<table>
<tr><td colspan="2" rowspan="3">Uniwersytet Śląski w Katowicach</td><td colspan="4">Wydział Nauk Ścisłych i Technicznych</td></tr>
<tr><td colspan="4">Instytut Fizyki</td></tr>
<tr><td>Rok</td><td>III</td><td>Semestr</td><td>V</td></tr>
<tr><td colspan="2">Kierunek</td><td colspan="4">Informatyka stosowana</td></tr>
<tr><td colspan="2">Przedmiot</td><td colspan="4">SiNWO - laboratorium</td></tr>
<tr><td colspan="2">Prowadzący</td><td colspan="4">dr Wojciech Gurdziel</td></tr>
<tr><td colspan="2">Tytuł ćwiczenia</td><td colspan="2">Git - aplikacje</td><td rowspan="2">Nr ćwiczenia</td><td rowspan="2">II</td></tr>
<tr><td colspan="2">Sprawozdanie wykonał:
(Imię i Nazwisko)</td><td colspan="2">Jakub Kraus</td></tr>
<tr><td colspan="2">Data wykonania ćwiczenia</td><td>23.01.2024</td><td>Data oddania sprawozdania</td><td colspan="2">01.02.2024</td></tr>
</table>

Spis treści

Spis rysunków

Listings

1 Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z aplikacjami do obsługi systemu kontroli wersji Git.

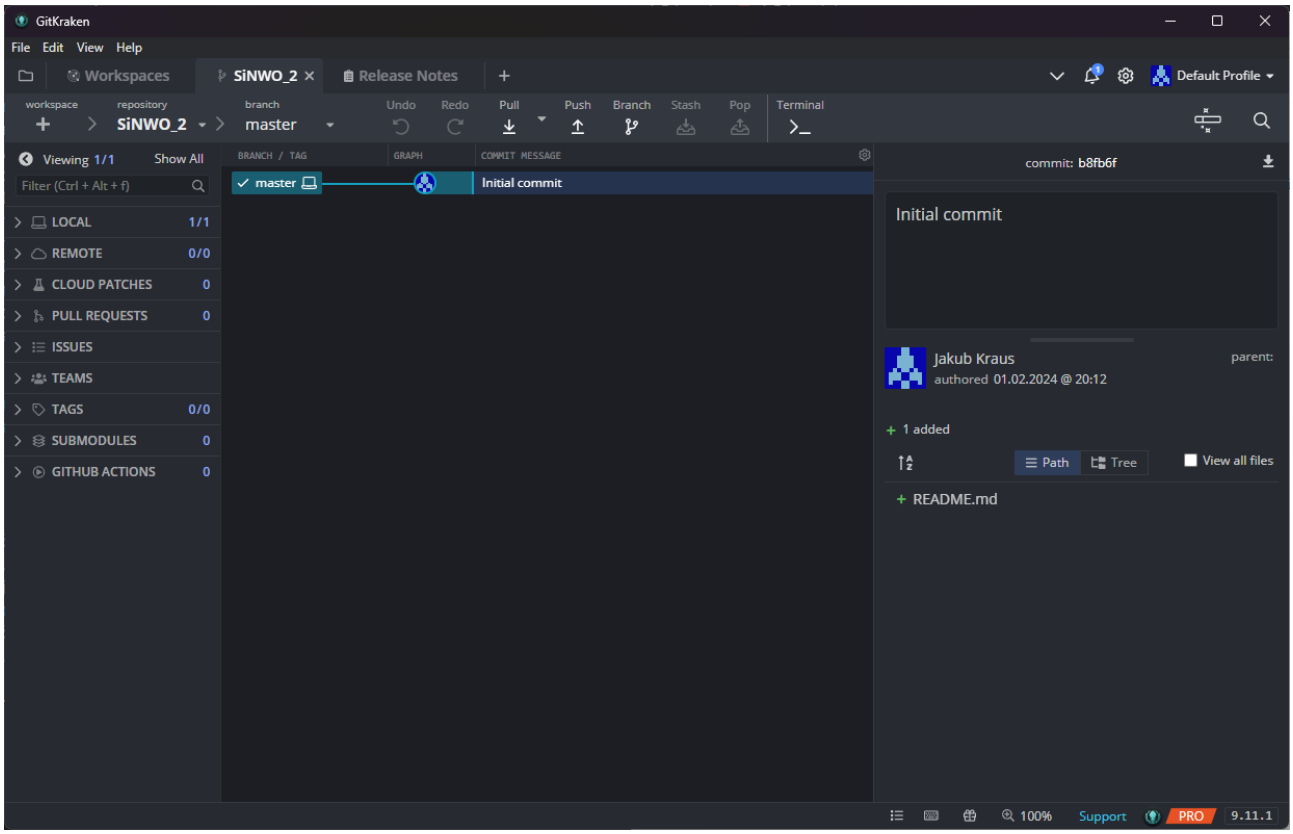
2 Przebieg ćwiczenia

Do wykonania ćwiczenia postanowiłem użyć narzędzie GitKraken oraz system operacyjny Windows 11.

2.1 Blame

```
$ git blame 344120
```

Listing 1: Blame w terminalu

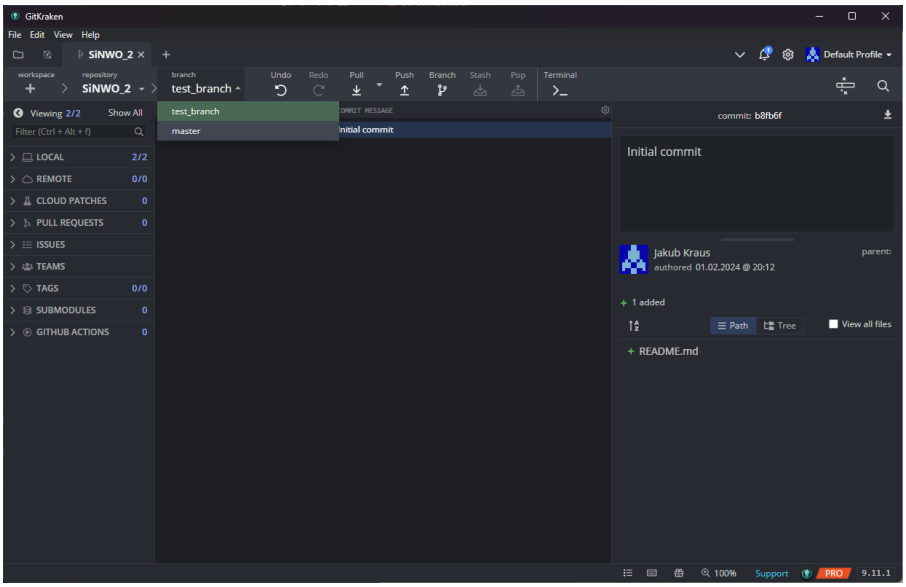


Rysunek 1: git blame w GitKraken

2.2 Checkout

```
$ git checkout test_branch
```

Listing 2: Checkout w terminalu

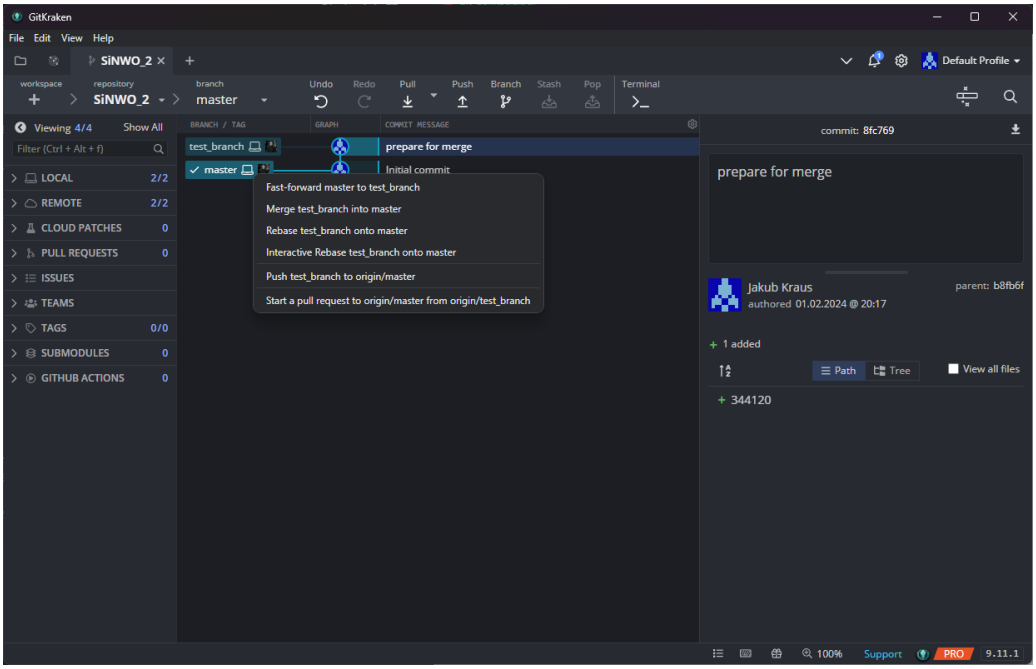


Rysunek 2: git checkout w GitKraken

2.3 Merge

```
$ git merge test_branch
```

Listing 3: Scalanie w terminalu

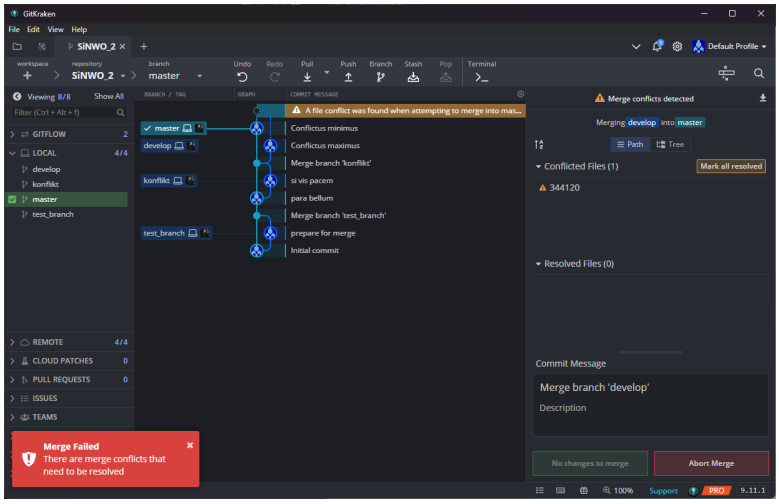


Rysunek 3: git merge w GitKraken

2.4 Konflikt scalania

```
$ git merge test_branch
Auto-merging 344120
CONFLICT (content): Merge conflict in merge.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Listing 4: Konflikt scalania w terminalu



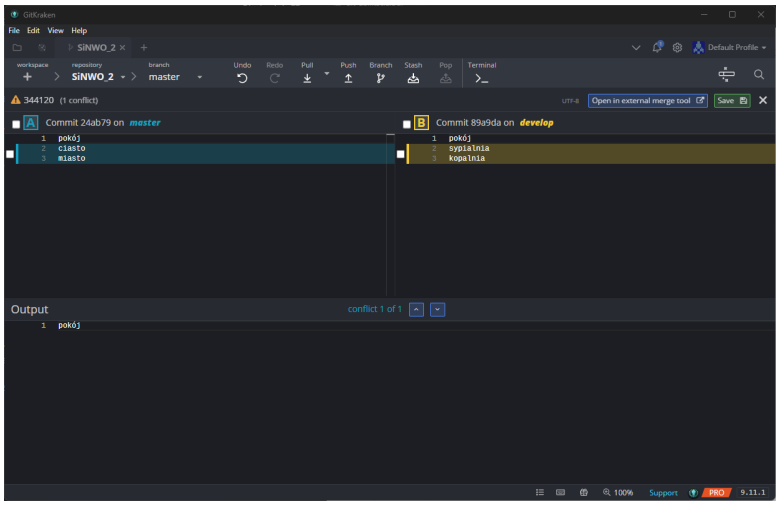
Rysunek 4: Konflikt scalania w GitKraken

2.5 Merge tool

Git mergetool można w skonfigurować, aby używał wybranego przez nas narzędzia do rozwiązywania konfliktów. W moim przypadku jest to vimdiff.

```
$ git config --global merge.tool vimdiff
$ git mergetool
```

Listing 5: Komenda mergetool w terminalu



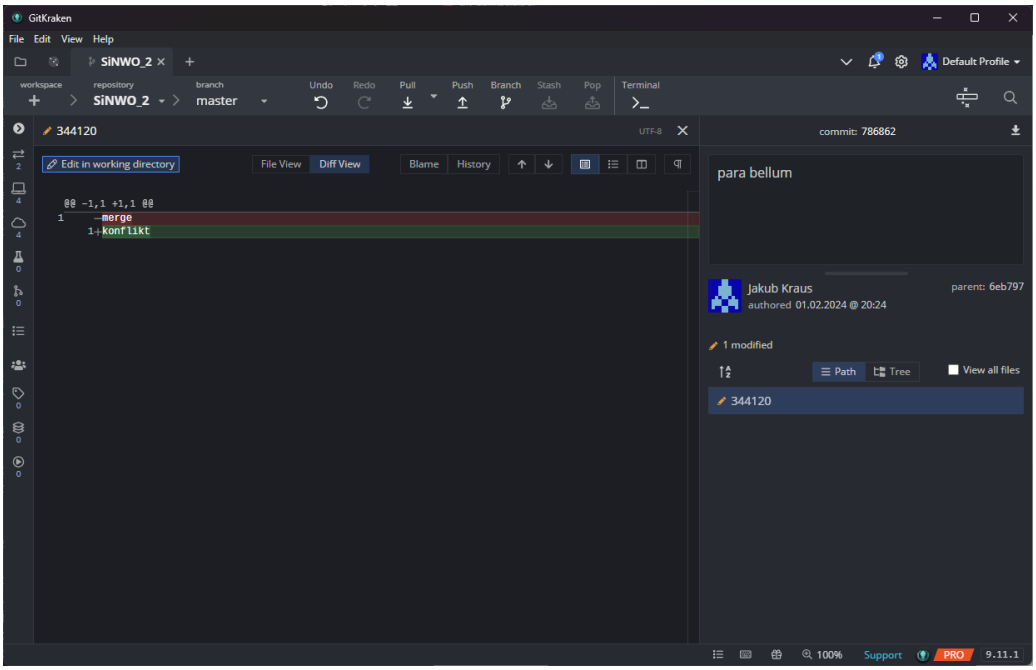
Rysunek 5: Mergetool w GitKraken

2.6 diff –word-diff

Word diff jest domyślnie włączony w GitKrakenie. W terminalu można go włączyć za pomocą polecenia:

```
$ git diff --word-diff
```

Listing 6: diff –word-diff w terminalu



Rysunek 6: diff-word-diff w GitKraken

2.7 log –graph –oneline

```
$ git log --graph --oneline
```

Listing 7: log –graph –oneline w terminalu

Rysunek 7: log –graph –oneline w GitKraken

2.8 Aliasy

Nie udało mi się znaleźć w GitKrakenie opcji do tworzenia aliasów.

```
$ git config --global alias.lg "log --graph --oneline"
```

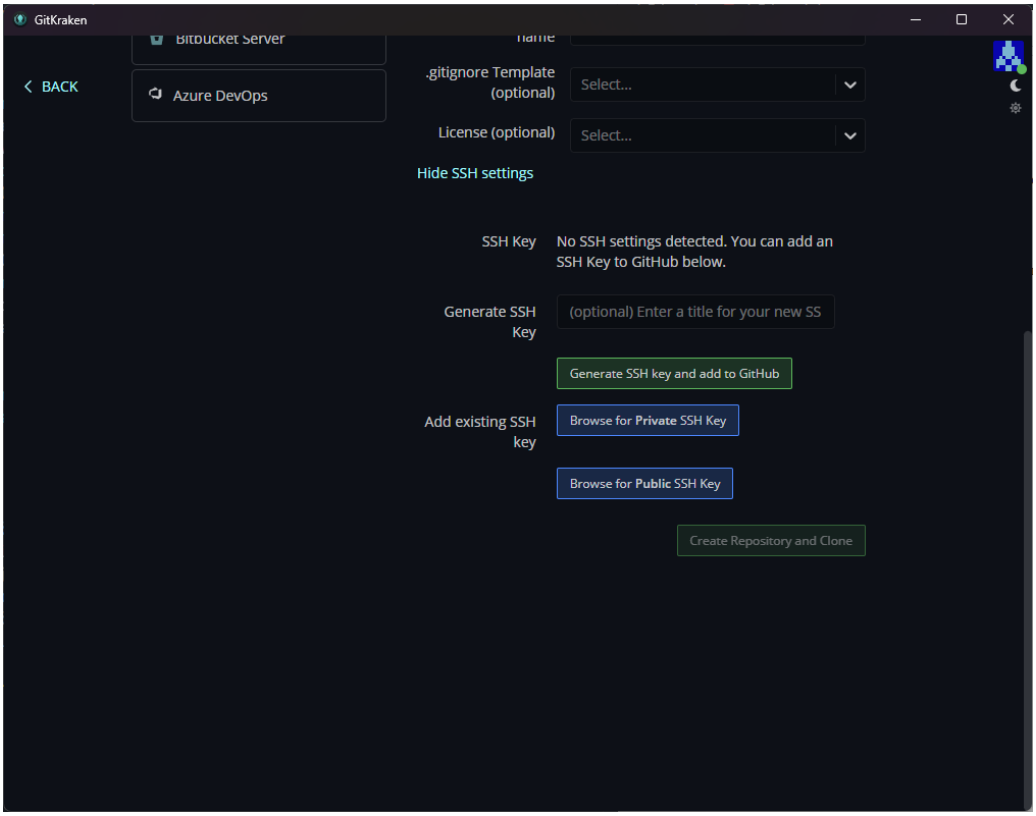
Listing 8: Przykładowy alias w terminalu

2.9 Klonowanie z SSH

Aby móc sklonować repozytorium z SSH należy najpierw wygenerować klucz SSH, a następnie dodać go do swojego konta na GitHubie. W przeciwieństwie do konfiguracji SSH w terminalu, w GitKrakenie nie trzeba robić nic, oprócz wygenerowania klucza i dodania go do konta na GitHubie. Po zrobieniu tego, można już klonować repozytorium z SSH.

```
$ git clone git@github.com:alkatraz445/SiNWO_2.git
```

Listing 9: Klonowanie z SSH w terminalu



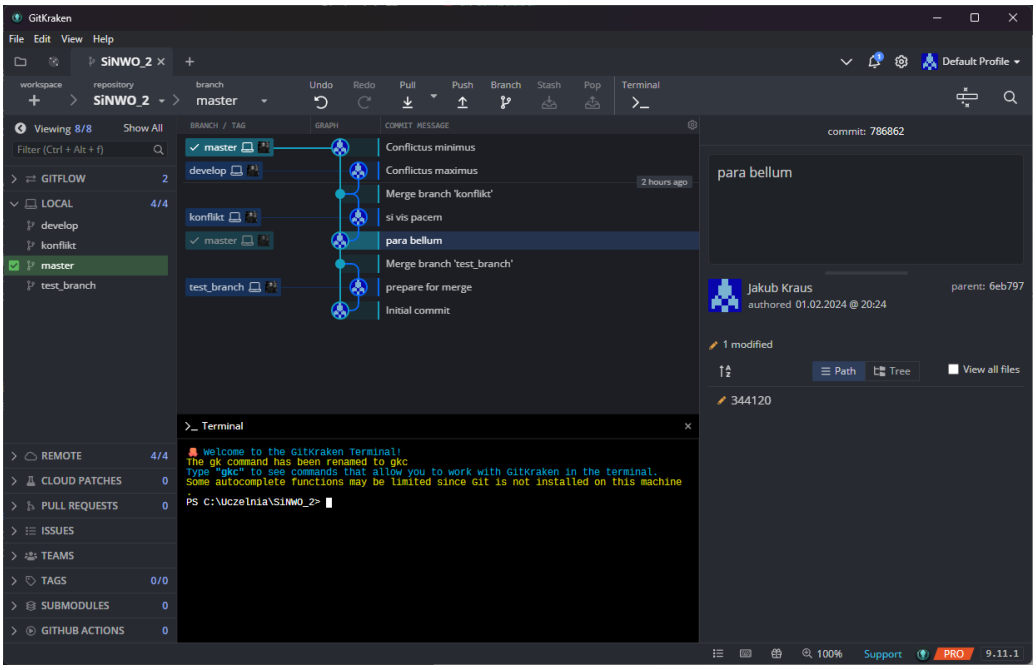
Rysunek 8: Klonowanie z SSH w GitKraken

2.10 Kopia zapasowa

GitKraken nie posiada opcji do tworzenia kopii zapasowej. Można natomiast użyć do tego celu polecenia archive w wbudowanym terminalu.

```
$ git archive --output=./SiNWO_2.zip --format=zip HEAD ./build
```

Listing 10: Kopia zapasowa w terminalu



Rysunek 9: Kopia zapasowa w GitKraken

2.11 GitFlow

Gitflow to model rozwoju oprogramowania oparty na systemie kontroli wersji Git. Został stworzony przez Vincenta Driessena i opisany w jego artykule. Model ten ma na celu ułatwienie pracy zespołom programistycznym, zwłaszcza w projektach oprogramowania o większym zakresie.

2.11.1 Gałąź Master

Jest to główna gałąź (branch) zawierająca kod produkcyjny. Każda zmiana akceptowana do produkcji jest zintegrowana z tej gałęzi.

2.11.2 Gałąź Develop

Gałąź ta jest używana jako gałąź podstawowa dla pracy zespołu. Zawiera najnowsze zmiany i funkcje, które są w trakcie rozwoju.

2.11.3 Gałąź Feature

Dla każdej nowej funkcji lub zadania tworzy się oddzielną gałąź. Zmiany te są wprowadzane i testowane na tej gałęzi, a następnie integrowane z gałęzią Develop po zakończeniu prac.

2.11.4 Gałąź Release

Przed wypuszczeniem nowej wersji oprogramowania, tworzona jest gałąź Release. Na tej gałęzi dokonuje się ostatnich poprawek, testów i przygotowań do wersji produkcyjnej.

2.11.5 Gałąź Hotfix

Jeśli po wypuszczeniu nowej wersji produkcyjnej pojawią się pilne błędy wymagające szybkiej naprawy, tworzona jest gałąź Hotfix. Po naprawieniu błędu, zmiany są integrowane zarówno z gałęzią Master, jak i Develop.

2.11.6 Schemat prac

- Nowe funkcje i zadania są rozwijane na gałęziach Feature.
- Po ukończeniu funkcji, gałąź Feature jest scalana z gałęzią Develop.
- Przed wydaniem nowej wersji tworzona jest gałąź Release. Na tej gałęzi dokonywane są ostatnie prace i testy.
- Po zakończeniu testów gałąź Release jest scalana z gałęzią Master, a także integrowana z gałęzią Develop, aby uwzględnić ostatnie zmiany.
- Na gałęzi Master można utworzyć gałąź Hotfix, aby szybko naprawić błędy na produkcji.
- Po naprawie, gałąź Hotfix jest integrowana zarówno z gałęzią Master, jak i Develop.

2.11.7 Podsumowanie

Model Gitflow pomaga w zarządzaniu procesem rozwoju oprogramowania, zwłaszcza w projektach, które wymagają równoczesnej pracy nad wieloma funkcjami. Dzięki klarownym gałęziom i regułom scalania, ten model przyczynia się do utrzymania porządku w repozytorium, a także umożliwia skuteczne zarządzanie wersjami oprogramowania.

3 Wnioski

Po przeprowadzeniu działań z wykorzystaniem GitKrakena, można stwierdzić, że narzędzie to dostarcza intuicyjny interfejs graficzny, ułatwiający pracę z systemem kontroli wersji Git. Funkcje takie jak "Blame", "Checkout", czy "Merge"są łatwo dostępne i wygodne w użyciu, co przyspiesza i ułatwia codzienną pracę programistyczną.

Rozwiązanie konfliktów scalania w GitKraken odbywa się sprawnie dzięki wbudowanemu mergetoolowi, który wspomaga użytkowników podczas rozstrzygania problemów. Jednak warto zauważyć, że umiejętność manualnego rozwiązywania konfliktów poprzez edycję plików jest równie istotna.

Podczas korzystania z GitKrakena, klonowanie repozytorium przy użyciu SSH jest zadaniem prostym do wykonania dzięki intuicyjnemu interfejsowi graficznemu. Działa to efektywnie, szczególnie dla użytkowników, którzy wolą unikać linii poleceń. Niestety tego nie można powiedzieć o tworzeniu kopii zapasowej, które wymaga użycia wbudowanego terminala. W tym przypadku, użytkownicy, którzy nie są zaznajomieni z polecaniami Git w linii poleceń, mogą mieć problemy z wykonaniem tego zadania.

W kontekście preferencji co do narzędzi do kontroli wersji, wybór GitKrakena wynika głównie z jego przyjaznego interfejsu i łatwości obsługi, co przekłada się na szybszą i bardziej efektywną pracę. Jednak dla zaawansowanych użytkowników, znajomość poleceń Git w linii poleceń pozostaje kluczowym elementem, a zrozumienie różnic pomiędzy tymi dwiema formami pracy może być korzystne w zróżnicowanym środowisku programistycznym.

Bibliografia

[1] *A successful Git branching model*. en. URL: `http://nvie.com/posts/a-successful-git-branching-model/` (term. wiz. 23. 01. 2024).

[2] *GitKraken Legendary Git Tools | GitKraken*. en-US. URL: `https://www.gitkraken.com` (term. wiz. 23. 01. 2024).

[3] Microsoft. pl-pl. URL: `https://www.microsoft.com/pl-pl/windows` (term. wiz. 23. 01. 2024).

[4] *What is Git Flow | How to use Git Flow | Learn Git*. en-US. URL: `https://www.gitkraken.com/learn/git/git-flow` (term. wiz. 23. 01. 2024).