100 sposobów na BSD

Autor: Dru Lavigne
Tłumaczenie: Marek Pętlicki (wstęp, rozdz. 1 – 4),
Grzegorz Werner (rozdz. 5 – 9)
ISBN: 83-7361-867-8
Tytuł oryginału: BSD Hacks
Format: B5, stron: 456

Przydatne porady dla wszystkich użytkowników systemów z rodziny BSD

- Skonfiguruj środowisko pracy i przyspiesz działanie systemu
- Wykorzystaj nieznane możliwości BSD
- Poznaj sposoby niestandardowej konfiguracji usług sieciowych
- Zabezpiecz system przed awariami i atakami z sieci

Systemy operacyjne z rodziny BSD coraz częściej konkurują z systemami linuksowymi. Twórcy BSD brali udział w tworzeniu Uniksa, a sporą część wyniesionych z tego doświadczeń wykorzystali, pisząc nowy, dostępny na licencji open source, system operacyjny. Różne odmiany systemów z rodziny BSD znane są z elastyczności i wysokiego poziomu bezpieczeństwa. Wykorzystuje się je w serwerach internetowych i stacjach roboczych. Firma Apple swój najnowszy system operacyjny – Mac OS X oparła na jądrze systemu BSD, co doskonale pokazuje uznanie, jakim cieszy się BSD w branży informatycznej.

Książka „100 sposobów na BSD" przeznaczona jest dla wszystkich użytkowników systemów BSD, niezależnie od stopnia zaawansowania. Opisuje sposoby radzenia sobie z przeróżnymi zadaniami i problemami, napotykanymi w codziennej pracy. Zawiera porady dotyczące wiersza poleceń, zabezpieczania systemów, śledzenia zmian w plikach i wykonywania kopii zapasowych. Po przeczytaniu tej książki każdy stanie się profesjonalnym użytkownikiem systemów BSD.

- Dostosowywanie środowiska tekstowego i graficznego
- Wyszukiwanie i modyfikowanie plików
- Dostęp do zasobów Windows bez użycia serwera
- Konfigurowanie procedur uruchamiania systemu
- Zarządzanie hasłami dostępowymi
- Tworzenie kopii zapasowych systemu
- Administrowanie połączeniami sieciowymi i serwerem pocztowym
- Zabezpieczanie systemu i detekcja włamań
- Tworzenie firewalla za pomocą pakietu PF

Przekonaj się, jak wiele tajemnic i ciekawych funkcji kryje w sobie system BSD

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

Spis treści

ROZDZIAŁ PIERWSZY

Dostosowanie środowiska użytkownika

Sposoby 1. – 12.

Użytkownicy systemów operacyjnych z rodziny Unix tworzonych na zasadach open source (*http://opensource.org*) są ciekawym typem osobników. Lubią zaglądać „pod maskę", aby sprawdzić, w jaki sposób to wszystko działa, z nadzieją odnalezienia ciekawych sposobów realizacji typowych zadań informatycznych. W skrócie taki sposób postępowania określa się mianem „hackowania".

Ta książka dotyczy przede wszystkim systemu BSD, lecz wiele technik można wykorzystać w dowolnym systemie operacyjnym open source. Każdy przedstawiony tu sposób stanowi po prostu demonstrację metody rozwiązania powszechnego problemu z wykorzystaniem nietypowego punktu widzenia i może być potraktowany jako punkt wyjścia do własnych, indywidualnych rozwiązań. Jeśli w danym systemie operacyjnym nie jest dostępne narzędzie zastosowane w konkretnej poradzie, można użyć innego istniejącego narzędzia lub stworzyć własne.

W tym rozdziale można znaleźć wiele informacji pozwalających na wykorzystanie systemu operacyjnego w jak najszerszym zakresie. Zapoznamy się z powłoką systemową i sposobami realizacji codziennych zadań za pomocą kilku naciśnięć klawiszy lub kliknięć myszą. Zostanie również przedstawionych trochę sposobów zabezpieczenia się przed błędami, które łatwo popełnić, pracując w wierszu poleceń. Przede wszystkim postaramy się jednak pokazać, że „hackowanie" systemów BSD może być przyjemne i ciekawe. Nadszedł zatem czas, aby ustawić swój fotel przed wybranym systemem operacyjnym i rozpocząć przygodę „hackera".

SPOSÓB 1.

Jak najpełniejsze wykorzystanie powłoki systemowej

Jak stać się demonem szybkości w powłoce systemowej.

W systemie BSD większość czasu spędza się, korzystając z powłoki, co może być uznane za zaletę lub wadę, w zależności od upodobań. Osoby przyzwyczajone do systemu Linux rozczarują się, gdy przekonają się, że domyślną powłoką zarówno konta użytkownika *root*, jak i zwykłych kont systemowych w systemach BSD nie jest `bash`.

Nie ma jednak powodu do rozpaczy. Domyślna powłoka `tcsh` systemu FreeBSD nie ustępuje powłoce `bash` pod względem możliwości przyspieszenia pracy za pomocą skrótów, które pozwalają uprościć wiele skomplikowanych zadań. Warto poświęcić kilka chwil na naukę tych sposobów, a z pewnością doceni się funkcjonalność `tcsh`. Poradę tę dedykuję użytkownikom nieprzyzwyczajonym do wiersza poleceń oraz słabo posługującym się klawiaturą. Unix może okazać się o wiele łatwiejszym narzędziem, niż się początkowo wydaje.

W systemach NetBSD oraz OpenBSD domyślną powloką jest tak zwana powłoka `C`. Nie musi to jednak być stary, dobry `tcsh` — często jest to uproszczona wersja `csh`, w której większość przedstawionych technik nie zadziała.

Powłokę `tcsh` można znaleźć w kolekcjach pakietów zarówno w NetBSD, jak i w OpenBSD.

Historia i uzupełnianie poleceń

Trudno byłoby mi pracować bez trzech klawiszy: strzałki w górę, strzałki w dół i klawisza *Tab*. Z tego powodu można mnie łatwo rozpoznać w tłumie, gdy mówię sama do siebie, narzekając w przypadku trafienia na system, w którym te klawisze nie są zdefiniowane w taki sposób, jaki lubię.

W powłoce `tcsh` strzałki w górę i w dół są wykorzystywane do przeglądania historii poleceń. Złota reguła informatyki mówi: „nie powinieneś być zmuszany do wpisywania polecenia więcej niż jeden raz". Gdy wystąpi konieczność powtórzenia polecenia, wystarczy nacisnąć klawisz strzałki w górę kilka razy, aż odszuka się odpowiednie polecenie. Następnie należy nacisnąć *Enter* i już można cieszyć się z zaoszczędzonych naciśnięć klawiszy. Jeśli przewinie się historię za daleko, można się wycofać, naciskając klawisz strzałki w dół.

Klawisz *Tab* został przewidziany zarówno dla osób piszących wolno, jak i popełniających błędy literowe. Obserwacja osoby wpisującej żmudnie wielowierszowe polecenie tylko po to, aby na końcu przekonać się, że po drodze popełniony został błąd, nie jest z reguły przyjemnym przeżyciem. Jeśli taka osoba nie zdaje sobie sprawy z tego, że ma do dyspozycji historię, i próbuje to polecenie wpisać od nowa, można rzeczywiście dostać nerwicy. Nic dziwnego, że wiele osób nie cierpi pracować w wierszu poleceń!

Tymczasem po naciśnięciu klawisza *Tab* włączane jest uzupełnianie. Oznacza to, że wystarczy wpisać kilka znaków polecenia lub nazwy pliku, nacisnąć *Tab*, a powłoka automatycznie uzupełni resztę. Jeśli jednak po naciśnięciu klawisza *Tab* rozlegnie się pisk, oznacza to, że nie można jednoznacznie rozstrzygnąć, o co chodzi. Załóżmy na przykład, że próbujemy wywołać polecenie `sockstat` i wpiszemy:

```
% so
```

Po naciśnięciu klawisza *Tab* głośniczek komputera zapiszczy, ponieważ od liter `so` rozpoczyna się wiele poleceń systemowych. Jeśli jednak dopiszemy więcej liter, wynik będzie lepszy:

```
% soc
```

Po naciśnięciu klawisza *Tab* powłoka prawidłowo rozwinie polecenie:

```
% sockstat
```

Edycja i nawigacja w wierszu poleceń

Powłoka obsługuje wiele innych kombinacji klawiszy, dzięki którym można przyspieszyć i uprościć pracę. Jeśli przed chwilą skończyliśmy pracę w programie edytora tekstów, za pomocą klawisza strzałki w górę przywrócimy polecenie uruchamiające edycję:

```
% vi dokumenty/biezace/bardzodluganazwa
```

Można bez trudu sprawdzić liczbę znaków w pliku, zastępując nazwę edytora odpowiednim poleceniem:

```
% wc dokumenty/biezace/bardzodluganazwa
```

Do nazwy `vi` można przejść, naciskając odpowiednio klawisz strzałki w lewo, lecz prościej będzie nacisnąć tylko raz klawisz *a*, przytrzymując klawisz *Ctrl*. Spowoduje to przesunięcie kursora na początek wiersza — można będzie od razu zmienić nazwę polecenia na odpowiednie. Dla uproszczenia warto zwrócić uwagę, że *a* jest pierwszą literą alfabetu, zatem kombinacja klawisza *a* z klawiszem *Ctrl* powoduje przejście do pierwszego znaku w wierszu poleceń powłoki `tcsh`.

Do wykonania polecenia nie jest konieczne wielokrotne naciskanie klawisza strzałki w prawo w celu ustawienia kursora na końcu wiersza. Gdy polecenie zostanie dostosowane do potrzeb, można nacisnąć *Enter* niezależnie od pozycji kursora.

Czasem pojawi się potrzeba przeniesienia kursora na koniec wiersza poleceń, aby na przykład dopisać tam odpowiednie opcje. Załóżmy, że konieczne będzie sprawdzenie liczby znaków w dwóch plikach, w tym w ostatnio edytowanym. Mamy więc nasze ostatnie polecenie (kursor jest ustawiony za wpisanymi dopiero znakami `wc`):

```
% wc dokumenty/biezace/bardzodluganazwa
```

Po przytrzymaniu klawisza *Ctrl* i naciśnięciu klawisza *e* kursor zostanie przeniesiony na koniec wiersza, można więc bez trudu dopisać resztę polecenia (ang. *end*, czyli koniec).

Jeśli w środku długiego polecenia zdecydujemy się, aby zrezygnować z wprowadzonych poprawek i zacząć od początku, wystarczy nacisnąć *Ctrl+u* (ang. *undo*, czyli cofnij zmiany).

Dla użytkowników systemów Cisco lub PIX IOS: wymienione wyżej sposoby działają w wierszu poleceń IOS.

Polecenie `cd` również zawiera wbudowane skróty. Najprostszy z nich polega na przejściu do katalogu użytkownika. W tym celu wystarczy wywołać polecenie `cd` bez parametrów:

```
% cd
```

To proste i wygodne. Czy można jednak przejść do poprzedniego katalogu? Załóżmy, że z katalogu */usr/share/doc/en_US. ISO8859-l/books/handbook* przejdziemy do */usr/X11R6/etc/X11*. Chcemy wrócić do pierwszego z tych katalogów. Z pewnością niewielu Czytelników chciałoby wprowadzać całą tę długą ścieżkę. Można oczywiście skorzystać z historii i odszukać tam polecenie, za pomocą którego przeszliśmy tam pierwszy raz, lecz prawie na pewno wyszukanie w historii wymaga naciśnięcia większej liczby klawiszy niż ręczne wpisanie ścieżki od nowa.

Na szczęście jest na to sposób. Wystarczy wpisać następujące polecenie:

```
% cd -
```

Ponowne wykonanie tego polecenia spowoduje powrót do drugiego z katalogów i tak na przemian. Zmiany można obserwować w tekście zachęty powłoki (ang. *prompt*). Jeśli tekst zachęty nie zawiera informacji o bieżącej ścieżce, nie stanowi to problemu — przejdziemy do tego za chwilę w sposobie 2. „Przydatne opcje pliku konfiguracyjnego powłoki tcsh".

Zaawansowane wykorzystanie historii

Potrafimy już poruszać się szybko w historii poleceń, warto jednak nieco pogłębić temat. Ile razy zdarzało się nam wyszukiwać ostatnie polecenie po to tylko, by je nieco poprawić? Za przykład niech posłuży następujący scenariusz zdarzeń.

W powyższych przykładach utworzyłam plik. Zamiast przywołać z historii polecenie z nazwą pliku by je zmodyfikować, w celu sprawdzenia liczby znaków w tym pliku można wywołać następujące polecenie:

```
% wc !$
  wc dokumenty/biezace/bardzodluganazwa
         19           97        620 dokumenty/biezace/bardzodluganazwa
```

Opcja `!$` wskazuje, że powłoka ma pobrać ostatni parametr z ostatnio wywołanego polecenia. Poleceniem tym było, jak pamiętamy:

```
% vi dokumenty/biezace/bardzodluganazwa
```

Z tego powodu `!$` zostało zastąpione ścieżką do pliku z poprzednio wywołanego polecenia.

Znak wykrzyknika (`!`) ma wiele innych pożytecznych zastosowań związanych z wykorzystaniem poprzednio wywołanych poleceń. Załóżmy, że przez ostatnią godzinę z mozołem wykonywaliśmy polecenia, których nazbierało się kilkanaście. Chcemy powtórzyć niektóre z nich. Można oczywiście wciskać klawisz strzałki w górę kilkadziesiąt razy, aż trafi się na odpowiednie polecenie. Można jednak uniknąć poszukiwań, ponieważ wyręczy nas w tym znak `!`.

Chcemy na przykład powtórzyć polecenie `mailstats`. W tym celu po znaku `!` należy wpisać odpowiednią liczbę znaków, aby jednoznacznie określić polecenie z historii:

```
$ !ma
```

Znak ! wybierze z historii ostatnie polecenie rozpoczynające się od zadanego ciągu znaków (ma). Jeśli jednak po poleceniu mailstats było wykonane polecenie man, powłoka tcsh powtórzy to ostatnie. Aby temu zapobiec, określmy polecenie bardziej wyraźnie:

```
% !mai
```

Jeśli chcemy uniknąć metody prób i błędów, warto sprawdzić historię, wykorzystując następujące polecenie:

```
% history
```

Dla szczególnie leniwych przewidziano następujący skrót:

```
% h
```

Każde polecenie w historii ma swój numer. Polecenie można wywołać, wpisując jego numer poprzedzony znakiem wykrzyknika. W naszym przykładzie chcemy powtórzyć polecenie mailstats:

```
% h
  165   16:51   mailstats
  166   16:51   sockstat
  167   16:52   telnet localhost 25
  168   16:54   man sendmail

% !165
```

Wyciszanie sygnalizacji błędu automatycznego uzupełniania poleceń

Ostatnia porada w tym podrozdziale będzie szczególnie przydatna dla tych, których denerwują piski pojawiające się w przypadku, gdy powłoka nie potrafi jednoznacznie uzupełnić polecenia. Przyda się też tym, którzy nie pamiętają prawidłowej pisowni polecenia lub nazwy pliku. Wpisujemy na przykład następujący fragment polecenia:

```
% ls -l b
```

Następnie przytrzymujemy klawisz *Ctrl* i wciskamy *d*:

```
backups/   bin/   book/   boring.jpg
% ls -l b
```

Pojawią się wszystkie kombinacje nazw plików lub katalogów rozpoczynających się od znaku b, a kursor powróci w to samo miejsce. Załóżmy, że chcemy sprawdzić szczegóły pliku *boring.jpg*, uzupełnimy więc odpowiednio znaki, aby funkcja automatycznego uzupełniania znalazła odpowiednią nazwę pliku:

```
% ls -l bor
```

Teraz wystarczy nacisnąć klawisz *Tab*.

Zobacz również:

- man tcsh.

SPOSÓB 2.

Przydatne opcje pliku konfiguracyjnego powłoki tcsh

Jak uczynić z powłoki łatwe w obsłudze narzędzie.

Gdy już postawi się pierwsze kroki „w powłoce", warto wykorzystać jej plik konfiguracyjny, aby usprawnić i uprzyjemnić pracę. Dobrym punktem początkowym może być tekst zachęty (ang. *prompt*).

Zaprzęganie do pracy tekstu zachęty

Domyślny tekst zachęty powłoki `tcsh` zawiera jedynie znak `%` (podczas logowania jako zwykły użytkownik) lub `nazwahosta#` (w przypadku zalogowania na koncie *root*). Pomaga to zorientować się, kiedy jesteśmy zalogowani jako superużytkownik, lecz powłoka potrafi znacznie więcej.

Każdy użytkownik w systemie, w tym superużytkownik, posiada w swoim katalogu plik *.cshrc*. Moje ustawienia tekstu zachęty zapisane w tym pliku są następujące:

```
dru@: grep prompt ~/.cshrc
if ($?prompt) then
        set prompt = "%B%n@%~%b: "
```

Nie jest to domyślne ustawienie systemowe, ale używam go już od kilku lat. Dopuszczalne opcje konfiguracji tekstu zachęty łatwo zrozumieć, lecz przydaje się tu lista dostępnych opcji z opisami. Są one objaśnione w podręczniku systemowym `man cshrc` — aby się do nich dokopać, wykonamy następujące polecenia:

```
dru@: man cshrc
/prompt may include
```

Po wywołaniu podręcznika systemowego posłużyliśmy się wyszukiwaniem (uaktywnianym klawiszem /). Poszukujemy fragmentu `prompt may include`, który występuje w części dotyczącej opcji tekstu zachęty.

Definicja wykorzystywanego przeze mnie tekstu zachęty jest następująca:

```
set prompt = "%B%n@%~%b: "
```

To dość zagmatwane — dla uproszczenia rozłóżmy tę definicję na składowe, które są przeanalizowane w tabeli 1.1.

Dzięki temu tekstowi zachęty nigdy nie ma problemu z identyfikacją nazwy użytkownika oraz bieżącego katalogu. W tekście zachęty można również wykorzystać nazwę komputera, do którego jesteśmy zalogowani (co może być przydatne w przypadku zdalnej administracji). W tym celu w definicji tekstu zachęty należy umieścić sekwencję `%M` lub `%m`.

Tabela 1.1. Omówienie elementów definicji przykładowego tekstu zachęty

Sekwencja znaków	Znaczenie
"	Początek definicji tekstu zachęty
%B	Włączenie pogrubienie
%n	Nazwa użytkownika
@	Znak separatora, uatrakcyjniający napis zachęty
%~	Wypisuje bieżący katalog. Sekwencja ta ma podobne działanie do %/ — z tą różnicą, że pełna ścieżka do katalogu użytkownika zostaje skrócona do znaku ~
%b	Wyłącza pogrubienie
:	Znak dwukropka którego używam by oddzielić tekst zachęty od tekstu wprowadzanego przez użytkownika
"	Koniec definicji tekstu zachęty

Przełączanie się na konto superużytkownika

Plik *.cshrc* superużytkownika (umieszczony w katalogu */root*) zawiera identyczną definicję napisu zachęty. To świetna okazja, aby ujawnić pewną ważną cechę polecenia `su`, wykorzystywanego do przełączania się pomiędzy użytkownikami. Załóżmy, że jesteśmy zalogowani jako użytkownik *dru*. Wiersz poleceń ma następującą postać:

```
dru@/usr/ports/net/ethereal:
```

Przełączamy się na konto superużytkownika. Zwróćmy uwagę na postać wiersza poleceń:

```
dru@/usr/ports/net/ethereal: su
Password:
dru@/usr/ports/net/ethereal:
```

Nic się nie zmieniło. Wywołanie polecenia `whoami` spowoduje jeszcze większe zamieszanie:

```
dru@/usr/ports/net/ethereal: whoami
dru
```

Prawdę ujawnia dopiero polecenie `id`:

```
dru@/usr/ports/net/ethereal: id
uid=0(root) gid=0(wheel) groups=0(wheel), 5(operator)
```

Okazuje się, że domyślne wywołanie polecenia `su` nie powoduje zalogowania się na konto *root*. Po prostu przyznawane są przywileje konta *root*, natomiast domyślne ustawienia oryginalnego użytkownika (w tym jego ustawienia powłoki) nadal obowiązują.

Aby naprawdę przełączyć się na konto superużytkownika, należy wywołać polecenie `su` z opcją `-l`:

```
dru@/usr/ports/net/ethereal: su -l
Password:
root@~: whoami
root
root@~: id
uid=0(root) gid=0(wheel) groups=0(wheel), 5(operator)
```

Zachęcam do eksperymentowania z różnymi kombinacjami sekwencji formatowania napisu zachęty, aż do uzyskania pożądanego wyniku. Można wykorzystywać wiele cech, w tym informacje o dacie i godzinie, jak również numery poleceń w historii **[Sposób 1.]**. Możliwe jest również nadawanie atrybutów graficznych, jak migotanie czy podkreślenie znaków tekstu zachęty.

Ustawianie zmiennych powłoki

Definicja tekstu zachęty jest przykładem zmiennej powłoki. Istnieje wiele innych zmiennych wpływających na funkcjonowanie powłoki. Definiuje się je w pliku *.cshrc*. Aby odszukać w podręczniku systemowym odpowiednie opcje, należy posłużyć się następującym sposobem:

```
dru@~: man cshrc
/variables described
```

Jak wskazuje nazwa, zmienne powłoki mają wpływ wyłącznie na polecenia wbudowane w powłokę. Nie należy mylić zmiennych powłoki ze zmiennymi środowiska, które mają wpływ na całe środowisko pracy i każde wywoływane polecenie.

Zmienne środowiska w pliku *~/.cshrc* można poznać po tym, że ich nazwy są zapisane wielkimi literami i są deklarowane za pomocą polecenia `setenv`. Nazwy zmiennych powłoki są natomiast zapisane małymi literami i są deklarowane za pomocą polecenia `set`.

Zmienną powłoki można włączyć za pomocą polecenia `set`, również wydawanego z wiersza poleceń. Aby usunąć taką zmienną, należy zastosować polecenie `unset`. Zmienne tego typu mają wpływ wyłącznie na bieżącą sesję i jej sesje potomne — można eksperymentować z różnymi ustawieniami bez obaw o to, że coś zostanie trwale popsute w systemie. W przypadku problemów wystarczy wylogować się, po czym ponownie zalogować i wszystko wróci do normy.

Jeśli jakąś zmienną zechcemy ustawić na stałe, należy wpisać jej deklarację w pliku *~/.cshrc* w pobliżu domyślnych deklaracji zmiennych powłoki (rozpoczynających się poleceniem `set`). Przyjrzyjmy się najciekawszym z nich.

Jeśli kogoś zaciekawił efekt kombinacji klawiszy *Ctr+d* z podrozdziału „Jak stać się demonem szybkości w powłoce systemowej" **[Sposób 1.],** z pewnością jeszcze bardziej polubi tę opcję:

```
set autolist
```

Teraz po naciśnięciu klawisza *Tab*, przy braku możliwości wykonania jednoznacznego dopełnienia, nie rozlegnie się pisk. Zamiast tego zostanie wypisana lista dostępnych możliwości. Nie ma potrzeby korzystania z kombinacji *Ctrl+d*!

Kolejna opcja może uratować przed niebezpieczeństwem utraty danych wskutek nieuwagi:

```
set rmstar
```

Działanie tej opcji sprawdzimy na utworzonych w tym celu katalogach i plikach:

```
dru@~: mkdir test
dru@~: cd test
dru@~/test: touch a b c d e
```

Usuńmy teraz pliki z katalogu *test*:

```
dru@~/test: rm *
Do you really want to delete all files? [n/y]
```

Tekst zachęty informuje o katalogu bieżącym, mam więc możliwość zastanowienia się, czy rzeczywiście usuwane przeze mnie pliki są tymi, których chcę się pozbyć.

Osoby podatne na popełnianie literówek powinny wziąć również pod uwagę następującą opcję:

```
set correct=all
```

Powłoka zareaguje na błędy literowe, sugerując prawidłową wersję:

```
dru@~: cd /urs/ports
CORRECT>cd /usr/ports  (y|n|e|a)?
```

Po naciśnięciu klawisza *y* (ang. *yes*) zaakceptujemy poprawioną pisownię polecenia i zostanie ono wykonane przez powłokę. Po naciśnięciu klawisza *n* (ang. *no*) zostanie uruchomione polecenie napisane błędnie. Naciśnięcie klawisza *e* (ang. *edit*) spowoduje powrót do edycji wiersza poleceń. W tym przypadku najlepiej będzie zatwierdzić poprawioną pisownię, ponieważ sugestia powłoki jest właściwa. Jeśli jednak w wyniku sugestii powłoki spanikujemy, możemy zrezygnować z dalszych działań, naciskając klawisz *a* (ang. *abort*), co spowoduje anulowanie operacji i przejście do pustego wiersza poleceń.

Bardziej leniwi mogą wykorzystać następującą opcję:

```
set implicitcd
```

Od tej pory nie będzie potrzeby wpisywania polecenia `cd`. Aby zmienić katalog bieżący, wystarczy wprowadzić nazwę katalogu i nacisnąć *Enter*.

SPOSÓB 3.

Definicja kombinacji klawiszy dla powłoki

Jak wytresować powłokę do wykonywania poleceń w odpowiedzi na naciśnięcie klawiszy.

Każdy zapewne miał okazję słyszeć pochwały użytkowników systemu Windows dotyczące zalet konfigurowania kombinacji klawiszy w aplikacjach. Zapewne niejeden z Czytelników sam miał okazję korzystać z klawiatur zawierających specjalizowane klawisze, wywołujące określone funkcje systemu. Jednak również w systemach Unix można tak skonfigurować system, aby w odpowiedzi na naciśnięcia klawiszy lub ich kombinacji były wykonywane określone operacje.

Jednym ze sposobów uzyskania takiej funkcjonalności jest wbudowane w powłokę `tcsh` polecenie `bindkey`. Jak sugeruje nazwa, polecenie to służy do kojarzenia operacji z określonymi kombinacjami klawiszy. Aby sprawdzić aktualnie zdefiniowane powiązania, wystarczy wywołać polecenie `bindkey` bez parametrów. Wynik działania tego polecenia zajmuje kilka stron ekranowych, zamieszczam jedynie niewielki ich wybór. Niektóre z tych skrótów są już znane z podrozdziału „Jak stać się demonem szybkości w powłoce systemowej" **[Sposób 1.]**.

```
Standard key bindings
"^A"          -> beginning-of-line
"^B"          -> backward-char
"^E"          -> end-of-line
"^F"          -> forward-char
"^L"          -> clear-screen
"^N"          -> down-history
"^P"          -> up-history
"^U"          -> kill-whole-line

Arrow key bindings
down          -> history-search-forward
up            -> history-search-backward
left          -> backward-char
right         -> forward-char
home          -> beginning-of-line
end           -> end-of-line
```

Znak ^ oznacza, że należy przytrzymać klawisz *Ctrl*. Na przykład kombinacja *Ctrl+l* spowoduje wyczyszczenie ekranu (operacja *clear-screen*), co jest odpowiednikiem polecenia `clear`. Nie ma znaczenia, czy w kombinacji klawiszy zostanie wykorzystana litera mała czy wielka.

Utworzenie dowiązania

Jedno z moich ulubionych dowiązań nie jest skonfigurowane w domyślnych ustawieniach powłoki. Chodzi o operację *complete-word-fwd*. Przed zdefiniowaniem dowiązania należy sprawdzić, które z kombinacji klawiszy są już zajęte:

```
dru@~: bindkey | grep undefined
"^G"            -> is undefined
"\3O5"          -> is undefined
"\3O7"          -> is undefined
<snip>
```

Choć operację można dowiązywać również do kodów numerycznych, nie uważam takiego rozwiązania za wygodne. Jednak dostępna kombinacja *Ctrl+g* doskonale nada się do tego celu. Sprawdźmy, co się stanie po zdefiniowaniu dowiązania:

```
dru@~: bindkey "^G" complete-word-fwd
```

Brak komunikatu po wywołaniu tego polecenia sygnalizuje, że operacja zakończyła się poprawnie. Gdy po wpisaniu `ls -l /etc/` zostanie naciśnięta kombinacja klawiszy *Ctrl+g*, kolejno zaczną uzupełniać się kolejne pozycje z katalogu */etc*:

```
ls -l /etc/COPYRIGHT
ls -l /etc/X11
ls -l /etc/aliases
ls -l /etc/amd.map
```

Dzięki temu można przejrzeć pasujące pozycje aż do momentu, gdy natrafimy na właściwą. Jeśli znamy pierwszą literę nazwy pliku z katalogu */etc*, możemy jeszcze przyspieszyć przeszukiwanie. Załóżmy, że poszukujemy pliku o nazwie rozpoczynającej się na literę *a*:

```
ls -l /etc/a
ls -l /etc/aliases
ls -l /etc/amd.map
ls -l /etc/apmd.conf
ls -l /etc/auth.conf
ls -l /etc/a
```

Gdy przejrzymy wszystkie dostępne dopasowania, powłoka przywróci oryginalnie wpisany fragment polecenia i rozlegnie się pisk ostrzegawczy.

Jeśli wolimy przełączać dopasowania wstecz, zamiast operacji *complete-word-fwd* należy dowiązać do kombinacji klawiszy operację *complete-word-back*.

Za pomocą polecenia `bindkey` można zdefiniować dowiązanie kombinacji klawiszy do dowolnej operacji zrozumiałej dla powłoki. Listę takich operacji można poznać, wykorzystując następującą technikę:

```
dru@~ man csh
/command is bound
```

Oczywiście listę zdefiniowanych powiązań wypisujemy, wywołując polecenie `bindkey` bez parametrów. Można również odczytać dowiązanie określonej kombinacji klawiszy. W tym celu należy podać poszukiwaną kombinację klawiszy jako jedyny parametr polecenia `bindkey`. Aby sprawdzić dowiązanie kombinacji *Ctrl+g*, posłużymy się następującym poleceniem:

```
dru@~: bindkey "^G"
"^G"     ->   complete-word-fwd
```

Określanie tekstów poleceń

Na szczęście nie jesteśmy ograniczeni wyłącznie do zdefiniowanej z góry listy operacji powłoki (dostępnej w podręczniku systemowym `man csh`). Znacznik `-s` polecenia `bindkey` pozwala zdefiniować tekst polecenia systemowego, które można powiązać z kombinacją klawiszy. Zdefiniujmy teraz powiązanie kombinacji *Ctrl+w* z wywołaniem tekstowej przeglądarki WWW lynx:

```
dru@~: bindkey -s "^W" "lynx\n"
```

Litera W została wybrana z powodu skojarzenia z WWW. Dlaczego po nazwie polecenia znalazła się sekwencja znaków `\n`? Otóż sygnalizuje ona powłoce, że ma zasymulować naciśnięcie klawisza *Enter*. Dzięki temu wciśnięcie kombinacji klawiszy *Ctrl+w* spowoduje natychmiastowe uruchomienie przeglądarki lynx.

Powyższe polecenie spowoduje zastąpienie domyślnego dowiązania kombinacji *Ctrl+w*. Można zatem zdefiniować własne dowiązania, lepiej przystosowane do indywidualnych potrzeb. Jeśli na przykład nie zdarza nam się wykorzystywać operacji obsługiwanej przez domyślne dowiązanie kombinacji *Ctrl+j*, można je z powodzeniem zmodyfikować zgodnie z własnymi potrzebami.

Dowiązań jest wiele, zatem przejrzenie całej listy zdefiniowanych kombinacji może być utrudnione. Jeśli jednak wystarczą jedynie kombinacje z klawiszem *Ctrl*, za pomocą następującego polecenia można skrócić listę dowiązań:

```
dru@~: bindkey | head -n 28
```

Podobnie jak w przypadku innych modyfikacji ustawień powłoki, eksperymenty z poleceniem `bindkey` proponuję przeprowadzać poprzez wywoływanie go bezpośrednio z wiersza poleceń. Jeśli powstaną problemy, zawsze będzie można wylogować się i zalogować ponownie. Gdy trafimy na dowiązanie, które wyda się użyteczne, należy skonfigurować je na stałe. W tym celu odpowiednie wywołanie polecenia `bindkey` zapisujemy w pliku *.cshrc*. Oto przykład:

```
dru@~:cp ~/.cshrc ~/.cshrc.orig
dru@~:echo 'bindkey "^G" complete-word-fwd' >> ~/.cshrc
```

Zwracam uwagę na konieczność wykonania kopii zapasowej pliku *.cshrc*, na wypadek, gdyby coś się nie udało. Kluczowy jest tutaj zastosowany w drugim poleceniu operator `>>`. Gdybym użyła operatora `>`, usunęłabym całą zawartość pliku `.cshrc`, zastępując ją jednym wywołaniem polecenia `bindkey`. Nie zalecam testowania operatora `>` z jakimkolwiek plikiem zawierającym dane, które chcemy zachować.

Przy okazji: polecenie `set noclobber` pozwala zapobiec omyłkowemu nadpisaniu pliku wskutek pominięcia jednego znaku `>` w operacji przekierowania do pliku wyjścia z polecenia. Gdy po jakimś czasie pojawi się następujący komunikat, będziemy mieli pewność, że właśnie udało się uniknąć utraty danych zapisanych w pliku:

```
.cshrc: File exists.
```

Zobacz również:

- `man tcsh`;
- „Przydatne opcje pliku konfiguracyjnego powłoki tcsh" **[Sposób 2.]**.

SPOSÓB 4.

Wykorzystanie dowiązań terminala oraz systemu X

Wykorzystanie możliwości terminala.

Nie tylko powłoka `tcsh` daje możliwość definiowania skrótów klawiszowych. Terminal systemu FreeBSD również udostępnia tę możliwość poprzez odpowiednie skonfigurowanie sterownika klawiatury za pomocą programu `kbdcontrol`. Niestety opcja ta nie jest dostępna w systemie NetBSD ani OpenBSD. W tych ostatnich można jednak wykorzystać odwzorowania klawiatury w systemie X, opisane w dalszej części tego podrozdziału.

Tworzenie tymczasowych odwzorowań

Zaczniemy od eksperymentów z tymczasowymi odwzorowaniami. Składnia definicji odwzorowania klawiatury za pomocą polecenia `kbdcontrol` jest następująca:

```
kbdcontrol -f kod_liczbowy "polecenie"
```

Tabela 1.2 zawiera listę dostępnych kodów wraz z objaśnieniami związanych z nimi kombinacji klawiszy.

Ostatnie trzy kombinacje klawiszy mogą nie być dostępne, w zależności od modelu posiadanej klawiatury. Moja klawiatura firmy Logitech posiada klawisz z logo Windows umieszczony obok lewego klawisza *Ctrl* — to właśnie lewy klawisz specjalny GUI. Drugi taki sam klawisz znajduje się obok prawego klawisza *Alt*; to jest właśnie prawy specjalny klawisz GUI. Obok niego znajduje się klawisz z logo przypominającym menu z ikoną wskaźnika myszy — jest to klawisz *Menu*.

Znamy już dostępne kody klawiszy, zdefiniujmy więc wywołanie programu lynx po naciśnięciu klawisza *Menu*:

```
% kbdcontrol -f 64 "lynx"
```

Polecenie musi być ujęte w cudzysłowach i musi znajdować się w ścieżce. Można co prawda podać je z pełną ścieżką, lecz wkrótce poznamy pewne niekorzystne ograniczenie związane z tą opcją.

Jeśli po wywołaniu powyższego polecenia naciśniemy klawisz *Menu* w terminalu, zostanie wpisane polecenie `lynx`. Wystarczy nacisnąć klawisz *Enter*, aby uruchomić przeglądarkę. Na początku wymóg potwierdzenia polecenia klawiszem *Enter* może wydać się nieco kłopotliwy, lecz z czasem można docenić tę cechę. Dzięki temu unikniemy na przykład wywołania niewłaściwego polecenia wskutek pomyłki, gdy zapomnimy, które kombinacje klawiszy zostały przypisane do poszczególnych poleceń.

Tabela 1.2. Kody klawiszy

Liczba	Kombinacja klawiszy
1,2,… 12	*F1*,*F2*,… *F12*
13,14,… 24	*Shift*+*F1*, *Shift*+*F2*,… *Shift*+*F12*
25, 26,… 36	*Ctrl*+*F1*, *Ctrl*+*F2*,… *Ctrl*+*F12*
37, 38,… 48	*Shift*+*Ctrl*+*F1*, *Shift*+*Ctrl*+*F2*,… *Shift*+*Ctrl*+*F12*
49	*Home*
50	Strzałka w górę
51	Page Up
52	– (minus) na klawiaturze numerycznej (wyłączony *Num Lock*)
53	Strzałka w lewo (działa również w edytorze)
54	*5* na klawiaturze numerycznej (wyłączony *Num Lock*)
55	Strzałka w prawo
56	+ (plus) na klawiaturze numerycznej (wyłączony *Num Lock*)
57	*End*
58	Strzałka w dół (ma wpływ na obsługę historii powłoki `csh`)
59	*Page Down*
60	*Ins*
61	*Del*
62	Lewy klawisz specjalny GUI (ikona Windows obok lewego klawisza *Ctrl*)
63	Prawy klawisz specjalny GUI (ikona Windows obok prawego klawisza *Alt*)
64	Menu (ikona menu obok prawego klawisza *Ctrl*)

Sprawdźmy, co się stanie, gdy nieco zmodyfikujemy wywołanie odwzorowania:

```
% kbdcontrol -f 64 "lynx www.google.pl"
kbdcontrol: function key string too long (18 > 16)
```

Definiując własne odwzorowania, należy pamiętać, że polecenie nie może być dłuższe niż 16 znaków. Poza tym ograniczeniem można zdefiniować zupełnie dowolne polecenia.

Dowiązania powłoki a dowiązania terminala

Zanim przejdziemy dalej, warto porównać dowiązania kombinacji klawiszy na poziomie powłoki, które poznaliśmy w podrozdziale „Definicja kombinacji klawiszy dla powłoki" **[Sposób 3.]**, z dowiązaniami kombinacji klawiszy przedstawionymi w tym podrozdziale.

Jedną z zalet polecenia `kbdcontrol` jest możliwość definicji dowiązań klawiszy w dowolnym terminalu, niezależnie od rodzaju powłoki. Druga zaleta polega na tym, że dowiązania można zdefiniować dla dowolnego klawisza na klawiaturze. Dowiązania kombinacji klawiszy na poziomie powłoki mogą być jednak nieco skomplikowane, jeśli zdecydujemy się wyjść poza schemat *Ctrl*+litera.

Odwzorowania na poziomie terminala posiadają ograniczenia, które nie dotyczą odwzorowań powłoki `tcsh`. W przypadku powłoki nie istnieje limit 16 znaków, dzięki czemu nie ma problemu z pełnymi ścieżkami do programów i plików. W łatwy sposób można też wymusić na powłoce wyręczenie nas w naciśnięciu klawisza *Enter*, aby natychmiast wywołać dowiązane polecenie.

Dowiązania na poziomie terminala mają zastosowanie wyłącznie w terminalu bieżącego użytkownika. Użytkownicy zalogowani na innych terminalach w systemie nie będą doświadczali modyfikacji odwzorowań. Jeśli jednak zmiany zostaną wprowadzone w pliku *rc.conf* (co jest możliwe wyłącznie z konta superużytkownika), będą one dotyczyły wszystkich użytkowników. Dowiązania odbywają się na poziomie terminala, więc dopóki praca odbywa się na tym samym terminalu, niczego nie zmieni nawet zalogowanie się na konto innego użytkownika (np. za pomocą polecenia `su`).

Inne uwagi dotyczące dowiązań

Decydując się na formę dowiązań klawiszy, należy wziąć pod uwagę kilka informacji. Użytkownicy stosujący `tcsh` i korzystający z historii **[Sposób 1.]** rozczarują się, gdy przedefiniują klawisze strzałek w górę i w dół. Modyfikacja znaczenia klawiszy strzałek w prawo i w lewo również może sprawić problemy użytkownikom wykorzystującym te klawisze do nawigacji, na przykład w edytorze. Jeśli z systemem FreeBSD pracujemy lokalnie, klawisze *F1 – F8* są wykorzystywane do przełączania się pomiędzy terminalami wirtualnymi, a *F9* obsługuje terminal trybu graficznego GUI. Klawisze *F10 – F12* są niewykorzystane.

Jeśli podczas eksperymentów zdarzy się zdefiniować szczególnie niewygodne odwzorowania klawiszy, można przywrócić odwzorowania do ustawień domyślnych za pomocą jednego polecenia:

```
% kbdcontrol -F
```

Jeśli uda się zdefiniować szczególnie pożyteczne odwzorowanie, warto skonfigurować je na stałe. Posiadacze uprawnień superużytkownika mogą pokusić się o dodanie odpowiednich ustawień w pliku */etc/rc.conf* (z zachowaniem odpowiedniej rozwagi). Poniżej przedstawiam dwa odwzorowania. Pierwsze z nich dodaje wywołanie polecenia `lynx` po naciśnięciu klawisza *Menu*, drugie kojarzy polecenie `startx` z lewym klawiszem specjalnym GUI:

```
keychange="64 lynx"
keychange="62 startx"
```

Ustawienia będą dotyczyły wszystkich użytkowników systemu. Jeśli ktoś preferuje indywidualne ustawienia dla jednego użytkownika, może wpisać odpowiednie polecenia `kbdcontrol` do pliku konfiguracyjnego powłoki. Poniższe dwa wiersze dopisałam do swojego pliku *.cshrc* przed ostatnim wierszem zawierającym instrukcję `endif`:

```
kbdcontrol -f 64 "lynx"
kbdcontrol -f 62 "startx"
```

Wykorzystanie odwzorowań klawiszy w systemie X

Omówione konfiguracje są bardzo wygodne, lecz co się stanie z nowo odwzorowanymi klawiszami w sesji X Window? Proponuję spróbować, ale od razu ostrzegam, że to nic nie da. Nie rozlegnie się nawet ostrzegawczy pisk pojawiający się przy błędach występujących podczas pracy w powłoce. Dzieje się tak z tej przyczyny, że urządzenia wejścia-wyjścia podczas sesji X są obsługiwane niezależnie przez protokół X.

Pracując w sesji X, mamy natomiast możliwość zdefiniowania własnych dowiązań klawiszy; do tego celu służy kilka różnych mechanizmów. Jeden z nich jest wbudowany w menedżerze okien. Większość nowoczesnych menedżerów okien oferuje graficzne narzędzia do konfiguracji skrótów klawiszowych. Moim ulubionym narzędziem jest aplikacja `xbindkeys_config`, dostępna w kolekcji portów **[Sposób 84.]**.

```
# cd /usr/ports/x11/xbindkeys_config
# make install clean
```

Ten port wymaga wcześniejszej instalacji portu `xbindkeys`:

```
# cd /usr/ports/x11/xbindkeys
# make install clean
```

Zamiast kompilować obydwa porty, można dopisać następujący wiersz w pliku */usr/ports/x11/xbindkeys_config/Makefile*:

```
BUILD_DEPENDS=   xbindkeys:${PORTSDIR}/x11/xbindkeys
```

To spowoduje, że wystarczy wykonać polecenie `make install clean` w porcie `xbindkeys_config`, aby skonfigurować i zainstalować obydwa porty.

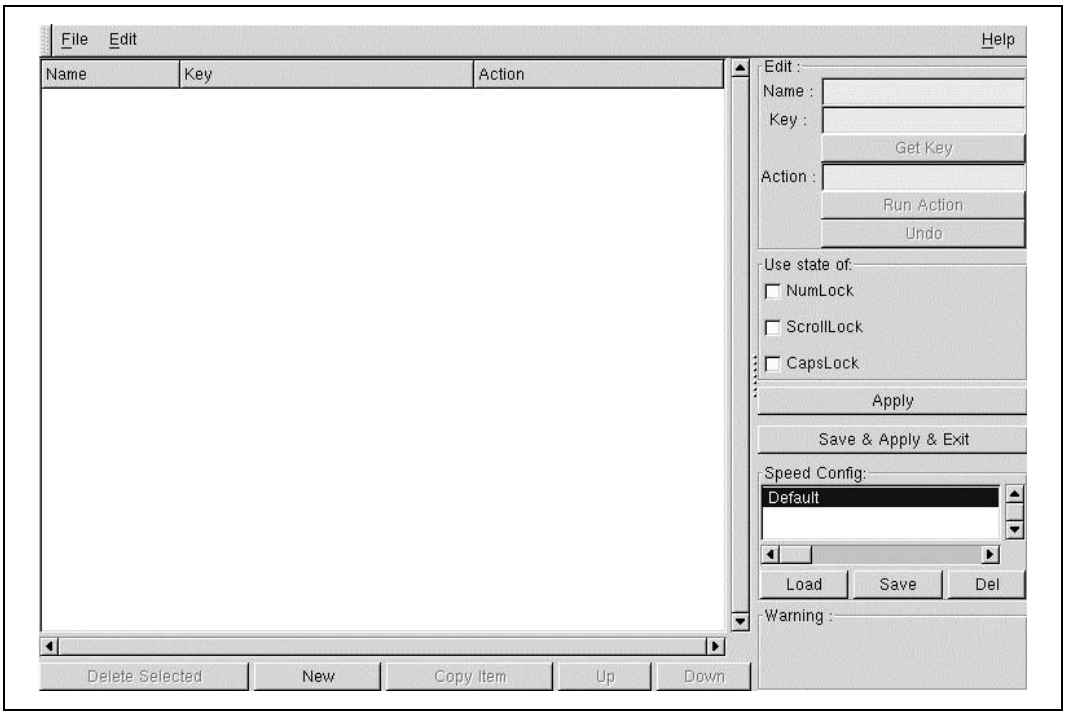
Po zakończeniu kompilacji otwieramy xterminal i wykonujemy następujące polecenia:

```
% xbindkeys --defaults ~/.xbindkeysrc
% xbindkeys_config
```

Po wykonaniu drugiego z poleceń pojawi się interfejs użytkownika przedstawiony na rysunku 1.1.

Utworzenie nowego dowiązania polega na kliknięciu przycisku *New* i wpisaniu informacyjnej nazwy w polu *Name:*. Następnie klikamy przycisk *Get Key*, po czym otworzy się nowe okno. Naciskamy odpowiednią kombinację klawiszy i gotowe: w oknie aplikacji pojawi się odpowiednio wypełniona pozycja odwzorowania kombinacji klawiszy. W polu *Action:* uzupełniamy polecenie związane z kombinacją. Zmiany akceptujemy klawiszem *Save & Apply & Exit*.

Wszelkie odwzorowania klawiszy utworzone za pomocą tego programu zostaną zapisane w pliku konfiguracyjnym *~/.xbindkeysrc*.



Rysunek 1.1. Program xbindkeys_config

Zobacz również:

- `man kbdcontrol`;
- `man atkbd`;
- Strona domowa programu `xbindkeys`: *http://hocwp.free.fr/xbindkeys/xbindkeys.html*.

SPOSÓB 5.

Wykorzystanie myszy w terminalu

Kopiowanie i wklejanie tekstu w terminalu za pomocą myszy.

Osoby przyzwyczajone do pracy w środowisku graficznym mogą podczas pracy w terminalu czuć się trochę nieswojo. Oczywiście można skonfigurować skróty klawiszowe i wykorzystywać sztuczki nawigacyjne, lecz to wszystko przestaje mieć znaczenie, gdy pojawi się potrzeba skopiowania i wklejenia tekstu.

Nie ma powodów do obaw — w terminalu mysz wcale nie musi pozostawać bezużyteczna. Być może demon obsługi myszy jest nawet aktywny; zależy to od konfiguracji systemu podczas instalacji. Zadaniem tego demona jest nasłuchiwanie danych przesyłanych przez mysz i przekazywanie ich do sterownika konsoli.

Użytkownicy programu `screen` **[Sposób 12.]** również mogą skorzystać z mechanizmu kopiowania i wklejania tekstu za pomocą myszy.

Jeżeli zainstalowany jest system X

Jeśli podczas instalacji systemu został zainstalowany również system X, demon `moused` prawdopodobnie również będzie uruchomiany po uruchomieniu systemu. Można to sprawdzić w następujący sposób:

```
% grep moused /etc/rc.conf
moused_port="/dev/psm0"
moused_type="auto"
moused_enable="YES"
```

Demon `moused` potrzebuje trzech rodzajów informacji:

- portu myszy (w tym przypadku */dev/psm0*, co oznacza port PS/2);
- typu protokołu (w tym przypadku `auto`, co oznacza, że protokół zostanie skonfigurowany automatycznie);
- ustawienia automatycznego uruchamiania.

Jeśli wynik powyższego polecenia będzie zbliżony do przykładowego, jesteśmy gotowi do kopiowania i wklejania.

Aby skopiować tekst do schowka, wystarczy zaznaczyć go, przytrzymując **lewy** przycisk myszy. Następnie umieszczamy kursor w miejscu, w którym chcemy wkleić skopiowany tekst, i klikamy **środkowy** klawisz myszy. To wszystko.

Aby zaznaczyć cały wyraz, wystarczy kliknąć go dwukrotnie. Aby zaznaczyć cały wiersz tekstu, klikamy go trzykrotnie.

Konfiguracja myszy dwuprzyciskowej

Co zrobić, jeśli mysz ma tylko dwa przyciski? Jako superużytkownik dopisujemy w pliku */etc/rc.conf* następujący wiersz (o ile go tam nie ma):

```
moused_flags="-m 2=3"
```

Ta opcja wskazuje demonowi `moused`, że prawy przycisk myszy ma być interpretowany jak trzeci (czyli środkowy) przycisk. Teraz można bez przeszkód używać prawego przycisku do wklejania tekstu ze schowka.

Aby wprowadzone zmiany w konfiguracji weszły w życie, należy ponownie uruchomić demon `moused`:

```
# /etc/rc.d/moused restart
Stopping moused.
Starting moused:.
```

Zmiany należy przetestować, kopiując tekst do schowka za pomocą lewego przycisku i wklejając go za pomocą prawego przycisku myszy.

Jeżeli system X nie jest zainstalowany

Takie same rezultaty można uzyskać bez konieczności instalacji systemu X. W tym celu należy jednak dopisać do pliku */etc/rc.conf* wymienione wyżej opcje konfiguracyjne demona `moused`.

Przedstawiony przykład dotyczy myszy PS/2. Użytkownicy innych modeli myszy powinni zapoznać się z częścią podręcznika `man moused` zatytułowaną „Configuring Mouse Daemon". Znajdują się tam szczegółowe informacje na temat konfiguracji różnych modeli myszy oraz typów protokołów je obsługujących, jak również dane na temat konfiguracji laptopów i obsługi kilku myszy: jednej podczas pracy w podróży (najczęściej w postaci wbudowanego manipulatora) i drugiej podczas pracy po przyłączeniu laptopa do stacji dokującej.

Użytkownicy myszy USB najczęściej muszą port */dev/psm0* zastąpić wpisem */dev/usm0*.

Mysz szeregowa przyłączona do portu COM1 wymaga konfiguracji portu */dev/cuaa0*. Czasem może okazać się, że niezbędne będzie dokonanie kilku prób z różnymi ustawieniami. Jak zwykle najlepszym przewodnikiem będzie podręcznik systemowy.

Zobacz również:

- `man moused`
- Dokumentacja dotycząca włączania obsługi myszy w NetBSD: *http://www.netbsd.org/Documentation/wscons/*;
- Dokumentacja dotycząca włączania obsługi myszy w OpenBSD: *http://www.openbsd.org/faq/faq7.html*.

SPOSÓB 6.

Dzienna dawka błahostek

Jak uprzyjemnić pracę, konfigurując urozmaicenia terminalu.

Jak mówi stare powiedzenie: „Tylko praca, bez zabawy, uschnie z nudy Jack niebawem". Ale co ma biedny Jacek czy Agatka zrobić, gdy całe dnie spędza tylko przed ekranem komputera? Mogliby na przykład wybrać się na stronę *http://www.thinkgeek.net/* i kupić sobie odjazdowy kubek albo inny gadżet. Można też skorzystać z jednego z programów rozrywkowych.

„Fortunki"

Rozpocznijmy od rozrywek na ekranie terminalu. Czy za każdym razem po zalogowaniu się system wita Was wesołym, dowcipnym lub tajemniczym cytatem? Jeśli tak, oznacza to, że otrzymaliście tak zwaną „fortunkę":

```
login: dru
Password:
Last login: Sat 1 10:10:16 on ttyv7

"You can't have everything, where would you put it?"
                -- Steven Wright
```

Jeśli „fortunka" się nie pojawia, należy z konta superużytkownika wywołać polecenie */stand/sysinstall*. Następnie wybieramy *Configure/Distributions* i zaznaczamy spacją opcję *games*. Klawiszem *Tab* podświetlamy przycisk *OK*. Po zakończeniu instalacji zamykamy program `sysinstall`.

Sprawdzamy, czy w pliku *~/.cshrc* znajduje się wywołanie programu `fortune`:

```
% grep fortune ~/.cshrc
/usr/games/fortune
```

Jeśli go tam nie będzie, należy dopisać na końcu tego pliku odpowiednie wywołanie:

```
% echo '/usr/games/fortune' >> ~/.cshrc
```

Nie należy zapomnieć o użyciu dwóch znaków większości, w przeciwnym razie cała zawartość pliku *.cshrc* zostanie usunięta. Aby sprawdzić wprowadzone zmiany, należy zastosować polecenie `source`, które ponownie załaduje plik konfiguracyjny. Ten sposób może okazać się przydatny w przypadku, gdy zostanie zaktualizowany alias i chcemy natychmiast wprowadzić zmiany w życie:

```
% source ~/.cshrc
Indifference will be the downfall of mankind, but who cares?
```

Jeśli po wylogowaniu się z systemu również chcemy otrzymywać „fortunkę", należy dopisać do pliku *.logout* następujący wiersz:

```
% echo '/usr/games/fortune' > ~/.logout
```

Jeśli takiego pliku nie było (a domyślnie nie jest on tworzony automatycznie), zostanie utworzony w wyniku wykonania powyższego polecenia. Tym razem został zastosowany jeden znak większości, ponieważ wiem, że plik taki nie istnieje. Gdyby jednak istniał, należy zastosować dwa znaki większości, co spowoduje dopisanie wiersza na końcu pliku.

Program `fortune`, choć trudno w to uwierzyć, posiada różne opcje. Niektóre z nich dają całkiem zabawne wyniki. Poznanie szczegółów pozostawiam Czytelnikom — odsyłam do podręcznika systemowego `man fortune`.

Dalsze zgłębianie błahostek

Lubię błahostki, więc korzystam z polecenia `calendar`. Wbrew logicznym domysłom nie wypisuje ono w terminalu kalendarza bieżącego miesiąca (tym zajmuje się polecenie `cal`). Zamiast tego `calendar` wypisuje zdarzenia związane z bieżącą datą:

```
% calendar
Nov 27          Alfred Nobel establishes Nobel Prize, 1895
Nov 27          Friction match invented, England, 1826
Nov 27          Hoosac Railroad Tunnel completed, 1873, in NW Massachusetts
Nov 28          Independence Day in Albania and Mauritania
Nov 28          Independence from Spain in Panama
Nov 28          Proclamation of the Republic in Chad
Nov 27          Jimi Hendrix (Johnny Allen Hendrix) is born in Seattle, 1942
```

Świetnie, zapomniałam, że dziś jest rocznica otwarcia tunelu Hoosac — zdarzenia, które spowodowało, że na mapie pojawiło się moje rodzinne miasteczko.

Wywołanie polecenia `calendar` można łatwo zautomatyzować. Jeśli ktoś chce poznać wydarzenia historyczne związane z bieżącą datą po zalogowaniu się lub po wylogowaniu z systemu, powinien dodać odpowiedni wpis do pliku *.cshrc* lub *.logout*. Chodzi oczywiście o ścieżkę do programu `calendar`, można więc posłużyć się następującym poleceniem, upraszczającym nieco to zadanie:

```
% echo `which calendar` >> .cshrc
```

Nie należy pomylić się przy znakach `>>` lub wcześniej ustawić w pliku `.cshrc` opcję `noclobber` zgodnie z opisem w **[Sposób 2.]**.

Dalsze rozrywki

Oczywiście istnieje więcej ciekawostek związanych z czasem. Opiszę jeszcze dwie, które mogą znaleźć amatorów.

Bieżący czas

Z pewnością każdemu zdarzyło się sprawdzić godzinę, korzystając z komputera. Wykorzystanie w tym celu polecenia `date` jest być może intuicyjne, ale jakże przy tym nudne. Następnym razem przy takiej okazji proponuję skorzystać z następującego polecenia:

```
% grdc
```

Ojej, to widać z drugiego końca pokoju. Niezły sposób, aby delikatnie zasugerować koledze wyjście na lunch.

Zdarzało mi się wpisywać polecenie */usr/games/grdc* do swojego pliku *~/.logout*. Gdy się wylogowałam, terminal wyświetlał godzinę, aż nacisnęłam *Ctrl+c* i zalogowałam się ponownie. Można to uznać za zabezpieczony hasłem wygaszasz ekranu działający w terminalu.

Faza księżyca

Czy zdarzyło się komuś czytać `man pom`? To jeden z ciekawszych podręczników systemowych, jakie znam. W tłumaczeniu na język polski brzmi mniej więcej tak:

> Program `pom` wyświetla bieżącą fazę księżyca. Może być przydatny do określenia terminu realizacji projektu i przewidywania nastrojów kadry menedżerskiej.

Brzmi, jakby palce w tym programie maczał sam Dilbert. Gdy do pliku *~/.cshrc* dopiszemy wywołanie */usr/games/pom*, po zalogowaniu się będziemy mieli okazję pogłębić swoją wiedzę astronomiczną:

```
% pom
The Moon is Waxing Gibbous (53% of Full)
```

Prosty programik, a jakże może ożywić pogawędki przy ekspresie do kawy.

Dodawanie terminalowi kolorów

Czy ktoś próbował poniższego polecenia?

```
% vidcontrol show
0                8 grey
1 blue           9 lightblue
2 green         10 lightgreen
3 cyan          11 lightcyan
4 red           12 lightred
5 magenta       13 lightmagenta
6 brown         14 yellow
7 white         15 lightwhite
```

Przypominają się stare czasy systemu DOS i plik *ansi.sys*. Tak, terminal pozwala na stosowanie koloru — powyższe polecenie wypisuje dostępne kolory. W terminalu powyższe polecenie wypisze wynik w kolorze, czego nie da się niestety oddać w tej książce.

Jeśli komuś spodobają się wybrane kolory, może dodać je do terminala. Na przykład poniższe polecenie ustawi żółty kolor czcionki i niebieskie tło:

```
% vidcontrol yellow blue
```

Jako tło można ustawić wyłącznie kolory o kodach od 1 do 7, w przeciwnym wypadku wystąpi błąd składni (*syntax error*). Warto popróbować z różnymi kombinacjami, aż uda się uzyskać zadowalający efekt. Można również ustawić kolorową ramkę okna:

```
% vidcontrol -b red
```

Powyższe ustawienia mają wpływ wyłącznie na bieżący terminal. Oczywiście odpowiednie wywołania polecenia `vidcontrol` można dopisać do pliku *~/.cshrc*, aby były wprowadzane w życie przy każdym zalogowaniu do systemu.

Jeśli mamy problem z odszukaniem kursora, można spróbować następującego polecenia:

```
% vidcontrol -c blink
```

Inne polecenie o podobnym działaniu:

```
% vidcontrol -c destructive
```

Zmiana kursora ma wpływ na wszystkie terminale w systemie. Jeśli inni użytkownicy zaczną narzekać na wprowadzone „usprawnienie" kursora, można przywrócić go do standardowego stanu następującym poleceniem:

```
% vidcontrol -c normal
```

Zobacz również:

- `man fortune`;
- `man calendar`;
- `man vidcontrol`;
- pakiety *games* w NetBSD oraz OpenBSD.