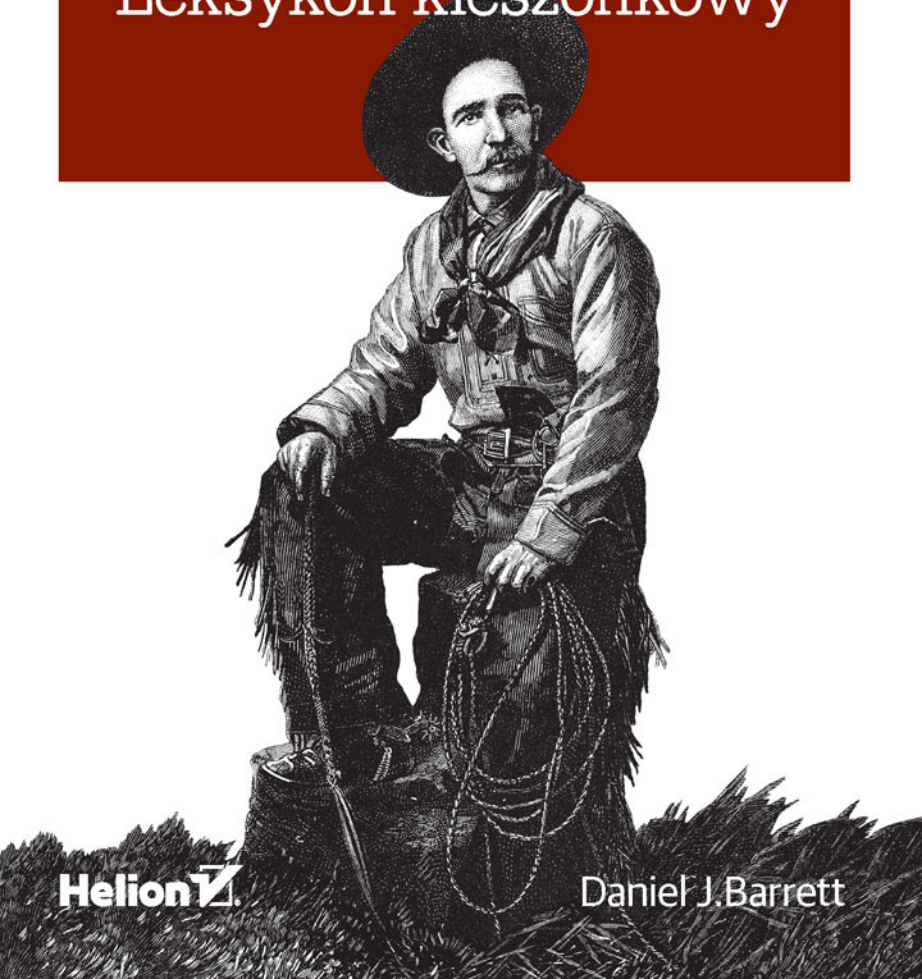
O'REILLY®

Wydanie III

Linux

Leksykon kieszonkowy



Helion

Daniel J. Barrett

Tytuł oryginału: Linux Pocket Guide, Third Edition

Tłumaczenie: Przemysław Szeremiota

ISBN: 978-83-283-3065-8

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/linlk3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Powłoka

Aby mieć możliwość wydawania Linuksowi poleceń, będziemy potrzebowali miejsca, w którym można by je wpisywać. To miejsce nazywane jest *powłoką*, która w Linuksie spełnia rolę interfejsu użytkownika wiersza poleceń. Polecenia są w niej wpisywane, a po naciśnięciu klawisza *Enter* powłoka uruchamia program (lub programy) podany w powłoce (informacje o tym, jak uruchomić powłokę, zamieszczono w sekcji „Uruchamianie powłoki").

Na przykład w celu sprawdzenia, kto jest zalogowany do systemu, można w powłoce uruchomić następujące polecenie:

```
→ who
kowalski      :0     Wrz 23 20:44
nowak       pts/0    Wrz 15 12:51
malinowski  pts/1    Wrz 22 21:15
kowalski    pts/2    Wrz 22 21:18
```

(pamiętajmy, że znak → reprezentuje tu znak zachęty powłoki, oznajmiający gotowość powłoki do przyjmowania poleceń). Pojedyncze polecenie może też uruchomić jednocześnie wiele programów, a nawet połączyć je w taki sposób, aby ze sobą współpracowały. Oto polecenie przekierowujące wyjście programu `who` i podłączające je do wejścia programu `wc`, który zajmuje się zliczaniem wierszy w pliku tekstowym. W efekcie otrzymamy liczbę wierszy w tekście wygenerowanym przez program `who`:

```
→ who | wc -l
4
```

informującą, ile osób jest zalogowanych w systemie[5]. Połączenie między programami `who` i `wc` wykonuje pionowa kreseczka, która nazywana jest *potokiem*.

Sama powłoka tak naprawdę również jest zwykłym programem, a w Linuksie dostępnych jest kilka różnych powłok. W tej książce skoncentrujemy się na powłoce bash (*Bourne-Again Shell*), która znajduje się w katalogu */bin/bash* i która w systemach Linux najczęściej jest powłoką domyślną. Aby sprawdzić, czy dana powłoka to bash, należy użyć polecenia:

```
→ echo $SHELL
/bin/bash
```

Jeśli okazało się, że powłoka Czytelnika to nie bash, wystarczy uruchomić tę powłokę bezpośrednio, używając polecenia `bash` (aby wyjść z niej potem do poprzedniej powłoki, należy wpisać polecenie `exit`). Aby ustawić powłokę

[5] Tak naprawdę dowiemy się, ile powłok zostało uruchomionych przez użytkowników. Jeżeli jeden użytkownik będzie miał uruchomionych kilka powłok, jak w naszym przykładzie użytkownik `kowalski`, na liście pojawi się on kilka razy.

bash (albo wybraną inną) jako domyślną powłokę danego użytkownika, należy skorzystać z polecenia `chsh` (patrz sekcja „chsh" w późniejszym wykazie poleceń).

Powłoka a programy

Polecenie w momencie uruchomienia może wywołać jakiś program (na przykład `who`), ale może też ono być tak zwanym *poleceniem wbudowanym* (ang. *built-in command*), czyli funkcją wykonywaną przez samą powłokę. Takie polecenia można odróżniać za pomocą polecenia `type`:

```
→ type who
who is /usr/bin/who
→ type cd
cd is shell builtin
```

Dobrze jest wiedzieć, jakie funkcje udostępnia sama powłoka, a jakie są częścią Linuksa. W kilku kolejnych sekcjach będziemy opisywać wyłącznie funkcje powłoki.

Wybrane funkcje powłoki bash

Powłoka ma znacznie więcej funkcji niż tylko umożliwienie uruchomienia poleceń. Potrafi też bardzo ułatwić tę pracę: nazwy wieloznaczne (*wildcards*), pozwalające na dopasowanie nazw plików; „historia poleceń" do szybkiego przywoływania wykonanych uprzednio poleceń; potoki umożliwiające połączenie wyjścia jednego programu z wejściem drugiego; zmienne przechowujące wartości wykorzystywane przez powłokę i wiele innych funkcji. Poświęcenie nieco czasu na naukę tych funkcji pozwoli szybciej i wydajniej pracować w Linuksie. Ta krótka prezentacja przedstawiła zaledwie ułamek możliwości powłoki, czas dowiedzieć się więcej (pełną dokumentację powłoki można wyświetlić za pomocą polecenia `info bash`).

Nazwy wieloznaczne

Nazwy wieloznaczne to swego rodzaju skrót do grupy plików o podobnych nazwach. Na przykład `a*` oznacza wszystkie pliki, których nazwy zaczynają się od litery „a". Nazwy wieloznaczne są „rozwijane" przez powłokę w nazwy rzeczywistych plików, pasujących do podanego wzorca. Wobec czego, jeżeli wpiszemy polecenie:

```
→ ls a*
```

powłoka najpierw rozwinie parametr `a*` w nazwy plików rozpoczynających się na literę „a”, znajdujących się w aktualnym katalogu roboczym, tak jakby zostało wpisane polecenie:

```
→ ls andrychow ametyst agrest
```

Polecenie `ls` nigdy nie dowie się, że użytkownik stosował nazwy wieloznaczne, zawsze otrzyma tylko listę nazw plików przygotowaną przez powłokę. Bardzo ważne jest zapamiętanie, że dosłownie *każde* polecenie Linuksa, niezależnie od jego pochodzenia, może być używane z nazwami wieloznacznymi i innymi funkcjami powłoki *shell*. To doprawdy zaskakujące, jak wielu użytkowników Linuksa sądzi, że poszczególne programy samodzielnie wykonują rozwijanie parametrów z symbolami wieloznacznymi — otóż nie, programy nigdy tego nie robią, robi to dla nich powłoka i odbywa się to w całości, jeszcze zanim dany program w ogóle zostanie uruchomiony.

W nazwach wieloznacznych nigdy nie wolno stosować dwóch znaków: myślnika na początku oraz ukośnika (/); znaki te muszą być zawsze używane w ich oryginalnym znaczeniu, co oznacza, że do wyszukania pliku o nazwie *.profile* trzeba użyć nazwy wieloznacznej `.pro*`, a aby wyświetlić wszystkie pliki w katalogu */etc* z końcówką *conf*, należy wpisać `/etc/*conf`.

Nazwa wieloznaczna	Znaczenie
*	Zero lub więcej kolejnych znaków
?	Dowolny pojedynczy znak
[*zbior*]	Dowolny pojedynczy znak z podanego *zbioru*. Najczęściej zapisywane są tutaj sekwencje znaków, takie jak `[aeiouAEIOU]`, oznaczająca wszystkie samogłoski, lub zakresy definiowane za pomocą myślnika, takie jak `[A-Z]`, oznaczający wszystkie wielkie litery
[^*zbior*]	Dowolny pojedynczy znak *nie* znajdującysię w podanym *zbiorze* (wcześniejszy przykład)
[!*zbior*]	To samo, co `^zbior`.

Pliki z kropką

Pliki, których nazwa jest poprzedzona kropką, zwane po prostu plikami z kropką, to specjalne pliki w Linuksie. Takie pliki nie są wyświetlane w niektórych programach:

- polecenie ls nie wyświetli ich, chyba że zostanie wykonane z opcją `-a`;
- nazwy wieloznaczne powłoki również ich nie używają.

W praktyce oznacza to, że są one niewidoczne do czasu, aż użytkownik wprost nie zażąda ich wyświetlenia. Dlatego czasem określa się je jako „ukryte pliki”.

Jeżeli w podawanym zbiorze znaków chcielibyśmy umieścić znak myślnika, trzeba zapisać go jako pierwszy lub ostatni znak zbioru. Aby w zbiorze umieścić znak zamykającego nawiasu kwadratowego (]), należy umieścić go na pierwszej pozycji w zbiorze. Aby w zbiorze umieścić znak ^ lub wykrzyknika (!), nie należy umieszczać go na pierwszej pozycji w zbiorze.

Rozwijanie nawiasów klamrowych

Podobnie jak nazwy wieloznaczne, wyrażenia zamknięte w nawiasach klamrowych również rozwijane są w wiele parametrów przekazywanych poleceniom. Wyrażenie rozdzielane przecinkami:

```
{X,YY,ZZZ}
```

zostanie rozwinięte najpierw do X, następnie do YY, a w końcu do ZZZ w postaci:

```
→ echo kan{X,Y,ZZZ}apka
kanXapka kanYapka kanZZZapka
```

Nawiasy klamrowe można stosować z dowolnymi ciągami znaków, więc przykład działa niezależnie od tego, jakie pliki znajdują się w aktualnym katalogu roboczym.

Zmienne powłoki

Zmienne powłoki i ich wartości można zdefiniować poprzez operację przypisania:

```
→ MOJAZMIENNA=3
```

Aby odwołać się do wartości przechowywanej w zmiennej, wystarczy umieścić znak dolara przed nazwą zmiennej:

```
→ echo $MOJAZMIENNA
3
```

Niektóre zmienne są standardowe i najczęściej definiowane są przez samą powłokę w momencie logowania do systemu.

Zmienna	Znaczenie
DISPLAY	Nazwa wyświetlacza systemu okien X
HOME	Nazwa katalogu domowego użytkownika, na przykład */home/kowalski*
LOGNAME	Nazwa użytkownika stosowana w czasie logowania, na przykład *kowalski*
MAIL	Ścieżka do katalogu z pocztą przychodzącą, na przykład */var/spool/mail/kowalski*
OLDPWD	Poprzedni katalog roboczy powłoki, używany przed ostatnim poleceniem cd
PATH	Ścieżka przeszukiwania powłoki; katalogi są rozdzielane dwukropkami

Zmienna	Znaczenie
PWD	Aktualny katalog roboczy powłoki
SHELL	Ścieżka do powłoki użytkownika, na przykład */bin/bash*
TERM	Typ wykorzystywanego terminala, na przykład *xterm* lub *vt100*
USER	Nazwa użytkownika

Zakres zmiennej (to znaczy zbiór programów, które wiedzą o jej istnieniu) domyślnie definiowany jest jako powłoka, w której została ona zdefiniowana. Aby udostępnić zmienną i jej wartość innym programom wywoływanym z powłoki (na przykład podpowłokom), należy użyć polecenia `export`:

```
→ export MOJAZMIENNA
```

albo jej skrótu:

```
→ export MOJAZMIENNA=3
```

Taka zmienna nazywana jest wtedy *zmienną środowiskową*, ponieważ dostępna jest też z innych programów w „środowisku" powłoki. W powyższym przykładzie wyeksportowana zmienna `MOJAZMIENNA` jest dostępna dla wszystkich programów działających w tej samej powłoce (dotyczy to również skryptów powłoki; więcej informacji w sekcji „Zmienne").

Aby zobaczyć wszystkie zmienne powłoki, należy wydać polecenie:

```
→ printenv
```

Aby pewnemu programowi jednokrotnie udostępnić wartość zmiennej, wystarczy nadać jej tymczasową wartość, wpisując *`zmienna=wartość`* w powłoce tuż przed samym poleceniem:

```
→ echo $HOME
/home/kowalski
→ HOME=/home/nowak echo "Mój katalog domowy to $HOME"
Mój katalog domowy to /home/nowak
→ echo $HOME
/home/kowalski          Oryginalna wartość zmiennej nie zmienia się
```

Ścieżka wyszukiwania

Programy są porozrzucane po całym systemie plików Linuksa, katalogach takich jak */bin* czy */usr/bin*. W jaki sposób powłoka odnajduje program uruchamiany z linii poleceń? Krytyczne znaczenie ma zmienna `PATH`, która przekazuje powłoce informacje, gdzie można znaleźć programy. Gdy wpiszemy w wierszu poleceń dowolne polecenie:

```
→ who
```

powłoka musi odnaleźć program `who` w katalogach Linuksa. Sprawdza więc zawartość zmiennej `PATH`, w której zapisana jest sekwencja katalogów rozdzielanych dwukropkami:

```
→ echo $PATH
/usr/local/bin:/bin:/usr/bin
```

Następnie powłoka sprawdza, czy program `who` nie znajduje się w jednym z katalogów zapisanych w zmiennej `PATH`. Jeżeli powłoce uda się znaleźć program `who` (na przykład */usr/bin/who*), wtedy zostanie on uruchomiony. W przeciwnym wypadku wyświetlony zostanie jedynie komunikat:

```
bash: who: command not found
```

Aby tymczasowo dodać katalogi do ścieżki wyszukiwania powłoki, można zmodyfikować zmienną `PATH`. Na przykład poniższe polecenie do ścieżki wyszukiwania dodaje katalog */usr/sbin*:

```
→ PATH=$PATH:/usr/sbin
→ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/sbin
```

Zmiana zostanie wprowadzona tylko w bieżącej powłoce. Aby wprowadzić ją na stałe do ścieżki wyszukiwania, należy zmodyfikować wartość zmiennej `PATH` w pliku *~/.bash_profile*, co zostanie opisane w sekcji „Dostosowywanie zachowań powłoki". Trzeba się tylko wylogować i ponownie zalogować albo po prostu trzeba ręcznie uruchomić plik startowy *~/.bash_profile* w każdej uruchomionej już powłoce.

```
→ . $HOME/.bash_profile
```

Aliasy

Wbudowane polecenie `alias` pozwala na definiowanie wygodnych skrótów dłuższych poleceń, co wydatnie zmniejsza liczbę wpisywanych znaków. Na przykład polecenie:

```
→ alias ll='ls -lG'
```

definiuje nowe polecenie `ll`, które uruchamia polecenie `ls -lG`.

```
→ ll
razem 436
-rw-r--r--     1 kowalski    3584 paz 11 14:59 plik1
-rwxr-xr-x     1 kowalski      72 sie  6 23:04 plik2
...
```

Aby aliasy były dostępne po każdym logowaniu do systemu, należy je zdefiniować w pliku *~/.bash_aliases* (proszę zobaczyć sekcję „Dostosowywanie zachowań powłoki")[6]. Aby wypisać wszystkie zdefiniowane aliasy, należy

[6] W niektórych konfiguracjach wykorzystuje się do tego osobny plik *~/.bashrc*.

uruchomić polecenie `alias`. Jeżeli aliasy nie wydają się wystarczającym narzędziem (nie przyjmują parametrów i nie pozwalają na wykonywanie rozgałęzień), proszę przeczytać sekcję „Programowanie skryptów powłoki", uruchomić polecenie `info bash` i przeczytać sekcję „Tworzenie i uruchamianie skryptów powłoki".

Przekierowanie wejścia i wyjścia

W powłoce możliwe jest przekierowanie standardowego wejścia, standardowego wyjścia i standardowego strumienia błędów (proszę zobaczyć sekcję „Wejście i wyjście") programów do i z plików. Innymi słowy, za pomocą operatora < można spowodować, że każdy program, odczytujący dane ze standardowego wejścia, może je otrzymać z podanego pliku:

```
→ polecenie < plik_wejsciowy
```

Podobnie każde polecenie, zapisujące wyniki do standardowego wyjścia, może zapisywać je do podanego pliku:

```
→ polecenie > plik_wyjsciowy      Utworzenie lub nadpisanie pliku wyjściowego
→ polecenie >> plik_wyjsciowy     Dopisanie danych na końcu pliku
```

Polecenie zapisujące cokolwiek do standardowego strumienia błędów również może zostać przekierowane tak, żeby zapisywać do pliku, podczas gdy dane na standardowym wyjściu wciąż będą wyświetlane na ekranie:

```
→ polecenie 2> plik_bledow
```

Aby przekierować do plików zarówno strumień błędów, jak i standardowe wyjście:

```
→ polecenie > plik_wyjsciowy 2> plik_bledow   Dwa osobne pliki
→ polecenie >& plik_wyjsciowy                 Do tego samego pliku
→ polecenie &> plik_wyjsciowy                 Do tego samego pliku
```

Potoki

Za pomocą operatora potoku (|) możliwe jest takie przekierowanie standardowego wyjścia jednego programu, aby stało się ono standardowym wejściem innego. Na przykład polecenie:

```
→ who | sort
```

przesyła wyjście programu `who` do wejścia programu `sort`, wypisując posortowaną listę zalogowanych w systemie użytkowników. Potoków można używać wielokrotnie. Dane wyjściowe polecenia `who` zostały posortowane (polecenie `sort`). Następnie za pomocą programu `awk` został wyodrębniony pierwszy wiersz, a zastosowanie polecenia `less` pozwoliło wyświetlić te dane na pojedynczej stronie (ekranie):

```
→ who | sort | awk '{print $1}' | less
```

Podstawienie procesu

Potoki pozwalają na wysłanie wyjścia z jednego programu do wejścia innego programu. Bardziej zaawansowana funkcja powłoki, zwana podstawieniem procesu (ang. *process substitution*), pozwala na zamaskowanie tego przekierowania pod postacią *nazwanego* pliku. Weźmy na przykład program, który porównuje zawartość dwóch plików — za pomocą operatora podstawienia procesu możemy w miejsce nazw plików podstawić wyniki uruchomienia dwóch programów.

Załóżmy, że istnieje katalog zawierający mnóstwo par plików tekstowych i plików graficznych JPEG:

```
→ ls plikijpeg
plik1.jpg  plik1.txt  plik2.jpg  plik2.txt …
```

Naszym zadaniem jest potwierdzenie, że każdemu plikowi JPEG odpowiada jeden plik tekstowy i odwrotnie. Można to zrobić, tworząc pliki tymczasowe: jeden, który będzie zawierał nazwy plików JPEG, i drugi, który będzie zawierał nazwy plików tekstowych; z obu należałoby potem usunąć rozszerzenia nazw plików (za pomocą polecenia `cut`) i tak uzyskane listy można by porównać za pomocą programu `diff`:

```
→ cd plikijpeg
→ ls *.jpg | cut -d. -f1 > /tmp/jpegs
→ ls *.txt | cut -d. -f1 > /tmp/texts
→ diff /tmp/jpegs /tmp/texts
5a6
> plik6        Brak pliku plik6.jpg
8d8
< plik9        Brak pliku plik9.txt
```

Dzięki funkcji podstawienia procesu to samo zadanie można wykonać pojedynczym poleceniem, bez tworzenia plików tymczasowych:

```
→ diff <(ls *.jpg|cut -d. -f1) <(ls *.txt|cut -d. -f1)
```

W obu wystąpieniach operator `<()` pełni funkcję nazw plików dla polecenia `diff`, widzianych przez ten program tak, jakby zawierały wyniki wykonania odpowiednich poleceń `ls` i `cut`.

Łączenie poleceń

Aby w wierszu poleceń wywołać kilka poleceń jedno za drugim, należy rozdzielać je znakiem średnika (`;`):

```
→ polecenie1 ; polecenie2 ; polecenie3
```

Aby wykonać powyższą sekwencję poleceń, a jednocześnie zatrzymać wykonywanie kolejnych, jeżeli jedno z nich nie wykona się prawidłowo, należy polecenia rozdzielać znakami &&:

```
→ polecenie1 && polecenie2 && polecenie3
```

Aby wykonać powyższą sekwencję poleceń, a jednocześnie zatrzymać wykonywanie kolejnych, jeżeli jedno z nich wykona się prawidłowo, należy polecenia rozdzielać znakami ||:

```
→ polecenie1 || polecenie2 || polecenie3
```

Cytowanie

Powłoka traktuje standardowo znaki białe jako znaki rozdzielające słowa w wierszu poleceń. Jeżeli polecenie wymaga, aby któreś ze słów *zawierało* w sobie takie znaki (na przykład nazwa pliku zawierająca w sobie spacje), należy zamknąć je wewnątrz pojedynczych lub podwójnych cudzysłowów. Tak oznaczone słowa powłoka będzie traktować jako niepodzielną całość. Słowa zamknięte w pojedynczych cudzysłowach nie ulegają żadnym zmianom, natomiast słowa zamknięte w podwójnych cudzysłowach pozwalają na przetworzenie zawartych w nich konstrukcji powłoki, takich jak zmienne.

```
→ echo 'Zmiennej HOME przypisano wartość $HOME'
Zmiennej HOME przypisano wartość $HOME
→ echo "Zmiennej HOME przypisano wartość $HOME"
Zmiennej HOME przypisano wartość /home/kowalski
```

Cudzysłowy odwrócone powodują, że zawarty w nich tekst traktowany jest jako polecenie powłoki; tekst ten zamieniany jest standardowym wyjściem tego polecenia:

```
→ date +%Y    Polecenie wypisze bieżącą datę (rok)
2016
→ echo Mamy rok `date +%Y`
Mamy rok 2016
```

Analogiczną funkcję pełni nawias poprzedzony znakiem $:

```
→ date +%Y    Polecenie wypisze bieżącą datę (rok)
2016
→ echo Mamy rok $(date +%Y)
Mamy rok 2016
```

z tym że jest to podmiana bardziej uniwersalna, bo może być zagnieżdżana:

```
→ echo Mamy rok $(date +%Y)
Mamy rok 2016
→ echo Zbliża się rok $(expr $(date +%Y) + 1)
Zbliża się rok 2017
```

Znaki specjalne

Jeżeli znak ma znaczenie specjalne, ale istnieje potrzeba użycia bezpośredniego (na przykład znaku gwiazdki (*) jako rzeczywistej gwiazdki, a nie znaku nazwy wieloznacznej), należy poprzedzić go znakiem ukośnika (\).

```
→ echo a*            Jako nazwa wieloznaczna, nazwy plików rozpoczynające się literą „a”
andrychow  ametyst  agrest
→ echo a\*                                  Jako dosłowny znak gwiazdki
a*
→ echo "Po angielsku DOM to $HOME"          Znak dolara oznacza wartość zmiennej
Po angielsku DOM to /home/kowalski
→ echo "Po angielsku DOM to \$HOME"         Dosłownie symbol dolara
Po angielsku DOM to $HOME
```

Podobny mechanizm można stosować ze znakami sterującymi (tabulacje, nowe wiersze, ^*D* itd.), aby stosować je w wierszu poleceń. Wystarczy umieścić przed nimi znak ^*V*. Ta metoda przydaje się szczególnie w przypadku znaków tabulacji (^*I*), które normalnie zostałyby użyte przez powłokę do uzupełnienia nazwy pliku (zobacz sekcję „Uzupełnianie nazw plików”).

```
→ echo "Tabulacja rozciąga się odtąd^V^Idotąd"
Tabulacja rozciąga się odtąd          dotąd
```

Edycja w wierszu poleceń

Powłoka bash pozwala na edytowanie wiersza poleceń za pomocą kombinacji klawiszy wywodzących się z edytorów `emacs` i `vi` (proszę zobaczyć sekcję „Tworzenie i edytowanie plików”). Aby włączyć w edycji wiersza poleceń kombinacje klawiszy edytora `emacs`, należy uruchomić następujące polecenie (umieszczenie go w pliku *~/.bash_profile* uruchomi ten tryb na stałe):

```
→ set -o emacs
```

Aby uzyskać kombinacje klawiszy edytora `vi` (lub `vim`) należy uruchomić polecenie:

```
→ set -o vi
```

Kombinacje klawiszy edytora Emacs	Kombinacje klawiszy edytora vi (najpierw należy nacisnąć klawisz ESC)	Znaczenie
^*P* lub klawisz strzałki w górę	*k*	Powrót do poprzedniego polecenia
^*N* lub klawisz strzałki w dół	*j*	Przejście do następnego polecenia
^*R*		Interaktywne szukanie poprzedniego polecenia
^*F* lub klawisz strzałki w prawo	*l*	Przejście o jeden znak do przodu

Kombinacje klawiszy edytora Emacs	Kombinacje klawiszy edytora vi (najpierw należy nacisnąć klawisz ESC)	Znaczenie
^*B* lub klawisz strzałki w lewo	*h*	Przejście o jeden znak do tyłu
^*A*	*0*	Przejście na początek wiersza
^*E*	*$*	Przejście na koniec wiersza
^*D*	*x*	Usunięcie następnego znaku
^*U*	^*U*	Usunięcie całego wiersza

Historia poleceń

Możliwe jest przywołanie poprzednio uruchomionych poleceń (czyli *historii* powłoki), edytowanie ich i ponowne uruchomienie. Poniżej podano kilka przydatnych poleceń związanych z historią powłoki.

Polecenie	Znaczenie
`history`	Wypisuje historię powłoki
`history` *N*	Wypisuje ostatnich *N* poleceń z historii powłoki
`history -c`	Usuwa historię powłoki
`!!`	Ponownie uruchamia poprzednie polecenie
`!`*N*	Ponownie uruchamia polecenie numer *N* w historii powłoki
`!-`*N*	Ponownie uruchamia polecenie wpisane *N* poleceń wcześniej

`!$`	Ostatni parametr poprzedniego polecenia; doskonale nadaje się np. do sprawdzania, przed usuwaniem plików, czy one rzeczywiście istnieją:
	`→ ls a*`
	`agrest ametyst andrychow`
	`→ rm !$`

`!*`	Wszystkie parametry poprzedniego polecenia
	`→ ls mojplik pustyplik duzyplik`
	`mojplik pustyplik duzyplik`
	`→ wc !*`
	`11 90 637 mojplik`
	`0 0 0 pustyplik`
	`333563 2737539 18577838 duzyplik`
	`333574 2737629 18578475 total`

Uzupełnianie nazw plików

Po naciśnięciu klawisza *Tab* w czasie wpisywania polecenia w wierszu poleceń powłoka automatycznie uzupełni wpisywaną nazwę pliku. Jeżeli do wpisanej do tej pory części nazwy pasuje kilka nazw plików, powłoka sygnałem dźwiękowym poinformuje o tej wieloznaczności. Szybkie ponowne naciśnięcie

klawisza *Tab* spowoduje, że powłoka wypisze na ekranie wszystkie dostępne nazwy plików. Proszę spróbować uruchomić takie polecenia:

```
→ cd /usr/bin
→ ls un<Tab><Tab>
```

Powłoka wyświetli wszystkie pliki w katalogu */usr/bin/* rozpoczynające się od *un*, takie jak *unique* czy *unzip*. Aby zawęzić wyszukiwanie, należy dodać jeszcze kilka znaków i ponownie wcisnąć klawisz *Tab*.

Kontrola zadań powłoki

`jobs`	Wypisuje listę zadań.
`&`	Uruchamia zadanie w tle.
`^Z`	Wstrzymuje działanie aktualnego zadania.
`suspend`	Wstrzymuje działanie powłoki.
`fg`	Wznowienie zadania i przeniesienie go na pierwszy plan.
`bg`	Sprawia, że wstrzymane zadanie wznawia pracę w tle.

Wszystkie powłoki w Linuksie posiadają mechanizm *kontroli zadań*, czyli możliwość uruchamiania poleceń w tle (działają one bez kontaktu z ekranem i klawiaturą) i na pierwszym planie (aktywny proces podłączony do ekranu i klawiatury). *Zadanie* to po prostu część pracy powłoki. W momencie interaktywnego uruchomienia polecenia powłoka zaczyna traktować je jak kolejne zadanie. Po zakończeniu działania przez polecenie, związane z nim zadanie znika. Zadania to pojęcie wyższego poziomu niż procesy systemu Linux; sam system w ogóle nie zajmuje się zadaniami, są one konstrukcjami stosowanymi wyłącznie przez powłoki. Z kontrolą zadań związanych jest kilka ważnych pojęć:

zadanie pierwszoplanowe
: Działa w powłoce, zajmując ją tak, że nie jest możliwe wydanie kolejnych poleceń.

zadanie w tle
: Działa w powłoce, ale nie zajmuje jej, w związku z czym możliwe jest wydawanie kolejnych poleceń w tej samej powłoce.

zawieszenie
: Tymczasowe wstrzymanie działania zadania pierwszoplanowego.

wznowienie
: Powoduje wznowienie działania wstrzymanego wcześniej zadania do wykonywania pierwszoplanowego.

jobs

Wbudowane polecenie `jobs` wypisuje listę zadań działających aktualnie w powłoce.

```
→ jobs
[1]- Running                 emacs mojplik &
[2]+ Stopped                 su
```

Liczba całkowita z lewej strony to numer zadania, a znak dodawania (+) oznacza domyślne zadanie, na które wpływ będą miały polecenia `fg` (*foreground* — pierwszy plan) i `bg` (*background* — tło).

&

Znak `&` umieszczony na końcu wiersza polecenia powoduje, że polecenie zostanie uruchomione jako zadanie w tle.

```
→ emacs mojplik &
[2] 28090
```

Powłoka odpowie, wypisując numer zadania (2) i identyfikator procesu polecenia (28090).

^Z

Naciśnięcie klawiszy ^*Z* w powłoce, w czasie gdy na pierwszym planie działa jakieś zadanie, spowoduje wstrzymanie tego zadania. Zadanie przestaje działać, ale jego stan jest zapamiętywany.

```
→ sleep 10                   Odczekanie 10 sekund
^Z
[1]+ Stopped                     sleep 10
→
```

Teraz możliwe jest wpisanie polecenia `bg` i odesłanie zadania w tło lub wpisanie `fg` i wznowienie zadania na pierwszym planie. Ale można też pozostawić polecenie w zawieszeniu i uruchamiać w międzyczasie inne polecenia.

suspend

Wbudowane polecenie `suspend`, jeżeli jest to możliwe, spowoduje wstrzymanie aktualnej powłoki, tak jakby zostały naciśnięte klawisze ^*Z* w odniesieniu do samej powłoki. Na przykład jeżeli za pomocą polecenia `sudo` uruchomiona zostanie powłoka z uprawnieniami superużytkownika, a następnie użytkownik będzie chciał wrócić do pierwotnej powłoki:

```
→ whoami
kowalski
→ sudo bash
Password: *******
# whoami
root
# suspend
[1]+ Stopped            sudo bash
→ whoami
kowalski
```

bg

`bg [%numer_zadania]`

Wbudowane polecenie `bg` odsyła wstrzymane zadanie do pracy w tle. Wpisane bez żadnych argumentów polecenie `bg` będzie działać na ostatnio wstrzymanym zadaniu. Aby podać konkretne zadanie (z listy podawanej przez polecenie `jobs`), należy wpisać numer tego zadania poprzedzony znakiem procenta (`%`).

```
→ bg %2
```

Niektóre rodzaje zadań interaktywnych nie mogą działać w tle, na przykład ze względu na to, że oczekują na podanie danych. Jeżeli użytkownik będzie próbował odesłać takie zadanie w tło, powłoka wstrzyma je i wypisze na ekranie:

```
[2]+ Stopped            treść_polecenia
```

Teraz możliwe jest tylko wznowienie zadania poleceniem `fg`.

fg

`fg [%numer_zadania]`

Wbudowane polecenie `fg` przywraca na pierwszy plan zadanie działające w tle lub wstrzymane. Wpisane bez żadnych argumentów wybiera zadanie ostatnio wstrzymane lub odesłane w tło. Aby podać konkretne zadanie, należy wpisać numer tego zadania poprzedzony znakiem procenta (`%`):

```
→ fg %2
```

Równoczesne korzystanie z wielu powłok

Kontrola zadań pozwala na równoczesne wykonywanie wielu poleceń, ale tylko jedno z nich działa w danym momencie na pierwszym planie. Alternatywnie można natomiast korzystać z wielu powłok równocześnie, a w każdej z nich może się niezależnie wykonywać inne polecenie pierwszoplanowe i dowolna liczba poleceń w tle.

Jeśli komputer posiada zainstalowany system graficznego interfejsu użytkownika (w rodzaju KDE czy Gnome), korzystanie z wielu powłok sprowadza się do otwarcia wielu okien terminala (proszę zobaczyć sekcję „Uruchamianie powłoki"). Ponadto niektóre z programów terminalowych, jak na przykład konsole ze środowiska KDE, pozwalają na otwieranie wielu kart z osobnymi powłokami w obrębie pojedynczego okna.

Ale nawet w systemie bez interfejsu okienkowego (na przykład w obrębie połączenia SSH) można łatwo korzystać z wielu powłok. Celuje w tym program `screen`, który na bazie terminala ASCII emuluje obecność wielu okien powłoki. Pomiędzy tymi oknami można się przełączać za pomocą specjalnych kombinacji klawiszy (innym przykładem takiego multipleksera powłoki jest program `tmux`.) Aby rozpocząć sesję programu `screen`, wystarczy wpisać polecenie:

```
→ screen
```

Po ewentualnym wypisaniu komunikatów wprowadzających program prezentuje zwyczajny znak zachęty powłoki. Wygląda to tak, jakby nic się nie zmieniło, ale od tego momentu powłoka działa w wirtualnym „oknie"; program `screen` obsługuje 10 takich okien, etykietowanych numerami od 0 do 9.

Można to łatwo sprawdzić, wpisując dowolne polecenie (na przykład `ls`), a następnie naciskając `^A^C` (*Control-A*, *Control-C*). Ekran zostanie wyczyszczony, a w oknie pojawi się świeży znak zachęty powłoki. Operujemy teraz w drugim „oknie", niezależnym od pierwszego. Można tu uruchomić dowolne inne polecenie (na przykład `df`), a następnie nacisnąć `^A^A`, aby powrócić do pierwszego okna, gdzie wciąż widać wynik wykonania polecenia `ls`. Drugie naciśnięcie `^A^A` spowoduje przełączenie z powrotem na drugie okno terminala. Poniżej znajduje się wykaz innych często używanych kombinacji klawiszy sterujących programem `screen` (kompletną dokumentację programu umieszczono na stronie dokumentacji systemowej *man*, a skrótowa pomoc wewnętrzna programu `screen` dostępna jest pod kombinacją klawiszy `^A?`):

`^A?`	Lista obsługiwanych skrótów klawiszowych.
`^A^C`	Nowe okno powłoki.
`^A0, ^A1, ... ^A9`	Przełączanie się pomiędzy numerowanymi oknami powłoki.
`^A'`	Zapytanie o numer i przełączenie się do wybranego okna.
`^A^N`	Przełączenie się do następnego okna.
`^A^A`	Przełączenie się do poprzednio aktywnego okna (używane wielokrotnie powoduje przełączanie się pomiędzy dwoma ostatnio aktywnymi oknami).
`^A^W`	Lista otwartych okien.
`^AN`	(`wielkie N`) Wypisanie numeru aktywnego okna.

`^Aa`	(małe a) Wysłanie do powłoki symbolu *Control-A* (normalnie jest on przechwytywany przez program screen; w powłoce `bash` kombinacja *Control-A* przesuwa kursor na początek polecenia).
`^D`	Zakończenie aktywnej powłoki; *Control-D* jest standardowym symbolem końca wiersza (proszę zobaczyć sekcję „Kończenie działania powłoki"), który wymusza zamknięcie powłoki.
`^A\`	Zabicie wszystkich okien powłoki i zakończenie programu screen.

Podczas korzystania z programu `screen` trzeba pamiętać o przechwytywaniu przez program niektórych kombinacji klawiszy; bywa, że te same kombinacje są używane na przykład w edytorze tekstu. Żeby faktycznie przekazać do programu uruchomionego w `screenie` kombinację *Control-A*, należy skorzystać z polecenia `^Aa`.

Zabijanie działającego polecenia

Jeżeli polecenie uruchomione w powłoce musi zostać natychmiast zatrzymane, należy nacisnąć klawisze ^*C*. Dla powłoki naciśnięcie tych klawiszy oznacza: „Natychmiast zakończ działanie zadania pierwszoplanowego". W związku z tym, jeżeli wyświetlany jest bardzo duży plik (na przykład poleceniem `cat`) i chcielibyśmy zatrzymać ten proces, wystarczy wcisnąć klawisze ^*C*:

```
→ cat wielkiplik
To jest bardzo duży plik z wieloma wierszami tekstu bla bla
↪bla bla bla bla bla bla bla bla blablablabla ^C
→
```

Aby zabić zadanie działające w tle, można przywołać je na pierwszy plan poleceniem `fg` i nacisnąć klawisze ^*C*:

```
→ sleep 50 &
[1] 12752
→ jobs
[1]- Running    sleep 50 &
→ fg %1
sleep 50
^C
→
```

Można też zastosować polecenie `kill` (proszę zobaczyć sekcję „Kontrola procesów").

Stosowanie klawiszy ^*C* to nie najlepszy sposób na kończenie pracy programów. Jeżeli tylko program posiada własną procedurę zakończenia — należy ją wykorzystać (więcej informacji w ramce na następnej stronie).

Jak przeżyć zabicie

Zabicie pierwszoplanowego programu za pomocą klawiszy ^*C* może spowodować, że powłoka będzie działać w dziwnym trybie, może nie wypisywać na ekranie naciskanych klawiszy. Tak się dzieje, ponieważ zabicie programu nie daje mu możliwości „posprzątania po sobie". Jeżeli to się zdarzy:

- Należy nacisnąć klawisze ^*J*. Daje to takie same efekty, jak klawisz *Enter*, ale zadziała na pewno nawet wtedy, gdy klawisz *Enter* nie działa.
- Należy wpisać polecenie `reset` (nawet jeżeli litery nie pojawiają się na ekranie w czasie wpisywania) i nacisnąć klawisze ^*J* w celu uruchomienia tego polecenia. To powinno przywrócić powłokę do normalnego stanu.

Kombinacja klawiszy ^*C* działa jedynie w powłoce. Najprawdopodobniej nie będzie miała żadnego wpływu na okna nie będące oknami powłoki. Dodatkowo niektóre programy pisane są tak, aby „przechwytywać" tę kombinację klawiszy i ignorować ją. Przykładem takiego programu może być edytor tekstu `emacs`.

Kończenie działania powłoki

Aby zakończyć działanie powłoki, należy wpisać polecenie `exit` lub nacisnąć klawisze ^*D*[7].

```
→ exit
```

Dostosowywanie zachowań powłoki

Aby wszystkie powłoki uruchamiane w systemie działały dokładnie w ten sam sposób, należy odpowiednio edytować pliki *.bash_profile* i *.bashrc* w swoim katalogu domowym. Te pliki uruchamiane są za każdym razem, gdy użytkownik loguje się do systemu (*~/.bash_profile*) lub otwiera powłokę (*~/.bashrc*). Można w nich ustalać zmienne i aliasy, uruchamiać programy, drukować swój horoskop i wykonywać dowolne inne operacje.

Te dwa pliki są przykładami *skryptów powłoki*: plików wykonywalnych, zawierających polecenia powłoki. Na ten temat więcej informacji można znaleźć w sekcji „Programowanie skryptów powłoki".

[7] Znak *Control+D* dla każdego programu odczytującego dane ze standardowego wejścia oznacza „koniec pliku". W tym przypadku programem jest sama powłoka, która kończy wtedy swoje działanie.

To koniec krótkiego wprowadzenia do Linuksa i powłoki. W dalszej części zostaną omówione wymienione i opisane najbardziej przydatne polecenia służące do pracy z plikami, procesami, użytkownikami, multimediami, w sieci i inne.

Podstawowe operacje na plikach

`ls` Wypisuje pliki z podanego katalogu.

`cp` Kopiuje plik.

`mv` Zmienia nazwę pliku („przesuwa" go).

`rm` Usuwa podany plik.

`ln` Tworzy dowiązania do plików (ich nazwy alternatywne).

Jednymi z pierwszych operacji, jakie wykonują użytkownicy w systemie Linux, jest manipulowanie plikami: kopiowanie, zmiana nazw, usuwanie i tak dalej.

ls

stdin **stdout** -file **--opt** **--help** **--version**

`ls [opcje] [pliki]`

Polecenie `ls` (wymawia się je *el-es*) wypisuje atrybuty plików i katalogów. Możliwe jest wypisanie plików z aktualnego katalogu:

```
→ ls
```

z podanych katalogów:

```
→ ls katalog1 katalog2 katalog3
```

lub poszczególnych plików:

```
→ ls mojplik mojplik2 mojplik3
```

Najważniejszymi opcjami tego polecenia są opcje `-a`, `-l` i `-d`. Domyślnie polecenie `ls` ukrywa wszystkie pliki, których nazwy rozpoczynają się od kropki — zgodnie z informacjami zamieszczonymi w ramce „Pliki z kropką". Zastosowanie opcji `-a` pozwala na wypisanie wszystkich plików.

```
→ ls
mojplik mojplik2
→ ls -a
.ukryty_plik mojplik mojplik2
```

Opcja `-l` powoduje, że polecenie podaje więcej informacji o plikach:

```
→ ls -l moje.dane
-rw-r--r--   1 kowalski uzytkownicy   149 Paz 28 2002 moje.dane
```

w których zawarte są od lewej do prawej: uprawnienia do pliku (`-rw-r--r--`), liczba sztywnych dowiązań do pliku (1), właściciel pliku (`kowalski`), grupa

(uzytkownicy), rozmiar (149 bajtów), data ostatniej modyfikacji (28 października 2002 roku) i nazwa pliku. Dokładniejsze informacje na ten temat można znaleźć w sekcji „Zabezpieczenia plików".

Opcja -d powoduje wyświetlenie informacji o samych katalogach, pomijane są informacje o plikach znajdujących się w katalogu:

```
→ ls -ld mojkatalog
drwxr-xr-x 1 kowalski kowalski 4096 wrz 29 2011 mojkatalog
```

Przydatne opcje

- **-a** Wypisuje wszystkie pliki, włącznie z tymi, których nazwy zaczynają się od kropki.
- **-l** Dłuższy wydruk, zawierający atrybuty plików. Dodanie opcji -h spowoduje, że wielkości plików będą podawane w kilobajtach, megabajtach i gigabajtach, a nie w bajtach.
- **-G** Dłuższy wydruk, z pomijaniem uprawnień grupowych.
- **-F** Pewne nazwy plików uzupełniane są specjalnymi symbolami, oznaczającymi typy plików. Do nazw katalogów dodawany jest znak lewego ukośnika (/), do plików wykonywalnych — znak gwiazdki (*), do dowiązań symbolicznych — znak (@), do potoków nazwanych — znak pionowej kreski — (|), a do gniazdek — znak równości (=). Są to tylko graficzne oznaczenia typu plików przeznaczone dla użytkowników; nie są one częścią nazw tych plików.
- **-S** Sortowanie listy plików według rozmiarów plików.
- **-t** Sortowanie listy według daty i czasu ostatniej modyfikacji plików.
- **-R** Jeżeli wypisywana jest zawartość katalogu, wypisywana jest też zawartość jego podkatalogów.
- **-d** Jeżeli wypisywana jest zawartość katalogu, sama zawartość jest pomijania, a wypisywany jest sam katalog.

---

cp

stdin stdout -file **--opt** **--help** **--version**

cp [*opcje*] *pliki* (*plik* | *katalog*)

Polecenie cp normalnie wykonuje kopiowanie pliku:

```
→ cp plik plik2
```

albo kopiuje wiele plików do jednego katalogu:

```
→ cp plik1 plik2 plik3 plik4 katalog_docelowy
```

Za pomocą opcji -a lub -r możliwe jest też rekursywne kopiowanie zawartości katalogów.

Przydatne opcje

- `-p` Kopiowana jest nie tylko zawartość pliku, ale również wszystkie związane z nim uprawnienia, znaczniki czasu, a jeżeli pozwalają na to uprawnienia użytkownika, także dane jego właściciela i grupy. Normalnie właścicielem kopiowanych plików staje się kopiujący je użytkownik, są one oznaczane aktualnym czasem, a ich uprawnienia powstają przez połączenie wartości *umask* użytkownika z oryginalnymi uprawnieniami.
- `-a` Hierarchia katalogów jest kopiowana rekursywnie, z zachowaniem właściwości plików i dowiązań.
- `-r` Hierarchia katalogów jest kopiowana rekursywnie. Opcja ta nie zachowuje właściwości plików, takich jak uprawnienia czy oznaczenia czasowe. Zachowane zostają dowiązania symboliczne.
- `-i` Tryb interaktywny. Przed usunięciem plików docelowych użytkownik zostanie zapytany o pozwolenie.
- `-f` Wymuszenie wykonania kopiowania. Jeżeli plik docelowy już istnieje, zostanie on bezwarunkowo usunięty.

---

mv

stdin stdout -file **--opt** **--help** **--version**

`mv [opcje] źródło cel`

Polecenie `mv` (*move* — przenieś) pozwala na zmianę nazwy pliku:

```
→ mv jakisplik innyplik
```

lub przeniesienie plików i katalogów do katalogu docelowego:

```
→ mv plik1 plik2 katalog3 katalog4 katalog_docelowy
```

Przydatne opcje

- `-i` Tryb interaktywny. Przed usunięciem plików docelowych użytkownik zostanie zapytany o pozwolenie.
- `-f` Wymuszenie wykonania przeniesienia. Jeżeli plik docelowy już istnieje, zostanie on bezwarunkowo usunięty.

---

rm

stdin stdout -file **--opt** **--help** **--version**

`rm [opcje] pliki | katalogi`

Polecenie `rm` (*remove* — usuń) umożliwia usuwanie plików:

```
→ rm niepotrzebny niepotrzebny2
```

lub rekursywne usuwanie całych katalogów:

```
→ rm -r katalog1 katalog2
```

Przydatne opcje

`-i` Tryb interaktywny. Użytkownik jest pytany o pozwolenie na usunięcie każdego z plików.

`-f` Wymuszenie usunięcia. Wszystkie błędy i ostrzeżenia są ignorowane.

`-r` Rekursywne usuwanie katalogu i jego zawartości. Tej opcji należy używać ostrożnie, szczególnie w połączeniu z opcją `-f`, która może spowodować usunięcie wszystkich plików użytkownika.

ln

stdin stdout -file **--opt** **--help** **--version**

`ln [opcje] źródło cel`

Dowiązanie (*link*) jest odnośnikiem do innego pliku, tworzonym przez polecenie `ln`. Można powiedzieć, że dowiązania umożliwiają nadanie temu samemu plikowi wielu nazw, dzięki czemu może on istnieć w co najmniej dwóch lokalizacjach jednocześnie.

Istnieją dwa rodzaje dowiązań. *Dowiązanie symboliczne* odnosi się do innego pliku poprzez jego ścieżkę; podobnie działają „skróty" w systemie Windows i „aliasy" w systemie Mac OS X. Aby utworzyć dowiązanie symboliczne, należy użyć opcji `-s`:

```
→ ln -s mojplik dowiazanie_symboliczne
```

Jeżeli oryginalny plik zostanie usnięty, wtedy dowiązanie symboliczne będzie nieprawidłowe, wskazując na nieistniejący już plik. Z drugiej strony, *dowiązanie sztywne* jest tylko inną nazwą tego samego pliku na jednym dysku. Mówiąc bardziej technicznie — wskazuje ono na ten sam węzeł inode. Usunięcie pliku oryginalnego nie powoduje unieważnienia dowiązania sztywnego. Na rysunku 5. przedstawiono różnicę między tymi typami dowiązań. Aby utworzyć dowiązanie sztywne, należy wpisać:

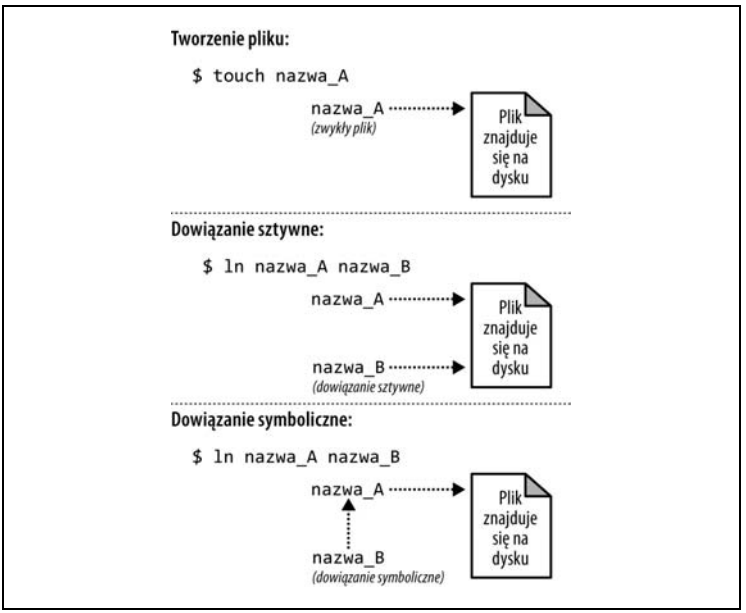
```
→ ln mojplik dowiazanie_sztywne
```

Dowiązania symboliczne mogą wskazywać pliki na innych partycjach, ponieważ są odnośnikami do ścieżki pliku. Dowiązania sztywne nie mogą wykroczyć poza granice partycji, ponieważ numer węzła inode jednej partycji nie ma żadnego znaczenia w innej. Poza tym dowiązania symboliczne mogą wskazywać na katalogi, a dowiązania sztywne — nie, chyba że utworzy je superużytkownik z wykorzystaniem opcji `-d`.

Przydatne opcje

`-s` Utworzenie dowiązania symbolicznego (zamiast sztywnego).

`-i` Tryb interaktywny. Przed usunięciem plików docelowych użytkownik zostanie zapytany o pozwolenie.



Rysunek 5. Dowiązania sztywne i symboliczne

`-f` Wymuszenie utworzenia dowiązania. Jeżeli plik docelowy istnieje, zostanie on bezwarunkowo usunięty.

`-d` Tworzenie dowiązania sztywnego do katalogu (tylko dla superużytkownika).

Za pomocą jednego z poniższych poleceń bardzo łatwo można sprawdzić, gdzie wskazuje dowiązanie symboliczne:

```
→ readlink dowiazanie_symboliczne
mojplik
→ ls -l dowiazanie_symboliczne
-rwxr-xr-x 1 kowalski ... dowiazanie_symboliczne -> mojplik
```

Dowiązania symboliczne mogą wskazywać na kolejne dowiązania symboliczne; aby prześledzić cały łańcuch dowiązań aż do pliku docelowego, najlepiej skorzystać z polecenia `readlink -f`.

Operacje na katalogach

`cd` Zmiana aktualnego katalogu.

`pwd` Wypisanie aktualnego katalogu, tzn. określenie „gdzie się znajdujemy" w systemie plików.

`basename` Wypisanie ostatniej części ścieżki pliku.

Skorowidz

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

O'REILLY®

Z taką ściągawką pokochasz swojego Linuksa!

Linux jest popularnym systemem operacyjnym o otwartych źródłach. Jego cechą charakterystyczną jest wyjątkowa łatwość konfiguracji. Z powodzeniem konkuruje z systemami Microsoft Windows i Mac OS X. Prawdziwą siłą systemu jest interfejs tekstowy, zwany powłoką, w którym wpisuje się i uruchamia polecenia. Umiejętność posługiwania się powłoką ma ogromne znaczenie dla użytkownika Linuksa.

W tym wydaniu przewodnika, który docenią zarówno początkujący, jak i zaawansowani użytkownicy, pojawiły się nowe polecenia służące do przetwarzania plików audiowizualnych, odczytywania i zapisywania zawartości schowka systemowego oraz do wykonywania operacji na plikach PDF. Nie pominięto też idiomów powłoki, takich jak podstawianie poleceń czy przekazywanie ich potokiem do powłoki.

Dr Daniel J. Barrett – jest inżynierem oprogramowania, administratorem systemów, muzykiem i satyrykiem. Od wczesnych lat 90. XX wieku pisze o technologiach informatycznych. Dla wydawnictwa O'Reilly napisał wiele książek, między innymi: *SSH, Secure Shell: The Definitive Guide*, *MediaWiki* i *Linux. Bezpieczeństwo. Receptury*.

Najważniejsze zagadnienia omówione w książce:

- system plików i powłoka
- kopie zapasowe i zdalne przechowywanie danych
- przeglądanie i kontrola procesów
- zarządzanie kontem użytkownika i uprawnienia superużytkownika
- połączenia sieciowe i programowanie skryptów powłoki

ISBN 978-83-283-3065-8

cena: 29,90 zł