Praca z repozytorium Git

Źródło: Maarten van den Heuvel, domena publiczna.

Najważniejsze informacje dotyczące repozytorium kodu znajdziesz w e-materiale Repozytorium kodu. Pora nauczyć się z niego korzystać.

Załóżmy, że pracujesz nad pewnym projektem. Ktoś prosi cię o jego kopię oraz o wprowadzenie kilku zmian w plikach. Co robisz? Kopiujesz projekt do oddzielnego katalogu, wprowadzasz zasugerowane zmiany i udostępniasz cały folder. Jeśli okaże się, że ty również chcesz wprowadzić zmiany w swojej wersji projektu, musisz skopiować odpowiednie pliki do oryginalnego katalogu.

Jednak przy większej liczbie plików (lub gdy opisana sytuacja będzie się powtarzać) taka „żonglerka" staje się bardzo uciążliwa, a ty najprawdopodobniej stracisz kontrolę nad projektem – zwyczajnie pogubisz się wśród kolejnych wersji tych samych plików.

Aby temu zapobiec, możesz wykorzystać system kontroli wersji Git i mechanizm pracy z gałęziami projektu (ang. *branch*).

Przedstawiony na wstępie przykład jest tylko jednym z wielu możliwych scenariuszy. Gałęzie możesz wykorzystać m.in. podczas pracy zespołowej, przy wprowadzaniu nowych funkcji oraz gdy chcesz testować tymczasowe wersje programów, bez wpływania na ich stabilne wersje. Zmiany wprowadzone w gałęzi możesz włączyć do wersji docelowej (gałęzi głównej) lub je anulować.

Twoje cele

- Wyjaśnisz, czym są gałęzie w systemie Git.
- Scharakteryzujesz podstawowe operacje na gałęziach: tworzenie i łączenie.
- Przeanalizujesz, do czego służy polecenie `pull request`.
- Opiszesz projekt za pomocą języka Markdown.
- Wyjaśnisz, do czego służy plik `.gitignore` w repozytorium.

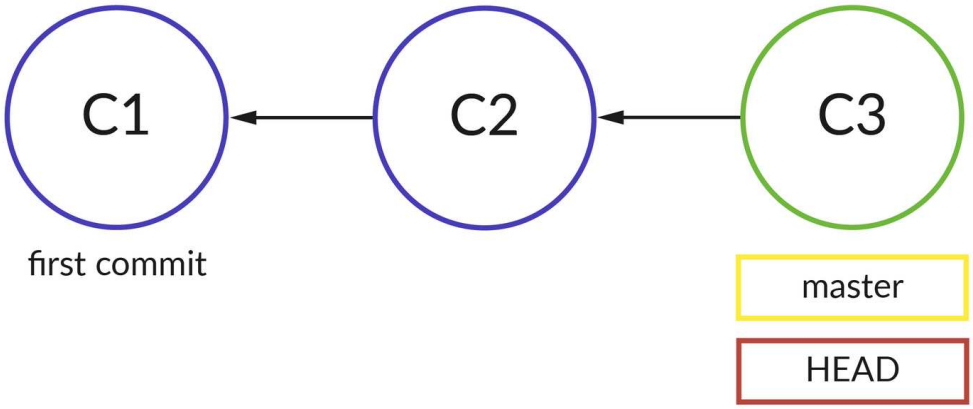
Przeczytaj

Czym są gałęzie w systemie Git?

W systemie kontroli wersji Git każda zatwierdzona zmiana projektu jest podpisana unikatowym identyfikatorem, a dokładnie: sumą kontrolną. Dzięki temu możemy śledzić historię zmian wprowadzonych do projektu. Dodanie jakiejkolwiek poprawki jest powiązane z poprzednią zmianą: powstaje w ten sposób relacja typu rodzic-dziecko.

Przedstawimy ciąg zmian wprowadzonych do projektu. Kolejne wersje na rysunkach oznaczymy symbolami C1, C2, . . ., Cn.

Zwrot strzałek na przedstawionych rysunkach wskazuje na rodzica, od którego pochodzi dany *commit*.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Między obiektami pokazanymi na rysunku zachodzą następujące relacje:

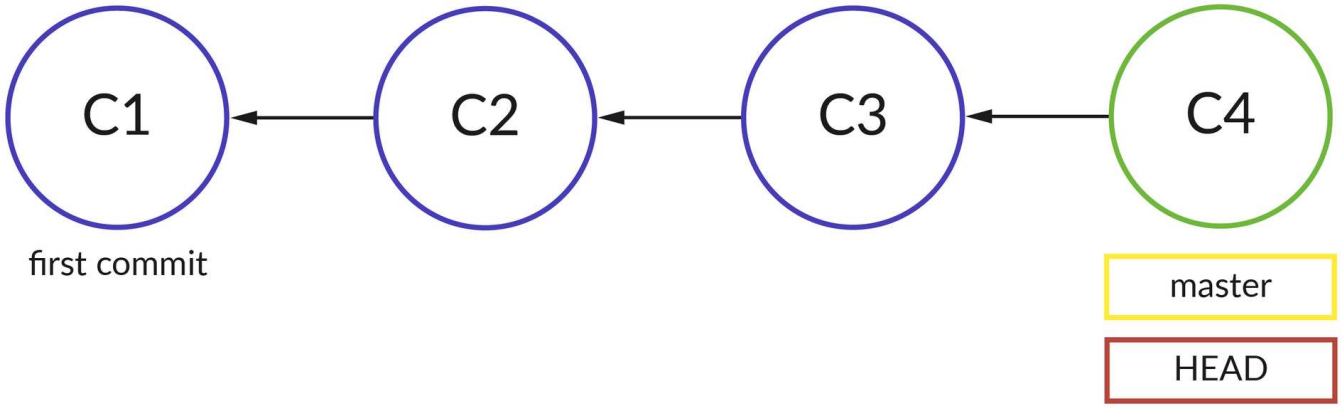
- C1 jest rodzicem C2,
- C2 jest rodzicem C3 i dzieckiem C1,
- C3 jest dzieckiem C2.

Wprowadzając zmiany do projektu, tworzymy strukturę drzewa. Poszczególne wersje projektu, w których zmiany zostały zatwierdzone, nazywamy **commitami**.

Poświęćmy kilka słów terminowi *commit*. Jest to bowiem jeden z przypadków, w których język naturalny nie nadąża za technologią. W języku polskim nie powstał na razie zwięzły, a zarazem oddający istotę rzeczy odpowiednik tego słowa. Chcąc nie chcąc, użyjemy pojęcia commit, ilekroć będziemy mieli na myśli zatwierdzoną wersję projektu.

W powyższym przykładzie mamy do czynienia z prostą liniową strukturą bez rozgałęzień.

Gałąź (*branch*) w systemie Git jest ciągiem kolejnych wersji projektu, do których wprowadziliśmy zmiany (czyli są to następujące po sobie commity).



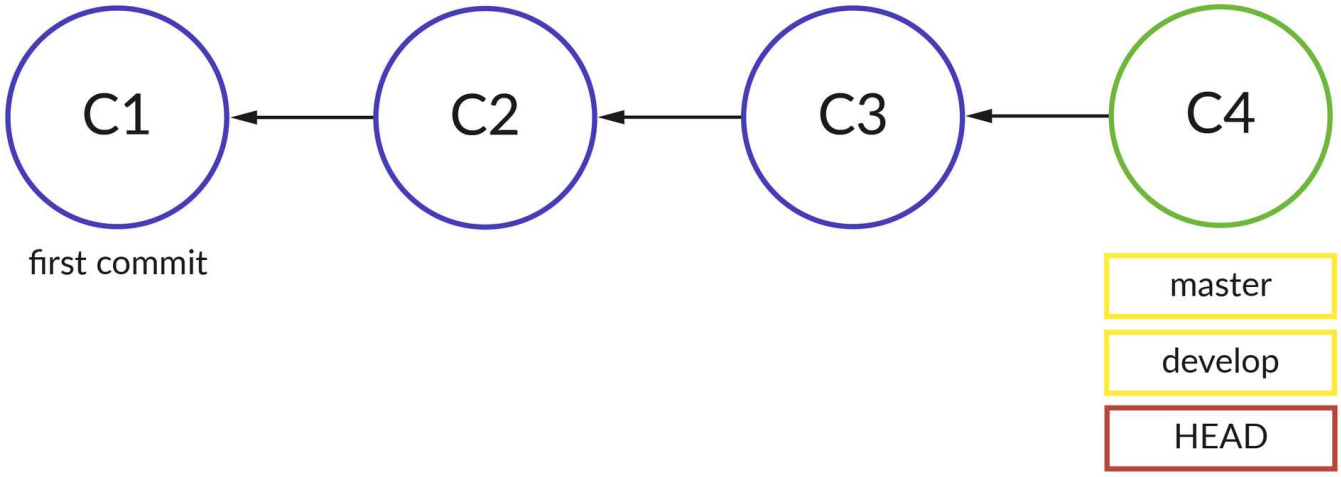
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Domyślną gałęzią w systemie Git jest `master`. Dodatkowo istnieje wskaźnik `HEAD`. Informuje on, która wersja projektu jest obecnie wczytana do katalogu roboczego – są to pliki, nad którymi aktualnie pracujesz. Wskaźnik ten automatycznie przesuwa się wraz ze zmianami wprowadzanymi do projektu (czyli wraz z powstawaniem kolejnych commitów). W naszym przypadku wskaźnik `HEAD` pokazuje commit C4 należący do gałęzi `master`.

Utworzenie nowej gałęzi sprowadza się do utworzenia nowego wskaźnika. Nie powstaje natomiast dodatkowa kopia istniejącego już projektu – jest to rozwiązanie bardzo efektywne.

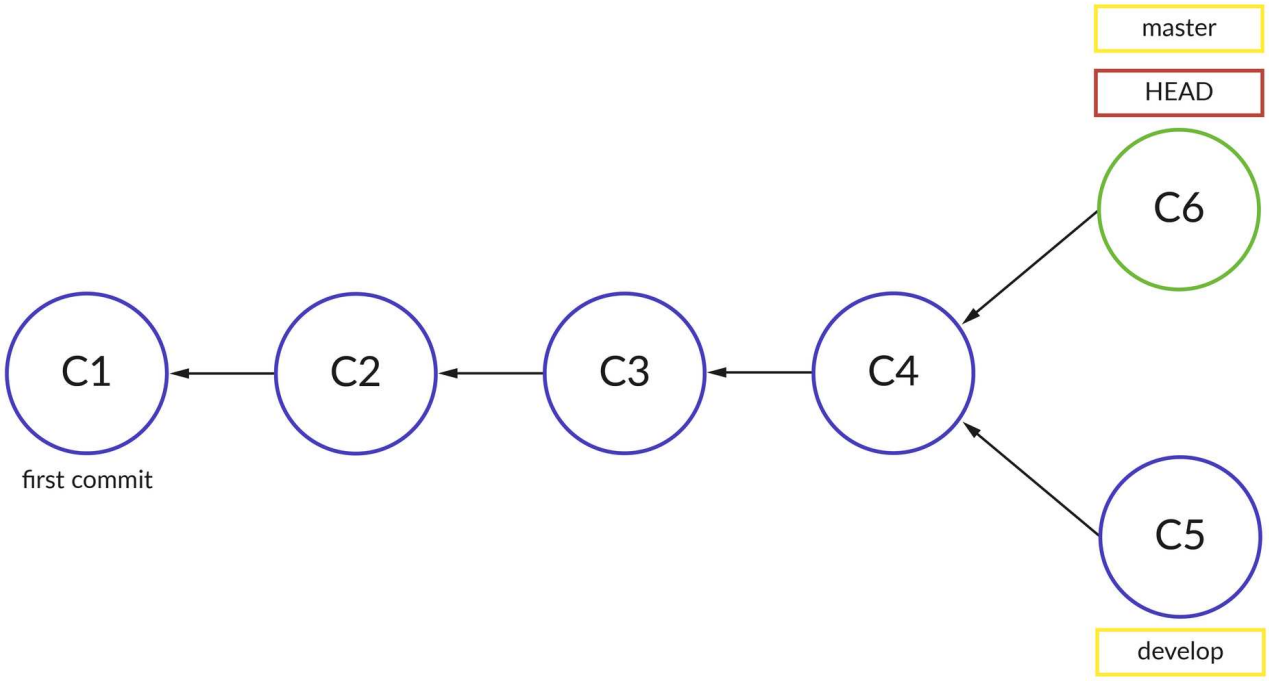
Po utworzeniu nowej gałęzi można powiedzieć, że mamy dwie niezależne wersje projektu. Zmiany wprowadzone w jednej gałęzi nie wpływają na pozostałe gałęzie. Jeśli zajdzie taka potrzeba, gałęzie można połączyć.

Częstą praktyką przy pracy z systemem Git jest rozwijanie projektu na gałęzi o nazwie `develop`. Gdy dana grupa zmian zostanie przetestowana oraz zatwierdzona, włącza się ją do głównej gałęzi.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Wprowadzając zmiany w obu wersjach projektu, zaobserwujemy powstanie rozgałęzienia. Utworzyły się dwie linie projektu, czyli odrębne gałęzie.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Do czego używane są gałęzie?

Wykorzystywanie gałęzi jest przydatne zarówno podczas pracy zespołowej, jak i wtedy, gdy samodzielnie pracujesz z repozytorium.

Istnieje kilka technik pracy z wykorzystaniem gałęzi. Obowiązujące w nich zasady ustalają członkowie zespołu. Przykładowo, w jednej gałęzi (zazwyczaj `master`) utrzymywany jest zawsze aktualny, działający projekt. Gdy chcemy wprowadzić do niego zmiany, tworzymy nową gałąź i z nią właśnie pracujemy. Gdy uznamy, że osiągnęliśmy wyznaczony cel, włączamy wprowadzone zmiany do gałęzi głównej.

W praktyce utworzenie nowej gałęzi można wykorzystać m.in. przy:

1. Dodawaniu nowych funkcji do projektu. Tworzysz nową gałąź, w której wprowadzasz nowe elementy do działającej wersji programu. W głównej gałęzi istnieje wciąż projekt w postaci niezmienionej, do której w razie konieczności można się cofnąć. Gdy skończysz pracę nad zmianami, łączysz obie gałęzie.
2. Wprowadzaniu poprawek. Jest to przykład bardzo podobny do poprzedniego. Poprawki wprowadzane są w osobnej gałęzi, a po ich zatwierdzeniu gałęzie są łączone.
3. Eksperymentach. Gdy chcesz coś przetestować, tworzysz nową gałąź i bezpiecznie sprawdzasz rezultat zmian wprowadzonych do projektu.

Operacje na gałęziach: rozgałęzianie i scalanie

Wizualizacja gałęzi w repozytorium

Aby podejrzeć historię zmian i pokazać gałęzie, wydaj polecenie w katalogu repozytorium:

```
git log --graph --full-history --all --pretty
```

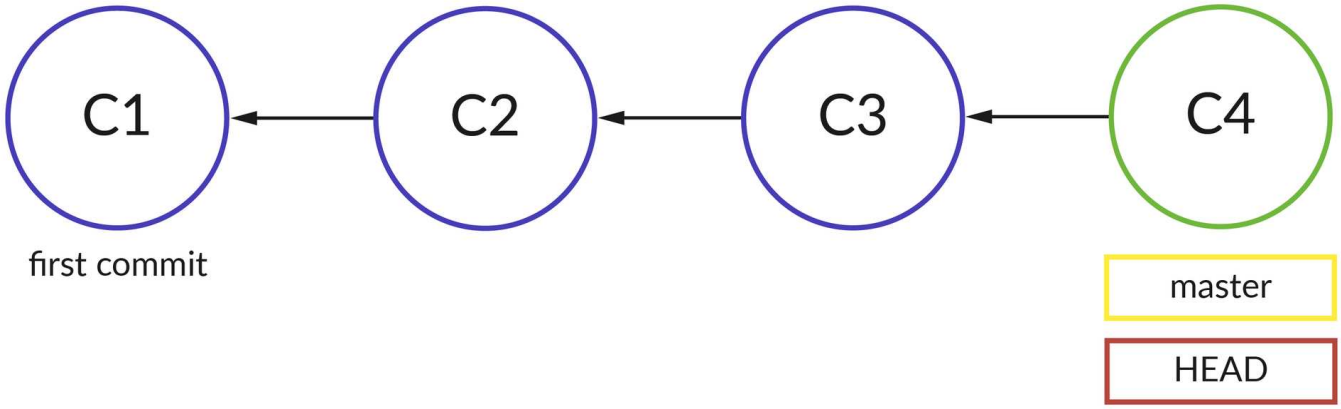
Do przeglądania historii zmian na jednej wybranej gałęzi należy użyć polecenia:

```
git log <nazwa_brancha>
```

lub graficznego narzędzia do przeglądania repozytorium (np. gitk).

Tworzenie nowej gałęzi

Załóżmy, że nasze repozytorium wygląda tak jak na poniższym rysunku. Mamy kilka commitów oraz jedną gałąź `master`, pokazywaną przez wskaźnik `HEAD`:



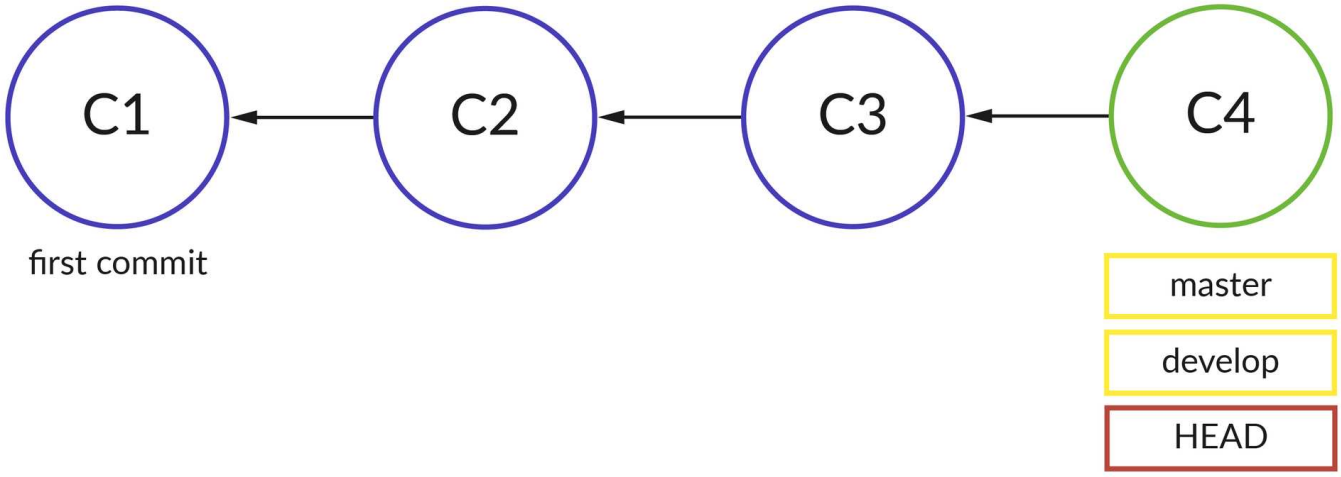
Utworzymy teraz nową gałąź i do niej właśnie przejdziemy (czyli będziemy z nią pracować). W tym celu wydajemy polecenia:

```
git branch develop  # tworzy nową gałąź o nazwie develop
git checkout develop    # zmiana gałęzi na develop
```

Zamiast tych dwóch poleceń możesz także użyć jednego:

```
git checkout -b develop
```

Oto stan repozytorium:

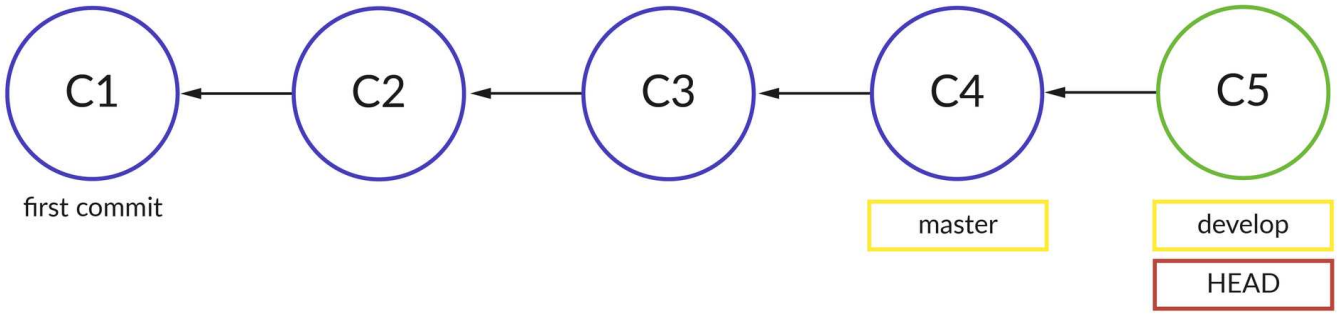


Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Wprowadź zmiany w repozytorium (np. dodaj plik), a następnie je zatwierdź:

```
git add <ścieżka_do_pliku>
git commit -m "opis wprowadzonych zmian"
```

Repozytorium wygląda następująco:



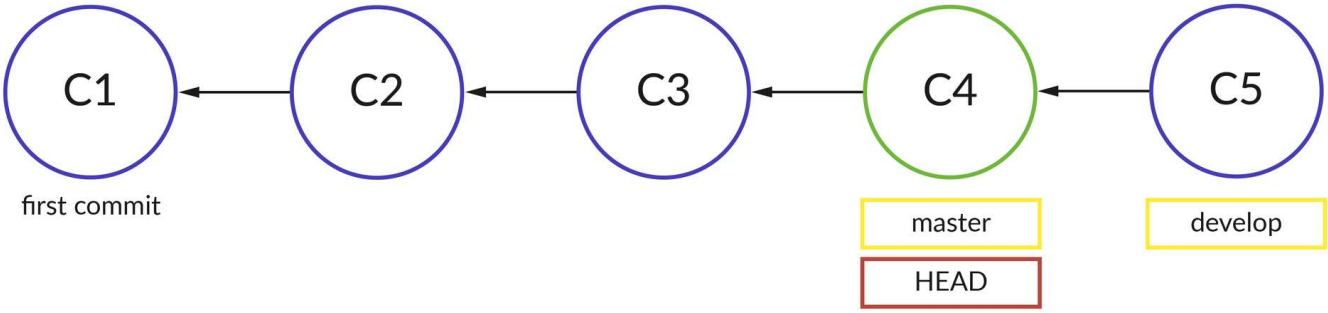
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

W celu wyświetlenia istniejących gałęzi użyjemy polecenia:

```
git branch
```

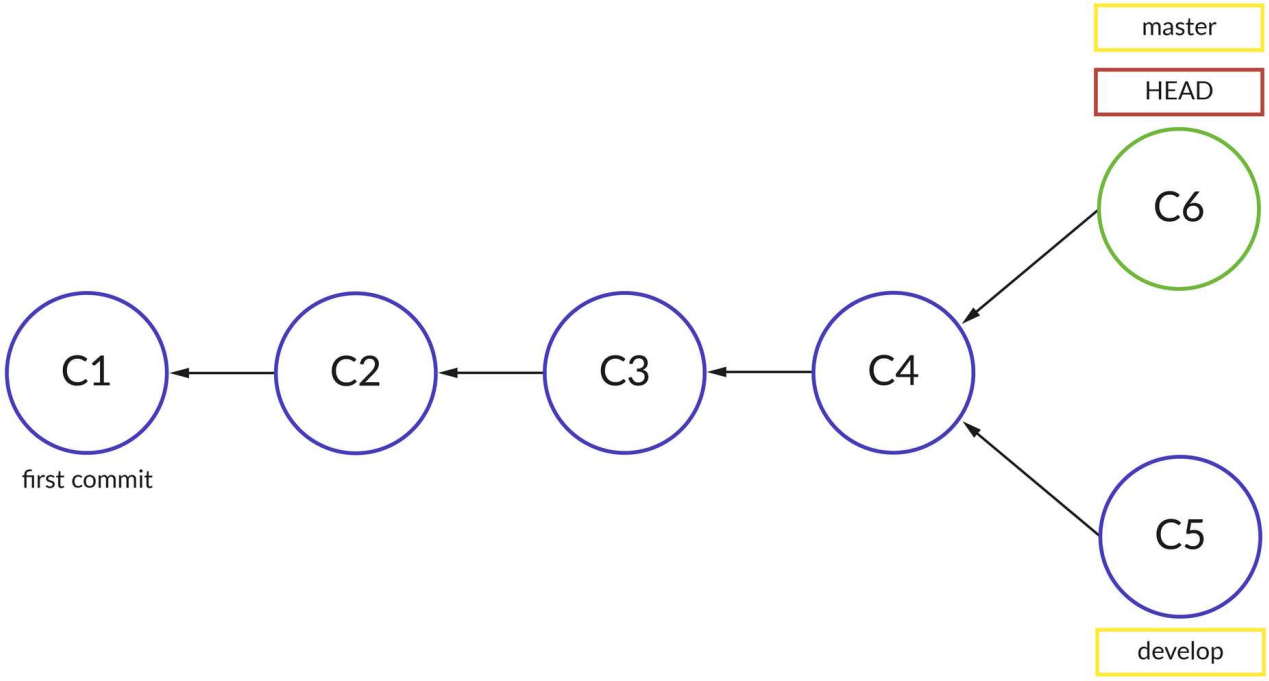
Powróćmy teraz do gałęzi `master`:

```
git checkout master
```



Teraz pracujesz z gałęzią `master`. Wskaźnik `HEAD` ustawiony jest właśnie na nią.

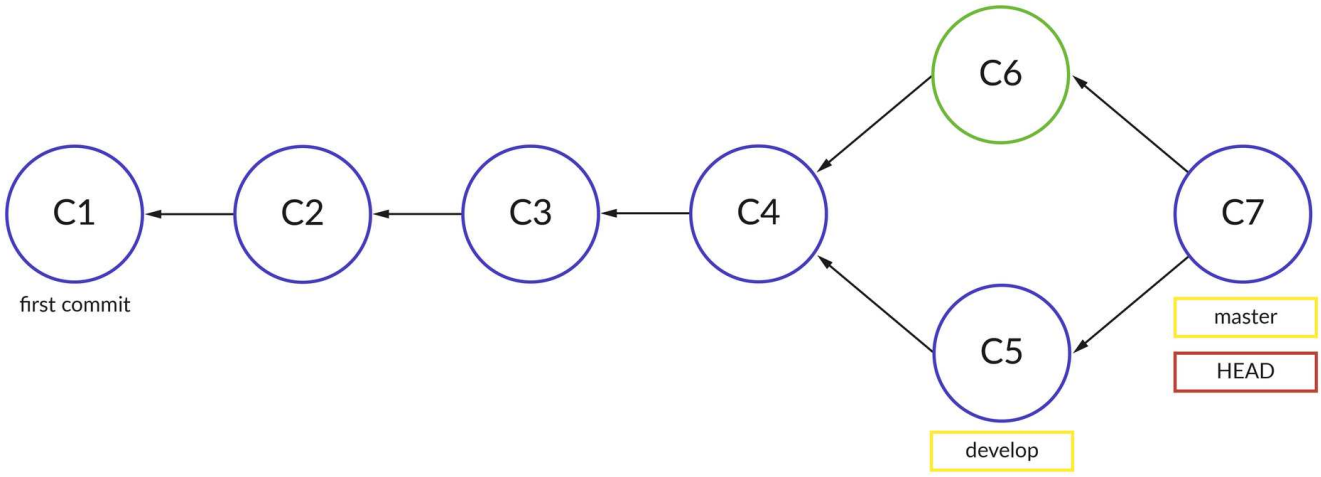
Ponownie dodaj i zatwierdź zmiany w repozytorium. Graficzna reprezentacja wprowadzonych zmian wygląda następująco:



Scalanie gałęzi

Załóżmy, że zakończyliśmy pracę w gałęzi `develop` i chcemy ją scalić z gałęzią `master` (czyli włączyć wprowadzone poprawki do gałęzi głównej). Aby to zrobić, wydajemy następujące polecenia:

```
git checkout master     # przejście do gałęzi master
git merge develop   # scalenie gałęzi develop z gałęzią master
```



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

W wyniku scalania został utworzony nowy commit. Teraz gałąź `master` będzie zawierała zmiany, jakie zostały dokonane w gałęzi `develop`.

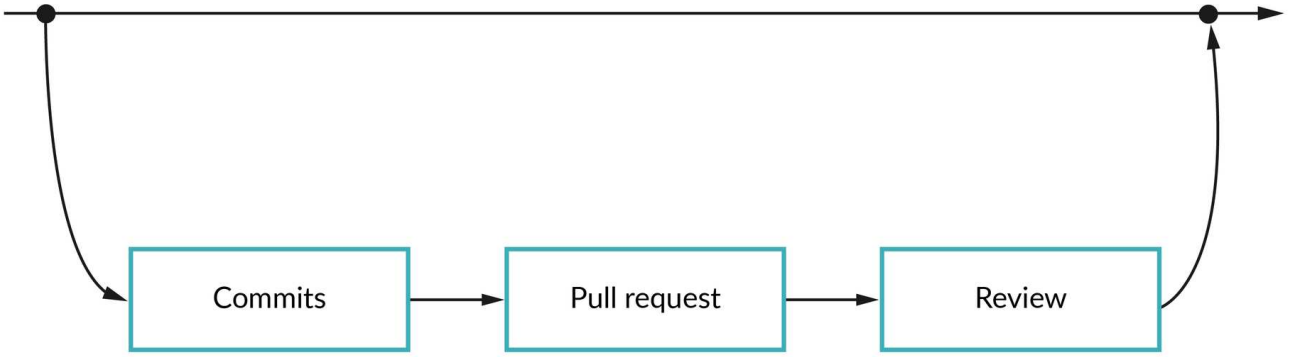
Pull request – co to takiego?

Pull request to prośba o zaakceptowanie zmian wprowadzonych do projektu. Mechanizm ten przydaje się podczas pracy grupowej. Pozwala on:

- poinformować osoby zaangażowane w projekt o zmianach, jakich dokonano w wydzielonej gałęzi,
- przedyskutować wprowadzone zmiany,
- odrzucić lub włączyć zmiany do innej gałęzi projektu.

Pull request jest często stosowany podczas pracy przy dużych projektach typu *open source*. Tylko uprawnione osoby mogą dokonywać zmian w repozytorium. Jeśli pobraliśmy takie repozytorium na dysk komputera i wprowadziliśmy jakieś poprawki do kodu, a następnie chcemy, by zostały one włączone do oficjalnego projektu (i były dostępne dla wszystkich), musimy wystawić żądanie *pull request*.

Po tym następuje *Review*.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Opis projektu w języku Markdown

Przeglądając repozytoria w serwisie GitHub, można zauważyć, że towarzyszy im estetyczny, sformatowany opis. Może on zawierać spis treści, różnej wielkości nagłówki, odnośniki, przykłady użycia projektu czy wycinki z dokumentacji.

W serwisie GitHub dokument taki zapisany jest w pliku `README.md`, utworzonym z wykorzystaniem języka Markdown.

Markdown jest językiem znaczników, służącym do formatowania tekstu. Plik zapisany w formacie `.md` przekształcany jest do postaci w języku HTML.

Podstawowa składnia języka to:

- Nagłówki: znacznik #

```
# H1

## H2

### H3

#### H4
```

- Emfaza: znacznik * lub _

```
*kursywa*

**pogrubienie**

***pogrubiona kursywa***
```

- Listy: znacznik * lub + lub -

```
- poz1
- poz2
- poz3
```

- Listy numerowane: 1, 2, 3 itd.

```
1. poz1
2. poz2
```

- Odnośniki: znacznik [ ] i ( )

```
To jest [Odnośnik](http://epodreczniki.pl/).
```

- Spis treści – z użyciem odnośników do nagłówków: [ ] i ( )

```
[Informacje](#informacje)
```

- Fragment kodu otwieramy: ```` ```nazwa_języka ```` i zamykamy: ```` ``` ````

````
```python
print("Hello World!")
```
````

Plik `.gitignore` – co zawiera i do czego służy?

System kontroli wersji śledzi wszystkie zmiany w plikach znajdujących się w katalogu roboczym repozytorium. Często zdarza się jednak, że należy pominąć niektóre pliki lub całe katalogi.

System Git używa specjalnego pliku o nazwie `.gitignore`. Można w nim zapisać, które katalogi lub pliki nie powinny być monitorowane pod kątem wprowadzanych w nich zmian. Oznacza to, że jeżeli dany plik nie został nigdy „zacommitowany", to nie będzie on przechowywany w repozytorium.

Jeśli nie widzisz w swoim repozytorium pliku `.gitignore`, możesz go utworzyć i zmieniać jego zawartość, korzystając z edytora tekstu.

Oto przykładowa zawartość pliku `.gitignore`:

```
/Debug/*.o
data.db
```

W przypadku tak przygotowanego pliku, `.gitignore` system Git nie będzie śledził zmian w plikach z rozszerzeniem `.o` znajdujących się w katalogu `Debug` oraz w pliku `data.db`.

Słownik

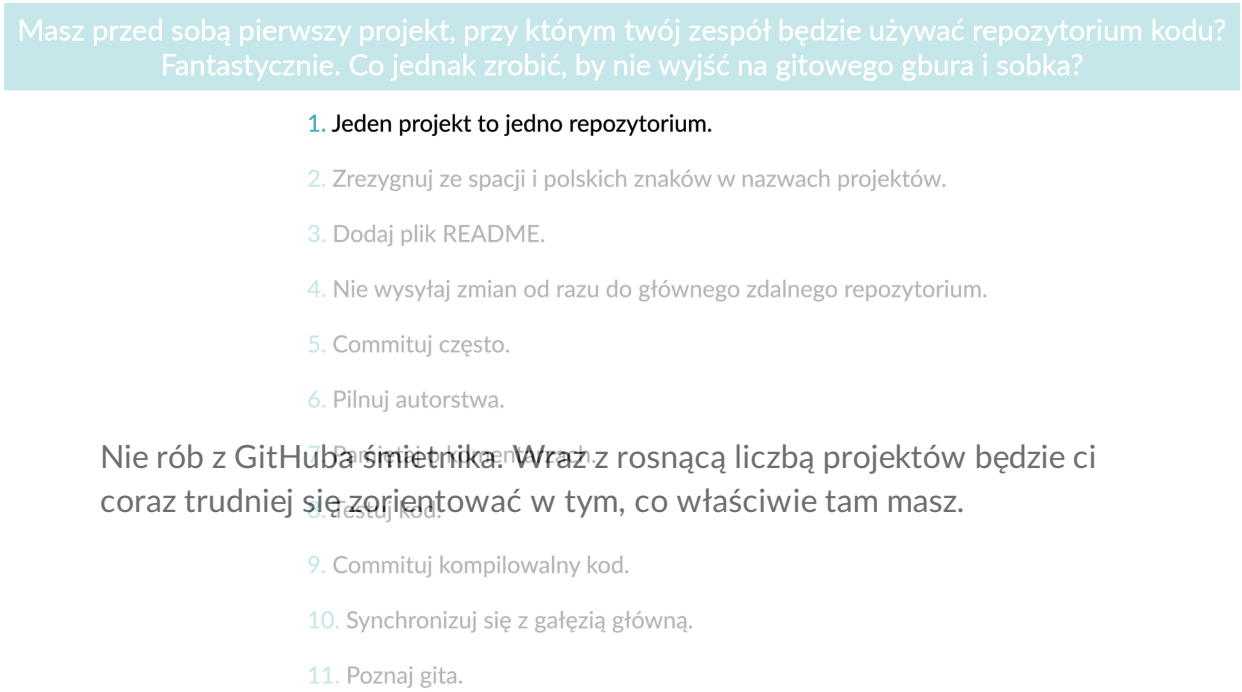
> **`HEAD`**
>
> wskaźnik do zatwierdzonej wersji projektu (commitu), z którą obecnie pracujesz
> ***review***
>
> przegląd i ocena implementacji danej funkcji przez członka zespołu; przegląd taki ma za zadanie wykrycie i poprawienie ewentualnych błędów; wpływa to pozytywnie na jakość tworzonego projektu
> **suma kontrolna**
>
> identyfikator służący do sprawdzenia poprawności danych (zbadania, czy są one wolne od błędów i nie zostały zmanipulowane); zazwyczaj jest to ciąg składający się z cyfr i liter; ciąg wyliczany jest według specjalnego algorytmu, od którego zależy m.in. długość sumy kontrolnej i stopień ochrony

Infografika

> **Polecenie 1**
>
> Zapoznaj się z infografiką. Następnie porozmawiaj z koleżanką lub kolegą z ławki o najważniejszych zasadach pracy z repozytorium Git. Zastanówcie się, jakie problemy może wywołać ich nieprzestrzeganie.

Jak nie być git gburem?

1. Jeden projekt to jedno repozytorium.



2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

3. Dodaj plik README.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Pamiętaj, że coraz częściej pracujemy w środowiskach międzynarodowych. Do dobrych praktyk należy stosowanie takich nazw, które będą jasne dla większej liczby użytkowników.

Umieść w nim krótki opis projektu, wykorzystane technologie, języki, biblioteki etc.

4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Choć komenda `git push master` może być kusząca (wszak twój kod jest na pewno bezbłędny!), dobrą praktyką jest to, żeby zmiany wysyłać do osobnej gałęzi, a dopiero po ich weryfikacji scalać je ze zdalnym repozytorium.
Dzięki temu osoby, które razem z tobą pracują nad danym projektem mogą łatwo przejrzeć twoje zmiany, dodać swoje sugestie czy otworzyć dyskusję.

5. Commituj często.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Nie bój się tego, że twoje zmiany przesadnie obciążą pamięć. Git to narzędzie, które zapisuje tylko zmiany, nie tworzy zupełnie nowych plików.

6. Pilnuj autorstwa.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Za pomocą komendy `git blame` sprawdzisz, kto jest autorem zmian. Dzięki temu wiesz, z kim się kontaktować, jeśli chcesz coś doprecyzować czy zwrócić na coś uwagę. Z tego względu pamiętaj o tym, by poprawnie skonfigurować przynajmniej swoje imię oraz adres e-mail.

7. Pamiętaj o komentarzach.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Wraz z commitem wysyłasz wiadomość. Choć w danym momencie komunikat „poprawka do błędu” czy „fixing bug” może być całkowicie jasny, już za tydzień możesz mieć problem, by przypomnieć sobie cokolwiek na jego temat.

8. Testuj kod.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pamiętaj o komentarzach.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Zanim zdecydujesz się wysłać commit, przetestuj swój kod.

9. Commituj kompilowalny kod.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pisz testy do swojego kodu.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

10. Synchronizuj się z gałęzią główną.

Nie musisz commitować dopiero wtedy, kiedy skończysz całą zaplanowaną pracę. Wprost przeciwnie! Wystarczy, jeśli będzie często wysyłać małe zmiany, ale takie, które już działają.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Fantastycznie. Co jednak zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.
2. Zrezygnuj ze spacji i polskich znaków w nazwach projektów.
3. Dodaj plik README.
4. Nie wysyłaj zmian od razu do głównego zdalnego repozytorium.
5. Commituj często.
6. Pilnuj autorstwa.
7. Pisz testy do swojego kodu.
8. Testuj kod.
9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Nie ma nic bardziej frustrującego, niż poprawić błąd i zorientować się, że zostało to już zrobione dużo wcześniej. Dlatego tak ważne jest, by pamiętać o częstej synchronizacji.
Dzięki temu będziemy również na bieżąco rozwiązywać konflikty – dlatego nie stroń od komendy `rebase`.

11. Poznaj gita.

Masz przed sobą pierwszy projekt, przy którym twój zespół będzie używać repozytorium kodu? Zastanów się, co powinieneś zrobić, by nie wyjść na gitowego gbura i sobka?

1. Jeden projekt to jedno repozytorium.

Polecenie 2

Zastanów się, jakie zalety i wady ma korzystanie z repozytorium Git w trakcie zespołowej pracy nad programem.

Najważniejszą być może radą jest to, by nie bać się poznawać nowych funkcji i możliwości. W sieci znajdziesz wiele kompletnych poradników, które bezboleśnie przeprowadzą cię przez proces poznawania działania repozytorium kodu.

9. Commituj kompilowalny kod.
10. Synchronizuj się z gałęzią główną.
11. Poznaj gita.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Połącz każde polecenie z właściwą definicją.

`git checkout -b nazwa_gałęzi`	scalenie gałęzi o podanej nazwie z aktualną
`git branch`	utworzenie nowej gałęzi i przełączenie się na nią
`git checkout nazwa_gałęzi`	wyświetlenie istniejących gałęzi
`git merge nazwa_gałęzi`	utworzenie nowej gałęzi o podanej nazwie
`git branch nazwa_gałęzi`	przełączenie się na daną gałąź

Ćwiczenie 2

Zaznacz prawidłową odpowiedź.

Czy w historii zmian projektu mogą pojawić się dwa commity o tych samych identyfikatorach (sumach kontrolnych)?

- ◯ nie
- ◯ tak

Ćwiczenie 3

Dopasuj znaczniki języka Markdown do właściwych wyników ich zastosowania.

znacznik #	nagłówek
znacznik * lub + lub −	lista wypunktowana
znacznik ** lub _ _	lista numerowana
znacznik [] i ()	odnośnik
znacznik 1, 2, 3 itp.	kursywa
znacznik * lub _	pogrubienie

Ćwiczenie 4

Zaznacz wszystkie właściwe dokończenia zdania.

Wskaźnik HEAD:

- [ ] służy do ochrony gałęzi przed połączeniem jej z inną.
- [ ] jest zawsze stały i wskazuje gałąź `master`.
- [ ] zmienia swoje położenie podczas zmiany gałęzi, po wydaniu polecenia `git checkout`.
- [ ] wskazuje aktualną zmianę (commit).
- [ ] jest stały i wskazuje początek drzewa.

Ćwiczenie 5

Zaznacz prawidłowy opis pliku `.gitignore`.

- ◯ plik ignorowany przez system Git
- ◯ plik dodatkowy, traktowany jako kopia zapasowa
- ◯ plik, dzięki któremu użytkownik określa, w jakich plikach nie są śledzone zmiany
- ◯ plik pomocniczy systemu kontroli wersji, którego użytkownik nie może edytować

Ćwiczenie 6

Zaznacz prawidłową odpowiedź.
Jak w serwisie GitHub nazywa się główny plik z opisem projektu w języku Markdown?

- ◯ `MARKDOWN.md`
- ◯ `README.md`
- ◯ `DESCRIPTION.md`
- ◯ `INFO.md`

Ćwiczenie 7

Wskaż wszystkie polecenia, za pomocą których można utworzyć nową gałąź.

- ☐ `git branch`
- ☐ `git checkout -b nazwa`
- ☐ `git branch nazwa`
- ☐ `git commit`

Ćwiczenie 8

Wskaż prawidłową odpowiedź.
Czy *`pull request`* zawsze jest akceptowany?

- ◯ nie
- ◯ tak

Ćwiczenie 9

Wskaż prawidłową odpowiedź.
Czy *commit* może mieć dwóch rodziców?

- ◯ nie
- ◯ tak (w wyniku łączenia dwóch gałęzi)

Ćwiczenie 10

Wskaż prawidłową odpowiedź.
Czy *commit* może nie mieć rodzica?

- ◯ nie
- ◯ tak (jest to pierwszy commit)

Ćwiczenie 11

Utwórz plik `.gitignore` w którym następujące pliki/elementy nie powinny być śledzone przez system Git:

- pliki z rozszerzeniem `*.exe`, `*.app` oraz `*.gif`
- plik o nazwie `secret.docx`,
- katalog o nazwie `Tymczasowe`.

Ćwiczenie 12

Przygotuj plik `README.md` z opisem projektu. Przykładowy rezultat możesz znaleźć poniżej.

```
# Projekt

Program napisany w ***języku C++***

## Zawartość

* [Pobranie kodu źródłowego](#pobieranie)
* [Kompilacja](#kompilacja)
* [Informacje](#informacje)

## Pobieranie

W celu pobrania źródeł programu w terminalu wykonaj polecenie:

```shell
git clone
```

Kompilacja

Aby skompilować program wykonaj:

```shell
g++ main.cpp -o prog
```

Informacje

- licencja - [MIT](https://epodreczniki.pl)
- Autor: Jan Nowak
```

Ćwiczenie 13

Pracujesz w zespole, który zajmuje się wprowadzaniem poprawek w grze. Na jednym z forów gracze zgłosili pewien bug – jedna z postaci jest źle oskryptowana. Zespół obsługi klienta oznaczył go kodem #bug131120. Przyjęliście, że gałęzie tworzone dla konkretnych błędów noszą ich nazwy.

Wykonaj kolejne czynności w repozytorium:

- Sprawdź, czy ktoś utworzył już gałąź dla danego błędu.
- Załóżmy, że gałąź nie istnieje i musisz ją dopiero utworzyć. Utwórz gałąź dla wskazanego błędu i się na nią przełącz.
- Coś na chwilę oderwało cię od pracy. Być może w tym czasie ktoś ze współpracowników zajął się tym błędem. Sprawdź, czy dla utworzonej gałęzi wprowadzono jakieś zmiany.
- Wprowadź swój plik z poprawką do systemu kontroli wersji (`fix_bug131120.txt`),
- Udało ci się poprawić skrypt, postać poprawnie rozpoznaje imię gracza. Dodaj `commit` z opisem `fix charname`.
- Przełącz się na gałąź `master`,
- Połącz nowo utworzoną gałąź z gałęzią `master`.

Dla nauczyciela

Autor: Marcin Jeliński
Przedmiot: Informatyka

Temat: Praca z repozytorium Git

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

IV. Rozwijanie kompetencji społecznych, takich jak: komunikacja i współpraca w grupie, w tym w środowiskach wirtualnych, udział w projektach zespołowych oraz zarządzanie projektami.

Treści nauczania – wymagania szczegółowe

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

2) do realizacji rozwiązań problemów prawidłowo dobiera środowiska informatyczne, aplikacje oraz zasoby, wykorzystuje również elementy robotyki;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

5) współtworzy otwarte zasoby i aktywności oraz umieszcza je w sieci, m.in. na platformie do e-nauczania.

IV. Rozwijanie kompetencji społecznych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) przy realizacji zespołowego projektu programistycznego posługuje się środowiskiem przeznaczonym do współpracy i realizacji projektów zespołowych, w tym środowiskiem w chmurze; współtworzy zasoby udostępniane na platformach do e-nauczania;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Wyjaśnisz, czym są gałęzie w systemie Git.
- Scharakteryzujesz podstawowe operacje na gałęziach: tworzenie i łączenie.
- Przeanalizujesz, do czego służy polecenie `pull request`.
- Opiszesz projekt za pomocą języka Markdown.
- Wyjaśnisz, do czego służy plik `.gitignore` w repozytorium.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Praca z repozytorium Git". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel inicjuje rozmowę wprowadzającą w temat lekcji. Przedstawia cele zajęć oraz kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. Uczniowie przypominają sobie najważniejsze komendy, używane w systemie kontroli wersji GitHub.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna", uczniowie zapoznają się z prezentacją. Następnie w parach dyskutują o najważniejszych zasadach pracy z repozytorium Git. Zastanawiają się, jakie problemy może wywołać ich nieprzestrzeganie. W kolejnym kroku indywidualnie realizują polecenie 2. Chętne osoby dzielą się swoimi spostrzeżeniami na temat zalet i wad zespołowej pracy z repozytorium Git.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1-10, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Uczniowie łączą się w grupy i wspólnie tworzą repozytorium projektu, nad którym będą pracować. Przykładowe projekty: strona WWW (wcześniej wymagane jest opanowanie przez uczniów umiejętności pisania prostych stron), gra (wcześniej uczniowie powinni opanować podstawy np. UNITY), prosty chatbot.
5. Uczniowie rozdzielają między sobą role w projekcie. Przygotowują repozytorium projektu.
6. Wymogi dotyczące projektu:
    - każdy z uczniów powinien wykonać przynajmniej trzy commity;
    - każdy commit musi mieć właściwe komentarz;
    - wszyscy uczniowie na GitHubie podpisują swoje commity inicjałami;
    - każda osoba dodaje przynajmniej jeden plik do repozytorium;
    - jeśli uczniowie tworzą np. stronę, powinna ona składać się przynajmniej ze strony głównej, menu, galerii oraz strony z kontaktem do autorów

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

Praca domowa:

1. Uczniowie wykonują ćwiczenie 11-13 z sekcji „Sprawdź się”.
2. Uczniowie realizują projekt, który rozpoczęli na zajęciach. Dodatkowo mają wprowadzić zmiany w przynajmniej jednym pliku należącym do innej osoby z zespołu, a następnie scalić go z plikiem głównym i rozwiązać ew. konflikty.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla systemu kontroli wersji Git.

Wskazówki metodyczne:

- E-materiał można wykorzystać równolegle do pracy z serią dotyczącą Unity, Unreal Engine, obróbki audio, video czy tworzenia prostej gry (węża).