Zaawansowany Programista JS

Dzień 1

Plan

- Podstawy Terminala
  - komendy `ls`, `cd`, `pwd`, `mkdir`, `touch`, `vim`
- Podstawy GIT
  - Wprowadzenie do GIT i Github
  - Tworzenie Repozytoriów na Github
  - Generowanie klucza SSH
  - Wrzucanie plików do Github za pomocą `git add`, `git commit`, `git push`
  - Praca z repozytoriami za pomocą komend `git status`, `git log`, `git reset --hard`
  - Synchronizacja repozytorium z Github za pomocą Download zip, `git clone`, `git pull`
  - Zakładka do GIT w Visual Studio Code
- Prework JS
  - Sprawdzenie aktualnej wiedzy z JS

Podstawy Terminala

Terminal to takie miejsce na komputerze, gdzie możemy wpisywać specjalne komendy, które wykonują różne zadania. Dzięki terminalowi możemy na przykład uruchamiać programy, przenosić pliki, instalować nowe oprogramowanie i wiele więcej.

Często programiści korzystają z terminala, ponieważ umożliwia on szybkie i efektywne wykonywanie różnych zadań. Na przykład, zamiast klikać w różne ikony na ekranie, możemy w terminalu jedną komendą skopiować całą bazę danych, wysłać ją do innego serwera, i jeszcze otworzyć plik tekstowy do edycji.

Komenda `ls`

Komenda `ls` służy do wyświetlania listy plików i katalogów znajdujących się w bieżącym katalogu roboczym.

Kiedy pracujemy w terminalu, często musimy wiedzieć, jakie pliki i katalogi znajdują się w danej lokalizacji, aby móc nimi zarządzać. Na przykład, jeśli chcemy otworzyć plik tekstowy, musimy wiedzieć, jak nazywa się ten plik i gdzie jest zapisany.

Używając komendy "ls", możemy szybko zobaczyć, jakie pliki i katalogi znajdują się w bieżącym katalogu roboczym i jakie są ich nazwy.

Komenda `ls` ma też wiele opcji, które pozwalają na dostosowanie sposobu wyświetlania listy plików i katalogów, na przykład, można wyświetlić informacje o rozmiarze plików, dacie modyfikacji, czy też kolorować nazwy plików i katalogów w zależności od ich typu.

Komenda `cd`

Komenda `"cd` służy do zmiany aktualnego katalogu roboczego.

Kiedy pracujemy w terminalu, często musimy przemieszczać się między różnymi katalogami, aby wykonywać różne zadania. Na przykład, jeśli pracujemy z plikami znajdującymi się w folderze "Projekty", musimy najpierw przejść do tego folderu, aby móc otworzyć i edytować pliki.

Używając komendy "cd", możemy szybko zmienić katalog roboczy na inny. Na przykład, wpisując komendę `cd Projekty`, przechodzimy do katalogu "Projekty", a wpisując `cd ..` przechodzimy do katalogu nadrzędnego.

Komenda "cd" jest bardzo przydatna w pracy z terminalami, ponieważ pozwala na szybkie przemieszczanie się między różnymi katalogami i wykonywanie operacji na plikach znajdujących się w tych katalogach.

Komenda `pwd`

Komenda `pwd` to skrót od "print working directory" i służy do wyświetlenia pełnej ścieżki do aktualnego katalogu roboczego.

Kiedy pracujemy w terminalu, często musimy wiedzieć, w jakim katalogu roboczym się znajdujemy, aby móc wykonywać różne operacje na plikach w tym katalogu. Na przykład, gdy chcemy skopiować plik do innego katalogu, musimy najpierw upewnić się, że jesteśmy w odpowiednim katalogu roboczym.

Używając komendy "pwd", możemy szybko sprawdzić, w jakim katalogu roboczym aktualnie się znajdujemy. Komenda "pwd" wyświetla pełną ścieżkę do bieżącego katalogu roboczego, czyli listę katalogów, które trzeba przejść, aby dostać się do bieżącego katalogu.

Komenda "pwd" jest przydatna w pracy z terminalami, ponieważ pozwala na szybkie sprawdzenie, w jakim katalogu roboczym się znajdujemy i w jakiej lokalizacji pracujemy z plikami i katalogami.

```
$ pwd
$ /Users/jankowalski/Desktop/kurs
```

Komenda `mkdir`

Komenda `mkdir` to skrót od "make directory" i służy do tworzenia nowych katalogów w bieżącym katalogu roboczym.

Dlaczego to ważne? Kiedy pracujemy w terminalu, często musimy tworzyć nowe katalogi, aby organizować nasze pliki i przechowywać je w porządku. Na przykład, gdy pracujemy nad projektem, możemy utworzyć katalog o nazwie "Projekt_X" i przechowywać w nim wszystkie pliki związane z tym projektem.

Używając komendy `mkdir`, możemy szybko utworzyć nowy katalog w bieżącym katalogu roboczym. Wystarczy wpisać `mkdir nazwa_katalogu` i nowy katalog zostanie utworzony.

Komenda "mkdir" może tworzyć także wiele katalogów jednocześnie. Na przykład, wpisanie `mkdir katalog1 katalog2 katalog3` utworzy trzy nowe katalogi o nazwach "katalog1", "katalog2" i "katalog3".

Komenda `mkdir` jest przydatna w pracy z terminalami, ponieważ pozwala na szybkie tworzenie nowych katalogów i organizowanie naszych plików w porządku.

Komenda `touch`

Komenda `touch` służy do tworzenia nowych plików w bieżącym katalogu roboczym lub do aktualizacji daty modyfikacji istniejącego pliku.

Kiedy pracujemy z plikami w terminalu, czasami musimy utworzyć nowy plik, aby zacząć w nim pisać lub zapisać dane. Może to być przydatne, gdy piszemy skrypty, dokumenty tekstowe lub konfiguracyjne pliki.

Używając komendy "touch", możemy szybko utworzyć nowy pusty plik o określonej nazwie w bieżącym katalogu roboczym. Na przykład, wpisanie `touch nowy_plik.txt` utworzy nowy pusty plik o nazwie "nowy_plik.txt". **Jeśli plik już istnieje, użycie komendy "touch" aktualizuje datę modyfikacji pliku do bieżącego czasu.**

Komenda "touch" jest przydatna w pracy z terminalami, ponieważ pozwala na szybkie tworzenie nowych plików i aktualizowanie daty modyfikacji istniejących plików. Może być także przydatna, gdy tworzymy skrypty, które wymagają, aby określony plik istniał w celu prawidłowego działania.

Komenda `vim`

Komenda "vim" jest edytorem tekstu dla terminala. Pozwala na tworzenie, edytowanie i zapisywanie plików tekstowych bez konieczności opuszczania terminala.

Edycja tekstu jest częścią codziennej pracy programisty. Często trzeba wprowadzać zmiany w plikach konfiguracyjnych lub skryptach, które działają na serwerze. W tym przypadku, używanie GUI lub aplikacji z graficznym interfejsem jest niemożliwe lub bardzo utrudnione. W takim przypadku, edytory tekstowe z interfejsem konsolowym, takie jak "vim", są bardzo przydatne.

Komenda "vim" pozwala na tworzenie nowych plików lub edycję istniejących plików tekstowych w terminalu. Jest to bardzo przydatne, gdy pracujemy zdalnie i mamy tylko dostęp do terminala, lub gdy potrzebujemy szybko wprowadzić zmiany w pliku.

Edytor "vim" oferuje wiele zaawansowanych funkcji, takich jak podświetlanie składni, automatyczne uzupełnianie, wcięcia i nawigację. Dzięki temu, "vim" jest narzędziem, które pozwala na bardzo wydajną i dokładną edycję plików tekstowych.

- Przemieszczanie się po pliku: Strzałki klawiatury lub klawisze h, j, k, l: poruszanie się w górę, w dół, w lewo i w prawo. G: przejście na koniec pliku. 1G: przejście na początek pliku. :[numer linii]: przejście do określonej linii w pliku.
- Tryb edycji: i: wejście w tryb edycji w miejscu, w którym znajduje się kursor. a: wejście w tryb edycji na następnej pozycji po kursorze. o: wstawianie nowej linii poniżej kursora.

- Zapisywanie i zamykanie pliku:

:w: zapisanie zmian w pliku. :w [nazwa_pliku]: zapisanie zmian w nowym pliku o podanej nazwie. :q: zamknięcie pliku. :q!: zamknięcie pliku bez zapisywania zmian.

- Usuwanie tekstu:

x: usunięcie pojedynczego znaku. dw: usunięcie jednego słowa. dd: usunięcie całej linii. D: usunięcie tekstu od kursora do końca linii.

- Cofanie zmian:

u: cofnięcie ostatniej zmiany. Ctrl+r: przywrócenie ostatnio cofniętej zmiany.

Podstawy GIT

Wprowadzenie do GIT i Github

Git to system kontroli wersji, który służy do śledzenia i zarządzania zmianami w kodzie źródłowym projektów informatycznych. Został stworzony przez Linusa Torvaldsa w celu ułatwienia pracy programistów przy tworzeniu kodu źródłowego jądra systemu operacyjnego Linux.

Git pozwala na zapisywanie historii zmian w kodzie źródłowym projektu i łatwe porównywanie różnych wersji plików. Programiści mogą pracować na tym samym kodzie równocześnie, a Git pozwala na śledzenie wszystkich wprowadzonych zmian, co ułatwia integrację kodu z wielu źródeł.

Git umożliwia również tworzenie i przechowywanie w repozytorium kodu źródłowego w różnych gałęziach (branchach). Jest to przydatne, gdy programiści pracują nad różnymi funkcjonalnościami lub wersjami aplikacji, a ich kod musi być równocześnie przechowywany w repozytorium.

Git pozwala także na tworzenie kopii zapasowych repozytorium (clone), co ułatwia przenoszenie projektów między różnymi serwerami i środowiskami.

Git jest szeroko stosowany w branży IT i jest podstawowym narzędziem w zarządzaniu kodem źródłowym projektów. Dzięki niemu programiści mogą pracować skuteczniej i efektywniej, a zmiany w kodzie są bezpiecznie przechowywane i łatwo śledzone.

GitHub z kolei to platforma internetowa, która umożliwia przechowywanie projektów opartych na Git oraz udostępnianie ich publicznie lub prywatnie. Jest to usługa, która oferuje repozytoria Git w chmurze i dodatkowe narzędzia do pracy w grupie. Programiści mogą udostępniać swoje projekty na GitHub, aby inni programiści mogli je zobaczyć, współpracować nad nimi lub użyć ich w swoich własnych projektach.

Oprócz przechowywania projektów, GitHub oferuje także narzędzia do przeglądania kodu źródłowego, tworzenia zgłoszeń błędów (issues) oraz zarządzania projektem. Wszystkie te funkcjonalności ułatwiają pracę z repozytoriami Git, umożliwiają lepszą współpracę w zespole oraz przyczyniają się do lepszej jakości kodu źródłowego.

Tworzenie Repozytoriów na Github

Aby stworzyć nowe repozytorium na GitHubie, wykonaj następujące kroki:

1. Zaloguj się do swojego konta na GitHubie i przejdź do strony głównej.

2. Kliknij przycisk "New" na górnym pasku nawigacyjnym.
3. Wybierz typ repozytorium, które chcesz utworzyć - publiczne lub prywatne. Wybierz także opcję inicjalizacji pliku README, jeśli chcesz utworzyć plik README w repozytorium.
4. Wprowadź nazwę repozytorium oraz krótki opis. Możesz także wybrać licencję, jeśli chcesz, aby Twój kod był dostępny pod określoną licencją.
5. Wybierz opcję "Initialize this repository with a README", jeśli chcesz utworzyć plik README, który będzie zawierał opis Twojego projektu.
6. Kliknij przycisk "Create repository".
7. Twoje nowe repozytorium zostanie utworzone i będzie gotowe do użycia. Możesz teraz dodać pliki do repozytorium, zarządzać projektem, a także udostępniać repozytorium innym użytkownikom.

Następnym krokiem jest połączenie repozytorium na Githubie z folderem na dysku.

Generowanie klucza SSH

Aby połączyć GitHub z Twoim komputerem za pomocą klucza SSH, musisz najpierw wygenerować parę kluczy SSH na swoim komputerze, a następnie dodać klucz publiczny do Twojego konta GitHub. Możesz to zrobić, wykonując następujące kroki:

1. Otwórz terminal lub konsolę na swoim komputerze.
2. Wygeneruj parę kluczy SSH za pomocą polecenia "ssh-keygen". Możesz wybrać domyślne ustawienia, naciskając enter na każdym pytaniu, lub wprowadzić swoje własne ustawienia, jak nazwa klucza czy hasło.

```
ssh-keygen
```

3. Otwórz plik z kluczem publicznym za pomocą polecenia "cat":

```
cat ~/.ssh/id_rsa.pub
```

4. Skopiuj klucz publiczny.
5. Zaloguj się do swojego konta na GitHub.
6. Kliknij swoją ikonę profilu w prawym górnym rogu strony i wybierz opcję "Settings".
7. Wybierz zakładkę "SSH and GPG keys".
8. Kliknij przycisk "New SSH key".
9. Wprowadź nazwę klucza i wklej klucz publiczny, który skopiowałeś wcześniej.
10. Kliknij przycisk "Add SSH key".

Teraz Twój komputer jest połączony z Twoim kontem GitHub za pomocą klucza SSH. Możesz teraz łatwo klonować, aktualizować i wysyłać zmiany do repozytoriów GitHub, bez konieczności wprowadzania hasła przy każdej operacji.

Wrzucanie plików do Github za pomocą `git add`, `git commit`, `git push`

Aby wrzucić zmiany w plikach do GitHub, musisz wykonać kilka kroków:

1. Upewnij się, że masz skonfigurowane połączenie SSH z GitHub
2. Stwórz lokalne repozytorium na swoim komputerze i zainicjuj go jako repozytorium Git.

```
mkdir my-project
cd my-project
git init
```

Dodaj pliki, których chcesz dokonać zmian do stage (staging area) za pomocą komendy `git add .`

`git add file1.txt file2.txt`

Możesz również dodać wszystkie zmodyfikowane pliki na raz, wykonując komendę `git add .` (kropka oznacza wszystkie pliki w bieżącym katalogu).

3. Zatwierdź zmiany, wykonując komendę "git commit". Podaj krótki opis zmian, które wprowadziłeś.

`git commit -m "Added file1.txt and file2.txt"`

4. Skopiuj adres URL repozytorium z GitHuba.
5. Podłącz swoje lokalne repozytorium do repozytorium zdalnego na GitHubie za pomocą komendy `git remote add`. Zastąp "user/repo" adresem URL swojego repozytorium.

`git remote add origin git@github.com:user/repo.git`

6. Wyślij zmiany do repozytorium zdalnego na GitHubie, wykonując komendę "git push".

```
git push -u origin main
```

Zastąp `main` nazwą swojego głównego brancha, jeśli używasz innej nazwy.

Twoje zmiany zostaną przesłane na GitHub i będą dostępne dla innych osób, które mają dostęp do Twojego repozytorium.

Praca z repozytoriami za pomocą komend `git status`, `git log`, `git reset --hard`

Aby pracować z repozytoriami przy pomocy komend Git, należy znać kilka podstawowych komend. Poniżej przedstawiam krótki opis komend git status, git log oraz git reset --hard:

`git status`: Ta komenda wyświetla aktualny stan Twojego repozytorium Git, czyli informacje o zmianach w plikach, które zostały dodane do śledzenia lub zmodyfikowane, ale jeszcze nie dodane do stage (staging area). Komenda wyświetla też informacje o tym, na jakim branchu się znajdujesz.

`git log` Ta komenda wyświetla historię Twojego repozytorium Git. Wyświetla wszystkie zatwierdzone zmiany, które zostały wprowadzone do repozytorium, wraz z informacjami o autorze, dacie, komentarzu i unikalnym identyfikatorze SHA. Możesz także filtrować wyniki i wyświetlać tylko zmiany z określonego brancha, na przykład git log master.

`git reset --hard`: Ta komenda pozwala cofnąć zmiany w repozytorium do wcześniejszego stanu. Komenda usuwa wszystkie zatwierdzone zmiany, które nie zostały wysłane na zdalne repozytorium. Ta komenda jest bardzo potężna i należy jej używać ostrożnie,

ponieważ może spowodować utratę niezapisanych zmian. Dlatego warto użyć komendy git status i git log przed wykonaniem tej komendy, aby upewnić się, że cofasz zmiany do właściwego stanu.

Przykładowe użycie tych komend:

- `git status` Wyświetla bieżący stan repozytorium.
- `git log` Wyświetla historię zmian w repozytorium.
- `git reset --hard HEAD~1` Cofa ostatnie zatwierdzone zmiany w repozytorium. Znak tilde (~) i liczba oznaczają, do którego wcześniejszego stanu chcesz się cofnąć. W tym przypadku cofnięcie się o jeden commit. Pamiętaj, że ta komenda usuwa wszystkie niezatwierdzone zmiany, więc nie powinna być stosowana bezpośrednio na produkcji.

Synchronizacja repozytorium z Github za pomocą Download zip, `git clone`, `git pull`

Synchronizacja repozytorium z Github polega na pobieraniu zmian z zdalnego repozytorium na serwerach Github na Twój lokalny komputer. Aby to zrobić, możesz użyć dwóch podstawowych komend Git: `git clone` i `git pull`.

`git clone`: Ta komenda pobiera kopię zdalnego repozytorium na Twój lokalny komputer i tworzy nowe repozytorium Git. Komenda git clone jest wykonywana tylko raz - podczas tworzenia lokalnej kopii repozytorium. Ta komenda kopiuje całą historię zmian w repozytorium, więc jest ona bardzo przydatna, jeśli chcesz zacząć pracę z repozytorium, które jest już istniejące.

`git pull`: Ta komenda pobiera najnowsze zmiany ze zdalnego repozytorium i łączy je z Twoim lokalnym repozytorium. Komenda git pull jest wykonywana, gdy chcesz zaktualizować swoje lokalne repozytorium i wprowadzić do niego najnowsze zmiany. Ta komenda jest przydatna, jeśli pracujesz z repozytorium, które jest już zainicjowane i posiada historię zmian.

Przykładowe użycie tych komend:

- `git clone https://github.com/username/repository.git` Pobiera zdalne repozytorium i tworzy lokalną kopię na Twoim komputerze.
- `git pull origin main` Pobiera najnowsze zmiany z brancha main na zdalnym repozytorium i wprowadza je do Twojego lokalnego repozytorium. origin to alias zdalnego repozytorium, a main to nazwa brancha, z którego chcesz pobrać najnowsze zmiany.

Używanie tych komend pozwala na synchronizację lokalnego repozytorium z repozytorium na Github. Dzięki temu masz pewność, że Twoje lokalne repozytorium jest zawsze aktualne i zawiera najnowsze zmiany, które zostały wprowadzone przez innych użytkowników repozytorium.

Zakładka do GIT w Visual Studio Code

Po zainstalowaniu Git i rozszerzenia Git dla Visual Studio Code, otwórz projekt lub repozytorium, którego chcesz użyć, w Visual Studio Code. W lewej dolnej części edytora znajdziesz zakładkę Git. Kliknij na nią, aby otworzyć panel Git.

W panelu Git znajdziesz różne opcje i polecenia, takie jak:

- Initialize Repository: Umożliwia inicjalizację nowego repozytorium Git w bieżącym katalogu.
- Clone Repository: Umożliwia sklonowanie istniejącego repozytorium Git z Githuba lub innej platformy do bieżącego katalogu.
- Add Changes: Dodaje zmienione lub nowe pliki do indeksu Git.
- Commit: Tworzy nowy commit z wprowadzonymi zmianami i dodaje go do lokalnego repozytorium Git.
- Push: Wysyła lokalne commity do zdalnego repozytorium na Github.
- Pull: Pobiera zmiany z zdalnego repozytorium i łączy je z lokalnym repozytorium.
- Branches: Umożliwia przeglądanie istniejących branchy oraz tworzenie i usuwanie nowych.

Wszystkie te funkcje można wywołać bezpośrednio z poziomu zakładki Git w Visual Studio Code. Panel Git wyświetla również listę commitów, w której można przeglądać historię zmian w repozytorium i dokonywać porównań między różnymi wersjami plików.

Prework JS

Zadanie 1. JS

```
// Policz sume produktow

const products = [
  {
    name: 'Chleb',
    price: 4.99
  },
  {
    name: "Pomidory",
    price: 9.99
  },
  {
    name: "Olowek",
    price: 1.20
  }
]
```

Zadanie 2. HTML/CSS + DOM

```
// 1. Stworz formularz z inputem o nazwie task-title i buttonem Add Todo.
// 2. Stworz pusta liste z 0 elementami
// 3. Po wcisnieciu buttona, dodaj do listy zadanie (tresc zadania pobierz z inputa)
```
