6 Tryb tekstowy

Edycja plików tekstowych

Jedną z podstawowych czynności wykonywanych na komputerze jest tworzenie i modyfikowanie plików tekstowych. Najczęściej używamy do tego edytorów graficznych. Niejeden raz zachodzi potrzeba by taką modyfikację przeprowadzić w konsoli - z pomocą przychodzą edytory tekstów przystosowane do takiej pracy.

Edytory konsolowe

Typowe edytory (po prawej komenda/kombinacja klawiszy zamykająca program):

`pico`	`Ctrl+X`
`nano`	`Ctrl+X`
`mcedit`	`F10`
`emacs`	`Ctrl+X Ctrl+C`
`vim`	`:q`

Edytory graficzne

* `kate`
* `gedit`
* `gvim`
* `geany`
* `leafpad`

Pracując z edytorem ważna jest dobra konfiguracja i znajomość skrótów – wpływa to znacznie na wydajność i swobodę pracy. Warto wiedzieć, że większość edytorów:

* udostępnia skróty klawiszowe
* często pozwala na modyfikację/dodawanie własnych skrótów
* pozwala na zmianę rozmiaru czcionki
* koloruje składnię w zależności od rozszerzenia otwartego pliku
* pozwala definiować własne zasady kolorowania składni
* pozwala na konwersję formatu zapisu pliku
* pozwala na zmianę wyglądu / definiowanie własnego stylu

Część edytorów to tak naprawdę małe środowiska programistyczne - przykładowo `geany` "umie" kompilować programy C, "umie" kompilować pliki latex, posiada code-folding, itp.

Warto pouzywać kilku edytorów i nauczyć się sprawnej obsługi któregoś z nich.

Tryb tekstowy

System operacyjny Linux oprócz trybu graficznego, zazwyczaj udostępnia również czysto tekstowy tryb pracy. Przełączanie między tzw. „wirtualnymi terminalami" dokonuje się za pomocą kombinacji `Ctrl + Alt + F#` gdzie `#` to numer terminala. Zwykle terminale `1-6` są tekstowe, natomiast `7` jest graficzny.

Oprócz tego taki tryb pracy uzyskujemy w momencie pracy zdalnej na innym serwerze (`ssh user@host`). Chociaż tutaj, jeśli serwer na to pozwala, możemy włączyć możliwość uruchamiania zdalnych programów w trybie graficznym logując się poprzez `ssh -X` bądź `ssh -Y user@host`.

Ciekawostka
Dla bash-a, mamy możliwość zmiany prompta (tekstu wyświetlanego automatycznie przed każdą komendą). Reguluje to zmienna `$PS1`:

```
echo $PS1
PS1='\u@\h \w\\$ '                          # "user@host cur_path$ "
PS1='\[\e[90m\][\u@\h \w]\$\[\e[00m\] '     # to samo z kolorem
PS1='[\W]: '                                # "[cur_dir]: "
```

Podstawy Vim

Plik otwieramy/tworzymy podając jego nazwę jako argument `vim`, np.
`$ vim plik.txt`

Bardzo ważna rzecz, która może być na początku frustrująca - `vim` pracuje w kilku trybach. Po otwarciu pliku **nie jesteśmy** w trybie edycji!

Domyślnie vim uruchamia się w **trybie normalnym** (normal mode) - w tym trybie możemy przemieszczać się po pliku i wykonywać komendy. Nie możemy od razu wpisywać do niego nowego tekstu, bo każda litera ma swoje specjalne znaczenie.

Główne komendy są poprzedzone dwukropkiem:

:w	zapisuje pliku na dysku
:q	zamyka program
:wq	zapisuje i zamyka program
:q!	zamyka program, nie zapisując żadnych zmian
:sav <nazwa>	zapisuje plik pod nową nazwa `<nazwa>`
:<N>	przeskakuje do linii numer <N>

Do **trybu edycji** (insert mode) możemy przejść na kilka sposobów:

* `i` (insert) — przechodzi do trybu edycji w miejscu położenia kursora

* `A` — przechodzi do trybu edycji na końcu aktualnej linii
* `R` (replace) — włącza tryb edycji z nadpisywaniem.

Powrót z trybu edycji do trybu normalnego - klawisz `ESC`.

Nawigowanie

Vim dysponuje szeroką gamą środków pozwalających na przesuwanie się po tekście. Najprostrzą metodą, dostępną w trybie normalnym i edycji, są strzałki, które przesuwają kursor o jedno pole w dowolnym kierunku. W trybie normalnym można jednak poruszać się szybciej:

`k`, ↑	przesuwa kursor do linii powyżej
`j`, ↓	przesuwa kursor do linii poniżej
`h`, ←	przesuwa kursor o jedną pozycję w lewo
`l`, →	przesuwa kursor o jedną pozycję w prawo
`^`	przesuwa kursor do pierwszego nie-białego znaku w linii. Białe znaki to niedrukowane elementy, takie jak spacja, tabulator czy nowa linia.
`e`	przesuwa kursor na koniec aktualnego lub następnego słowa
`w`	przesuwa kursor na początek następnego słowa
`b`	przesuwa kursor na początek aktualnego lub poprzedniego słowa (separatorem są zarówno znaki białe jak i znaki interpunkcyjne)
`E, W, B`	jak wyżej, tylko jako separatory traktuje jedynie białe znaki
`/wzorzec`	szuka wzorca w części tekstu po aktualnej pozycji kursora
`?wzorzec`	szuka wzorca w części tekstu przed aktualną pozycją kursora
`n`	powtórne szukanie "do przodu"
`N`	powtórne szukanie "do tyłu"

Prawie wszystkie komendy przedstawione powyżej (także strzałki) mogą zostać poprzedzone liczbą - co oznacza powtórzenie operacji tyle razy. Przykładowo `3w` przejdzie 3 słowa do przodu, a `10`↓ – 10 linii w dół.

Modyfikowanie fragmentów

Vim daje wygodne możliwości kopiowania, usuwania, itp. danego fragmentu tekstu:

`y`	rozpoczyna kopiowanie (yankowanie[1])
`d`	rozpoczyna usuwanie/wycinanie
`=`	koryguje wcięcia wierszy

[1] ang. yank — dosłownie można przetłumaczyć jako szarpnięcie

Po wybraniu funkcji, dokonujemy dowolną operację nawigującą. Po wykonaniu F`<ruch>`, funkcja F (kopiowanie/usuwanie/...) zostanie zaaplikowana do fragmentu tekstu nad którym kursor wykonał swój `<ruch>`.

Dla przykładu:

`d`↓	usuwa *dwie* linie: aktualną i po niej następującą.
`y20`↓	kopiuje 21 linii.
`d<END>`	usuwa wszystkie znaki od aktualnej pozycji kursora do końca linii.

Ponadto, FF aplikuje operację do aktualnej linii.

`dd`	kasuje (wycina) bieżącą linię
`yy`	kopiuje bieżącą linię

Wklejanie:
Operacje kopiowania i usuwania zapamiętują wybrany fragment w specjalnym buforze tekstu. Tekst z bufora można wkleić w dowolną inną pozycję w tekście.

`P, p`	wkleja zawartość bufora przed/po aktualnej pozycji kursora.

Jak inne komendy, `P` można poprzedzić liczbą, co spowoduje wielokrotne wklejenie bufora.

Cofanie operacji:

`u`	cofa ostatnią operację. Kolejne komendy `u` przechodzą dalej wstecz.
`Ctrl+R`	przechodzi w przód w czasie, przywracając operacje cofnięte za pomocą `u`.