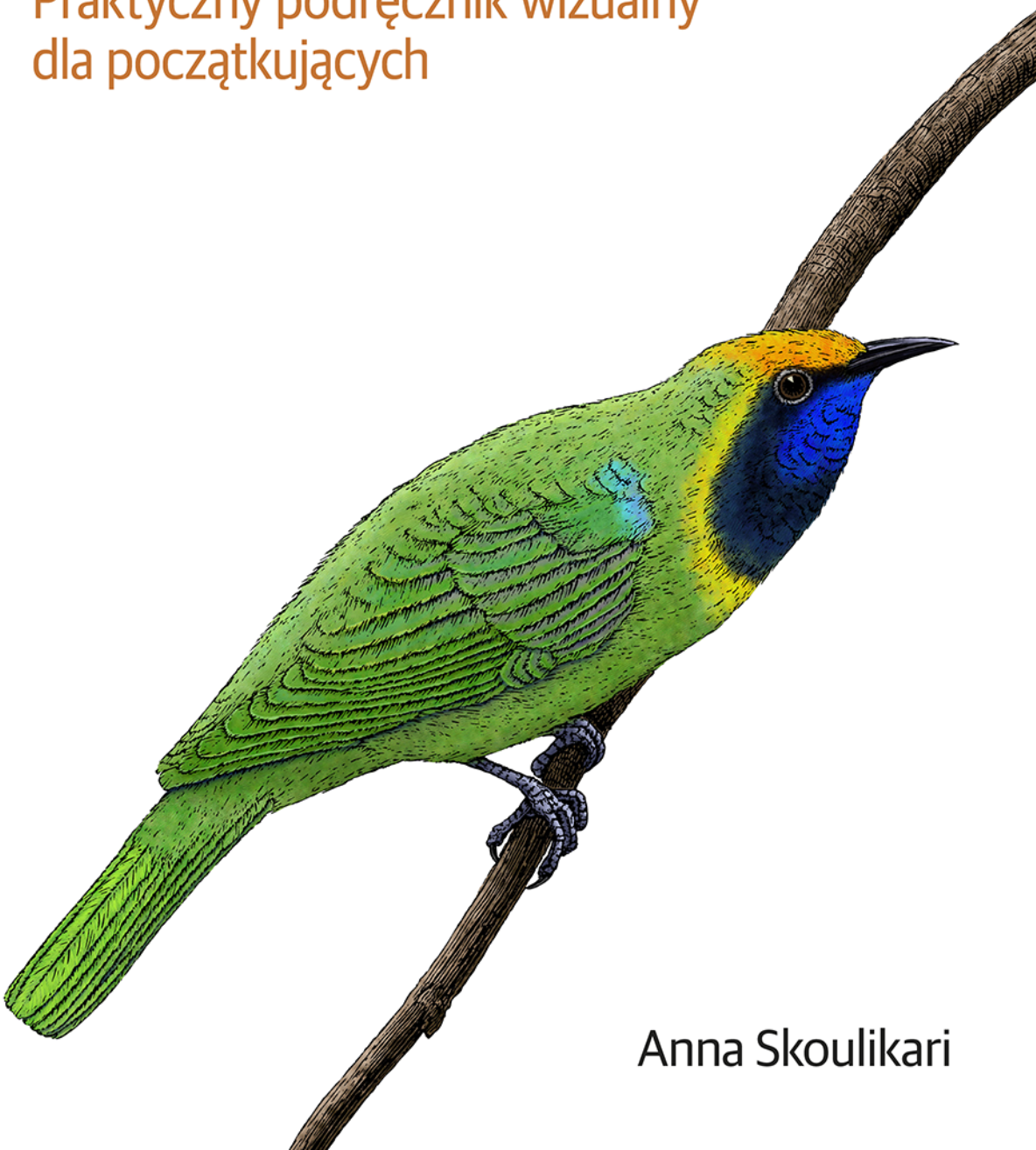
O'REILLY®

Helion

Naucz się Gita

Praktyczny podręcznik wizualny dla początkujących



Anna Skoulikari

Tytuł oryginału: Learning Git: A Hands-On and Visual Guide to the Basics of Git

Tłumaczenie: Jacek Janusz

ISBN: 978-83-289-1948-8

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/nagita
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

[spis treści]

[4]

Gałęzie

W poprzednim rozdziale poznałeś proces wykonywania commitów, a także wykonałeś swój pierwszy commit w repozytorium *rainbow*.

W tym rozdziale dowiesz się, czym są gałęzie i dlaczego ich używamy. Będziesz kontynuować wykonywanie commitów w repozytorium *rainbow*, a także sprawdzisz, jaki związek mają te działania z gałęziami istniejącymi w Twoim projekcie. Wreszcie utworzysz nową gałąź i dowiesz się, jak się na nią przełączyć (lub jak ją zmienić). Ponadto w trakcie procesu wykonywania kolejnych commitów w repozytorium *rainbow* zapoznasz się z koncepcją plików niezmodyfikowanych oraz zmodyfikowanych, a także dowiesz się, w jaki sposób commity są ze sobą powiązane.

Stan repozytorium lokalnego

W rozdziale 2. utworzyliśmy diagram Gita zawierający cztery ważne składniki: katalog roboczy, przechowalnię, historię commitów i repozytorium lokalne. W punkcie „Zwizualizuj to 4.1" widzimy diagram Gita, który przedstawia stan repozytorium *rainbow* na początku tego rozdziału.

[ZWIZUALIZUJ TO 4.1]

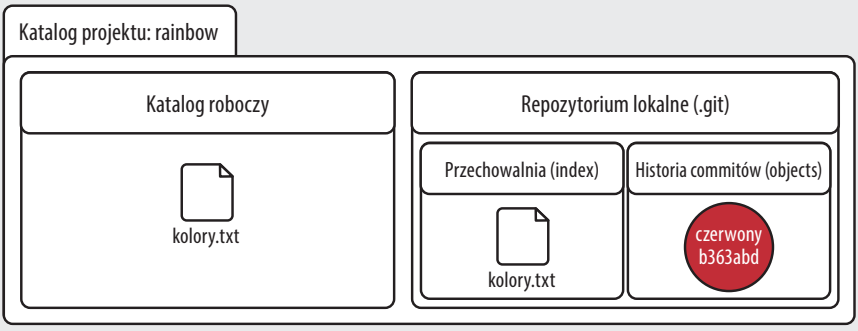


Diagram Gita pokazujący stan repozytorium *rainbow* na początku rozdziału 4., zawierającego jednego commita — czerwonego.

[UWAGA]

Od tego momentu diagramy w punktach „Zwizualizuj to” będą w przypadku commitów zawierać jedynie nazwę koloru lub jej skrót. Nie będzie się już pojawiać pierwszych siedem znaków skrótu commita.

Aby bardziej skoncentrować się na historii commitów, wprowadzę teraz nowy schemat o nazwie *Diagram repozytorium*. Diagram repozytorium zawiera tylko reprezentację historii commitów w repozytorium oraz odpowiednich gałęzi i odniesień. Repozytorium lokalne jest przedstawiane jako prostokąt, w którego lewym górnym narożniku znajduje się nazwa repozytorium. W punkcie „Zwizualizuj to 4.2” pokazano obecny stan repozytorium *rainbow* w formie diagramu repozytorium.

[ZWIZUALIZUJ TO 4.2]

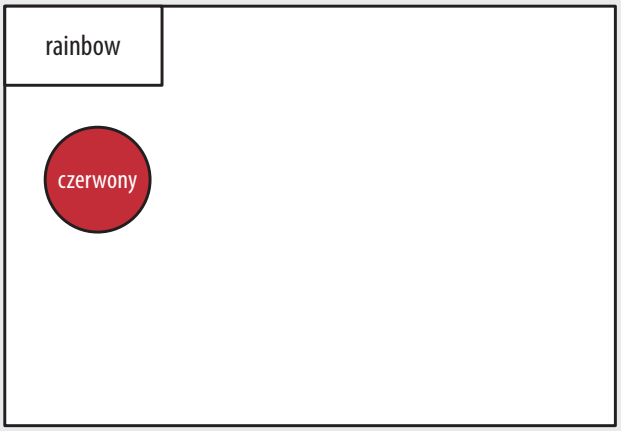


Diagram repozytorium pokazujący obecny stan repozytorium *rainbow* zawierającego jednego commita — czerwonego.

Dlaczego używamy gałęzi?

Zanim przejdziemy do bardziej szczegółowego opisu gałęzi w Gicie, chciałabym wyjaśnić, dlaczego są one tak przydatne. Istnieją dwa główne powody korzystania z gałęzi:

- umożliwiają pracę nad tym samym projektem na różne sposoby,
- pozwalają wielu osobom pracować nad tym samym projektem w tym samym czasie.

Gałąź można traktować jako oddzielną linię rozwojową. Projekt Gita może zawierać wiele gałęzi (lub linii rozwojowych). Każda z nich jest samodzielną wersją projektu. W zależności od potrzeb osób pracujących nad projektem różne projekty Gita mogą wykorzystywać gałęzie na odmienne sposoby.

Jednym z powszechnie stosowanych schematów wykorzystania gałęzi jest używanie jednej, oficjalnej, głównej linii rozwojowej, a na jej podstawie tworzenie gałęzi drugorzędnych, zwanych **gałęziami tematycznymi**, które są używane do pracy tylko nad określoną częścią projektu. Takie gałęzie tematyczne są krótkotrwałe — ostatecznie łączy się je lub scala z główną gałęzią, a następnie usuwa. Dwa procesy, których można użyć do zintegrowania jednej gałęzi z inną, nazywane są złączaniem i przebazowaniem. Omówimy je bardziej szczegółowo w rozdziałach 5., 9., 10., 11. i 12. (tak, to są obszerne zagadnienia!).

Ten schemat polegający na użyciu jednej, głównej linii rozwojowej jest metodą, którą zastosujesz w projekcie *Rainbow*. Aby powyższe dywagacje uczynić bardziej zrozumiałymi, przyjrzyjmy się punktowi „Przykładowy projekt Book 4.1", by się dowiedzieć, w jaki sposób gałęzie mogą być używane w realnym projekcie.

Przykładowy projekt Book 4.1

Załóżmy, że oficjalną gałęzią mojego projektu *Book* jest `main`. Nie chcę umieszczać kolejnych danych w oficjalnej linii rozwojowej, dopóki mój redaktor nie sprawdzi i nie zatwierdzi tej gałęzi. Tak więc za każdym razem, gdy chcę edytować jakiś rozdział, mogę utworzyć dodatkową gałąź, pracować nad nią, a następnie przesłać redaktorowi do sprawdzenia. Po zatwierdzeniu przez niego pracy wykonanej w tej gałęzi mogę ją połączyć z gałęzią `main`.

Jeśli w pewnym momencie zdecyduję się pracować nad książką wspólnie z kolegą, każde z nas może wykorzystywać własne, drugorzędne gałęzie. Dopiero gdy praca nad daną gałęzią drugorzędną zostanie zatwierdzona zarówno przez jej autora, jak i redaktora, będzie uznana za gotową do połączenia z gałęzią `main`.

Gdy już wiesz, dlaczego używamy gałęzi, postaramy się dokładniej wyjaśnić, jak działają one w Gicie.

CZYM DOKŁADNIE SĄ GAŁĘZIE W GICIE?

Gałęzie w Gicie są ruchomymi wskaźnikami do commitów. Podczas wyświetlania za pomocą polecenia `git log` listy commitów w repozytorium lokalnym można zobaczyć informacje o tym, które gałęzie wskazują na jakie commity. Przejdź do punktu „Zrób to sam 4.1", aby sprawdzić, jak to wygląda w przypadku repozytorium *rainbow*.

[ZRÓB TO SAM 4.1]

```
1 ~/Pulpit/rainbow$ git log
  commit b363abdf6e7e92adb908b231ef028a6574126abb (HEAD -> main)
  Author: annaskoulikari <gitlearningjourney@gmail.com>
  Date:   Wed Jul 31 15:47:04 2024 +0200

      czerwony
```

Ważne:

- W danych wyjściowych otrzymanych po wykonaniu polecenia `git log` obok skrótu commita jest widoczny w nawiasach tekst `HEAD -> main`.

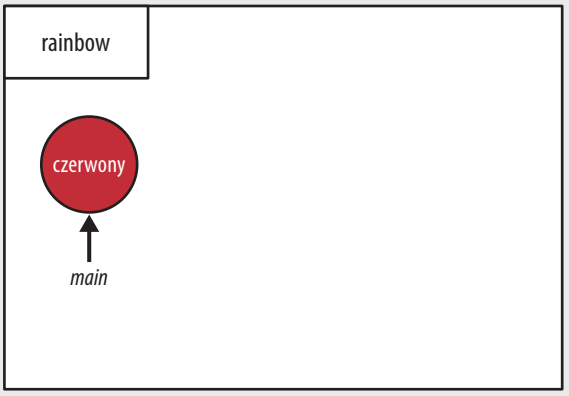
Gałąź lub gałęzie, których nazwy pojawiają się w nawiasach obok określonego skrótu commita w danych wyjściowych wygenerowanych przez polecenie `git log`, wskazują na ten commit.

[UWAGA]

Słowo HEAD (zapisane wielkimi literami) nie jest gałęzią. Czym jest, wyjaśnimy wkrótce w podrozdziale „Co to jest HEAD?”.

Gałąź `main` w repozytorium *rainbow* wskazuje na czerwonego commita. Zostało to zilustrowane w punkcie „Zwizualizuj to 4.3”.

[ZWIZUALIZUJ TO 4.3]



Gałąź `main` wskazuje na czerwonego commita w repozytorium *rainbow*.

W dalszej części tego rozdziału będziesz kontynuować wykonywanie commitów w gałęzi `main`, a dzięki temu zobaczysz, że po każdej operacji wskazuje ona nowo utworzony commit.

[UWAGA]

Główną gałęzią w repozytorium *Rainbow* jest gałąź `main`. W pierwszej części tej książki, w której w projekcie *Rainbow* zajmujesz się jedynie repozytorium lokalnym, będziesz pracować tylko z gałęziami lokalnymi. W drugiej części książki, gdy wprowadzimy repozytoria zdalne, poznasz koncepcję zdalnych i zdalnie śledzonych gałęzi.

Aby lepiej zrozumieć koncepcję gałęzi jako ruchomych wskaźników do commitów, przejdź do punktu „Zrób to sam 4.2".

[ZRÓB TO SAM 1.11]

1. W otwartym oknie systemu plików przejdź do katalogu projektu *rainbow*.
2. Wyświetl wszystkie ukryte pliki i katalogi w katalogu projektu *rainbow*. Więcej informacji na temat wyświetlania ukrytych plików i katalogów można znaleźć w podrozdziale „Wyświetlanie zawartości katalogów" dostępnym w rozdziale 1.
3. Będąc w oknie systemu plików w katalogu projektu *rainbow*, przejdź do pliku *.git/refs/heads/main*.
4. Dwukrotnie kliknij plik *main*. W przypadku systemu Linux używanego w tej książce plik zostanie automatycznie otwarty w prostym edytorze tekstowym o nazwie xed. W systemie Microsoft Windows plik można otworzyć za pomocą edytora tekstu o nazwie Notatnik.

   Edytory xed i Notatnik będą używane tylko do przeglądania zawartości niektórych plików istniejących w katalogu *.git*. Mogą się one różnić od edytora tekstowego używanego w celu zarządzania plikami w projekcie *Rainbow*.

Ważne:

- Po wykonaniu kroku 4. zauważysz w treści pliku *main* skrót czerwonego commita z repozytorium *rainbow*.

Aby w kroku 3. z punktu „Zrób to sam 4.2" dostać się do pliku *main*, należało przejść do katalogu *.git*, katalogu *refs*, a następnie do katalogu *heads*. Termin „refs" oznacza „referencje". Katalog *heads* przechowuje po jednym pliku dla każdej lokalnej gałęzi z repozytorium lokalnego. W tej chwili masz tylko jedną gałąź lokalną (`main`), więc w tym katalogu znajduje się tylko jeden plik. Możesz przyjąć, że plik ten przechowuje zawsze główny element tej gałęzi — innymi słowy, jego najnowszego commita.

Gdy dowiedziałeś się już, czym są gałęzie i jak działają, przeanalizujmy dokładniej konwencje nazewnicze stosowane w przypadku gałęzi początkowej.

TROCHĘ O HISTORII GITA — MASTER I MAIN

Podczas inicjowania repozytorium lokalnego za pomocą polecenia `git init` wywołanego bez żadnych opcji Git w tle tworzy gałąź o nazwie `master`. Jednak słowo „master" nie jest uważane za termin inkluzywny, więc w ostatnich latach duża część społeczności Gita zdecydowała się jako domyślnej nazwy gałęzi używać słowa `main` (lub innego).

To dlatego w rozdziale 2. podczas inicjowania repozytorium *rainbow* użyto polecenia `git init` z opcją `-b` i przekazano parametr `main`. Angielskie słowo *main* nie jest w żaden sposób specjalne — mogłeś nadać gałęzi początkowej zupełnie inną nazwę poprzez przekazanie odpowiedniej wartości do polecenia `git init -b`. Gdy wykonałeś pierwszego commita w repozytorium *rainbow*, spowodowało to zaktualizowanie gałęzi `main`, aby wskazywała na tego commita. W projekcie *Rainbow* gałąź `main` jest główną linią rozwojową.

W trakcie nauki Gita natkniesz się na wiele zasobów edukacyjnych, które wciąż używają nazwy `master`. Ważne jest, aby zrozumieć, że nie ma w tym nic nadzwyczajnego — jest to po prostu domyślna nazwa pierwszej gałęzi utworzonej w Gicie.

W tym momencie jesteś gotowy, aby dodać nowy kolor do pliku *kolory.txt* i wykonać kolejnego commita. Zanim jednak do tego przejdziemy, chciałabym przedstawić kilka dodatkowych stanów, w jakich może się znajdować plik w projekcie Gita.

Pliki niezmodyfikowane i zmodyfikowane

W rozdziale 2. wprowadziłam pojęcie plików nieśledzonych i śledzonych. Git „ma świadomość" istnienia pliku *kolory.txt*, ponieważ został on uwzględniony w commicie, a zatem jest to plik śledzony.

Pliki śledzone istniejące w katalogu roboczym mogą występować w jednym z dwóch stanów. **Pliki niezmodyfikowane** to pliki dostępne w katalogu roboczym, które nie były edytowane od ostatniego commita. Gdy plik w katalogu roboczym zostanie zmodyfikowany (i zapisany za pomocą edytora tekstowego), stanie się **plikiem zmodyfikowanym**. Od ostatniego commita nie modyfikowałeś pliku *kolory.txt*, dlatego jest on plikiem niezmodyfikowanym.

[UWAGA]

Aby Git wiedział, że plik był modyfikowany, *musi* zostać zapisany za pomocą edytora tekstowego. Jeśli dokonałeś edycji pliku, ale nie zapisałeś zmian, Git wyświetli go jako plik niezmodyfikowany.

W rozdziale 3. dowiedziałeś się o poleceniu `git status`, które pokazuje stan katalogu roboczego i przechowalni oraz różnicę między nimi. Wyświetla ono listę wszystkich plików *zmodyfikowanych* i informuje, czy zostały dodane do przechowalni. Nie prezentuje jednak listy plików niezmodyfikowanych.

Przejdź do punktu „Zrób to sam 4.3", aby użyć polecenia `git status` i zobaczyć, że plik *kolory.txt* jest plikiem niezmodyfikowanym. Następnie zmodyfikuj go i zapisz oraz ponownie użyj polecenia *git status*, aby sprawdzić, jak zmienia się stan pliku.

[ZRÓB TO SAM 4.3]

1. ```
   ~/Pulpit/rainbow$ git status
   Na gałęzi main
   nic do złożenia, drzewo robocze czyste
   ```

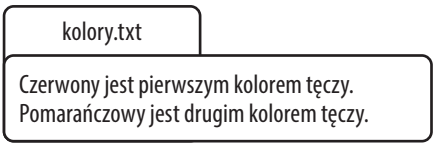
2. Będąc w katalogu projektu *rainbow*, otwórz plik *kolory.txt* w edytorze tekstowym, wprowadź zdanie „Pomarańczowy jest drugim kolorem tęczy" w wierszu 2., a następnie zapisz plik.

3. ```
   ~/Pulpit/rainbow$ git status
   Na gałęzi main
   Zmiany nieprzygotowane do złożenia:
     (użyj "git add <plik>...", żeby zmienić, co zostanie złożone)
     (użyj "git restore <plik>...", aby odrzucić zmiany w katalogu roboczym)
       zmieniono:       kolory.txt

   brak zmian dodanych do zapisu (użyj "git add" i/lub "git commit -a")
   ```

Ważne:

- W kroku 1. plik *kolory.txt* jest plikiem niezmodyfikowanym. Nie został jeszcze uwzględniony w danych wyjściowych polecenia `git status`.
- W kroku 3. plik *kolory.txt* jest plikiem zmodyfikowanym. Jest on teraz wyświetlany po wykonaniu polecenia `git status`.
- Plik *kolory.txt* nie został przygotowany do commita — innymi słowy, nie został dodany do przechowalni.

Byłeś właśnie świadkiem tego, jak plik *kolory.txt* po jego edycji i zapisaniu zmian zmienił stan z pliku niezmodyfikowanego na plik zmodyfikowany. Przykładową zawartość pliku *kolory.txt* przedstawiono na rysunku 4.1.



RYSUNEK 4.1.
Plik kolory.txt po dodaniu zdania o kolorze pomarańczowym

W punkcie „Zrób to sam 4.4” dodasz plik *kolory.txt* do przechowalni, aby mógł zostać uwzględniony w następnym commicie, a następnie sprawdzisz, jak zmienił się wynik polecenia `git status`.

[ZRÓB TO SAM 4.4]

```
1 ~/Pulpit/rainbow$ git add kolory.txt
2 ~/Pulpit/rainbow$ git status
  Na gałęzi main
  Zmiany do złożenia:
    (użyj "git restore --staged <plik>...", aby wycofać)
      zmieniono:       kolory.txt
```

Ważne:

- Plik *kolory.txt* został przygotowany do commita — innymi słowy, został dodany do przechowalni.

Właśnie zakończyłeś pierwszy etap procesu wykonywania commita, który polegał na dodaniu plików do przechowalni. W kolejnym kroku zakończysz wykonywanie commita i zobaczysz, jak wpłynie to na gałąź `main`.

Wykonywanie commitów na gałęzi

Jesteś gotowy, aby wykonać drugiego commita w repozytorium *rainbow*. Tym razem dodasz kolor pomarańczowy, więc komunikatem zatwierdzenia będzie tekst „pomarańczowy". Przejdź do punktu „Zrób to sam 4.5", aby wykonać commita.

[UWAGA]

Jeśli dane wyjściowe uzyskane po wykonaniu polecenia `git log` wykraczają poza rozmiar okna wiersza poleceń, należy nacisnąć klawisz *Enter* (*Return*) lub użyć strzałki w dół, aby wyświetlić pozostałą treść. W celu wyjścia z obsługi polecenia, należy nacisnąć *Q*.

[ZRÓB TO SAM 4.5]

1
```
~/Pulpit/rainbow$ git commit -m "pomarańczowy"
[main 0b39b66] pomarańczowy
 1 file changed, 1 insertion(+)
```

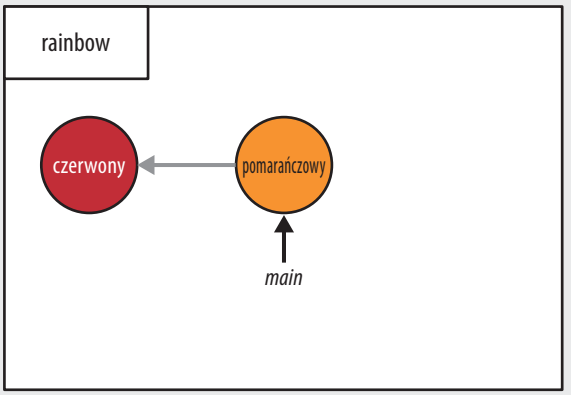
2
```
~/Pulpit/rainbow$ git log
commit 0b39b66318f158a75bb8283f3595e774b2590b38 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Aug 5 14:48:57 2024 +0200

    pomarańczowy

commit b363abdf6e7e92adb908b231ef028a6574126abb
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Wed Jul 31 15:47:04 2024 +0200

  czerwony
```

Ważne:

- Wykonałeś kolejnego commita — pomarańczowego. Skrót dla tego commita jest w tej książce równy `0b39b66318f158a75bb8283f3595e774b2590b38`. W Twoim przypadku wartość ta będzie inna.
- Tekst `HEAD -> main` pojawia się w nawiasach obok commita pomarańczowego.

W punkcie „Zwizualizuj to 4.4" przedstawiono stan repozytorium *rainbow* po wykonaniu poleceń z punktu „Zrób to sam 4.5".

[ZWIZUALIZUJ TO 4.4]



Repozytorium *rainbow* po wykonaniu commita pomarańczowego.

Ważne:

- Pojawił się drugi commit — pomarańczowy.
- Commit pomarańczowy wskazuje na commit czerwony.
- Gałąź `main` wskazuje na commit pomarańczowy.

W punkcie „Zwizualizuj to 4.4” widzimy, że szara strzałka prowadzi od commita pomarańczowego do czerwonego. Ta strzałka reprezentuje powiązanie nadrzędne. Każdy commit, który nie jest pierwszy w repozytorium, ma commita nadrzędnego (niektóre mogą mieć więcej niż jednego rodzica — omówimy to w rozdziale 5.). Rodzicem commita pomarańczowego jest commit czerwony, dlatego też commit pomarańczowy wskazuje na commit czerwony.

Powiązania nadrzędne opisują, w jaki sposób commity są ze sobą połączone. Zrozumienie tych struktur pozwala na zwizualizowanie historii commitów i prześledzenie, jakie prace zostały wykonane w poszczególnych gałęziach.

[UWAGA]

W diagramach „Zwizualizuj to” szare strzałki są używane do reprezentowania połączeń z rodzicami, a czarne strzałki służą do reprezentowania wskaźników gałęzi.

Aby sprawdzić, co jest rodzicem danego commita, można użyć polecenia `git cat-file` z opcją `-p`, której należy przekazać skrót commita: `git cat-file -p <skrót_commitu>`. Większość użytkowników Gita prawdopodobnie nie będzie używać tego polecenia w swojej codziennej pracy. Ponieważ jednak jest ono narzędziem przydatnym podczas nauki, przetestujemy je w działaniu.

Przejdź do punktu „Zrób to sam 4.6”, aby odczytać skrót commita dla rodzica commita pomarańczowego. Aby to zrobić, musisz do polecenia `git cat-file -p` przekazać skrót commita pomarańczowego. Najprostszym sposobem na wyszukanie skrótu określonego commita jest przejrzenie listy commitów wygenerowanej przez polecenie `git log`, a następnie skopiowanie interesującej nas wartości. Alternatywnym rozwiązaniem jest przeanalizowanie danych wyjściowych uzyskanych w punkcie „Zrób to sam 4.5” i przekazanie pierwszych siedmiu znaków skrótu commita, który tam występuje.

[ZRÓB TO SAM 4.6]

1. Pobierz skrót dla commita pomarańczowego (możesz go skopiować z danych wyjściowych uzyskanych po wykonaniu polecenia `git log` w punkcie ”Zrób to sam 4.5”). W kroku 2. musisz przekazać ten skrót jako argument do polecenia `git cat-file -p`. Możesz skopiować i wkleić cały skrót lub wprowadzić tylko pierwsze siedem znaków, jak pokazano poniżej.

2.
```
~/Pulpit/rainbow$ git cat-file -p 0b39b66
tree 3f8264686b8f76fa646049868c273023adac6c78
parent b363abdf6e7e92adb908b231ef028a6574126abb
author annaskoulikari <gitlearningjourney@gmail.com> 1722862137 +0200
committer annaskoulikari <gitlearningjourney@gmail.com> 1722862137 +0200

pomarańczowy
```

Ważne:

- W danych wyjściowych uzyskanych po wykonaniu polecenia `git cat-file -p` widzimy, że obok słowa `parent` widnieje skrót commita czerwonego. Twoje dane wyjściowe będą oczywiście zawierać inną wartość tego skrótu.

Wykonałeś drugiego commita w repozytorium *rainbow*, a w punkcie „Zwizualizuj to 4.4” mogłeś zauważyć, że podczas wykonywania operacji wskaźnik gałęzi przesuwa się tak, by zawsze wskazywać najnowszego commita. Teraz wyjaśnimy, jak można tworzyć inne gałęzie, a dzięki temu pracować nad innymi liniami rozwojowymi.

Tworzenie gałęzi

Obecnie w repozytorium *rainbow* jest dostępna tylko jedna gałąź lokalna o nazwie `main`. Aby wyświetlić listę gałęzi w repozytorium lokalnym, można użyć polecenia `git branch`. Aby utworzyć nową gałąź, należy do tego polecenia przekazać odpowiednią nazwę. Należy pamiętać, że nazwy gałęzi nie mogą zawierać spacji.

[PRZYDATNE POLECENIE]

`git branch`

Wyświetla gałęzie lokalne

`git branch <nazwa_nowej_gałęzi>`

Tworzy nową gałąź

Jak wspomniano w rozdziale 3., jeśli pracujesz nad projektem z innymi osobami, powinieneś sprawdzić, czy podobnie jak w przypadku komunikatów zatwierdzenia istnieją jakieś zasady określające sposób nazywania gałęzi. Ponieważ w projekcie *Rainbow* nie ma określonych reguł nazewnictwa gałęzi, użyjesz ogólnej nazwy `feature` dla gałęzi, którą utworzysz w punkcie „Zrób to sam 4.7”. Należy jednak pamiętać, że w prawdziwym projekcie Gita nazwa gałęzi będzie zazwyczaj bardziej opisowa i będzie odnosiła się do funkcji lub zagadnienia, nad którym pracujesz.

[ZRÓB TO SAM 4.7]

1.
```
~/Pulpit/rainbow$ git branch
* main
```

2.
```
~/Pulpit/rainbow$ git branch feature
```

3.
```
~/Pulpit/rainbow$ git branch
   feature
* main
```

4.
```
~/Pulpit/rainbow$ git log
commit 0b39b66318f158a75bb8283f3595e774b2590b38 (HEAD -> main, feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Aug 5 14:48:57 2024 +0200

    pomarańczowy

commit b363abdf6e7e92adb908b231ef028a6574126abb
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Wed Jul 31 15:47:04 2024 +0200

    czerwony
```

5. W oknie systemu plików przejdź do katalogu *.git/refs/heads*, aby zobaczyć, jakie pliki się w nim obecnie znajdują.

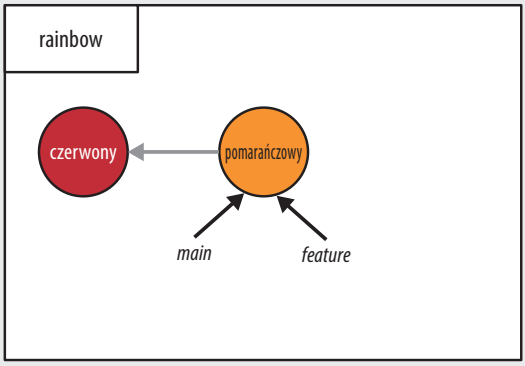
6. Otwórz plik *feature*. Powinien on zawierać skrót commita.

Ważne:

- W kroku 2. utworzyłeś nową gałąź o nazwie `feature`, która wskazuje na commita pomarańczowego.

Stan repozytorium *rainbow* po wykonaniu poleceń z punktu „Zrób to sam 4.7” został zilustrowany w punkcie „Zwizualizuj to 4.5”.

[ZWIZUALIZUJ TO 4.5]



Repozytorium *rainbow* po utworzeniu gałęzi `feature`.

W punkcie „Zwizualizuj to 4.5” widzimy, że obecnie istnieją dwie strzałki, reprezentujące gałęzie `main` i `feature`, wskazujące na commita pomarańczowego. Nowa gałąź będzie początkowo wskazywać na tego commita, w którym znajdowałeś się podczas jej tworzenia. W tym przypadku można stwierdzić, że „utworzyłeś gałąź `feature` z gałęzi `main`”, dlatego też obie wskazują teraz na tego samego commita.

W danych wyjściowych uzyskanych po wykonaniu polecenia `git log` w punkcie „Zrób to sam 4.7” widzimy, że obok commita *pomarańczowy* widnieje w nawiasach tekst `HEAD -> main, feature`. Jak już wiadomo, `main` i `feature` to nazwy gałęzi, ale czym jest `HEAD`?

Co to jest HEAD?

W danym momencie masz do czynienia z określoną wersją swojego projektu. Dlatego też znajdujesz się w danej gałęzi, która wskazuje na commita. `HEAD` jest po prostu wskaźnikiem, który informuje, w której gałęzi się znajdujesz. Nazwę `HEAD` zawsze zapisuje się wielkimi literami, jednak jest to po prostu konwencja — nie jest to żaden akronim.

[UWAGA]

Czasami commit może nie być wskazywany przez żadną gałąź. W terminologii Gita nazywa się to „stanem odłączonego wskaźnika HEAD". Taką sytuację przeanalizujemy w podrozdziale „Sprawdzanie commitów" z rozdziału 5.

Przejdź do punktu „Zrób to sam 4.8", aby przez sprawdzenie zawartości katalogu *.git* dowiedzieć się, czym jest `HEAD`.

[ZRÓB TO SAM 4.8]

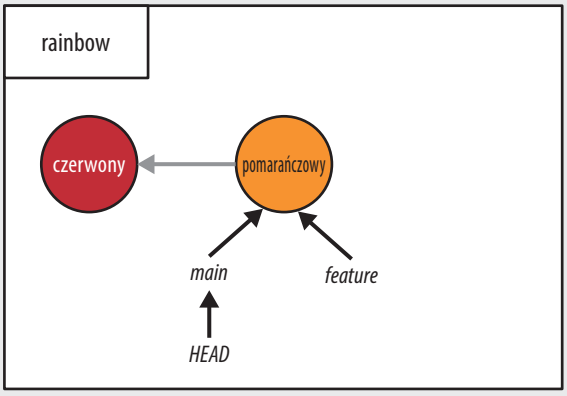
1. Za pomocą okna systemu plików przejdź do pliku *rainbow/.git/HEAD.*
2. Otwórz plik *HEAD*. Jego zawartość powinna być następująca: `ref: refs/heads/main`.

Ważne:

- Plik `HEAD` zawiera treść `ref: refs/heads/main` — jest to referencja do pliku *main* reprezentującego gałąź `main`.

Powyższe spostrzeżenie zostało zilustrowane w punkcie „Zwizualizuj to 4.6".

[ZWIZUALIZUJ TO 4.6]



Wskaźnik `HEAD` dostępny w repozytorium *rainbow* wskazuje na gałąź `main`..

Na rysunku z punktu „Zwizualizuj to 4.6" możemy zauważyć, że od wskaźnika `HEAD` prowadzi strzałka do gałęzi `main`. Oznacza to, że obecnie znajdujemy się na gałęzi `main`.

[UWAGA]

Nazwa HEAD (pisana wielkimi literami) nie powinna być mylona z katalogiem *heads*, do którego prowadzi ścieżka *git/refs/heads*. Katalog *heads* przechowuje plik dla każdej lokalnej gałęzi istniejącej w repozytorium lokalnym, podczas gdy HEAD wskazuje, w której gałęzi się znajdujesz, odwołując się do określonego pliku w katalogu *heads*. Te dwie nazwy można rozróżnić, ponieważ słowo HEAD jest zawsze pisane wielkimi literami.

Inną metodą sprawdzenia, w jakiej gałęzi się obecnie znajdujesz, jest analiza danych wyjściowych polecenia `git branch` lub `git log`. W przypadku danych wyjściowych polecenia `git branch` nazwa gałęzi, w której się obecnie znajdujesz, będzie miała obok siebie znak gwiazdki. Jeśli spojrzysz na dane wyjściowe uzyskane w punkcie „Zrób to sam 4.7", zobaczysz, że gwiazdka znajduje się obok gałęzi `main`. W danych wyjściowych polecenia `git log` pojawi się odpowiednia treść w nawiasach, która będzie oznaczać, że HEAD wskazuje na bieżącą gałąź.

Po utworzeniu nowej gałęzi możesz zacząć z niej korzystać, jednak na razie wciąż znajdujesz się w gałęzi `main`. Wykonasz więc operację przełączania na inną gałąź, co spowoduje przeniesienie wskaźnika HEAD do nowo utworzonej gałęzi `feature`.

Przełączanie na inne gałęzie

Aby w projekcie Gita pracować nad inną gałęzią (lub linią rozwojową), musisz się na nią przełączyć.

Obecnie masz dwie gałęzie: `main` i `feature`. Jak jednak właśnie zauważyłeś, samo utworzenie gałęzi w Gicie nie oznacza automatycznego przełączenia się na nią. Musisz wyraźnie poinstruować Gita, że chcesz się przełączyć na określoną gałąź. Można to zrobić za pomocą poleceń `git switch` lub `git checkout`, w których przekazuje się nazwę gałęzi, na którą chce się przełączyć.

[UWAGA]

Jeśli używasz Gita w wersji starszej niż 2.23, nie będziesz mieć dostępu do polecenia `git switch`, więc musisz użyć polecenia `git checkout`. Polecenie `git checkout` jest dostępne dla wszystkich użytkowników Gita.

[PRZYDATNE POLECENIE]

`git switch <nazwa_gałęzi>`

Przełącza na określoną gałąź.

`git checkout <nazwa_gałęzi>`

Przełącza na określoną gałąź.

Jedynym rezultatem po wykonaniu polecenia `git switch` jest przełączenie się na określoną gałąź, natomiast polecenie `git checkout` może wykonywać więcej działań. Więcej o tym drugim opowiemy w podrozdziale „Sprawdzanie commitów" z rozdziału 5.

Od tej pory w punktach „Zrób to sam" będziemy używać składni `git switch`, ponieważ jest to wyspecjalizowane polecenie dostępne w najnowszych wersjach Gita. Zawsze jednak możesz zastosować polecenie `git checkout`, ponieważ są one sobie równoważne.

Polecenie `git switch` (lub `git checkout`) wykonuje trzy operacje, gdy jest używane w celu przełączania na inną gałąź:

1. Odpowiednio zmienia wskaźnik `HEAD`, tak by wskazywał na gałąź, na którą się przełączasz.
2. Wypełnia przechowalnię migawką commita, na którego się przełączasz.
3. Kopiuje zawartość przechowalni do katalogu roboczego.

[UWAGA]

Jeśli chcesz sobie przypomnieć, czym są przechowalnia i katalog roboczy, zapoznaj się z podrozdziałem „Ważne składniki Gita" z rozdziału 2.

W skrócie: gdy przełączasz się na inną gałąź, zmieniasz także commita, którego używasz, ale pod warunkiem, że dwie gałęzie wskazują na dwa różne commity. W tej chwili obie gałęzie w repozytorium *rainbow* wskazują na tego samego commita, więc zostanie wykonana tylko pierwsza operacja, ponieważ commit, na którym się znajdujesz, nie ulegnie zmianie. W rozdziale 5. zostanie przeanalizowany przykład, w którym zostaną wykonane wszystkie trzy wymienione tutaj akcje.

Przejdź do punktu „Zrób to sam 4.9”, by przełączyć się na gałąź `feature`.

[ZRÓB TO SAM 1.11]

1.
```
~/Pulpit/rainbow$ git branch
   feature
* main
```

2.
```
~/Pulpit/rainbow$ git switch feature
Przełączono na gałąź "feature"
```

3.
```
~/Pulpit/rainbow$ git branch
* feature
  main
```

4.
```
~/Pulpit/rainbow$ git log
commit 0b39b66318f158a75bb8283f3595e774b2590b38 (HEAD -> feature, main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Aug 5 14:48:57 2024 +0200

    pomarańczowy

commit b363abdf6e7e92adb908b231ef028a6574126abb
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Wed Jul 31 15:47:04 2024 +0200

    czerwony
```

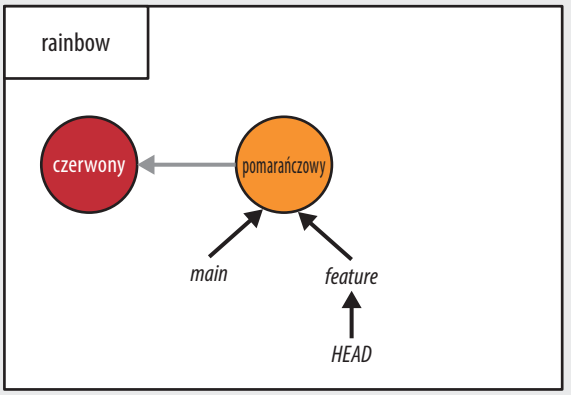
5. Korzystając z okna systemu plików, przejdź do pliku *rainbow/.git/HEAD,* a następnie otwórz go w nowym oknie. Zobaczysz, że zawartość pliku uległa zmianie i odnosi się teraz do gałęzi `feature`: `refs/heads/feature`.

Ważne:

- W kroku 4. wykonanie polecenia `git log` spowodowało wygenerowanie danych wyjściowych, których zawartość informuje, że `HEAD` wskazuje teraz na gałąź `feature`.
- Przełączyłeś się na gałąź `feature`, więc to jest obszar, w którym będziesz teraz pracować.

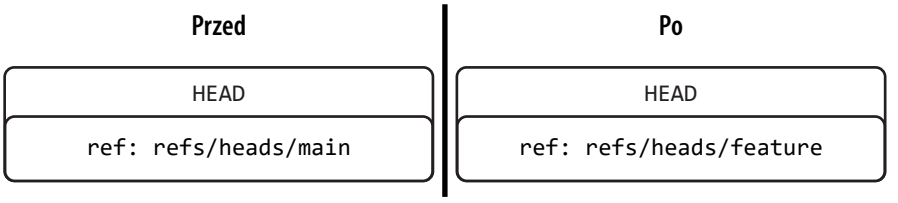
Powyższe wnioski przedstawiono w sposób graficzny w punkcie „Zwizualizuj to 4.7”.

[ZWIZUALIZUJ TO 4.7]



Wygląd repozytorium *rainbow* po przełączeniu się na gałąź `feature`.

Jak widać, przełączenie gałęzi zmieniło zawartość pliku `HEAD` w katalogu *.git*. Na rysunku 4.2 przedstawiono treść pliku `HEAD` przed i po przełączeniu gałęzi.



RYSUNEK 4.2.
Zawartość pliku HEAD przed i po przełączeniu z gałęzi main na gałąź feature w repozytorium rainbow

Po przełączeniu się na gałąź `feature` zobaczmy, co się stanie, gdy będziesz w niej pracował.

Praca w innej gałęzi

Znajdujesz się obecnie w gałęzi `feature`, a w punkcie „Zrób to sam 4.10” dodasz kolor żółty do repozytorium *rainbow*.

[ZRÓB TO SAM 4.10]

1. Będąc w katalogu projektu *raibow*, otwórz w edytorze tekstowym plik *kolory.txt*, a następnie umieść w trzecim wierszu zdanie „Żółty jest trzecim kolorem tęczy”. Po wykonaniu tej czynności zapisz plik.

2.
```
~/Pulpit/rainbow$ git add kolory.txt
```

3.
```
~/Pulpit/rainbow$ git commit -m "żółty"
[feature d5e4b3d] żółty
 1 file changed, 1 insertion(+)
```

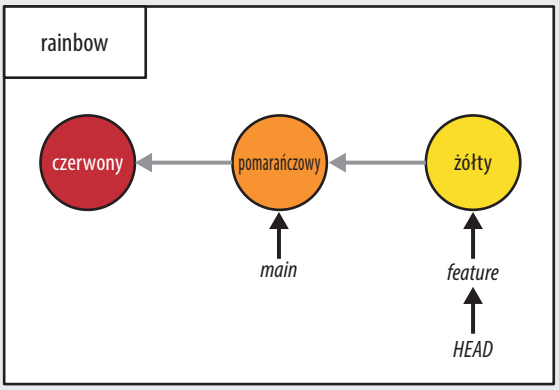
4.
```
~/Pulpit/rainbow$ git log
commit d5e4b3dfe573ef3b7464ab7b3c2c70bf142a9f7f (HEAD -> feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Wed Aug 7 14:28:07 2024 +0200

    żółty

commit 0b39b66318f158a75bb8283f3595e774b2590b38 (main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Aug 5 14:48:57 2024 +0200

    pomarańczowy

commit b363abdf6e7e92adb908b231ef028a6574126abb
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Wed Jul 31 15:47:04 2024 +0200

    czerwony
```

Ważne:

- Gałąź `feature` wskazuje na najnowszego commita — żółtego.
- Gałąź `main` nadal wskazuje na commita pomarańczowego.

Powyższe wnioski zilustrowano w punkcie „Zwizualizuj to 4.8”.

[ZWIZUALIZUJ TO 4.8]



Wygląd repozytorium *rainbow* po wykonaniu commita żółtego

Jak wcześniej wspomniano, gdy wykonujesz commita, gałąź, w której się znajdujesz, aktualizuje ustawienia, aby wskazywać na nowo utworzonego commita. Gałęzie `main` i `feature` nie wskazują już na tego samego commita, ponieważ `feature` została odpowiednio zaktualizowana, aby obecnie wskazywać na nowego commita żółtego. `HEAD` nadal wskazuje na gałąź `feature`.

Podsumowanie

W tym rozdziale wprowadziliśmy pojęcie gałęzi jako sposobu na rozróżnianie odmiennych linii rozwojowych, a także udowodniliśmy, że w praktyce są one ruchomymi wskaźnikami do commitów. Wyjaśniliśmy, dlaczego warto z nich korzystać, a także nauczyliśmy się gałęzie wyświetlać, tworzyć i zmieniać. Omówiliśmy, dlaczego pierwsza gałąź w repozytorium *rainbow* nazywa się `main`, a także dlaczego w innych zasobach edukacyjnych związanych z Gitem można się natknąć na gałęzie o nazwie `master`. Dowiedziałeś się również, że `HEAD` jest wskaźnikiem do gałęzi, w której się aktualnie znajdujesz.

Podczas wykonywania kolejnych commitów zaobserwowałeś, że pliki śledzone zmieniają stan z niezmodyfikowanego na zmodyfikowany, gdy są poddawane edycji. Zauważyłeś również, że to właśnie ta gałąź, w której się znajdujesz, wskazuje na najnowszego commita wykonanego w lokalnym repozytorium. Mogłeś się również dowiedzieć, w jaki sposób za pomocą powiązań nadrzędnych są ze sobą połączone commity.

Gdy już wyjaśniliśmy, jak używać gałęzi do jednoczesnej pracy nad różnymi liniami rozwojowymi, przejdźmy do rozdziału 5., gdzie dowiesz się, jak można je łączyć za pomocą operacji złączania.

Jest to niezwykle przystępny i dokładny przewodnik, pełen praktycznej wiedzy o Gicie.

Robert C. Martin aka Uncle Bob, twórca oprogramowania i autor książki *Czysty kod*

Git – kultowe, darmowe i dojrzałe oprogramowanie. Im większy zespół i im bardziej złożony projekt, tym ważniejsze jest skuteczne zarządzanie wersjami. Git umożliwia wyrafinowaną konfigurację i pozwala na zaspokajanie szczególnych potrzeb. Jeśli chcesz w pełni korzystać z jego potencjału, musisz zdobyć solidną wiedzę o podstawach tego systemu.

Dzięki tej książce dobrze zrozumiesz działanie Gita. Wiedza jest w niej przekazywana w prosty i konsekwentny sposób, a zastosowane techniki wizualne, opowiadane historie i liczne praktyczne ćwiczenia pozwolą Ci na skuteczną naukę krok po kroku. Stopniowo będziesz się zapoznawać z kluczowymi informacjami i dogłębnie zrozumiesz znaczenie poszczególnych terminów i koncepcji. Książkę docenią zwłaszcza osoby używające Gita w projektach prywatnych lub zawodowych, na przykład studenci i uczestnicy kursów programowania, młodsi programiści, specjaliści przetwarzania danych i pisarze techniczni.

Z tej książki rzeczywiście możesz się nauczyć Gita!

Ben Straub, współautor książki Pro Git

Dzięki książce dowiesz się, jak:

- pobierać oprogramowanie Git i inicjalizować repozytorium lokalne
- dodawać pliki do przechowalni i wykonywać commity
- tworzyć, przełączać i usuwać gałęzie
- złączać i przebazowywać gałęzie
- obsługiwać repozytoria zdalne
- używać żądań pobrania podczas współpracy z innymi użytkownikami

Anna Skoulikari korzysta ze swojej umiejętności opowiadania historii, aby uczyć Gita w sposób prosty, konkretny i wizualny. Prowadzi wysoko oceniane kursy Gita, w tym jeden dostępny online. Pracowała jako projektantka UX, programistka frontendów i dokumentalistka.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1948-8
9 788328 919488
Cena: 79,00 zł