6 Vim*

Jednym z najbardziej popularnych edytorów konsolowych jest `vim`. Nie jest to intuicyjny edytor, ale posiada wiele narzędzi do pracy z tekstem. Nie bez powodu jest on często wykorzystywany przez programistów i administratorów.

Vim pracuje w pięciu trybach, które można łatwo przełączać:

* Tryb normalny (normal mode) — vim wykonuje polecenia przypisane do pojedynczych klawiszy.
* Tryb edycji (insert mode) — tryb wpisywania nowego tekstu do pliku. Jedynie kilka klawiszy ma specjalne znaczenie.
* Tryb wizualny (visual mode) — tryb pozwalający zaznaczyć fragmenty tekstu i wykonywania operacji na nim.
* Tryb wiersza poleceń (command-line mode) — Podobnie jak polecenia powłoki, wyrażenia te mogą być bardzo specyficzne i skomplikowane.
* Tryb ex (ex-mode) — Tryb wykonywania skryptów

Podstawy

Zanim wyjaśnimy w szczegółach jak nawigować i edytować otwarty już plik, wyjaśnijmy wpierw jak plik otworzyć i zamknąć. Bez tych podsdtawowych operacji nie będziemy wstanie zrobić nic pożytecznego.

Najprościej otworzyć plik, podając jego nazwę jako argument to `vim`, np.
```
$ vim plik.txt
```

Jeśli jednak wywołaliśmy `vim` bez argumentów, można otworzyć plik poprzez wbudowany wiersz poleceń, wpisując
```
:e <nazwa>
```

Dwukropek na początku jest istotny i poprzedza on każdą komendę w vimie.

: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>`

Wprowadzanie tekstu

Domyślnie vim uruchamia się w trybie normalnym. Gdy otworzymy nowy plik, nie możemy od razu wpisywać do niego nowego tekstu, bo każda litera ma swoje specjalne znaczenie. Do trybu edycji możemy przejść naciskając jeden z klawiszy:

* `i` (insert) — przechodzi do trybu edycji w miejscu położenia kursora
* `a` (append) — przechodzi do trybu edycji na kolejnej pozycji za kursorem
* `I` — przechodzi do trybu edycji na początku aktualnej linii
* `A` — przechodzi do trybu edycji na końcu aktualnej linii
* `R` (replace) — włącza tryb edycji z nadpisywaniem. Każdy nowy znak zastępuje istniejący, znajdujący się pod kursorem.
* `s` (substitute) — usuwa pojedynczy znak i przechodzi do trybu edycji
* `S` — usuwa całą linię i przechodzi do trybu edycji
* `o` — tworzy pustą linię pod aktualną linią i przechodzi do trybu edycji,
* `O` — tworzy pustą linię nad aktualną linią i przechodzi do trybu edycji,
* `C` — usuwa linię od aktualnej pozycji kursora i przechodzi do trybu edycji

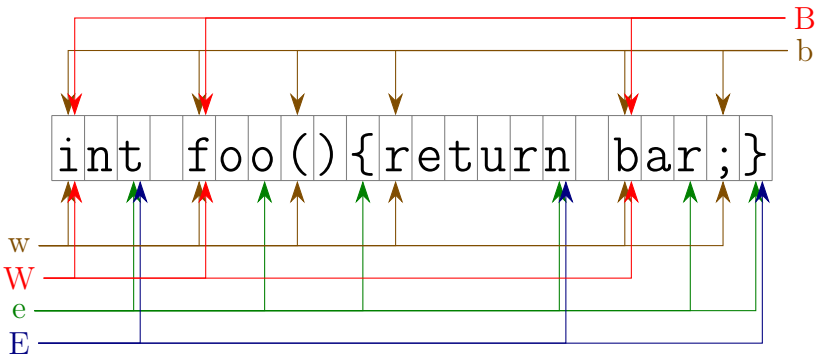
Jak widać, na wiele sposobów możemy zacząć coś pisać. Najważniejsza operacja to `i`, zaś pozostałe zachowania można uzyskać też w inny sposób, poprzez sekwencję komend. Z czasem jednak, nabierając wprawy i ćwicząc, te dodatkowe opcje mogą znacznie przyspieszyć pracę.

Będąc w trybie edycji, każdy zwykły klawisz (litera, cyfra, znaki interpunkcyjne, etc.) powodują dodanie odpowiedniego znaku do pliku. Aby opuścić tryb edycji i powrócić do normalnego wystarczy nacisnąć `ESC`. Można też nacisnąć `Ctrl+O` by wykonać jedną operację w trybie normalnym. Po jej wykonaniu, vim powraca do trybu edycji.

Nawigowanie

Vim dysponuje szeroką gamą środków pozwalających na przesuwanie się po tekście. Naturalnie, 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 na koniec linii.
`0`	przesuwa kursor na początek linii.
`^`	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 słowa. Separatorem słów traktowane są zarówno znaki białe jak i znaki interpunkcyjne. Jeśli kursor już jest na takim końcu, to przechodzi dalej, na koniec kolejnego słowa.
`E`	przesuwa kursor na koniec słowa, traktując jedynie białe znaki jako separatory.
`w`, `W`	przesuwa kursor na początek następnego słowa, traktując separatory jak powyżej
`b`, `B`	przesuwa kursor na początek aktualnego słowa. Jeśli kursor jest już na początku, przechodzi do poprzedniego słowa.
`)`	przesuwa kursor na początek kolejnego zdania. Zdania to ciąg znaków zakończony kropką i znakiem białym. Za koniec zdania przyjmuje się też jedną pustą linię, nawet jak poprzedzający go tekst nie kończy się kropką.
`(`	przesuwa kursor na początek aktualnego zdania. Jeśli już jest na początku, to przechodzi do poprzedniego.
`}`	przesuwa kursor na początek kolejnego paragrafu, tj. kolejnej pustej linii lub pierwszego znaku pliku.
`{`	przesuwa kursor na początek aktualnego paragrafu, tj. poprzedniej pustej linii lub ostatniego znaku pliku.
`%`	przesuwa kursor do pasującego nawiasowania. Działa także z komentarzami w języku C `/* */`. Ignorowane są nawiasy zawarte w cudzysłowiu.
`]}`	przesuwa kursor do zamknięcia aktualnego bloku `{...}`.
`[{`	przesuwa kursor do początku aktualnego bloku `{...}`.
`])`	przesuwa kursor do zamknięcia aktualnego bloku `(...)`.
`[(`	przesuwa kursor do początku aktualnego bloku `(...)`.
`g_`	przesuwa kursor do ostatniego nie białego znaku w linii
`gg`	przesuwa kursor do pierwszej linii
`|`	przesuwa kursor do pierwszej kolumny
`G`	przesuwa kursor do ostatniej linii
`f`C	przesuwa kursor do następnego wystąpienia znaku C w aktualnej linii
`F`C	przesuwa kursor wstecz do poprzedniego wystąpienia znaku C w aktualnej linii
`t`C	przesuwa kursor do znaku poprzedzającego następne wystąpienie znaku C w aktualnej linii
`T`C	przesuwa kursor wstecz do znaku następującego po znaku C w aktualnej linii
`/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. Taka liczba N oznacza powtórzenie operacji N razy. Tak więc `3w` przejdzie 3 słowa do przodu, a `10`↓ – 10 linii w dół.

W niektórych przypadkach liczba zmienia nieco znaczenie komendy. Np:

N`$`	przechodzi do końca N-tej linii poniżej. Wielokrotne naciśnięcie `$` nie daje takiego efektu.
N`g_`	przechodzi do ostatniego nie-białego znaku N-tej linii poniżej. Wielokrotne wpisanie `g_` nie daje takiego efektu.
N`%`	przechodzi do N% pliku. Np `50%` umieści kursor w połowie pliku.
N`gg`	przechodzi do N-tej linii w pliku.
N`|`	przechodzi do N-tej kolumny. Powtarzanie samego `|` nie daje takiego efektu.
N`t`C	przechodzi do znaku poprzedzającego N-te wystąpienie znaku C w danej linii, licząc od aktualnej pozycji kursora. Powtarzanie samego `t`C nie daje takiego efektu.

Tak szeroki zestaw narzędzi często pozwala na dotarcie do wybranego miejsca w pliku przy niewielkiej ilości operacji. Szczególnie gdy korygujemy kod programu, operacje które pozwalają np. przejść na początek czy koniec nazwy zmiennej bywają bardzo pomocne.

Pracując z większym kodem źródłowym, zdarza się, że musimy na moment przejść do innego miejsca w pliku (np. sprawdzić definicję struktury), by potem powrócić w to samo miejsce. W takich sytuacjach można sobie pomóc stosując zakładki (bookmarki), w nomenklaturze vim zwane po prostu `mark`.

`m`C ustala zakładkę C w danym pliku na aktualnej pozycji kursora. C może być dowolną literą. Małe litery przydzielane są każdemu pliku z osobna. Duże litery mają zasięg globalny, pośród wszystkich plików otwartych w vimie. Zakładki są zapamiętywane przez vim pomiędzy sesjami – zamknięcie i ponowne otwarcie programu zachowuje zapisane zakładki.

`'`C	przeskakuje do linii w której została zapisana zakładka C. Jeśli C jest dużą literą, zostanie otwarty plik w którym zapisano zakładkę.
`` ` ``C	przeskakuje do dokładnej pozycji w której została zapisana zakładka C.
`:marks`	wypisuje wszystkie zapisane zakładki
`` `. ``	przeskakuje do miejsca ostatniej zmiany w aktualnym pliku
`` `" ``	przeskakuje do miejsca ostatniej pozycji kursora w chwili gdy plik był poprzednio zamknięty
`` `0 ``	przeskakuje do ostatniego otwartego pliku
`` `` ``	przeskakuje do poprzedniej pozycji kursora przed skokiem

`''`	przeskakuje do poprzedniej linii w której był kursor przed skokiem
`:delmarks` C...	usuwa wszystkie wymienione zakładki.
`:delmarks!`	usuwa wszystkie zakładki przypisane do małych liter, dla aktualnego pliku.

Modyfikowanie fragmentów

Czasami coś trzeba zrobić z fragmentem tekstu: skopiować, usunąć, skorygować wcięcia, itd.

`y`	rozpoczyna yankowanie[1] (kopiowanie)
`d`	rozpoczyna usuwanie/wycinanie
`c`	rozpoczyna usuwanie/wycinanie, ale po wykonanej operacji przechodzi do trybu edycji
`=`	koryguje wcięcia wierszy

Po wybraniu funkcji, dokonujemy dowolną operację nawigującą spośród tych wymienionych wcześniej. Po wykonaniu F`<ruch>`, funkcja reprezentowana przez znak F zostanie zaaplikowana do fragmentu tekstu nad którym kursor wykonał swój `<ruch>`. Przesunięcia które z założenia operują na liniach aplikują funkcję do całych wierszy. Pozostałe aplikują funkcję dokładnie do znaków nad którymi kursor się przesunął.

Dla przykładu:

`d`↓	usuwa *dwie* linie: aktualną i po niej następującą.
`d20`↓	usuwa 21 linii.
`d$`	usuwa wszystkie znaki od aktualnej pozycji kursora do końca linii. Znak końca linii pozostaje nienaruszony.
`df`C	usuwa wszystkie znaki aż do wystąpienia znaku C (włącznie)
`'ad'b`	usuwa wszystkie linie pomiędzy zakładką a i b.

Ponadto, dla każdej funkcji reprezentowanej przez F istnieje kilka kombinacji specjalnych, takich jak:

FF	aplikuje F do aktualnej linii, wraz ze znakiem końca linii
F`iw`	aplikuje F do aktualnego słowa, bez białych znaków
F`aw`	aplikuje F do aktualnego słowa, wraz z wszystkimi białymi znakami je otaczającymi

Czasami jednak ciężko wybrać jednym ruchem cały fragment do którego chcielibyśmy zaaplikować daną funkcję. W takiej sytuacji, najwygodniejszą, a zarazem najbardziej jednoznaczną metodą jest użycie trybu wizualnego, uruchamianego poprzez `v`. W tym trybie

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

vim podświetla cały tekst od miejsca rozpoczęcia, do aktualnej pozycji kursora. Możemy wykonać dowolną liczbę ruchów by dokładnie wybrać fragment nas interesujący. Na końcu, gdy jesteśmy gotowi, naciskamy klawisz F odpowiadający interesującej nas funkcji. Spowoduje to zaaplikowanie F do zaznaczonego fragmentu i powrót do trybu normalnego.

Ponadto `V` pozwala wejść w tryb wizualny który zaznacza tylko pełne linie.

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

- `P` wkleja zawartość bufora przed aktualną pozycję kursora. Znak na którym aktualnie znajduje się kursor będzie bezpośrednio *po* wklejonym tekście.
- `p` wkleja zawartość bufora po aktualnej pozycji kursora. Znak na którym aktualnie znajduje się kursor będzie bezpośrednio *przed* wklejonym tekstem.

Tak jak z innymi komendami, `P` można poprzedzić numerem, co spowoduje wielokrotne wklejenie bufora.

Drobna korekta

Vim posiada kilka drobnych narzędzi ułatwiających dokonywanie korekt w istniejącym już tekście:

- `r`C zamienia znak pod kursorem na C.
- `x` natychmiast usuwa pojedynczy znak pod kursorem (poprzedzony liczbą, usuwa wiele znaków). Podobnie jak przy funkcji `d`, usunięty tekst zapisany jest w buforze.
- `.` powtarzą ostatnią dokonaną zmianę w tekście pod aktualną pozycją kursora. Zmianą może być nowy tekst wprowadzony w trybie edycji, ale też dowolna inna komenda z wymienionych powyżej, która zmieniała treść pliku. Przydatne gdy taka sama korekta musi być wprowadzona w kilku miejscach.
- `c` (change) usuwa tekst w taki sam sposób jak `d`, ale ponadto przechodzi do trybu edycji, pozwalając użytkownikowi na wprowadzenie nowej treści.

Czasami zdarza się, że przez pomyłkę wykonaliśmy inną operację niż zamierzaliśmy. Wtedy bardzo przydatnym jest możliwość cofnięcia ostatniej operacji.

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

Praca z kodem źródłowym

Vim jest szczególnie przygotowany do pracy z kodem źródłowym. Wiele języków wspieranych jest poprzez kolorowanie składni i automatyczne wcinanie linii.

Język rozpoznawany jest na podstawie rozszeżenia pliku; czasami na podstawie całej nazwy (e.g. `Makefile`). Listę wszystkich rozpoznawanych języków można na komputerach TCSu znaleźć pod:

* `/usr/share/vim/vim74/indent` — reguły wcinania
* `/usr/share/vim/vim74/syntax` — reguły kolorowania

Jak można zauważyć lista jest bardzo pokaźna. Gdyby jednak brakowało jakiegoś języka, bądź użytkownik pragnął zmienić lub wzbogacić istniejące już rozwiązania, odpowiednie pliki i katalogi należy tworzyć w katalogu domowym, pod: /.vim . Nie będziemy jednak tutaj omawiać tak zaawansowanych zagadnień.

Reguły kolorowania automatycznie dostosowują się do tego co użytkownik napisał. Reguły wcinania aplikowane są jedynie do aktualnej edytowanej linii. Gdy np. zaczniemy pisać w języku C:

```
if (n%i==0)
    █
```

i przejdziemy do następnej linii, vim automatycznie wetnie nową linię o 4 kolumny głębiej niż linia z `if`-em. Nasz kursor od razu pojawi się na pozycji oznaczonej █. Co więcej, jeśli będziemy pisać dalej, wszystkie kolejne linie będą tak wcięte (albo nawet bardziej), aż napiszemy znak `;`

Edytor automatycznie wcina tylko *aktualną* linię. Wszystkie wcześniejsze i późniejsze linie pozostają nienaruszone. Takie zachowanie zapobiega niechcianym zmianom gdy wpisujemy nowy fragment kodu gdzieś w środku pliku. Czasami jednak zdaża się, że w wyniku zmian należy skorygować wcięcia w jakimś fragmencie kodu. Do tego służy funkcja `=`, którą już wspomnieliśmy wcześniej mówiąc o modyfikowaniu fragmentów tekstu.

Tak więc na przykład `==` skoryguje wcięcie linii w której znajduje się kursor. Operacja ta zostanie zrobiona względem wcięcia linii poprzedzającej. Jeśli i ta powyżej jest wcięta niepoprawnie, to trzeba `=` zaaplikować do niej wcześniej. Najwygodniej przejść kursorem na początek zmian, i zaaplikować `=` od góry, np. wykonując jakiś daleki skok, lub przy pomocy trybu wizualnego.

Grep w edytorze Vim

W naszym ulubiony edytorze `vim` możemy używać polecenia `:grep` w trybie normalnym. Uwaga: vim ma swój własny styl wyrażeń regularnych, który nie jest ani BRE ani PCRE!

Wyrażenia regularne mogą pojawiać się też przy wyszukiwania (`/`) i podmienianiu (`:s`).

Poza tym jest wbudowana komenda:

```
:vim[grep][!] /{pattern}/[g][j] {file} ...,
```

która szuka wzorca `{pattern}` w plikach `{file}...`.

Użycie flagi `g` pozwala na zwrócenie więcej niż jednego dopasowania w linii. Jeśli nie użyjemy flagi `j` skaczemy do pierwszego wystąpienia wzorca `pattern`. Możemy znaleźć tylko pierwsze k wystąpień jeśli poprzedzimy komendę liczbą `k`. W następującym przykładzie znajdujemy tylko pierwsze wystąpienie.

```
:1vim[grep][!] /{pattern}/[g][j] {file} ...
```

Jeśli użyjemy `[!]`, to wszystkie zmianny w aktualnie modyfikowanym buforze (pliku) zostaną porzucone.

Komenda `:s` w edytorze `vim`

```
:[range]s[ubstitute]/{pattern}/{string}/[flags] [count]
```

Dla każdej linii w zawężeniu: `{range}` polecenie `:s[ubstitute]` zastępuje `{pattern}` przez `{string}`. Zasięg można definiować tak, jak w `sed`zie. Można podać pojedynczą linię, albo przedział linii. Często używany jest skrót dla `1,$` to `%`. Poza tym można używać bookmarków: `'c` lub `'C` w definiowaniu zawężenia.

W przypadku kiedy podany jest `[count]` zastępowanie następuje w `[count]` liniach, licząc od końca zawężenia. Przykładową flagą w `flags` jest `g`, która podobnie, jak w `sed`zie pozwala na zastępowanie wszystkich wystąpień w rozpatrywanej linii. Więcej o poleceniu można dowiedzieć się używając `vim`-owskiej komendy: `:help`.