z odpowiednim komunikatem:
    - **git commit -m "Zaktualizowano plik nowy_plik.txt"**
7. **Praca z katalogami (5 pkt):**
    - Utwórz nowy katalog o nazwie nowy_katalog:
    - **mkdir nowy_katalog**
    - Przenieś plik nowy_plik.txt do nowego katalogu:
    - **mv nowy_plik.txt nowy_katalog/**
    - Wyświetlanie struktury katalogów i plików
    - Użyj polecenia **ls -R**, aby wyświetlić strukturę katalogów i plików w repozytorium.
    - Dodaj zmiany do śledzenia przez Git:
    - **git add nowy_katalog/nowy_plik.txt**
    - Zatwierdź zmiany z odpowiednim komunikatem:
    - **git commit -m "Przeniesiono nowy_plik.txt do nowego_katalogu"**

8. **Stashowanie i popowanie zmian (10 pkt):**

Co to jest stash?

Stash w Git to mechanizm, który pozwala tymczasowo zapisać zmiany wprowadzone w kopii roboczej, aby móc przełączyć się na inną gałąź lub pracować nad czymś innym bez konieczności commitowania tych zmian. Zmiany są zapisywane w specjalnym schowku (stash), skąd można je później przywrócić. Jest to szczególnie przydatne, gdy trzeba szybko zmienić kontekst pracy, ale nie chce się jeszcze zatwierdzać wprowadzonych zmian.

Ćwiczenie 1: Stashowanie i popowanie zmian

1. Wprowadź zmiany w pliku nowy_plik.txt.
2. Zapisz zmiany do schowka:
3. **git stash**
4. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały zapisane:
5. **git status**
6. Przywróć zmiany ze schowka:
7. **git stash pop**
8. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały przywrócone:
9. **git status**

Wyjaśnienie kroków:

1. **Wprowadź zmiany w pliku** nowy_plik.txt**:**
    - Otwórz plik example.txt w edytorze tekstu i wprowadź dowolne zmiany, np. dodaj nową linię tekstu.
2. **Zapisz zmiany do schowka:**
    - Użyj polecenia git stash, aby tymczasowo zapisać zmiany wprowadzone w pliku nowy_plik.txt.
3. git stash**:**
    - To polecenie zapisuje wszystkie niezatwierdzone zmiany (zarówno te dodane do indeksu, jak i te, które nie zostały jeszcze dodane) i przywraca czysty stan roboczy. Dzięki temu możesz pracować nad czymś innym bez utraty wprowadzonych zmian.
4. **Sprawdź status repozytorium, aby upewnić się, że zmiany zostały zapisane:**
    - Użyj polecenia git status, aby sprawdzić, czy stan roboczy jest czysty i czy zmiany zostały zapisane do schowka.
5. git status**:**
    - To polecenie pokazuje aktualny status repozytorium, w tym informacje o zmodyfikowanych plikach, plikach dodanych do indeksu oraz plikach, które nie są śledzone przez Git.
6. **Przywróć zmiany ze schowka:**
    - Użyj polecenia git stash pop, aby przywrócić zmiany zapisane w schowku.
7. git stash pop**:**
    - To polecenie przywraca ostatnio zapisane zmiany ze schowka i usuwa je z listy schowków. Dzięki temu możesz kontynuować pracę nad wcześniej zapisanymi zmianami.

8. **Sprawdź status repozytorium, aby upewnić się, że zmiany zostały przywrócone:**
    - Użyj polecenia git status, aby sprawdzić, czy zmiany zostały przywrócone do stanu roboczego.
9. git status**:**
    - To polecenie ponownie pokazuje aktualny status repozytorium, tym razem powinno pokazać, że zmiany w pliku nowy_plik.txt zostały przywrócone.

Ćwiczenie 2: Stashowanie nieśledzonych plików

10. Utwórz nowy plik newfile.txt i wprowadź w nim zmiany.
11. Zapisz zmiany do schowka, w tym nieśledzone pliki:
12. git stash -u
13. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały zapisane:
14. git status
15. Przywróć zmiany ze schowka:
16. git stash pop
17. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały przywrócone:
18. git status

Różnica między git stash **a** git stash -u

git stash

Polecenie git stash zapisuje wszystkie niezatwierdzone zmiany w katalogu roboczym i indeksie (staged changes), a następnie przywraca czysty stan roboczy. Obejmuje to zmiany w plikach, które są już śledzone przez Git. Jednak nie obejmuje to plików, które są nieśledzone (untracked) lub ignorowane (ignored).

git stash -u

Polecenie git stash -u (lub git stash --include-untracked) działa podobnie do git stash, ale dodatkowo zapisuje również nieśledzone pliki. Oznacza to, że oprócz zmian w plikach śledzonych, zapisuje także nowe pliki, które zostały utworzone, ale nie zostały jeszcze dodane do śledzenia przez Git.

Kiedy używać którego polecenia?

- git stash: Używaj, gdy chcesz tymczasowo zapisać zmiany w plikach, które są już śledzone przez Git, i nie chcesz zapisywać nowych, nieśledzonych plików.
- git stash -u: Używaj, gdy chcesz tymczasowo zapisać wszystkie zmiany, w tym nowe, nieśledzone pliki, aby móc wrócić do czystego stanu roboczego.

Wyjaśnienie rozwiązania ćwiczenia:

Ćwiczenie 2: Stashowanie nieśledzonych plików

1. **Utwórz nowy plik** newfile.txt **i wprowadź w nim zmiany:**
    - Otwórz terminal i przejdź do katalogu repozytorium.
    - Utwórz nowy plik o nazwie newfile.txt:
    - touch newfile.txt
    - Otwórz plik newfile.txt w edytorze tekstu i wprowadź dowolne zmiany, np. dodaj tekst “To jest nowy plik”.
2. **Zapisz zmiany do schowka, w tym nieśledzone pliki:**
    - Użyj polecenia git stash -u, aby zapisać zmiany do schowka, w tym nieśledzone pliki. Opcja -u (lub --include-untracked) powoduje, że Git zapisuje również pliki, które nie są jeszcze śledzone przez repozytorium.
    - git stash -u
3. **Sprawdź status repozytorium, aby upewnić się, że zmiany zostały zapisane:**
    - Użyj polecenia git status, aby sprawdzić, czy stan roboczy jest czysty i czy zmiany zostały zapisane do schowka.
    - git status
    - Powinieneś zobaczyć komunikat informujący, że nie ma żadnych zmian do zatwierdzenia, co oznacza, że zmiany zostały poprawnie zapisane do schowka.
4. **Przywróć zmiany ze schowka:**
    - Użyj polecenia git stash pop, aby przywrócić zmiany zapisane w schowku. To polecenie przywraca ostatnio zapisane zmiany ze schowka i usuwa je z listy schowków.
    - git stash pop
5. **Sprawdź status repozytorium, aby upewnić się, że zmiany zostały przywrócone:**
    - Użyj polecenia git status, aby sprawdzić, czy zmiany zostały przywrócone do stanu roboczego.
    - git status
    - Powinieneś zobaczyć, że plik newfile.txt jest teraz widoczny jako zmodyfikowany lub nieśledzony, w zależności od tego, czy został dodany do śledzenia przed stashowaniem.

Ćwiczenie 3: Stashowanie wybranych fragmentów zmian

19. Wprowadź zmiany w pliku nowy_plik.txt.
20. Zapisz wybrane fragmenty zmian do schowka:
21. **git stash -p**
    - Git przeprowadzi Cię przez każdy fragment zmian, pytając, czy chcesz go stashować.
22. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały zapisane:
23. **git status**
24. Przywróć zmiany ze schowka:
25. **git stash pop**
26. Sprawdź status repozytorium, aby upewnić się, że zmiany zostały przywrócone:
27. **git status**

9. **Usuwanie pliku (5 pkt):**
    - Usuń plik nowy_katalog/nowy_plik.txt:
    - **rm nowy_katalog/nowy_plik.txt**
    - Dodaj zmiany do śledzenia przez Git:
    - **git add nowy_katalog/nowy_plik.txt**
    - Zatwierdź zmiany z odpowiednim komunikatem:
    - **git commit -m "Usunięto plik nowy_plik.txt"**

10. **Przywracanie pliku (5 pkt):**
    - Przywróć usunięty plik z ostatniego commita:
    - **git checkout HEAD^ nowy_katalog/nowy_plik.txt**
    - Dodaj zmiany do śledzenia przez Git:
    - **git add nowy_katalog/nowy_plik.txt**
    - Zatwierdź zmiany z odpowiednim komunikatem:
    - **git commit -m "Przywrócono plik nowy_plik.txt"**

Wyjaśnienie polecenia git checkout HEAD^ nowy_katalog/nowy_plik.txt

Polecenie git checkout jest używane do przełączania się między różnymi wersjami plików, commitów lub gałęzi w repozytorium Git. W kontekście git checkout HEAD^ nowy_katalog/nowy_plik.txt, polecenie to ma na celu przywrócenie konkretnego pliku do stanu z poprzedniego commita.

Składnia polecenia:

- git checkout: Podstawowe polecenie do przełączania się między wersjami.
- HEAD^: Odnosi się do poprzedniego commita (HEAD^ oznacza "parent commit" bieżącego HEAD, czyli jeden commit wstecz).
- nowy_katalog/nowy_plik.txt: Ścieżka do pliku, który ma zostać przywrócony.

Co robi to polecenie?

1. Przywraca plik do stanu z poprzedniego commita:
    - Polecenie git checkout HEAD^ nowy_katalog/nowy_plik.txt przywraca plik nowy_plik.txt znajdujący się w katalogu nowy_katalog do stanu, w jakim był w poprzednim commicie. Oznacza to, że wszelkie zmiany wprowadzone w tym pliku od czasu ostatniego commita zostaną cofnięte.
2. Nie wpływa na inne pliki:
    - To polecenie dotyczy tylko wskazanego pliku. Inne pliki w repozytorium pozostaną bez zmian.

Kiedy używać tego polecenia?

- Cofanie niechcianych zmian: Jeśli wprowadziłeś zmiany w pliku, które chcesz cofnąć do stanu z poprzedniego commita, to polecenie jest idealne.
- Przywracanie usuniętych plików: Jeśli przypadkowo usunąłeś plik i chcesz go przywrócić do stanu z poprzedniego commita.

11. **Sprawdzanie statusu repozytorium (5 pkt):**
    - Sprawdź status repozytorium, aby zobaczyć, które pliki zostały zmodyfikowane, dodane lub usunięte:
    - git status
12. **Wyświetlanie historii commitów (5 pkt):**
    - Wyświetl historię commitów, aby zobaczyć wszystkie zmiany wprowadzone w repozytorium:
    - **git log**

Uwagi:

- Upewnij się, że każda operacja jest poprawnie wykonana i zatwierdzona.
- W razie problemów, skonsultuj się z dokumentacją Git lub zapytaj prowadzącego.