



Szkoła Programowania Online

Obsługa linii komend

Wydanie z dnia 13.11.2017

Spis treści

Wstęp	3
1. Terminal, co to takiego?	4
2. Jak wygląda terminal?	5
2.1. Prompt - znak zachęty	6
2.2. Sesja	7
3. Rozpoczęcie pracy z terminalem	9
3.1. Otwieranie terminala	9
3.2. Katalog roboczy	12
3.3. Listowanie zawartości katalogów	13
3.4. Przechodzenie do innych katalogów	15
4. Manipulacja plikami	18
4.1. Tworzenie katalogów	18
4.2. Tworzenie pustych plików	19
4.3. Kopiowanie plików i katalogów	19
4.4. Przenoszenie i zmiana nazwy plików lub katalogów	22
4.5. Usuwanie plików i katalogów	23
5. Czytanie i edycja plików w terminalu	25
5.1. Czytanie zawartości plików	25
5.2. Edytory plików w oknie terminala	27
6. Przydatne elementy składni	38
6.1. Asterisk, czyli "gwiazdka"	38
6.2. Łączenie kilku komend	39
6.3. Zapisywanie komunikatu zwrotnego komendy do pliku	42
7. Obsługa terminala	45
7.1. Edycja wpisanej komendy	45
7.2. Historia komend	45
7.3. Kopiowanie i wklejanie	45
7.4. Uzupełnianie komend i ścieżek	46
8. Czytanie komunikatów zwrotnych	47
9. Zadanie treningowe	49
10. Podsumowanie	50

Wstęp

Udostępniamy Wam ten poradnik jako uzupełnienie wiedzy przekazywanej w trakcie naszych Bootcampów. Znajdziesz w nim informacje dot. podstawowych komend, takich jak `cd` czy `touch`, a także informacje dot. edytorów tekstu dostępnych z poziomu terminala oraz przydatnych elementów składni i obsługi terminala.

Poradnik ten został napisany z myślą o osobach, które do tej pory nie miały kontaktu z terminalem linii komend, albo korzystały z niego sporadycznie. Przyda się również osobom, które do tej pory korzystały jedynie z wiersza poleceń w systemie Windows.

Informacja dla użytkowników systemu Windows

W tym poradniku uczymy obsługi linii komend typu **Bash**, która jest domyślną w systemach macOS oraz Linux. Jest to o tyle istotne, że większość serwerów pracuje pod kontrolą Linuksa, a w swojej pracy możesz mieć potrzebę wykonania prostych operacji za pomocą terminala.

Oczywiście, nie musisz zmieniać systemu operacyjnego tylko po to, aby skorzystać z tego poradnika. Wystarczy, że skorzystasz z programu **Git Bash**, który jest instalowany wraz z **Gitem**. Jeśli jeszcze nie masz go zainstalowanego, możesz zrobić to teraz - wystarczy pobrać i uruchomić [instalator Gita](#). W trakcie instalacji możesz pozostawić wszystkie domyślne opcje bez zmiany.

Pamiętaj, że w tym poradniku zawsze będziemy mieć na myśli Git Bash, pisząc o "terminalu" czy "konsoli".

1. Terminal, co to takiego?

Terminal jest jednym z podstawowych narzędzi web developera, służącym do komunikacji z komputerem. Jedyną różnicą jest to, że komunikować się będziemy za pomocą tekstu, a nie za pomocą ikon i guzików, jak ma to miejsce w interfejsie graficznym (**GUI** - [Graphical User Interface](#)).

Terminal pozwala m.in. na korzystanie z narzędzi, które nie posiadają interfejsu graficznego - a nawet tworzenie własnych narzędzi. Co więcej, po nabraniu wprawy, korzystanie z linii komend (**CLI** - [Command Line Interface](#)) staje się znacznie wygodniejsze i szybsze również w zakresie obsługi plików (np. tworzenia, przenoszenia czy usuwania).

Uwaga językowa

Określenia takie jak **terminal**, **konsola**, **linia komend**, **wiersz poleceń**, **bash**, **shell** czy **CLI** (Command Line Interface) są używane zamiennie. Bywa to tematem sporów, ponieważ każde z nich ma nieco inne znaczenie. Formalnie będziemy w tym kursie odnosić się do **CLI**, czyli **wiersza poleceń**. Pozwolimy sobie jednak na używanie określeń "**terminal**" czy "**konsola**", zgodnie z ich potocznym znaczeniem.

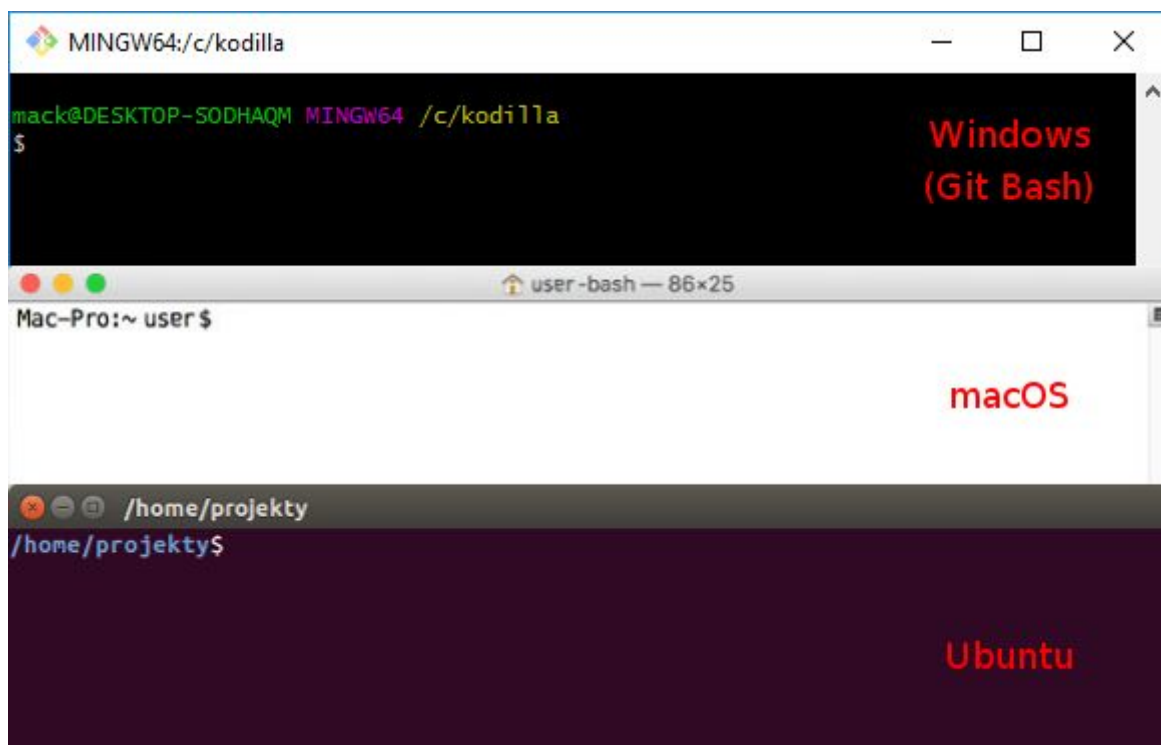
Wielu programistów preferuje korzystanie z **CLI**, ponieważ:

- nie trzeba odrywać rąk od klawiatury - obsługa myszki czy touchpada zajmuje więcej czasu,
- umożliwia korzystanie z narzędzi które nie mają **GUI**, albo przynajmniej oszczędza czas na poszukiwanie narzędzi z interfejsem graficznym,
- znacznie ułatwia konfigurowanie obsługi projektu, która pozwala na uruchamianie wielu narzędzi (takich jak np. Sass) za pomocą jednej komendy task runnera (np. Gulpa),
- przy pracy na serwerze często do dyspozycji jest wyłącznie **CLI** - w takich sytuacjach umiejętność kopiowania, przenoszenia czy rozpakowania plików może być niezbędna.

Zanim jednak przejdziemy do nauki pracy z terminalem, musimy zapoznać się z pewnymi podstawowymi kwestiami. Wiesz już że terminal jest narzędziem, za pomocą którego komunikujemy się z komputerem przez wydawanie mu poleceń wpisywanych z klawiatury. W tym poradniku dowiesz się w jaki sposób działa i jak go wykorzystać do własnych potrzeb.

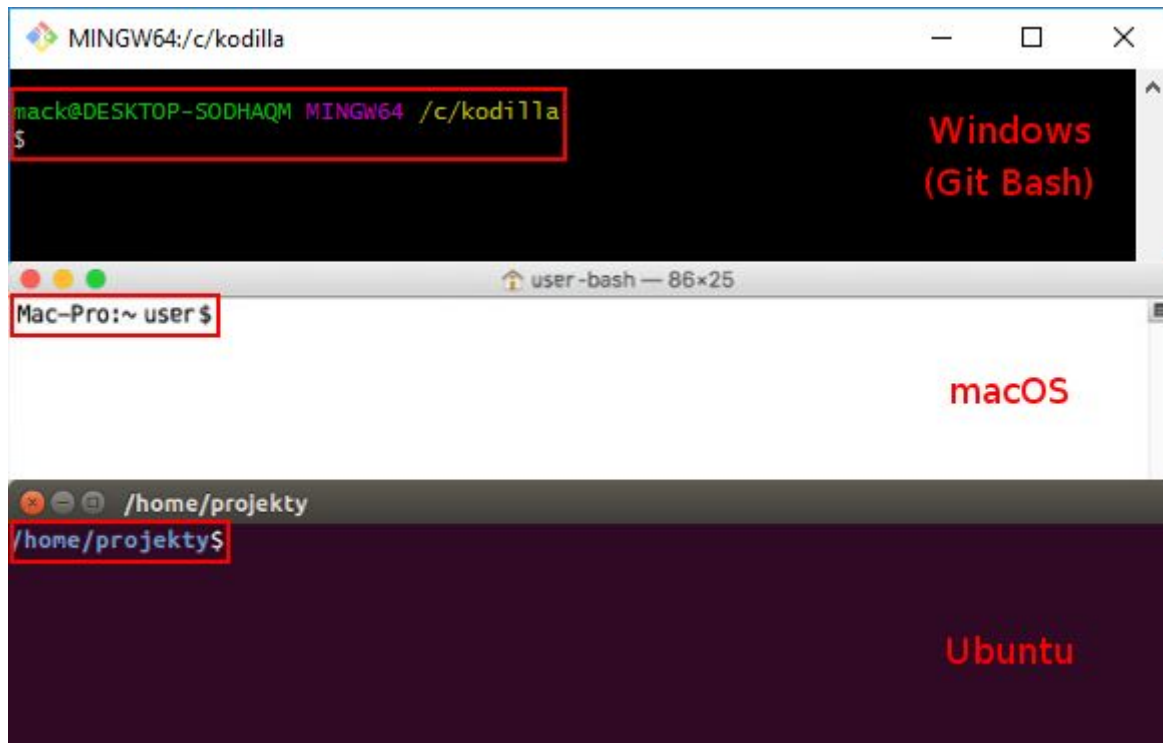
2. Jak wygląda terminal?

Są różne rodzaje linii komend. Każdy system operacyjny ma swoją, np. **cmd** dla Windowsa czy **bash** na MacOS. Istnieją też inne terminale, które wprowadzają dodatkowe funkcjonalności (np. dla Windowsa warto wspomnieć o programie **Git Bash**). Mimo tego wszystkie z nich wyglądają i działają podobnie:



2.1. Prompt - znak zachęty

W terminalu warto zwrócić uwagę na tzw. znak zachęty (ang. **prompt**), zaznaczony na poniższej ilustracji.



Może się on różnić między systemami operacyjnymi, ale przeważnie zawiera ścieżkę do katalogu, w którym aktualnie się znajdujesz. To właśnie w tym katalogu będą wykonywane komendy, np. tworzone pliki.

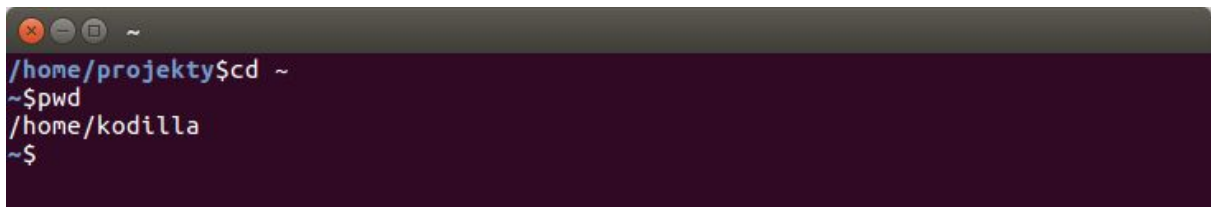
Notatka dla użytkowników systemu Windows

Windows wyświetla ścieżki do plików z **backslashem**, czyli "\". W pozostałych systemach, jak również na serwerach, używa się jednak **forward slash**a, czyli "/.

Pamiętaj, aby zarówno w terminalu jak i w plikach używać tylko forward-slasha. Na większości klawiatur wpisuje się go za pomocą kombinacji klawiszy **shift+?**.

Zwróć też uwagę na inne oznaczenie dysku w **Git Bashu** - na powyższej ilustracji widzisz w znaku zachęty ścieżkę **"/c/kodilla"**, która odpowiada ścieżce **"C:\kodilla"** w eksploratorze plików.

Musisz też wiedzieć, że znak `~` (tylda) oznacza katalog domowy użytkownika. Pełną ścieżkę tego katalogu możemy poznać przechodząc do niego (komenda `cd ~`) i sprawdzając jego ścieżkę (komenda `pwd`).

A terminal window with a dark purple background. The prompt is `/home/projekty$`. The user enters `cd ~` and the prompt changes to `~$`. Then the user enters `pwd` and the terminal outputs `/home/kodilla`. The prompt returns to `~$`.

```
/home/projekty$ cd ~
~$ pwd
/home/kodilla
~$
```

Prompt zwykle zakończony jest znakiem `$`, po którym wpisujemy nasze komendy.

2.2. Sesja

Duża część wpisywanych komend zakończy się automatycznie. Poznasz to po fakcie, że ponownie pojawi się **prompt** (znak zachęty). Niektóre komendy będą zwracać jakiś komunikat (np. listę plików), zaś inne nie wyświetlą żadnego komentarza, co zwykle oznacza, że wykonały się poprawnie.

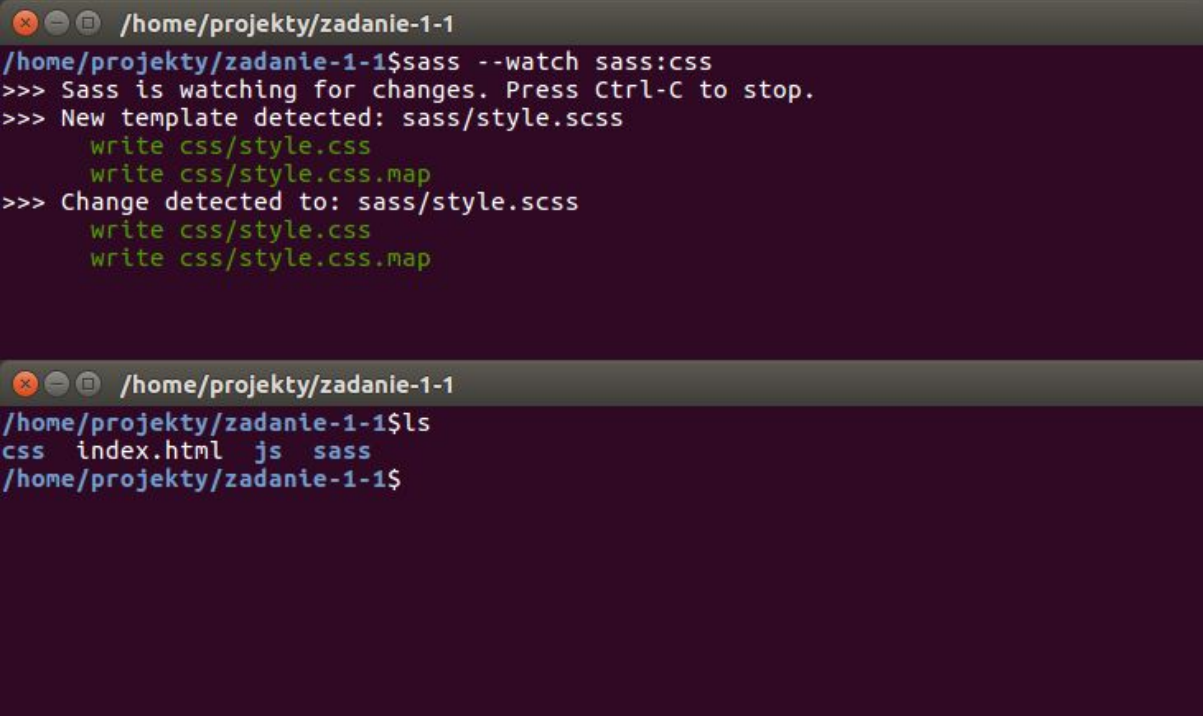
Na poniższej ilustracji możesz zobaczyć efekt wylistowania plików (komenda `ls`), jak również stworzenie nowego katalogu (komenda `mkdir`, brak komunikatu), oraz ponowne wylistowanie plików w celu pokazania, że komenda `mkdir` zadziałała poprawnie.

A terminal window with a dark purple background. The prompt is `/home/projekty$`. The user enters `ls` and the terminal outputs `zadanie-1-1` and `zadanie-1-2`. The prompt returns to `/home/projekty$`. The user enters `mkdir zadanie-1-3` and there is no output. The prompt returns to `/home/projekty$`. The user enters `ls` and the terminal outputs `zadanie-1-1`, `zadanie-1-2`, and `zadanie-1-3`. The prompt returns to `/home/projekty$`.

```
/home/projekty$ ls
zadanie-1-1  zadanie-1-2
/home/projekty$ mkdir zadanie-1-3
/home/projekty$ ls
zadanie-1-1  zadanie-1-2  zadanie-1-3
/home/projekty$
```

Może się zdarzyć, że jakaś komenda będzie miała za zadanie działać ciągle, i nie zakończy się automatycznie. Przykładem może być uruchomienie kompilatora Sassa z flagą `--watch`, dzięki której kompilator będzie działał "do odwołania" i obserwował pliki `*.scss` i `*.sass`. Jak sobie wtedy poradzić z wykonaniem jakiegoś polecenia w linii komend?

Wystarczy otworzyć kolejne okno terminala. Każde z nich to jedna sesja, których możesz otworzyć praktycznie nieograniczoną ilość. Dzięki temu każde okno terminala może mieć uruchomiony jakiś proces, a my tak czy inaczej możemy otworzyć kolejną sesję, aby uruchomić inną komendę bez przerywania działającego procesu (np. kompilatora Sass).

The image shows two overlapping terminal windows. The top window has a title bar with standard Linux window controls and the text "/home/projekty/zadanie-1-1". The terminal content shows the command "sass --watch sass:css" being executed. The output indicates that Sass is watching for changes and has detected a new template, "sass/style.scss", which it has compiled into "css/style.css" and "css/style.css.map". The bottom window also has a title bar with the same text. Its terminal content shows the command "ls" being executed, with the output listing the files "css", "index.html", "js", and "sass". The prompt character "\$" is visible at the end of the line.

Na powyższej ilustracji możesz zobaczyć dwa jednocześnie otwarte okna terminala:

- górne okno, w którym ciągle pracuje `sass --watch`, więc nie da się w nim uruchomić innej komendy bez przerywania pracy tego programu,
- dolne okno, w którym po wykonaniu komendy `ls` z powrotem pojawił się znak zachęty (prompt), mówiący nam że możemy wpisać kolejną komendę.

Jeżeli usłyszysz lub przeczytasz "otwórz nową sesję" lub "otwórz nowy terminal", oznacza to właśnie uruchomienie kolejnego okna terminala.

3. Rozpoczęcie pracy z terminalem

Zaczynamy osvajanie się z terminalem. Najpierw otwórz terminal...

3.1. Otwieranie terminala

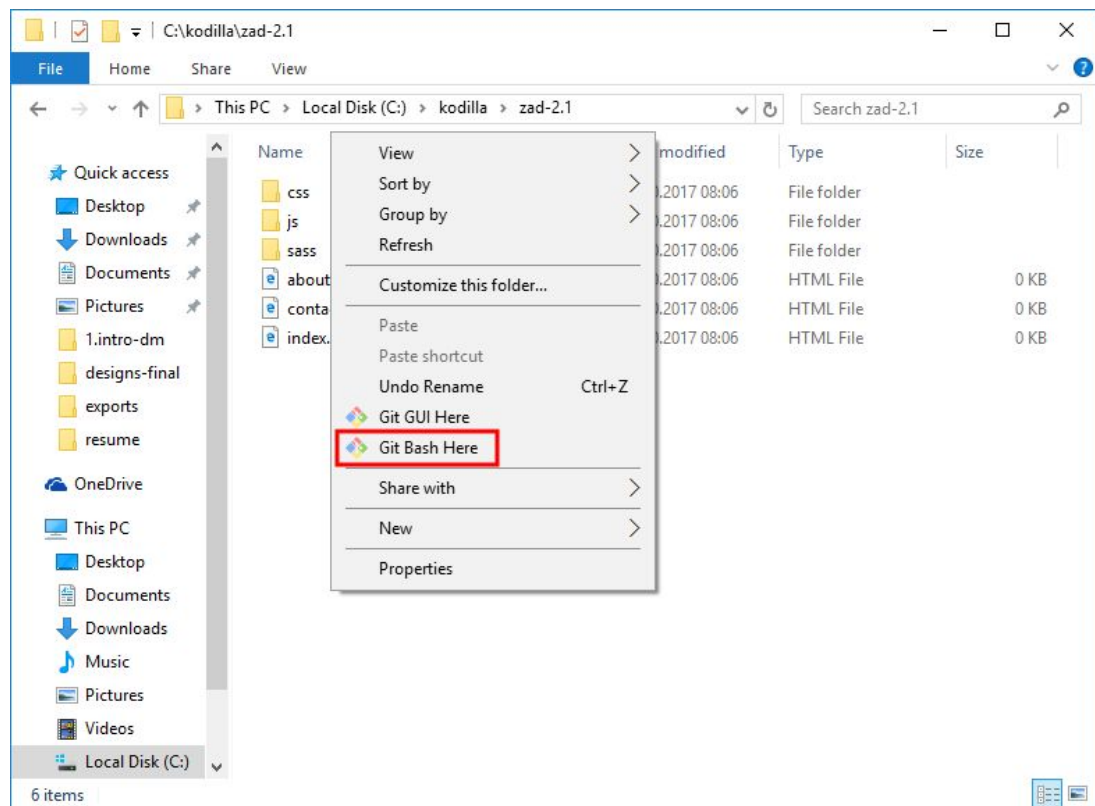
Terminal możesz otworzyć tak samo jak każdy inny program - np. z menu Start w Windowsie albo katalogu Programy w macOS. Innym sposobem jest otwarcie terminala za pomocą przeglądarki plików z menu kontekstowego (pod **PPM**, czyli po kliknięciu prawym przyciskiem myszy).

W zależności od systemu operacyjnego procedura wygląda nieco inaczej i może wymagać dodatkowej konfiguracji. Z tego względu poniżej znajdziesz instrukcje dla systemów Windows, macOS oraz Linux.

Windows

Możesz uruchamiać Git Bash w jeden z następujących sposobów:

1. Z menu start uruchom program Git Bash - znajdziesz go w katalogu Git lub za pomocą wyszukiwarki.
2. Ze względu na katalog roboczy terminala (za chwilę wyjaśnimy co to jest) wygodniej jest otwierać Git Basha za pomocą eksploratora plików, w którym mamy wyświetlony np. katalog projektu:
 - a. kliknij PPM w dowolny katalog lub puste tło na liście plików,
 - b. z menu kontekstowego wybierz "Git Bash Here":



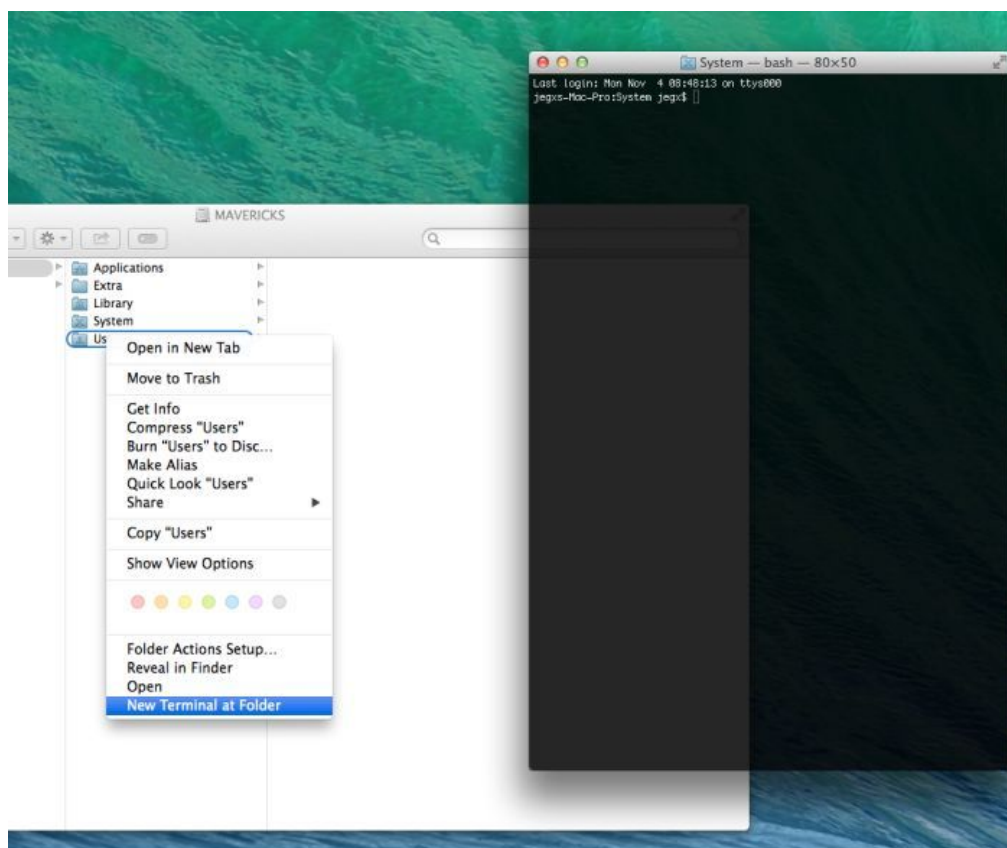
macOS

Możesz uruchomić terminal w jeden z następujących sposobów:

1. W *Finderze* wejdź do katalogu *Programy* (ang. Applications), następnie wejdź do katalogu *Narzędzia* (ang. Utilities), tam znajdziesz program *Terminal*.
2. Zamiast tego możesz też skorzystać z wyszukiwarki Spotlight, za pomocą lupy w prawym górnym rogu ekranu (lub skrótu klawiszowego $\text{⌘} + \text{Spacja}$) - w wyszukiwarce wpisz "terminal".
3. Ze względu na katalog roboczy terminala (za chwilę wyjaśnimy co to jest) wygodniej jest otwierać terminal za pomocą Findera, w którym mamy wyświetlony katalog z projektami - wymaga to jednak jednorazowego włączenia tej opcji:
 - a. otwórz Preferencje Systemowe,
 - b. wybierz sekcję Klawiatura,
 - c. przejdź do Skrótów Klawiszowych,
 - d. w sekcji Usługi (ang. Services) zaznacz pozycję "Nowy terminal w katalogu" (ang. "New Terminal at Folder"):



- e. od teraz po kliknięciu PPM dowolnego katalogu zobaczysz w menu kontekstowym opcję otwarcia terminala w tym katalogu:



Linux - Ubuntu

Możesz uruchomić terminal w jeden z następujących sposobów:

1. Otwórz program Terminal w dashu, podobnie jak otwierasz inne programy.

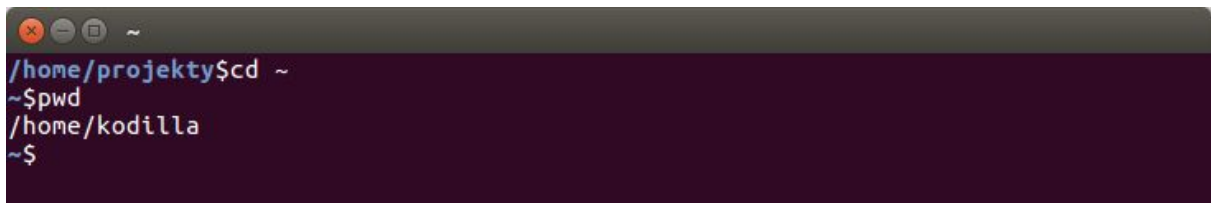
2. Ze względu na katalog roboczy terminala (za chwilę wyjaśnimy co to jest) wygodniej jest otwierać terminal za pomocą przeglądarki plików, w której mamy wyświetlony katalog z projektami:
 - a. w przeglądarce plików kliknij PPM na dowolny katalog (lub tło listy plików),
 - b. z menu kontekstowego wybierz "Otwórz w terminalu".

Linux - inne dystrybucje

Aby otworzyć terminal, spróbuj znaleźć go w menu, z którego otwierasz inne programy. Jeśli nie uda Ci się go znaleźć, użyj Google wpisując nazwę swojej dystrybucji i hasło "open terminal", np. "mint linux open terminal".

3.2. Katalog roboczy

Jak już wspominaliśmy, terminal zawsze wskazuje na jakiś katalog, podobnie jak przeglądarka plików wyświetla zawartość jednego katalogu. Zwykle katalog roboczy jest wyświetlany w znaku zachęty (prompt), ale zawsze możesz też sprawdzić to za pomocą komendy `pwd`.



```
/home/projekty$cd ~
~$pwd
/home/kodilla
~$
```

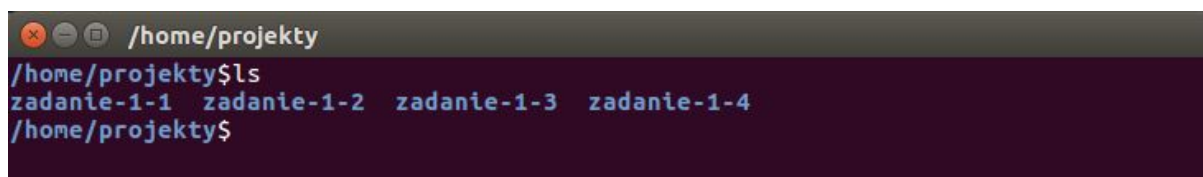
Przypomnienie

Pamiętaj, że znak `~` (tylda) oznacza katalog domowy Twojego użytkownika. Możesz sprawdzić ścieżkę tego katalogu na dysku za pomocą komendy `pwd` po przejściu do katalogu domowego (komenda `cd ~`).

Kiedy w terminalu wykonamy np. komendę tworzenia nowego katalogu (komenda `mkdir`, którą wyjaśnimy za chwilę) podając tylko jego nazwę, katalog zostanie stworzony właśnie w aktualnym katalogu roboczym. Lepiej zrozumiesz tę kwestię za chwilę, kiedy zaczniesz używać komend w terminalu.

3.3. Listowanie zawartości katalogów

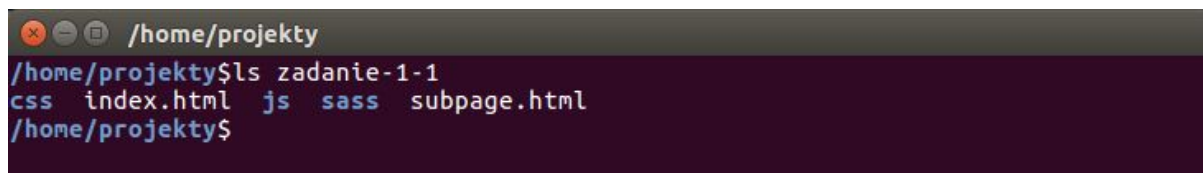
Wiesz już jak sprawdzić ścieżkę katalogu roboczego, a za chwilę nauczysz się przechodzenia do innych katalogów. Wcześniej jednak musisz wiedzieć w jaki sposób wypisać listę plików i katalogów w aktualnym katalogu roboczym. Do tego służy komenda `ls`.



```
/home/projekty
/home/projekty$ls
zadanie-1-1  zadanie-1-2  zadanie-1-3  zadanie-1-4
/home/projekty$
```

Domyślnie komenda `ls` wyświetla listę plików i katalogów (o ile nie są ukryte - o tym za chwilę) w aktualnym katalogu roboczym, czyli tym wyświetlanym w znaku zachęty (przed znakiem `$`).

Możemy jednak również wylistować zawartość z któregoś z podkatalogów, podając nazwę tego podkatalogu:



```
/home/projekty
/home/projekty$ls zadanie-1-1
css  index.html  js  sass  subpage.html
/home/projekty$
```

Jak widzisz w ostatniej komendzie na powyższej ilustracji użyliśmy nie tylko nazwy podkatalogu, ale ścieżki obejmującej dwa poziomy. Ta komenda wylistuje pliki i katalogi znajdujące się w katalogu `sass`, wewnątrz katalogu `zadanie-1-1`, który znajduje się w aktualnym katalogu roboczym, czyli `/home/projekty`.

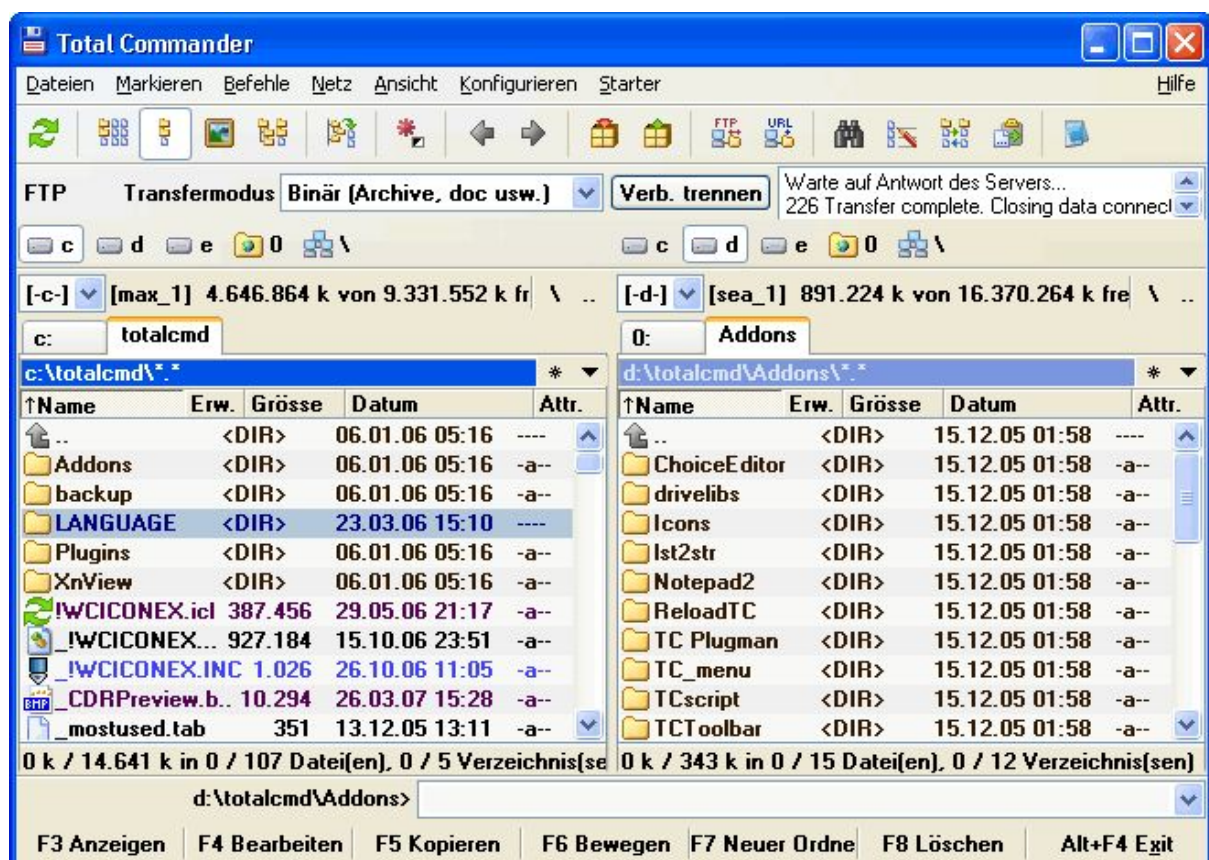
Zwróć uwagę, że nasz katalog roboczy, wyświetlany w znaku zachęty, nie zmienił się. Komenda `ls` tylko wyświetla listę plików i katalogów, ale nie zmienia katalogu roboczego terminala.

Analogicznie możesz stosować wszystkie rodzaje ścieżek, które poznasz za chwilę w rozdziale dot. przechodzenia do innych katalogów. Zanim jednak do tego przejdziemy, przećwicz używanie komendy `ls`.

Musisz wiedzieć, że w **Bashu** pliki i katalogi, których nazwa rozpoczyna się od kropki, są uznawane za ukryte i nie są wyświetlane przez komendę `ls`. Aby je wyświetlić należy dodać flagę `-a`, która wyświetli:

- wszystkie pliki i katalogi w danej lokalizacji, włącznie z ukrytymi,
- pozycję `...`, która oznacza ścieżkę do katalogu wyższego poziomu (tzn. katalogu, w którym znajduje się katalog, którego zawartość listujemy),
- pozycję `.`, która oznacza ścieżkę do obecnie przeglądanego katalogu (czyli "tutaj").

Te dwie ostatnie pozycje mogą Cię zdziwić, jeśli nie zdarzyło Ci się wcześniej korzystać z terminala, ale są one standardowym elementem struktury katalogów we wszystkich popularnych systemach operacyjnych. Nie są one jednak wyświetlane w większości przeglądarek plików - chyba że pamiętasz *Total Commandera*. ;)

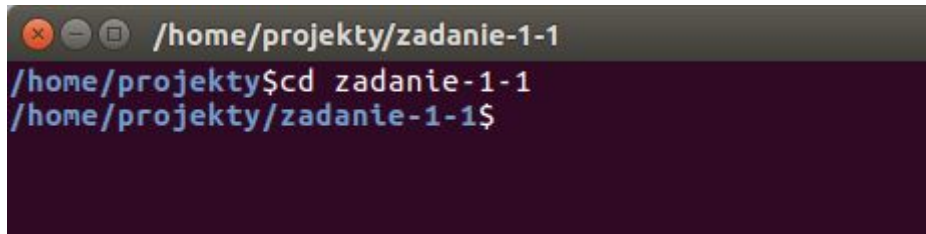


3.4. Przechodzenie do innych katalogów

Jak już wspomnieliśmy, do przechodzenia do innych katalogów (czyli zmiany katalogu roboczego terminala) będziemy używali komendy `cd`.

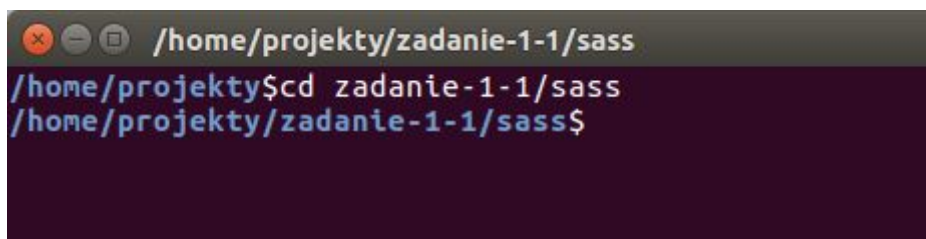
Można wykorzystać tę komendę na kilka sposobów:

- `cd nazwa-katalogu` - przejdź do katalogu, który znajduje się w aktualnym katalogu roboczym terminala:



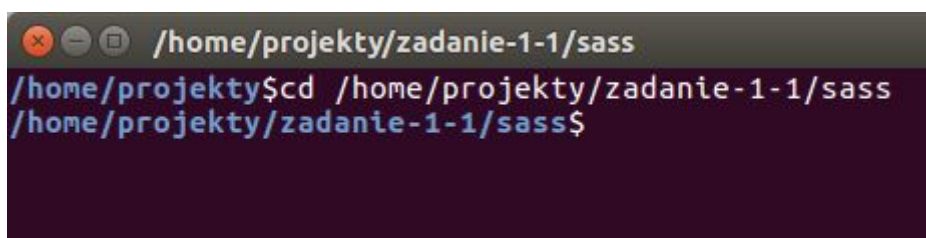
```
/home/projekty/zadanie-1-1
/home/projekty$ cd zadanie-1-1
/home/projekty/zadanie-1-1$
```

- `cd nazwa-katalogu/nazwa-podkatalogu` - rozpoczynając od aktualnego katalogu roboczego, przejdź do katalogu *nazwa-katalogu*, a w nim do *nazwa-podkatalogu*:



```
/home/projekty/zadanie-1-1/sass
/home/projekty$ cd zadanie-1-1/sass
/home/projekty/zadanie-1-1/sass$
```

- `cd /nazwa-katalogu/nazwa-podkatalogu` - jeśli ścieżkę rozpoczniemy od znaku `/` (*forward slash*), będzie to ścieżka absolutna, czyli musimy wtedy podać pełną ścieżkę, niezależnie od tego jaki jest katalog roboczy naszego terminala:



```
/home/projekty/zadanie-1-1/sass
/home/projekty$ cd /home/projekty/zadanie-1-1/sass
/home/projekty/zadanie-1-1/sass$
```

- `cd ..` - w tym przypadku dwie kropki oznaczają "poziom wyżej", co pozwala nam do wyższego katalogu (`cd ..`), albo kilka poziomów wyżej (np. `cd ../../`), lub też do równoległego katalogu (np. `cd ../nazwa-katalogu`):

```

/home
/home/projekty/zadanie-1-1/sass$cd ..
/home/projekty/zadanie-1-1$cd ../inne-zadanie
/home/projekty/inne-zadanie$cd ../../
/home$

```

- `cd ~` - znak `~` (tylda) oznacza "katalog domowy":

```

/home/projekty$cd ~
~$pwd
/home/kodilla
~$

```

Możesz dowolnie na różne sposoby łączyć powyższe rodzaje ścieżek, aby w najwygodniejszy sposób zmieniać katalog roboczy.

Przypomnienie dla użytkowników systemu Windows

Pamiętaj, że w Git Bashu nie funkcjonują ścieżki jakie znasz z eksploratora plików.

Dla przykładu ścieżka:

C:\Windows\Fonts

w **Git Bashu** będzie miała postać:


/c/Windows/Fonts

czyli nazwa partycji **c** jest traktowana jako katalog najwyższego poziomu.

Wynika to z faktu, że w systemie Windows katalogi znajdują się na poszczególnych partycjach (potocznie "dyskach"), natomiast w systemach macOS i Linux partycje są podłączane jako wirtualne podkatalogi, a cały system (niezależnie od ilości dysków i partycji) ma tylko jeden katalog najwyższego rzędu, którego ścieżka to po prostu **/**.

Jako że zdecydowana większość serwerów działa w oparciu o dystrybucje linuksowe, w naszym kursie uczymy Cię właśnie tego podejścia i dlatego korzystamy z **Git Basha**.

Na poniższej ilustracji możesz zobaczyć jak wygląda terminal, którego aktualnym katalogiem roboczym jest **C:\kodilla**.



```
mingw64:/c/kodilla
mack@DESKTOP-SODHAQM MINGW64 /c/kodilla
$
```

Poćwicz przechodzenie pomiędzy katalogami, aby nabrać wprawy. Pamiętaj o używaniu komendy **ls** w celu wyświetlenia listy plików i katalogów. Jak już wspomnieliśmy, możesz też używać komendy **ls** ze wszystkimi rodzajami ścieżek wymienionymi dla komendy **cd**, czyli np. **ls ..** lub **ls /nazwa-katalogu/nazwa-podkatalogu**.

WIELKIE i małe litery

Pamiętaj, że w **Bashu** wielkość liter ma znaczenie. Oznacza to, że katalog *Projekty-Kodilla* to zupełnie inny katalog niż *projekty-kodilla*, mimo że składa się z tych samych liter.

4. Manipulacja plikami

Umiesz już poruszać się po drzewie katalogów. W takim razie czas na wprowadzanie zmian na dysku!

4.1. Tworzenie katalogów

Jeśli chcesz utworzyć podkatalog w aktualnym katalogu roboczym terminala, wystarczy że wykorzystasz komendę `mkdir` (skrót od ang. *make directory*) i wpiszesz w terminalu `mkdir nazwa-podkatalogu`.

```
/home/projekty$ls
zadanie-1-1  zadanie-1-2
/home/projekty$mkdir zadanie-1-3
/home/projekty$ls
zadanie-1-1  zadanie-1-2  zadanie-1-3
/home/projekty$
```

Pamiętaj, że komenda `mkdir` nie wyświetla żadnego komunikatu, chyba że wystąpił błąd - wedle zasady "brak wiadomości to dobre wiadomości". Dla przykładu, zobacz co się stanie, gdy spróbujemy ponownie utworzyć podkatalog o tej samej nazwie:

```
/home/projekty$ls
zadanie-1-1  zadanie-1-2
/home/projekty$mkdir zadanie-1-3
/home/projekty$ls
zadanie-1-1  zadanie-1-2  zadanie-1-3
/home/projekty$mkdir zadanie-1-3
mkdir: cannot create directory 'zadanie-1-3': File exists
/home/projekty$
```

Wyświetlony komunikat informuje nas o tym, że nie udało się stworzyć katalogu o nazwie `zadanie-1-3`, ponieważ już istnieje katalog (lub plik) o takiej nazwie (a konkretniej, o tej samej ścieżce, czyli `/home/projekty/zadanie-1-3`). W komunikacie jest mowa o pliku, ponieważ dla komendy `mkdir` nie robi różnicy czy istnieje taki plik czy katalog - ważne, że ta ścieżka jest już *zajęta*.

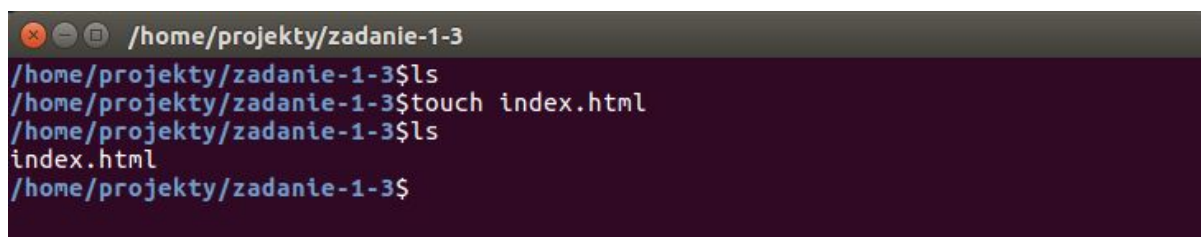
Z komendą `mkdir` możesz używać wszystkich rodzajów ścieżek, które wymieniliśmy przy komendzie `cd`, więc zadziała np. komenda:

```
mkdir ../nazwa-katalogu
```

4.2. Tworzenie pustych plików

Za pomocą terminala można też bardzo łatwo tworzyć puste pliki. Przydaje się to np. w momencie przygotowywania struktury plików nowego projektu, kiedy chcemy szybko stworzyć wiele plików, które później będą nam potrzebne.

Jedną z bardziej popularnych metod tworzenia pustych plików jest komenda `touch` (z ang. *dotknij*). W swoim założeniu komenda ta służy do zmiany daty modyfikacji pliku na aktualną (bez zmiany zawartości), ale jeśli plik o danej nazwie nie istnieje, zostanie utworzony. Dzięki temu jest to bezpieczna komenda, która nie usunie nam zawartości istniejącego pliku.



```
/home/projekty/zadanie-1-3
/home/projekty/zadanie-1-3$ls
/home/projekty/zadanie-1-3$touch index.html
/home/projekty/zadanie-1-3$ls
index.html
/home/projekty/zadanie-1-3$
```

Tak samo jak przy komendzie `mkdir`, z komendą `touch` możemy używać różnego rodzaju ścieżek do plików, które poznaliśmy przy okazji komendy `cd`.

4.3. Kopiowanie plików i katalogów

Jak już zapewne się domyślasz, kopiowanie plików i katalogów będzie działać bardzo podobnie do komendy `touch`. Główna różnica polega na tym, że w przypadku komendy `cp` (skrót od ang. *copy*) będziemy musieli podać dwie ścieżki (z nazwami lub bez nich) - pierwsza powie nam co skopiować, a druga - dokąd.

Zacznijmy od skopiowania pliku - założmy, że przypadkowo w pliku `_header.scss` zostały napisane style grida, które wolisz mieć w osobnym pliku. Po upewnieniu się, że jesteś w katalogu `sass`, możesz wykonać komendę `cp _header.scss _grid.scss`.

```
/home/projekty/zadanie-1-1/sass
/home/projekty/zadanie-1-1/sass$ls
_footer.scss _header.scss style.scss
/home/projekty/zadanie-1-1/sass$cp _header.scss _grid.scss
/home/projekty/zadanie-1-1/sass$ls
_footer.scss _grid.scss _header.scss style.scss
/home/projekty/zadanie-1-1/sass$
```

Oczywiście, możemy osiągnąć dokładnie ten sam efekt np. kiedy katalogiem roboczym naszego terminala jest wyższy katalog. Cofnijmy się w czasie i zobaczmy jak moglibyśmy wykonać tę samą akcję zamiast powyższego przykładu:

```
/home/projekty/zadanie-1-1
/home/projekty/zadanie-1-1$ls sass
_footer.scss _header.scss style.scss
/home/projekty/zadanie-1-1$cp sass/_header.scss sass/_grid.scss
/home/projekty/zadanie-1-1$ls sass
_footer.scss _grid.scss _header.scss style.scss
/home/projekty/zadanie-1-1$
```

Podobnie jak przy komendzie `cd`, możemy używać różnego rodzaju kombinacji ścieżek.

Warto też wspomnieć, że drugi parametr ("dokąd") może, ale nie musi zawierać nazwy kopiowanego pliku lub katalogu. Weźmy na przykład dwie sytuacje - jeśli kopiujemy plik *index.html* do innego katalogu, ale nie zmieniamy jego nazwy, wystarczy komenda `cp projekt1/index.html projekt2`, ale gdybyśmy chcieli jednocześnie zmienić nazwę kopiowanego pliku z *index.html* na *blog.html*, moglibyśmy użyć komendy `cp projekt1/index.html projekt2/blog.html`.

```
/home/projekty
/home/projekty$ls
zadanie-1-1 zadanie-1-2 zadanie-1-3
/home/projekty$ls zadanie-1-1
css index.html js sass
/home/projekty$ls zadanie-1-2
/home/projekty$cp zadanie-1-1/index.html zadanie-1-2
/home/projekty$ls zadanie-1-2
index.html
/home/projekty$cp zadanie-1-1/index.html zadanie-1-2/blog.html
/home/projekty$ls zadanie-1-2
blog.html index.html
/home/projekty$
```

A co zrobić, kiedy odpowiedź na pytanie "dokąd?" brzmi "tutaj" i nie chcemy zmieniać nazwy pliku? Wiesz już, że skopiowanie z podaniem nowej nazwy pliku wyglądałoby tak: `cp ../projekt1/index.html index.html`. Skoro jednak nie zmieniamy nazwy pliku, możemy użyć pojedynczej kropki, która oznacza właśnie "tutaj", czyli aktualny katalog roboczy terminala. Użyjemy do tego komendy `cp ../projekt1/index.html .`

```
/home/projekty/zadanie-1-3
/home/projekty/zadanie-1-3$ls
/home/projekty/zadanie-1-3$cp ../zadanie-1-1/index.html .
/home/projekty/zadanie-1-3$ls
index.html
/home/projekty/zadanie-1-3$
```

Spróbujmy teraz użyć komendy `cp` do skopiowania katalogu `sass` z jednego projektu do drugiego:

```
/home/projekty
/home/projekty$cp zadanie-1-1/sass zadanie-1-2/sass
cp: omitting directory 'zadanie-1-1/sass'
/home/projekty$ls zadanie-1-2
/home/projekty$
```

Wyświetlony komunikat informuje nas, że katalog (ang. *directory*) który chcieliśmy skopiować, został pominięty (ang. *omitting*). I faktycznie, wylistowanie katalogu z drugim projektem pokazuje, że jest on pusty.

Stało się tak dlatego, że domyślnie komenda `cp` nie kopiuje katalogów, a wyłącznie pliki. Aby skopiować również katalogi (wraz z zawartymi w nich plikami i katalogami), należy dodać flagę `-r`, która oznacza "kopiuj rekursywnie". Jeśli kiedykolwiek zapomnisz w jaki sposób osiągnąć ten efekt, możesz wpisać komendę `cp --help`, która wyświetli większość dostępnych flag i opcji, jak również składnię polecenia. Większość programów obsługiwanych z linii komend będzie wyświetlała takie informacje po dodaniu flagi `--help` (zwróć uwagę, że używamy dwóch myślników). Sprawdź samodzielnie co wyświetli komenda `cp --help`.

Wracając do naszego przykładu, dodanie flagi `-r` sprawi, że katalog zostanie skopiowany wraz z zawartością:

```
/home/projekty
/home/projekty$ cp -r zadanie-1-1/sass zadanie-1-2/sass
/home/projekty$ ls zadanie-1-2
sass
/home/projekty$ ls zadanie-1-2/sass
_footer.scss _grid.scss _header.scss style.scss
/home/projekty$
```

4.4. Przenoszenie i zmiana nazwy plików lub katalogów

W tej kwestii proponujemy małe ćwiczenie samodzielności - spróbuj samodzielnie zastosować komendę `mv` (skrót od ang. *move*), która przenosi pliki i/lub katalogi.

Poniżej znajdziesz kilka podpowiedzi:

- komenda `mv` działa bardzo podobnie do komendy `cp`,
- możesz wytłumaczyć to sobie tak, że komenda `mv` robi dokładnie to samo co komenda `cp`, tyle że kasuje źródło z którego kopiowaliśmy,
- użyj flagi `--help` aby poznać opcje komendy `mv`,
- czy komenda `mv` potrzebuje flagi `-r` do przenoszenia katalogów?
- jeśli nie, to czy flaga `-r` oznacza coś innego dla komendy `mv` niż oznaczała dla komendy `cp`?
- dlaczego w nazwie tego podrozdziału zawarliśmy zarówno przenoszenie, jak i zmianę nazwy plików, mimo że mówimy tutaj tylko o jednej komendzie?

Jeśli będziesz mieć problemy z odpowiedziami na te pytania, porozmawiaj o nich na czacie swojej grupy Bootcampa.

4.5. Usuwanie plików i katalogów

Nawet jeśli wcześniej terminal był dla Ciebie czarną magią, pewnie już widzisz, że nie ma się czego bać. Aż do teraz! ;)

Przechodzimy do usuwania plików i katalogów, a więc tymi komendami możemy sobie narobić problemów, jeśli usuniemy nie to co trzeba. Pamiętaj, że **pliki usuwane za pomocą terminala nie trafiają do kosza**, tylko są od razu kasowane z dysku. Dlatego najlepiej zachować ostrożność i - korzystając z komend poznanych już w tym poradniku - stworzyć sobie katalog z zawartością przeznaczoną specjalnie do treningu kasowania plików.

Zacznijmy od usuwania plików - do tego służy komenda **rm** (skrót od ang. *remove*). Wystarczy, że po tej komendzie podasz ścieżkę do pliku, na przykład:

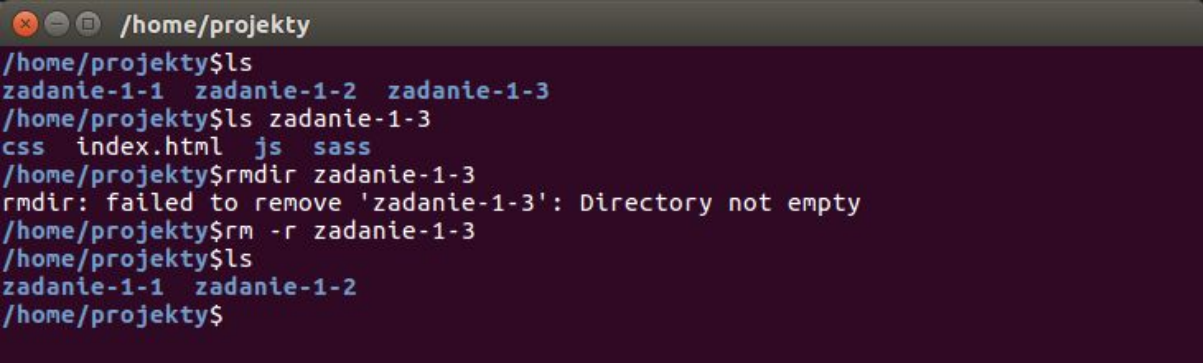
```
/home/projekty/zadanie-1-2/sass
/home/projekty/zadanie-1-2/sass$ls
_footer.scss _grid.scss _header.scss style.scss
/home/projekty/zadanie-1-2/sass$rm _footer.scss
/home/projekty/zadanie-1-2/sass$ls
_grid.scss _header.scss style.scss
/home/projekty/zadanie-1-2/sass$
```

W przypadku pustych katalogów kasowanie wygląda bardzo podobnie, tyle że za pomocą komendy **rmdir** - pokażemy to na przykładzie katalogu stworzonego *przypadkowo* z błędną nazwą:

```
/home/projekty/zadanie-1-2
/home/projekty/zadanie-1-2$ls
sass
/home/projekty/zadanie-1-2$mkdir csssss
/home/projekty/zadanie-1-2$ls
csssss sass
/home/projekty/zadanie-1-2$rmdir csssss
/home/projekty/zadanie-1-2$ls
sass
/home/projekty/zadanie-1-2$
```

Specjalnie mówimy tutaj o pustych katalogach, ponieważ twórcy tych komend przewidzieli, że w ten sposób można łatwo pozbawić się potrzebnych plików.

Właśnie dlatego usuwanie katalogów, w których jest jakakolwiek zawartość wymaga użycia komendy `rm` z flagą `-r`, która (podobnie jak przy komendzie `cp`) oznacza "kasuj rekursywnie". **Jeszcze raz zalecamy ostrożność przy stosowaniu tej komendy!**

A terminal window with a dark purple background and a title bar showing window controls and the path /home/projekty. The terminal text shows a series of commands and their outputs. First, 'ls' lists 'zadanie-1-1', 'zadanie-1-2', and 'zadanie-1-3'. Then, 'ls zadanie-1-3' shows its contents: 'css', 'index.html', 'js', and 'sass'. Next, 'rmdir zadanie-1-3' fails with the message 'rmdir: failed to remove 'zadanie-1-3': Directory not empty'. Finally, 'rm -r zadanie-1-3' is entered, and a subsequent 'ls' command shows that 'zadanie-1-1' and 'zadanie-1-2' remain, while 'zadanie-1-3' has been removed.

```
/home/projekty$ls
zadanie-1-1 zadanie-1-2 zadanie-1-3
/home/projekty$ls zadanie-1-3
css index.html js sass
/home/projekty$rmdir zadanie-1-3
rmdir: failed to remove 'zadanie-1-3': Directory not empty
/home/projekty$rm -r zadanie-1-3
/home/projekty$ls
zadanie-1-1 zadanie-1-2
/home/projekty$
```

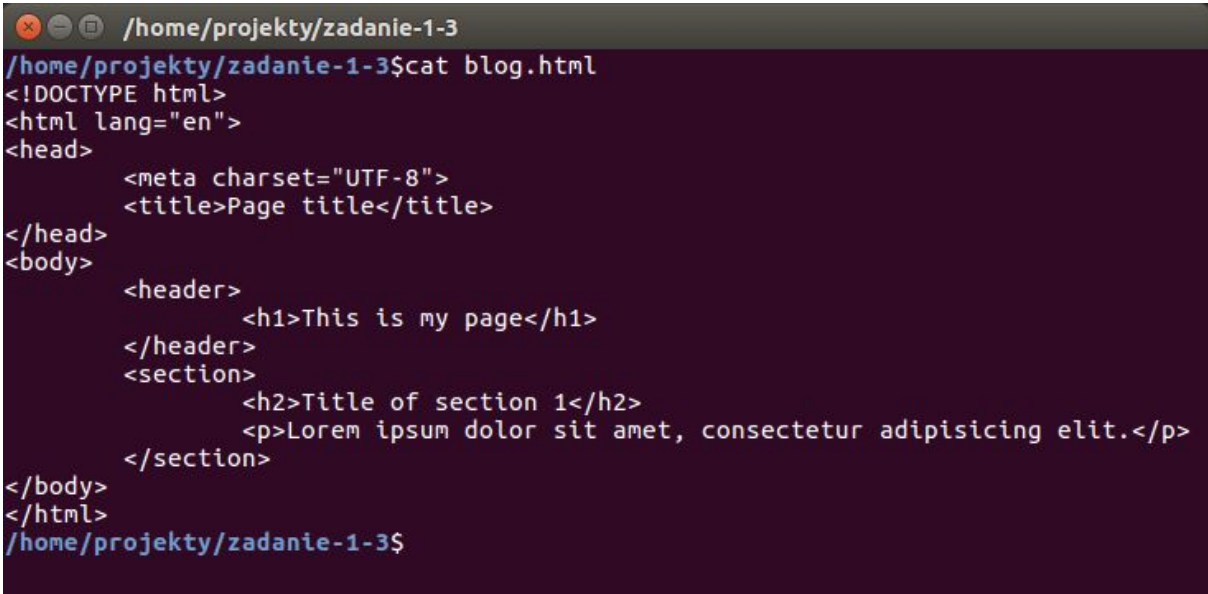

5. Czytanie i edycja plików w terminalu

W codziennej pracy niezbyt często zdarza się potrzeba czytania czy edycji plików z poziomu terminala - zwykle ma to miejsce w dwóch sytuacjach. Pierwszą z nich jest edycja pełnego opisu commita (o którym dowiesz się w kolejnym submodule).

Drugą, natomiast, jest praca zdalna na serwerze za pośrednictwem konsoli. W takich sytuacjach zwykle jest możliwość edycji plików lokalnie na komputerze i wysłania ich na serwer, ale zwykle dużo szybciej jest wprowadzić drobne zmiany bezpośrednio na serwerze.

5.1. Czytanie zawartości plików

Zacznijmy od wyświetlenia treści pliku. Podstawową komendą, która do tego służy, jest **cat**.



```
/home/projekty/zadanie-1-3
/home/projekty/zadanie-1-3$cat blog.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Page title</title>
</head>
<body>
  <header>
    <h1>This is my page</h1>
  </header>
  <section>
    <h2>Title of section 1</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
  </section>
</body>
</html>
/home/projekty/zadanie-1-3$
```

Bywa on jednak niewygodny przy dłuższych plikach, jak widać na poniższej ilustracji:

```

/home/projekty/zadanie-1-3
lla beatae nisi. Voluptatibus, debitis, placeat. Facilis quos vero, quis dolores
fuga!</p>
    </section>
    <section>
        <h2>Title of section 3</h2>
        <p>Laborum quam fugit repudiandae vel eveniet recusandae aperiam
repellendus consectetur enim modi molestiae consequatur ea corporis autem digni
ssimos minima, nihil reiciendis unde eum, placeat, facere ullam! Quam impedit po
rro sapiente.</p>
    </section>
    <footer>
        <nav>
            <ul>
                <li><a href="#">Link 1</a></li>
                <li><a href="#">Link 2</a></li>
                <li><a href="#">Link 3</a></li>
                <li><a href="#">Link 4</a></li>
                <li><a href="#">Link 5</a></li>
            </ul>
        </nav>
    </footer>
</body>
</html>
/home/projekty/zadanie-1-3$

```

Dlatego do dłuższych plików lepiej sprawdza się komenda **less**, która przełącza terminal w tryb czytania pliku. Możesz w nim używać m.in. strzałek w górę i w dół na klawiaturze, aby przewijać zawartość. Aby powrócić do linii komend, wciśnij klawisz **q**.

```

/home/projekty/zadanie-1-3
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Page title</title>
</head>
<body>
    <header>
        <h1>This is my page</h1>
        <nav>
            <ul>
                <li><a href="#">Link 1</a></li>
                <li><a href="#">Link 2</a></li>
                <li><a href="#">Link 3</a></li>
                <li><a href="#">Link 4</a></li>
                <li><a href="#">Link 5</a></li>
            </ul>
        </nav>
    </header>
    <section>
        <h2>Title of section 1</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Rep
ellat optio inventore deleniti sapiente perspiciatis tempora ab, officiis explic
index.html

```

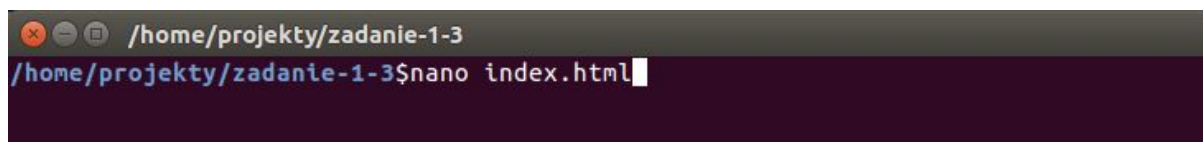
5.2. Edytory plików w oknie terminala

Znasz już edytor kodu, taki jak np. Sublime Text. Jest to jednak edytor z interfejsem graficznym, mimo że większość jego okna to tekst. Istnieją również edytory tekstu działające w oknie terminala.

Dwa z najpopularniejszych edytorów tego typu to **nano** i **vim**. W zależności od systemu operacyjnego możesz mieć jeden lub oba z nich. Omówimy sobie pokrótce ich podstawową obsługę, ale jeśli nie masz któregoś z nich - nie przejmuj się, wystarczy ten, który masz zainstalowany.

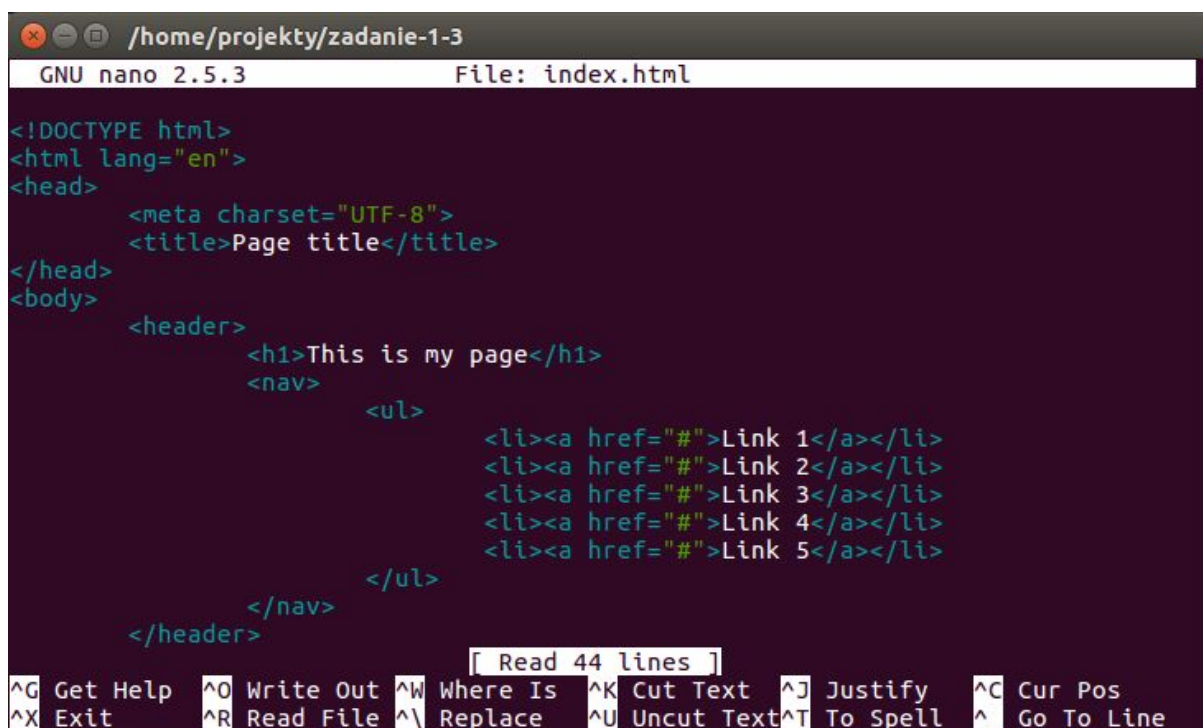
Edytor nano

Edytor **nano** uruchamiamy za pomocą komendy `nano`, podając po niej ścieżkę do edytowanego pliku (lub tworzonego, jeśli nie istnieje plik o takiej nazwie).



```
/home/projekty/zadanie-1-3
/home/projekty/zadanie-1-3$ nano index.html
```

Po zatwierdzeniu komendy enterem wyświetli się (o ile masz zainstalowany edytor **nano**) widok edytora:



```
GNU nano 2.5.3      File: index.html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Page title</title>
</head>
<body>
  <header>
    <h1>This is my page</h1>
    <nav>
      <ul>
        <li><a href="#">Link 1</a></li>
        <li><a href="#">Link 2</a></li>
        <li><a href="#">Link 3</a></li>
        <li><a href="#">Link 4</a></li>
        <li><a href="#">Link 5</a></li>
      </ul>
    </nav>
  </header>
</body>
</html>

[ Read 44 lines ]

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

Od razu po jego wyświetleniu możesz zacząć edytować plik - działają też klawisze strzałek, page up, page down, home, end, backspace, delete i enter. Możesz śmiało zmienić zawartość pliku.

U dołu ekranu widoczne są podpowiedzi dot. komend obsługi. W tym wypadku znak "^" oznacza klawisz **ctrl**, czyli np. aby wyjść z edytora musisz wcisnąć kombinację klawiszy **ctrl+x**. Jest to absolutnie najważniejsza komenda w **nano** - nie musisz nawet umieć zapisywać pliku, ponieważ przy próbie wyjścia **nano** zapyta Cię czy chcesz zapisać zmiany, a jeśli tak (klawisz **y**), zapyta o nazwę pliku (aby zapisać w dotychczasowym pliku, po prostu wciśnij **enter**).

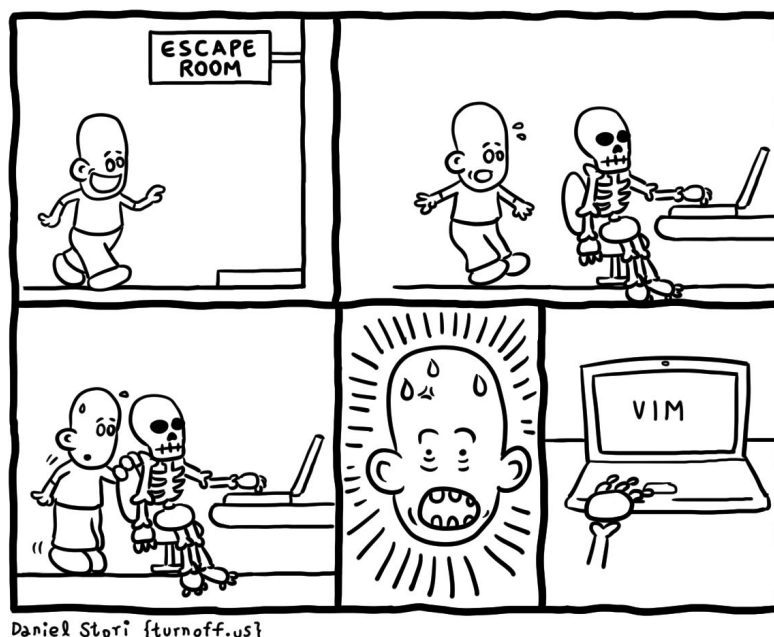


I to tyle - plik został zapisany z wprowadzonymi zmianami.

Edytor vim

Drugim edytorem, o którym chcemy wspomnieć, jest **vim**. Uruchamiamy go za pomocą komendy **vim** (lub **vi**, w zależności od wersji zainstalowanej w Twoim systemie).

Musisz wiedzieć, że jest to edytor o potężnych możliwościach, ale jednocześnie dość skomplikowany w obsłudze. Do tego stopnia, że tematem żartów stało się zadawanie studentom na pierwszych zajęciach zadania w postaci "uruchom vim i wyjdź z niego". ;)



Za chwilę jednak takie zadanie nie będzie już dla Ciebie straszne! Po uruchomieniu edytora **vim** pojawi się okno edycji, jednak w przeciwieństwie do poprzedniego edytora, nie możesz od razu w nim niczego pisać ani zmieniać. Aby było to możliwe, musisz wcisnąć klawisz i aby przejść do trybu edycji. Rozpoznasz go po napisie "INSERT" u dołu ekranu.


```
/home/projekty/zadanie-1-3
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>New page title</title>
</head>
<body>
  <header>
    <h1>This is my page</h1>
    <nav>
      <ul>
        <li><a href="#">Link 1</a></li>
        <li><a href="#">Link 2</a></li>
        <li><a href="#">Link 3</a></li>
        <li><a href="#">Link 4</a></li>
        <li><a href="#">Link 5</a></li>
      </ul>
    </nav>
  </header>
  <section>
    <h2>Title of section 1</h2>
  </section>
</body>
</html>
@
@
-- INSERT -- 1,1 Top
```

Aby wyjść z edytora musisz nacisnąć klawisz **esc**, a następnie wpisać (tekst pojawi się na dole ekranu):

- **:wq** - aby zapisać zmiany, lub
- **:q!** - aby wyjść bez zapisywania zmian,

a następnie zatwierdzić klawiszem **enter**.

Te komendy mogą się wydawać trudne do zapamiętania, ale wystarczy pamiętać, że:

- najpierw wpisujemy dwukropek,
- litera **w** oznacza "zapisz" (ang. "write"),
- litera **q** oznacza "wyjdź" (ang. "quit"),
- wykrzyknik oznacza "odrzuć wprowadzone zmiany".

```
/home/projekty/zadanie-1-3
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>New homepage title</title>
</head>
<body>
  <header>
    <h1>This is my page</h1>
    <nav>
      <ul>
        <li><a href="#">Link 1</a></li>
        <li><a href="#">Link 2</a></li>
        <li><a href="#">Link 3</a></li>
        <li><a href="#">Link 4</a></li>
        <li><a href="#">Link 5</a></li>
      </ul>
    </nav>
  </header>
  <section>
    <h2>Title of section 1</h2>
  </section>
</body>
</html>
```

Oczywiście, w razie potrzeby możesz też wpisać w Google "vim how to exit" i na pewno szybko znajdziesz odpowiedź.

Dla utrwalenia możesz obejrzeć animację całej procedury edycji oraz wyjścia z zapisem na animacji, którą znajdziesz pod [tym linkiem](#).

Uruchamianie standardowego edytora za pomocą linii komend

Jak wspomnieliśmy na początku tego rozdziału, edytowanie plików w oknie terminala jest przydatne sporadycznie - choć zdecydowanie warto znać podstawy, to tylko niektórzy informatycy preferują codzienną pracę np. w edytorze **vim**.

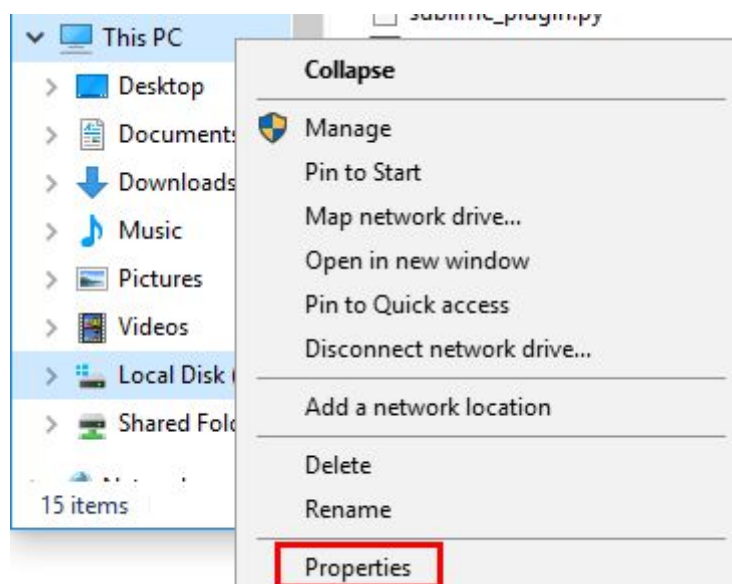
Tobie dużo częściej przyda się uruchamianie Twojego standardowego edytora kodu z poziomu terminala. Pokażemy to na przykładzie edytora **Sublime Text**, ale ten sposób powinien działać analogicznie dla innych edytorów kodu. W razie wątpliwości wpisz w wyszukiwarce Google nazwę edytora oraz hasło "open from command line".

Jeśli jeszcze nie masz zainstalowanego **Sublime Text**, a chcesz go używać, najpierw zainstaluj ten edytor pobierając instalator ze [strony producenta](#).

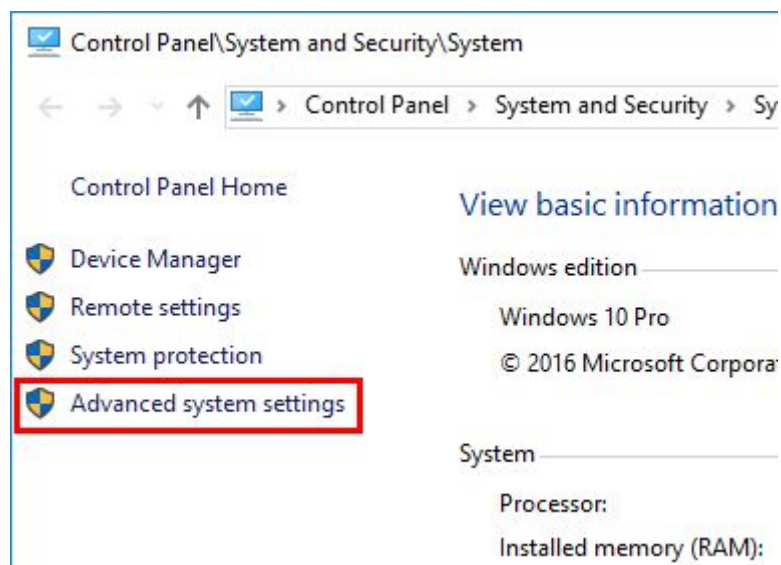
W systemach **Windows** oraz **macOS** otwieranie plików w **Sublime Text** wymaga jednorazowej konfiguracji. Poniżej znajdziesz instrukcje dla tych systemów operacyjnych.

Windows

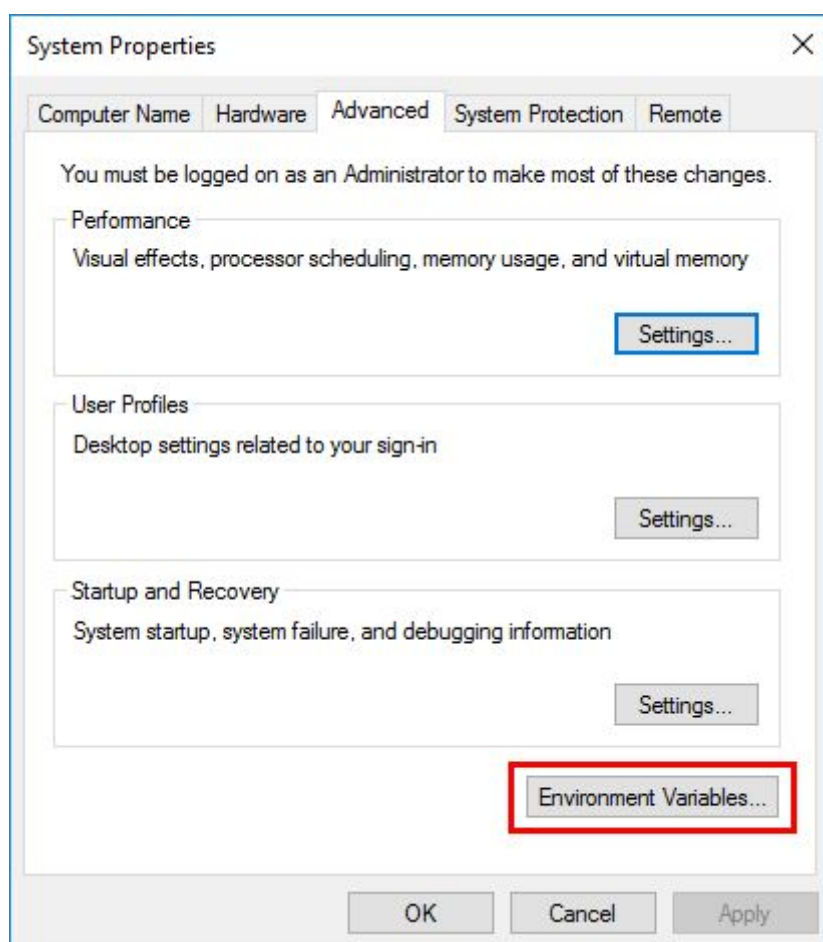
1. Sprawdź w terminalu (Git Bash) czy po wpisaniu komendy `subl` i zatwierdzeniu enterem otworzy się Sublime Text - jeśli tak, możesz pominąć tę instrukcję.
2. Upewnij się, że Sublime Text jest zainstalowany w katalogu "**C:\Program Files\Sublime Text 3**" (jeśli znajduje się w innym katalogu, zapisz lub zapamiętaj jego ścieżkę).
3. W eksploratorze plików kliknij PPM (prawym przyciskiem myszy) na "Ten komputer" w lewej kolumnie i wybierz **Właściwości**:



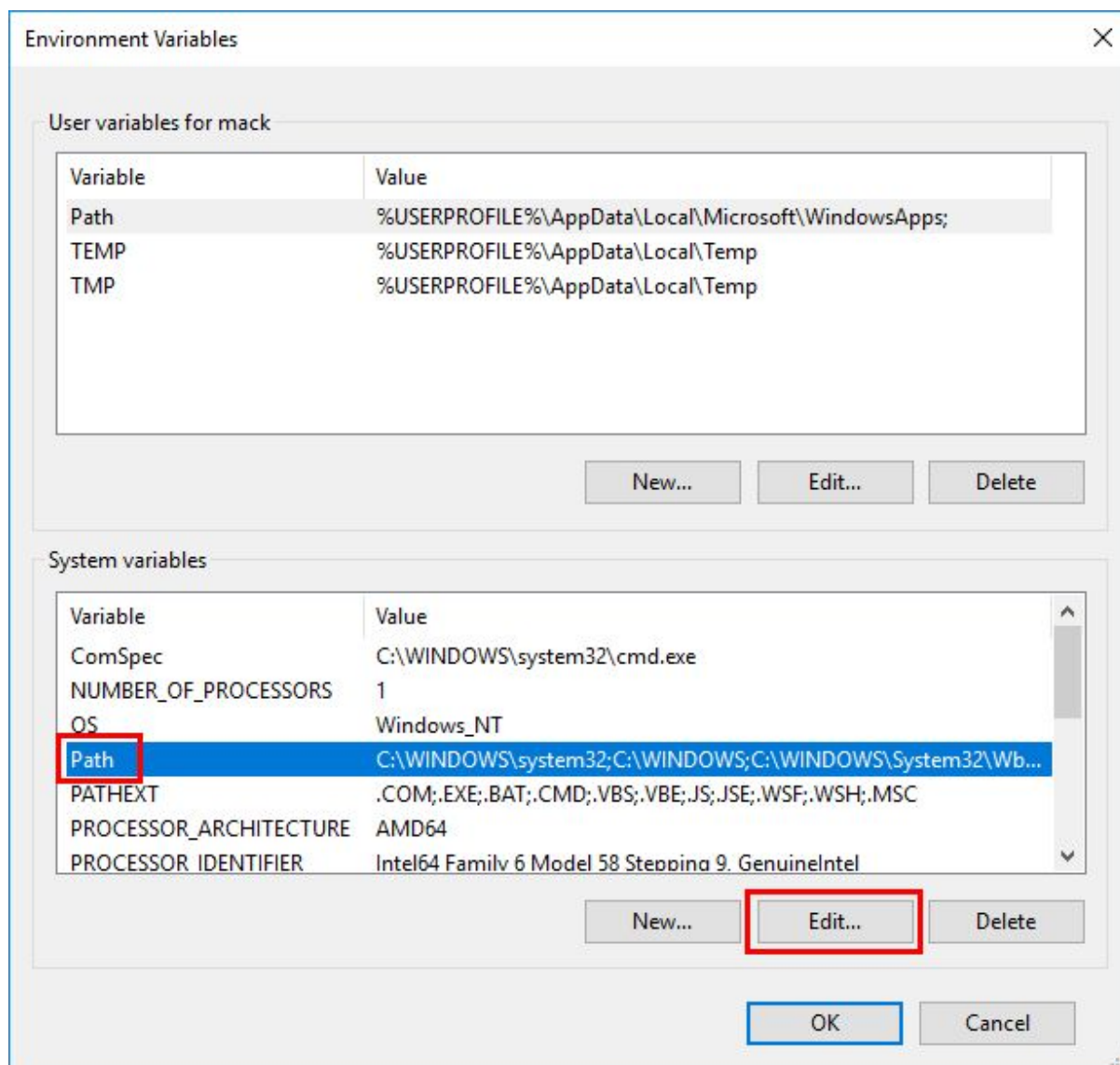
4. W przypadku **Windows 10**, z lewej kolumny wybierz "**Zaawansowane ustawienia systemu**" (w starszych systemach Windows zamiast tego kroku należy przejść na zakładkę "Zaawansowane"):



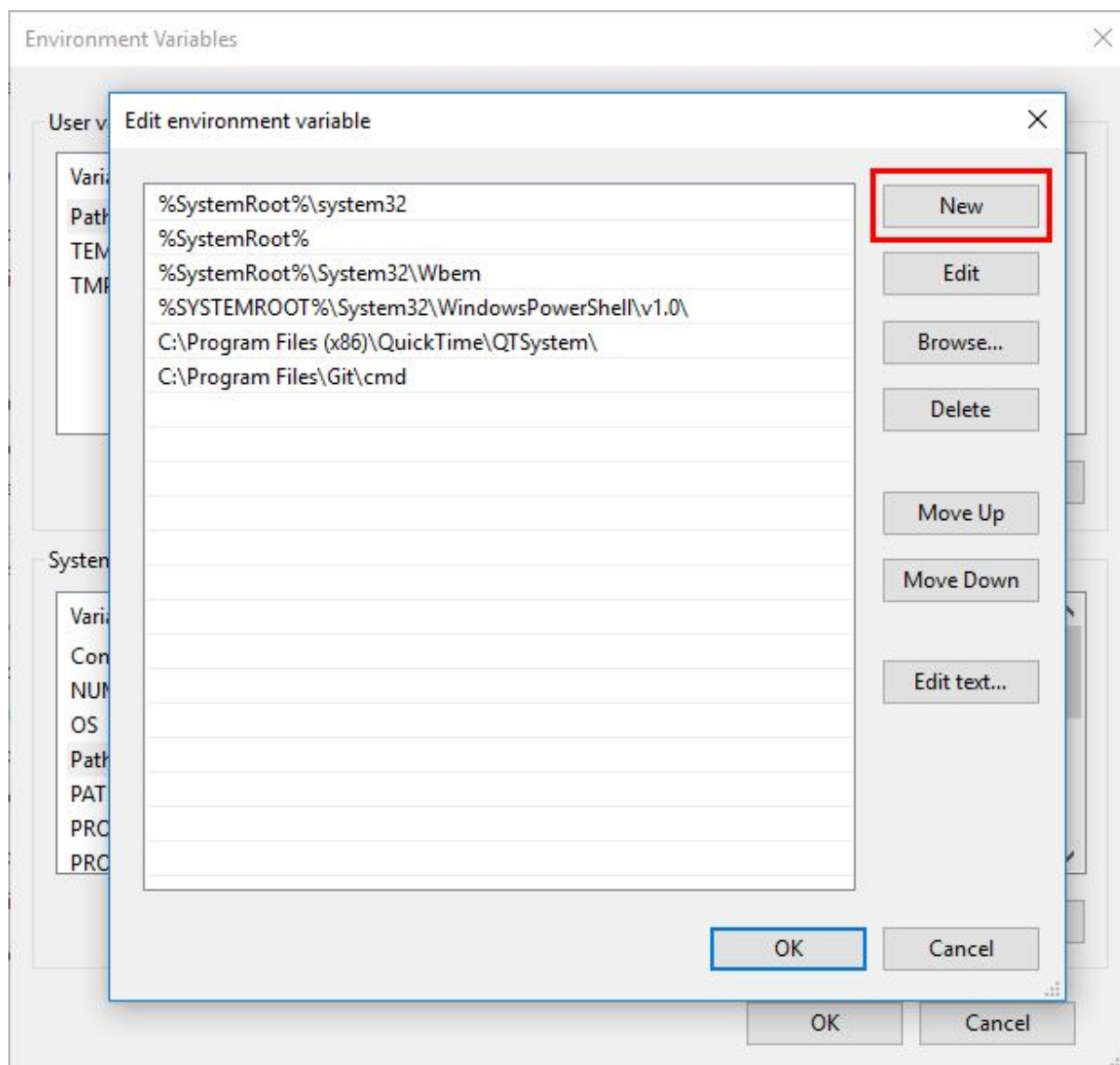
5. Kliknij w guzik "**Zmienne środowiskowe**":



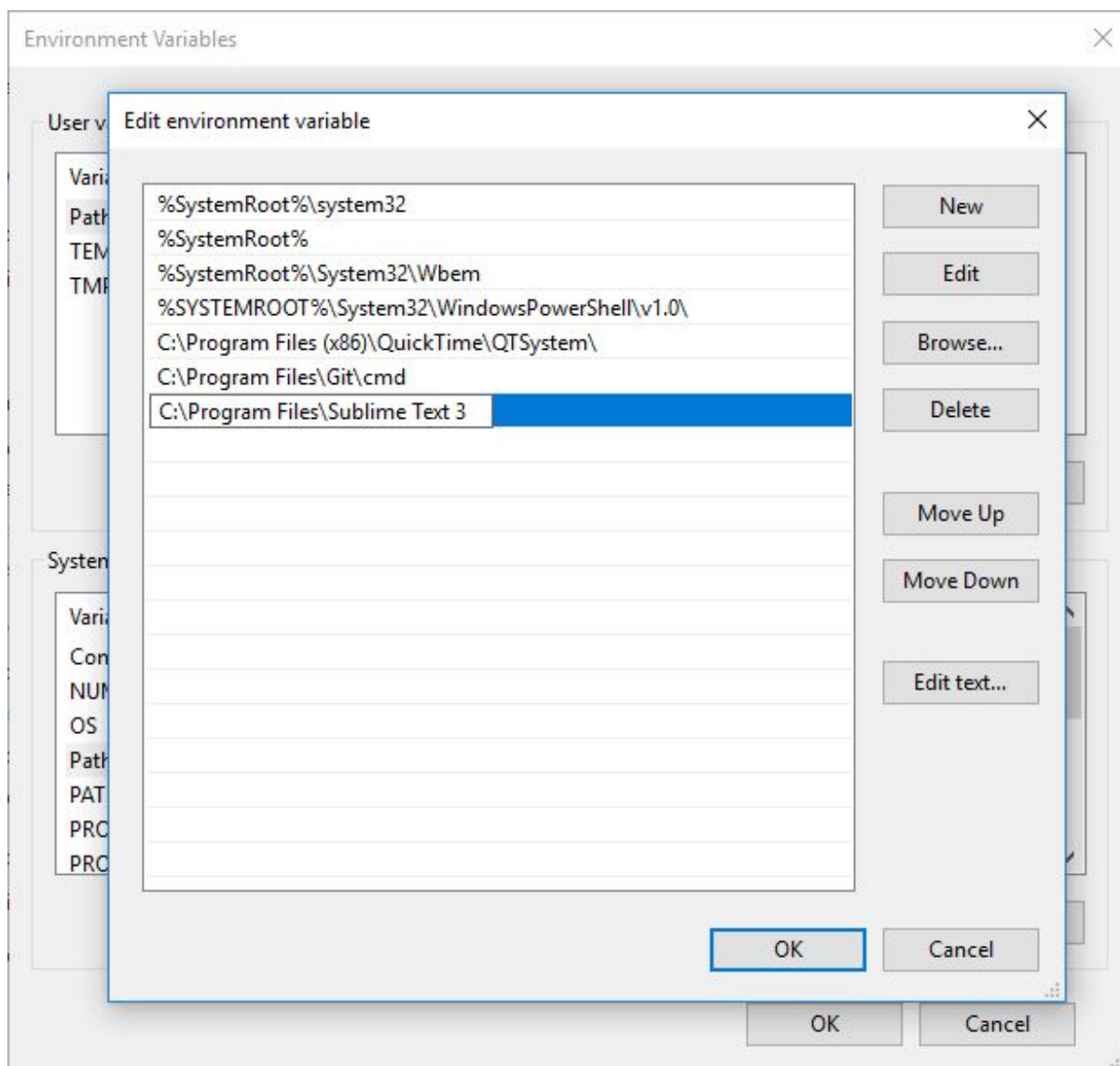
6. W sekcji "**Zmienne systemowe**" znajdź i zaznacz pozycję "**Path**" i kliknij w guzik "**Edytuj**" poniżej:



7. Kliknij w guzik "**Nowy**" (w starszych systemach Windows nie ma tego guzika, tylko edytuje się zmienną Path w jednym polu tekstowym, które ma już jakąś zawartość - należy przejść na koniec tej zawartości, wpisać średnik, a następnie kontynuować wykonywanie instrukcji od następnego punktu):



8. Wpisz ścieżkę instalacji Sublime Text (domyślnie "C:\Program Files\Sublime Text 3" bez cudzysłówów):



9. Kliknij "ok" w tym i wszystkich pozostałych oknach, które zostały otwarte w czasie tej procedury.
10. Od teraz w terminalu powinna działać komenda `subl`.

macOS

1. Sprawdź w terminalu czy po wpisaniu komendy `subl` i zatwierdzeniu enterem otworzy się Sublime Text - jeśli tak, możesz pominąć tę instrukcję.
2. Wpisz do terminala poniższą komendę (to jest jedna komenda, powinna być w jednej linii):

```
ln -s "/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl" /usr/local/bin/subl
```
3. Od teraz w terminalu powinna działać komenda `subl`.

Linux

Komenda `subl` powinna działać w terminalu, ale jeśli tak nie jest, wpisz w Google nazwę swojej dystrybucji linuxa oraz hasło "Sublime Text open from command line".

Podsumowanie

Teraz wystarczy, że w terminalu wpiszesz komendę `subl`, a po niej dowolną nazwę lub ścieżkę do pliku, podobnie jak przy wcześniejszych komendach terminala. Jeśli wskazany plik nie istnieje, edytor otworzy nowy plik - wystarczy go zapisać w edytorze (**ctrl+s**), aby został utworzony na dysku pod nazwą podaną przy jego otwieraniu.

Przetestuj zarówno otwieranie istniejących plików, jak i tworzenie nowych tym sposobem. Pamiętaj, że w obu wypadkach wystarczy komenda

`subl nazwa-pliku`, i możesz wykorzystywać wszystkie rodzaje ścieżek do plików, które wymieniliśmy przy komendzie `cd`.

6. Przydatne elementy składni

Terminal jest potężnym narzędziem i posiada wiele usprawnień, które pozwalają na usprawnienie korzystania z niego. W tym rozdziale poznasz kilka elementów składni, której używamy w komendach terminala.

6.1. Asterisk, czyli "gwiazdka"

Pierwszy z przydatnych elementów składni to znak *****, potocznie nazywany "gwiazdką". Oznacza on dowolny ciąg znaków, gdzie "dowolny" może oznaczać pusty. Zobaczmy to na przykładzie skopiowania wszystkich plików z rozszerzeniem `.html` z jednego projektu do drugiego:

```
/home/projekty
/home/projekty$ls zadanie-1-4
/home/projekty$cp zadanie-1-2/*.html zadanie-1-4
/home/projekty$ls zadanie-1-4
blog.html index.html
/home/projekty$
```

W ten sam sposób możemy użyć znaku ***** np. do zastąpienia nazwy katalogu w ścieżce do pliku (lub katalogu):

```
/home/projekty
/home/projekty$ls zadanie-*/sass
zadanie-1-1/sass:
_footer.scss _grid.scss _header.scss style.scss

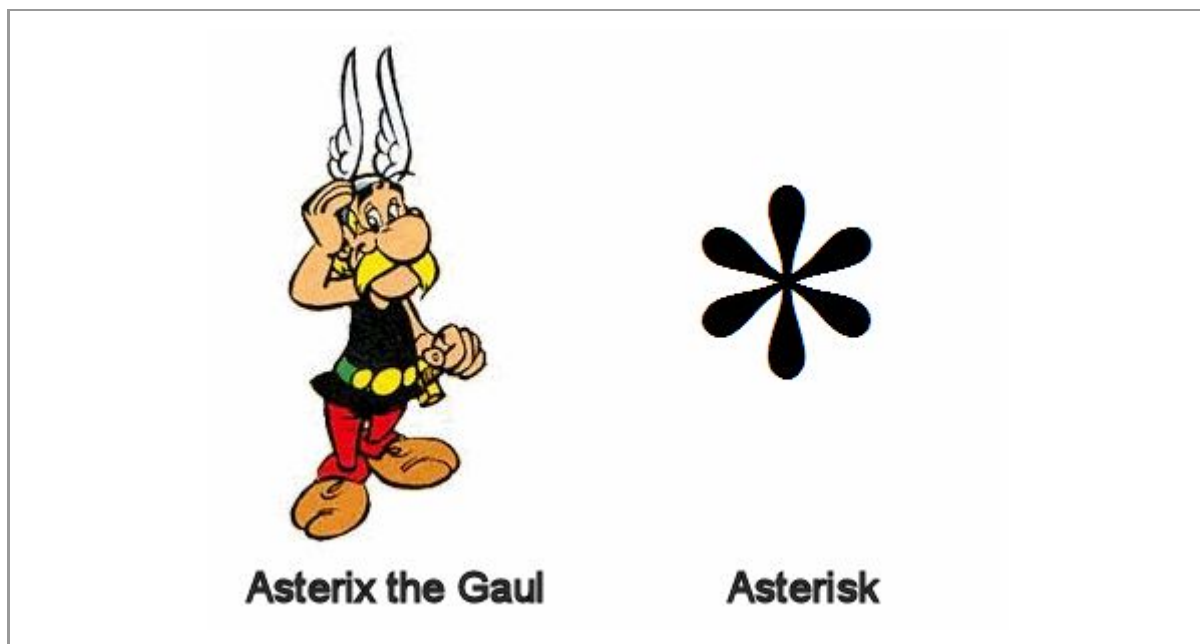
zadanie-1-3/sass:
_footer.scss _header.scss style.scss
/home/projekty$
```

Jak już wspomnieliśmy, może to być również pusty ciąg znaków. Ścieżka

`page*.html` zadziała zarówno dla pliku `page.html` jak i `page-2.html`.

Uwaga językowa

Znak ***** nosi nazwę "asterisk", nie "asterix", choć wiele osób popełnia ten błąd. Staraj się wykazać wyrozumiałością, i nie poprawiać innych za każdym razem, kiedy popełnią tę gafę. Od czasu do czasu można im jednak wysłać taki obrazek:



6.2. Łączenie kilku komend

Istnieje kilka sposobów na połączenie kilku komend w jedną, ale najczęściej używanym jest `&&`, który oznacza "jeśli powiodła się pierwsza komenda, wykonaj kolejną". Pokażemy ją na przykładzie skopiowania pliku `sass/style.scss` z innego projektu, w sytuacji kiedy nie mamy jeszcze katalogu `sass`:

```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$ls
blog.html index.html
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass/style.scss
cp: cannot create regular file 'sass/style.scss': No such file or directory
/home/projekty/zadanie-1-4$
```

Komunikat błędu informuje nas, że nie może skopiować pliku, ponieważ nie istnieje katalog `sass`. Co ciekawe, jeśli w drugim argumencie komendy `cp` podamy tylko nazwę katalogu `sass` zamiast ścieżki `sass/style.scss`, pojawi się inny problem:


```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$ls
blog.html index.html
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass/style.scss
cp: cannot create regular file 'sass/style.scss': No such file or directory
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass
/home/projekty/zadanie-1-4$cd sass
bash: cd: sass: Not a directory
/home/projekty/zadanie-1-4$ls
blog.html index.html sass
/home/projekty/zadanie-1-4$cat sass
body {
    background: green;
}
/home/projekty/zadanie-1-4$
```

Sama operacja kopiowania powiodła się, ale nie możemy wejść do katalogu sass - komenda `cd` zwróciła błąd "Not a directory". Skoro nie było katalogu sass, to komenda `cp ../zadanie-1-1/sass/style.scss sass` uznała, że drugi argument to nazwa jaką ma nadać plikowi po skopiowaniu. Czyli zamiast katalogu sass mamy plik o nazwie sass (bez rozszerzenia), który ma zawartość skopiowaną ze wskazanego pliku *style.scss*. Kompletnie nie o to nam chodziło, więc usuniemy ten plik i spróbujemy wykonać komendę z użyciem `&&`:

```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$ls
blog.html index.html
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass/style.scss
cp: cannot create regular file 'sass/style.scss': No such file or directory
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass
/home/projekty/zadanie-1-4$cd sass
bash: cd: sass: Not a directory
/home/projekty/zadanie-1-4$ls
blog.html index.html sass
/home/projekty/zadanie-1-4$cat sass
body {
    background: green;
}
/home/projekty/zadanie-1-4$rm sass
/home/projekty/zadanie-1-4$ls
blog.html index.html
/home/projekty/zadanie-1-4$mkdir sass && cp ../zadanie-1-1/sass/style.scss sass
/home/projekty/zadanie-1-4$ls
blog.html index.html sass
/home/projekty/zadanie-1-4$ls sass
style.scss
/home/projekty/zadanie-1-4$
```

Tym razem wszystko zadziałało poprawnie! Nasza komenda z `&&` wykonała najpierw `mkdir`, a potem `cp`. Czym w takim razie różni się komenda z `&&` od

wpisania tych komend osobno? Zobaczmy to przy ponownym wykonaniu tej samej komendy:

```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: green;
}
/home/projekty/zadanie-1-4$nano sass/style.scss
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: purple;
}
/home/projekty/zadanie-1-4$mkdir sass && cp ../zadanie-1-1/sass/style.scss sass
mkdir: cannot create directory 'sass': File exists
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: purple;
}
/home/projekty/zadanie-1-4$
```

Przejdźmy po kolei przez powyższy przykład:

1. najpierw wyświetliliśmy zawartość naszego pliku *style.scss* za pomocą komendy `cat`,
2. następnie zmieniliśmy kolor *green* na *purple*, korzystając z edytora **nano**,
3. ponownie pokazaliśmy zawartość tego pliku, aby pokazać na powyższej ilustracji, że zawartość się zmieniła,
4. wykonaliśmy naszą komendę z `&&`,
5. wyświetliliśmy ponownie plik *style.scss*, aby pokazać że nadal jest w nim kolor *purple*.

Skupmy się na czwartym punkcie. Najpierw została wykonana komenda `mkdir`, która zwróciła błąd "File exists", ponieważ już istnieje katalog o takiej nazwie. Skoro pierwsza komenda w tej linii zwróciła błąd, a `&&` oznacza "wykonaj drugą komendę, jeśli pierwsza wykonała się poprawnie", to komenda `cp` w ogóle nie została wykonana. Dzięki temu nasz plik *style.scss* nie został nadpisany poprzez skopiowanie go z innego projektu.

Zobaczmy jak wyglądałaby ta sama sytuacja, gdybyśmy wykonali `mkdir` i `cp` osobno:

```

/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: green;
}
/home/projekty/zadanie-1-4$nano sass/style.scss
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: purple;
}
/home/projekty/zadanie-1-4$mkdir sass && cp ../zadanie-1-1/sass/style.scss sass
mkdir: cannot create directory 'sass': File exists
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: purple;
}
/home/projekty/zadanie-1-4$mkdir sass
mkdir: cannot create directory 'sass': File exists
/home/projekty/zadanie-1-4$cp ../zadanie-1-1/sass/style.scss sass
/home/projekty/zadanie-1-4$cat sass/style.scss
body {
    background: green;
}
/home/projekty/zadanie-1-4$

```

Tym razem komenda `cp` została wykonana niezależnie, więc nadpisała plik `style.scss`, kopiując go ponownie z katalogu `../zadanie-1-1/sass`.

6.3. Zapisywanie komunikatu zwrotnego komendy do pliku

Kolejnym często używanym elementem składni są łączniki `>` i `>>`, który pozwala zapisać komunikat z dowolnej komendy do pliku.

Na początek założmy, że chcemy zapisać sobie listę wszystkich plików w naszym projekcie. Wykorzystamy do tego komendę `ls -R`, która wyświetla listę plików i katalogów nie tylko dla aktualnego katalogu roboczego terminala, ale też dla każdego listowanego katalogu:

```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$ls -R
.:
blog.html  contact.html  css  index.html  js  products.html  sass

./css:
style.css

./js:
jquery.js  script.js

./sass:
_footer.scss  _grid.scss  _header.scss  style.scss
/home/projekty/zadanie-1-4$ls -R > lista-plikow.txt
/home/projekty/zadanie-1-4$
```

Jak widzisz, komenda nie wyświetliła niczego kiedy dopisaliśmy do niej

`> lista-plikow.txt` - wynik komendy `ls -R` został za to zapisany do pliku *lista-plikow.txt*, który wcześniej nie istniał. Jego zawartość wygląda następująco:

```
/home/projekty/zadanie-1-4
/home/projekty/zadanie-1-4$cat lista-plikow.txt
.:
blog.html
contact.html
css
index.html
js
lista-plikow.txt
products.html
sass

./css:
style.css

./js:
jquery.js
script.js

./sass:
_footer.scss
_grid.scss
_header.scss
style.scss
/home/projekty/zadanie-1-4$
```

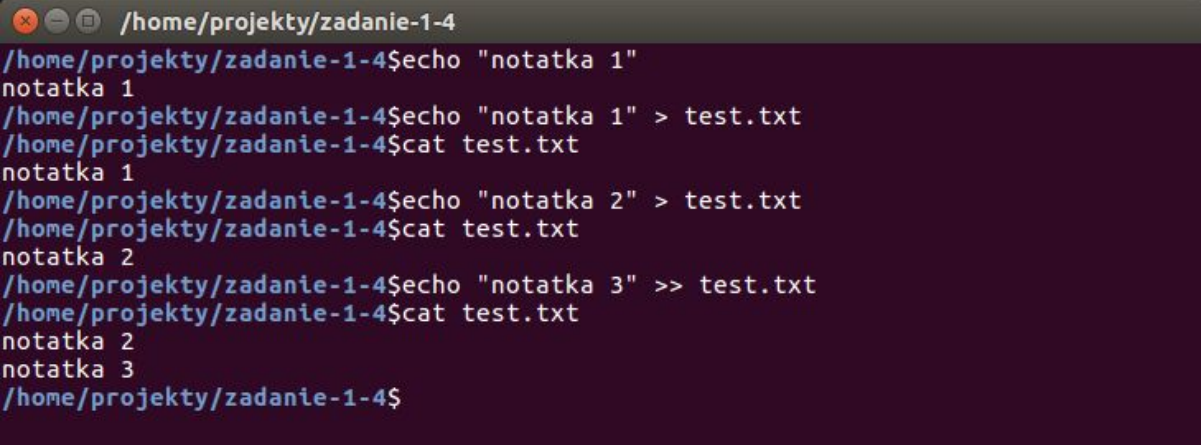
Kiedy będziemy chcieli zaktualizować listę w pliku **lista-plikow.txt**, wystarczy że ponownie wykonamy tę samą komendę.

Różnica pomiędzy `>` a `>>` polega na tym, że jeśli plik już istnieje, to pierwszy z nich zamieni zawartość pliku, a drugi doda nową zawartość na końcu pliku.

Przykładem użycia `>>` może być szybkie dodanie jednej linii na końcu pliku. Do tego będziemy też potrzebowali komendy `echo`, która po prostu wyświetla tekst

podany w tej komendzie. Zobaczmy na przykładzie czym będzie się różniło użycie

`>` od `>>`.

A terminal window titled '/home/projekty/zadanie-1-4' with a dark purple background. It shows a series of commands and their outputs. The first command 'echo "notatka 1"' outputs 'notatka 1'. The second command 'echo "notatka 1" > test.txt' outputs nothing, as the output is redirected to a file. The third command 'cat test.txt' outputs 'notatka 1'. The fourth command 'echo "notatka 2" > test.txt' outputs nothing. The fifth command 'cat test.txt' outputs 'notatka 2'. The sixth command 'echo "notatka 3" >> test.txt' outputs nothing. The seventh command 'cat test.txt' outputs 'notatka 2' followed by 'notatka 3' on a new line. The prompt returns to '/home/projekty/zadanie-1-4\$'.

Przetestuj używanie wymienionych elementów składni, aby utrwalić sobie ich działanie.

7. Obsługa terminala

Standardem wśród terminali, niezależnie od systemu operacyjnego, jest kilka podstawowych usprawnień użytkowania terminala. Jeśli do tej pory korzystanie z terminala było dla Ciebie uciążliwe, mamy nadzieję że za chwilę zmienisz zdanie.

7.1. Edycja wpisanej komendy

Możesz korzystać ze strzałek w lewo i prawo na klawiaturze, aby przechodzić kursorem w ramach wpisanej komendy. Dzięki temu możesz poprawić błąd na początku komendy bez wpisywania jej od początku.

7.2. Historia komend

W terminalu funkcjonuje historia wykonanych komend, którą możesz przeglądać za pomocą strzałek w górę i w dół. Po znalezieniu komendy, którą chcesz ponownie wykorzystać, możesz od razu ją wykonać wciskając **enter**, lub zacząć ją edytować (np. zmienić nazwę pliku).

Jeśli chcesz zakończyć przeglądanie historii, możesz wrócić do najnowszej pozycji (naciskasz strzałkę w dół aż pojawi się pusta linia), albo skasuj komendę którą widzisz i wciśnij **enter**.

Nie martw się edycją czy kasowaniem komendy z historii - to nie zmieni historii, tylko doda nową komendę na końcu historii, a oryginalna pozostanie bez zmiany.

W większości systemów do historii nie są dodawane identyczne komendy występująca jedna po drugiej, czyli jeśli trzy razy wykonasz `cd ..`, to w historii będzie występować tylko raz.

7.3. Kopiowanie i wklejanie

Być może zdarzyło Ci się już użyć kombinacji klawiszy **ctrl+v** do wklejenia komendy i zamiast oczekiwanego efektu w konsoli pojawiły się znaki **^V** (analogicznie **^C** po

wciśnięciu **ctrl+c**). Wynika to z tego, że w terminalu klawisze wciskane z **ctrl** mają specjalne znaczenie - np. **ctrl+c** przerywa działanie komendy, np. kompilatora **Sass** z flagą **--watch**.

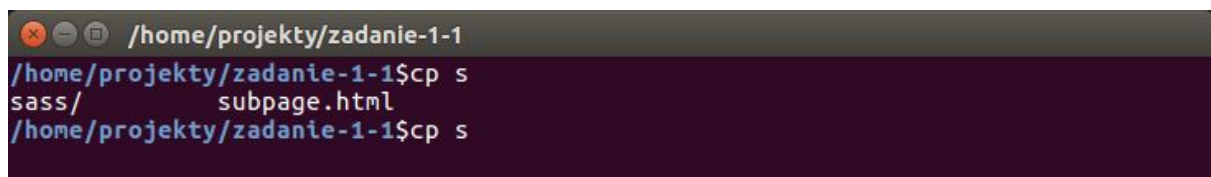
W większości systemów do kopiowania i wklejania można użyć menu kontekstowego (po kliknięciu prawym klawiszem myszy), ale wygodniej jest używać kombinacji **ctrl+shift+c** oraz **ctrl+shift+v**, które pozwalają na szybkie skopiowanie zaznaczonego tekstu oraz wklejenie komendy.

7.4. Uzupełnianie komend i ścieżek

Przy korzystaniu z terminala bardzo przydaje się klawisz **Tab**, który pozwala na uzupełnianie komend i nazw katalogów lub plików. Np. jeśli w katalogu projektu masz podkatalog o nazwie **sass**, wystarczy że wpiszesz **ls s** i wciśniesz **Tab**.

Możliwe są dwa rezultaty:

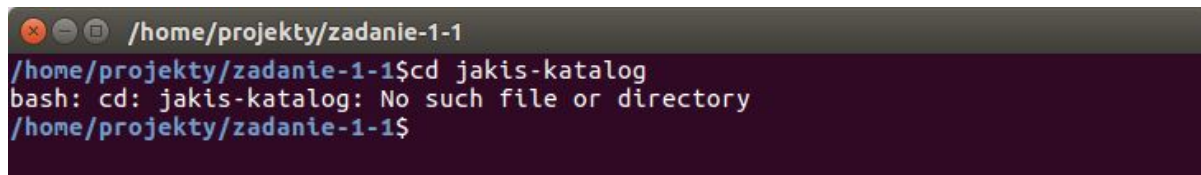
- jeśli wpisany fragment pasuje tylko do jednego pliku lub katalogu w tej lokalizacji (tylko jeden ma nazwę na "s"), w komendzie zostanie uzupełniona nazwa katalogu, np. **sass**,
- jeśli wpisany fragment nie pasuje do żadnego lub pasuje do wielu plików lub katalogów, nic się nie wyświetli w terminalu (może pojawić się dźwięk systemowy), ale po kolejnym wciśnięciu klawisza **Tab**:
 - jeśli wpisany fragment nie pasuje do żadnego pliku ani katalogu, nic nie zostanie wyświetlone w terminalu (może zostać odtworzony dźwięk systemowy),
 - jeśli wpisany fragment pasuje do wielu plików lub katalogów, zostanie wyświetlona ich lista, jak na ilustracji poniżej.



```
/home/projekty/zadanie-1-1$  
/home/projekty/zadanie-1-1$ scp s  
sass/ subpage.html  
/home/projekty/zadanie-1-1$
```


8. Czytanie komunikatów zwrotnych

Jedną z najważniejszych umiejętności w pracy z terminalem jest czytanie ze zrozumieniem. Jak już wspominaliśmy, w przypadku jakiegokolwiek problemu z wykonaniem komendy pojawi się komunikat błędu.

A screenshot of a terminal window with a dark background. The title bar at the top shows standard window controls and the path `/home/projekty/zadanie-1-1`. The terminal content shows the prompt `/home/projekty/zadanie-1-1$` followed by the command `cd jakis-katalog`. Below this, an error message is displayed: `bash: cd: jakis-katalog: No such file or directory`. The prompt `/home/projekty/zadanie-1-1$` appears again on the next line.

W niektórych przypadkach w komunikacie błędu może być nawet podpowiedź jak rozwiązać problem albo jakiej użyć komendy. Dlatego zawsze w takim przypadku warto:

1. Przeczytać i spróbować zrozumieć komunikat błędu.
2. Jeśli to nie rozwiązało błędu, skopiować treść błędu i wkleić do wyszukiwarki Google - często pośród kilku pierwszych wyników można znaleźć rozwiązanie problemu albo trafną wskazówkę.
3. Zastosować [metodę gumowej kaczuszki](#) (to nie żart, a sposób potwierdzony latami doświadczeń), który działa ponieważ kiedy próbujemy komuś (nawet gumowej kaczuszce) wytłumaczyć nasz problem, układamy nasze myśli w logiczne zdania, co pomaga zauważyć w czym leży problem.
4. Zapytać na czacie Kodilli w odpowiedniej grupie (np. [#help-web-tools](#) w przypadku problemów z obsługą terminala).
5. Zwrócić się do mentora - ale tylko w ostateczności, ponieważ umiejętność samodzielnego poradzenia sobie z problemem jest jedną z cech najbardziej cenionych przez pracodawców, więc warto ją rozwijać.

Pamiętaj też, że komunikaty wyświetlają się **nie tylko w przypadku błędów**. Np. po uruchomieniu kompilatora Sass z flagą `--watch` pojawi się informacja, że **możesz** zatrzymać ją za pomocą kombinacji klawiszy **ctrl+c**, ale to nie oznacza że musisz to zrobić. Jest to tylko przypomnienie sposobu na przerwanie działania, ale zwykle Twoim celem nie będzie zatrzymanie tego programu zaraz po jego uruchomieniu.

Dlatego właśnie liczy się nie tylko czytanie komunikatów i wykonywanie poleceń, ale **czytanie ze zrozumieniem.**

9. Zadanie treningowe

Aby przećwiczyć sobie wiedzę z tego poradnika, wykonaj poniższe zadanie. W razie problemów możesz zwrócić się o pomoc na czacie, w grupie [#help-web-tools](#).

Za pomocą terminala wykonaj poniższe operacje:

1. Otwórz terminal i przejdź do swojego katalogu z projektami.
2. Utwórz nowy katalog, np. *learning-cli* i przejdź do tego katalogu.
3. Stwórz puste pliki: *index.html*, *about.html* i *blog.html* i wyświetl listę plików.
4. Za pomocą jednej komendy (używając łączników):
 - a. stwórz katalog *sass*,
 - b. w nim stwórz plik *style.scss*,
 - c. ten plik od razu (od momentu stworzenia) ma mieć w sobie tekst:
`"// imports only in this file!"`.

10. Podsumowanie

Jeśli dopiero zaczynasz swoją przygodę z terminalem, warto dalej rozwijać swoją wiedzę, np. za pomocą darmowych lekcji interaktywnych na [Codecademy](#). Bardziej zaawansowane osoby może natomiast zainteresować kurs [LearnShell](#).

Najważniejszym i najlepszym sposobem nauki jest jednak używanie terminala na bieżąco w codziennej pracy. Nabranie wprawy w obsłudze terminala przyspieszy Twoją pracę i z pewnością będzie mile widziane kiedy dołączysz do zespołu programistów.