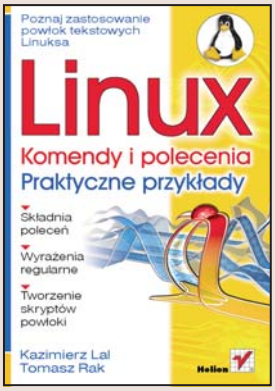
Linux. Komendy i polecenia. Praktyczne przykłady



Autorzy: Kazimierz Lal, Tomasz Rak
ISBN: 83-7361-994-1
Format: B5, stron: 264

Poznaj zastosowanie powłok tekstowych Linuksa

- Składnia poleceń
- Wyrażenia regularne
- Tworzenie skryptów powłoki

Rosnąca popularność Linuksa pociągnęła za sobą rozwój środowisk graficznych, dzięki którym możliwe jest korzystanie z systemu bez jakiejkolwiek znajomości konsoli tekstowej i jej poleceń. Narzędzia oferowane przez środowiska graficzne takie jak KDE czy Gnome umożliwiają również administrowanie systemem, ale w nieco ograniczonym zakresie. Pełny dostęp do wszystkich funkcji systemu nadal możliwy jest jedynie za pośrednictwem trybu tekstowego. Jednak ilość poleceń i ich parametrów skutecznie odstrasza wielu użytkowników Linuksa od jakichkolwiek prób opanowania możliwości konsoli.

Książka „Linux. Komendy i polecenia. Praktyczne przykłady" przedstawia najczęstsze zastosowania powłoki tekstowej Linuksa. Czytając ją, poznasz typy powłok tekstowych stosowane w różnych dystrybucjach systemu Linux oraz nauczysz się korzystać z wyrażeń regularnych, filtrów i potoków. Dowiesz się, jak definiować zmienne i jak tworzyć skrypty powłoki za pomocą języka C. Znajdziesz tu również zasady administrowania systemem za pomocą poleceń wydawanych w trybie tekstowym. Dzięki zadaniom znajdującym się w książce zastosujesz poznaną wiedzę w praktyce.

- Metaznaki i wyrażenia regularne
- Stosowanie filtrów i potoków
- Definiowanie zmiennych powłoki
- Wyrażenia arytmetyczne i konstrukcje sterujące
- Tworzenie skryptów powłoki
- Zarządzanie procesami, kontami użytkowników i prawami dostępu
- Tworzenie prostych elementów graficznych

Jeśli chcesz wykorzystać pełnię możliwości Linuksa, powinieneś poznać zasady korzystania z trybu tekstowego. Dzięki tej książce zrobisz to bez problemu.

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

Spis treści

Rozdział 4.
Cytowanie

Domyślnym działaniem powłoki jest obliczanie wartości i podstawianie. Wiersz poleceń jest przeglądany, spacje dzielą go na słowa, napotkane znaki specjalne wskazują, jakie czynności mają być wykonane, a wyniki są wstawiane w odpowiednie miejsce wiersza. Nie zawsze jednak takie działanie powłoki jest pożądane [1] [11] [13] [18]. Może się zdarzyć, że w pewnych sytuacjach powłoka zinterpretuje wprowadzone komendy niezgodnie z intencją użytkownika. Można się przed tym zabezpieczyć, odbierając niektórym znakom (traktowanym przez powłokę w sposób szczególny) ich specjalne znaczenie. Służy do tego mechanizm cytowania znaków (ang. *quoting*). Za pomocą cytowania chroni się spacje występujące w wierszu poleceń (cytowany ciąg znaków jest interpretowany jak pojedyncze słowo) oraz nadzoruje rozwinięcia (w szczególności nazw plików i zmiennych). Znaki cytowania rozróżniane przez powłokę zostały przedstawione w tabeli 4.1.

Tabela 4.1. *Znaki cytowania*

Znak(i)	Opis
Brak cytowania	Brak cytowania oznacza, że wszystkie słowa wydzielone za pomocą spacji są interpretowane przez powłokę.
Apostrofy *'ciag_znakow'*	Tekst zawarty w apostrofach jest chroniony. Cały *ciag_znakow* jest przesyłany jako argument do polecenia.
Cudzysłowy "*ciag_znakow* "	Tylko znaki $, \ oraz znak akcentu ` mają specjalne znaczenie w *ciągu_znaków*. Wszystkie inne znaki są chronione. Cudzysłów zapewnia jednoznaczną interpretację ciągu znaków, nawet w przypadku użycia znaku spacji.
Lewy ukośnik \c	Znak występujący po \ jest chroniony przed interpretacją przez powłokę.
Akcenty `*polecenie*`	*Polecenie* umieszczone w akcentach jest wykonywane przez podpowłoki i w jego miejsce podstawiany jest uzyskany wynik. Analiza dokonywana przez podpowłoki przebiega w zwykły sposób. Brane są pod uwagę włączone znaki cytowania.

4.1. Apostrofy

Apostrofy są wygodnym sposobem na to, by ciąg znaków pozbawić specjalnego znaczenia. Znaki znajdujące się pomiędzy otwierającym i zamykającym apostrofem stają się znakami zwykłymi. Ponadto apostrofy łączą tekst w jeden argument. Spacja nie jest wtedy separatorem argumentów.

Przykład 4.1.

Apostrofy użyte w poniższym poleceniu `grep` sprawiają, że ciąg znaków `zapisz wiadomosc` potraktowany zostanie jako jeden argument wyszukiwany w pliku *dane*:

```
grep 'zapisz wiadomosc' dane
```

Jeżeli nie użylibyśmy znaków apostrofu i zapisalibyśmy polecenie `grep` w takiej postaci:

```
grep zapisz wiadomosc dane
```

słowo *zapisz* szukane byłoby w plikach *wiadomosc* i *dane*.

Przykład 4.2.

Załóżmy, że wykonana została sekwencja poleceń:

```
zmienna='wartosc wpisana do zmiennej'
echo $zmienna
```

Wynikiem będzie wyświetlenie na ekranie ciągu znaków:

```
wartosc wpisana do zmiennej
```

Gdy użyjemy polecenia:

```
echo '$zmienna'
```

znak `$` utraci specjalne znaczenie i na ekranie otrzymamy:

```
$zmienna
```

W analogiczny sposób swoje specjalne znaczenie tracą wszystkie inne znaki specjalne. Łatwo możemy się domyślić, co by się wydarzyło, gdybyśmy nie zastosowali apostrofów.

4.2. Cudzysłowy

Cudzysłowy działają podobnie jak apostrofy, ale nie pozbawiają wszystkich znaków specjalnych ich znaczenia. Znaki `$`, `\` oraz `` ` `` utrzymują swoje specjalne znaczenie w ciągu znaków. W cudzysłowach możliwe jest podstawianie wartości zmiennych, ale podstawianie nazw plików nie jest już możliwe. Podstawianie wykonywane na zmiennych umieszczonych wewnątrz cudzysłowów różni się nieco od zwykłego podstawiania w wierszu poleceń. Wynika to z ochrony zapewnianej przez cudzysłowy.

Przykład 4.3.

Przeanalizujmy skrypt:

```
$ zmienna="Ochrona   spacji   zawartych  w  tekscie"
$ echo $zmienna
Ochrona spacji zawartych w tekscie
$ echo "$zmienna"
Ochrona   spacji   zawartych  w  tekscie
```

Umieszczenie zmiennej w cudzysłowie powoduje ochronę spacji w niej umieszczonych. Normalnie powłoka przetwarza zmienną w ten sposób, że usuwa wszystkie nadmiarowe spacje i każdy element traktuje oddzielnie. W naszym przykładzie polecenie:

```
$ echo $zmienna
```

usuwa spacje poprzedzające poszczególne słowa, pobiera te słowa jako oddzielne argumenty i wyświetla, rozdzielając je pojedynczymi spacjami. Gdy mamy do czynienia z zapisem w cudzysłowach:

```
$ echo "$zmienna"
```

spacje są chronione i polecenie `echo` otrzymuje cały ciąg znaków jako pojedynczy argument.

Przykład 4.4.

Rozważmy teraz przykład obrazujący różnice między użyciem apostrofów, cudzysłowów lub nieużyciem żadnego z tych znaków. Zakładamy, że w katalogu bieżącym istnieje plik *tekst* oraz istnieje zmienna *zm* o wartości `napis`.

```
$ echo Plik tekst zawiera $zm
Plik tekst zawiera napis
$echo 'Plik teks* zawiera $zm'
Plik teks* zawiera $zm
$echo "Plik teks* zawiera $zm"
Plik teks* zawiera napis
```

Apostrofy pozbawiają specjalnego znaczenia znaki * i `$`, a cudzysłów pozbawia specjalnego znaczenia tylko znak *.

Przykład 4.5.

Załóżmy, że chcemy za pomocą polecenia `grep` wyszukać wiersze w pliku *abonenci* według wzorca zawartego w zmiennej *dane* o wartości `Jan Kowalski`. Polecenie:

```
grep $dane abonenci
```

po wstawieniu wartości zmiennej będzie miało postać:

```
grep Jan Kowalski abonenci
```

i spowoduje, że `grep` będzie szukał wyrazu *Jan* w plikach *Kowalski* i *abonenci*. Dopiero zastosowanie cudzysłowów:

```
grep "$dane" abonenci
```

pozwoli uzyskać polecenie:

```
grep "Jan Kowalski" abonenci
```

i da pożądany efekt.

4.3. Lewy ukośnik

Lewy ukośnik \ służy do pozbawienia specjalnego znaczenia znaku, który występuje za nim. Można w ten sposób chronić dowolny znak w dowolnym miejscu. Znak \ jest często używany wewnątrz cudzysłowów do ochrony następnych znaków cudzysłowu, na przykład:

```
$ echo "Powiesc pt. \"Lalka\""
Powiesc pt. "Lalka"
```

Innym zastosowaniem znaku \ jest ochrona przed podstawianiem przez powłokę nazw plików lub wartości zmiennych. W tym celu za jego pomocą należy „zasłonić" znaki specjalne:

```
$ echo To chyba za drogo\?
To chyba za drogo?
```

Znak lewego ukośnika jest więc wygodnym narzędziem, które bardzo ułatwia nadzorowanie podstawień wykonywanych przez powłokę.

4.4. Znaki akcentu

Znaki akcentu ` powodują, że powłoka uważa wszystko, co znajduje się pomiędzy nimi, za polecenie. Dzięki temu są one bardzo użyteczne przy pisaniu skryptów i programów, a szczególnie przydają się do przypisywania wartości zmiennym. Pomiędzy apostrofami można umieścić dowolne polecenie lub zestaw poleceń. Każde z nich będzie wykonywane, jakby zostało wprowadzone w wierszu poleceń. Taki efekt bierze się stąd, że każde polecenie jest wykonywane we własnej podpowłoce. Oznacza to, że polecenie takie nie będzie pamiętane po powrocie do powłoki, z której zostało wywołane.

Przykład 4.6.

Przeanalizujmy skrypt:

```
$ data=date
$ echo $data
date
$ data=`date`
$ echo $data
Mon May 29 19:58:29 MDT 2005
```

Początkowo zmiennej *data* został przypisany ciąg znaków `date`. Powłoka uznała wszystkie znaki po prawej stronie znaku równości za część stałej, która ma stanowić wartość zmiennej. W kolejnym poleceniu użyte zostały znaki akcentu:

```
$ data=`date`
```

dzięki czemu wynik polecenia `date` zastąpił samo polecenie. Ostatecznie zmiennej *data* została przypisana bieżąca data.

Przykład 4.7.

Aby wyświetlić informację o liczbie plików w bieżącym katalogu, można użyć polecenia:

```
$ echo "Jest `ls|wc -l` pliki w `pwd`"
```

Znaki ` spowodują podstawienie wartości w miejsce poleceń `ls|wc -l` i `pwd`, co przykładowo może dać:

```
Jest 23 pliki w /home/jan
```