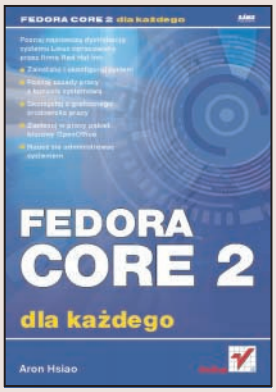
# Fedora Core 2 dla każdego




Autor: Aron Hsiao
ISBN: 83-7361-552-0
Tytuł oryginału: Teach Yourself Red Hat Linux Fedora in 24 Hours
Format: B5, stron: 528

Fedora Core 2 to otwarta platforma linuksowa przeznaczona do użytku osobistego albo dla małych firm. Jej twórca – firma Red Hat Inc. wykorzystuje ją w roli podstawy dla systemów klasy wyższej, takich jak Red Hat Enterprise Linux. Systemy operacyjne firmy Red Hat pozostają najlepiej znaną i najbardziej lubianą spośród wszystkich odmian Linuksa. Uważa się, że to właśnie dzięki firmie Red Hat Linux trafił ze świata idei na komercyjny rynek. Systemy operacyjne Red Hat cechują się także doskonałym zrównoważeniem mocy, wygody instalacji i łatwości zarządzania. Jeśli chcesz wypróbować ten system, otwarta platforma firmy Red Hat – Fedora Core 2 – nadaje się do tego wręcz idealnie.

Książka „Fedora Core 2 dla każdego" to wprowadzenie do systemu Fedora Core 2, przeznaczone dla osób, które nigdy wcześniej nie miały do czynienia z systemami linuksowymi. Zawiera informacje związane z instalacją i konfiguracją systemu, wykorzystywaniem go w domu i pracy oraz podstawami administracji Fedorą.

- Podział dysku na partycje i instalacja systemu
- Pierwsze uruchomienie Linuksa
- Praca z systemem plików
- Korzystanie z konsoli systemowej
- System składu dokumentów LaTeX
- Korzystanie z internetu z poziomu powłoki systemowej
- Tworzenie skryptów powłoki
- Graficzne środowiska pracy Gnome i KDE
- Pakiet biurowy OpenOffice
- System X Window
- Administracja systemem w środowisku tekstowym i graficznym
- Mechanizmy zabezpieczeń
- Instalowanie oprogramowania
- Konfigurowanie usług sieciowych
- Tworzenie kopii zapasowych

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

# Spis treści

Rozdział 5.

# Zaprzęganie konsoli do pracy

W tym rozdziale poznasz kilka pojęć, które będą tym bardziej użyteczne, im dłużej będziesz używał Linuksa. Zaczniemy od przedstawienia dwóch najpopularniejszych edytorów, z którymi będziesz stale spotykał się w świecie Linuksa. Następnie pokażemy, jak ułatwić sobie pracę na konsoli.

Podobnie jak w rozdziale 4. („Nawigowanie po Linuksie za pomocą konsoli") także tutaj chodzi przede wszystkim o nabranie wprawy w posługiwaniu się powłoką oraz jej poleceniami i technikami, a nie o to, aby natychmiast zostać profesjonalistą. Staraj się zrozumieć, o co chodzi w każdym przykładzie, i nie przejmuj się, jeśli wszystkiego nie zapamiętasz.

Przed przystąpieniem do lektury niniejszego rozdziału zaloguj się do konsoli wirtualnej i przygotuj do wpisywania poleceń.

## Tworzenie, edytowanie i zapisywanie plików tekstowych

W poprzednim rozdziale utworzyłeś kilka plików tekstowych za pomocą polecenia `touch`, ale były to *puste* pliki tekstowe — niezbyt przydatne do zwykłych zastosowań. Kiedy tworzysz plik tekstowy, zwykle chcesz, żeby zawierał wprowadzony przez Ciebie tekst.

W wierszu poleceń Linuksa do tworzenia plików tekstowych zwykle używa się jednego z dwóch skrajnie różnych edytorów. Pierwszy to klasyczne narzędzie uniksowe o nazwie `vi` (mówiąc dokładniej, w systemie Fedora Core 2 `vi` jest aliasem do programu `vim`); drugi to duży, rozszerzalny procesor tekstu o nazwie `emacs`. Oba programy

dobrze spełniają wymagania stawiane edytorom tekstu. Edytor `vi` podoba się osobom, które cenią minimalizm i szybkość, a `emacs` jest zwykle preferowany przez zaawansowanych użytkowników. Wybór edytora używanego w codziennej pracy jest w dużej mierze kwestią gustu. Jednakże edytor `vi` jest wykorzystywany przez niektóre narzędzia administracyjne, więc powinieneś nauczyć się przynajmniej podstawowej obsługi tego programu.

**Różnice między edytorem i procesorem tekstu**

Procesor tekstu zawiera funkcje związane ze składem komputerowym, takie jak wybór kroju czcionki, sterowanie układem strony oraz opcje formatowania wydruku. Edytor tekstu służy wyłącznie do edytowania liter i liczb; nie pozwala zmieniać wyglądu ani układu drukowanego tekstu.

# Tworzenie plików tekstowych za pomocą edytora vim

Edytor `vi` (w systemie Fedora Core 2 `vim`) jest popularny, bo — w przeciwieństwie do edytora `emacs` — jest dostępny w niemal każdym systemie uniksowym, nawet najbardziej ograniczonym. Ponadto jest znacznie mniejszy i szybszy.

Zacznij teraz pracę nad pustym dokumentem, wpisując polecenie `vi` oraz argument w postaci nazwy pliku, który chcesz utworzyć. Nadaj plikowi nazwę *moj_plik_vi.txt*:

```
[ty@fedora2 ty]$ vi moj_plik_vi.txt
```

**Wczytywanie istniejących plików**

Uruchomienie edytora `vi` nie zawsze powoduje utworzenie pliku. Jeśli podasz programowi `vi` nazwę istniejącego pliku, zostanie on wczytany do edytora i będziesz mógł go zmodyfikować.

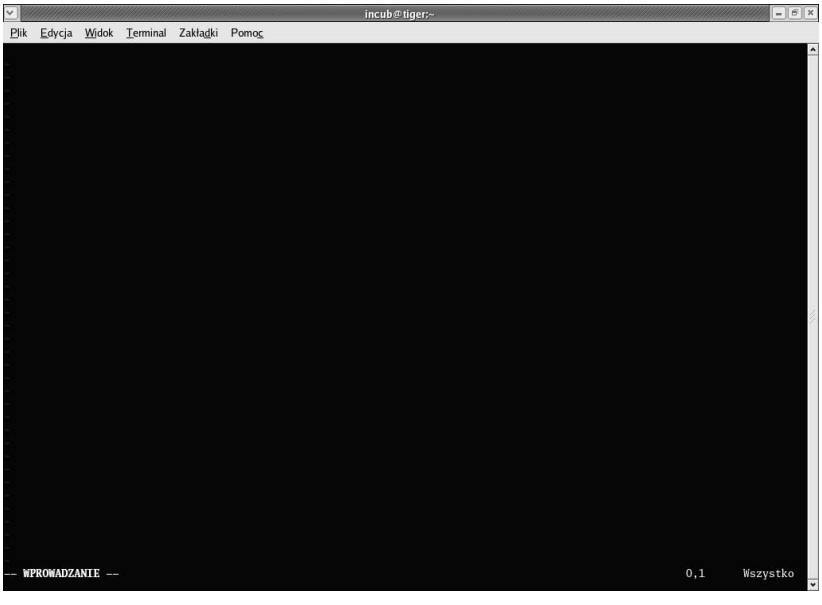
Kiedy program `vi` się uruchomi, zobaczysz kolumnę znaków tyldy (~) po lewej stronie konsoli. Znaki te wskazują, że w poszczególnych wierszach nie ma jeszcze żadnego tekstu. Nie próbuj niczego pisać; na razie jest to niemożliwe. Program `vi` to edytor wierszowy, który ma dwa tryby pracy. Kiedy uruchamiasz `vi`, program zaczyna działać w trybie nieprzeznaczonym do wprowadzania tekstu.

Jest to tak zwany **tryb poleceń**. Wszystko, co wpisujesz w tym trybie, łącznie ze zwykłymi znakami alfabetu, nie pojawia się w dokumencie, lecz jest interpretowane przez `vi` jako żądanie wykonania określonej operacji. W trybie poleceń możesz zapisywać plik, przesuwać kursor, usuwać frazy albo wiersze tekstu itd.

Bezpośrednie wprowadzanie tekstu jest możliwe tylko w **trybie wstawiania**; kiedy `vi` działa w tym trybie. Ta cecha edytora nieodmiennie sprawia problemy początkującym użytkownikom, ale zwykle odrobina praktyki wystarczy, żeby sprawnie posługiwać się systemem z dwoma trybami.

# Wstawianie tekstu w edytorze vi

Aby zacząć wstawiać tekst do pliku, nad którym pracujesz (*moj_plik_vi.txt*), naciśnij klawisz *i*. Zwróć uwagę, że na dole ekranu pojawi się słowo `WPROWADZANIE` czyli *Insert* (zobacz rysunek 5.1).




**Rysunek 5.1.** *Edytor vi działa w trybie wstawiania i oczekuje na wprowadzenie tekstu. Wszystkie znaki wpisane z klawiatury są teraz interpretowane jako dane wejściowe, dopóki nie naciśniesz klawisza Esc*

Wpisz kilka wierszy tekstu. Możesz przepisać przykładowe zdania albo wprowadzić własny tekst. Zauważ, że edytor `vi` nie zawija wierszy; kiedy dotrzesz do końca wiersza, musisz nacisnąć klawisz *Enter*, aby rozpocząć nowy wiersz:

```
Edytor vi jest bardzo rozpowszechniony w systemie Unix. Z tego powodu wszyscy
przyszli użytkownicy Uniksa powinni nauczyć się jego obsługi.
```

**Brak zawijania znaków w edytorze vi**

Jeśli nie naciśniesz klawisza *Enter*, kiedy dotrzesz do prawego marginesu, tekst na ekranie zawinie się w środku słowa do następnego wiersza. Pamiętaj jednak, że tekst w buforze pamięciowym edytora `vi` pozostanie niezawinięty. Jeśli wypełnisz tekstem cały ekran, nie naciskając klawisza *Enter*, zapisany plik będzie zawierał *jeden długi wiersz* tekstu.

Może to mieć nieoczekiwane konsekwencje podczas drukowania, wysyłania pocztą, a nawet edytowania plików tekstowych.

Kiedy skończysz wpisywać tekst, naciśnij klawisz *Esc*, aby opuścić tryb wstawiania i wrócić do trybu poleceń. Zwróć uwagę, że słowo `WPROWADZANIE` zniknie z wiersza na dole ekranu.

Wstawiłeś kilka słów. A jeśli chciałbyś zmienić to, co napisałeś do tej pory? Sposoby dokonywania zmian w edytorze `vi` nie zawsze są oczywiste.

# Edycja tekstu w edytorze vi

W edytorze `vi` do nawigacji oraz do edycji tekstu używa się poleceń klawiaturowych. Najczęściej używane polecenia zebrano w tabeli 5.1.

**Tabela 5.1.** *Polecenia klawiaturowe programu `vi` do nawigacji i edycji tekstu*

| Klawisz | Operacja |
|---|---|
| *l* | Przesuwa kursor o jeden znak w prawo. |
| *h* | Przesuwa kursor o jeden znak w lewo. |
| *j* | Przesuwa kursor w dół o jeden wiersz *tekstu* (nie wiersz ekranu). |
| *k* | Przesuwa kursor w górę o jeden wiersz *tekstu* (nie wiersz ekranu). |
| *x* | Usuwa znak pod kursorem. |
| *d*#<spacja> | Usuwa # znaków pod kursorem i na prawo od kursora. |
| *dd* | Usuwa bieżący wiersz. |
| *i* | Wstawianie: włącza tryb wstawiania; wstawianie zaczyna się od bieżącej pozycji kursora. |
| *a* | Uzupełnianie: włącza tryb wstawiania; wstawianie zaczyna się na prawo od bieżącej pozycji kursora. |
| *A* | Uzupełnianie: włącza tryb wstawiania; wstawianie zaczyna się od końca bieżącego wiersza *tekstu* (nie wiersza ekranu). |
| *Esc* | Włącza tryb poleceń. |

Spróbuj użyć klawiszy nawigacyjnych i edycyjnych w trybie poleceń do zmodyfikowania wpisanego przed chwilą tekstu, do usunięcia wiersza albo kilku znaków oraz do wstawienia lub uzupełnienia tekstu w pliku. Jeśli poczujesz się zdezorientowany albo klawiatura przestanie reagować podczas edytowania tekstu, naciśnij kilkakrotnie klawisz *Esc*, aby upewnić się, że wróciłeś do trybu poleceń.

**Klawisze strzałek w edytorze vi**

Jeśli wykonywałeś ćwiczenia i eksperymentowałeś z edytorem `vi`, prawdopodobnie odkryłeś, że jego obsługa jest łatwiejsza, niż sugerowałby powyższy opis. Fedora zawiera ulepszoną wersję edytora `vi` o nazwie `vim`. W tej wersji `vi` można poruszać się po tekście za pomocą klawiszy strzałek, nawet w trybie wstawiania, pod warunkiem, że terminal jest poprawnie skonfigurowany (konsola Linuksa spełnia ten warunek).

Ulepszona wersja `vi` nie jest jednak dostępna w większości systemów uniksowych, a nawet nie we wszystkich systemach linuksowych. Co więcej, nawet w Fedorze niektóre operacje administracyjne ograniczają Cię do tradycyjnych klawiszy `vi`. Powinieneś zapoznać się więc z ich działaniem na wypadek, gdybyś w przyszłości musiał pracować z tradycyjnym edytorem `vi`.

Poeksperymentuj przez chwilę z edytorem `vi` i spróbuj przyswoić sobie zasady jego obsługi. W Linuksie będziesz miał z nim do czynienia na tyle często, że nie będzie to stracony czas.

# Zapisywanie pliku, zamykanie edytora i dalsza lektura

Aby zapisać właśnie utworzony plik, trzeba wprowadzić dość tajemnicze polecenie. W trybie poleceń wpisz dwukropek (`:`). Zauważ, że dwukropek pojawia się w lewym dolnym rogu ekranu, a kursor przenosi się tuż obok dwukropka; edytor `vi` czeka teraz na wpisanie bardziej złożonego polecenia. Wpisz małą literę `w` i naciśnij klawisz *Enter*. Edytor `vi` wyświetli w dolnym ekranu następujący komunikat:

```
"moj_plik_vi.txt" [Nowy] 2L, 141C written
```

Edytor podaje najpierw nazwę zapisanego pliku; w tym przypadku jest to *moj_ plik_ vi.txt*. Następnie wyświetla liczbę wierszy w pliku (2) oraz liczbę znaków (141). Zapisałeś swój pierwszy plik `vi`.

W edytorze `vi` istnieje też funkcja „zapisz jako”. Aby ponownie zapisać tekst, tym razem w nowym pliku, wpisz dwukropek, literę `w`, spację i nazwę `moj_nowy_plik_vi.txt`. Edytor wyświetli komunikat:

```
"moj_nowy_plik_vi.txt" [Nowy] 2L, 141C written
```

Teraz w swoim katalogu macierzystym masz dwie kopie pliku — *moj_plik_vi.txt* i *moj_nowy_plik_vi.txt*. Ponieważ jest to książka dla początkujących, nie będziemy się więcej zajmować edytorem `vi`. Aby zamknąć program, wpisz dwukropek i literę `q`. W ten sposób opuścisz edytor i wrócisz do wiersza poleceń.

**Przypomnienie o zapisie danych w edytorze vi**

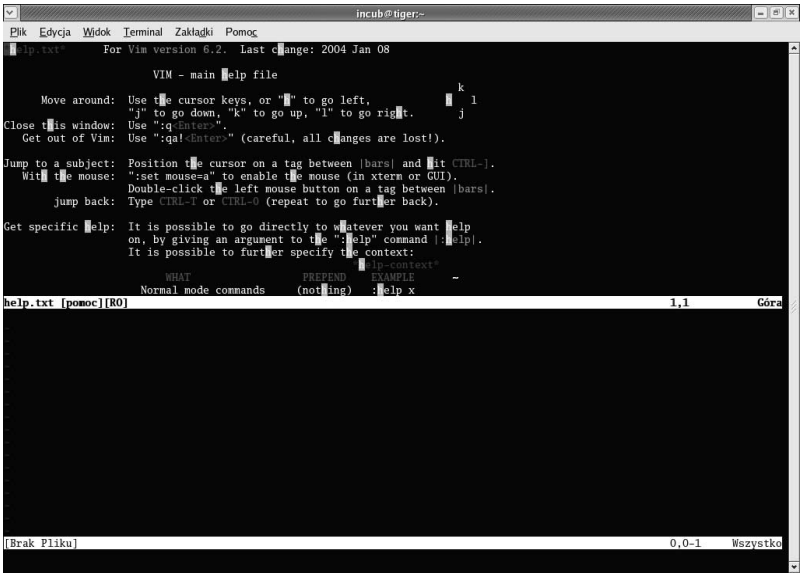
Jeśli spróbujesz zamknąć edytor bez zapisania pliku, nad którym pracowałeś, edytor `vi` wyświetli odpowiedni komunikat o błędzie, a następnie wróci do trybu poleceń.

Jeśli rzeczywiście chcesz zamknąć program bez zapisywania pliku, wpisz polecenie `:q!` (ze znakiem wykrzyknika) zamiast `:q`; edytor zastosuje się do Twojego życzenia.

Edytor `vi` jest bardzo zaawansowany i wart dokładniejszego przestudiowania, zarówno ze względu na liczne funkcje zwiększające wydajność edytowania tekstu, jak i dlatego, że często będziesz miał z nim do czynienia podczas pracy w wierszu poleceń Linuksa lub Uniksa. Jeśli masz odwagę, możesz przejrzeć obszerny system pomocy edytora `vi`, ponownie uruchamiając edytor i wpisując `:help` w trybie poleceń. Stronę tytułową systemu pomocy pokazano na rysunku 5.2.

**Więcej informacji o edytorze vi**

Bardziej szczegółowy przewodnik po edytorze `vi` znajdziesz w książce *Sams Teach Yourself Unix in 24 hours*, której autorem jest Dave Taylor.




**Rysunek 5.2.** *Ekran tytułowy systemu pomocy w edytorze vi (vim). Pomoc jest dość obszerna; wystarczy przeczytać te dokumenty, aby zostać biegłym użytkownikiem vi*

# Tworzenie plików tekstowych za pomocą edytora emacs

Jeśli edytor `vi` jest zbyt minimalistyczny i nietypowy jak na Twój gust, prawdopodobnie bardziej spodoba Ci się `emacs`, choć i w nim nie brak rozwiązań ekscentrycznych. Edytor `vi` skupia się na minimalizmie, podstawowej funkcjonalności i szybkości, natomiast `emacs` jest rozszerzalny, programowalny i z biegiem lat rozrósł się do monstrualnej wielkości. Bardzo niewiele osób wie wszystko o tym edytorze; do opanowania są setki trybów, poleceń i kompletny język programowania (Emacs LISP).

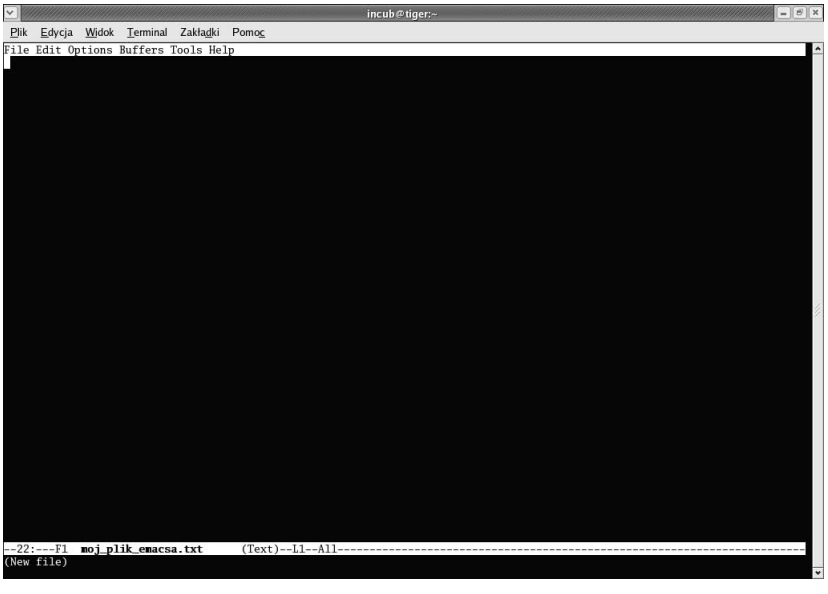
Na naszym poziomie `emacs` będzie jednak wydawał się przyjazny i prosty w porównaniu z `vi`. Możesz go uruchomić tak samo jak `vi`: wpisz polecenie `emacs` oraz argument w postaci nazwy pliku, który chcesz otworzyć albo utworzyć. Plikowi nadamy nazwę *moj_plik_emacsa.txt*:

```
[ty@fedora2 ty]$ emacs moj_plik_emacsa.txt
```

W zależności od konfiguracji systemu `emacs` może wczytywać się dłużej niż `vi`; jest to znacznie większy i bardziej skomplikowany edytor. Po uruchomieniu programu pojawi się ekran, pokazany na rysunku 5.3.

Wstawianie tekstu w edytorze `emacs` jest proste: wystarczy zacząć pisać. Przekonasz się, że klawisze strzałek, klawisze *Backspace* i *Delete* oraz klawisze *Page Up*, *Page Down*, *Home* i *End* działają zgodnie z oczekiwaniami.

Kiedy dotrzesz do końca pierwszego wiersza, zauważysz, że `emacs` — podobnie jak `vi` — nie zawija automatycznie wierszy. `emacs` jest edytorem wierszowym, więc musisz pamiętać o naciskaniu klawisza *Enter* na końcu każdego wiersza, bo w przeciwnym razie zapisany plik będzie zawierał tylko jeden bardzo długi wiersz tekstu.




**Rysunek 5.3.** *Edytor emacs wygląda zdecydowanie bardziej przystępnie niż vi i funkcjonuje w sposób zbliżony do tego, do którego przyzwyczajeni są użytkownicy Windowsa*

# System menu w edytorze emacs

Choć na górze ekranu edytora `emacs` znajduje się coś, co wygląda jak menu, podczas pracy na konsoli nie da się go uaktywnić. Można jednak w niemal dowolnym momencie nacisnąć klawisz *F10*, aby uzyskać dostęp do względnie intuicyjnego menu. Po naciśnięciu klawisza *F10* ekran dzieli się na dwie części. W dolnej części wyświetlane są opcje, które można uaktywnić jednym naciśnięciem klawisza (zobacz rysunek 5.4).


**Rysunek 5.4.** *Po wyświetleniu menu edytora emacs podczas pracy na konsoli możesz wybierać poszczególne pozycje, naciskając odpowiedni klawisz*

Aby zapisać plik, nad którym pracowałeś, wpisz `f`, aby otworzyć menu *File*, a następnie wpisz `s`. W niewielkim obszarze na dole ekranu (nazywanym **minibuforem**) pojawi się komunikat o zapisaniu pliku:

```
wrote /home/ty/moj_plik_emacsa.txt
```

Za pomocą menu przywoływanego klawiszem *F10* możesz wykonywać większość podstawowych operacji edycyjnych oraz zapisywać i wczytywać pliki. Większość użytkowników woli posługiwać się edytorem `emacs` niż `vi` właśnie ze względu na menu *F10* oraz prostszy sposób edytowania tekstu.

# Najważniejsze polecenia klawiaturowe edytora emacs

Jeśli masz zręczne palce, będziesz mógł używać edytora `emacs` i menu *F10* przez długie miesiące, nie wpadając w kłopoty. Jednakże przypadkowo naciśnięty klawisz może sprawić, że `emacs` zacznie piszczeć i wyświetlać zagadkowe komunikaty o błędzie albo niezrozumiałe żądania w minibuforze.

W takich okolicznościach z pomocą przychodzi kombinacja klawiszy zapisana w dokumentacji edytora `emacs` jako `C-g` (przytrzymaj wciśnięty klawisz *Ctrl* i naciśnij klawisz *g*). Jest to polecenie przerwania bieżącej operacji. Edytor `emacs` jest tak rozbudowany, że daje wiele okazji do przełączenia się w nieznany stan; w większości przypadków wystarczy kilkakrotnie nacisnąć `C-g`, aż wszystko wróci do normy.

W tabeli 5.2 zebrano kilka innych interesujących poleceń klawiaturowych edytora `emacs` — niektóre z nich są dostępne w systemie menu, inne nie. Aby zrozumieć tę tabelę, musisz najpierw zapoznać się z formatem poleceń edytora:

- `C-`*x*, gdzie *x* jest literą, oznacza: przytrzymaj naciśnięty klawisz *Ctrl* i naciśnij klawisz litery.
- `C-`*xy*, gdzie *x* i *y* są literami, oznacza: przytrzymaj wciśnięty klawisz *Ctrl*, naciśnij pierwszą literę, zwolnij klawisz *Ctrl* i pierwszą literę, a następnie naciśnij drugą literę.
- `M-`*x*, gdzie *x* jest literą, oznacza: przytrzymaj wciśnięty klawisz *Alt* i naciśnij literę.
- `C-`*x* `C-`*y*, gdzie *x* i *y* są literami, oznacza: przytrzymaj wciśnięty klawisz *Ctrl*, naciśnij pierwszą literę, a następnie bez zwalniania klawisza *Ctrl* naciśnij drugą literę.

Wielkość liter w tych kombinacjach ma znaczenie, więc wpisuj wskazane litery, a nie ich małe lub wielkie odpowiedniki.

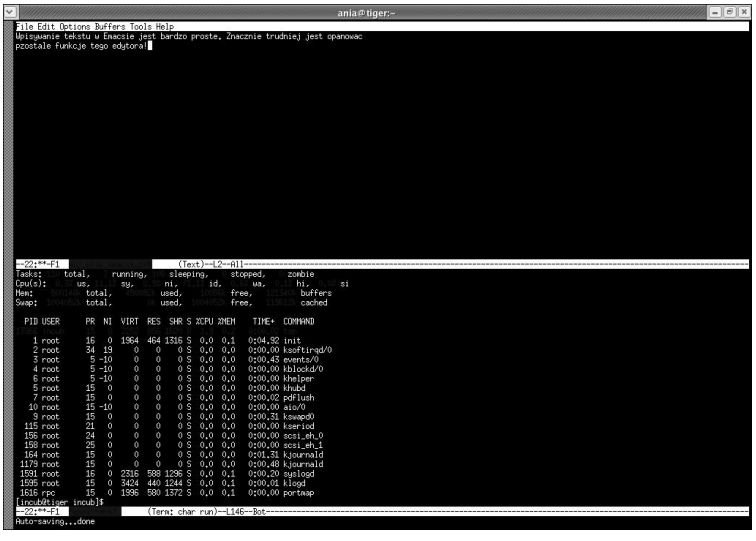
Zanim skończymy opis edytora `emacs`, podamy jeszcze jedno polecenie, od którego zaczyna się droga do zostania prawdziwym znawcą tego programu. Aby poznać dostępne polecenia, naciśnij `M-x`, a następnie klawisz *Tab*. Pojawi się bufor z listą dostępnych poleceń edytora `emacs`. Naciśnij `C-xo`, aby przenieść kursor do tego bufora; możesz teraz przeglądać dostępne polecenia za pomocą klawiszy *Page Up* i *Page Down*. Jeśli znajdziesz jakieś interesujące polecenie, wpisz je w minibuforze i naciśnij klawisz *Enter*.

**Tabela 5.2.** *Przydatne polecenia klawiaturowe programu emacs*

| Polecenie | Opis |
|---|---|
| `C-ht` | Uruchamia samouczek edytora `emacs`, w którym udokumentowanych jest znacznie więcej kombinacji klawiszy. |
| `C-x C-f` | Otwiera lub tworzy plik w bieżącym okienku edycyjnym; nazwę pliku albo ścieżkę do pliku wpisuje się w minibuforze. |
| `C-x C-s` | Zapisuje bieżący plik. |
| `C-x2` | Dzieli pionowo bieżące okienko edycyjne. |
| `C-x3` | Dzieli poziomo bieżące okienko edycyjne. |
| `C-x1` | Rozciąga bieżące okienko edycyjne na cały ekran (ukrywa lub usuwa inne okienka). |
| `C-xo` | Wybiera następne okienko edycyjne. |
| `C-xk` | Zamyka (bez zapisywania) bieżący bufor. Możesz użyć tego polecenia na przykład do zamknięcia samouczka. |
| `C-xb` | Przełącza bieżący bufor; nazwę bufora wpisuje się w minibuforze. Jeśli nie pamiętasz nazw otwartych buforów, naciśnij klawisz *Tab*, aby wyświetlić pełną listę. |
| `C-xi` *plik* | Wstawia plik w bieżącym położeniu kursora. Nazwę pliku wpisuje się w minibuforze. |
| `C-x C-c` | Zamyka edytor `emacs`. |
| `C-g` | Przerywa bieżącą operację edytora. |

Wśród wielu poleceń edytora `emacs` znajdziesz poniższe:

- `M-x dunnet` uruchamia tekstową grę przygodową.
- `M-x auto-fill-mode` włącza zawijanie wierszy.
- `M-x calendar` otwiera nowe okienko z kalendarzem na trzy miesiące.
- `M-x ansi-term` otwiera powłokę w bieżącym okienku. Zanim wydasz to polecenie, wpisz `C-x2` albo `C-x3`, aby móc jednocześnie edytować plik i pracować z powłoką, przełączając się między okienkami za pomocą kombinacji `C-xo` (zobacz rysunek 5.5).




**Rysunek 5.5.** *Ekran edytora emacs podzielony poziomo za pomocą kombinacji C-x2. W dolnym okienku uruchomiono powłokę poprzez wpisanie C-xo, a następnie M-x ansi-term. Można przełączać się między okienkami za pomocą kombinacji C-xo*

**Tworzenie kopii zapasowych plików przez program emacs**

Jeśli zaczniesz korzystać z edytora `emacs`, wkrótce zauważysz, że od czasu do czasu w Twoim katalogu domowym pojawiają się pliki o nazwach podobnych do tych, z którymi pracowałeś, ale z niewielkimi uzupełnieniami. Jeśli na przykład pracowałeś z plikiem *moj_plik.txt*, w tym samym katalogu możesz znaleźć plik *#moj_plik. txt#* albo *moj_plik.txt~*.

Są to „pliki bezpieczeństwa", utworzone przez edytor `emacs` w celu ochrony Twoich danych.

Kiedy zamykasz program bez zapisywania pliku, `emacs` zapisuje go i tak, pod tą samą nazwą, ale ze znakami krzyżyka (#) z przodu i z tyłu, na wypadek, gdybyś później postanowił przywrócić dokonane zmiany.

Kiedy dokonujesz zmian w istniejącym pliku i zapisujesz je, `emacs` zachowuje pierwotny (niezmodyfikowany) plik pod tą samą nazwą, uzupełniając ją o znak tyldy (~), na wypadek, gdybyś chciał później wrócić do pierwotnej wersji pliku.

Jeśli nie potrzebujesz automatycznie zapisanych plików ani ich kopii zapasowych, możesz je usunąć poleceniem `rm`, choć przedtem warto wczytać je do edytora `emacs` i sprawdzić, czy nie zawierają ważnych danych.

# Efektywne zarządzanie grupami plików

Kiedy zaczniesz regularnie pracować z większą liczbą plików oraz z edytorami, takimi jak `vi` i `emacs`, zbudujesz bibliotekę danych, raportów i niedokończonych plików.

Polecenia takie jak `mv`, `cp`, `rm` i `ls` przydają się do organizowania plików. Kiedy jednak Twoja kolekcja plików się powiększy, będziesz potrzebował dodatkowych narzędzi, dostarczanych przez powłokę oraz kilku bardziej zaawansowanych poleceń. Możesz grupować pliki, aby szybciej i wydajniej je wyszukiwać. W kolejnych podrozdziałach wyjaśnimy, jak grupować pliki w Linuksie, aby sprawniej nimi zarządzać.

## Grupowanie plików w wierszu poleceń

W rozdziale 4. nauczyłeś się kilku podstawowych metod zarządzania plikami, polegających na wpisywaniu poleceń i nazw plików — po jednej lub po dwie — za monitem powłoki. Czasem warto jednak mieć możliwość odwołania się do wielu plików jednocześnie, bez wpisywania wszystkich nazw. Powłoka dostarcza narzędzi, które umożliwiają zgrupowanie podobnych nazw plików, dzięki czemu nie trzeba wpisywać ich pojedynczo.

Zilustrujmy to na przykładzie. Przypuśćmy, że chcesz zapisać pliki *moj_plik_vi.txt*, *moj_nowy_plik_vi.txt* oraz *moj_plik_emacsa.txt* w katalogu o nazwie *pierwsze_pliki*. Korzystając z umiejętności nabytych w podrozdziale „Manipulowanie plikami i katalogami" w rozdziale 4. najpierw tworzysz katalog za pomocą polecenia `mkdir`, a następnie przenosisz poszczególne pliki za pomocą polecenia `mv`:

```
[ty@fedora2 ty]$ mkdir pierwsze_pliki
[ty@fedora2 ty]$ mv moj_plik_vi.txt moj_nowy_plik_vi.txt moj_plik_emacsa.txt
pierwsze_pliki
[ty@fedora2 ty]$
```

To bardzo dużo pisania jak na zwykłe przenoszenie trzech plików tekstowych w dogodniejsze miejsce. Wydaje się, że powinien istnieć jakiś łatwiejszy sposób — i on istnieje. W wierszu poleceń gwiazdka (*) jest jednym ze znaków specjalnych, które służą do **rozwijania nazw plików** — logicznego grupowania plików w taki sposób, żeby nie trzeba było wpisywać poszczególnych nazw i żeby dało się manipulować wszystkimi plikami jednocześnie. Rozwijanie nazw plików to po części nauka, a po części sztuka; korzystasz z narzędzi do dopasowywania wzorców, aby zebrać pliki w grupę, a następnie przekazujesz tę grupę powłoce. Brzmi to skomplikowanie, ale rozwijanie nazw plików jest w rzeczywistości bardzo proste. Oto przykład zastosowania tej techniki w opisanym przed chwilą zadaniu. Spróbuj użyć poniższego polecenia `mv`:

```
[ty@fedora2 ty]$ mv *.txt pierwsze_pliki
```

Sprawdźmy, czy polecenie odniosło pożądany skutek:

```
[ty@fedora2 ty]$ ls
pierwsze_pliki
[ty@fedora2 ty]$ ls pierwsze_pliki
moj_nowy_plik_vi.txt moj_plik_emacsa.txt moj_plik_vi.txt
[ty@fedora2 ty]$
```

Najwyraźniej nowe, uproszczone polecenie zadziałało poprawnie. Fraza `*.txt` zgrupowała wszystkie pliki, których nazwy są zakończone czterema znakami `.txt`.

Zanim przejdziemy do innych przykładów, przyjrzyj się tabeli 5.3, w której wymienione są znaki najczęściej używane do rozwijania nazw plików.

**Tabela 5.3.** *Znaki używane do dopasowywania wzorców i ich działanie*

| Wzorzec | Opis |
|---|---|
| * | Dopasowuje wszystkie znaki, niezależnie od ich liczby. |
| ? | Dopasowuje dowolny pojedynczy znak. |
| [a-b] (zakres) | Dopasowuje pojedynczy znak we wskazanym zakresie; na przykład [A-Z] dopasowuje litery X lub P, ale nie cyfrę 9 albo literę a. |
| [AaBbCc] (lista) | Dopasowuje dowolny znak spośród wymienionych na liście. |

**Nieoczekiwane działanie zakresów**

Litery w komputerze są reprezentowane wewnętrznie za pomocą specjalnego kodu o nazwie ASCII, którego układ odbiega od naszych przyzwyczajeń, więc niektóre wzorce mogą odnosić skutek inny od zamierzonego.

Na przykład wzorzec [a-Z] nie dopasuje niczego, natomiast [A-z] dopasuje kilka znaków, które nie są wielkimi ani małymi literami. Z tego powodu powinieneś używać tylko takich zakresów, w których oba znaki są wielkimi literami, małymi literami albo cyframi.

Kilka przykładów pomoże Ci lepiej zrozumieć, co oznaczają te wzorce i jak za ich pomocą można grupować pliki w Linuksie. Utwórzmy tabelę przykładów. Przypuśćmy, że chcesz usunąć grupę plików ze swojego katalogu domowego. Tabela 5.4 przedstawia przykładowe argumenty polecenia `rm` oraz pliki, które zostałyby dopasowane (albo nie) przez każdy z nich.

**Tabela 5.4.** *Przykładowe polecenia korzystające z rozwijania nazw plików i ich skutki*

| Polecenie | Skutek |
|---|---|
| `rm *.txt` | Usunie każdy plik z rozszerzeniem `.txt` — na przykład *a.txt*, *b.txt*, *halo.txt* albo *wszyscy_sa_super.txt*, ale nie *rower.gif*, *rachunki.xls* albo *stary_txt.* |
| `rm a*jpg` | Usunie wszystkie pliki, których nazwa zaczyna się od litery `a`, a kończy trzema literami `jpg` — na przykład *a.jpg*, *ananas.jpg*, *andromeda.jpg* i *anastazja_jpg*, ale nie *kajak.jpg*, *plik.txt*, *malpy.gif* albo *makarony_lista.* |
| `rm k*a.?if` | Usunie pliki *kuchnia.gif*, *kura.tif*, *ka.zif* i *kawiarnia..if*, ale nie *kariera.if*, *kasyno.gif* albo *fontanna.tif.* |
| `rm k[aoi]t.*` | Usunie pliki *kot.gif*, *kit.txt*, *kat.jpg*, *kit.do.okien*, *kot.i.pies.txt* oraz *kat.astrofa*, ale nie *ket.gif*, *kotek.txt* albo *kot.* |
| `rm [a-f]*` | Usunie pliki *ananas.txt*, *banan.jpg*, *czarny.kot*, *dynia.i.melon*, *energia.mp3* i *faworyci_fanatyka*, ale nie *gildia.txt*, *xawery.jpg* albo *Barbie.gif.* |
| `rm *` | Usunie każdy plik w bieżącym katalogu roboczym. W przypadku dodania opcji `-rf` usunie każdy plik i każdy katalog w bieżącym katalogu roboczym (czy pamiętasz opcje `-r` i `-f` z rozdziału 4.?). |

**Postępowanie z wzorcami**

Zachowaj ostrożność, kiedy grupujesz pliki za pomocą rozwijania nazw; nieprzemyślane wzorce mogą przynieść niepożądane skutki. Kiedy na przykład usuwasz pliki za pomocą poleceń pokazanych w tabeli 5.4, Linux przed usunięciem wszystkich plików pasujących do podanego wzorca nie ostrzega Cię ani nie pyta, czy jesteś tego pewien.

Rozwijania plików możesz używać w wielu okolicznościach, aby ułatwić sobie pracę na konsoli poprzez zmniejszanie liczby wpisywanych znaków przy zarządzaniu dużymi grupami plików. Na koniec spróbujemy przenieść pliki utworzone w edytorze `vi` z powrotem do katalogu domowego, a plik utworzony w edytorze `emacs` pozostawimy w katalogu *pierwsze_pliki*:

```
[ty@fedora2 ty]$ mv pierwsze_pliki/*vi* .
[ty@fedora2 ty]$ ls
moj_nowy_plik_vi.txt  moj_plik_vi.txt  pierwsze_pliki
[ty@fedora2 ty]$ ls pierwsze_pliki
moj_plik_emacsa.txt
[ty@fedora2 ty]$
```

# Zapobieganie rozwijaniu nazw plików

Czasem rozwijanie nazw plików jest niepożądane — na przykład wtedy, kiedy chcesz użyć znaków specjalnych (takich jak gwiazdka albo znak zapytania) w nazwie pliku albo jako argumentu polecenia.

Przypuśćmy, że chcesz utworzyć w swoim katalogu domowym katalog o nazwie **nowy** i zapisać w nim kilka nowych plików, nad którymi ostatnio pracowałeś. W Linuksie nazwy plików i katalogów mogą zawierać gwiazdki. Kiedy jednak spróbujesz utworzyć taki katalog, otrzymasz następujący komunikat o błędzie:

```
[ty@fedora2 ty]$ mkdir *nowy*
mkdir: `moj_nowy_plik_vi.txt' istnieje, ale nie jest katalogiem
[ty@fedora2 ty]$
```

Powłoka zinterpretowała argument *nowy* jako wzorzec i spróbowała dopasować go do plików w bieżącym katalogu roboczym. Pech chciał, że powłoka znalazła pasujący plik — wzorzec *nowy* pasuje do nazwy *moj_nowy_plik_vi.txt*, więc powłoka zadziałała tak, jakbyś wprowadził polecenie:

```
[ty@fedora2 ty]$ mkdir moj_nowy_plik_vi.txt
mkdir: `moj_nowy_plik_vi.txt' istnieje, ale nie jest katalogiem
[ty@fedora2 ty]$
```

W takich przypadkach chciałbyś, żeby powłoka traktowała wpisaną frazę jak zwykły tekst, a nie jak wzorzec. W tym celu musisz **przytoczyć** tekst (czyli ująć go w cudzysłów). Możesz to zrobić za pomocą pojedynczego lub podwójnego cudzysłowu (jest między nimi pewna różnica, o której będzie mowa w dalszych rozdziałach). Póki co, użyj pojedynczego cudzysłowu, aby utworzyć katalog *nowy* w swoim katalogu macierzystym:

```
[ty@fedora2 ty]$ mkdir '*nowy*'
[ty@fedora2 ty]$ ls
moj_nowy_plik_vi.txt  moj_plik_vi.txt  pierwsze_pliki  *nowy*
```

Udało ci się utworzyć katalog *nowy* dzięki ujęciu w cudzysłów frazy, która inaczej zostałaby potraktowana jako wzorzec do rozwijania nazw plików.

# Szybkie wyszukiwanie plików i katalogów

Dowiedziałeś się, jak wyświetlić listę plików za pomocą polecenia `ls` i jak zmienić katalog roboczy za pomocą polecenia `cd`. Jeśli jednak chciałbyś wyszukać konkretny plik tylko za pomocą poleceń `ls` i `cd`, musiałbyś wędrować po całym drzewie katalogu domowego, wyświetlać zawartość każdego katalogu, katalogów w tych katalogach i tak dalej, dopóki nie znalazłbyś żądanego pliku. Polecenia `ls` i `cd` sprawdzają się w przypadku zbioru dwóch czy trzech plików, ale jeśli zbiór liczy sto plików? Albo tysiąc? Prędzej czy później wszystkie nazwy plików zaczynają wyglądać jednakowo.

Zamiast tracić cały dzień na ręczne wyszukiwanie plików, użyj poleceń `find` i `locate`, aby odnaleźć zagubione pliki szybko i bez wysiłku.

# Szukanie plików za pomocą polecenia find

Polecenie `find` przeszukuje całe drzewa katalogów albo listy drzew katalogów pod kątem określonych nazw plików albo nazw pasujących do podanego wzorca. Oto składnia polecenia `find`:

```
find drzewo1 [drzewo2 ...] -name nazwapliku -print
```

Polecenie `find` wywołane w taki sposób szuka pliku o określonej nazwie w podanych drzewach katalogów. Możesz na przykład wyszukać plik *moj_plik_emacsa.txt* w swoim katalogu domowym:

```
[ty@fedora2 ty]$ find ~ -name moj_plik_emacsa.txt -print
/home/ty/pierwsze_pliki/moj_plik_emacsa.txt
[ty@fedora2 ty]$
```

**Działanie znaku tyldy**

Pamiętaj, że znak tyldy (~) jest dla powłoki odpowiednikiem katalogu */home/ty*.

Polecenie `find` szybko znalazło plik *moj_plik_emacsa.txt*. Znajduje się on w katalogu */home/ty/pierwsze_pliki*. W podobny sposób możesz wyszukać w swoim katalogu domowym wszystkie pliki, których nazwy kończą się rozszerzeniem `.txt`, podając wzorzec podobny do używanego podczas rozwijania nazw plików. Musisz jednak ująć wzorzec w cudzysłów, bo w przeciwnym razie powłoka dopasuje wzorzec do plików tekstowych w Twoim katalogu domowym, zanim polecenie `find` będzie miało szansę go zinterpretować:

```
[ty@fedora2 ty]$ find /home/ty -name '*.txt' -print
/home/ty/pierwsze_pliki/moj_plik_emacsa.txt
/home/ty/moj_plik_vi.txt
/home/ty/moj_nowy_plik_vi.txt
[ty@fedora2 ty]$
```

Polecenie `find` przeszukało Twój katalog domowy i znalazło trzy pliki o nazwie zakończonej na `.txt`. Wyświetlone zostały pełne ścieżki do tych plików.

# Przeszukiwanie całego systemu plików za pomocą polecenia locate

Czasem zdarza się, że chcesz wyszukać konkretny plik w *całym* systemie plików Linuksa. W takich przypadkach polecenie `locate` często działa lepiej niż `find`. Wiesz już, jak przeszukać cały system plików za pomocą polecenia `find`; przypuśćmy teraz, że chcesz znaleźć wszystkie obrazy JPEG w systemie plików Linuksa. Możesz to zrobić za pomocą polecenia:

```
[ty@fedora2 ty]$ find / -name '*.jpg' -print
[...]
```

Przekonasz się jednak, że tego rodzaju wyszukiwanie bywa czasochłonne, ponieważ polecenie `find` zagląda do każdego zakątka dysku twardego, katalog po katalogu, szukając plików o nazwach zakończonych na `.jpg`. Na ekranie pojawi się też mnóstwo komunikatów o błędach, ponieważ polecenie `find` napotka wiele miejsc, do których — jako zwykły użytkownik — nie masz dostępu. Polecenie `locate` działa w takich przypadkach wydajniej, ponieważ odwołuje się do bazy danych, która indeksuje wszystkie pliki w systemie. Używanie polecenia `locate` jest proste — podaj mu po prostu argument w postaci tekstu, który chcesz wyszukać:

```
[ty@fedora2 ty]$ locate '*.jpg'
/usr/share/nautilus/patterns/stucco.jpg
/usr/share/nautilus/patterns/dark-gnome.jpg
/usr/share/nautilus/patterns/burlap.jpg
/usr/share/nautilus/patterns/chalk.jpg
/usr/share/gimp/2.0/scripts/images/texture1.jpg
/usr/share/gimp/2.0/scripts/images/texture3.jpg
/usr/share/gimp/2.0/scripts/images/texture.jpg
/usr/share/gimp/2.0/scripts/images/beavis.jpg
/usr/share/gimp/2.0/scripts/images/texture2.jpg
/usr/share/gtk-2.0/demo/background.jpg
[...]
/usr/games/chromium/data/doc/images/useItem00.jpg
/usr/local/games/ut2003_demo/Web/images/h_logo.jpg
/usr/local/games/ut2003_demo/Web/images/h_rgfx.jpg
/usr/local/games/enemy-territory/Docs/Help/Tech Help/vendor/gamelogo.jpg
/usr/local/games/enemy-territory/Docs/Help/Tech Help/vendor/redstripe.jpg
/usr/local/games/enemy-territory/Docs/Help/Tech Help/vendor/Activision.jpg
[ty@fedora2 ty]$
```

**Wstrzymywanie wyświetlanie wyników**

Wyniki polecenia `locate` prawdopodobnie przewiną się poza ekran. Możesz wstrzymać wyświetlanie wyników polecenia `locate` (i innych poleceń), dołączając na końcu polecenia tekst `|more`:

```
locate '.*jpg' |more
```

Polecenie to będzie wyświetlać wyniki strona po stronie — naciśnięcie klawisza spacji spowoduje pokazanie następnego ekranu informacji.

Technika ta nosi nazwę **łączenia potokowego** i zostanie opisana w dalszym podrozdziale, zatytułowanym „Łączenie poleceń za pomocą potoków”.

Choć rzeczywista lista rysunków zależy od tego, jakie środowiska pulpitowe i inne pakiety oprogramowania zainstalowałeś w rozdziale 2. („Instalowanie systemu Fedora”), niemal natychmiast otrzymasz długą listę pasujących plików.

Baza danych używana przez polecenie `locate` jest aktualizowana raz dziennie, kiedy system wykonuje polecenie `updatedb`. Dane wyświetlane przez polecenie `locate` czasem nie uwzględniają pracy, którą wykonałeś w ciągu ostatnich kilku godzin. Jednakże wykorzystanie bazy danych umożliwia bardzo szybkie wyszukiwanie wielu plików na całym dysku.

# Zapisywanie listy znalezionych plików

Czasem warto zapisać wyniki wyszukiwania. Powłoka umożliwia zapisanie wyników poleceń, takich jak `find` i `locate`, przez *przekierowanie* ich wyjścia do pliku. Możesz następnie wczytać ten plik do edytora, takiego jak `emacs` lub `vi`, aby je zmodyfikować, wydrukować lub wykorzystać w inny sposób. Aby przekierować do pliku **standardowe wyjście** polecenia (informacje, które polecenie wyświetla na konsoli, kiedy realizuje Twoje żądania), użyj znaku większości (>) i wpisz po nim nazwę docelowego pliku. Spróbuj teraz zapisać listę plików JPEG w swoim systemie w pliku o nazwie *moje_pliki_jpeg.txt*:

```
[ty@fedora2 ty]$ locate '*.jpg' > moje_pliki_jpeg.txt
[ty@fedora2 ty]$ ls -l
-rw-rw-r--  1 ty ty 50990 lip 27 14:55 moje_pliki_jpeg.txt
-rw-rw-r--  1 ty ty     6 lip 27 14:30 moj_nowy_plik_vi.txt
-rw-rw-r--  1 ty ty     6 lip 27 14:29 moj_plik_vi.txt
drwxrwxr-x  2 ty ty  4096 lip 27 14:39 *nowy*
drwxrwxr-x  2 ty ty  4096 lip 27 14:36 pierwsze_pliki
[ty@fedora2 ty]$
```

Jak widzisz, w Twoim katalogu domowym pojawił się nowy, całkiem spory plik o nazwie *moje_pliki_jpeg.txt*. Plik ten zawiera listę plików JPEG w systemie plików Linuksa, znalezionych przez polecenie `locate`. Jeśli chcesz obejrzeć plik, wczytaj go do edytora `emacs` lub `vi`.

Możesz przekierować wyjście niemal każdego polecenia dostępnego w powłoce. Dołączenie standardowego wyjścia polecenia do istniejącego pliku można zrealizować także za pomocą podwójnego znaku większości (>>). Aby na przykład dodać listę wszystkich plików GIF w systemie do pliku *moje_pliki_jpeg.txt*, wydaj polecenie:

```
[ty@fedora2 ty]$ locate '*.gif' >> moje_pliki_jpeg.txt
[ty@fedora2 ty]$
```

**Ostrożność podczas przekierowywania wyjścia poleceń**

Aby uniknąć przypadkowego nadpisania ważnych plików, zachowaj ostrożność podczas przekierowywania wyjścia poleceń do istniejącego pliku. Kiedy używasz znaku większości (>) w celu przekierowania wyjścia do pliku, który już istnieje, zawartość pierwotnego pliku zostaje nadpisana nowymi danymi. Pamiętaj więc, żeby używać podwójnego znaku większości (>>), aby nie nadpisać ważnych plików.

# Wyszukiwanie wzorców w plikach tekstowych

Utworzyłeś w swoim katalogu domowym długi plik tekstowy o nazwie *moje_pliki_jpeg.txt*, który zawiera listę wszystkich obrazów JPEG i GIF w całym systemie plików Linuksa. Przypuśćmy jednak, że chciałbyś zawęzić tę listę i znaleźć tylko obrazy piłek.

Mógłbyś wczytać plik *moje_pliki_jpeg.txt* do edytora `emacs` lub `vi` i przejrzeć listę w poszukiwaniu plików spełniających to kryterium, ale szybciej znajdziesz żądane obrazy dzięki przeszukaniu pliku *moje_pliki_jpeg.txt* za pomocą polecenia `grep`. Polecenie

grep służy właśnie do wyszukiwania określonych przez Ciebie słów lub wzorców w plikach tekstowych. Wywołuje się je w następujący sposób:

```
grep wzorzec plik1 [plik2 ...]
```

*Wzorzec* to łańcuch tekstowy lub wzorzec, który chcesz znaleźć, a *plik1*, *plik2* itd. to pliki, które należy przeszukać. Wiersze, które zawierają tekst pasujący do łańcucha lub wzorca, zostaną wypisane na standardowym wyjściu (to znaczy na konsoli). Spróbuj teraz wyszukać obrazy piłek (ang. *ball*):

```
[ty@fedora2 ty]$ grep ball moje_pliki_jpeg.txt
/usr/share/apps/klines/balls.jpg
/usr/lib/ooo-1.1/share/gallery/bullets/bluball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/gryball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/golfball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/redball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/poliball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/darkball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/ylwball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/orgball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/grnball.gif
 [...]
/usr/share/doc/HTML/en/kdevelop/reference/C/PROBLEMS/whiteball.gif
/usr/share/doc/HTML/en/kdevelop/reference/C/blueball.gif
/usr/local/apache2/icons/ball.red.gif
/usr/local/apache2/icons/ball.gray.gif
/var/www/icons/ball.red.gif
/var/www/icons/ball.gray.gif
[ty@fedora2 ty]$
```

Polecenie grep znalazło w pliku *moje_pliki_jpeg.txt* kilka nazw plików, które zawierają słowo ball. Polecenie grep to doskonałe narzędzie do wydobywania danych z długich list.

# Wyszukiwanie plików, które zawierają określone słowa

Za pomocą polecenia grep można również łatwo znaleźć pliki, które zawierają określone słowo. Na przykład w Twoim katalogu domowym obecnie znajdują się trzy pliki tekstowe: *moje_pliki_jpeg.txt*, *moj_nowy_plik_vi.txt* oraz *moj_plik_vi.txt*. Przypuśćmy, że chcesz się dowiedzieć, który z tych plików zawiera słowo Unix.

Polecenie grep z opcją -l (lista plików) pozwala szybko i łatwo uzyskać wyniki:

```
[ty@fedora2 ty]$ grep -l Unix *
moj_nowy_plik_vi.txt
moj_plik_vi.txt
[ty@fedora2 ty]$
```

W tym przypadku program grep poinformował, że dwa pliki w katalogu roboczym — *moj_nowy_plik_vi.txt* oraz *moj_plik_vi.txt* — zawierają słowo Unix.

**Uwaga na wielkość liter**

Kiedy używasz polecenia `grep -l`, przeprowadzasz wyszukiwanie z uwzględnieniem wielkości liter; aby polecenie `grep` znalazło w pliku pasujące słowo, jego litery muszą mieć taką samą wielkość, jak litery w kryterium wyszukiwania.

Aby polecenie `grep` przeprowadziło wyszukiwanie bez uwzględniania wielkości liter, oprócz opcji `-l` podaj opcję `-i`:

```
grep -il unix *
```

# Używanie wyników poleceń do złożonych zadań

Jedną z największych zalet wiersza poleceń Linuksa jest możliwość wiązania wielu poleceń i ich danych wyjściowych na wiele różnorodnych sposobów. Dzięki wiązaniu poleceń możesz wykonywać skomplikowane zadania, które składają się z wielu etapów albo wymagają specyficznego zawężenia danych wyjściowych.

Dwie najważniejsze techniki wiązania wielu poleceń to potoki, dzięki którym dane wyjściowe jednego polecenia mogą zostać wykorzystane jako dane wejściowego drugiego, oraz podstawianie poleceń, które pozwala zmodyfikować działanie jednego polecenia w zależności od danych wyjściowych drugiego.

## Łączenie poleceń za pomocą potoków

Czasem trzeba użyć danych wyjściowych jednego polecenia jako danych wejściowych drugiego. Służą do tego **potoki**, które w powłoce definiuje się za pomocą pionowej kreski (|).

Oto przykład. W poprzednim podrozdziale wyświetliłeś listę wszystkich obrazów piłek (w formacie JPEG i GIF), przechowywanych w systemie; utworzyłeś tę listę, przekierowując wyniki dwóch poleceń `locate` do pliku *moje_pliki_jpeg.txt*, a następnie szukając w tym pliku słowa `ball` za pomocą polecenia `grep`. Przypuśćmy, że teraz chcesz się dowiedzieć, czy w systemie plików Linuksa są jakieś obrazy piłek zapisane w formatach `.png` lub `.tif`. Czy nie byłoby wygodnie wysłać wyników polecenia `locate` bezpośrednio do polecenia `grep`, aby przeszukało ono dane „w locie”?

Możesz to zrobić za pomocą potoku (|).

```
[ty@fedora2 ty]$ locate '*.gif' '*.jpg' '*.tif' '*.png' | grep ball
/usr/lib/ooo-1.1/share/gallery/bullets/bluball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/gryball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/golfball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/redball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/poliball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/darkball.gif
```

```
/usr/lib/ooo-1.1/share/gallery/bullets/ylwball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/orgball.gif
/usr/lib/ooo-1.1/share/gallery/bullets/grnball.gif
/usr/lib/ooo-1.1/share/gallery/rulers/grnballs.gif
/usr/lib/ooo-1.1/share/gallery/rulers/gldballs.gif
/usr/lib/ooo-1.1/share/gallery/rulers/whtballs.gif
/usr/lib/ooo-1.1/share/gallery/rulers/blkballs.gif
/usr/lib/python2.3/site-packages/Ft/Share/Data/Icons/ball.red.gif
/usr/lib/python2.3/site-packages/Ft/Share/Data/Icons/ball.gray.gif
/usr/share/latex2html/icons/purpleball.gif
/usr/share/latex2html/icons/redball.gif
/usr/share/latex2html/icons/yellowball.gif
/usr/share/latex2html/icons/whiteball.gif
/usr/share/latex2html/icons/blueball.gif
/usr/share/latex2html/icons/orangeball.gif
/usr/share/latex2html/icons/pinkball.gif
/usr/share/latex2html/icons/greenball.gif
/usr/share/doc/gettext/javadoc1/images/cyan-ball.gif
/usr/share/doc/gettext/javadoc1/images/yellow-ball-small.gif
/usr/share/doc/gettext/javadoc1/images/magenta-ball.gif
/usr/share/doc/gettext/javadoc1/images/cyan-ball-small.gif
/usr/share/doc/gettext/javadoc1/images/green-ball.gif
 [...]
/usr/share/apps/kbounce/pics/ball0022.png
/usr/share/apps/kbounce/pics/ball0002.png
/usr/share/apps/kbounce/pics/ball0016.png
/usr/share/apps/kbounce/pics/ball0013.png
/usr/share/apps/kbounce/pics/ball0020.png
/usr/share/apps/kbounce/pics/ball0023.png
/usr/share/apps/kbounce/pics/ball0017.png
/usr/share/apps/kbounce/pics/ball0000.png
/usr/share/apps/kbounce/pics/ball0010.png
/usr/share/apps/kbounce/pics/ball0001.png
/usr/share/apps/kbounce/pics/ball0019.png
/usr/local/apache2/icons/ball.red.png
/usr/local/apache2/icons/ball.gray.png
/var/www/icons/ball.red.png
/var/www/icons/ball.gray.png
[ty@fedora2 ty]$
```

Utworzyłeś listę wszystkich obrazów piłek, przechowywanych w czterech różnych formatach w całym systemie plików Linuksa. Wyniki polecenia `locate` (któremu podałeś cztery argumenty) zostały przesłane bezpośrednio do polecenia `grep`, które przeszukało je pod kątem słowa `ball`.

Bardziej przyziemnym, ale równie użytecznym zastosowaniem potoków jest stronicowanie wyników poleceń. Spróbuj na przykład wpisać poniższe polecenie:

```
[ty@fedora2 ty]$ ls -l /usr/bin
[...]
```

Listing jest bardzo długi; znaczna jego część zniknie za górną krawędzią ekranu, zanim będziesz miał okazję obejrzeć wyniki. Jeśli jednak wyślesz wyniki tego polecenia przez potok do innego polecenia o nazwie `more`, problem zostanie rozwiązany. Polecenie `more`

to program stronicujący: wyświetla plik albo dane wejściowe strona po stronie, zatrzymując się i oczekując na naciśnięcie klawisza spacji przed pokazaniem następnego ekranu informacji. Spróbuj teraz połączyć wyjście polecenia `ls` z wejściem polecenia `more`:

```
[ty@fedora2 ty]$ ls -l /usr/bin | more
[...]
```

Listing nadal zawiera wiele informacji, ale teraz możesz oglądać je w dowolnie wybranym tempie. Nie przejmuj się, jeśli zastosowania potoków wydają Ci się niejasne; przyzwyczaisz się do nich, kiedy będziesz pracował z powłoką.

# Używanie wyników jednego polecenia jako argumentów drugiego

Innym zaawansowanym mechanizmem powłoki jest **podstawianie poleceń**. Dzięki podstawianiu poleceń wyniki jednego polecenia mogą zostać wykorzystane jako zbiór argumentów drugiego. Procedura ta umożliwia modyfikowanie działania drugiego polecenia w zależności od wyników pierwszego, co wpływa na wyniki drugiego polecenia i pozwala *dostosować* je do określonej sytuacji.

Przypuśćmy, że chcesz utworzyć katalog o nazwie *pliki_jpeg* i zgromadzić w nim wszystkie obrazy JPEG przechowywane w systemie, aby mieć do nich łatwy dostęp. Wiesz już, jak użyć polecenia `locate` w celu znalezienia nazw plików zakończonych na `.jpg` w całym systemie plików Linuksa. Dysponując tymi wynikami, możesz posłużyć się podstawianiem poleceń, aby zgromadzić pliki w jednym miejscu:

```
[ty@fedora2 ty]$ mkdir pliki_jpeg
[ty@fedora2 ty]$ ln -s $(locate '*.jpg') pliki_jpeg
[ty@fedora2 ty]$
```

Umieszczenie polecenia `locate '.jpg'` w nawiasie i poprzedzenie go znakiem dolara (`$`) sprawiło, że nazwy plików wygenerowane przez polecenie `locate` zostały potraktowane tak, jakby użytkownik wpisał je jedna po drugiej jako argumenty polecenia `ln -s` (które służy do tworzenia dowiązań symbolicznych).

**Dowiązania symboliczne**

Dowiązania symboliczne i polecenie `ln -s` omówiono w rozdziale 4.

Użyj polecenia `ls`, aby wyświetlić długi listing plików w katalogu *pliki_jpeg*. Przekonasz się, że utworzone zostały dowiązania symboliczne do każdego pliku `.jpg` w całym systemie; wszystkie obrazy JPEG są zgromadzone w jednym miejscu i masz do nich łatwy dostęp.

Podobnie jak w przypadku potoków, szeroka gama zastosowań podstawiania poleceń prawdopodobnie nie będzie od razu oczywista. Nie martw się tym — będziesz miał okazję użyć go ponownie w następnych rozdziałach.

**Dwa sposoby podstawiania poleceń**

Podstawienie poleceń w powłoce `bash` może zachodzić na dwa sposoby. Pierwszy, pokazany w powyższym przykładzie, polega na umieszczeniu podstawianego polecenia w nawiasie poprzedzonym znakiem dolara:

```
$(locate moj_plik)
```

Drugi, tradycyjny sposób podstawiania poleceń polega na umieszczeniu polecenia między znakami odwróconego apostrofu:

```
`locate moj_plik`
```

W tej książce korzystamy z mniej tradycyjnej metody, ponieważ odwrócone apostrofy często trudno odróżnić od zwykłych, co może prowadzić do nieporozumień.

# Sterowanie programami powłoki

W następnych rozdziałach będziemy uruchamiać z poziomu konsoli dość skomplikowane aplikacje. Zanim dotrzemy do bardziej zaawansowanych zagadnień, musisz poznać metody sterowania zadaniami, działającymi pod kontrolą powłoki.

Możesz zalogować się na kilku konsolach wirtualnych, aby jednocześnie uruchomić wiele pełnoekranowych aplikacji — na przykład edytor `emacs` na konsoli pierwszej, a `vi` na konsoli drugiej. Przełączanie się między konsolami wirtualnymi bywa jednak dezorientujące i niewygodne. Co więcej, kiedy nauczysz się logować zdalnie za pomocą programów `telnet` lub `ssh`, będziesz musiał używać wiersza poleceń, nie dysponując wieloma konsolami wirtualnymi.

Aby uniknąć związanych z tym problemów, musisz nauczyć się technik wstrzymywania i wznawiania wielu zadań z pojedynczego wiersza poleceń. Dzięki nim będziesz mógł przełączać aplikacje, przenosić zadania na pierwszy plan lub do tła oraz przerywać niepotrzebne zadania. Poniższe podrozdziały nauczą Cię, jak używać omawianych technik w celu usprawnienia pracy nad wieloma projektami na pojedynczej konsoli Linuksa.

## Przełączanie otwartych aplikacji

Przypuśćmy, że chcesz uruchomić jednocześnie edytory `emacs` i `vi` oraz przełączać się między nimi, nie zmieniając bieżącego stanu żadnej z tych aplikacji. Może pracujesz nad artykułem porównującym oba edytory, a może po prostu edytujesz plik w `vi`, a od czasu do czasu robisz sobie przerwę na grę `dunnet`, wspomnianą przy okazji omawiania edytora `emacs`.

Za pomocą kombinacji klawiszy *Ctrl* + *Z* możesz wstrzymać pracę bieżącej aplikacji, aby uruchomić nowy program. Polecenie `jobs` podaje listę wszystkich otwartych aplikacji.

Wczytaj plik *moj_plik_vi.txt*, utworzony wcześniej w tym rozdziale, do edytora `vi`:

```
[ty@fedora2 ty]$ vi moj_plik_vi.txt
```

Kiedy na ekranie pojawi się edytor `vi`, monit poleceń zniknie i nie będziesz miał do niego dostępu. Możesz teraz edytować plik, dokonując w nim niezbędnych zmian. Co jednak zrobić, gdy przyjdzie czas na sesję gry `dunnet`?

Wstrzymaj proces `vi`, naciskając klawisze *Ctrl* + *Z*:

```
[1]+ Stopped                 vim moj_plik_vi.txt
[ty@fedora2 ty]$
```

Proces `vi` został wstrzymany. Zwróć uwagę na liczbę 1 w nawiasach kwadratowych; jest to **numer zadania** wstrzymanego procesu `vi`. Teraz możesz uruchomić edytor `emacs` i wpisać polecenie `M-x dunnet`, aby wczytać grę `dunnet`:

```
[ty@fedora2 ty]$ emacs
```

Po chwili relaksu trzeba będzie wrócić do dokumentu w edytorze `vi`; nie można grać w nieskończoność. Nie chcesz jednak zamykać edytora `emacs` i tracić bieżącej gry. Możesz wstrzymać proces `emacs` w taki sam sposób, w jaki wcześniej wstrzymałeś proces `vi` — naciskając klawisze *Ctrl* + *Z*:

```
[2]+ Stopped                 vim moj_plik_vi.txt
[ty@fedora2 ty]$
```

Teraz na liście zadań masz dwa wstrzymane edytory tekstu. Aby zobaczyć listę bieżących zadań, wpisz polecenie `jobs`:

```
[ty@fedora2 ty]$ jobs
[1]- Stopped                 vim moj_plik_vi.txt
[2]+ Stopped                 emacs
[ty@fedora2 ty]$
```

Zadania mogą pozostawać wstrzymane dowolnie długo; możesz kontynuować pracę w powłoce i wykonywać inne zadania, a `vi` i `emacs` będą gotowe do wznowienia pracy dokładnie w tym punkcie, w którym je zostawiłeś.

**Przerywanie działającego zadania**

Jeśli chcesz przerwać, a nie tylko wstrzymać działające zadanie, naciśnij klawisze *Ctrl + C*. Czasem kombinacja ta nie zadziała. Nie uda Ci się na przykład przerwać w ten sposób pracy edytora `emacs` ani `vi`, choć będziesz mógł zatrzymać wyświetlanie szczególnie długiego listingu polecenia `locate`.

# Wznawianie zadania za pomocą polecenia fg

Aby wrócić do edytowania dokumentu w programie `vi`, użyj polecenia `fg`, przekazując mu argument w postaci znaku procentu (`%`) oraz właściwego numeru zadania:

```
[ty@fedora2 ty]$ fg %1
```

Jak pokazało polecenie `jobs`, proces `vi` ma numer zadania 1. Kiedy wpiszesz polecenie `fg` (od ang. *foreground* — pierwszy plan), edytor `vi` wróci na ekran dokładnie w takim samym stanie, w jakim był, kiedy wstrzymywałeś jego działanie.

Możesz powtarzać tę procedurę tak często, jak to jest potrzebne, wstrzymując i wznawiając pracę programów w dowolnej kolejności i kombinacji.

# Uruchamianie zadania w tle za pomocą polecenia bg

Niektóre polecenia Linuksa wykonują się dość długo, zwłaszcza w starszych systemach. W takich przypadkach powłoka oferuje idealne rozwiązanie. Polecenia, które mogą działać bez interwencji użytkownika, można uruchomić **w tle** za pomocą polecenia `bg`. Po wydaniu tego polecenia możesz wrócić do pracy w innej otwartej aplikacji.

Przypuśćmy, że podczas edycji dokumentu w `vi` i gry w programie `emacs` postanawiasz wygenerować listę wszystkich plików, które może wyszukać polecenie `locate`. Aby zapisać tę listę w pliku o nazwie *lista_plikow.txt*, musisz najpierw wstrzymać działanie edytora za pomocą kombinacji klawiszy *Ctrl* + *Z*, a następnie wprowadzić poniższe polecenie:

```
[ty@fedora2 ty]$ locate '*' >lista_plikow.txt
```

Polecenie `locate` otrzymuje wzorzec, który pasuje do każdego pliku w jego bazie danych. Wyjście zostało przekierowane do pliku *lista_plikow.txt*. Zadanie jest nieskomplikowane, ale w większości komputerów jego wykonanie zajmie sporo czasu. Wstrzymajmy je więc i przenieśmy w tło. Naciśnij klawisze *Ctrl* + *Z*, aby wstrzymać działanie polecenia:

```
[3]+ Stopped       locate '*' >lista_plikow.txt
[ty@fedora2 ty]$
```

Zauważ, że polecenie otrzymało numer zadania 3 i że na ekranie ponownie pojawił się monit poleceń. Polecenie jest teraz wstrzymane; jeśli go nie wznowimy, nigdy nie zakończy pracy. Aby wznowić pracę polecenia w tle, użyj polecenia `bg` (od ang. *background* — tło):

```
[ty@fedora2 ty]$ bg %3
[3]+ locate '*' >lista_plikow.txt &
[ty@fedora2 ty]$
```

Polecenie działa teraz w tle; Linux będzie je wykonywał, dopóki zadanie nie zostanie zrealizowane. Ty tymczasem możesz wrócić do edytora albo do gry `dunnet` za pomocą polecenia `fg`. Aby najpierw zobaczyć zaktualizowaną listę zadań, wpisz polecenie `jobs`:

```
[ty@fedora2 ty]$ jobs
[1]- Stopped              vim moj_plik_vi.txt
[2]+ Stopped              emacs
[3]  Running              locate '*' >lista_plikow.txt &
[ty@fedora2 ty]$
```

W pewnym momencie komunikat o zakończeniu procesu działającego w tle wygląda tak:

```
[3]  Done         locate '*' >lista_plikow.txt &
```

Zakończony proces nie będzie się pojawiał na liście zadań wraz z działającymi lub wstrzymanymi procesami.

## Ostatnie uwagi o sterowaniu zadaniami

Istnieją jeszcze dwa polecenia związane ze sterowaniem zadaniami, które mogą uprościć Ci pracę w wierszu poleceń. Pierwsze z nich to `kill`. Polecenie `kill` z argumentem w postaci numeru zadania przerywa zadanie, które jest już niepotrzebne:

```
[ty@fedora2 ty]$ kill %2
[1]- Stopped              vim moj_plik_vi.txt
[2]+ Stopped              emacs
 [2] Stopped               emacs
[ty@fedora2 ty]$
```

Polecenie `kill` pomaga w usuwaniu zadań, natomiast znak ampersand (`&`) pomaga w ich tworzeniu. Jeśli chcesz od początku uruchomić zadanie w tle, zamiast naciskać *Ctrl* + *Z* i wpisywać polecenie `bg` wystarczy, że zakończysz wiersz polecenia znakiem ampersand. We wcześniejszym przykładzie mógłbyś użyć polecenia:

```
[ty@fedora2 ty]$ locate '*' > lista_plikow.txt &
[2] 14159
[ty@fedora2 ty]$
```

Pierwsza liczba zwracana po uruchomieniu zadania w tle to znany Ci już numer zadania. Druga liczba to systemowy numer procesu; szczegółowe informacje o systemowej tabeli procesów znajdziesz w rozdziale 18.

**Zadania działające po wylogowaniu użytkownika**

W zwykłych okolicznościach zadania, które działają w tle, są przerywane, kiedy się wylogowujesz — tak, jakbyś usunął je własnoręcznie.

Aby temu zapobiec i pozwolić na kontynuowanie zadania nawet po wylogowaniu, użyj polecenia `nohup`:

```
nohup %zadanie
```

Parametr *zadanie* zastąp numerem zadania, które ma działać nawet wówczas, kiedy się wylogujesz. Pamiętaj jednak, że jeśli komputera używa wiele osób, pozostawianie zadań działających w tle jest niekulturalne, ponieważ spowalnia to działanie komputera.

# Podsumowanie

W tym rozdziale nabyłeś umiejętności, dzięki którym z czasem zaczniesz sprawnie posługiwać się wierszem poleceń Linuksa. Utworzyłeś pliki tekstowe za pomocą dwóch popularnych edytorów i nauczyłeś się grupować pliki, aby efektywnie nimi zarządzać. Poznałeś też polecenia do szybkiego wyszukiwania nazw plików, typów plików oraz

słów lub wzorców w plikach. Nauczyłeś się dwóch sposobów łączenia poleceń, które pozwalają oszczędzić czas, wysiłek i miejsce na dysku. Dowiedziałeś się wreszcie, jak przełączać działające programy powłoki, nie zamykając ich ani nie korzystając z wielu konsoli.

Przy okazji nauczyłeś się wielu poleceń, kombinacji klawiszy i znaków specjalnych, używanych w wierszu poleceń powłoki. Pamiętaj, że:

- znak > przekierowuje standardowe wyjście do nowego pliku;
- znak >> dołącza standardowe wyjście do istniejącego pliku;
- znak & uruchamia proces w tle;
- kombinacja klawiszy *Ctrl* + *Z* wstrzymuje działający proces;
- polecenia `bg`, `fg` i `jobs` służą — odpowiednio — do wznawiania zadania w tle, wznawiania zadania na pierwszym planie oraz wyświetlania wszystkich bieżących zadań;
- polecenie `find` wyszukuje pliki w czasie rzeczywistym;
- polecenie `grep` szuka w pliku tekstowym albo w zbiorze plików określonego tekstu lub wzorca i zwraca nazwy plików, które go zawierają;
- polecenie `locate` wyszukuje lokalizację pliku za pomocą często aktualizowanej bazy danych;
- polecenie `more` wyświetla strona po stronie pliki utworzone w programach takich, jak `vi` lub `emacs`.

Poznałeś powłokę, polecenia i konsolę na tyle dobrze, że możesz przystąpić do rzeczywistej pracy. Dopóki w rozdziale 9. („Ujarzmianie potęgi powłoki”) nie zajmiemy się zaawansowanymi aspektami powłoki, nie będziesz już miał do czynienia z tego rodzaju technicznymi szczegółami — czas zająć się informacjami i aplikacjami.

# Pytania i odpowiedzi

**P**: Kiedy przekierowuję wyjście niektórych poleceń do pliku za pomocą znaku większości, niektóre komunikaty i tak pojawiają się na konsoli. Dlaczego?
**O**: Komunikaty te nie są wysyłane na standardowe wyjście, lecz na standardowe wyjście błędu; informują one użytkownika, że stało się coś nieoczekiwanego. Aby przekierować standardowe wyjście błędu, wpisz na końcu polecenia symbol `2>`, a po nim nazwę pliku, w którym mają być zapisywane komunikaty o błędach.

**P**: Czy da się po prostu odrzucić komunikaty kierowane na standardowe wyjście lub standardowe wyjście błędu — kiedy na przykład chcę, żeby polecenie działało „cicho”?
**O**: Tak, przekieruj jedno albo oba wyjścia do specjalnego pliku urządzenia */dev/null*, potocznie nazywanego „**koszem na bity**”. System odrzuca wszystkie dane kierowane do pliku */dev/null*.

# Warsztat

Warsztat ma pomóc Ci w przewidywaniu możliwych pytań, powtarzaniu nabytej wiedzy oraz przekładaniu teorii na praktykę.

## Quiz

1. Jakiego wzorca użyłbyś w poleceniu `rm`, aby usunąć pliki *cena_domu.txt*, *ocena.gif* i *samochod.cena*?
2. Jak wyszukałbyś wiersze, zawierające słowo `czerwone` w pliku *kolory_butow.txt*?
3. Jak wyświetliłbyś listę działających zadań edytora `emacs`? (**Wskazówka**: użyj potoku).
4. Jak szybko przeszukać cały system plików, aby odnaleźć plik *ReleaseNotes.html*?

## Odpowiedzi

1. `*cena*`
2. `grep czerwone kolory_butow.txt`
3. `jobs | grep emacs`
4. `locate ReleaseNotes.html`

## Ćwiczenia

1. Poświęć trochę czasu na przestudiowanie samouczków w edytorach `emacs` oraz `vi` i zapoznaj się z możliwościami obu programów.
2. Użyj poleceń `locate` i `more`, aby uzyskać listę wszystkich plików typu `.txt`, `.gif`, `.jpg` i `.gz` oraz wyświetlić ją na ekranie strona po stronie.
3. Uruchamiaj wiele zadań i kolejno je wstrzymuj, dopóki wyniki polecenia `jobs` nie wypełnią całego ekranu. Następnie zakończ lub usuń każde zadanie po kolei.
4. Uruchom edytor `emacs` i zagraj w `dunnet`.