

Helion 

Linux

Wiersz poleceń
i skrypty powłoki

BIBLIA

Wydanie IV

Richard Blum

Christine Bresnahan

WILEY

Tytuł oryginału: Linux® Command Line and Shell Scripting Bible, 4th Edition

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-8322-074-1

Copyright © 2021 by John Wiley & Sons, Inc., Indianapolis, Indiana

All Rights Reserved. This translation published under license with the original publisher John Wiley & Sons, Inc.

Translation copyright © 2023 by Helion S.A.

Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Linux is a registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise without the prior written permission of the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/linwb4>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorach	19
O redaktorze merytorycznym	20
Podziękowania	21
Wprowadzenie	23
Część I. Wiersz poleceń Linuksa	27
<hr/>	
Rozdział 1. Powłoki Linuksa — podstawy	29
Poznanwanie Linuksa	29
Jądro Linuksa	30
Narzędzia GNU	35
Środowisko pulpitu Linuksa	36
Zapoznanie z dystrybucjami Linuksa	41
Podstawowe dystrybucje Linuksa	42
Podsumowanie	43
Rozdział 2. Uzyskiwanie dostępu do powłoki	45
Dostęp do wiersza poleceń	45
Konsole	46
Terminale graficzne	46
Dostęp do CLI przez konsolę Linuksa	47
Dostęp do CLI przez graficzny emulator terminala	50
Emulator terminala Gnome	50
Uzyskiwanie dostępu do terminala GNOME	50
Pasek menu	55
Emulator terminala Konsole	58
Dostęp do Konsole	58
Pasek menu	59
Emulator terminala xterm	64
Uzyskiwanie dostępu do xterm	65
Parametry wiersza poleceń	65
Podsumowanie	66

Rozdział 3. Podstawowe polecenia powłoki Bash	68
Uruchamianie powłoki	68
Praca z wierszem poleceń powłoki	69
Posługiwanie się podręcznikiem Bash	69
Przeglądanie systemu plików	73
System plików Linuksa	73
Poruszanie się po katalogach	76
Wyświetlanie list plików i katalogów	79
Wyświetlanie podstawowej listy	79
Długi format listy	81
Filtrowanie listy wyników	82
Obsługa plików	83
Tworzenie plików	83
Kopiowanie plików	84
Uzupełnianie poleceń	86
Dowiązania do plików	86
Zmienianie nazw plików	88
Usuwanie plików	89
Zarządzanie katalogami	90
Tworzenie katalogów	90
Usuwanie katalogów	91
Przeglądanie zawartości plików	92
Sprawdzanie typu pliku	92
Wyświetlanie całego pliku	93
Wyświetlanie części pliku	95
Podsumowanie	97
Rozdział 4. Więcej poleceń powłoki Bash	99
Monitorowanie programów	99
Podglądanie procesów	99
Monitorowanie procesów w czasie rzeczywistym	105
Zatrzymywanie procesów	107
Monitorowanie przestrzeni dyskowej	109
Montowanie mediów	109
Polecenie df	112
Polecenie du	113
Praca z plikami danych	114
Przechowywanie danych	114
Wyszukiwanie danych	117
Kompresja danych	119
Archiwizacja danych	120
Podsumowanie	121
Rozdział 5. Rozszerzenie wiadomości o powłoce	122
Typy powłoki	122
Relacja rodzic-dziecko między powłokami	125
Wyświetlanie listy procesów	129
Pomysłowe wykorzystanie podpowłok	130

Polecenia zewnętrzne i wbudowane	134
Polecenia zewnętrzne	134
Polecenia wbudowane	135
Podsumowanie	140
Rozdział 6. Zmienne środowiskowe w systemie Linux	142
Zmienne środowiskowe — informacje ogólne	142
Globalne zmienne środowiskowe	143
Lokalne zmienne środowiskowe	144
Zmienne zdefiniowane przez użytkownika	145
Ustawianie lokalnych zmiennych zdefiniowanych przez użytkownika	145
Ustawianie globalnych zmiennych środowiskowych	147
Usuwanie zmiennych środowiskowych	148
Sprawdzanie domyślnych zmiennych środowiskowych powłoki	149
Ustawianie zmiennej środowiskowej PATH	153
Lokalizacja systemowych zmiennych środowiskowych	154
Proces logowania do powłoki	154
Proces powłoki interaktywnej	159
Proces powłoki nieinteraktywnej	159
Ustawianie zmiennych środowiskowych na stałe	160
Zmienne tablicowe	161
Podsumowanie	162
Rozdział 7. Uprawnienia do plików w systemie Linux	163
Zabezpieczenia Linuksa	163
Plik /etc/passwd	164
Plik /etc/shadow	165
Dodawanie użytkownika	165
Usuwanie użytkownika	168
Modyfikowanie konta użytkownika	169
Grupy w Linuksie	171
Plik /etc/group	172
Tworzenie nowych grup	173
Modyfikowanie grup	173
Dekodowanie uprawnień do plików	174
Symbole uprawnień do plików	174
Domyślne uprawnienia do plików	175
Zmiana ustawień zabezpieczeń	177
Zmiana uprawnień	177
Zmiana właściciela	178
Wspólne pliki	179
Listy kontroli dostępu	181
Podsumowanie	183
Rozdział 8. Zarządzanie systemami plików	185
Systemy plików Linuksa	185
Ewolucja systemu plików Linuksa	185
Systemy plików z księgowaniem	186
Systemy plików z zarządzaniem woluminami	188

Praca z systemami plików	190
Tworzenie partycji	190
Polecenie <code>gdisk</code>	192
Polecenie <code>GNU parted</code>	194
Tworzenie systemu plików	194
Sprawdzanie i naprawianie systemu plików	196
Woluminy logiczne	197
Struktura LVM	197
LVM w Linuksie	198
Używanie LVM w Linuksie	200
Podsumowanie	201
Rozdział 9. Instalacja oprogramowania	203
Zarządzanie pakietami	203
Systemy oparte na Debianie	204
Zarządzanie pakietami za pomocą polecenia <code>apt</code>	205
Instalowanie pakietów oprogramowania za pomocą polecenia <code>apt</code>	206
Aktualizacja oprogramowania za pomocą polecenia <code>apt</code>	208
Dezinstalacja oprogramowania za pomocą polecenia <code>apt</code>	209
Repozytoria <code>apt</code>	210
Systemy oparte na Red Hat	212
Wyświetlanie listy zainstalowanych pakietów	212
Instalowanie oprogramowania za pomocą polecenia <code>dnf</code>	213
Aktualizacja oprogramowania za pomocą polecenia <code>dnf</code>	214
Dezinstalacja oprogramowania za pomocą polecenia <code>dnf</code>	215
Uszkodzone zależności	215
Repozytoria RPM	215
Zarządzanie oprogramowaniem za pomocą kontenerów	216
Kontenery <code>snap</code>	216
Kontenery <code>flatpak</code>	218
Instalacja z kodu źródłowego	219
Podsumowanie	222
Rozdział 10. Praca z edytorami	224
Praca z edytorem <code>vim</code>	224
Sprawdzanie pakietu <code>vim</code>	225
Podstawy edytora <code>vim</code>	226
Edytowanie danych	228
Kopiowanie i wklejanie	229
Wyszukiwanie i zastępowanie	229
Edytor <code>nano</code>	230
Edytor <code>Emacs</code>	232
Sprawdzanie pakietu <code>Emacs</code>	232
Edytor <code>Emacs</code> w trybie konsolowym	234
Edytor <code>Emacs</code> z graficznym interfejsem użytkownika	238
Rodzina edytorów KDE	240
Edytor <code>KWrite</code>	240
Edytor <code>Kate</code>	245

Edytor GNOME	249
Uruchamianie gedit	249
Podstawowe funkcje edytora gedit	250
Zarządzanie wtyczkami	253
Podsumowanie	256

Część II. Podstawy skryptów powłoki

259

Rozdział 11. Podstawy budowy skryptów	261
Używanie kilku poleceń na raz	261
Tworzenie pliku skryptu	262
Wyświetlanie komunikatów	264
Zmienne	265
Zmienne środowiskowe	265
Zmienne użytkownika	266
Podmienianie poleceń	268
Przekierowywanie wejścia i wyjścia	269
Przekierowywanie wyjścia	270
Przekierowywanie wejścia	270
Potoki	271
Wykonywanie działań matematycznych	274
Polecenie expr	274
Nawiasy	276
Rozwiązanie zmiennoprzecinkowe	277
Wychodzenie ze skryptu	280
Sprawdzanie stanu wyjścia	280
Polecenie exit	281
Praktyczny przykład	283
Podsumowanie	284
Rozdział 12. Polecenia strukturalne	286
Instrukcja if-then	286
Instrukcja if-then-else	289
Zagnieżdżanie instrukcji if	290
Polecenie test	293
Porównywanie liczb	295
Porównywanie łańcuchów	296
Porównywanie plików	300
Testy złożone	310
Zaawansowane właściwości instrukcji if-then	311
Pojedyncze nawiasy	311
Podwójne nawiasy	312
Podwójne nawiasy prostokątne	313
Polecenie case	314
Praktyczny przykład	316
Podsumowanie	320

Rozdział 13. Więcej poleceń strukturalnych	322
Polecenie for	322
Wczytywanie listy wartości	323
Wczytywanie złożonych list wartości	324
Wczytywanie listy ze zmiennej	325
Wczytywanie wartości z polecenia	326
Zmiana separatora pól	327
Wczytywanie katalogu za pomocą symboli wieloznaczących	328
Polecenie for w stylu języka C	330
Polecenie for z języka C	330
Używanie więcej niż jednej zmiennej	331
Polecenie while	332
Podstawowy format polecenia while	332
Używanie kilku poleceń testowych	333
Polecenie until	334
Zagnieżdżanie pętli	336
Przeglądanie za pomocą pętli danych w plikach	338
Kontrolowanie pętli	339
Polecenie break	339
Polecenie continue	342
Przetwarzanie wyników pętli	344
Kilka praktycznych przykładów	346
Wyszukiwanie plików wykonywalnych	346
Tworzenie wielu kont użytkownika	347
Podsumowanie	348
Rozdział 14. Przyjmowanie danych od użytkownika	350
Przekazywanie parametrów	350
Wczytywanie parametrów	350
Wczytywanie nazwy skryptu	352
Testowanie parametrów	354
Specjalne zmienne parametryczne	354
Liczenie parametrów	354
Pobieranie wszystkich danych	356
Przesuwanie parametrów	358
Opcje	359
Wyszukiwanie opcji	359
Polecenie getopt	363
Polecenie getopt_s	366
Standaryzacja opcji	368
Odbieranie danych od użytkownika	369
Podstawy wczytywania	369
Kontrola czasu	371
Wczytywanie bez wyświetlania	372
Wczytywanie danych z pliku	372
Praktyczny przykład	373
Podsumowanie	377

Rozdział 15. Prezentowanie danych	378
Wejście i wyjście	378
Standardowe deskryptory plików	378
Przekierowywanie błędów	381
Przekierowywanie wyjścia w skryptach	382
Przekierowania tymczasowe	382
Trwałe przekierowania	383
Przekierowywanie wejścia w skryptach	384
Tworzenie własnych przekierowań	385
Tworzenie wyjściowych deskryptorów plików	385
Przekierowywanie deskryptorów plików	386
Tworzenie wejściowych deskryptorów plików	387
Tworzenie deskryptora odczytu/zapisu plików	387
Zamykanie deskryptorów plików	388
Wyświetlanie listy otwartych deskryptorów plików	389
Tłumienie wyników poleceń	391
Pliki tymczasowe	392
Tworzenie lokalnego pliku tymczasowego	392
Tworzenie tymczasowego pliku w katalogu	393
Tworzenie tymczasowego katalogu	394
Zapisywanie komunikatów w dzienniku	395
Praktyczny przykład	396
Podsumowanie	397
Rozdział 16. Kontrola w skryptach	399
Obsługa sygnałów	399
Wysyłanie sygnałów do powłoki Bash	399
Generowanie sygnałów	400
Przechwytywanie sygnałów	402
Przechwytywanie sygnału wyjścia w skrypcie	403
Modyfikowanie lub usuwanie pułapki	404
Wykonywanie skryptów w tle	406
Wykonywanie w tle	406
Wykonywanie wielu zadań w tle	408
Wykonywanie skryptów bez rozłączania	409
Kontrolowanie zadań	410
Wyświetlanie zadań	410
Ponowne uruchamianie zatrzymanych zadań	412
Czy warto być uprzejmym	413
Polecenie nice	414
Polecenie renice	414
Wykonywanie według zegara	415
Planowanie wykonywania zadań za pomocą polecenia at	415
Regularne wykonywanie skryptów	419
Uruchamianie skryptów z nową powłoką	422
Praktyczny przykład	423
Podsumowanie	428

Część III. Zaawansowane techniki skryptowe

431

Rozdział 17. Tworzenie funkcji	433
Podstawy funkcji skryptowych	433
Tworzenie funkcji	434
Używanie funkcji	434
Zwracanie wartości przez funkcję	436
Domyślny stan wyjścia	436
Polecenie return	437
Przechwytywanie wyjścia funkcji	438
Używanie zmiennych w funkcjach	439
Przekazywanie parametrów do funkcji	439
Zakres dostępności zmiennych w funkcji	441
Zmienne tablicowe i funkcje	443
Przekazywanie tablic do funkcji	443
Zwracanie tablic przez funkcje	445
Funkcje rekurencyjne	445
Tworzenie biblioteki	447
Używanie funkcji w wierszu poleceń	448
Tworzenie funkcji w wierszu poleceń	448
Definiowanie funkcji w pliku .bashrc	449
Praktyczny przykład	451
Pobieranie i instalacja	451
Budowa biblioteki	451
Funkcje biblioteki shtool	453
Używanie biblioteki	453
Podsumowanie	454
Rozdział 18. Pisanie skryptów dla pulpitów graficznych	456
Tworzenie menu tekstowych	456
Tworzenie struktury menu	456
Tworzenie funkcji menu	458
Dodawanie logiki menu	459
Łączenie wszystkiego razem	459
Polecenie select	460
Tworzenie okien	462
Pakiet dialog	462
Opcje polecenia dialog	468
Polecenie dialog w skryptach	470
Interfejs graficzny	472
Środowisko KDE	472
Środowisko GNOME	475
Praktyczny przykład	479
Podsumowanie	482

Rozdział 19. Wprowadzenie do edytorów sed i gawk	483
Praca z tekstem	483
Edytor sed	483
Podstawy programu gawk	486
Podstawowe polecenia edytora sed	492
Więcej opcji zastępowania	492
Używanie adresów	494
Usuwanie wierszy	496
Wstawianie i dołączanie tekstu	498
Zmienianie wierszy	500
Zamienianie znaków	501
Jeszcze parę słów o drukowaniu	502
Praca z plikami w edytorze sed	505
Praktyczny przykład	507
Podsumowanie	511
Rozdział 20. Wyrażenia regularne	512
Podstawy wyrażeń regularnych	512
Definicja	512
Typy wyrażeń regularnych	513
Definiowanie wzorców BRE	514
Zwykły tekst	514
Znaki specjalne	515
Znaki zakotwiczenia	516
Kropka	518
Klasy znaków	519
Negacja klas znaków	521
Zakresy	521
Specjalne klasy znaków	522
Gwiazdka	523
Rozszerzone wyrażenia regularne	524
Znak zapytania	524
Znak plusa	525
Klamry	526
Symbol potoku	527
Grupowanie wyrażeń	527
Praktyczny przykład	528
Liczenie plików w katalogach	528
Sprawdzanie poprawności numeru telefonu	529
Sprawdzanie adresów e-mail	531
Podsumowanie	533
Rozdział 21. Zaawansowane funkcje edytora sed	534
Polecenia wielowierszowe	534
Polecenie n	535
Wielowierszowe polecenie usuwania	538
Wielowierszowe polecenie drukowania	539

Schowek tymczasowy	540
Negacja poleceń	541
Zmiana przepływu sterowania	544
Rozgałęzianie	544
Testowanie	546
Wymiana przez wzorzec	547
Znak &	547
Zamiana pojedynczych słów	548
Polecenia edytora sed w skryptach	549
Opakowania	549
Przekierowywanie wyników edytora sed	550
Tworzenie narzędzi edytora sed	551
Dodawanie wierszy odstępu	551
Dodawanie pustych wierszy, gdy trochę już ich jest	551
Numerowanie wierszy w pliku	552
Drukowanie ostatnich wierszy	553
Usuwanie wierszy	555
Usuwanie znaczników HTML-a	557
Praktyczny przykład	558
Podsumowanie	563
Rozdział 22. Zaawansowane funkcje edytora gawk	565
Zmienne	565
Zmienne wbudowane	566
Zmienne zdefiniowane przez użytkownika	571
Tablice	572
Definiowanie zmiennych tablicowych	573
Iteracja przez zmienne tablicowe	573
Usuwanie zmiennych tablicowych	574
Wzorce	575
Wyrażenia regularne	575
Operator dopasowywania	575
Wyrażenia matematyczne	576
Polecenia strukturalne	577
Instrukcja if	577
Instrukcja while	578
Instrukcja do-while	580
Instrukcja for	580
Formaty drukowania	581
Funkcje wbudowane	583
Funkcje matematyczne	583
Funkcje łańcuchowe	585
Funkcje czasu	586
Funkcje zdefiniowane przez użytkownika	587
Definiowanie funkcji	587
Używanie własnych funkcji	588
Tworzenie biblioteki funkcji	588
Praktyczny przykład	589
Podsumowanie	590

Rozdział 23. Praca z alternatywnymi powłokami	591
Powłoka Dash	591
Funkcjonalność powłoki Dash	592
Parametry wiersza poleceń powłoki Dash	593
Zmienne środowiskowe powłoki Dash	593
Wbudowane polecenia powłoki Dash	595
Skrypty powłoki Dash	596
Tworzenie skryptów powłoki Dash	596
Co nie będzie działać	597
Powłoka zsh	599
Budowa powłoki zsh	599
Opcje powłoki	599
Polecenia wbudowane	600
Pisanie skryptów powłoki zsh	604
Działania matematyczne	604
Polecenia strukturalne	605
Funkcje	606
Praktyczny przykład	607
Podsumowanie	608

Część IV. Tworzenie praktycznych skryptów i zarządzanie nimi 609

Rozdział 24. Pisanie prostych narzędzi skryptowych	611
Wykonywanie kopii zapasowych	611
Codzienne wykonywanie kopii zapasowej plików	612
Tworzenie skryptu archiwizacji godzinnej	618
Usuwanie kont	622
Potrzebne funkcje	622
Tworzenie skryptu	630
Uruchamianie skryptu	634
Monitorowanie systemu	636
Domyślne funkcje audytu powłoki	636
Funkcje monitorowania uprawnień	640
Tworzenie skryptu	642
Uruchamianie skryptu	644
Podsumowanie	646
Rozdział 25. Organizacja skryptów	647
Kontrola wersji	647
Katalog roboczy	649
Poczekalnie	649
Repozytorium lokalne	649
Repozytorium zdalne	649
Rozgałęzianie	650
Klonowanie	650
Git jako system kontroli wersji	650

Konfiguracja środowiska Git 651
Zatwierdzanie w Git 654
Podsumowanie 661

Dodatki **663**

Dodatek A. Przewodnik po poleceniach powłoki Bash 665
Dodatek B. Skrócony przewodnik po edytorach sed i gawk 677
Skorowidz 687

Podstawy budowy skryptów

W TYM ROZDZIALE:

- Używanie kilku poleceń na raz
- Tworzenie pliku skryptu
- Wyświetlanie komunikatów
- Zmienne
- Przekierowywanie wejścia i wyjścia
- Potoki
- Wykonywanie działań matematycznych
- Wychodzenie ze skryptu

Znasz już podstawy systemu Linux i wiersza poleceń, więc możesz zacząć pisać kod. W tym rozdziale omawiamy podstawy pisania skryptów powłoki. Musisz je opanować, abyś mógł zacząć pisać własne arcydzieła sztuki skryptowej.

Używanie kilku poleceń na raz

Do tej pory używaliśmy interfejsu wiersza poleceń (CLI) powłoki do wpisywania poleceń i przeglądania zwracanych przez nie wyników. Podstawę skryptów stanowi możliwość wykonywania wielu poleceń i przetwarzania ich wyników, które mogą być nawet przekazywane od jednego polecenia do innego. Powłoka umożliwia tworzenie łańcuchów poleceń, które są wykonywane w jednym kroku.

Jeśli chcesz wykonać dwa polecenia razem, możesz je wpisać w wierszu poleceń, rozdzielając średnikiem:

```
$ date ; who
pon 01 cze 15:36:09 EST 2020
Christine tty2      2020-06-01 15:26
Samantha  tty3      2020-06-01 15:26
Timothy   tty1      2020-06-01 15:26
user      tty7      2020-06-01 14:03 (:0)
user      pts/0     2020-06-01 15:21 (:0.0)
$
```

Gratulacje, to był Twój pierwszy skrypt! W tym prostym przykładzie użyto dwóch poleceń powłoki Bash. Polecenie `date` zostaje wykonane pierwsze i powoduje wyświetlenie aktualnej daty i godziny. Następnie zostaje wykonane polecenie `who`, które pokazuje, kto jest aktualnie zalogowany w systemie. Przy użyciu tej techniki można połączyć dowolną liczbę poleceń, pod warunkiem że łączna liczba zawartych w nich znaków nie przekroczy 255.

Choć w przypadku niewielkich skryptów ta technika dobrze się sprawdza, jej poważną wadą jest to, że za każdym razem, gdy chcemy wykonać skrypt, całe polecenie musimy wpisywać w wierszu poleceń od nowa. Aby tego nie robić, wszystkie polecenia można zapisać w pliku tekstowym. Potem taki plik można zwyczajnie uruchomić, gdy trzeba wykonać znajdujące się w nim polecenia.

Tworzenie pliku skryptu

Aby zapisać polecenia w pliku tekstowym, należy użyć edytora tekstu (rozdział 10.).

Podczas tworzenia pliku skryptu powłoki w jego pierwszym wierszu należy określić używaną powłokę za pomocą następującego formatu:

```
#!/bin/bash
```

W normalnych skryptach powłoki krzyżyk (`#`) oznacza komentarz. Komentarze znajdujące się w skryptach powłoki nie są przetwarzane przez powłokę. Jednak pierwszy wiersz w takim pliku jest wyjątkowy i w jego przypadku krzyżyk z wykrzyknikiem informuje powłokę, w jakiej powłoce ma zostać wykonany dany skrypt (tak, można używać powłoki Bash i wykonywać skrypty przy użyciu innej powłoki).

Po określeniu powłoki można wpisać polecenia, po jednym na wiersz. W razie potrzeby za pomocą krzyżyka można też wprowadzać komentarze, np.:

```
#!/bin/bash
# ten skrypt wyświetla datę i zalogowanych użytkowników
date
who
```

To wszystko. Ewentualnie można umieścić dwa polecenia w jednym wierszu, rozdzielając je średnikiem, ale w skryptach powłoki polecenia można umieszczać także w osobnych wierszach. Skrypt wykona je w kolejności wpisania w pliku.

Zwróć też uwagę na dodatkowy wiersz zaczynający się od krzyżyka, który jest komentarzem. Wiersze zaczynające się od tego znaku (z wyjątkiem pierwszego wiersza zaczynającego się od znaków `#!`) nie są interpretowane przez powłokę. To świetny sposób na pozostawianie informacji dla samego siebie o tym, co się dzieje w danym miejscu skryptu, aby po paru latach móc sobie to szybko przypomnieć.

Zapisz ten skrypt w pliku o nazwie `test1`, ale zanim będzie można go uruchomić, należy wykonać jeszcze parę czynności.

Jeśli teraz spróbujesz wykonać swój plik, to czeka Cię rozczarowanie:

```
$ test1
bash: test1: command not found
$
```


Przed wszystkim musimy sprawić, aby powłoka Bash znalazła nasz plik skryptu. Może pamiętasz z rozdziału 6., że system Linux znajduje polecenia dzięki zmiennej środowiskowej o nazwie PATH. Wyświetlenie jej zawartości pozwala się dowiedzieć, w czym tkwi nasz problem:

```
$ echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/bin:/usr/bin
:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/user/bin $
```

Zmienna PATH określa tylko kilka katalogów, w których mogą się znajdować polecenia. Aby powłoka znalazła skrypt *test1*, musimy wykonać jedną z dwóch czynności:

- dodać katalog zawierający nasz plik skryptu do zmiennej środowiskowej PATH;
- podać bezwzględną lub względną ścieżkę do pliku skryptu w wierszu poleceń.



Niektóre dystrybucje Linuksa mają katalog *\$HOME/bin* w zmiennej środowiskowej PATH. W ten sposób zostaje utworzone miejsce w katalogu głównym każdego użytkownika, w którym można umieszczać pliki przeznaczone do wykonywania przez powłokę.

W tym przykładzie skorzystamy z drugiej metody, aby poinformować powłokę, gdzie dokładnie znajduje się nasz plik skryptu. Przypomnijmy, że do pliku znajdującego się w bieżącym katalogu można się odnieść za pomocą kropki:

```
$ ./test1
bash: ./test1: Permission denied
$
```

Teraz powłoka znalazła skrypt, ale pojawił się nowy problem. Otrzymaliśmy informację, że nie mamy uprawnień do wykonania tego pliku. Wystarczy zerknąć na jego uprawnienia, aby się dowiedzieć, w czym tkwi problem:

```
$ ls -l test1
-rw-r--r-- 1 user user 73 Jun 02 15:36 test1
$
```

Kiedy tworzyliśmy plik *test1*, jego domyślne uprawnienia zostały zdeterminowane przez wartość *umask*, a ponieważ zmienna *umask* ma wartość 022 (rozdział 7.), system nadał właścicielowi tego pliku uprawnienia tylko do jego odczytu i zapisu.

Następnym krokiem jest więc nadanie właścicielowi uprawnień do wykonywania pliku za pomocą polecenia *chmod* (rozdział 7.):

```
$ chmod u+x test1
$ ./test1
pon 01 cze 15:38:19 EST 2020
Christine tty2 2020-06-01 15:26
Samantha tty3 2020-06-01 15:26
Timothy tty1 2020-06-01 15:26
user tty7 2020-06-01 14:03 (:0)
user pts/0 2020-06-01 15:21 (:0.0) $
```

Sukces! Wszystkie elementy układanki znalazły się na swoim miejscu, dzięki czemu udało się wykonać nowy plik skryptu powłoki.

Wyświetlanie komunikatów

Większość poleceń powłoki generuje własne wyniki, wyświetlane w konsoli, w której został uruchomiony skrypt. Czasami jednak chcielibyśmy dodać własne wiadomości tekstowe, które pomagają użytkownikowi skryptu zrozumieć, co się dzieje. Służy do tego polecenie `echo`, które wyświetla proste łańcuchy tekstu wpisane za nim:

```
$ echo To jest test
To jest test
$
```

Zauważ, że domyślnie nie trzeba ujmować tekstu do wyświetlenia w cudzysłowach. Czasami jednak, kiedy tekst zawiera cudzysłowy, sprawy mogą się komplikować. Spójrz na poniższy przykład w języku angielskim:

```
$ echo Let's see if this'll work
Lets see if thisll work
$
```

Polecenie `echo` do oznaczania granic łańcuchów tekstu używa prostych cudzysłowów pojedynczych lub podwójnych. Jeśli jednego z tych dwóch typów użyjesz w swoim tekście, to cały łańcuch musisz ująć w drugim:

```
$ echo "This is a test to see if you're paying attention"
This is a test to see if you're paying attention
$ echo 'Rich says "scripting is easy".'
Rich says "scripting is easy".
$
```

Teraz wszystkie cudzysłowy są prawidłowo wyświetlone w wyniku.

Polecenie `echo` można umieścić w dowolnym miejscu skryptu, w którym chcemy wyświetlić dodatkowe informacje:

```
$ cat test1
#!/bin/bash
# ten skrypt wyświetla datę i zalogowanych użytkowników
echo Data i godzina:
date
echo "Sprawdźmy, kto jest zalogowany:"
who
$
```

Wynik działania tego skryptu będzie następujący:

```
$ ./test1
Data i godzina:
pon 01 cze 15:41:13 EST 2020
Sprawdźmy, kto jest zalogowany:
Christine tty2      2020-06-01 15:26
Samantha tty3      2020-06-01 15:26
Timothy tty1       2020-06-01 15:26
user tty7          2020-06-01 14:03 (:0)
user pts/0         2020-06-01 15:21 (:0.0)
$
```

W porządku, ale jak wyświetlić łańcuch tekstu w tym samym wierszu co wynik polecenia? W tym celu można użyć parametru `-n` instrukcji `echo`. W pierwszej naszej instrukcji `echo` należałoby wprowadzić taką zmianę:

```
echo -n "Data i godzina: "
```

W tym przypadku łańcuch do wyświetlenia musimy umieścić w cudzysłowie, aby na jego końcu móc dodać spację. Wynik polecenia zaczyna się dokładnie w miejscu zakończenia łańcucha. Teraz wynik całego skryptu wygląda tak:

```
$ ./test1
Data i godzina: pon 01 cze 15:41:13 EST 2020
Sprawdźmy, kto jest zalogowany:
Christine tty2      2020-06-01 15:26
Samantha  tty3      2020-06-01 15:26
Timothy   tty1      2020-06-01 15:26
user      tty7      2020-06-01 14:03 (:0)
user      pts/0     2020-06-01 15:21 (:0.0)
$
```

Doskonale! Polecenie `echo` to niezwykle istotny element w skryptach powłoki, które oddziałują z użytkownikiem. Będziesz go używać bardzo często, szczególnie przy wyświetlaniu wartości zmiennych skryptów. Sprawdźmy więc, jak się to robi.

Zmienne

Wykonywanie pojedynczych poleceń w skryptach powłoki to przydatna, choć ograniczona umiejętność. W poleceniach powłoki często chcemy umieszczać różne dane, które powinny zostać przetworzone. Do tego celu można używać **zmiennych**, które pozwalają na czasowe przechowywanie informacji w skrypcie, aby były dostępne dla poleceń. W tym podrozdziale pokazujemy, jak posługiwać się zmiennymi w skryptach powłoki.

Zmienne środowiskowe

Znasz już jeden rodzaj zmiennych Linuksa — zmienne środowiskowe, które opisaliśmy w rozdziale 6. Dostęp do ich wartości można uzyskać także w skryptach powłoki.

Powłoka przechowuje zmienne środowiskowe zawierające różne informacje na temat systemu, takie jak nazwa systemu, nazwa zalogowanego użytkownika, identyfikator systemowy użytkownika (tzw. UID), domyślny katalog główny użytkownika i ścieżka wyszukiwania używana przez powłokę do znajdowania programów. Listę wszystkich aktywnych zmiennych środowiskowych można uzyskać za pomocą polecenia `set`:

```
$ set
BASH=/bin/bash
...
HOME=/home/Samantha
HOSTNAME=localhost.localdomain
HOSTTYPE=i386
IFS=$' \t\n'
IMSETTINGS_INTEGRATE_DESKTOP=yes
IMSETTINGS_MODULE=none
LANG=en_US.utf8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=24
LOGNAME=Samantha
...
```

Do zawartości tych zmiennych środowiskowych w skryptach można się dostać przez podanie nazwy poprzedzonej znakiem dolara, jak pokazano w poniższym przykładzie:

```
$ cat test2
#!/bin/bash
# wyświetla informacje o użytkownikach z systemu
echo "Informacje na temat użytkownika $USER"
echo UID: $UID
echo HOME: $HOME
$
```

Zmienne środowiskowe \$USER, \$UID i \$HOME umożliwiają wyświetlenie informacji o zalogowanym użytkowniku. Wyniki powinny wyglądać tak:

```
$ chmod u+x test2
$ ./test2
Informacje na temat użytkownika Samantha
UID: 1001
HOME: /home/Samantha
$ $
```

Zwróć uwagę, że zmienne środowiskowe użyte w poleceniu echo w chwili wykonania skryptu zostały zastąpione przez ich obecne wartości. Zauważ też, że choć w pierwszym łańcuchu umieściliśmy zmienną systemową \$USER w podwójnym cudzysłowie, to skrypt i tak prawidłowo odczytał nasz zamiar. Ta metoda ma jednak pewną wadę. Spójrz na poniższy przykład:

```
$ echo " The cost of the item is $15"
The cost of the item is 5
```

Zdecydowanie nie o to nam chodziło. Kiedy interpreter skryptu znajduje znak dolara w obrębie cudzysłowu, zakłada, że jest to odwołanie do zmiennej. W tym przykładzie skrypt spróbował wyświetlić wartość zmiennej \$1 (która nie jest zdefiniowana), a po niej wyświetlił liczbę 5. Aby wyświetlić znak dolara, należy go poprzedzić wstecznym ukośnikiem:

```
$ echo "The cost of the item is \$15"
The cost of the item is $15
```

Teraz jest lepiej. Ukośnik sprawił, że znak dolara w skrypcie został zinterpretowany jako zwykły znak, a nie oznaczenie zmiennej. W następnej sekcji pokazujemy, jak tworzyć własne zmienne w skryptach.



Można spotkać także zmienne w formacie `${zmienna}`. Klamra jest często używana do oddzielania nazwy zmiennej od znaku dolara.

Zmienne użytkownika

W skryptach powłoki można używać nie tylko zmiennych środowiskowych, ale także własnych zmiennych. Pozwala to na tymczasowe przechowywanie danych do użytku w różnych miejscach skryptu, co czyni go bardziej podobnym do prawdziwego programu komputerowego.

Nazwa zmiennej użytkownika może zawierać maksymalnie 20 znaków — liter, cyfr i znaków podkreślenia. Wielkość liter ma znaczenie, a więc `var1` różni się od `var1`. Ta zasada często wpędza początkujących programistów w kłopoty.

Wartość zmiennej przypisuje się za pomocą znaku równości. Między nazwą zmiennej, znakiem równości i wartością nie może być ani jednej spacji (kolejna pułapka na początkujących). Oto kilka przykładów przypisania wartości zmiennym użytkownika:

```
var1=10
var2=-57
var3=testowanie
var4="dalsze testowanie"
```

Skrypt powłoki przechowuje wszystkie zmienne jako łańcuchy tekstu i to od poleceń zależy ich interpretacja jako określonego typu danych. Zmienne zdefiniowane w skrypcie powłoki istnieją przez cały czas działania skryptu i zostają usunięte wraz z jego zakończeniem.

Do zmiennych użytkownika, tak jak do zmiennych systemowych, można się odwoływać za pomocą znaku dolara:

```
$ cat test3
#!/bin/bash
# testowanie zmiennych
days=10
guest="Katie"
echo "$guest zameldowała się $days dni temu."
days=5
guest="Jessica"
echo "$guest zameldowała się $days dni temu."
$
```

Wynik działania tego skryptu jest następujący:

```
$ chmod u+x test3
$ ./test3
Katie zameldowała się 10 dni temu.
Jessica zameldowała się 5 dni temu.
$
```

W miejscu każdego odwołania do zmiennej jest wstawiana aktualnie przypisana jej wartość. Należy pamiętać, że znaku dolara używa się tylko w odwołaniach do wartości zmiennych, natomiast do przypisywania wartości zmiennym znaku tego się nie używa. Poniższy przykład ilustruje, co mamy na myśli:

```
$ cat test4
#!/bin/bash
# przypisanie wartości zmiennej do innej zmiennej

value1=10
value2=$value1
echo Otrzymana wartość to $value2
$
```

W instrukcji przypisania wartości zmiennej `value1` innej zmiennej użycie znaku dolara było konieczne. Wynik wykonania tego kodu jest następujący:

```
$ chmod u+x test4
$ ./test4
Otrzymana wartość to 10
$
```

Jeśli zapomnimy dodać znak dolara i zmiennej `value2` przypiszemy wartość w taki sposób:

```
value2=value1
```

to otrzymamy następujący wynik:

```
$ ./test4
Otrzymana wartość to value1
$
```

Bez znaku dolara powłoka interpretuje nazwę zmiennej jako zwykły tekst, co prawie na pewno jest błędem.

Podmienianie poleceń

Jedną z najbardziej przydatnych cech skryptów powłoki jest możliwość wyciągania informacji z wyników polecenia i przypisywania ich do zmiennych. Po przypisaniu wyników do zmiennej można jej używać w dowolnym miejscu skryptu. Przydaje się to, kiedy przetwarzamy dane w swoich skryptach.

Wynik polecenia można przypisać do zmiennej na dwa sposoby:

- za pomocą odwróconego apostrofu (```),
- za pomocą formatu `$()`.

Zwróć uwagę na odwrócony apostrof — to nie jest zwykły pojedynczy znak cudzysłowu używany w łańcuchach tekstu. Nie jest on zbyt często używany poza skryptami powłoki, przez co wiele osób nawet nie wie o jego obecności na klawiaturze. Musisz jednak się z nim zapoznać, ponieważ w wielu skryptach powłoki odgrywa bardzo ważną rolę. Podpowiedź: na standardowej klawiaturze zazwyczaj znajduje się w tym samym miejscu co tylda (`~`).

Technika zastępowania poleceń umożliwia przypisanie wyniku polecenia powłoki do zmiennej. Choć nie wydaje się to niczym niesamowitym, jest to jedna z najważniejszych technik w programowaniu skryptowym.

Całe polecenie należy umieścić między odwrotnymi apostrofami:

```
testing=`date`
```

lub należy użyć formatu `$()`:

```
testing=$(date)
```

Powłoka wykonuje polecenia znajdujące się między znakami podmieniania poleceń i przypisuje wynik do zmiennej `testing`. Zwróć uwagę na brak spacji przed i za znakiem równości. Poniżej znajduje się przykład utworzenia zmiennej przy użyciu wyniku zwróconego przez normalne polecenie powłoki:

```
$ cat test5
#!/bin/bash
testing=$(date)
echo "Data i godzina: " $testing
$
```

Zmienna `testing` otrzymuje wynik polecenia `date` i zostaje użyta w poleceniu `echo`, które wyświetla jej zawartość. Wynik wykonania powyższego skryptu wyglądałby tak:

```
$ chmod u+x test5
$ ./test5
Data i godzina: pon 01 cze 15:45:25 EDT 2020
$
```

Ten przykład nie jest jakoś specjalnie fascynujący (równie dobrze polecenie można było umieścić w instrukcji `echo`), ale z wynikiem polecenia zapisanym w zmiennej można zrobić niemal wszystko.

Poniżej znajduje się popularny przykład pokazujący, jak za pomocą podmieniania poleceń pobrać bieżącą datę i użyć jej do utworzenia niepowtarzalnej nazwy pliku w skrypcie:

```
#!/bin/bash
# kopiuje listing zawartości katalogu /usr/bin do pliku dziennika
today=$(date +%y%m%d)
ls /usr/bin -al > log.$today
```

Zmiennej `today` zostaje przypisany wynik sformatowanego polecenia `date`. Ta technika jest często używana w celu pobierania daty do tworzenia nazw plików dziennika. Format `+%y%m%d` nakazuje poleceniu `date` przedstawienie daty w postaci dwucyfrowych oznaczeń roku, miesiąca i dnia:

```
$ date +%y%m%d
200601
$
```

Ten skrypt przypisuje zmiennej wartość, która następnie zostaje wykorzystana przy tworzeniu nazwy pliku. Sam plik zawiera przekierowany wynik (szerzej na ten temat piszemy w punkcie „Przekierowywanie wejścia i wyjścia”) listingu zawartości katalogu. Po uruchomieniu tego skryptu w katalogu powinien się pojawić nowy plik:

```
-rw-r--r--  1 user  user    769 01 cze 16:15 log.200601
```

Nazwa pliku dziennika, który pojawił się w katalogu, zawiera wartość zmiennej `$today`. Sam plik zawiera listę zawartości katalogu `/usr/bin`. Jeśli ten skrypt zostanie uruchomiony następnego dnia, utworzy plik dziennika o nazwie `log.200602` odpowiadającej nowemu dniowi.



Technika podmieniania poleceń tworzy tzw. podpowłokę w celu wykonania danego polecenia. Podpowłoka to osobna powłoka wygenerowana z powłoki wykonującej skrypt. Z tego powodu wszystkie zmienne utworzone w skrypcie są niedostępne poleceniom wykonywanym w podpowłoce.

Podpowłoki są także tworzone w czasie wykonywania poleceń w wierszu poleceń przy użyciu ścieżki `./`, natomiast nie są tworzone, jeżeli polecenie zostanie wykonane bez podania ścieżki. Jeśli jednak użyjesz wbudowanego polecenia powłoki, to podpowłoka nie zostanie wygenerowana. Zachowaj ostrożność podczas wykonywania skryptów w wierszu poleceń powłoki!

Przekierowywanie wejścia i wyjścia

Czasami wyniki polecenia chcemy nie tylko wyświetlić na monitorze, ale także chcemy je gdzieś zapisać. Powłoka Bash udostępnia kilka operatorów, które umożliwiają **przekierowywanie** wyników polecenia do innego miejsca (np. pliku). Przekierowanie może dotyczyć zarówno wejścia, jak i wyjścia, a także może oznaczać przekazanie pliku na wejście polecenia. W tym podrozdziale opisujemy, co należy zrobić, aby zastosować przekierowanie w swoich skryptach powłoki.

Przekierowywanie wyjścia

Najbardziej podstawowym typem przekierowania jest wysłanie wyników polecenia do pliku. W powłocie Bash służy do tego znak większości (>):

```
polecenie > plikwyjściowy
```

Wszystko, co polecenie wyświetliłoby na monitorze, zostanie zapisane w pliku o podanej nazwie:

```
$ date > test6
$ ls -l test6
-rw-r--r-- 1 user  user          29 Jun 01 16:56 test6
$ cat test6
pon 01 cze 16:56:58 EDT 2020
$
```

Operator przekierowania utworzył plik o nazwie *test6* (przy użyciu domyślnych ustawień maski *umask*) i przekierował do niego wynik z polecenia *date*. Gdyby ten plik już istniał, operator zastąpiłby jego zawartość nowymi danymi:

```
$ who > test6
$ cat test6
rich pts/0      01 cze 16:55
$
```

Teraz plik *test6* zawiera wynik polecenia *who*.

Czasami zamiast nadpisywać zawartość pliku, wolimy dodać do niej wyniki polecenia — np. kiedy tworzymy plik dziennika mający zawierać dokumentację czynności wykonywanych w systemie. W takiej sytuacji, aby dodać nowe dane na końcu, możemy użyć podwójnego znaku większości (>>):

```
$ date >> test6
$ cat test6
rich pts/0      01 cze 16:55
pon 01 cze 01 17:02:14 EDT 2020
$
```

Plik *test6* nadal zawiera wcześniejsze dane z poprzednio wykonanego polecenia *who* i dodatkowo zawiera nowe wyniki z polecenia *date*.

Przekierowywanie wejścia

Przekierowywanie wejścia to odwrotność przekierowywania wyjścia. Zamiast przekierowywać wyniki polecenia do pliku, pobieramy zawartość pliku i kierujemy ją do polecenia.

Symbolem tej operacji jest znak mniejszości (<):

```
polecenie < plikwejściowy
```

Łatwym sposobem na zapamiętanie tego jest zwrócenie uwagi na fakt, że polecenie zawsze znajduje się na pierwszym miejscu w wierszu, a symbol „wskazuje” kierunek przepływu danych. Znak mniejszości oznacza, że dane płyną z pliku wejściowego do polecenia.

Poniżej znajduje się przykład użycia przekierowania wejścia z poleceniem *wc*:

```
$ wc < test6
  2   11   60
$
```


Polecenie `wc` zwraca dane statystyczne dotyczące tekstu. Domyślnie uwzględnia trzy wartości:

- liczbę wierszy tekstu,
- liczbę słów,
- liczbę bajtów.

Przekierowując plik tekstowy do polecenia `wc`, można szybko się dowiedzieć, ile zawiera wierszy i słów oraz ile zajmuje bajtów. Z przykładu wynika, że plik `test6` zawiera 2 wiersze i 11 słów i zajmuje 60 bajtów.

Istnieje jeszcze jedna metoda przekierowywania wejścia, która nazywa się **śródliniowym przekierowywaniem wejścia** (ang. *inline input redirection*). Za jej pomocą dane wejściowe do przekierowania można określić w wierszu poleceń zamiast w pliku. W pierwszej chwili może się to wydać dziwne, ale metoda ta ma kilka zastosowań (takich jak opisane w podrozdziale „Wykonywanie działań matematycznych”).

Symbol śródliniowego przekierowania wejścia to podwójny znak mniejszości (`<<`). Oprócz niego należy określić marker tekstowy oznaczający początek i koniec danych używanych na wejściu. Może to być dowolny łańcuch, oby na początku i na końcu danych był taki sam:

```
 polecenie << marker
 dane
 marker
```

Podczas korzystania ze śródliniowego przekierowania wejścia w wierszu poleceń powłoka będzie prosić o dane przy użyciu dodatkowego wiersza poleceń zdefiniowanego w zmiennej środowiskowej `PS2` (rozdział 6.). Tak to wygląda w praktyce:

```
$ wc << EOF
> test łańcuch 1
> test łańcuch 2
> test łańcuch 3
> EOF
      3      9      45
$
```

Dodatkowy wiersz poleceń prosi o podawanie kolejnych porcji danych, aż użytkownik wpisze marker oznaczający koniec. Polecenie `wc` policzy wiersze, słowa i bajty wprowadzonych danych.

Potoki

Czasami trzeba wysłać dane wyjściowe jednego polecenia na wejście innego polecenia. Można to zrobić za pomocą przekierowywania, choć jest to dość niezgrabne rozwiązanie:

```
$ rpm -qa > rpm.list
$ sort < rpm.list
abattis-cantarell-fonts-0.0.25-1.e17.noarch
abrt-2.1.11-52.e17.centos.x86_64
abrt-addon-ccpp-2.1.11-52.e17.centos.x86_64
abrt-addon-kernelevents-2.1.11-52.e17.centos.x86_64
abrt-addon-pstoreoops-2.1.11-52.e17.centos.x86_64
abrt-addon-python-2.1.11-52.e17.centos.x86_64
abrt-addon-vmcore-2.1.11-52.e17.centos.x86_64
abrt-addon-xorg-2.1.11-52.e17.centos.x86_64
abrt-cli-2.1.11-52.e17.centos.x86_64
abrt-console-notification-2.1.11-52.e17.centos.x86_64
...
```

Polecenie `rpm` zarządza pakietami oprogramowania zainstalowanymi w systemach zawierających narzędzie Red Hat Package Management (RPM), a więc takich jak np. CentOS. Po dodaniu parametrów `-qa` zwraca listę zainstalowanych pakietów, ale nie sortuje jej w żadnej konkretnej kolejności. Jeśli szukasz konkretnego pakietu lub grupy pakietów, to w wynikach polecenia `rpm` może być Ci trudno je znaleźć.

Za pomocą standardowego przekierowania wyników przekierowano wyniki polecenia `rpm` do pliku o nazwie `rpm.list`. Gdy polecenie zakończyło działanie, w pliku `rpm.list` znalazła się lista wszystkich zainstalowanych pakietów oprogramowania w systemie. Następnie przekierowanie wejścia zostało użyte w celu wysłania zawartości pliku `rpm.list` do polecenia `sort` w celu alfabetycznego posortowania nazw pakietów.

Udało się, ale to też był niezbyt zgrabny sposób na uzyskanie informacji. Zamiast przekierowywać wyniki polecenia do pliku, można przekierować je do innego polecenia. W tym procesie wykorzystuje się tzw. **potok**.

Symbol potoku, podobnie jak znak podmieniania poleceń (`'`), poza skryptami powłoki jest rzadko używany. Składa się z dwóch pionowych kresek ustawionych jedna nad drugą, ale w druku zazwyczaj wygląda jak jedna pionowa kreska (`|`). Na standardowej klawiaturze ten znak zazwyczaj znajduje się na tym samym klawiszu co ukośnik wsteczny (`\`). Potok umieszcza się między poleceniami, aby przekierować wyniki jednego do drugiego:

```
 polecenie1 | polecenie2
```

Nie wyobrażaj sobie potoku jako wykonywania dwóch poleceń obok siebie. System Linux wykonuje je jednocześnie, łącząc je wewnętrznie. Kiedy pierwsze polecenie zwróci wynik, zostaje on natychmiast wysłany do drugiego polecenia. Do przesyłania danych nie są używane żadne pośrednie pliki ani bufor.

Teraz za pomocą potoku możemy bez problemu przekazać wyniki polecenia `rpm` bezpośrednio do polecenia `sort`, aby uzyskać posortowaną listę:

```
$ rpm -qa | sort
abattis-cantarell-fonts-0.0.25-1.e17.noarch
abrt-2.1.11-52.e17.centos.x86_64
abrt-addon-ccpp-2.1.11-52.e17.centos.x86_64
abrt-addon-kernelevents-2.1.11-52.e17.centos.x86_64
abrt-addon-pstoreoops-2.1.11-52.e17.centos.x86_64
abrt-addon-python-2.1.11-52.e17.centos.x86_64
abrt-addon-vmcore-2.1.11-52.e17.centos.x86_64
abrt-addon-xorg-2.1.11-52.e17.centos.x86_64
abrt-cli-2.1.11-52.e17.centos.x86_64
abrt-console-notification-2.1.11-52.e17.centos.x86_64
...
```

Jeśli nie umiesz czytać z niezwykłą prędkością, to pewnie nie udało Ci się wiele zobaczyć w wynikach wygenerowanych przez to polecenie. Potok działa na bieżąco, co oznacza, że gdy tylko polecenie `rpm` wygeneruje dane, polecenie `sort` natychmiast bierze się za ich obróbkę. Zanim polecenie `rpm` zakończy zwracanie danych, polecenie `sort` ma je już posortowane i rozpoczyna wyświetlanie ich na ekranie monitora.

W poleceniu można użyć nieograniczonej liczby potoków, tzn. można dowolnie długo przekazywać wyniki kolejnych poleceń do następnych, aby uzyskać dokładnie to, czego się potrzebuje.

Jako że w tym przypadku wyniki polecenia `sort` bardzo szybko przelatują przed oczami, można użyć jednego z poleceń paginacji tekstu (np. `less` lub `more`), aby wymusić zatrzymywanie wyników na każdym ekranie danych:

```
$ rpm -qa | sort | more
```

W tej sekwencji najpierw zostaje wykonane polecenie `rpm`, którego wyniki zostają przekazane do polecenia `sort`, którego wyniki z kolei zostaną przesłane do polecenia `more` w celu ich wyświetlenia po jednym ekranie na raz. Teraz możesz zatrzymywać wyświetlanie danych, aby je spokojnie przejrzeć, jak pokazano na rysunku 11.1.

```

a@localhost:~
File Edit View Search Terminal Help
[a@localhost ~]$ rpm -ga | sort | more
rpm: one type of query/verify may be performed at a time
[a@localhost ~]$ rpm -qa | sort | more
abattis-cantarell-fonts-0.0.25-1.el7.noarch
abrt-2.1.11-60.el7.centos.x86_64
abrt-addon-ccpp-2.1.11-60.el7.centos.x86_64
abrt-addon-kerneloops-2.1.11-60.el7.centos.x86_64
abrt-addon-pstoreoops-2.1.11-60.el7.centos.x86_64
abrt-addon-python-2.1.11-60.el7.centos.x86_64
abrt-addon-vmcore-2.1.11-60.el7.centos.x86_64
abrt-addon-xorg-2.1.11-60.el7.centos.x86_64
abrt-cli-2.1.11-60.el7.centos.x86_64
abrt-console-notification-2.1.11-60.el7.centos.x86_64
abrt-dbus-2.1.11-60.el7.centos.x86_64
abrt-desktop-2.1.11-60.el7.centos.x86_64
abrt-gui-2.1.11-60.el7.centos.x86_64
abrt-gui-libs-2.1.11-60.el7.centos.x86_64
abrt-libs-2.1.11-60.el7.centos.x86_64
abrt-python-2.1.11-60.el7.centos.x86_64
abrt-retrace-client-2.1.11-60.el7.centos.x86_64
abrt-tui-2.1.11-60.el7.centos.x86_64
accountsservice-0.6.50-7.el7.x86_64
accountsservice-libs-0.6.50-7.el7.x86_64
acl-2.2.51-15.el7.x86_64
adcli-0.8.1-15.el7.x86_64
adobe-mappings-cmap-20171205-3.el7.noarch
-More-

```

RYSUNEK 11.1. Przesłanie danych do polecenia `more` za pomocą potoku

Aby było ciekawiej, wraz z potokiem można użyć przekierowania, by zapisać wyniki do pliku:

```

$ rpm -qa | sort > rpm.list
$ more rpm.list
abrt-1.1.14-1.fc14.i686
abrt-addon-ccpp-1.1.14-1.fc14.i686
abrt-addon-kerneloops-1.1.14-1.fc14.i686
abrt-addon-python-1.1.14-1.fc14.i686
abrt-desktop-1.1.14-1.fc14.i686
abrt-gui-1.1.14-1.fc14.i686
abrt-libs-1.1.14-1.fc14.i686
abrt-plugin-bugzilla-1.1.14-1.fc14.i686
abrt-plugin-logger-1.1.14-1.fc14.i686
abrt-plugin-runapp-1.1.14-1.fc14.i686
acl-2.2.49-8.fc14.i686
...

```

Zgodnie z oczekiwaniami teraz dane w pliku *rpm.list* są posortowane!

Zdecydowanie najpopularniejszym sposobem wykorzystania potoków jest przesyłanie wyników poleceń generujących duże ilości danych do polecenia *more*. W szczególności dotyczy to polecenia *ls*, jak pokazano na rysunku 11.2.

```

a@localhost:/etc
File Edit View Search Terminal Help
[a@localhost etc]$ ls -al | more
total 1432
drwxr-xr-x. 144 root root      8192 Sep 13 04:41 .
dr-xr-xr-x.  17 root root      224 Sep 13 04:40 ..
drwxr-xr-x.   3 root root      101 Sep 13 04:35 abrt
-rw-r--r--.   1 root root       16 Sep 13 04:40 adjtime
-rw-r--r--.   1 root root     1529 Mar 31 2020 aliases
-rw-r--r--.   1 root root    12288 Sep 13 04:41 aliases.db
drwxr-xr-x.   3 root root       65 Sep 13 04:36 alsa
drwxr-xr-x.   2 root root     4096 Sep 13 04:39 alternatives
-rw-----.   1 root root     541 Aug  8 2019 anacrontab
-rw-r--r--.   1 root root     55 Aug  8 2019 asound.conf
-rw-r--r--.   1 root root       1 Oct 30 2018 at.deny
drwxr-x----.  3 root root       43 Sep 13 04:35 audisp
drwxr-x----.  3 root root       83 Sep 13 04:41 audit
-rw-r--r--.   1 root root    15137 Sep 30 2020 autofs.conf
-rw-----.   1 root root     232 Sep 30 2020 autofs_ldap_auth.conf
-rw-r--r--.   1 root root     795 Sep 30 2020 auto.master
drwxr-xr-x.   2 root root       6 Sep 30 2020 auto.master.d
-rw-r--r--.   1 root root     524 Sep 30 2020 auto.misc
-rwxr-xr-x.   1 root root     1260 Sep 30 2020 auto.net
--More--

```

RYSUNEK 11.2. Użycie polecenia *more* w połączeniu z poleceniem *ls*

Polecenie *ls -l* generuje długą listę wszystkich plików znajdujących się w katalogu.

W przypadku katalogów zawierających dużo plików ta lista naprawdę może być bardzo długa. Przez przekazanie wyników do polecenia *more* wymusiliśmy zatrzymywanie prezentacji danych po każdym ekranie.

Wykonywanie działań matematycznych

Kolejną niezbędną cechą każdego języka programowania jest możliwość wykonywania działań na liczbach. Niestety w skryptach powłoki wykonywanie działań matematycznych jest dość toporne i może być realizowane na dwa sposoby.

Polecenie *expr*

Pierwotnie powłoka Bourne zawierała specjalne polecenie służące do obliczania równań matematycznych. Miało ono nazwę *expr* i umożliwiało wykonywanie obliczeń w wierszu poleceń, ale nie sprawiało dobrego wrażenia:

```
$ expr 1 + 5
6
```

Polecenie *expr* rozpoznaje kilka różnych operatorów matematycznych i łańcuchowych, które wymieniono w tabeli 11.1.

TABELA 11.1. Operatory polecenia `expr`

Operator	Opis
$ARG1 \mid ARG2$	Zwraca $ARG1$, jeśli żaden argument nie jest <code>nu11</code> ani <code>0</code> . W przeciwnym razie zwraca $ARG2$
$ARG1 \& ARG2$	Zwraca $ARG1$, jeśli żaden z argumentów nie jest <code>nu11</code> ani <code>0</code> . W przeciwnym razie zwraca <code>0</code>
$ARG1 < ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ jest mniejszy od $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 <= ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ jest nie większy od $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 = ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ jest równy $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 != ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ nie jest równy $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 >= ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ jest nie mniejszy niż $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 > ARG2$	Zwraca <code>1</code> , jeśli $ARG1$ jest większy od $ARG2$. W przeciwnym razie zwraca <code>0</code>
$ARG1 + ARG2$	Zwraca sumę arytmetyczną $ARG1$ i $ARG2$
$ARG1 - ARG2$	Zwraca różnicę arytmetyczną $ARG1$ i $ARG2$
$ARG1 * ARG2$	Zwraca iloczyn arytmetyczny $ARG1$ i $ARG2$
$ARG1 / ARG2$	Zwraca wynik dzielenia arytmetycznego $ARG1$ przez $ARG2$
$ARG1 \% ARG2$	Zwraca resztę z dzielenia arytmetycznego $ARG1$ przez $ARG2$
$\text{\textasciitilde{LAŃCUCH}} : REGEXP$	Zwraca dopasowanie wzorca, jeśli $REGEXP$ pasuje do wzorca w ciągu $\text{\textasciitilde{LAŃCUCH}}$
<code>match</code> $\text{\textasciitilde{LAŃCUCH}} REGEXP$	Zwraca dopasowanie wzorca, jeśli $REGEXP$ pasuje do wzorca w ciągu $\text{\textasciitilde{LAŃCUCH}}$
<code>substr</code> $\text{\textasciitilde{LAŃCUCH}} \text{\textasciitilde{POZ}} \text{\textasciitilde{DŁUGOŚĆ}}$	Zwraca podłańcuch ciągu $\text{\textasciitilde{LAŃCUCH}}$ o długości $\text{\textasciitilde{DŁUGOŚĆ}}$, zaczynający się na pozycji $\text{\textasciitilde{POZ}}$ (zaczyna liczenie od 1)
<code>index</code> $\text{\textasciitilde{LAŃCUCH}} \text{\textasciitilde{ZNAKI}}$	Zwraca pozycję w ciągu znaków $\text{\textasciitilde{LAŃCUCH}}$, na której znaleziono $\text{\textasciitilde{ZNAKI}}$. W przeciwnym razie zwraca <code>0</code>
<code>length</code> $\text{\textasciitilde{LAŃCUCH}}$	Zwraca liczbę określającą długość ciągu $\text{\textasciitilde{LAŃCUCH}}$
<code>+</code> $TOKEN$	Interpretuje $TOKEN$ jako łańcuch nawet, jeśli jest to słowo kluczowe
$(WYRAŻENIE)$	Zwraca wartość wyrażenia $WYRAŻENIE$

Standardowe operatory dobrze działają w poleceniu `expr`, ale sprawiają problemy w skryptach i wierszu poleceń. Wiele z nich (np. gwiazdka) ma inne znaczenie w powłoce. Ich obecność w poleceniu `expr` sprawia, że otrzymujemy dziwne wyniki:

```
$ expr 5 * 2
expr: syntax error
$
```

Aby rozwiązać ten problem, należy za pomocą znaku modyfikacji (ukośnika wstecznego) oznaczyć wszystkie znaki, które mogłyby zostać źle zinterpretowane przez powłokę:

```
$ expr 5 \* 2
10
$
```

Zaczyna się robić nieelegancko! Posługiwanie się poleceniem `expr` w skryptach wcale nie wygląda lepiej:

```
$ cat test6
#!/bin/bash
# przykład użycia polecenia expr
var1=10
var2=20
```

```
var3=$(expr $var2 / $var1)
echo Wynik wynosi $var3
```

Aby przypisać wynik równania matematycznego do zmiennej, należy użyć techniki podstawiania poleceń, która pozwala pobrać wynik wygenerowany przez wyrażenie `expr`:

```
$ chmod u+x test6
$ ./test6
Wynik wynosi 2
$
```

Na szczęście powłoka Bash ma pewne ulepszenie, ułatwiające przetwarzanie operatorów matematycznych, które opisujemy poniżej.

Nawiasy

Powłoka Bash ma polecenie `expr`, by zachować zgodność z powłoką Bourne, ale oprócz tego umożliwia obliczanie wartości równań matematycznych w znacznie łatwiejszy sposób. W powłoce Bash, kiedy przypisujemy wartość matematyczną zmiennej, równanie możemy umieścić w kwadratowym nawiasie poprzedzonym znakiem dolara (`$[operacja]`):

```
$ var1=$((1 + 5))
$ echo $var1
6
$ var2=$((var1 * 2))
$ echo $var2
12
$
```

Nawiasy znacznie ułatwiają wykonywanie działań matematycznych w porównaniu z poleceniem `expr`. Ta technika działa także w skryptach powłoki:

```
$ cat test7
#!/bin/bash
var1=100
var2=50
var3=45
var4=$((var1 * (var2 - var3)))
echo Ostateczny wynik to $var4
$
```

Ten skrypt zwróci następujący wynik:

```
$ chmod u+x test7
$ ./test7
Ostateczny wynik to 500
$
```

Ponadto zwróć uwagę, że w metodzie obliczania wartości równań z użyciem prostokątnych nawiasów nie istnieje problem dotyczący znaku mnożenia ani błędnej interpretacji jakiegokolwiek innego znaku. Powłoka wie, że to nie jest symbol wieloznaczny, ponieważ znajduje się on w prostokątnym nawiasie.

Wykonywanie działań matematycznych w skryptach powłoki Bash ma jedno ważne ograniczenie. Spójrz na poniższy przykład:

```
$ cat test8
#!/bin/bash
var1=100
```

```
var2=45
var3=$(( $var1 / $var2 ))
echo Ostateczny wynik to $var3
$
```

Teraz wykonaj ten skrypt i zobacz, co się stanie:

```
$ chmod u+x test8
$ ./test8
Ostateczny wynik to 2
$
```

Operatory matematyczne powłoki Bash obsługują tylko arytmetykę całkowitoliczbową. To ogromne ograniczenie, jeśli zamierzasz wykonywać jakiegokolwiek rzeczywiste obliczenia matematyczne.



Powłoka z (zsh) obsługuje pełną arytmetykę zmiennoprzecinkową. Jeśli chcesz wykonywać obliczenia zmiennoprzecinkowe w swoich skryptach powłoki, możesz zainteresować się tą powłoką (jej opis znajduje się w rozdziale 23.).

Rozwiązanie zmiennoprzecinkowe

Ograniczenie powłoki Bash do wykonywania tylko działań arytmetyki całkowitoliczbowej można zlikwidować na kilka sposobów. Najpopularniejszy polega na użyciu wbudowanego kalkulatora Bash o nazwie `bc`.

Podstawy kalkulatora `bc`

Kalkulator Bash to język programowania, który umożliwia wpisywanie wyrażeń zmiennoprzecinkowych w wierszu poleceń, aby je następnie zinterpretować, obliczyć i zwrócić wynik. Rozpoznaje następujące elementy:

- liczby (zarówno całkowite, jak i zmiennoprzecinkowe);
- zmienne (zarówno proste, jak i tablice);
- komentarze (wiersze zaczynające się od krzyżyka i komentarze w stylu języka C — `/* */`);
- wyrażenia;
- instrukcje programistyczne (takie jak instrukcje `if-then`);
- funkcje.

Dostęp do kalkulatora Bash można uzyskać przez wpisanie polecenia `bc` w wierszu poleceń:

```
$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
12 * 5.4
64.8
3.156 * (3 + 5)
25.248
quit
$
```

Przykład zaczyna się od wyrażenia $12 * 5.4$. Kalkulator Bash zwraca wynik. Każde kolejne polecenie wpisane do kalkulatora zostaje obliczone i następuje wyświetlenie wyniku. Aby wyjść z kalkulatora Bash, należy wpisać `quit`.

Arytmetyką zmiennoprzecinkową steruje wbudowana zmienna o nazwie `scale`. Jej wartość należy ustawić na liczbę miejsc dziesiętnych, jaką chcemy otrzymać w wyniku. Jeśli tego nie zrobimy, możemy otrzymać inny wynik, niż się spodziewaliśmy:

```
$ bc -q
3.44 / 5
0
scale=4
3.44 / 5
.6880
quit
$
```

Domyślną wartością zmiennej `scale` jest 0, a więc jeśli jej nie zmienimy, kalkulator będzie zwracał wyniki z zerem miejsc dziesiętnych. Po ustawieniu jej wartości na 4 kalkulator zwraca wyniki z dokładnością do czterech miejsc po przecinku. Parametr wiersza poleceń `-q` wyłącza długi tekst powitalny kalkulatora Bash.

Kalkulator Bash oprócz zwykłych liczb rozpoznaje także zmienne:

```
$ bc -q
var1=10
var1 * 4
40
var2 = var1 / 5
print var2
2
quit
$
```

Kiedy zmienna jest zdefiniowana, można jej używać wszędzie w obrębie sesji kalkulatora Bash. Instrukcja `print` umożliwia drukowanie zmiennych i liczb.

Zastosowanie kalkulatora bc w skryptach

Teraz pewnie się zastanawiasz, w jaki sposób kalkulator Bash może Ci się przydać przy wykonywaniu zmiennoprzecinkowych działań arytmetycznych w skryptach powłoki. Pamiętasz swojego dobrego przyjaciela odwrotny apostrof? Tak, za pomocą znaku podstawiania poleceń można wykonywać polecenia `bc`, a ich wyniki można zapisywać w zmiennej! Podstawowy format tej operacji wygląda tak:

```
zmienna=$(echo "opcje; wyrażenie" | bc)
```

Pierwsza część, *opcje*, umożliwia ustawienie wartości zmiennych. Jeśli ustawiasz wartości więcej niż jednej zmiennej, rozdziel je średnikami. Parametr *wyrażenie* reprezentuje wyrażenie matematyczne, które ma zostać obliczone przez `bc`. Oto krótki przykład zastosowania tej techniki w skrypcie:

```
$ cat test9
#!/bin/bash
var1=$(echo " scale=4; 3.44 / 5" | bc)
echo Odpowiedź to $var1
$
```


W tym przykładzie ustawiliśmy zmienną `scale` na cztery miejsca dziesiętne, a następnie podaliśmy konkretne wyrażenie do obliczenia. Ten skrypt zwróci następujący wynik:

```
$ chmod u+x test9
$ ./test9
Odpowiedź to .6880
$
```

Świetnie! W wyrażeniach możemy używać nie tylko liczb, ale również zmiennych zdefiniowanych w skrypcie powłoki:

```
$ cat test10
#!/bin/bash
var1=100
var2=45
var3=$(echo "scale=4; $var1 / $var2" | bc)
echo Odpowiedź w tym przypadku to $var3
$
```

Ten skrypt definiuje dwie zmienne, które zostały użyte w wyrażeniu wysłanym do polecenia `bc`. Pamiętaj, aby za pomocą znaku dolara zaznaczać, że masz na myśli wartości zmiennych, a nie same zmienne. Wynik tego skryptu jest następujący:

```
$ ./test10
Odpowiedź w tym przypadku to 2.2222
$
```

Oczywiście kiedy w zmiennej zostanie zapisana wartość, można jej użyć w kolejnym działaniu:

```
$ cat test11
#!/bin/bash
var1=20
var2=3.14159
var3=$(echo "scale=4; $var1 * $var1" | bc)
var4=$(echo "scale=4; $var3 * $var2" | bc)
echo Ostateczny wynik to $var4
$
```

Ta metoda dobrze się sprawdza w przypadku wykonywania krótkich obliczeń, ale czasami działania są bardziej skomplikowane. Jeśli chcemy wykonać więcej niż tylko parę operacji, to lista wielu wyrażeń w jednym wierszu może być mało przejrzysta.

Ten problem można rozwiązać. Polecenie `bc` rozpoznaje przekierowywanie wejścia, dzięki czemu można do niego skierować zawartość pliku do przetworzenia. Jednak to też nie jest idealne rozwiązanie, ponieważ wszystkie wyrażenia trzeba zapisać w pliku.

Najlepszą metodą jest użycie śródliniowego przekierowania wejścia, które umożliwia przekierowanie danych bezpośrednio z wiersza poleceń. W skrypcie powłoki wynik przypisujemy do zmiennej:

```
zmienna=$(bc << EOF
opcje
instrukcje
wyrażenia
EOF
)
```

Łańcuch `EOF` oznacza początek i koniec śródliniowego przekierowania danych. Pamiętaj, że do przypisania wyniku polecenia `bc` do zmiennej potrzebne są znaki podmieniania polecenia.

Teraz wszystkie poszczególne elementy kalkulatora Bash można umieścić w osobnych wierszach w pliku skryptu. Oto przykład zastosowania tej techniki w skrypcie:

```
$ cat test12
#!/bin/bash

var1=10.46
var2=43.67
var3=33.2
var4=71

var5=$(bc << EOF
scale = 4
a1 = ( $var1 * $var2)
b1 = ($var3 * $var4)
a1 + b1
EOF
)

echo Ostateczna odpowiedź w tym całym zamieszaniu to $var5
$
```

Umieszczenie każdej opcji i każdego wyrażenia w osobnym wierszu w skrypcie sprawia, że tekst jest znacznie bardziej czytelny. Łańcuch EOF oznacza początek i koniec danych, które mają zostać przekierowane do polecenia bc. Oczywiście należy użyć znaków podmieniania poleceń, aby wskazać polecenie, które ma zostać przypisane do zmiennej.

Ponadto w tym przykładzie zauważ, że w kalkulatorze Bash można przypisywać zmienne. Należy pamiętać, że wszystkie zmienne utworzone w tym kalkulatorze są dostępne tylko w nim i nie można ich używać w skrypcie powłoki.

Wychodzenie ze skryptu

W naszych dotychczasowych przykładowych skryptach zakończenie pracy następowało bardzo raptownie. Po zakończeniu wykonywania ostatniego polecenia po prostu kończyliśmy skrypt. Można to jednak zrobić znacznie bardziej elegancko.

Każde polecenie, które jest wykonywane w powłoce, po zakończeniu pracy zwraca do powłoki **kod stanu zakończenia**. Jest to liczba z przedziału od 0 do 255, którą polecenie przekazuje powłoce, gdy skończy działanie. Wartość tę można przechwycić i wykorzystać w swoich skryptach.

Sprawdzanie stanu wyjścia

W systemie Linux można używać specjalnej zmiennej \$?, która przechowuje stan wyjścia ostatnio wykonanego polecenia. Należy jej użyć bezpośrednio po poleceniu, które nas interesuje, ponieważ jej wartość zmienia się za każdym razem, gdy w powłoce zostaje wykonane jakieś polecenie:

```
$ date
pon 01 cze 16:01:30 EDT 2020
$ echo $?
0
$
```

Standardowo kod stanu wyjścia polecenia, które zostało pomyślnie wykonane, to 0. Jeśli wykonywanie polecenia zakończy się błędem, to jego stan wyjścia jest oznaczony całkowitą liczbą dodatnią:

```
$ asdfg
-bash: asdfg: command not found
$ echo $?
127
$
```

Nieprawidłowe polecenie zwraca kod stanu wyjścia 127. Kody stanu wyjścia z błędem w Linuksie nie są ustandaryzowane, ale można się kierować paroma wytycznymi, które przedstawiono w tabeli 11.2.

TABELA 11.2. Kody stanu wyjścia Linuksa

Kod	Opis
0	Polecenie zostało wykonane pomyślnie
1	Ogólny nieznaný błąd
2	Nieprawidłowe użycie polecenia powłoki
126	Nie można wykonać polecenia
127	Nie znaleziono polecenia
128	Nieprawidłowy argument wyjścia
128+x	Błąd krytyczny z sygnałem Linuksa x
130	Polecenie zakończono przez naciśnięcie klawiszy <i>Ctrl+C</i>
255	Stan wyjścia poza zakresem

Kod stanu wyjścia 126 oznacza, że użytkownik nie miał odpowiednich uprawnień do wykonania polecenia:

```
$ ./myprog.c
-bash: ./myprog.c: Permission denied
$ echo $?
126
$
```

Innym często spotykanym błędem jest przekazanie do polecenia nieprawidłowego parametru:

```
$ date %t
date: invalid date '%t'
$ echo $?
1
$
```

Powoduje to wygenerowanie ogólnego stanu kodu wyjścia 1, który oznacza, że w poleceniu wystąpił nieznaný błąd.

Polecenie exit

Domyślnie skrypt powłoki zwraca kod wyjścia odpowiadający stanowi ostatniego wykonanego w nim polecenia:

```
$ ./test6
Wynik wynosi 2
$ echo $?
0
$
```

Użytkownik może to zmienić, aby zwracać własny kod stanu wyjścia. Służy do tego polecenie `exit`, umożliwiające określenie stanu wyjścia po zakończeniu wykonywania skryptu:

```
$ cat test13
#!/bin/bash
# test stanu wyjścia
var1=10
var2=30
var3=$(( $var1 + var2 ))
echo Odpowiedź to $var3
exit 5
$
```

Gdy sprawdzisz stan wyjścia skryptu, otrzymasz wartość podaną jako parametr polecenia `exit`:

```
$ chmod u+x test13
$ ./test13
Odpowiedź to 40
$ echo $?
5
$
```

W poleceniu `exit` można także używać zmiennych:

```
$ cat test14
#!/bin/bash
# test stanu wyjścia
var1=10
var2=30
var3=$(( $var1 + var2 ))
exit $var3
$
```

Kiedy wykonasz to polecenie, wygeneruje ono następujący kod stanu wyjścia:

```
$ chmod u+x test14
$ ./test14
$ echo $?
40
$
```

Z tej możliwości należy jednak korzystać ostrożnie, ponieważ maksymalna wartość kodu stanu wyjścia wynosi 255. Spójrz, co się dzieje w tym przykładzie:

```
$ cat test14b
#!/bin/bash
# test stanu wyjścia
var1=10
var2=30
var3=$(( $var1 * var2 ))
echo Wartość wynosi $var3
exit $var3
$
```

Wynik wykonania tego skryptu będzie następujący:

```
$ ./test14b
Wartość wynosi 300
$ echo $?
44
$
```

Kod stanu wyjścia został zredukowany do wartości mieszczącej się w przedziale od 0 do 255. W tym celu powłoka stosuje arytmetykę modulo, czyli pobiera resztę z dzielenia. Otrzymana w wyniku liczba jest resztą z dzielenia podanej liczby przez 246. W tym przypadku podzielono 300 (wynik) przez 246, co dało resztę 44 i ta wartość została zwrócona jako kod stanu wyjścia.

W następnym rozdziale pokazujemy, jak za pomocą instrukcji i `f-then` sprawdzać stan błędu zwrócony przez polecenie, aby się dowiedzieć, czy jego wykonanie się powiodło.

Praktyczny przykład

Znasz już podstawy tworzenia skryptów powłoki, więc możemy spróbować napisać jakiś skrypt o praktycznym zastosowaniu. Utworzymy program, który będzie obliczał liczbę dni dzielącą dwie daty. Użytkownik będzie mógł podać datę w dowolnym formacie rozpoznawanym przez linuksowe polecenie `date`. Najpierw zapiszemy obie daty w zmiennych:

```
$date1="01/01/2020"
$date2="01/01/2021"
```

Arytmetyka na datach jest trudna, ponieważ wymaga wiedzy, które miesiące mają 28, 30 i 31 dni i które lata to lata przestępne. Na szczęście samo polecenie `date` może nam trochę pomóc.

Polecenie `date` pozwala na określenie konkretnej daty za pomocą opcji `-d` (w dowolnym formacie), a następnie wyświetlenie jej w dowolnym innym zdefiniowanym przez nas formacie. Aby wykonać nasze obliczenia, skorzystamy z funkcji Linuksa o nazwie **czas epoki**.

Jest to liczba całkowita oznaczająca liczbę sekund, jaka upłynęła od północy 1 stycznia 1970 r. (stary uniksowy standard). Aby sprawdzić czas epoki dla 1 stycznia 2020 r., należy wykonać następujące polecenie:

```
$ date -d "01/01/2020" +%s
1577854800
$
```

Tą metodą sprawdzimy czas epoki dla obu dat, a następnie odejmiemy od siebie te dwie wartości, aby otrzymać liczbę dzielących je sekund. Potem wystarczy podzielić otrzymaną liczbę przez liczbę sekund w dniu (60 sekund w minucie, 60 minut w godzinie i 24 godziny w dobie), aby otrzymać różnicę między dwiema interesującymi nas datami w dniach.

Za pomocą techniki podmieniania poleceń prześlemy wynik polecenia `date` do zmiennej:

```
$time1=$(date -d "$date1" +%s)
```

Po otrzymaniu czasu epoki dla obu dat wystarczy użyć niedawno poznanego polecenia `expr` do obliczenia różnicy (moglibyśmy też użyć narzędzia `bc`, ale do pracy z liczbami całkowitymi wystarczy `expr`).

Po połączeniu wszystkich kawałków układanki otrzymujemy następujący skrypt:

```
$ cat mydate.sh
#!/bin/bash
# obliczanie liczby dni dzielących dwie daty
date1="01/01/2020"
date2="01/01/2021"

time1=$(date -d "$date1" +%s)
time2=$(date -d "$date2" +%s)
```

```
diff=$(expr $time2 - $time1)
secondsinday=$(expr 24 \* 60 \* 60)
days=$(expr $diff / $secondsinday)

echo "Różnica między $date2 i $date1 w dniach wynosi: $days"
$
```

Jeszcze tylko przypiszemy odpowiednie uprawnienia do pliku i możemy uruchomić skrypt:

```
$ chmod u+x mydate.sh
$ ./mydate.sh
Różnica między 01/01/2021 i 01/01/2020 w dniach wynosi: 366
$
```

Teraz w zmiennych możesz zapisać dowolne inne daty (w niemal każdym formacie, jaki Cię interesuje), aby otrzymać prawidłowy wynik.

Podsumowanie

W skryptach powłoki Bash można używać łańcuchów poleceń. Najprostszym sposobem na utworzenie skryptu jest wpisanie kilku poleceń w jednym wierszu i rozdzielenie ich średnikami. Powłoka wykona je po kolei i wyświetli wynik każdego z nich na ekranie monitora.

Skrypty powłoki składające się z wielu poleceń można także zapisywać w plikach. Plik skryptu powłoki musi określać powłokę, w której ma być wykonywany. Określenie to należy umieścić w pierwszym wierszu, zaczynającym się od znaków #!, po których następuje pełna ścieżka do powłoki.

W skrypcie powłoki można odwoływać się do wartości zmiennych środowiskowych za pomocą ich nazw poprzedzonych znakiem dolara. Ponadto można definiować własne zmienne do użycia w skrypcie, a także można przypisywać im wartości, a nawet wyniki poleceń, za pomocą znaku odwrotnego apostrofu lub formatu \$(). Wartości zmiennej można użyć w skrypcie przez postawienie znaku dolara przed jej nazwą.

Powłoka Bash umożliwia przekierowywanie zarówno wejścia, jak i wyjścia polecenia. Użytkownik może przekierować wynik dowolnego polecenia z monitora do pliku za pomocą symbolu większości, po którym powinna się znajdować nazwa pliku przeznaczonego do przechowywania danych. Dane wynikowe można dodać do istniejącego pliku przez użycie dwóch znaków większości. Znak mniejszości z kolei służy do przekierowywania wejścia poleceń. Użytkownik może przekierować dane z pliku do polecenia.

Polecenie potoku Linuksa (symbol dwóch pionowych kresek) umożliwia przekierowanie wyjścia polecenia bezpośrednio na wejście innego polecenia. System Linux wykonuje oba polecenia jednocześnie, wysyłając wyniki pierwszego na wejście drugiego bez użycia jakichkolwiek plików pośrednich.

Powłoka Bash umożliwia wykonywanie działań matematycznych w skryptach na dwa sposoby. Polecenie expr pozwala na proste wykonywanie działań całkowitoliczbowych. Ponadto powłoka Bash umożliwia wykonywanie podstawowych obliczeń matematycznych przez umieszczenie równań w nawiasach prostokątnych poprzedzonych znakiem dolara. Aby wykonać obliczenia arytmetyki zmiennoprzecinkowej, należy użyć polecenia bc, przekierowując wejście z danych śródliniowych i zapisując wyniki w zmiennej użytkownika.

W ostatniej części rozdziału omówiliśmy kody stanu wyjścia skryptów powłoki. Każde polecenie działające w powłoce zwraca kod stanu wyjścia. Jest to liczba całkowita z przedziału od 0 do 255, która określa, czy dane polecenie zostało wykonane poprawnie, a jeśli nie, to co może być tego przyczyną. Kod stanu wyjścia 0 oznacza pomyślne wykonanie polecenia. Za pomocą polecenia `exit` można zadeklarować kod stanu wyjścia po zakończeniu wykonywania skryptu.

W skryptach, które dotychczas pokazaliśmy, polecenia były wykonywane jedno po drugim w kolejności wpisania. W następnym rozdziale poznasz techniki modyfikacji logiki wykonywania poleceń skryptu.

Skorowidz

A

ACL, access control list, 181
aliasy poleceń, 139
anacron, 420
archiwa, *Patrz* kopie zapasowe
archiwizacja danych, 120

B

biblioteka, 447
 shtool, 451, 453
blob, 649
BRE, Basic Regular Expression, 685

C

cel, target, 33
CLI, command-line interface, 45
cron, 419
 budowa tabeli, 420
 przeglądanie katalogów, 420
 tabela, 419

D

dane
 archiwizacja, 120
 kompresja, 119
 przechowywanie, 114
 wyszukiwanie, 117
deskryptor plików
 STDERR, 381
 STDIN, 379
 STDOUT, 380
deskryptory plików
 do odczytu/zapisu, 387
 przekierowywanie, 386

tworzenie, 385
wejściowych, 387
wyświetlanie listy, 389
zamykanie, 388

dostęp

do CLI, 45
 przez graficzny emulator terminala, 50
 przez konsolę, 47
do powłoki, 45
do terminala
 GNOME, 50
 Konsole, 58
 xterm, 65

dowiązanie, link, 87
miękkie, 123
symboliczne, 87
twarde, 87

drukowanie, 581

dystrybucje, 42
 specjalistyczne, 43

dziennik

zapisywanie komunikatów, 395

E

edytor Emacs, 232
 bufory, 237
 edytowanie danych, 235
 edytowanie pliku, 234
 graficzny interfejs użytkownika, 238
 kopiowanie i wklejanie, 236
 okna w trybie konsolowym, 238
 okno graficzne, 239
 tryb konsolowy, 234
 wyszukiwanie i zastępowanie, 236
 wyszukiwarka plików, 237
 zabijanie tekstu, 235

- edytor gedit, 249
 - funkcje, 250
 - menedżer plików, 253
 - menu, 251
 - okienko boczne, 252
 - okno główne, 250
 - wtyczki, 253, 255
- edytor Kate, 245
 - okno
 - dialogowe konfiguracji, 249
 - główne, 246
 - terminala, 248
 - Zarządzanie wtyczkami, 247
- edytor KWrite, 240
 - menu Edit, 242
 - menu Narzędzia, 243
 - okno domyślne, 240
 - skrypty, 244
 - ustawienia konfiguracyjne, 245
 - wyszukiwanie tekstu, 242
- edytor nano, 230
 - okno główne, 231
 - polecenia sterujące, 231
- edytor sed, 483
 - adresowanie wierszy, 494, 678
 - drukowanie ostatnich wierszy, 553
 - flagi zastępowania, 492
 - grupowanie poleceń, 496
 - łączenie wierszy tekstu, 536
 - negowanie poleceń (!), 541
 - numerowanie wierszy w pliku, 552
 - polecenia
 - schowka, 540
 - w skryptach, 549
 - w wierszu poleceń, 484, 485
 - wielowierszowe, 534
 - z pliku, 486
 - polecenie
 - dołączania (a), 498, 679
 - drukowania (P), 539
 - drukowania (p), 502, 680
 - drukowania numerów wierszy (=), 503
 - listy (l), 504
 - przejęcia do następnego wiersza (n), 535
 - przejęcia do następnego wiersza (N), 536, 538
 - odczytu (r), 506, 681
 - rozgałęzienia (b), 544
 - testowe (t), 546
 - transformacji (y), 680
 - usuwania (d), 496, 679
 - usuwania (D), 538
 - wstawiania (i), 498, 679
 - zamieniania znaków (y), 501
 - zapisywania (w), 505, 680
 - zastępowania (s), 492, 502, 678
 - zmiany (c), 500, 680
 - przekierowywanie wyników, 550
 - schowek tymczasowy, 540
 - uruchamianie, 677
 - usuwanie pustych wierszy, 555
 - usuwanie znaczników HTML-a, 557
 - wstawianie pustych wierszy, 551
 - wzorce tekstowe, 495
 - zamiana pojedynczych słów, 548
 - zamiana znaków, 494
- edytor vim, 224
 - edytowanie danych, 228
 - kopiowanie i wklejanie, 229
 - okno główne, 226
 - tryb
 - Ex, 227
 - poleceń, 227
 - wizualny, 229
 - wstawiania, 227
 - wyszukiwanie i zastępowanie, 229
- emulator terminala, 46
 - Gnome, 50
 - Konsole, 58
 - xterm, 64
- emulatory graficzne terminala, 50

F

- filtrowanie listy wyników, 82
- format CSV, 396
- funkcja inversescreen, 49
- funkcje
 - audytu powłoki, 636
 - biblioteki shtool, 453
 - czasu, 586
 - definiowanie, 587
 - dostępność zmiennych, 441
 - łańcuchowe, 585
 - matematyczne, 583, 605
 - monitorowania uprawnień, 640
 - polecenie return, 437
 - programu gawk, 685
 - przekazywanie parametrów, 439
 - przekazywanie tablic, 443
 - rekurencyjne, 445
 - skryptowe, 433
 - tworzenie, 434

- tworzenie biblioteki, 588
- używanie, 434, 588
- w pliku .bashrc, 449
- w wierszu poleceń, 448
- wyjście, 438
- zwracanie tablic, 445
- zwracanie wartości, 436

G

- Git, 648
 - katalog roboczy, 649
 - klonowanie, 650
 - konfiguracja środowiska, 651
 - poczekalnia, 649
 - repozytorium lokalne, 649
 - repozytorium zdalne, 649, 658
 - rozgałęzianie, 650
 - zatwierdzanie skryptów, 654
- globbing, 328
- Gnome, 38, 50
 - dostęp do terminala, 50
 - menu
 - Edit, 56
 - File, 55
 - Search, 57
 - Tabs, 57
 - Terminal, 57
 - View, 56
 - okno Keyboard Shortcuts, 54
 - pasek menu, 55
 - w CentOS, 51
 - w Ubuntu, 52
- graficzny interfejs użytkownika, GUI, 238
- grupa woluminów, VG, 198
 - powiększanie i zmniejszanie, 201
 - tworzenie, 199
- grupy, 171
 - modyfikowanie, 173
 - tworzenie, 173

I

- identyfikator
 - grupy, GID, 172
 - ustawionego użytkownika, SUID, 180
 - ustawionej grupy, SGID, 180
 - użytkownika, UID, 163
- IFS, internal field separator, 327
- instalacja oprogramowania, 203, 206, 213, 219

- instrukcja
 - elif, 291, 292
 - if-then, 286, 310
 - zaawansowane właściwości, 311
 - zagnieżdżanie, 290
 - if-then-else, 289, 577
 - SQL INSERT, 397
- interfejs
 - graficzny, 472
 - elementy, 47
 - wiersza poleceń, CLI, 45

J

- jądro, 30
- język gawk, 565
 - definiowanie funkcji, 587
 - drukowanie, 581
 - funkcje
 - czasu, 586
 - łańcuchowe, 585
 - matematyczne, 583
 - instrukcja
 - do-while, 580
 - for, 580
 - if, 577
 - while, 578
 - iteracja, 573
 - tablice, 572
 - tworzenie biblioteki funkcji, 588
 - wyrażenia matematyczne, 576
 - wyrażenia regularne, 575
 - zmiennie
 - tablicowe, 573
 - użytkownika, 571
 - wbudowane, 566–569
- język Markdown, 657

K

- kalkulator bc, 277
- katalog, 75
 - główny, 73
 - roboczy, 77
 - wirtualny, 73
 - tymczasowy, 394
- katalogi
 - tworzenie, 90, 394
 - usuwanie, 91
- KDE Plasma, 37

- klasy znaków, 519
 - specjalne, 522
- kod źródłowy, 219
- komendy
 - polecenia fdisk, 191
 - polecenia gdisk, 193
- kompresja danych, 119, 612
- konsole, 46, 47, 58
 - dostęp do terminala, 58
- menu
 - Edycja (Edit), 60
 - Plik (File), 60
 - Pomoc (Help), 63
 - Ustawienia (Settings), 62
 - Widok (View), 61
 - Zakładki (Bookmarks), 62
- pasek menu, 59
- w Ubuntu, 59
- wirtualne, 46–49
- konta
 - systemowe, 164
 - użytkowników, 163
 - pobieranie nazwy, 622, 623
 - sprawdzanie, 626
 - usuwanie, 622
 - usuwanie procesów, 627
 - weryfikacja nazwy, 624
 - wyszukiwanie plików konta, 629
- kontenery aplikacji, 216
 - flatpak, 218
 - snap, 216
- kontrola wersji, 647, *Patrz także* Git
- kopie zapasowe
 - codzienne, 612
 - godzinne, 618
 - hierarchia katalogów, 619

L

- łepki bit, 180
- liczba uprzejmości, nice value, 413
- liczby
 - porównywanie, 295
- lista kontroli dostępu, ACL, 181
- login, 163
- logowanie do powłoki, 154
- LVM, Logical Volume Manager, 197
 - struktura, 197
 - używanie, 200

Ł

- łańcuchy
 - porównywanie, 296

M

- menedżer dysków logicznych, LVM, 197
- menu tekstowe, 456
 - funkcje, 458
 - logika, 459
 - struktura, 456
 - tworzenie okien, 462
 - widzety, 462
- metody inicjalizacyjne, 32
- migawka, snapshot, 188, 649
- moduły powłoki zsh, 603
- monitorowanie
 - procesów, 105
 - przestrzeni dyskowej, 109
 - systemu, 636
 - uprawnień, 640
 - mediów, 109

N

- narzędzia
 - do kompresji plików, 119
 - do modyfikacji kont użytkowników, 169
 - GNU, 35
- narzędzie, *Patrz także* polecenie
 - anacron, 420, 618
 - cron, 419
- nawias
 - klamrowy, 526
 - kwadratowy, 276
 - podwójny, 312
 - pojedynczy, 311
 - prostokątny podwójny, 313
- nazwy katalogów, 75
- nośnik wymienny, 109

O

- odwołania
 - bezwzględne do katalogów, 76
 - względne do katalogów, 78
- opakowanie, wrapper, 549
- opcje, 359
 - oddzielanie od parametrów, 360
 - przetwarzanie, 362

- standaryzacja, 368
- wiersza poleceń, 369
- wyszukiwanie, 359
- operator dopasowywania, 575
- operatory
 - matematyczne, 571
 - polecenia expr, 275
- oprogramowanie, 31, *Patrz* pakiety oprogramowania

P

- pager, 70
- pakiet
 - dialóg, 462
 - Emacs, 232
 - gawk, 486
 - gedit, 249
 - git, 651
 - GNU coreutils, 35
 - kdialóg, 472
 - sysstat, 219
 - vim, 225
 - zenity, 475
- pakiety oprogramowania, 203, 219
 - aktualizacja, 208, 214
 - deinstalacja, 209, 215
 - instalowanie, 206, 213
 - uszkodzone zależności, 215
 - wyświetlanie listy, 212
 - zależności, 203
- pamięć systemowa, 31
- parametry
 - liczenie, 354
 - łańcuchowe, 352
 - pozycyjne, 350
 - przekazywanie, 350
 - przesuwanie, 358
 - testowanie, 354
 - wczytywanie, 350
 - wiersza poleceń, 350
- partycja
 - tworzenie, 190
- pętla
 - do-while, 580
 - for, 322–332, 580
 - for z języka C, 330
 - until, 334
 - while, 332, 578
- pętle
 - polecenie break, 339
 - polecenie continue, 342
 - przeglądanie plików, 338
 - przetwarzanie wyników, 344
 - tworzenie wielu kont, 347
 - wyszukiwanie plików wykonywalnych, 346
 - zagnieżdżanie, 336, 338
- plik
 - .bashrc, 449
 - /etc/group, 172
 - /etc/passwd, 164
 - /etc/profile, 155
 - /etc/shadow, 165
 - biblioteki, 447
 - dziennika, 395
 - null, 391
 - skryptu, 262
 - sources.list, 211
- pliki
 - .tgz, 612
 - danych, 114
 - dowiązania, 86
 - ISO, 42
 - jednostek, unit file, 32
 - kody uprawnień, 176
 - kompresja, 119
 - kopiowanie, 84
 - porównywanie, 300
 - rozruchowe, 154, 158
 - sprawdzanie typu, 92
 - standardowe deskryptory, 378
 - środowiskowe, 154, 158
 - TAR, 219
 - tworzenie, 83, 392, 393
 - tymczasowe, 392
 - ukryte, 79
 - uprawnienia, 174
 - uprawnienia domyślne, 175
 - urządzeń, 33
 - usuwanie, 89
 - wyświetlanie, 93, 95
 - zmienianie nazw, 88
- podpowłoki, 127, 269
 - listy procesów w tle, 132
 - nieinteraktywne, 159
 - tryb tła, 130
 - współprzetwarzanie, 133
- podręcznik Bash, 69
- polecenia
 - edytora sed, 678
 - funkcja uzupełniania, 86
 - opcje, 359
 - parametry, 350

polecenia

- podmienianie, 268
- powłoki
 - Bash, 665–670
 - Dash, 595
 - zsh, 600, 602
- strukturalne, 286
- wbudowane, built-in commands, 134, 135
- wielowierszowe, 534
- zewnętrzne, external commands, 134

polecenie

- alias, 139
- anacron, 420, 618
- apt, 205–209
- at, 415, 416
- atq, 418
- atrm, 418
- bc, 278
- bg, 412
- break, 339, 361
- case, 314, 459
- cat, 93, 613
- cd, 76
- chage, 170
 - parametry, 171
- chfn, 170
- chgrp, 178, 179
- chmod, 177
- chown, 178
- chpasswd, 169
- chsh, 170
- clear, 457
- configure, 451
- continue, 342
- coproc, 133
- cp, 84
- cut, 625
- date, 270, 283, 618, 620
- df, 112
- dialog, 462
 - opcje, 468, 469, 470
 - tworzenie formularzy, 479
 - w skryptach, 470
- diff, 641
- dnf, 213, 214
- done, 344
- dpkg, 204
- du, 113
- echo, 264
- exec, 383–385, 388, 613
- exit, 128, 281
- export, 147
- expr, 274, 276
 - operatorzy, 275
- fdisk, 190
 - komendy, 191
- file, 92
- find, 641
- flatpak, 218
- for, 322–332
- fsck, 196
 - parametry, 196
- function, 598, 606
- gawk
 - format, 681
- gdisk, 192
 - komendy, 193
- getfacl, 181
- getopt, 363
- getopts, 366, 374, 423, 642
- git add, 655
- git commit, 656
- git config, 653
- git init, 653
- git pull, 659
- git remote, 658
- git status, 655
- GNU parted, 194
- grep, 117, 626
- groupadd, 173
- groupmod, 173
- head, 97
- help help, 73
- history, 136, 137, 139
- jobs, 410
 - parametry, 412
- kdialog, 472, 474
 - opcje, 472, 473
- kill, 108, 402, 412, 628
- kwite, 241
- less, 95
- ls, 79, 81, 174, 274
- lsuf, 390
 - domyślne wyniki, 390
- lvcreate, 200
- make, 221, 451
- man, 70
- meminfo, 471
- menu, 482
- mkdir, 90, 620
- mktemp, 392–394
- more, 94, 273, 274

- mount, 109
 - parametry, 110
- mv, 88
- nice, 414
- nohup, 409
- passwd, 169
- pkill, 108
- printenv, 143
- printf, 581
- ps, 99, 134, 406, 409
 - parametry BSD, 103
 - parametry GNU, 104
 - parametry uniksowe, 100
- pvcreate, 199
- read, 369–376
- renice, 414
- return, 437
- rm, 90
- rmdir, 91
- rpm, 272, 273
- search, 206
- sed
 - opcje, 484, 677
- select, 460
- set, 145
- setfacl, 182
- setterm, 49
- shift, 358, 361
- sleep, 131
- snap, 217
- sort, 114, 298
 - parametry, 116
- sudo, 615, 628
- tail, 96
- tar, 132, 612
 - parametry, 120
- tee, 395
- test, 293, 299, 597
- top, 105
- touch, 82, 83
- trap, 402, 403
- type, 134
- umask, 176
- umount, 111
- unset, 148, 161, 623
- until, 334
- upgrade, 208
- useradd, 165
 - parametry, 167
- userdel, 168
- usermod, 169
- vgcreate, 199
- vi, 225
- vim, 226
- wc, 271, 637
- which, 134, 232, 233
- while, 332
- zenity, 475
 - w skryptach, 477
- porównywanie
 - liczb, 295
 - łańcuchów, 296
 - plików, 300
- POSIX ERE, 514
- potoki, 271
- powłoka Bash, 68
 - polecenia wbudowane, 665–667
 - polecenia zewnętrzne, 668–670
 - zmienné środowiskowe, 671–675
- powłoka Dash, 591
 - parametry pozycyjne, 594
 - parametry wiersza poleceń, 593
 - polecenia wbudowane, 595
 - tworzenie skryptów, 596
 - zmienné środowiskowe, 593
- powłoka zsh, 599
 - działania matematyczne, 604
 - moduły, 603
 - parametry wiersza poleceń, 600
 - polecenia strukturalne, 605
 - polecenia wbudowane, 600, 602
 - tworzenie funkcji, 606
- powłoki, 36
 - domyślne systemu, sh, 124
 - interaktywne, 159
 - logowania, 124, 125, 154
 - nadrzędne, 125
 - potomne, 126
 - relacja rodzic-dziecko, 125
- priorytet planowania, scheduling priority, 413
- proces, 126
 - inicjalizacyjny, 32
- procesy
 - identyfikatory PID, 100
 - monitorowanie w czasie rzeczywistym, 105
 - podglądanie, 99
 - sygnały, 108
 - wyświetlanie listy, 129
 - zatrzymywanie, 107
- program gawk, 486, *Patrz także* język gawk
 - łączenie poleceń, 489
 - opcje, 487, 681

- program gawk
 - operator dopasowywania, 685
 - polecenia strukturalne, 685
 - przypisywanie zmiennych, 684
 - sekcja BEGIN, 490
 - sekcja END, 492
 - wczytywanie z pliku, 489, 682
 - wczytywanie z wiersza poleceń, 682
 - wykonywanie skryptów, 682
 - wyrażenia matematyczne, 685
 - wyrażenia regularne, 685
 - zmienne pól danych, 488
 - zmienne wbudowane, 683, 684
- przekierowania własne, 385
- przekierowywanie
 - błędów, 381
 - deskryptorów plików, 386
 - wejścia, 270, 384
 - wyjścia, 270, 382
 - wyników edytora sed, 550
- przestrzeń
 - wymiany, 31
 - wzorca, pattern space, 536, 540
- puła pamięci, storage pool, 188
- pulpit, 39, 41
 - GNOME, 38
 - KDE Plasma, 37
- punkt montowania, mount point, 74

R

- repozytoria, 203
 - apt, 210
 - RPM, 215
- rozgałęzianie, branch, 650
 - polecenia zewnętrznego, 135

S

- separator pól, 327
- skrypty, 36
 - archiwizacja cogodzinna, 618, 620
 - monitorowanie systemu, 636
 - monitorowanie uprawnień, 642, 644
 - odbieranie danych, 369
 - polecenia edytora sed, 549
 - powłoki Dash, 596
 - powłoki zsh, 604
 - programu gawk, 487
 - przechwytywanie sygnału wyjścia, 403

- przekazywanie parametrów, 350
- przekierowywanie wejścia, 384
- przekierowywanie wyjścia, 382
- sprawdzanie nazwy, 352
- tworzenie, 262
- tworzenie funkcji, 434
- uruchamianie z nową powłoką, 422
- usuwanie kont, 622, 630, 634
- wykonywane bez rozłączania, 409
- wykonywane w tle, 406
- wykonywanie kopii zapasowych, 611, 615
- słowo kluczowe
 - BEGIN, 490, 683
 - END, 683
 - function, 434, 587
- sourcing, 424
- sprzęt, 33
- stan wyjścia, 280
 - kody, 281
- STDERR, 380
- STDIN, 379
- STDOUT, 380
- struktura plików, 75
- sygnały, 107, 400
 - generowanie, 400
 - przechwytywanie, 402, 403
 - usuwanie pułapki, 404
 - wyjścia, 403
- symbol wieloznaczny (.), 537
- symbole wieloznaczne, 83, 328
- system
 - kontroli wersji, 648, *Patrz także* Git
 - plików, 34, 73, 185
 - Btrfs, 189
 - ext, 186
 - ext2, 186
 - ext3, 187
 - ext4, 187
 - JFS, 187
 - ReiserFS, 188
 - Stratis, 189
 - XFS, 188
 - ZFS, 189
 - zarządzania pakietami, 203
- systemd, 32
- systemy plików, 34, 73, 185
 - metody księgowania, 187
 - naprawianie, 196
 - sprawdzanie, 196
 - tworzenie, 194

wirtualne, VFS, 35
 z księgowaniem, 186
 z zarządzaniem woluminami, 188
 SysVinit, 32

Ś

środowisko pulpitu, 36

T

tabela crona, 419
 tablice, 161, 443
 asocjacyjne, 572
 terminal graficzny, 46
 testy złożone, 310

U

uprawnienia
 do plików, 174
 zmienianie, 177
 użytkownik
 dodawanie, 165
 modyfikowanie konta, 169
 usuwanie, 168

V

VCS, version control system, 648

W

wejście i wyjście, 269, 378
 węzeł, node, 34
 widżet
 fselect, 467
 inputbox, 465
 menu, 467
 msgbox, 463
 textbox, 466
 yesno, 464
 widżety
 dialog, 462
 kdialog, 472, 474
 zenity, 475
 wiersz poleceń
 emulatora xterm, 65
 opcje, 369
 parametry pozycyjne, 350

 powłoki Bash, 69, 128
 uzyskiwanie dostępu, 45, 47, 50
 używanie funkcji, 448
 wirtualny system plików, VFS, 35
 wolumin, 188
 fizyczny, PV, 197
 tworzenie, 199
 logiczny, LV, 197, 198
 formatowanie i montowanie, 200
 powiększanie i zmniejszanie, 201
 tworzenie, 200
 współprzetwarzanie, co-processing, 133
 wyrażenia regularne, 512, 575, 685
 definiowanie tekstu, 514
 grupowanie wyrażeń, 527
 gwiazdka, 523
 klamry, 526
 klasy znaków, 519
 liczenie plików, 528
 negacja klas znaków, 521
 POSIX ERE, 514
 rozszerzone, 524
 specjalne klasy znaków, 522
 sprawdzanie adresów e-mail, 531
 sprawdzanie numeru telefonu, 529
 symbol potoku, 527
 typy, 513
 zakresy, 521
 zamiana pojedynczych słów, 548
 znak &, 547
 znak plusa, 525
 znak zapytania, 524
 znaki specjalne, 515
 znaki zakotwiczenia, 516
 wyrażenie matematyczne, 576, 685
 wyszukiwanie danych, 117
 wyświetlanie
 komunikatów, 264
 listy procesów, 129
 plików i katalogów, 79
 zadań, 410

X

X Window, 37
 xterm, 64
 dostęp do terminala, 65
 parametry wiersza poleceń, 65

Z

- zabezpieczenia, 163
- zadanie, task, 106
 - planowanie wykonywania, 415
 - ponowne uruchamianie, 412
 - usuwanie, 418
 - wykonywane w tle, 408
 - wyświetlanie, 410
 - wyświetlanie zadań oczekujących, 418
- zarządzanie
 - oprogramowaniem, 31
 - pamięcią systemową, 31
 - sprzętem, 33
 - systemem plików, 34
- zmiana
 - uprawnień, 177
 - właściciela, 178
- zmienna, 265
 - \$#, 355, 356, 439
 - \$*, 356
 - \$?, 280, 436
 - \$@, 356
 - \$0, 352
 - środowiskowa
 - PATH, 153
 - IFS, 327, 338
- zmienne
 - globalne, 441
 - lokalne, 442
 - parametryczne, 354
 - pól danych, 566
 - programu gawk, 683
 - środowiskowe, 142, 265
 - domyślne, 149
 - globalne, 143, 147
 - lokalizacja, 154
 - lokalne, 144
 - powłoki Bash, 149–152, 671–675
 - powłoki Dash, 593
 - ustawienia stałe, 160
 - usuwanie, 148
 - zdefiniowane przez użytkownika, 145
 - tablicowe, 161, 443, 573
 - iteracja, 573
 - usuwanie, 574
 - użytkownika, 266
- znak
 - &, 382, 547
 - dwukropka (:), 636
 - gwiazdki (*), 82, 523
 - karetki (^), 516
 - końca pliku, 488
 - kropki (.), 78, 518, 537
 - krzyżyka (#), 262
 - negowania poleceń (!), 541
 - odwróconego apostrofu ('), 268, 272
 - plusa, 525
 - podwójnej większości (>>), 270
 - potoku (|), 373, 527, 637
 - reszty z dzielenia (%), 571
 - shebang, 507
 - tyldy (~), 77, 268, 575
 - ukośnika prawego (/), 76, 494
 - ukośnika wstecznego (\), 298
 - większości (>), 270
 - zachęty (\$), 69, 517
 - zapytania (?), 82, 524
- znaki
 - !!, 137
 - #!, 284
 - &-, 388
 - dwóch kropek (..), 78
 - łącznika (--), 360
 - potęgowania (^ lub **), 571
 - specjalne, 515
 - zakotwiczenia, 516

PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion

Wiersz poleceń: oto pełny potencjał Linuksa!

Odkąd systemy linuksowe mają atrakcyjne interfejsy graficzne, wielu administratorów wykonuje swoje zadania za pomocą myszy. Wiersz poleceń jest o wiele trudniejszym interfejsem, ten sposób pracy jednak pozwala na wykorzystanie pełnego potencjału Linuksa i uzyskanie dostępu do funkcji, które w inny sposób byłyby nieosiągalne. Umiejętność pisania skryptów wiersza poleceń i powłoki Linuksa wciąż jest niezwykle ważna dla każdego administratora, któremu zależy na efektywnym działaniu systemu.

Ta książka, podobnie jak inne z serii „Biblia”, zawiera zarówno niezbędne teoretyczne informacje, jak i mnóstwo praktycznych wskazówek i instrukcji, dzięki czemu jest znakomitą pomocą w nauce pisania skryptów dla Linuksa. Pokazano tu, kiedy efektywniej jest używać interfejsu graficznego, a kiedy lepiej sięgnąć do wiersza poleceń. Opisano elementy systemu Linux i przedstawiono podstawowe informacje o powłoce, a także o pracy w wierszu poleceń. Zaprezentowano techniki przetwarzania danych użytkownika w skryptach, pracę z systemami plików, wreszcie – sposoby instalacji i aktualizacji oprogramowania. Sporo miejsca poświęcono pracy z wyrażeniami regularnymi i zaawansowanym metodom manipulacji danymi, pokazano też, jak zmodyfikować skrypty powłoki, aby działały w innych powłokach Linuksa.

Najciekawsze zagadnienia:

- tworzenie praktycznych skryptów i narzędzi skryptowych i zarządzanie nimi
- alternatywne powłoki, takie jak dash i zsh
- edytory gawk i sed
- podstawowe i rozszerzone wyrażenia regularne
- tworzenie skryptów powłoki dla pulpityw graficznych w środowiskach KDE i GNOME
- pisanie podstawowych i zaawansowanych funkcji

Richard Blum od ponad 30 lat administruje systemami komputerowymi i angażuje się w różnego rodzaju przedsięwzięcia non profit. Jest autorem popularnych książek na temat Linuksa. Lubi grać na fortepianie i gitarze basowej.

Christine Bresnahan jest adiunktką w Ivy Tech Community College, gdzie prowadzi zajęcia certyfikacyjne z Linuksa i uczy programowania w języku Python. Jest też autorką książek i materiałów dydaktycznych. Jej pasją to ogród i piesze wycieczki.

	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	ISBN 978-83-8322-074-1	
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788383 220741	
Cena: 149,00 zł		

WILEY