Używaj VIM-a jak profesjonalista

Tim Ottinger

Tytuł oryginału: Use Vim Like A Pro
Tłumaczenie: Michał Bielawski
Korekta/skład: Adrian Bogdanowicz, Katarzyna Świderska, Marcin Karpezo
JakiLinux

Wersja: 1.0

Spis treści

1 Wstęp

Jest wiele edytorów tekstu. Niektóre z nich są bez wątpienia świetne i nic nie stoi na przeszkodzie, żebyś ich używał. Jednakże są pewne powody, dla których warto używać właśnie VIM-a. Nawet pomimo tego, że powody te nie są unikalnymi cechami tegoż edytora.

- Dzięki wzrostowi zainteresowania platformami Uniksowymi (głównie Linux i Mac OS X), VIM jest powszechnie używanym edytorem
- VIM potrzebuje niewielkich ilości pamięci RAM i mocy procesora. Nawet średniej klasy komputer jest w stanie udźwignąć wiele instancji edytora jednocześnie
- VIM ma wiele "supermocy", które sprawiają, że edycja plików przebiega dużo sprawniej
- VIM wytwarza wokół siebie aurę "geekowatości"
- VIM ma bardzo aktywną społeczność użytkowników i deweloperów. Zawsze ją miał.

1.1 Po co w ogóle pisać ten poradnik? (czyli podejście)

Są inne bardzo dobre tutoriale i Google na pewno pomoże Ci w znalezieniu ich. Zapewne kolejny tutorial do VIM-a nie jest artykułem pierwszej potrzeby, ale ten jest wyjątkowy – bo mój.
Przyjąłem nieco odmienne podejście niż inni autorzy. Uważam, że istnieje pewien sposób myślenia, który sprawia, że nauka VIM-a staje się dużo prostsza. Omawiam również nawyki, dzięki którym zaprzyjaźnisz się z tym edytorem. Szczerze mówiąc nie wiem, czy ktoś przedstawił to w ten sposób.
Długo układałem (i układałem i układałem...) ten tutorial tak, żeby materiał był w odpowiedniej kolejności. Cały czas stawiając sobie za cel napisanie tego poradnika tak, żeby czytelnik, po dobrnięciu do końca, umiał korzystać z VIM-a w jak najbardziej profesjonalny sposób. Prawdopodobnie lepiej niż niejeden ich bardziej doświadczony kolega.
Doradzam cierpliwość i poleganie na pamięci długoterminowej, ale jednocześnie uważam, że ten poradnik to jeden z najszybszych sposobów na usprawnienie Twojej pracy z VIM-em, jak również całkiem niezły na naukę od zera. Pisałem ten poradnik dla raczej niecierpliwego programisty.
Zdecydowałem, że nie chcę pisać i sprzedawać książki, ponieważ każdy powinien mieć możliwość używania VIM-a jak zawodowiec, w każdym miejscu i czasie, bez konieczności wnoszenia dodatkowych opłat.

1.2 Jak korzystać z tutoriala? (czyli instrukcja obsługi)

Traktuj każdy punkt poradnika jako osobną lekcję i poświęć chwilę czasu na wypróbowanie danych komend, zanim przejdziesz do następnej. Możesz poświęcić cały dzień na każdą lekcję, a może nawet kilka, jeżeli dany materiał okaże się wyjątkowo trudny do przyswojenia.
Postaraj się przerabiać po kilka lekcji tygodniowo. Nie spiesz się. Nie przeciążaj swojego mózgu ucząc się ciągle nowych rzeczy, bo zapomnisz o tym, czego uczyłeś się prędzej. Tutorial ten będzie istniał tak długo, jak będziesz tego potrzebował. Możesz złapać oddech.
Nie nauczysz się VIM-a nie używając go, więc powinieneś przygotować sobie jakieś pliki (najlepiej kod jakiegoś wolnodostępnego programu), na których będziesz pracował. Jeszcze lepiej, jeżeli używasz VIM-a w pracy. Pomocne może być również przerabianie tego "kursu" we dwie osoby, żebyście z partnerem mogli się nawzajem wspierać.

1.3 Co mogę z tym zrobić? (czyli licencja)

2 Tutorial (czyli część zasadnicza)

Nikt nie zna całego VIM-a. Jest tyle różnych poleceń, że wystarczy dla tysięcy istnień ludzkich, z których każde może używać VIM-a na swój własny sposób. Na szczęście nie musisz znać ich wszystkich. Wystarczy Ci tyle, ile potrzebujesz, żeby wykonać swoje zadanie.
Z tego również powodu, przez cały czas używania VIM-a będziesz poznawał nowe sztuczki. Sekret polega na tym, żeby nie poprzestawać na jednym konkretnym (kiepskim) sposobie wykonania danej czynności.
VIM ma uzupełnianie składni, historię operacji, skróty klawiszowe, personalizację ustawień klawiatury, makra i skrypty. Możesz go przemienić w **swój** edytor, przystosowany do **twojego** środowiska. Brzmi świetnie, ale pewnie ucieszysz się z faktu, że możesz być dużo wydajniejszy nawet bez dotykania tych zaawansowanych funkcji.
Jak mówi *Bram Moolenaar* (główny autor VIM-a), najlepszym sposobem na naukę VIM-a jest używanie go i zadawanie pytań. Ten poradnik pełen jest pytań, o których mogłeś nawet nie pomyśleć, żeby je zadać. To najbardziej wartościowa

rzecz, którą mogę Ci przekazać.
VIM ma wbudowany tutorial. Możesz go wypróbować, zwłaszcza jeśli nie spodoba Ci się mój. Żeby z niego skorzystać, wystarczy wywołać polecenie vimtutor z poziomu wiersza poleceń Twojego systemu. Jest on bardzo przystępny i raczej kompletny (w przeciwieństwie do mojego, który jest raczej przyjemny, ale na pewno nie kompletny).
Proponuję również rozważyć GVIM-a. Sprawi on, że będzie Ci się pracować dużo przyjemniej. Jeżeli masz pod ręką tylko VIM-a, to oczywiście nadal możesz korzystać z tego poradnika, ale GVIM wygląda dużo lepiej, pozwala na używanie myszy i ma menu oraz ikony dla tych z Was, którzy przywykli do takich rzeczy.

2.1 Tryby pracy

Oryginalny VI powstał w czasach, gdy czarno-zielone ekrany były szczytem osiągnięć techniki (zapytaj dziadka o terminale ASCII). Nie było wtedy tak wielu klawiszy modyfikujących (shift, alt, ctrl, super, fn), jak również urządzeń wskazujących. Przyjmijmy, że mieliśmy do dyspozycji tylko klawisze shift i ctrl (nawet jeśli to nieprawda).
Programując (tak jak i robiąc cokolwiek innego na komputerze) mieliśmy oczy cały czas skierowane na ekran, a dłonie położone na klawiaturze. VI sprawił, że pisanie programów odbywało się **szybko**, ponieważ VI jest trochę jak gra komputerowa, w której każde naciśnięcia klawisza sprawia, że coś się dzieje.
Jeżeli używasz vima i wciskanie klawiszy sprawia, że dzieją się albo rzeczy cudowne, albo straszne, to wiesz, że jesteś w trybie komend, który jest domyślnym trybem edytora. Komendy są przypisane do zwykłych klawiszy takich jak `o`, `y` czy `g`, zamiast kombinacji w stylu `Ctrl+Alt+Shift+Esc`.
VIM jest wyposażony w kombinacje klawiszy pozwalające wykorzystać specjalne moce, takie jak nawigacja między funkcjami w oddzielnych plikach, poprawianie formatowania całych list w samym środku dokumentu, uzupełnianie składni, skróty, szablony i inne takie, ale o tym później.
Musi istnieć również sposób na wprowadzanie tekstu, ale większość klawiszy ma już przypisane magiczne funkcje! Jedynym sensownym rozwiązaniem było stworzenie “trybu wprowadzania”, dzięki któremu wciśnięcie klawisza a sprawi, że w tekście pojawi się litera a. Zupełnie jak na maszynie do pisania (spytaj ojca co to). Nazywamy to “trybem wprowadzania”. Niewiele się w tym trybie dzieje, poza zwykły, starym, nudnym pisaniem. Powinieneś używać tego trybu tylko jeżeli chcesz coś napisać, bo fajne rzeczy dzieją się tylko w trybie zwykłym (komend).
Poznasz wiele wygodnych sposobów na przejście w tryb wprowadzania, ale na chwilę obecną powinieneś wiedzieć, że aby ten tryb opuścić i powrócić do trybu gry zręcznościowej, należy wcisnąć klawisz `Escape`. Gdy zrozumiesz, że w

VIM-ie istnieją dwa zasadnicze tryby pracy, twoja praca z nim przestanie wprawiać Cię w zakłopotanie i otworzy się przed Tobą droga do zostania guru tego edytora.

2.2 Poznaj schemat komend VIM-a

Przez większość czasu albo będziesz uzyskiwał rezultaty po pojedynczym wciśnięciu klawisza, albo będziesz wprowadzał komendy kończąc je komendą ruchu (często również powtarzając te same naciśnięcia: tzw. "podwójny skok"). Kiedy rozpoczniesz naukę innych rzeczy (rejestry, powtórzenia itd.) może Ci się wydać, że VIM nie jest spójny, ale tak nie jest. Schemat komend jest raczej spójny. Po prostu niektóre jego części są opcjonalne.

Rejestr (opcjonalnie, domyślnie rejestr ")
Powtórzenia (opcjonalnie): np. 13
Operacja: `y` (Od angielskiego *yank – szarpać*. Możemy to rozumieć jako *wyszarpnięcie* tekstu)
Przesunięcie (w zależności od operacji): `yy` (*podwójny skok*, żeby wykonać operację na aktualnym wierszu, konwencja używana w VI)

Komendy VIM-a działają zgodnie z powyższym schematem. Są komendy, które nie używają rejestrów i takie, które nie przyjmują polecenia przesunięcia, ale w większości przypadków tak to właśnie wygląda.
Rejestr to w zasadzie po prostu schowek (dla operacji typu kopiuj/wklej). Większość edytorów udostępnia tylko jeden taki schowek. W VIM-ie masz ich aż nadto, ale nie musisz ich wszystkich używać, więc się nimi nie przejmuj aż nie dotrzesz do lekcji o rejestrach.
Powtórzenia to liczba oznaczająca ile razy chcesz daną operację wykonać. Jeżeli nie podasz żadnej liczby, domyślną wartością jest 1.
Operacja to klawisz, który rozkazuje VIM-owi coś zrobić. Najczęściej są to zwykłe naciśnięcia klawiszy i większość z nim nie wymaga klawiszy modyfikujących jak `Shift`, `Ctrl` czy `Alt`.
Przesunięcie to polecenie oznaczające ruch kursora. Jest ich całkiem sporo, ponieważ jest wiele sposobów na poruszanie się. Nie panikuj jednak – na początku możesz używać klawiszy strzałek, jeżeli koniecznie musisz. Poświęciłem w tym poradniku całą sekcję na polecenia przesunięcia.
Spróbujmy wyjaśnić jak taki schemat działa. Jeżeli chcę skopiować 13 wierszy do mojego domyślnego rejestru, mogę pominąć rejestr, wpisać 13 jako liczbę powtórzeń, wcisnąć `y` jako komendę kopiowania i potem jeszcze raz `y` jako komendę ruchu (co oznacza aktualny wiersz). Takie polecenie (`13yy`) skopiuje 13 wierszy do domyślnego rejestru, poczynając od aktualnego. Jeżeli nacisnę teraz `p` (nie

precyzując rejestru, pomijając liczbę powtórzeń i wiedząc, że komenda `put` nie przyjmuje polecenia ruchu), to te wiersze zostaną wklejone zaraz za bieżącym.
Znając ten schemat będziesz mógł wpływać na wszystko, czego się nauczysz o VIM-ie. Naucz się wszystkich sposobów poruszania się, a będziesz wiedział jak wycinać, wklejać, poprawiać formatowanie, zmieniać poziom wcięcia i wiele innych rzeczy, które chciałbyś zrobić.

2.3 WYJDŹ!

Po tym jak uruchomisz sesję VIM, pewnie chciałbyś z niej kiedyś wyjść. Jest na to kilka sposobów. Spróbuj tych:

Polecenie	**Rezultat**
`:q`	Zamyka aktualne okno (lub edytor, jeżeli to jest jedyne), jeżeli nie ma w nim niezapisanych zmian
`:q!`	Zamyka aktualne okno, nawet jeżeli były w nim niezapisane zmiany
`:qa`	Zamyka wszystkie okna, jeżeli nie ma w nich niezapisanych zmian
`:qa!`	Zamyka wszystkie okna, nawet jeżeli były w nich niezapisane zmiany
`:wq` lub `:x` lub `ZZ`	Zapisuje zmiany i zamyka aktualne okno

Kiedy wciśniesz dwukropek, kursor przeniesie się do lewego dolnego rogu ekranu. Później dowiesz się dlaczego. Na razie wystarczy, że będziesz wiedział, że tak powinno się dziać, a te polecenia działają. Zauważ, że przed `ZZ` nie ma dwukropka.
Od tłumacza: Po poleceniach rozpoczynających się dwukropkiem należy wcisnąć klawisz `Enter`, inaczej nie odniosą one skutku.
Jeżeli nie możesz sobie poradzić z wyjściem z VIM-a, upewnij się, że Caps-Lock jest wyłączony i wciśnij Escape. Jeśli sprawi Ci to radość, możesz go nacisnąć kilkukrotnie. Jeżeli usłyszysz brzęczyk, będziesz wiedział, że wcisnąłeś go wystarczającą liczbę razy. Wtedy podane wyżej polecenia na pewno zadziałają.

2.4 Mnemonika

Nie wszystkie komendy są łatwe do zapamiętania. Twórcy VIM-a bardzo się starali, ale łatwo jest znaleźć więcej niż 26 czynności, które chciałbyś wykonywać korzystając z edytora, a poza tym tak się dziwnie składa, że niewiele jest słów zaczynających się na literę *q* i mających jakiś sens w programie do edycji tekstu. Jednakże wiele komend da się łatwo zapamiętać. Są to komendy do odnajdywania najbliższego znaku (ang. *Find* – `f`), przesunięcia o jedno słowo do tyłu (*Back one word* – `b`) lub do przodu (*Word* – `w`) itd.

Wiele poleceń jest mnemonicznych, pod warunkiem, że znasz żargon. Przykładowo: ponieważ zarówno *copy* (kopiuj) jak i *cut* (wytnij) zaczynają się na *c*, używamy slangowych *yank* (wyszarpnij – jako kopiuj), *delete* (usuń – jako wytnij) i *put* (połóż – jako wklej). `Y`, `D`, `P`. Może się to wydawać nieco dziwne, ale da się to zapamiętać. Pamiętaj, że z czasem przechodzi to do pamięci mięśniowej, ale autorzy VI i VIM-a starali się nie być całkowicie abstrakcyjni, gdy wszystko leżało w ich gestii. Czasem nie mieli jednak wielkiego wyboru.

2.5 Wywołanie

Gdy już wiesz jak wyjść z VIM-a, pora się nauczyć jak się do niego wejść. Zazwyczaj uruchamiamy VIM-a z wiersza poleceń, aczkolwiek całkiem możliwe, że masz do dyspozycji jakieś menu bądź inne sposoby na uruchamianie programów. Jest kilka sposobów na uruchomienie VIM-a:

Polecenie	**Rezultat**
`vim`	Uruchomienie z pustym oknem
`vim plik.txt`	Uruchomienie z załadowanym i gotowym do edycji plikiem plik.txt
`vim +N plik.txt`	Uruchomienie z załadowanym i gotowym do edycji plikiem plik.txt; rozpoczyna edycję w linii N
`vimtutor`	Uruchomienie w trybie samouczka – całkiem niezły pomysł
`vimdiff plik1.txt plik2.txt`	Uruchomienie w trybie porównywania i łączenia dwóch plików
`vim .`	Uruchomienie w trybie przeglądarki plików

Oprócz tych wymienionych powyżej, jest jeszcze więcej. Na początek te powinny jednak wystarczyć. Wypróbuj `vimdiff` i `vimtutor`. Niektóre ze sposobów uruchamiania nie zadziałają dopóki nie skonfigurujesz swojego *.vimrc*, ale o tym później.

Jeżeli zamiast `vim` napiszesz `gvim`, to uruchomi się wybajerzona wersja graficzna VIM-a (oczywiście zakładając, że jest zainstalowana). Ma ona kilka dodatkowych możliwości. Zapewne spodoba Ci się bardziej niż czysty VIM, bo jest ona jak VIM z polewą czekoladową. Wszystko, co mówimy tutaj o VIM-ie działa również w gVIM-ie, więc możesz bez obaw używać tego tutoriala.

Nie jesteś skazany na edycję tylko jednego pliku jednocześnie. Możesz urucho-

mić (g)VIM-a z wieloma nazwami plików jako argumentami. Jeżeli tak zrobisz, to jest kilka parametrów, dzięki którym możesz uzyskać ciekawe efekty. Oczywiście ciekawsze będą jak nauczysz się obsługiwać podzielone okna, więc pewnie będziesz chciał zajrzeć do nich później.

Parametr	**Rezultat**
`-o` (mała litera "o")	Otwiera wiele plików w oknach ułożonych w poziomie
`-O` (wielka litera "O")	Otwiera wiele plików w oknach ułożonych w pionie
`-p`	Otwiera wiele plików w osobnych kartach. (Osobiście nie cierpię tego)

2.6 Kontekst jest ważniejszy niż pozycja

Biedna duszyczka, używająca VIM-a po raz pierwszy będzie wciskać klawisze strzałek, poruszając się w kodzie strasznie nieefektywnie. Będzie przewijać ekran o całą stronę w górę i w dół (przy okazji: to odpowiednio `f̂` i `b̂`) i przeczesywać kod swoimi zmęczonymi oczami. Ta biedna osoba jest powolna, jeszcze nie uświadomiona i zapewne bierze VIM-a za kiepską wersję windowsowego notatnika, zamiast za potężne narzędzie jakim jest.
Przy okazji – klawisze strzałek nie zawsze działają w VIM-ie. Nie obarczaj jednak winą za to VIM-a! Tak naprawdę jest to sposób w jaki Twój terminal obsługuje te klawisze. Przez to VIM nie potrafi rozpoznać, że Twoje strzałki to strzałki. Jeżeli masz ten problem, będziesz musiał poszukać rozwiązania na własną rękę.
Żeby używać VIM-a porządnie, musisz się najpierw nauczyć porządnie poruszać. Nie przewijaj w poszukiwaniu tekstu. Nie używaj do tego swoich oczu. Nie wiem czy zauważyłeś, ale od jakiegoś czasu komputery robią to za nas. Masz tutaj garść najważniejszych poleceń ruchu. Najłatwiej jest się poruszać szukając:

Polecenie	**Rezultat**
`/`	Wyszukaj do przodu; zostaniesz zapytany o wzór
`?`	Wyszukaj do tyłu; zostaniesz zapytany o wzór
`n`	Powtórz ostatnie wyszukiwanie (tak jak kropka powtarza komendy)
`N`	Powtórz ostatnie wyszukiwanie, w przeciwnym kierunku
`tx`	Przesuń do (ang. *to*) litery x, zatrzymując się tuż przed nią – przydatne przy komendach usunięcia i zmiany
`fx`	Znajdź (ang. *find*) literę x, zatrzymując się na niej – również przydatne przy komendach usunięcia i zmiany

Jeżeli szukanie zbytnio Ci nie pomoże, rozważ skoki:

Polecenie	Rezultat
`gg`	Początek pliku
`G`	Koniec pliku
`0` (cyfra zero)	Początek wiersza
`w`	Jedno słowo do przodu
`W`	Następne słowo rozdzielone spacjami (pomija znaki specjalne)
`b`	Jedno słowo do tyłu
`B`	Poprzednie słowo rozdzielone spacjami (pomija znaki specjalne)
`e`	Koniec aktualnego słowa, lub następnego, jeżeli już jesteśmy na końcu
`E`	Koniec aktualnego słowa rozdzielonego spacjami (pomija znaki specjalne)

Poniższe komendy są bardzo przydatne, a przy okazji łatwe do zapamiętania, jeżeli znasz wyrażenia regularne:

Polecenie	Rezultat
`^`	Początek tekstu w danej linii - zdecydowanie lepsze niż korzystanie z klawiszy strzałek bądź komendy `h`
`$`	Koniec tekstu w danej linii. Zdecydowanie lepsze niż korzystanie z klawiszy strzałek bądź komendy `l`

Kilka ciekawszych komend:

Polecenie	Rezultat
`%`	Odpowiadający nawias, klamra itp.
`{`	Początek akapitu
`}`	Koniec akapitu (najbliższa pusta linia)
`(`	Początek zdania (zdania są oddzielone kropką i spacją)
`)`	Koniec zdania
`¨`	Miejsce ostatniej zmiany w aktualnym wierszu
`]]`	Następna funkcja/klasa (C, Java, C++, Python)
`[[`	Poprzednia funkcja/klasa (C, Java, C++, Python)

Ostatecznie, jeżeli nie jesteś w stanie wykorzystać wyszukiwania, skoków itd., nadal możesz korzystać z klawiatury, więc odłóż mysz na bok:

Polecenie	Rezultat
`h`	W lewo o jeden znak
`l`	W prawo o jeden znak
`k`	W górę o jeden wiersz
`j`	W dół o jeden wiersz
`^f`	W dół o jeden ekran
`^b`	W górę o jeden ekran

Dobrym pomysłem jest ustawienie opcji `hls` (*highlight search* – podświetlaj wyszukiwane) w pliku .vimrc. Niedługo się dowiesz jak to zrobić. Tymczasowo możesz wpisać `:set hls` i wcisnąć `Enter`.

2.7 Cytowanie metaznaków w wyrażeniach regularnych

Jeżeli nie wiesz czym są wyrażenia regularne, pomiń tę sekcję. Dla tych, który wiedzą i zdają sobie sprawę z tego, że komenda `/` przyjmuje wyrażenie regularne, a nie tylko zwykły tekst, będzie ona ważna. W pozostałych przypadkach wyda się pewnie kompletnie nie na miejscu i powinna zostać pominięta.
Powinieneś wiedzieć jak używać regeksów, bo kilka sztuczek z ich wykorzystaniem może sprawić, że będzie Ci się lepiej korzystało z Uniksa/Linuksa/Maca. Jest to zbyt rozległy temat, żeby go tutaj przedstawiać, ale możesz zerknąć na któryś ze świetnych tutoriali czy referencji dostępnych w sieci.
Najważniejszą sprawą do zapamiętania jest to, że VIM stawia na wygodę jeżeli chodzi o wyrażenia regularne. Ponieważ często szukasz, VIM zakłada, że wpisując `/+` chcesz znaleźć najbliższy znak +. W wyniku tego założenia wszystkie metaznaki muszą być poprzedzone wstecznym ukośnikiem (\). Czasem jest to problematyczne, ale jeżeli naprawdę chcesz znaleźć nawias poprzedzony plusem, jest to znaczne ułatwienie.

2.8 Nie panikuj, masz historię poleceń

Komendą cofającą zmiany jest `u` (ang. *undo*). Niezbyt trudne do zapamiętania, prawda? Jak już prędzej wspomniałem, wiele poleceń VIM-a jest mnemonicznych.
Komendą powtórzenie cofniętych zmian powinna być litera `r` (ang. *repeat*), ale jest już zajęta przez polecenie zamiany (*replace*); powiemy o tym nieco później. Pozostaje nam `Ctrl-R` (`^r`). Cóż... nie można mieć wszystkiego.
Jest jeszcze troche odnośnie cofania i powtarzania, ale na razie tyle powinno wystarczyć. Ciesz się, że już możesz cofać i potwarzać zmiany. VIM nie jest już taki

nieudolny i niewybaczający, jak mogło Ci się wydawać, chociaż wciąż nie musisz go lubić. Poczekaj tylko, aż do gry wejdzie pamięć mięśniowa.
Jeżeli narobisz naprawdę niezłego bałaganu, to po prostu wyłącz edytor bez zapisywania (`:qa!`).
Jeżeli bardzo się obawiasz o swoje pliki, albo jesteś po prostu bardzo ostrożny, powinieneś używać jakiegoś systemu kontroli wersji. Zalecam, żebyś tak czy siak zaczynał naukę na bezwartościowych plikach, w jakimś tymczasowym katalogu, ale gdy przejdziesz do prawdziwej pracy, nie powinieneś się obawiać wprowadzania zmian. Kontrola wersji jest dobrym zabezpieczeniem i może być z powodzeniem stosowana jako sposób na kopie bezpieczeństwa. Proponuję rozważyć Gita i Mercuriala – oba są jednocześnie proste i potężne.

2.9 Wielkie litery i ŚMIERĆ PRZEZ CAPSLOCK

W przypadku wielu poleceń `Shift` albo odwraca kierunek (`N` jest przeciwieństwem `n`; zobacz: następny punkt), albo całkowicie zmienia sposób działania. Poruszając się do przodu o jedno słowo (`w`) można wcisnąć `W`, żeby nadal przesuwać się o jedno słowo, ale traktując znaki specjalne jako jego część. To samo tyczy się przeciwnego kierunku – `b` i `B`.
Ponieważ jednak wielkie litery mogą mieć zupełnie inne znaczenie niż ich małe odpowiedniki, powinieneś uważać, żeby nie wcisnąć klawisza `CapsLock`! Czasem zdarzy się, że przez przypadek wciśniesz `CapsLock` i gdy będziesz chciał przejść do kolejnego wiersza (`j`), połączysz go z obecnym. Wiele innych niepożądanych zmian może się zdarzyć w takiej sytuacji, jeżeli Twoje palce będą szybko przemierzać klawiaturę w celu wykonania jakiegoś skomplikowanego ciągu komend. To okropne.
Kiedy spotka Cię ŚMIERĆ PRZEZ CAPSLOCK, powinieneś od razu go wyłączyć i wciskać `u` tak długo, aż cofniesz niechciane zmiany. Jeżeli uważasz jednak, że to gra nie warta świeczki, wpisz po prostu `:e!` i zatwierdź enterem, a VIM wczyta plik ponownie z dysku. Trochę to kłopotliwe, ale na pewno kiedyś Ci się to zdarzy, więc warto, żebyś o tym wiedział. Niektórzy z tego powodu wyłączają `CapsLock` na stałe.

2.10 Wstaw, nadpisz, zmień

W VIM-ie jest kilka różnych sposobów na rozpoczęcie wprowadzania tekstu, jak już zostało wspomniane w sekcji *Tryby pracy*.
Twoim standardowym trybem pracy powinien być tryb komend. Po wciśnięciu niektórych klawiszy przechodzisz w tryb wstawiania lub nadpisywania. W trybie wstawiania, wszystko co piszesz zostaje wstawione w miejsce kursora, a to co było za nim jest spychane do prawej bądź do kolejnego wiersza.

Z kolei w trybie nadpisywania, wszystkie naciśnięcia klawiszy są wpisywane podobnie jak w trybie wstawiania, z tym wyjątkiem, że tekst za kursorem zamiast zostać odepchniętym, jest zastępowany przez kolejne wpisywane znaki. Aby przejść do tego trybu, musisz wydać odpowiednie polecenie w trybie komend.
W trybie wykonywania (ang. *ex mode*) wpisujesz łańcuch znaków w pole u dołu ekranu. Przejdziemy do tego później, bo jest to potężna rzecz. Jest też nieco zawiła, więc poczekamy z nią trochę. Do tego trybu przechodzisz wciskając `:` w trybie komend.
Z każdego tych trybów możesz powrócić do trybu normalnego (komend) wciskając Escape. Bardzo przydatny klawisz.

Polecenie	**Rezultat**
`i`	Rozpocznij pisanie w miejscu kursora
`I`	Rozpocznij pisanie na początku wiersza. Zdecydowanie lepsze niż wciskanie ^, a następnie `i`
`a`	Rozpocznij pisanie za kursorem
`A`	Rozpocznij pisanie na końcu wiersza. Zdecydowanie lepsze niż wciskanie \$, a następnie `a`
`r`	Zamień pojedynczy znak w miejscu kursora
`R`	Przejdź do trybu nadpisywania, w którym niszczycielsko nadpisujesz wszystko, aż do naciśnięcia `Escape`
`s`	Zamień pojedynczy znak w miejscu kursora i przejdź do trybu wstawiania
`c`	Komenda zastąpienia. Wymaga komendy przesunięcia następującej po niej. Najczęściej używam `cw` i `cc`
`C`	Jak `c`, ale stosowane do całej linii
`o`	Wstawia pustą linię pod bieżącą
`O`	Wstawia pustą linię nad bieżącą
`:`	Przejdź w tryb wykonywania poleceń (dla zaawansowanych uczniów)
`!`	Przejdź w tryb filtrowania przy pomocy powłoki systemowej (dla bardzo zaawansowanych uczniów)

Zastanów się przez chwilę nad wagą komendy `c`. Jeżeli użyjesz jej w połączeniu z `t` lub `f`, staje się ona naprawdę potężnym narzędziem. Gdybyś miał kursor na literze *Z* na początku tego akapitu, mógłbyś wykonać komendę `ct.`, żeby przepisać całe zdanie, zachowując kropkę. To samo tyczy się innych, jak na przykład `d` do usuwania. Polecenia ruchu znacząco zwiększają możliwości komendy `c` i dlatego tak ważne jest, abyś je dobrze znał.

2.11 NIE POZOSTAWAJ W TRYBIE WSTAWIANIA

VIM jest stworzony do tego, żeby więcej się przemieszczać po tekście, niż edytować. Nagradza Cię za dostosowanie się do takiego trybu pracy – głównie tryb komend, z krótkimi przerwami na wprowadzanie.
Jeżeli będziesz próbował używać VIM-a jako kiepskiego zamiennika notatnika, tryby i nawigacja sprawią, że nigdy nie będziesz wydajny. Jak chcesz żeglować, musisz wsiąść do łódki. Jak chcesz efektywnie korzystać z VIM-a, musisz zaprzyjaźnić się z trybem komend.
Jeśli więc zatrzymujesz się, żeby nad czymś pomyśleć, wciśnij `Escape`. Jeżeli nie jesteś w trakcie pisania, powinieneś być w trybie komend. Jak chcesz przejść do następnej linijki czy w jakiekolwiek inne miejsce w pliku, wciśnij `Escape`. Jak chcesz odejść od klawiatury, wciśnij `Escape`. W przeciwnym wypadku, zaczniesz wprowadzać komendy i zanim zorientujesz się, że nie jesteś w trybie komend, stracisz możliwość sensownego używania `.` czy cofania. Natomiast kiedy chcesz w trakcie pisania wydać jedno polecenie możesz zrobić to przez wciśnięcie kombinacji `Ctrl+o`. Vim po wykonaniu go automatycznie powróci do trybu wstawiania.

2.12 Dobry .vimrc to szczęście

Kiedy uruchamiasz VIM-a, ten najpierw wczytuje Twoje ustawienia, zanim zacznie robić cokolwiek interesującego. Możesz edytować plik .vimrc w swoim katalogu domowym. Dotychczasowe lekcje dostarczyły Ci wystarczającej ilości informacji, żebyś był w stanie zmodyfikować .vimrc i dodać do niego poniższe komendy. Zabawa polega na tym, żebyś wprowadził komendy z lewej kolumny tabeli. Będziesz musiał je przepisać i poprawić ewentualne literówki. Kiedy już to zrobisz, zapiszesz plik i wyjdziesz z VIM-a, a gdy następnym razem go uruchomisz, Twoje ustawienia zostaną wczytane. Możesz edytować ten plik w dowolnej chwili.
W VIM-ie jest nieco wyrafinowanej magii. Często jednak zdarza się, że ta magia jest domyślnie wyłączona w dystrybucji. *Brakuje* uzupełniania składni wiersza poleceń, kolorowania składni, przeglądarki plików i wielu innych elementów, dopóki nie włączysz ich w swoim .vimrc.
Istnieje bardzo przyjemny przewodnik po ustawieniach VIM-a, a nawet interfejs do łatwego włączania i wyłączania poszczególnych opcji. Zdaje się, że nie jest to zbyt powszechna wiedza, bo dowiedziałem się o tym dopiero w tym roku. Chciałbym wiedzieć o tym od samego początku...

```
:options
:browse options
:browse
```

W oknie tym możesz przeglądać wszystkie dostępne opcje, a nawet zmieniać je. Możesz przeczytać krótki opis przypisany do każdej opcji, albo wcisnąć na nim `Enter` i przeczytać dokładniejszy opis. Jeżeli wciśniesz `Enter` na którejś z opcji, zostanie ona przełączona lub jej wartość zostanie zmieniona.
Każda z poniższych komend ma również wersję skróconą, ale te możesz poznać później. Na początek spróbuj wprowadzić pierwsze pięć komend (każdą w osobnej linii), zapisz plik i otwórz go ponownie, aby dodać kolejne. Zauważysz, że druga sesja będzie miała włączone kolorowanie składni i będzie wyświetlać dodatkowe informacje.

Komenda w .vimrc	**Znaczenie**
`syntax enable`	Włącza magię. Włączając w to przeglądarkę plików i kolorowanie składni
`set showmode`	Pokazuje kiedy jesteś w trybie wprowadzania/nadpisywania
`set showcmd`	Kiedy komenda się wykonuje, pokazuj ją w pasku stanu
`set wildmenu`	Włącza magię dla uzupełniania poleceń w trybie wykonywania
`set ruler`	Włącza *liniał* (informacje o stanie edytora) u dołu ekranu
`runtime ftplugin/man.vim`	Włączę obsługę stron man (wpisz `:Man <temat>`)
`set autoindent`	Dodawaj wcięcia inteligentnie, zamiast cały czas powracać do samego początku wiersza
`set expandtab`	Zamieniaj znak tabulacji na ciąg spacji
`set nowrap`	Nie zawijaj tekstu (sprawia, że okna wyglądają brzydko)
`set hlsearch`	Podświetlaj wszystkie wyniki wyszukiwania
`set showmatch`	Podświetlaj odpowiadające nawiasy, klamry itd.
`set ignorecase`	Ignoruj wielkość znaków przy wyszukiwaniu
`set smartcase`	Jeżeli wyszukiwana fraza zawiera wielkie litery, przestań ignorować wielkość znaków
`set path=., .., /usr/include/*, /usr/share/*`	Wskaż ścieżki, w których VIM ma szukać plików
`set spelllang=pl_pl`	Kiedy chcę sprawdzania pisowni, to chcę jej w języku polskim

2.13 Pomoc jest w drodze

VIM jest wyposażony w mechanizm pomocy. Powinieneś wiedzieć jak z niego korzystać.
Wpisz `:help`, a otworzy się nowe okno z treścią pomocy. Możesz się po nim poruszać przy pomocy klawiszy strzałek lub dowolnych komend ruchu, które poznałeś.
Zawsze możesz wprowadzać *dziwne* klawisze wciskając najpierw `^v`, a następnie ten klawisz. Jest to najbardziej przydatne właśnie w przypadku pomocy. Możesz wpisać `:help ^v^t` (`Ctrl+v`, `Ctrl+t`), by uzyskać pomoc na temat komendy `^t`. Zgodnie z konwencją, możesz uzyskać ten sam rezultat wpisując `:help CTRL-T`. To naprawdę ważna rzecz.
Większość dystrybucji VIM-a dostarcza program `vimtutor`. Program ten nauczy Cię jak używać VIM-a... przy pomocy VIM-a. Całkiem przydatne narzędzie (brawa dla autora!).
System pomocy jest wyposażony w linki. Jeżeli się na jakiś natkniesz, możesz przesunąć do niego kursor (tylko nie waż się korzystać ze strzałek!) i wcisnąć `^]` (`Ctrl+]`). Tak, to dziwna i dość abstrakcyjna kombinacja klawiszy. Nie tylko przeniesie Cię do miejsca wskazywanego przez link, ale też umieści Twoją aktualną pozycję na stosie, pozwalając Ci do niej wrócić. Jeżeli będziesz chciał to uczyć, po prostu wciśnij `^t` (tak, kolejny abstrakcyjny skrót), aby zdjąć aktualny link ze stosu i powrócić do poprzedniej pozycji. Komendy `^]` i `^t` nie są zbyt łatwe do zapamiętania, ale będziemy ich również używać do nawigacji w kodzie, więc nauczenie się ich nie będzie kompletną stratą energii.

2.14 Podwójny skok

Zgodnie z przyjętą konwencją, wciśnięcie tego samego klawisza polecenia podwójnie, spowoduje, że zadziała on dla całego bieżącego wiersza. Jeżeli chcesz skopiować daną linię, wciśnij `yy`. Jak chcesz ją usunąć, wciśnij `dd`. Przepisać na nowo – `cc`. Ta konwencja jest używana w całym VIM-ie, łącznie ze specjalnym przypadkiem pt. *zapisz i wyjdź* – `ZZ`. Wykonywanie operacji na całym wierszu jest dość częste, więc sensownym było sprawienie, żeby było to wygodne.

2.15 Pozbywanie się zbędnych rzeczy

Możesz się pozbyć znaku pod kursorem wciskając `x`. Jeśli chcesz go wcisnąć dziesięć razy, oszczędź sobie trudu i wpisz `10x`. Może to być przydatne, ale zapewne szybko się znudzisz albo liczeniem ile razy chcesz wykonać tę operację, albo przytrzymywaniem klawisza *x*. Ja bym się znudził.
Zdecydowanie wygodniejszą komendą do usuwania tekstu jest `d` (ang. *delete*). To

jedna z komend łatwych do zapamiętania.
`d` przyjmuje polecenie ruchu. Powinieneś poznać kilka z nich w trakcie poprzednich lekcji. Podstawowym założeniem jest, że będziesz usuwał tekst od znaku pod kursorem do jakiegoś innego miejsca w pliku.
Możesz usunąć bieżący wiersz, wciskając `dd`, albo linię aktualną wraz z następną, wciskając `dj`. W podobny sposób `d}` usunie tekst aż do końca akapitu, a `dG` – do końca pliku. Możesz zauważyć, że wszystkie komendy, które przyjmują polecenie ruchu działają w ten sposób (włącznie z `c`). Każda komenda ruchu, której się nauczysz, zwiększa Twoje umiejętności kopiowania, usuwania i poprawiania tekstu. Te dodatkowe moce są przyczyną, dla której powinieneś umieć się dobrze poruszać w VIM-ie.
`d` przyjmuje także licznik powtórzeń, więc jeśli wpiszesz `23dd`, usuniesz 23 wiersze, począwszy od bieżącego. To może się przydać.
I w końcu – mamy do dyspozycji rejestry. Rejestr w VIM-ie jest jak schowek, którego używałeś w innych programach. Kiedy usuwasz tekst jest on zapisywany do rejestru, żebyś mógł go potem wkleić w innym miejscu, wciskając `p` (*put* lub *paste*). Oczywiście możesz użyć innego rejestru niż domyślny. Definiuje się go przy pomocy cudzysłowu, po którym następuje nazwa rejestru (dowolna litera, z uwzględnieniem wielkości). Oznacza to, że możesz kopiować różne fragmenty tekstu przez *wyszarpywanie* (`y` – *yank*) lub usuwanie, i przechowywać je w różnych rejestrach. Potem wystarczy, że wklejając wybierzesz odpowiedni rejestr.
Wklejanie również używa rejestrów. Wygląda to wtedy tak: `"ap` (dla rejestru o nazwie *a*). Możesz nawet wkleić zawartość danego rejestru wielokrotnie, jeżeli użyjesz licznika powtórzeń (zerknij do lekcji o schemacie komend).
Tak naprawdę masz więcej rejestrów niż wymieniłem i możesz z nimi robić również inne rzeczy, ale tyle powinno wystarczyć na szybką lekcję o usuwaniu.

2.16 Używaj kropki

Każda akcja edycji jest nagrywana. Powiedzmy, że właśnie usunąłeś wiersz (wciskając `dd`). Edytor wie, że usunąłeś linię. Możesz powtórzyć akcję edycji (tj. usunąć kolejną linię) wciskając kropkę (`.`). Możesz nawet zastosować standardowy wzór i wskazać rejestr oraz liczbę powtórzeń i dopiero wtedy wcisnąć kropkę (ona pamięta komendę i przesunięcie). Jest to szczególnie przydatne, gdy ostatnio wykonaną komendą było `cw` – powtórzy operację zmiany na tekście pod kursorem.
Ponieważ kropka powtarza ostatnią edycję, jest to jeden z najważniejszy klawiszy. Powinieneś nauczyć się na nim polegać. Jest to jedna z najlepszych rzeczy, jakie daje Ci VIM.

2.17 Używaj gwiazdki

Gwiazdka (*asterysk*, `*`) jest świetną komendą, szczególnie jeżeli w swoim .vimrc masz włączoną opcję `hlsearch`. Przeskakuje ona do kolejnego wystąpienia słowa, na którym masz kursor. Jednocześnie podświetla wszystkie wystąpienia danego słowa w tekście.

Polecenie	Rezultat
*	Przeskocz do następnego wystąpienia aktualnego słowa w tekście
#	Przeskocz do poprzedniego wystąpienia aktualnego słowa w tekście

2.18 Rejestry

Większość edytorów udostępnia pojedynczy schowek. Kiedy kopiujesz bądź wycinasz tekst, tracisz to, co w nim do tej pory było. W rezultacie skaczesz po pliku kopiując z jednego miejsca w drugie. Jeżeli masz szczęście, możesz podzielić okno na dwie części i skakać między nimi. Dostarcza to sporo ciężkiej, ręcznej pracy i ćwiczeń koordynacji wzrokowo-ruchowej, żeby znaleźć tekst, zaznaczyć, wyciąć, znaleźć miejsce i wreszcie wkleić tekst w drodze do nirwany.
Najprawdopodobniej wystarczy kilka minut takiego przeklejania, żeby zrobiło Ci się niedobrze.
VIM daje nam inne rozwiązanie tego problemu. Niestety, ma też inną terminologię. Zamiast buforów mamy rejestry. Ten sam pomysł, inna nazwa (*bufor* oznacza coś zupełnie innego w VIM-ie). Możesz znaleźć miejsce, z którego chcesz kopiować, skopiować lub wyciąć tekst do różnych rejestrów, przeskoczyć do miejsca docelowego i wkleić zawartość tychże rejestrów. To wszystko wymaga tylko jednego większego ruchu.
Lista dostępnych rejestrów: (za dokumentacją VIM-a, dostępne przez `:help registers`)

Polecenie	Rezultat
"	Domyślny rejestr
a–z, A–Z	Małe i wielkie litery
+ i *	Rejestry zaznaczenia, powiązane ze schowkiem systemowym (pozwalają na kopiowanie tekstu z i do VIM-a)
_	Czarna dziura - używany jeżeli nie chcemy nadpisać zawartości domyślnego rejestru

*Od tłumacza: W systemach z rodziny Windows rejestry + i * oznaczają to samo – schowek systemowy, natomiast w systemach korzystających z X11 odpowiednio – schowek (wklejanie zazwyczaj przez `Ctrl+C`) i bieżące zaznaczenie (wklejanie środkowym przyciskiem myszy).*

Jest jeszcze kilka innych rejestrów specjalnego przeznaczenia, ale zostawiam je jako ciekawostkę, do doczytania w systemie pomocy. Są to takie rejestry jak rejestr *małych usunięć* czy rejestry numerowane. Naprawdę, nie musisz ich znać.
Żeby rozróżnić rejestr `y` od komendy `y`, VIM wymaga, żebyś poprzedził nazwę rejestru znakiem cudzysłowu. Tak więc `y` oznacza komendę, a `¨y` – rejestr. Jeżeli wpiszesz `¨y`, to VIM będzie oczekiwał na dokończenie wzoru, zapewne przez opcjonalną liczbę powtórzeń, komendę i polecenie ruchu (jeśli jest wymagane). Oto kilka przykładów o rosnącym stopniu skomplikowania:

Polecenie	**Rezultat**
`dd`	Wytnij bieżący wiersz do domyślnego rejestru
`¨add`	Wytnij bieżący wiersz do rejestru *a*
`¨y$`	Skopiuj tekst od kursora do końca wiersza, do rejestru *y*
`¨byy`	Skopiuj bieżący wiersz do rejestru *b*
`¨c24dd`	Wytnij 24 linie poczynając od bieżącej i wstaw je do rejestru *c*

Jestem przekonany, że `¨c24dd` wydaje się nieco szalone, ale pomyśl jak byś to zrobił korzystając z notatnika czy czegoś w tym rodzaju. Tutaj mamy tylko 6 klawiszy, tylko jeden z nich wciśnięty z shiftem, i nie musisz ani razu sięgać po mysz. Byłby to niesamowicie wydajny sposób na wycięcie 24 linii do nazwanego rejestru, zakładając że akurat wiesz ile linii chcesz skopiować. Bez tej wiedzy, praca włożona w policzenie wierszy bez problemu przeważy wygodę komendy. Przykład ten pozostaje więc w sferze akademickich rozważań, ale jednocześnie otwiera furtkę do wizualnego zaznaczania tekstu do skopiowania/wycięcia.

2.19 Zaznaczanie

VIM posiada dwa tryby zaznaczania tekstu: wizualny i niewizualny. Istnieje spore prawdopodobieństwo, że będziesz chciał skorzystać z trybu wizualnego do łatwiejszego kopiowania, więc poświęćmy mu chwilę.

Polecenie	**Rezultat**
`v`	Zaznaczanie znaków
`V`	Zaznaczanie wierszy
`^v` (`Ctrl+v`)	Zaznaczanie prostokątnego obszaru
`gv`	Ponowne zaznaczenie ostatnio zaznaczonego obszaru

Komenda do zaznaczania wizualnego to `v`. Możesz wcisnąć `v`, przejść do (lub wyszukać) końca interesującego Cię obszaru i wykonać polecenie kopiowania (`y`) bądź usuwania (`d`). Tak właściwie powinieneś to zrobić i powrócić. Wciśnij `v` ponownie, żeby opuścić tryb wizualny.
Czasem chcesz zaznaczać całe wiersze od razu. VIM korzysta w tym celu z komendy `V`. Działa ona dokładnie tak jak małe `v`, z tym że zawsze zaznacza całe linie. Oczywiście ponowne wciśnięcie `V` wyłącza tryb zaznaczania wierszy.

Innym razem może się zdarzyć tak, że będziesz chciał zaznaczyć prostokątny obszar zamiast całych wierszy czy ciągu znaków. Żeby wejść do trybu zaznaczania prostokątnego (blokowego), wciśnij `^v` (`Ctrl+v`). Zauważ, że gdy jesteś w trybie wprowadzania, `^v` ma zupełnie inne znaczenie. Niech Cię to nie zdziwi. Wklejanie prostokątnych obszarów to naprawdę świetna funkcja.
Inną fajną rzeczą jest to, że możesz wejść do trybu wizualnego przy pomocy `v`, zaznaczyć kilka znaków, a potem przełączyć się na zaznaczanie wierszy wciskając `V` albo nawet `^v`, żeby zaznaczyć prostokątny obszar.
Po tym jak opuścisz tryb wizualny, obszar, który zaznaczyłeś, nie jest już podświetlony. System pomocy VIM-a podpowiada, że możemy powrócić do poprzedniego trybu zaznaczania z ostatnio zaznaczonym obszarem wydając polecenie gv. Bawiłem się tym trochę i muszę powiedzieć, że jest to całkiem przydatne.
Zaznaczałem cały plik (`ggVG`) i kopiowałem go do systemowego schowka (`"+y`). Potem przełączałem się do edytora bloga i wciskałem standardowe `^a^v`, żeby nadpisać zawartość wpisu tekstem z VIM-a. Teraz mogę oszczędzić kilka uderzeń w klawisze korzystając z `gv` zamiast `ggVG` (po tym jak już wydałem to polecenie).
Zaznaczony obszar staje się kontekstem dla innych komend, więc możesz zrobić dużo więcej niż tylko proste kopiuj-wklej. Możesz wcisnąć `r`, a potem inny znak – na przykład `X`, żeby zamienić wszystkie znaki w zaznaczeniu na `X`. Możesz użyć tego obszaru do wywoływania na nim poleceń (o tym jeszcze nie mówiliśmy). Jest naprawdę sporo możliwości, ale zakończymy w tym miejscu lekcję o zaznaczaniu.
Skoro już i tak to czytasz, to chciałbym przypomnieć, żebyś unikał śmierci przez `CapsLock` i nigdy nie zostawał w trybie wprowadzania bez potrzeby. Nie należy to co prawda do materiału tej lekcji, ale po prostu musiałeś to usłyszeć ponownie.

2.20 Uzupełnianie

Nie bój się używać długich nazw i wyrazów, bo VIM ma uzupełnianie. Od razu uprzedzam, że to nie jest *intellisense*, ale spokojnie sobie poradzi z kończeniem słów za Ciebie. Wpisz wystarczająco dużo znaków, żeby słowo było rozpoznawalne i (bez wychodzenia z trybu wprowadzania) wciśnij `^n`. Jeżeli słowo, którego szukasz znajduje się w którymkolwiek z aktualnie otwartych plików (lub buforów), to VIM zaproponuje Ci najlepsze dopasowanie. Jeżeli to nie będzie słowo, które chciałeś wpisać, po prostu wciskaj `^n` tak długo, aż znajdziesz pasujące słowo albo skończą Ci się dostępne możliwości. Możesz też wcisnąć `^p`, aby powrócić do poprzedniej podpowiedzi.

Polecenie	Rezultat
^n	W trybie wprowadzania – znajdź następne dopasowanie
^p	W trybie wprowadzania – znajdź poprzednie dopasowanie

W nowszych wersjach GVIM-a (graficznego [*i konsolowego VIM-a od wersji 7.0 – dop. tłum.*]) pojawi się lista, z której będziesz mógł wybrać słowo albo wpisując tak dużo liter, aż będzie to jedyne dopasowanie, albo korzystając z klawiszy strzałek *[albo ^n i ^p – dop. tłum.]*.
Dostępny jest również mechanizm *uzupełniania całych wierszy*. Możesz wcisnąć ^x^l (Ctrl+x, Ctrl+l – mała litera *L*), aby przejść do specjalnego trybu uzupełniania. Dostępne opcje przeglądasz – podobnie jak w przypadku pojedynczych wyrazów – korzystając z ^n i ^p oraz klawiszy strzałek (jeżeli Twój VIM je obsługuje). Ponownie, jeżeli korzystasz z graficznego GVIM-a, pojawi się lista wyboru [*i ponownie – do konsolowego VIM-a od wersji 7.0 również się to odnosi – dop. tłum.*]. Bywa, że jest to bardziej przydatne niż kopiowanie w starym stylu.
Istnieje również mniej znany mechanizm uzupełniania nazw plików, dostępny przy pomocy ^x^f. Zazwyczaj z niego nie korzystam, więc potrzebuję ściągi takiej jak ten tutorial w tych rzadkich przypadkach. Mając pod ręką tryb przeglądania plików i wildmenu zazwyczaj nie potrzebuję uzupełniania nazw plików.

Polecenie	Rezultat
^x^l	W trybie wprowadzania – uzupełnianie wierszy
^n	Następne dopasowanie
^n	Poprzednie dopasowanie

Kiedy trafisz już na odpowiednie dopasowanie, po prostu kontynuuj pisanie. Dowolny klawisz inny niż klawisze wyboru (↑/↓/^n/^p) zostanie zaakceptowany jako nowy tekst, zarówno w trybie normalnym, jak i wprowadzania. Jest to nieco nieintuicyjne, ponieważ jesteś przyzwyczajony do zatwierdzania wyboru enterem lub tabulatorem.
Oprócz wymienionych, jest jeszcze kilka specjalnych komend dostępnych tylko w trybie wprowadzania.
Ponieważ jesteś wyposażony w uzupełnianie słów i wierszy, nie masz powodu, żeby używać krótkich i tajemniczych nazw zmiennych. Długie nazwy, które mają sens, są naprawdę wykonalne i wcale nie powoduja znużenia.

2.21 Zawsze miej tekst przed oczami

Są komendy, do przesuwania bieżącej linii. Programistom VIM-a zapewne zaczynało już brakować liter, więc przypięli je do klawisza z.

Polecenie	Rezultat
`zt`	Ustaw bieżący wiersz u góry ekranu
`zz`	Ustaw bieżący wiersz na środku ekranu
`zb`	Ustaw bieżący wiersz na dole ekranu

Możesz również ułatwić sobie zerkanie na kod referencyjny, używając podzielonych okien (ang. *split windows*). Załóżmy, że piszesz kod w pliku code.cpp i chcesz na chwilę zajrzeć do code.h.

Polecenie	Rezultat
`:split code.h<enter>`	Dzieli okno w poziomie i ładuje plik code.h w nowym oknie
`:vsplit code.h<enter>`	Dzieli okno w pionie i ładuje plik code.h w nowym oknie

Kiedy już podzielisz swoje okna, powinieneś wiedzieć jak się między nimi poruszać. Poniżej znajdziesz krótką listę komend (wszystkie podpięte pod `^w`), które Ci w tym pomogą. Zawsze możesz zamknąć okno (nawet podzielone) wykorzystując jedną z metod przedstawionych w rozdziale *WYJDŹ* (`:qq`, `ZZ` itd.).

Polecenie	Rezultat
`j` lub ↓	Przejdź do okna poniżej
`k` lub ↑	Przejdź do okna powyżej
`h` lub ←	Przejdź do okna po lewej
`l` lub →	Przejdź do okna po prawej
`c`	Zamknij bieżące okno
`o`	Zamknij wszystkie okna, poza bieżącym

Wszystkie powyższe komendy muszą zostać poprzedzone kombinacją `^c`. Zajrzyj do `:help CTRL-W`, żeby uzyskać więcej informacji na temat korzystania z okien.

2.22 Przeglądarka plików

VIM pozwala Ci na zarządzanie plikami i katalogami. Spróbuj. Jest dostępny odpowiedni rozdział w systemie pomocy, a więcej informacji możesz uzyskać wciskając i [*gdy już jesteś w trybie przeglądarki – dop. tłum.*]. VIM-owa przeglądarka plików to taki *Midnight Commander dla ubogich* albo też całkiem niezły zamiennik dla Eksploratora Windows. Jest całkiem poręczny i mocno polecany. Do działania wymagane jest ustawienie `syntax enable` w pliku .vimrc.

Polecenie	Rezultat
o	Otwórz plik w poziomo podzielonym oknie
v	Otwórz plik w pionowo podzielonym oknie
i	Przełącza między różnymi trybami wyświetlania listy plików (lista, szczegóły, drzewo)
s	Sortuj po kolumnie pod kursorem
r	Sortuj w odwrotnej kolejności
D	Usuń plik
d	Utwórz nowy katalog
enter	Otwórz plik w bieżącym oknie
%	Utwórz nowy plik w bieżącym katalogu i otwórz go do edycji (zapyta o nazwę)

2.23 Zmiana wcięcia

Zapomnij o używaniu taba. Zbyt wiele narzędzi używa standardowych tabów o szerokości 8 znaków. Jeżeli będziesz używał taba, to nawet jeśli zmienisz ustawienie `tabstop`, wiele programów będzie niepoprawnie wyświetlać i drukować Twój kod. Taby umarły, niech żyją wcięcia!
Polecam więc, żebyś ustawił `tabstop` na wartość 8 w pliku .vimrc, a `shiftwidth` na taką wartość, jaką uznasz za odpowiednią. Nie, tak naprawdę nie polecam, a wymagam od Ciebie, żebyś dodał te dwa polecenia do swojego .vimrc. Teraz. Poczekam. Naprawdę... idź to zrobić...

Polecenie	Rezultat
`set tabstop=8`	Używaj standardowych tabów o szerokości 8 znaków
`set shiftwidth=4`	Używaj 4-znakowych wcięć
`set shiftround`	Zmieniaj wcięcie wyrównując do wartości zmiennej `shiftwidth`
`set autoindent`	Automatycznie ustawiaj wcięcie nowej linii

Potrzebujesz również włączonej opcji `autoindent`, żebyś nie musiał ręcznie ustawiać wcięcia za każdym razem jak przejdziesz do nowej linii. Autoindent jest na tyle przydatną opcją, że dołączyłem ją w sekcji o niezbędnych ustawieniach w .vimrc. Jeżeli wykonywałeś polecenia z tego tutoriala, powinieneś mieć ją już włączoną. Nie włączenie jej jest głupotą – naprawdę chcesz ją mieć.
W trybie komend:

Polecenie	Rezultat
<	Przesunięcie w lewo (wymaga polecenia ruchu, działa na całych wierszach)
>	Przesunięcie w prawo (wymaga polecenia ruchu, działa na całych wierszach)

Jeżeli chcesz przesunąć cały akapit w lewo, to <} jest komendą, której potrzebujesz. Żeby przesunąć 3 wiersze w prawo, użyj 3». Komendy wcięcia są zgodne ze standardowym wzorem komend VIM-a (dlatego nazywa się on standardowym wzorem). Nie używają buforów.
W trybie wprowadzania/nadpisywania:

Polecenie	Rezultat
^t	Zwiększ wcięcie
^d	Zmniejsz wcięcie

2.24 Sprawdzanie pisowni

VIM może również sprawdzać Twoją pisownię. Wprowadź komendę `:set spell`, żeby włączyć sprawdzanie pisowni. Możesz również zdefiniować słownik i inne opcje, ale w tym celu odsyłam Cię do `:help spell`.
Nie polecam włączenia tej opcji na stałe. Wiele plików, które będziesz edytował będzie zawierało słowa, których nie znajdziesz w słowniku i może to być nieco denerwujące. Osobiście preferuję przełączanie tej opcji przy pomocy `:set spell` i `:set nospell`.
Bardziej dociekliwi mogą nauczyć się przełączać to ustawienie przy pomocy specjalnych skryptów uruchamianych w momencie ładowania pliku. Reszta może spokojnie to pominąć.

2.25 Drobne podpowiedzi

Istnieją pewne komendy pokazujące informacje w linii stanu lub w przewijalnym widoku. Kiedy potrzebujesz przypomnienia, ale nie chcesz przechodzić do innej części kodu, możesz z nich skorzystać.

Polecenie	Rezultat
[i	Pokazuje pierwszy wiersz, w którym pojawiło się słowo pod kursorem
[I	Pokazuje wszystkie wiersze, w których pojawiło się słowo pod kursorem
:g/wzór/	Pokazuje wszystkie wiersze pasujące do wzoru (wyrażenia regularnego)

2.26 Filtrowanie przy pomocy powłoki

Jeżeli pracowałeś już z wierszem poleceń, to możliwe, że wiesz jak korzystać z takich programów jak *sort* czy *grep*, a być może także jak wykonywać różne zadania przy pomocy *perla* czy *awk*. Wszystkie te programy to swego rodzaju filtry. Wczytują ze standardowego wejścia i zwracają wynik na standardowe wyjście.
Kiedy jesteś w VIM-ie, możesz robić dokładnie te same rzeczy. Niewątpliwie uciążliwe jest zapisanie fragmentu, który chcesz posortować, do pliku, przejście do linii komend, posortowanie tego pliku i załadowanie go z powrotem w miejsce nieposortowanych danych.
Co musisz wiedzieć, to to, że cała ta praca jest zbędna. Jeżeli chcesz posortować akapit, wystarczy że ustawisz kursor na jego początku i wciśniesz `!}sort`, a magia się dokona.
VIM jest napisany tak, że używa filtrów bezpośrednio. Jest to przydatne nie tylko ze względu na mnogość przydatnych filtrów dostarczanych razem z systemami uniksopodobnymi, ale także dlatego, że możesz w prosty sposób pisać swoje własne. Każdy program działający jak filtr, który napiszesz, będzie częścią zarówno Twojego edytora jak i powłoki systemowej. Ma to ogromny wpływ na sposób, w jaki edytujesz pliki. Jest to szczególnie ekscytujące, jeżeli już korzystasz z linii komend jak guru.

Polecenie	**Rezultat**
`!!polecenie`	Przefiltruj tylko bieżący wiersz
`!}polecenie`	Przefiltruj linie od bieżącej do końca akapitu
`!Gpolecenie`	Przefiltruj linie od bieżącej do końca pliku
`:%!polecenie`	Przefiltruj cały plik

2.27 Poprawianie formatowania kodu

Możesz zmieniać formatowanie kodu na kilka różnych sposobów. Jeden z nich to filtr powłoki:

Polecenie	**Rezultat**
`%!astyle`	Przeformatuj kod korzystając z programu astyle (Niezły program do poprawy formatowania)
`%!indent`	Przeformatuj kod korzystając z programu indent (Też dobry program, choć trochę starszy)

Innym sposobem jest komenda `gq`, która ponownie wykonuje zawijanie wierszy i potrafi poprawnie zawijać komentarze.

Polecenie	Rezultat
`gqq`	Popraw zawijanie bieżącego wiersza (podwójny skok!)
`gqj`	Popraw zawijanie bieżącego i kolejnego wiersza
`gq}`	Popraw zawijanie od bieżącego wiersza do końca akapitu

Możesz również zamienić znaki tabulacji na spacje, żeby u każdego wcięcia wyświetlały się tak samo. Aby to zrobić, ustaw zmienną `tabstop` na odpowiedni poziom wcięcia, włącz ustawienie `expandtab` i wykonaj polecenie `:retab`. Byłoby to zdecydowanie za dużo roboty, gdybym nie miał ustawionych `tabstop` i `expandtab` w domyślnej konfiguracji. Zazwyczaj ustawiam `tabstop`, wykonuję `:retab` i zapisuję plik. Jest to ciąg poleceń, który mogę zmapować pod jakiś klawisz lub zapisać jako makro.
Możesz również sprawić, że edytor będzie zawijał długie linie na bieżąco, zachowując poziom wcięć. Wszystko to ustawiasz przy pomocy opcji `linebreak`, `textwidth` i `autoindent`, o których możesz poczytać w systemie pomocy.

2.28 Tryb szybkich po prawej (QuickFix) Twoim przyjacielem

VIM potrafi uruchomić Twój plik Makefile i poprowadzić Cię przez wszystkie zmienne po kolei. Jeżeli korzystasz z testów jednostkowych jako części procesu budowania, a ich wynik zwracany jest w kompatybilnym formacie, to zostaniesz poprowadzony przez błędnie wykonane testy w taki sposób, jakby to były błędy kompilacji. Podobnie rzecz się ma z wszelkimi narzędziami do sprawdzania stylu, zakładając że zwracają wynik w odpowiednim formacie.
Jeżeli stosujesz technikę rozwoju sterowanego testami (Test Driven Development), to jest to krytyczna funkcja. Mając do dyspozycji tryb *QuickFix*, szybko wpadniesz w odpowiedni rytm pracy. Możesz nawet przypisać komendę `:make` pod jeden z klawiszy (patrz: `:help map`), żebyś nie musiał jej ciągle wpisywać. VIM jest pod tym względem dość elastyczny. W osobnej publikacji przedstawię swoje ustawienia VIM-a, nakierowane pod kątem TDD.
Podstawowe komendy trybu QuickFix:

Polecenie	Rezultat
`:make`	Wykonaj plik Makefile określony przez zmienną makefile
`:cw`	Pokaż okno z błędami kompilacji, jeżeli jakieś wystąpiły
`:cn`	Przejdź do następnego błędu kompilacji
`:cp`	Przejdź do poprzedniego błędu kompilacji

Jak zwykle, odsyłam do `:help quickfix` po więcej informacji na temat tego trybu.

2.29 Dostęp do stron podręcznika (man)

W rozdziale poświęconym plikowi .vimrc zalecałem włączenie funkcji `Man`. Ponieważ wykonywałeś moje polecenia, masz teraz dostęp do stron podręcznika z poziomu VIM-a.

`:Man 5 crontab` otworzy stronę dotyczącą tabeli `cron` w dzielonym oknie. Kursor będzie znajdował się w oknie pomocy, po którym będziesz mógł się poruszać tak jak w przypadku tagów: `^]`, żeby przejść do danego taga, `^t` - żeby powrócić. Kiedy skończysz korzystać z podręcznika, po prostu zamknij okno przez `:q` bądź `ZZ`.
Jeżeli szukasz strony podręcznika na temat czegoś w pliku, który edytujesz, nie musisz wciskać dwukropka i wpisywać słowa `Man`. Wystarczy, że wciśniesz znak lidera (`:help mapleader`) - domyślnie `\` - a następnie wielką literę *K*, żeby VIM znalazł odpowiednią stronę podręcznika i wyświetlił ją w nowym oknie. Oczywiście jeżeli zmienisz znak lidera, będziesz musiał odpowiednio dostosować poniższe komendy. Funkcja ta jest bardzo przydatna, jeżeli piszesz skrypty powłoki lub korzystasz z API języka C systemu Linux (lub Unix).

Polecenie	**Rezultat**
`:Man temat`	Wyświetl stronę podręcznika dotyczącą tematu
`\K`	Wyświetl stronę podręcznika dla słowa pod kursorem

Możesz poprawić przydatność tej funkcji upewniając się, że masz zainstalowane strony podręcznika dla wszystkich narzędzi i bibliotek, z których korzystasz. Albo przynajmniej prosząc administratora swojego systemu, żeby to zrobił. Jeżeli programujesz w perlu i nie masz zainstalowanych odpowiednich stron podręcznika, ta funkcja będzie dla Ciebie bezużyteczna.

2.30 Ctags pozwoli Ci na profesjonalną nawigację

Z całego serca polecam exuberant ctags jako program tagujący, dla niemal każdego języka. Potrafi on szybko przeskanować Twój kod i utworzyć plik *tagów*, który mówi VIM-owi wszystko, czego ten potrzebuje, żeby znaleźć odpowiednie symbole w kodzie. Plik tagów zawiera nazwy plików i wyrażenia regularne pozwalające znaleźć odpowiednią linię. Naprawdę nieźle sobie radzi.

Polecenie	**Rezultat**
`!ctags -R *`	Uruchamia ctags (najlepiej dodać to do pliku Makefile)
`^]`	Przejdź do definicji terminu znajdującego się pod kursorem (klasy/metody/zmiennej...)
`^t`	Powróć do poprzedniego miejsca

2.31 Zakładki

VIM pozwala na ustawienie zakładki na wierszu i skakanie od jednej zakładki, do drugiej.

Polecenie	Rezultat
`mx`	Utwórz zakładkę x w bieżącym wierszu
`'x`	Skocz do zakładki x

Możesz użyć dowolnej litery jako nazwy zakładki, ale jest różnica między wielkimi i małymi literami.

- Małe litery ustawiają zakładkę dla danego pliku, tak że `'a` w jednym pliku może Cię przenieść w zupełnie inne miejsce niż w innym.
- Wielkie litery ustawiają zakładkę globalną, więc `'A` przeniesie Cię do pliku i linii, w którym daną zakładkę utworzyłeś. Jest to bardzo przydatne, ale czasem może nie być do końca tym, czego byś chciał, ponieważ plik z zakładką jest ładowany w bieżącym oknie.

2.32 Wklejanie w trybie wstawiania

W trybie wstawiania nie jesteś ograniczony jedynie do pisania i uzupełniania wierszy. Są też inne komendy, a jedną z nich jest `^r`, która wkleja zawartość rejestru w miejscu kursora.
Załóżmy, że usunąłeś jakieś słowo - powiedzmy *kanarek* - i piszesz właśnie w zupełnie innym miejscu dokumentu. Wystarczy teraz, że wciśniesz `^r`, a następnie `"` (cudzysłów, nazwa domyślnego rejestru), żeby wkleić słowo *kanarek* w danym miejscu i kontynuować pisanie, pozostając w trybie wstawiania.
Jest to szczególnie przydatne przy operacjach typu `cw`, ponieważ komenda zmiany usuwa bieżące słowo (ładując je do rejestru) i przełącza w tryb wprowadzania. Załóżmy więc, że mój kursor jest umieszczony na słowie *kanarek*. Wykonuję następujące operacje:

Polecenie	Rezultat
`cw`	Usuwa bieżące słowo, umieszcza je w rejestrze ¨ i przełącza VIM-a w tryb wstawiania
`<b>`	Wprowadza tekst (jesteśmy w trybie wstawiania). Ten tekst to początek taga HTML oznaczającego tekst wytłuszczony
`^r`	Rozpoczyna wklejanie-w-trybie-wprowadzania
¨	Wkleja zawartość domyślnego rejestru (*kanarek*) w aktualnym położeniu kursora
`</b>`	Wprowadza tag zamykający
`Escape`	Powraca do trybu komend
`u`	Usuwa tag <b> ze słowa *kanarek*!

Uważaj, bo `.` nie wie o tym, że użyłeś komendy `^r`, więc jeśli przejdziesz do kolejnego słowa i wciśniesz kropkę, to zostanie ono zamienione na słowo *kanarek* objęte w tag wytłuszczenia. Jeżeli chcesz wytłuszczać wiele słów, to powinieneś nauczyć się nagrywać i odtwarzać makra - `:help q`

2.33 Skracaj!

Jeden z najprostszych sposobów, w jaki możesz dostosować swój edytor, to skróty. Dla przykładu, jedną z najczęściej wpisywanych błędnie linijek w języku Python jest ta słynna inwokacja funkcji `main`:

```
if __name__ == ¨__main__¨:
```

Chciałbym, żeby po wpisaniu słowa `pymain`, edytor wprowadzał tę inwokację za mnie. Banalne:

```
:ab pymain if __name__ = ¨__main__¨:
```

Teraz za każdym razem, gdy wpiszę dowolny znak spoza alfabetu po słowie pymain, zostanie ono automatycznie rozwinięte. Wszystko czego VIM-owi potrzeba to komenda ab i cały wyraz w trybie wprowdzania. Rozwinięcie jest natychmiastowe i automatyczne, nie trzeba wciskać żadnej kombinacji klawiszy. W rezultacie VIM będzie rozwijał wyrazy nawet wtedy, gdy tego nie chcesz. Za każdym razem, gdy wpiszę pymain, otrzymuję powyższe rozwinięcie; nawet jeżeli wpisałem to przez przypadek. Właściwie muszę intencjonalnie pisać to słowo z błędem, wychodzić z trybu wprowadzania i wracać, żeby je poprawić, bo nie mam innej możliwości wpisania tego słowa.

Mogę dodać to rozwinięcie do mojego .vimrc, o ile oczywiście pominę początkowy dwukropek. Mam w swoim .vimrc już kilka takich rozwinięć, ponieważ dobieram je bardzo starannie.

Zauważyłem, że czasem wpisuję *teh* zamiast *the*. Jest to proste to naprawienia:

```
:ab teh the
```

Nigdy nie piszę *teh* specjalnie, więc jest to całkiem niezły kandydat do *skrócenia*.

Dodając własne skróty dobrze je przemyśl, żebyś nie otrzymywał niechcianych rozwinięć. Jednak dobrze jest uwzględnić częste literówki czy dłuższe fragmenty kodu, które w przeciwnym razie musiałbyś wpisywać ręcznie.

2.34 Nagrywaj i odtwarzaj makra

W systemie pomocy nazywa się to *złożonym powtórzeniem* (ang. *complex-repeat*).
Do trybu nagrywania makra przełączasz się wciskając `q`, a następnie nazwę rejestru, do którego makro ma zostać zapisane. Możesz użyć dowolnej litery alfabetu (małej lub wielkiej) oraz cyfr. Oczywiście możesz mieć wszystkie 26 mały liter, 26 wielkich oraz 10 cyfr przypisanych jednocześnie do różnych makr, jeśli taka jest Twoja wola.
Każde klawisz, który naciśniesz zostanie nagrany, aż do ponownego naciśnięcia `q`.
Żeby odtworzyć makro, wciśnij `@` i nazwę rejestru.
Po tym jak odtworzysz dane makro, komenda u będzie je widziała jako jedną operację. Jest to bardzo przydatne, ponieważ makro może wykonywać zmiany w wielu wierszach w całym pliku.
VIM zapamiętuje ostatnie makro, które odtworzyłeś i pozwala Ci na powtórzenie go przy wykorzystaniu *podwójnego skoku*. W tym wypadku będzie to `@@`.
Komenda `.` również postrzega makro jako jedną operację. Jest to świetne, bo powyższa lekcja staje się dużo bardziej przydatna. Działa to jakoś tak:

Polecenie	**Rezultat**
`qa`	Rozpoczyna nagrywanie makra *a*
`cw`	Usuwa bieżące słowo, umieszcza je w rejestrze ¨ i przełącza VIM-a w tryb wstawiania
`<b>`	Wprowadza tekst (jesteśmy w trybie wstawiania). Ten tekst to początek taga HTML oznaczającego tekst wytłuszczony
`^r`	Rozpoczyna wklejanie-w-trybie-wprowadzania
¨	Wkleja zawartość domyślnego rejestru (*kanarek*) w aktualnym położeniu kursora
`</b>`	Wprowadza tag zamykający
`Escape`	Powraca do trybu komend
`q`	Kończy nagrywanie makra
`W`	Przechodzi do kolejnego słowa
`@@`	Odtwarza makro, obejmując słowo w tag wytłuszczenia
`W`	Przechodzi do kolejnego słowa
`.`	Ponownie odtwarza makro

Makro, które nagrałeś, to po prostu tekst w rejestrze. Możesz je wkleić do dokumentu, usprawnić jego działanie, skopiować z powrotem do rejestru, etc.

Makra to niezły sposób na usprawnienie skomplikowanych zmian.
Wpisz `:help q`, żeby dowiedzieć się więcej o używaniu makr. Nie powiedziałem wszystkiego.

2.35 Mapowanie klawiszy

Wszystko, co możesz zrobić ręcznie lub przy pomocy makra, możesz również zmapować pod jakiś klawisz. Mapowanie po prostu podpina makro pod dany klawisz. Na przykład:

¨ Przełączanie pomiędzy plikami na długiej liście `map <F3> :prev<CR>`
`map <F4> :next<CR>`
Poczytaj więcej o mapowaniu wpisując `:help map`.

2.36 Kolory i inne takie

VIM-a ma niesamowite możliwości dostosowania, włącznie z kolorami, których używa. Niektórzy udostępniają swoje schematy kolorów w Internecie, a kilka jest również dostarczanych wraz ze standardową dystrybucją. Ten, który najczęściej spotykasz to zapewne *default* (ang. *domyślny*). Jeżeli korzystasz z osobnego schematu kolorów, możesz go podejrzeć korzystając z komendy:

```
:echo g:colors_name
```

Wypróbuj istniejące schematy, takie jak *delek, darkblue, desert, koehler, elflord, peachpuff*, czy *slate*. Zawsze możesz powrócić do domyślnego albo po prostu opuścić edytor i włączyć go ponownie. Komenda, której potrzebujesz to:

```
:colorscheme nazwa_schematu
```

Możesz przejrzeć dostępne schematy w trybie przeglądarki wpisując:

```
:e $VIM/vim73/colors
```

(Zakładając, że używasz VIM-a 7.3. Możliwe, że będziesz musiał dostosować tę komendę do swojej wersji.)
Schematy kolorów zbudowane są z komend, które ustawiają poszczególne elementy, takie jak kolor tekstu czy tła linii stanu. Możesz się naprawdę wiele nauczyć analizując jeden czy dwa takie schematy.
Komendy kolorów zaczynają się od `hi` (skrót od highlight), po którym następuje nazwa elementu, który chcesz pokolorować (zwana nazwą grupy), oraz łańcuchy znaków odpowiadające kolorom do użycia odpowiednio w: zwykłym terminalu (spytaj ojca) - *term*, kolorowym terminalu - *cterm* i wreszcie w graficznym interfejsie - *gui*.
Poniżej znajdziesz kilka ustawień, które lubię w gvimie:

```
hi LineNr guibg=lightgray guifg=black
hi StatusLine guifg=yellow guifg=darkblue
hi NonText guibg=darkgray hi ToDo guifg=DarkRed
```

Kolorują one obszar, który znajduje się poza tekstem, kolumnę numerów linii po lewej stronie (jeśli wykonasz `set nu`) i koloruje aktywny pasek stanu odmiennie od nieaktywnego.
Po więcej informacji na temat schematów kolorów odsyłam do wbudowanej pomocy (`:help hi`) lub do jakiegoś innego, bardziej rozbudowanego przewodnika po VIM-ie.

2.37 Wykorzystywanie ścieżki (ang. *path*)

Istnieje specjalna zmienna, zwana `path`, która pomoże Ci odnaleźć pliki, do których odwołują się aktualnie przez Ciebie edytowane. Jest to szczególnie przydatne, gdy piszesz program w języku C i skierujesz zmienną path na katalogi `/usr/include/*`.
Wartość, którą przyjmuje ta zmienna, to lista ścieżek rozdzielonych przecinkami, w których VIM powinien szukać plików. Pozwala to na wskazanie standardowych katalogów plików dołączanych, jak również plików Twojego projektu czy dowolnych innych, które uznasz za przydatne.
Żeby skorzystać z tej funkcji, umieść kursor na nazwie pliku i (w trybie komend) wciśnij `gf` lub `^wf`. Plik zostanie załadowany w aktualnym oknie.

Polecenie	**Rezultat**
`set path=., .*, /usr/include/*, /usr/share/*`	Prawdopodobnie nadmiarowe, ale ustawia zmienną path na przeszukiwanie praktycznie wszystkiego - może zająć trochę czasu.
`gf`	Przejdź do pliku (ang. *Goto File*): otwiera plik, którego nazwa znajduje się pod kursorem.
`^wf`	Otwórz w oknie (ang. *Window File*): podobne do `gf` z tym, że otwiera plik w nowym oknie.
`:e#`	Powraca do poprzedniego pliku

Uważam, że otwieranie w nowym oknie jest generalnie bardziej przydatne, ale jednocześnie `gf` jest na tyle łatwiejsze do wpisania, że używam go częściej. Chciałbym, żeby kolejne edytowane pliki były odkładane na stosie, żeby można było do nich powracać przy pomocy `^t`, ale niestety tak nie jest. Można to obejść ustawiając zakładkę nazwaną wielką literą, żeby można było do niej powrócić z innego pliku. Wiem, że brzmi to nieco okropnie, bo takie jest, ale przynajmniej działa.
Mimo wszystko, programiście C/C++, tt \K w połączeniu z `^wf` pozwala na całkiem niezłe poruszanie się między plikami, a zakładki są przydatne do przechodzenia z/do plików nagłówkowych i innych.
Zmienna `path` jest również wykorzystywana przez mechanizm uzupełniania (`^n`) do znajdywania plików, w których szuka dopełnień.

Wybierz swój nowy system, poznaj Linuksa krok po kroku!

Twoje pierwsze źródło wiedzy o Linuksie

w wortalu:

- recenzje dystrybucji
- porównanie funkcji
- opis instalacji
- i dużo więcej...

ponad ćwierć miliona odwiedzin miesięcznie!

jakilinux.org

kontakt / reklama
borys.musielak@gmail.com