

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Skrypty powłoki. Od podstaw

Autorzy: Eric Foster-Johnson,
John C. Welch, Micah Anderson
Tłumaczenie: Przemysław Szeremiota
ISBN: 83-246-0209-7
Tytuł oryginału: [Beginning Shell Scripting](#)
Format: B5, stron: 576



Wykorzystaj wszystkie możliwości systemu operacyjnego

- Poznaj rodzaje powłok
- Naucz się pisać skrypty i używaj ich do różnych zadań
- Posługuj się skryptami do sterowania aplikacją

Mimo dynamicznego rozwoju graficznych systemów operacyjnych niektóre zadania można wykonywać jedynie z poziomu konsoli tekstowej. Korzystając z niej, administrator precyzyjniej kontroluje działanie systemu, szybciej rozwiązuje problemy sprzętowe i sprawniej optymalizuje standardowe procesy. Powłoki i skrypty stanowią odpowiednie narzędzia pracy dla doświadczonych użytkowników systemów operacyjnych.

Książka „Skrypty powłoki. Od podstaw” przedstawia metody korzystania z powłoki tekstowej w różnych systemach operacyjnych – Windows, Mac OS X, Linux i Unix. Omawia zarówno proste, jak i zaawansowane skrypty oraz pokazuje możliwości ich zastosowania. Każde zagadnienie przedstawione jest na przykładzie, co ułatwia przyswajanie wiedzy. Książka zawiera wszystkie wiadomości o skryptach niezbędne do samodzielnego ich tworzenia i wykorzystywania.

- Powłoki w różnych systemach operacyjnych
- Narzędzia do edycji skryptów powłoki
- Stosowanie zmiennych
- Operacje wejścia i wyjścia
- Sterowanie działaniem skryptów
- Interakcja skryptu z systemem operacyjnym
- Przetwarzanie tekstów
- Kontrolowanie procesów systemowych
- Testowanie skryptów
- Stosowanie skryptów w środowiskach graficznych

Poznaj techniki, dzięki którym wykorzystasz całą moc komputera



Spis treści

O autorach	11
Wprowadzenie	13
Rozdział 1. Powłoki — wprowadzenie	19
Czym jest powłoka?	20
Po co nam powłoki?	21
Rodzaje powłok	22
Powłoka Bourne'a	23
Powłoka C	23
Powłoka Korn	24
Powłoka bash	25
Powłoka T C	26
Inne powłoki	26
Powłoki graficzne	27
Oficjalna powłoka POSIX	28
Powłoki domyślne	28
Wybór powłoki	29
Zmiana powłoki domyślnej	30
Uruchamianie powłoki w powłoce	32
Więcej informacji o powłoce	33
Powłoki a środowisko graficzne	33
Uruchamianie powłoki w Linuksie	35
Uruchamianie powłoki w Mac OS X	37
Uruchamianie powłoki w systemach Unix	38
Uruchamianie powłoki w Windows — command.com to mało?	39
Uruchamianie powłoki na urządzeniach PDA i w innych systemach	40
Wprowadzanie poleceń	40
Jaka to powłoka?	42
Opcje wywołania	44
Edycja wiersza polecenia	48
Przywoływanie poleceń	49
Przeglądanie historii poleceń	53
Wywoływanie edytora wiersza polecenia	55
Uzupełnianie nazw plików	56
Symbole wieloznaczne	57
Symbol *	57
Symbol ?	59
Uruchamianie poleceń w tle	60
Podsumowanie	60

Rozdział 2. Wprowadzenie do skryptów powłoki	63
Czym są skrypty powłoki?	64
Alternatywne języki skryptowe	67
Perl	68
Python	70
Tcl	71
Pliki wsadowe MS-DOS	72
Narzędzia edycji skryptów powłoki	72
Edytory tradycyjne	72
Edytory graficzne	85
Pisanie skryptów	93
Odciążanie pamięci — skrypty wywołujące proste polecenia	93
Wyprowadzanie tekstu ze skryptów	95
Zmienne	99
Pozyskiwanie danych z wejścia	105
Komentarze w skryptach	107
Łamanie wierszy	110
Podsumowanie	111
Zadania	112
Rozdział 3. Sterowanie przebiegiem wykonywania skryptów	115
Odwołania do zmiennych	116
Pętle i iteracje	120
Przeglądanie listy plików	121
Pętla o określonej liczbie iteracji	124
Powłoka bash — pętla jak w języku C	127
Pętla w powłoce C	129
Pętla zagnieżdżona	130
Instrukcje warunkowe — jeśli	131
A jeśli nie?	133
Czym jest prawda?	133
Przekierowywanie wyjścia	137
elif — skrót od else if	143
Zagnieżdżanie instrukcji if	145
Testowanie poleceniem test	146
Porównywanie liczb	146
Porównywanie ciągów tekstowych	149
Testowanie plików	152
Operatory logiczne i operator negacji	152
Skrócony zapis polecenia test	155
Trudne decyzje a instrukcja wyboru	157
Obsługa problematycznych danych wejściowych	159
Instrukcja wyboru w powłoce C	161
Pętla while — powtarzanie pod warunkiem	163
Powtarzanie pod warunkiem — pętla until	164
Podsumowanie	166
Zadania	166

Rozdział 4. Interakcja ze środowiskiem skryptu	169
Zmienne środowiskowe	169
Odczytywanie wartości zmiennych środowiskowych	170
Ustawianie zmiennych środowiskowych	184
Dostosowywanie własnego konta	187
Rozruch powłoki Bourne'a	188
Rozruch powłoki Korn	188
Rozruch powłoki C	188
Rozruch powłoki T C	189
Rozruch powłoki bash	189
Obsługa argumentów wiersza poleceń	190
Wczytywanie argumentów wywołania w powłoce Bourne'a	191
Wczytywanie argumentów wywołania w powłoce C	195
Usamodzielnianie skryptów powłoki	195
Nadawanie skryptowi atrybutu wykonywalności	195
Magiczny wiersz #!	196
Podsumowanie	200
Zadania	200
Rozdział 5. Praca z plikami	201
Tworzenie archiwów plików	202
Manipulowanie uprawnieniami	203
Analizowanie atrybutów plików poleceniem test	205
Pliki w systemie Mac OS X	207
Spuścizna po NeXT	207
Mobilne systemy plików w Mac OS X	208
Nazewnictwo	210
Odwieczna wojna w Mac OS X — HFS+ kontra UFS	210
Pliki na widelcu	211
Pliki w plikach i programy interaktywne	213
Wyświetlanie komunikatów z wejścia wsobnego	214
Dynamiczny tekst wejścia wsobnego	215
Wejście wsobne a sterowanie przebiegiem poleceń interaktywnych	219
Blokowanie podstawiania zmiennych	222
Podsumowanie	223
Zadania	224
Rozdział 6. Przetwarzanie tekstu edytorem sed	225
sed — wprowadzenie	226
Wersje edytora sed	227
Instalowanie edytora sed	228
Instalacja — faza wstępna	228
Konfiguracja i właściwa instalacja	229
Zasada działania edytora sed	230
Wywoływanie edytora	231
Polecenia edycji	232
Wywoływanie edytora z opcją -e i nazwą pliku źródłowego	233
Opcje -n, -quiet i -silent	234
Błędy edycji	236

Wybieranie wierszy do obróbki	236
Adresowanie zakresowe	237
Negacja adresu	239
Postęp adresu	239
Podstawianie	240
Znaczniki podstawiania	242
Alternatywny separator ciągów	243
Adresowanie podstawiania	244
Zaawansowane wywołania sed	245
Komentarze	247
Polecenia wstawiania, dołączania i zmiany	248
Adresowanie zaawansowane	249
Adresowanie wyrażeniami regularnymi	250
Klasy znaków	253
Adresowanie zakresowe z użyciem wyrażeń regularnych	254
Adresowanie mieszane	255
Podstawienia zaawansowane	256
Odwołania do dopasowanych ciągów	256
Odwołania do podwyrażeń	258
Obszar zapasowy	259
Dodatkowe źródła informacji	261
Jednowierszowce	261
Wybrane polecenia sed	263
Mniej znane polecenia sed	263
Rozszerzenia GNU	264
Podsumowanie	264
Zadania	266

Rozdział 7. Przetwarzanie tekstu w języku awk 267

Czym jest awk (gawk, mawk, nawk, oawk)?	268
Gawk, czyli GNU awk	268
Jaka to wersja?	269
Instalowanie gawk	270
Zasada działania awk	272
Wywoływanie awk	273
Instrukcja print	276
Separatory pól	279
Instrukcja printf	280
Modyfikatory formatu instrukcji printf	282
Funkcja sprintf	283
Zmienne w awk	284
Zmienne użytkownika	284
Zmienne wbudowane	285
Instrukcje sterujące	288
Instrukcja if	289
Operatory relacji	290
Funkcje arytmetyczne	291
Przekierowywanie wyjścia	293
Pętle while	293
Pętle for	294

Funkcje	295
Dodatkowe źródła informacji	296
Podsumowanie	297
Zadania	297
Rozdział 8. Potoki poleceń	299
Obsługa standardowego wejścia i wyjścia	300
Przekierowywanie standardowego wejścia i wyjścia	300
Przekierowywanie wyjścia diagnostycznego	302
Przekierowywanie obu wyjść: standardowego i diagnostycznego	302
Dołączanie wyjścia do plików	304
Szybkie usuwanie zawartości plików	305
Pozbywanie się wyjścia	306
Przetwarzanie w potoku poleceń	307
Potoki poleceń uniksowych	308
Tworzenie potoków	309
Rozdział strumienia wyjściowego — polecenie tee	315
Podsumowanie	316
Zadania	317
Rozdział 9. Kontrola nad procesami	319
Procesy	319
Odczytywanie identyfikatorów procesów	320
System plików /proc	324
Eliminowanie procesów	328
Uruchamianie procesów	329
Uruchamianie procesów pierwszoplanowych	330
Uruchamianie procesów tła	330
Uruchamianie procesów w podpowłokach	330
Uruchamianie procesów poleceniem exec	331
Przechwytywanie wyjścia procesów	332
Podstawianie poleceń	332
Przechwytywanie kodów powrotnych poleceń zewnętrznych	341
Podsumowanie	342
Zadania	343
Rozdział 10. Funkcje w skryptach powłoki	345
Definiowanie funkcji	346
Opatrywanie bloków kodu nazwami	346
Błędy definicji funkcji	348
Używanie funkcji	349
Reguła pierwszeństwa definicji przed użyciem	350
Pliki funkcji	353
Typowe błędy użycia funkcji	354
Usuwanie funkcji	355
Funkcje z argumentami	355
Zwracanie wartości z funkcji	356
Zasięg zmiennych, czyli myśl globalnie, działaj lokalnie	359
Rekurencja	362
Podsumowanie	364
Zadania	365

Rozdział 11. Diagnostyka błędów w skryptach powłoki	367
Rozszyfrowywanie komunikatów o błędach	368
Szukanie brakujących elementów składni	369
Wyszukiwanie błędów składniowych	371
Techniki diagnostyczne w wykrywaniu błędów	374
Szukaj wstecz	374
Szukaj oczywistych pomyłek	374
Szukaj wszelkich dziwactw	375
Szukaj niejawnych założeń	376
Dziel i rządź	376
Podziel skrypt na fragmenty	377
Prześledź działanie skryptu	378
Co dwie głowy, to nie jedna	378
Uruchamianie skryptów w trybie diagnostycznym	379
Blokowanie wykonywania poleceń	379
Śledzenie przebiegu wykonywania	380
Połączenie opcji -n i -v	381
Nadzorowanie wykonywania skryptu	381
Unikanie błędów, czyli dobre nawyki	385
Porządek musi być	385
Zbawienny wpływ komentarzy	386
Treściwe komunikaty o błędach	386
Lepsze wrogiem dobrego	387
Testy, testy i jeszcze raz testy	387
Podsumowanie	387
Zadania	388
Rozdział 12. Obrazowanie danych — MRTG	391
Zasada działania MRTG	392
Monitorowanie innych danych	393
Instalowanie MRTG	393
Pisanie skryptów dla MRTG	394
Konfigurowanie MRTG	397
Konfigurowanie parametrów globalnych	398
Konfigurowanie obiektów obserwacji MRTG dla skryptów	399
Dostosowywanie wyjścia MRTG	400
Uruchamianie MRTG	404
Podgląd wyników MRTG	405
Konfigurowanie crona	406
Maksymalizacja wydajności MRTG	407
Monitorowanie stanu systemu z MRTG	408
Obrazowanie obciążenia pamięci	408
Obrazowanie obciążenia procesora	412
Obrazowanie zajętości dysku	415
Monitorowanie sieci z MRTG	418
Monitorowanie aplikacji z MRTG	421
Podsumowanie	428
Zadania	428

Rozdział 13. Skrypty w służbie administracji	431
Po co administratorom skrypty?	431
Skrypty odciążające pamięć (administratora)	432
Skrypty w diagnostyce systemu	435
Usuwanie pomniejszych niewygod	443
Rafinacja danych	445
Automatyzacja codziennych zadań	449
Podsumowanie	450
Zadania	450
Rozdział 14. Skrypty w środowiskach graficznych	453
Aplikacje biurowe	454
Skrypty w OpenOffice.org	454
Możliwości skryptowe edytora AbiWord	468
Możliwości skryptowe edytora NEdit	469
Skrypty w środowisku graficznym Mac OS X	470
Open Scripting Architecture	473
Podstawy języka AppleScript	474
Terminal w Mac OS X	486
Skrypty w służbie rozrywce	493
Skryptowe sterowanie odtwarzaczem XMMS	493
Skryptowe sterowanie odtwarzaczem Rhythmbox	494
Skryptowe sterowanie odtwarzaczem Totem	496
Stosowanie skryptów z innymi aplikacjami środowiska graficznego	496
Co dalej?	497
Podsumowanie	497
Zadania	498
A Rozwiązania zadań	499
B Wybrane polecenia	521
Skorowidz	555

4

Interakcja ze środowiskiem skryptu

Skrypt powłoki nie jest odosobnioną wyspą. Działa jako program ściśle osadzony w środowisku utworzonym dla niego przez system operacyjny. Na potrzeby skryptów powłoki większość systemów udostępnia środowisko charakterystyczne dla systemów uniksowych, co jest wielce pomocne, bo pozwala na spójne wykonywanie odwołań do elementów środowiska niezależnie od platformy programowej. Taką możliwość mamy w systemie Mac OS X (bazującym na Uniksie w wydaniu Berkeley) i Linuksie (systemie nie uniksowym, ale uniksopodobnym). Nawet w Windows i QNX skrypty powłoki otrzymują do dyspozycji środowisko charakterystyczne dla Uniksów.

Środowisko uniksowe cechuje się między innymi dostępnością *zmiennych środowiskowych*, przechowujących ważne dane, choćby o położeniu plików poleceń czy lokalizacji katalogu domowego użytkownika skryptu.

W rozdziale zajmiemy się:

- Analizowaniem ustawień środowiskowych, odczytywaniem i ustawianiem zmiennych środowiskowych.
- Dostosowywaniem własnego konta, głównie w zakresie rozruchu powłoki.
- Obsługą argumentów — wartości mających sterować działaniem skryptu a przekazywanych do skryptów za pośrednictwem wiersza polecenia.
- Oznaczaniem plików skryptów jako wykonywalnych, dzięki czemu można z nich korzystać tak, jak z pozostałych poleceń zainstalowanych w systemie (jak choćby `ls` czy `cp`).

Zmienne środowiskowe

Zmienne środowiskowe są rodzajem zmiennych powłoki.

Zasadniczo zmienne środowiskowe nie różnią się wiele od pozostałych zmiennych powłoki. Różnice można wypunktować następująco:

- Zmienne środowiskowe są ustawiane przez środowisko — system operacyjny; odbywa się to na etapie uruchamiania powłoki (i nie ma w tym żadnej magii, o czym przekonamy się w dalszej części rozdziału).
- Zmienne powłoki są zmiennymi lokalnymi względem danego egzemplarza procesu powłoki — na przykład egzemplarza powłoki wywołanego do uruchomienia skryptu. Tymczasem zmienne środowiskowe są dziedziczone przez wszystkie egzemplarze powłoki i każdy uruchamiany program.
- Zmienne środowiskowe posiadają specjalne, ustalone konwencjami znaczenie.
- Ustawienie zmiennej środowiskowej odbywa się za pośrednictwem osobnej, specjalnej operacji.

Wymienione tu różnice zostaną szerzej skomentowane w następnych punktach.

Odczytywanie wartości zmiennych środowiskowych

Zmienne środowiskowe obsługuje się w wielu kontekstach identycznie jak zmienne powłoki. Oznacza to, że jeśli w skład środowiska wchodzi zmienna środowiskowa o nazwie HOME, to za pomocą symbolu dolara (\$) można się odwołać do jej wartości (\$HOME). W tym aspekcie korzystanie ze zmiennych środowiskowych nie różni się od stosowania własnych zmiennych skryptu powłoki.

Najważniejszą cechą zmiennych środowiskowych jest to, że przechowują wartości, które powinny odwzorowywać kształt środowiska udostępnionego użytkownikowi, jak również uwzględniać jego (użytkownika) preferencje (często zaś odzwierciedlają preferencje administratora).

Na przykład zmienna środowiskowa LANG, o ile w ogóle zostanie ustawiona, powinna odzwierciedlać schemat lokalizacji przyjęty przez użytkownika — kombinację strony kodowej i reguł formatowania dla danego języka. Użytkownik anglojęzyczny może korzystać ze schematu lokalizacji o nazwie en. Mieszkańcy Zjednoczonego Królestwa zechcą zapewne korzystać ze schematu lokalizacji uwzględniającego lokalne reguły formatowania, o nazwie en_UK; Amerykanie w jego miejsce zastosują pewnie odmianę właściwą dla Stanów Zjednoczonych — en_US¹. Ustawienia te są o tyle istotne, że sterują działaniem wielu programów, choćby systemowego programu kontroli poprawności pisowni.

Skrypt powłoki powinien honorować ustawienia zmiennych środowiskowych wszędzie tam, gdzie jest to zasadne. Jest to o tyle skomplikowane, że w różnych systemach zestawy dostępnych zmiennych środowiskowych różnią się składem. Skrypt powinien więc każdorazowo, przed odwołaniem się do zmiennej środowiskowej, sprawdzić jej istnienie, ewentualnie korzystać z wartości domyślnej.

Listę najczęściej spotykanych zmiennych środowiskowych zawiera tabela 4.1:

¹ Dla języka polskiego odpowiednia może być wartość pl_PL — *przyp. tłum.*

Tabela 4.1. Wybrane zmienne środowiskowe

Zmienna	Znaczenie
COLUMNS	Liczba znaków mieszczących się w pojedynczym wierszu okna powłoki (terminala).
DISPLAY	Nazwa ekranu docelowego dla programów korzystających z usług serwera X Window System.
HISTSIZE	Liczba wpisów przechowywanych w historii poleceń (dotyczy powłok bash i ksh).
HOME	Nazwa katalogu domowego użytkownika.
HOSTNAME	Nazwa węzła (identyfikująca komputer w sieci).
LANG	Nazwa schematu lokalizacji językowej.
LINES	Liczba wierszy tekstu mieszczących się w oknie powłoki (terminalu)
PATH	Ścieżka przeszukiwania — lista katalogów przeszukiwanych przy wywoływaniu poleceń.
PWD	Nazwa bieżącego katalogu roboczego.
SHELL	Nazwa domyślnej powłoki.
TERM	Typ terminala.
USER	Nazwa konta użytkownika.

Listę zmiennych środowiskowych widocznych dla powłoki wywołuje się poleceniem `set`.

W `csh` i `tcsh` zamiast `set` należy stosować polecenie `setenv` (szczegóły w dalszej części omówienia).

Wbudowane polecenie powłoki o nazwie `set` służy między innymi do wypisywania list ustawionych zmiennych. Tam, gdzie jest możliwość korzystania z różnych systemów operacyjnych, warto porównać sobie wyniki jego działania. Przytoczone poniżej przykłady ilustrują dobór zmiennych środowiskowych w różnych systemach operacyjnych.

Zwróć uwagę na to, że wykaz zawiera nie tylko prawdziwe zmienne środowiskowe, ale o tym powiemy sobie w dalszej części omówienia.

spróbuj sam Środowisko powłoki w Linuksie

W dystrybucji Fedora Core 2 wywołanie polecenia `set` daje następujący efekt:

```
$ set
BASH=/bin/bash
BASH_VERSIONINFO=( [0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i386-redhat-
linux-gnu" )
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLORTERM=gnome-terminal
COLUMNS=73
DESKTOP_SESSION=default
DIRSTACK=( )
DISPLAY=:0.0
EUID=500
```

```
GDMSESSION=default
GNOME_DESKTOP_SESSION_ID=Default
GNOME_KEYRING_SOCKET=/tmp/keyring-tNORrP/socket
GROUPS=()
GTK_RC_FILES=/etc/gtk/gtkrc:/home2/ericfj/.gtkrc-1.2-gnome2
G_BROKEN_FILENAMES=1
HISTFILE=/home2/ericfj/.bash_history
HISTFILESIZE=1000
HISTSIZ=1000
HOME=/home2/ericfj
HOSTNAME=kirkwall
HOSTTYPE=i386
IFS=$' \t\n'
INPUTRC=/etc/inputrc
KDEDIR=/usr
LANG=en_US.UTF-8
LESSOPEN='/usr/bin/lesspipe.sh %s'
LINES=24
LOGNAME=ericfj
LS_COLORS='no=00:fi=00:di=00:34:ln=00:36:pi=40:33:so=00:35:bd=40:33:01:cd=40:33:01:or
=01:05:37:41:mi=01:05:37:41:ex=00:32:*.cmd=00:32:*.exe=00:32:*.com=00:32:*.btm=00:32:
*.bat=00:32:*.sh=00:32:*.csh=00:32:*.tar=00:31:*.tgz=00:31:*.arj=00:31:*.taz=00:31:*.
lzh=00:31:*.zip=00:31:*.z=00:31:*.Z=00:31:*.gz=00:31:*.bz2=00:31:*.bz=00:31:*.tz=00:3
1:*.rpm=00:31:*.cpio=00:31:*.jpg=00:35:*.gif=00:35:*.bmp=00:35:*.xbm=00:35:*.xpm=00:3
5:*.png=00:35:*.tif=00:35:'
MACHTYPE=i386-redhat-linux-gnu
MAIL=/var/spool/mail/ericfj
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home2/ericfj/bin:
/usr/java/j2sdk1.4.1_03/bin:/opt/jext/bin
PIPESTATUS=(["0"]="0")
PPID=19277
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%.*}:${PWD/#HOME/~}\007"'
PS1='\u@\h \w\$'
PS2='> '
PS4='+ '
PWD=/home2/ericfj/web/local
QTDIR=/usr/lib/qt-3.3
SESSION_MANAGER=local/kirkwall:/tmp/.ICE-unix/19167
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=2
SSH_AGENT_PID=19215
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-tke19167/agent.19167
SUPPORTED=en_US.UTF-8:en
TERM=xterm
UID=500
USER=ericfj
WINDOWID=20971638
XAUTHORITY=/home2/ericfj/.Xauthority
_=env
```

Jak to działa?

Z przytoczonego wypisu wynika, że w Linuksie zestaw zmiennych środowiskowych jest wcale pokazany. Wiele z tych zmiennych to wartości kojarzone z konkretnymi programami czy operacjami. Na przykład zmienne środowiskowe `MAILCHECK` i `MAIL` określają częstotliwość sprawdzania dostępności nowych wiadomości poczty elektronicznej i miejsce, w którym należy ich szukać.

Przyjęło się, że nazwy zmiennych środowiskowych zawierają wyłącznie wielkie litery — warto trzymać się tej konwencji.

Dokumentacja każdego programu, który korzysta ze zmiennych środowiskowych, powinna ten fakt sygnalizować i wymieniać wykorzystywane zmienne oraz ich wpływ na działanie programu. Na przykład polecenie służące do wyświetlania stron dokumentacji systemowej, `man`, bazuje na zmiennej środowiskowej `MANPATH`, która (o ile zostanie ustawiona) określa położenie plików dokumentacji.

spróbuj sam Środowisko powłoki w Mac OS X

W systemie Mac OS X powinieneś mieć dostęp do następujących zmiennych:

```
$ set
BASH=/bin/bash
BASH_VERSINFO=( [0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="powerpc-apple-darwin7.0" )
BASH_VERSION='2.05b.0(1)-release'
COLUMNS=80
DIRSTACK=()
EUID=501
GROUPS=()
HISTFILE=/Users/ericfj/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/Users/ericfj
HOSTNAME=Stromness.local
HOSTTYPE=powerpc
IFS=$' \t\n'
LINES=24
LOGNAME=ericfj
MACHTYPE=powerpc-apple-darwin7.0
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=darwin7.0
PATH=/bin:/sbin:/usr/bin:/usr/sbin
PIPESTATUS=( [0]="0" )
PPID=524
PS1='\h:\w \u\$ '
PS2='> '
PS4='+ '
PWD=/Users/ericfj
SECURITYSESSIONID=10967b0
```

```
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=1
TERM=vt100
TERM_PROGRAM=iTerm.app
UID=501
USER=ericfj
_=/etc/bashrc
__CF_USER_TEXT_ENCODING=0x1F5:0:0
```

Jak to działa?

Zwróćmy uwagę na podobieństwo wypisu uzyskanego w systemie Mac OS X z tym znanym z Linuksa. Dla skryptu powłoki oba systemy stanowią bardzo podobne środowiska wykonania. To wygodne (dla programisty skryptów), zwłaszcza jeśli ujednolicenie środowiska zestawia się z ogólną odmiennością obu tych systemów.

spróbuj sam Środowisko powłoki w Windows XP

W systemie Windows XP, z protezą w postaci pakietu Cygwin, skrypt ma dostęp do następujących zmiennych powłoki:

```
$ set
!::=':::\'
!C:='C:\cygwin\bin'
ALLUSERSPROFILE='C:\Document and Settings\All Users'
ANT_HOME='C:\ericfj\java\apache-ant-1.5.4'
APPDATA='C:\Documents and Settings\ericfj\Application Data'
BASH=/usr/bin/bash
BASH_VERSION=([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i686-pc-cygwin")
BASH_VERSION='2.05b.0(1)-release'
COLUMNS=80
COMMONPROGRAMFILES='C:\Program Files\Common Files'
COMPUTERNAME=GURNESS
COMSPEC='C:\WINDOWS\system32\cmd.exe'
CVS_RSH=/bin/ssh
DIRSTACK=()
EUID=1006
FP_NO_HOST_CHECK=NO
GROUPS=()
HISTFILE=/home/ericfj/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/home/ericfj
HOMEDRIVE=C:
HOMEPATH='Documents and Settings\ericfj'
HOSTNAME=kirkwall
HOSTTYPE=i686
IFS=$'\t\n'
INFOPATH=/usr/local/info:/usr/info:/usr/share/info:/usr/autotool/devel/info:/usr/autotool/stable/info:
JAVA_HOME='C:\j2sdk1.4.2_01'
LINES=25
LOGONSERVER='\\KIRKWALL'
```

```

MACHTYPE=i686-pc-cygwin
MAILCHECK=60
MAKE_MODE=unix
MANPATH=/usr/local/man:/usr/man:/usr/share/man:/usr/autotool/devel/man::/usr/ssl/man
MAVEN_HOME='C:\ericfj\java\namen-1.0-rc1'
NUMBER_OF_PROCESSORS=1
OLDPWD=/usr/bin
OPTERR=1
OPTIND=1
OS=Windows_NT
OSTYPE=cygwin
PALMTOPCENTERDIR='C:\Program Files\Sharp Zaurus 2\Qtopia Desktop'
PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/cygdrive/c/WINDOWS/system32:/cygdri
ve/c/WINDOWS:/cygdrive/c/WINDOWS/System32/Wbem:/cygdrive/c/ericfj/apps:/cygdrive/c/er
icfj/java/apache-ant-1.5.4/bin:/cygdrive/c/j2sdk1.4.2_01/bin:/usr/bin:.
PATHEXT='.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH'
PIPESTATUS=( [0]="0" )
PPID=1
PRINTER='HP LaserJet 2100 PCL6'
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER='x86 Family 15 Model 2 Stepping 9, GenuineIntel'
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0209
PROGRAMFILES='C:\Program Files'
PROMPT='$P$G'
PS1='$\[\033]0;\w\007\n\033[32m\]\u@\h \[\033[33m\w\033[0m\]\n$ '
PS2='> '
PS4='+ '
PWD=/home/ericfj
SESSIONNAME=Console
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=1
SYSTEMDRIVE=C:
SYSTEMROOT='C:\WINDOWS'
TEMP=/cygdrive/c/DOCUME~1/ericfj/LOCALS~1/Temp
TERM=cygwin
TMP=/cygdrive/c/DOCUME~1/ericfj/LOCALS~1/Temp
UID=1006
USER=ericfj
USERDOMAIN=ORKNEY
USERNAME=ericfj
USERPROFILE='C:\Documents and Settings\ericfj'
WINDIR='C:\WINDOWS'
_=/home/ericfj/.bashrc
f=

```

Jak to działa?

W tym przykładzie widać wyraźnie spuściznę po systemie DOS, choćby w ścieżkach dostępu, zaczynających się od litery dysku (C:). Pakiet Cygwin znakomicie jednoczy wymagania co do środowiska widocznego dla powłoki z realiami systemów z rodziny Windows. Widać tu na przykład, że zmienne TEMP (Unix) i TMP (Windows) są ustawione zgodnie z lokalizacją systemowych katalogów plików tymczasowych, podobnie jak zmienne USER (Unix) i USERNAME (Windows).

Zmienne lokalne a zmienne środowiskowe

Technicznie rzecz biorąc, polecenie `set`, wykorzystywane w kilku ostatnich przykładach, wypisuje *wszystkie* ustawione zmienne powłoki, w tym zarówno zmienne środowiskowe, jak i wszelkie pozostałe, zwane *zmiennymi lokalnymi powłoki*. Zmienne lokalne to zmienne dostępne w obrębie bieżącej powłoki. Zaś *zmienne środowiskowe* to te zmienne powłoki, które zostały wyeksportowane. Eksportowanie udostępnia te zmienne wszystkim programom potomnym, również nowouruchamianym powłokom, czyli powłokom uruchamianym z wnętrza bieżącej powłoki. Proste?

Możliwość uruchamiania powłoki z wnętrza powłoki oznacza możliwość tworzenia hierarchii powłok, co oznacza z kolei, że skrypty mogą działać w obrębie powłoki pierwotnej, podpowłoki, podpodpowłoki i tak dalej. Przy logowaniu użytkownika na jego konto systemowe, system uruchamia przypisaną do tego konta aplikację; zwykle jest to właśnie powłoka. Ta powłoka jest „rodzicem” wszystkich programów uruchamianych potem przez użytkownika, a więc i wszystkich wywoływanych później powłok. Powłoka-rodzic, czyli powłoka pierwotna, nosi miano powłoki logowania (bo jest uruchamiana przy logowaniu użytkownika). I to ona decyduje o kształcie środowiska dostępnego dla procesów potomnych. Powłoka pierwotna uruchamia potem szereg aplikacji. Na przykład w systemie Linux działającym w trybie graficznym powłoka logowania uruchamia proces serwera X Window System. Serwer X uruchamia potem zestaw aplikacji obsługujących środowisko graficzne, w tym wszelkie okna powłoki (okna wirtualnych terminali) udostępniane użytkownikowi tego środowiska. Każde z tak uruchomionych okien powłoki jest aplikacją graficzną i każde wywołuje z kolei program powłoki. Tak wywołane powłoki są podpowłokami.

Z tych podpowłok można uruchomić dalsze (pod)powłoki. Ćwiczyliśmy to wielokrotnie przy uruchamianiu kolejnych skryptów przykładowych. Za każdym razem kiedy w wierszu polecenia wpisywaliśmy `sh`, `bash` czy `csh`, uruchamialiśmy podpowłokę — powłokę potomną powłoki bieżącej (to właśnie między innymi dlatego w systemach uniksowych, jak również w Linuksie, działa stale duża liczba procesów).

Kiedy tworzy się zmienną środowiskową, ustaloną przy tej okazji wartość udostępnia się podpowłokom — powłokom potomnym powłoki bieżącej, uruchamiającej skrypt. Jest to pożądane, kiedy skrypt wywołuje zewnętrzne skrypty bądź polecenia i ma do nich przekazywać wartości sterujące; można to osiągnąć właśnie za pośrednictwem zmiennych środowiskowych. Ustawienie zmiennej środowiskowej nie modyfikuje jednak środowiska powłoki nadrzędnej (powłoki-rodzica) — eksport oznacza propagację zmiennej w dół hierarchii, nigdy w górę.

W większości przypadków środowisko powłok ustala się przy ich uruchamianiu. Dokonuje się tego za pośrednictwem specjalnych plików konfiguracyjnych, omawianych w podrozdziale „Dostosowywanie własnego konta”.

W razie wątpliwości należy trzymać się poniższych wytycznych:

- Środowisko własnego konta należy ustalać za pośrednictwem standardowych plików konfiguracyjnych, omawianych w podrozdziale „Dostosowywanie własnego konta”.

- Zastane środowisko należy modyfikować zmiennymi ustawianymi przez skrypty powłoki jedynie wtedy, kiedy skrypty te wywołują programy lub skrypty zewnętrzne, dla których konieczna jest modyfikacja bądź uzupełnienie środowiska.
- We wszystkich pozostałych przypadkach należy powstrzymać się od eksportowania zmiennych powłoki.

spróbuj sam Wykaz zmiennych środowiskowych

Do wypisania wartości zmiennych eksportowanych, czyli jedynych prawdziwych zmiennych środowiskowych, służy polecenie `env` albo `printenv`. Oto przykład:

```
$ printenv
SSH_AGENT_PID=22389
HOSTNAME=kirkwall
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
GTK_RC_FILES=/etc/gtk/gtkrc:/home2/ericfj/.gtkrc-1.2-gnome2
WINDOWID=20971679
OLDPWD=/home2/ericfj/writing/beginning_shell_scripting
QTDIR=/usr/lib/qt-3.3
USER=ericfj
LS_COLORS='no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33:01:cd=40;33:01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=00;35:*.png=00;35:*.tif=00;35:'
GNOME_KEYRING_SOCKET=/tmp/keyring-Q0LxNA/socket
SSH_AUTH_SOCK=/tmp/ssh-NVH22341/agent.22341
KDEDIR=/usr
SESSION_MANAGER=local/kirkwall:/tmp/.ICE-unix/22341
MAIL=/var/spool/mail/ericfj
DESKTOP_SESSION=default
PATH=/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home2/ericfj/bin:/usr/java/j2sdk1.4.1_03/bin:/opt/jext/bin
INPUTRC=/etc/inputrc
PWD=/home2/ericfj/writing/beginning_shell_scripting/scripts
LANG=en_US.UTF-8
GDMSESSION=default
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
HOME=/home2/ericfj
SHLVL=2
GNOME_DESKTOP_SESSION_ID=Default
MY_SHELL=/usr/bin/emacs2
LOGNAME=ericfj
LESSOPEN=|/usr/bin/lesspipe.sh %s
DISPLAY=:0.0
G_BROKEN_FILENAMES=1
COLORTERM=gnome-terminal
XAUTHORITY=/home2/ericfj/.Xauthority
_=/usr/bin/printenv
```

Rzecz jasna, w systemie innym niż testowy uzyskany wykaz może się różnić od powyższego.

Jak to działa?

Zestaw zmiennych wypisywanych przez polecenie `printenv` jest mniej obszerny niż podobny wypis uzyskany poleceniem `set`. Różnica obejmuje poniższe zmienne, wypisywane przez `set` a pomijane przez `printenv`:

```
BASH=/bin/bash
BASH_VERSINFO=( [0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i386-redhat-
linux-gnu" )
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=80
DIRSTACK=()
EUID=500
GROUPS=()
HISTFILE=/home2/ericfj/.bash_history
HISTFILESIZE=1000
HOSTTYPE=i386
IFS=$' \t\n'
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=24
LS_COLORS='no=00:fi=00:di=00;34:ln=00;36:pi=00;33:so=00;35:bd=40;33:01:cd=40;33:01:or
=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*.
lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;3
1:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xpm=00;3
5:*.png=00;35:*.tif=00;35:'
MACHTYPE=i386-redhat-linux-gnu
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PIPESTATUS=( [0]="0" )
PPID=22454
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}%. *};${PWD/#HOME/~}\007"'
PS1='[\u@\h \W]\$'
PS2='> '
PS4='+ '
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SUPPORTED=en_US.UTF-8:en_US:en
UID=500
_
=
```

Wymienione zmienne są zmiennymi lokalnymi powłoki, a nie zmiennymi środowiskowymi.

Wypisywanie zmiennych środowiskowych w powłoce C

W powłoce C również mamy do dyspozycji polecenie `set`, jednak nie ujawnia ono kompletnego środowiska:

```
$ set
COLORS /etc/DIR_COLORS.xterm
_      !! | sort
```

```

addsuffix
argv      ( )
cwd       /home2/ericfj
dirstack  /home2/ericfj
dspmbyte  euc
echo_style both
edit
file      /home2/ericfj/.i18n
gid       500
group     ericfj
history   100
home      /home2/ericfj
killring  30
owd
path      (/usr/kerberos/bin /usr/local/mozilla /bin /usr/bin /usr/local/bin
/usr/X11R6/bin /home2/ericfj/bin /usr/java/j2sdk1.4.1_03/bin /home2/ericfj/eclipse
/home2/ericfj/apache-ant-1.5.4/bin)
prompt    [%n%m %c]$
prompt2   CORRECT>%R (y|n|e|a)?
shell     /bin/tcsh
shlvl     2
sourced   1
status    0
tcsh      6.12.00
term      xterm
tty       pts/19
uid       500
user      ericfj
version   tcsh 6.12.00 (Astron) 2002-07-23 (i386-intel-linux) options
8b,nls,d1,a1,kan,rh,color,dspm,filec

```

Polecenie `set` wypisuje tu zestaw wewnętrznych ustawień powłoki C, a nie listę zmiennych środowiskowych. Odpowiednikiem polecenia `set` z powłoki Bourne'a jest tu `setenv`. Wywołanie polecenia `setenv` w powłoce T C w systemie Linux powinno dać rezultaty podobne do poniższego:

```

$ setenv | sort
COLORTERM=gnome-terminal
CVSROOT=:pserver:ericfj@localhost:/home2/cvsrepos
DESKTOP_SESSION=default
DISPLAY=:0.0
G_BROKEN_FILENAMES=1
GDMSESSION=default
GNOME_DESKTOP_SESSION_ID=Default
GNOME_KEYRING_SOCKET=/tmp/keyring-w8mvQR/socket
GROUP=ericfj
GTK_RC_FILES=/etc/gtk/gtkrc:/home2/ericfj/.gtkrc-1.2-gnome2
HOME=/home2/ericfj
HOST=kirkwall
HOSTNAME=kirkwall
HOSTTYPE=i386-linux
INPUTRC=/etc/inputrc
JAVA_HOME=/usr/java/j2sdk1.4.1_03
KDEDIR=/usr
LANG=en_US.UTF-8
LESSOPEN=|/usr/bin/lesspipe.sh %s
LOGNAME=ericfj

```

```

LS_COLORS=no=00:fi=00:di=00:34:ln=00:36:pi=40:33:so=00:35:bd=40:33:01:cd=40:33:01:or=
01:05:37:41:mi=01:05:37:41:ex=00:32:*.cmd=00:32:*.exe=00:32:*.com=00:32:*.btm=00:32:*.
.bat=00:32:*.sh=00:32:*.csh=00:32:*.tar=00:31:*.tgz=00:31:*.arj=00:31:*.taz=00:31:*.l
zh=00:31:*.zip=00:31:*.z=00:31:*.Z=00:31:*.gz=00:31:*.bz2=00:31:*.bz=00:31:*.tz=00:31
:*.rpm=00:31:*.cpio=00:31:*.jpg=00:35:*.gif=00:35:*.bmp=00:35:*.xpm=00:35
:*.png=00:35:*.tif=00:35:
MACHTYPE=i386
MAIL=/var/spool/mail/ericfj
OSTYPE=linux
PATH=/usr/kerberos/bin:/usr/local/bin:/usr/local/mozilla:/bin:/usr/bin:/usr/local/bin
:/usr/X11R6/bin:/home2/ericfj/bin:/usr/java/j2sdk1.4.1_03/bin:/home2/ericfj/eclipse:/
home2/ericfj/apache-ant-1.5.4/bin
PWD=/home2/ericfj
QTDIR=/usr/lib/qt-3.3
SESSION_MANAGER=local/kirkwall:/tmp/.ICE-unix/27573
SHELL=/bin/tcsh
SHLVL=2
SSH_AGENT_PID=27574
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-bxy27574/agent.27573
SUPPORTED=en_US.UTF-8:en_US:en
TERM=xterm
USER=ericfj
USERNAME=ericfj
VENDOR=intel
WINDOWID=23068746
XAUTHORITY=/home2/ericfj/.Xauthority

```

Do wypisywania listy zmiennych środowiskowych w powłoce C można wykorzystywać również polecenia `printenv` i `env`, jak poniżej:

```

# printenv
COLORTERM=gnome-terminal
CVSROOT=:pserver:ericfj@localhost:/home2/cvsrepos
DESKTOP_SESSION=default
DISPLAY=:0.0
G_BROKEN_FILENAMES=1
GDMSESSION=default
GNOME_DESKTOP_SESSION_ID=Default
GNOME_KEYRING_SOCKET=/tmp/keyring-w8mvQR/socket
GROUP=ericfj
GTK_RC_FILES=/etc/gtk/gtkrc:/home2/ericfj/.gtkrc-1.2-gnome2
HOME=/home2/ericfj
HOST=kirkwall
HOSTNAME=kirkwall
HOSTTYPE=i386-linux
INPUTRC=/etc/inputrc
JAVA_HOME=/usr/java/j2sdk1.4.1_03
KDEDIR=/usr
LANG=en_US.UTF-8
LESSOPEN=|/usr/bin/lesspipe.sh %s
LOGNAME=ericfj
LS_COLORS=no=00:fi=00:di=00:34:ln=00:36:pi=40:33:so=00:35:bd=40:33:01:cd=40:33:01:or=
01:05:37:41:mi=01:05:37:41:ex=00:32:*.cmd=00:32:*.exe=00:32:*.com=00:32:*.btm=00:32:*.
.bat=00:32:*.sh=00:32:*.csh=00:32:*.tar=00:31:*.tgz=00:31:*.arj=00:31:*.taz=00:31:*.l
zh=00:31:*.zip=00:31:*.z=00:31:*.Z=00:31:*.gz=00:31:*.bz2=00:31:*.bz=00:31:*.tz=00:31
:*.rpm=00:31:*.cpio=00:31:*.jpg=00:35:*.gif=00:35:*.bmp=00:35:*.xpm=00:35
:*.png=00:35:*.tif=00:35:

```

```

MACHTYPE=i386
MAIL=/var/spool/mail/ericfj
OSTYPE=linux
PATH=/usr/kerberos/bin:/usr/local/bin:/usr/local/mozilla:/bin:/usr/bin:/usr/local/bin
:/usr/X11R6/bin:/home2/ericfj/bin:/usr/java/j2sdk1.4.1_03/bin:/home2/ericfj/eclipse:/
home2/ericfj/apache-ant-1.5.4/bin
PWD=/home2/ericfj
QTDIR=/usr/lib/qt-3.3
SESSION_MANAGER=local/kirkwall:/tmp/.ICE-unix/27573
SHELL=/bin/tcsh
SHLVL=2
SSH_AGENT_PID=27574
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-bxy27574/agent.27573
SUPPORTED=en_US.UTF-8:en_US:en
TERM=xterm
USER=ericfj
USERNAME=ericfj
VENDOR=intel
WINDOWID=23068746
XAUTHORITY=/home2/ericfj/.Xauthority

```

Testowanie środowiska

Padło już stwierdzenie, że skrypt powłoki powinien honorować zastane ustawienia środowiskowe. Ale skrypt musi również obsługiwać sytuacje, w których potrzebne mu zmienne nie zostały w ogóle ustawione — trzeba być przygotowanym na wszystkie ewentualności.

Na przykład zmienna środowiskowa `DISPLAY` zawiera nazwę ekranu systemu X Window System, reprezentującego kombinację monitora i urządzeń wskazujących. Programy graficzne wykorzystują ustawienie zmiennej `DISPLAY` do kierowania komunikatów do odpowiedniego serwera X. W systemach wielodostępnych i rozproszonych ustawienie to ma krytyczne znaczenie.

Serwer X Window System jest podstawą podsystemu graficznego w każdym systemie uniksowym i linuksowym. W systemach Mac OS X serwer X działa jako podsystem dodatkowy. „Ikisy” da się nawet uruchomić w systemie Windows (za pomocą pakietu Cygwin).

Jeśli zmienna środowiskowa `DISPLAY` posiada wartość, programy odwołujące się do serwera X Window System powinny ją uwzględniać. W przypadku braku tej zmiennej trzeba jednak wybrać jedną z trzech opcji:

- przyjąć dla `DISPLAY` wartość domyślną (`:0.0`);
- założyć niedostępność serwera X Window System;
- powiadomić użytkownika o braku wartości zmiennej `DISPLAY` i przerwać działanie skryptu.

Nie ma tu żadnych czarów, potrzebna tylko decyzja. Podobne decyzje trzeba podejmować w obliczu braku innych potrzebnych skryptowi zmiennych. Strategie obsługi takich sytuacji przećwiczymy na poniższym przykładzie.

spróbuj sam Sprawdzenie zmiennych środowiskowych

Zapiszmy poniższy kod w pliku o nazwie *check_env*:

```
# Kontrola zmiennych środowiskowych.

# Usuń symbol komentarza z poniższych wierszy, aby usunąć definicję zmiennej.
#unset DISPLAY

if [ "$DISPLAY" == "" ]
then
    echo "Brak zmiennej DISPLAY, przyjmuję :0.0 jako wartość domyślną."
    DISPLAY=":0.0"
fi

#unset SHELL

if [ "$SHELL" == "" ]
then
    echo "Wybieram /bin/bash jako pożądaną powłokę."
    SHELL=/bin/bash
fi

#unset USER

if [ "$USER" == "" ]
then
    echo -n "Podaj nazwę swojego konta: "
    read USER
fi

#unset HOME

if [ "$HOME" == "" ]
then
    # Sprawdzenie katalogu domowego w Mac OS X.
    if [ -d "/Users/$USER" ]
    then
        HOME="/Users/$USER"

    # Sprawdzenie katalogu domowego w Linuksie.
    elif [ -d "/home/$USER" ]
    then
        HOME="/home/$USER"

    else
        echo -n "Podaj swój katalog domowy: "
        read HOME
        echo
    fi
fi
```

```
# Wypisanie wartości kontrolowanych zmiennych.
```

```
echo "DISPLAY=$DISPLAY"
echo "SHELL=$SHELL"
echo "USER=$USER"
echo "HOME=$HOME"
```

Skrypt powinien zachowywać się jak poniżej:

```
$ sh check_env
DISPLAY=:0.0
SHELL=/bin/bash
USER=ericfj
HOME=/home2/ericfj
```

Gdyby część ze zmiennych środowiskowych, do których odwołuje się skrypt, była nieustawiona, skrypt będzie zachowywał się nieco inaczej. Oto przykładowe uruchomienie skryptu w systemie Mac OS X:

```
$ sh check_env
Brak zmiennej DISPLAY. przyjmuję :0.0 jako wartość domyślną.
DISPLAY=:0.0
SHELL=/bin/bash
USER=ericfj
HOME=/home2/ericfj
```

Jak widać, w systemie Mac OS X serwer X Window System nie jest domyślnie uruchamiany.

Jak to działa?

W skrypcie `check_env` stosowana jest ogólna strategia polegająca na ustalaniu zdroworozsądkowych wartości dla nieobecnych zmiennych środowiskowych. Zaś w przypadku tych zmiennych, dla których nie można „odgadnąć” wartości, skrypt pyta o nie użytkownika.

W obliczu braku wartości zmiennej skrypt próbuje samodzielnie wytypować brakujące dane. Kiedy nie ma wartości dla zmiennej `HOME`, skrypt sprawdza istnienie katalogu `/Users/nazwa_konta`, który w Mac OS X jest typowo katalogiem domowym użytkownika konta `nazwa_konta`. Sprawdza też istnienie katalogu `/home/nazwa_konta`, charakterystycznego dla systemów uniksowych i Linuksa. Jeśli zaś nie uda się potwierdzić istnienia żadnego z tych katalogów, trzeba uciec się do odpytania użytkownika.

Przy okazji testowania skrypt ustawia wartości zmiennych środowiskowych. Ustawia np. zmienną `DISPLAY` na wartość domyślną `:0.0`. Takie ustawienie powinno być odpowiednie dla wielu systemów uniksowych i linuksowych, ale nie dla Mac OS X, gdzie domyślnie nie korzysta się z serwera X Window System.

Skrypt `check_env` w razie potrzeby samodzielnie ustawi zmienną `DISPLAY`. Zauważ, że ustawienie nie jest uzupełnione eksportem zmiennej, przez co jest widoczne jedynie w obrębie bieżącego skryptu. Dotyczy to również pozostałych testów i ustawień podejmowanych w skrypcie.

Test wartości zmiennej środowiskowej SHELL kończy się (w przypadku braku wartości) ustawieniem na wartość `/bin/bash` i wystosowaniem komunikatu informującego użytkownika o takim wyborze.

W obliczu braku zmiennej USER skrypt pyta użytkownika o nazwę jego konta. Można by co prawda podejmować próby odgadnięcia tej nazwy, ale jest to dość trudne — znacznie łatwiej zapytać o to samego zainteresowanego.

Test zmiennej środowiskowej HOME wymaga wcześniejszego ustalenia wartości zmiennej USER. Jeśli HOME nie ma wartości, skrypt próbuje ustalić dla niej wartość typową dla systemu Mac OS X, a jeśli to się nie uda, ustala wartość typową dla pozostałych systemów uniksowych i uniksopodobnych.

Jeśli żaden z testowanych katalogów domowych nie istnieje, skrypt odwołuje się do instancji ostatecznej — użytkownika.

Na końcu skryptu następuje wypisanie wartości wszystkich czterech zmiennych środowiskowych.

Działanie tych części skryptu, które ustalają domyślne (pożądane) wartości nieobecnych zmiennych, można zweryfikować, usuwając oznaczenie komentarza z poleceń `unset`. Polecenie `unset` usuwa zmienną, zmuszając skrypt do jej samodzielnego ustawienia z wykorzystaniem zaprogramowanych strategii.

Uaktywnienie wiersza oznaczonego symbolem komentarza sprowadza się do usunięcia znaku komentarza (#) z początku wiersza.

To często stosowana technika testowania działania poszczególnych części skryptów.

Ustawianie zmiennych środowiskowych

Zmienne środowiskowe odczytuje się tak samo jak wszelkie zmienne powłoki. Ustawia się je również podobnie. Ale aby ustawienia te trwale osadzić w środowisku, trzeba wykonać dodatkowe czynności.

Trzeba pamiętać, że owo środowisko obejmuje swym wpływem jedynie programy i skrypty wywoływane z wnętrza danego skryptu, czyli jedynie dla powłok i programów potomnych. Ustawienia pierwotne można modyfikować za pośrednictwem plików wymienianych w podrozdziale „Dostosowywanie własnego konta”.

Zmienne środowiskowe ustawia się z dwóch powodów:

- celem dostosowania środowiska pracy do własnych upodobań i potrzeb;
- celem ustawienia zmiennej środowiskowej na potrzeby skryptu, który się do tej zmiennej odwołuje.

Ustawienie zmiennej środowiskowej wymaga operacji eksportowania. Samo ustawienie zmiennej odbywa się przy użyciu zwykłej składni przypisania wartości do zmiennej:

```
ZMIENNA=WARTOŚĆ
```

Eksportowanie zmiennej do środowiska tak, aby jej wartość była widoczna dla procesów potomnych — czyli konwersja zwykłej zmiennej powłoki do postaci pełnoprawnej zmiennej środowiskowej — wymaga wywołania polecenia `export`:

```
export ZMIENNA
```

Stąd w wielu skryptach powłoki da się zauważyć następujące konstrukcje:

```
var=value
export var
```

Oba polecenia można łączyć do postaci pojedynczego wiersza polecenia — wystarczy skorzystać ze średnika:

```
var=value; export var
```

Można też skorzystać ze skróconego zapisu, angażującego pojedyncze polecenie `export`:

```
export var=value
```

Polecenie `export` może eksportować więcej niż jedną zmienną, jak tutaj:

```
var1=value1
var2=value2
var3=value3
export var1, var2, var3
```

W rozmaitych skryptach występują wszystkie przedstawione formy; dotyczy to zwłaszcza skryptów inicjalizacyjnych powłoki.

W powłoce `bash` można jednym poleceniem (`set -a`) dokonać eksportu do środowiska wszystkich ustawionych zmiennych powłoki:

```
$ set -a
```

W skryptach nie należy jednak polegać na tym, że użytkownik wywoła to polecenie, i należałoby samodzielnie eksportować potrzebne zmienne środowiskowe.

Działanie polecenia `export` i sposób tworzenia własnych zmiennych środowiskowych z użyciem tego polecenia przećwiczymy na kilku kolejnych skryptach przykładowych.

spróbuj sam Eksportowanie zmiennych

Poniższy wiersz zapiszmy w pliku o nazwie `echo_myvar`:

```
echo "w skrypcie potomnym MY_VAR=$MY_VAR"
```

Skrypt ogranicza się do wypisania wartości zmiennej powłoki o nazwie `MY_VAR`. Skrypt `echo_myvar` nie ustawia jednak zmiennej, polegając na bieżącym stanie środowiska. Ustawienie powinno odbyć się w ramach skryptu `set_myvar`, o następującej treści:

```
# Ustawienie my_var bez eksportowania.
MY_VAR="Tempest"
```

```
echo -n "Bez eksportu: "  
sh echo_myvar  
  
# Teraz eksport i ponowna próba.  
  
echo -n "Po eksporcie: "  
export MY_VAR  
  
sh echo_myvar
```

Test działania obu skryptów należy rozpocząć od uruchomienia skryptu `set_myvar`. Powinno to dać następujący efekt:

```
$ sh set_myvar  
Bez eksportu: w skrypcie potomnym MY_VAR=  
Po eksporcie: w skrypcie potomnym MY_VAR=Tempest
```

Jak to działa?

Skrypt `echo_myvar` ogranicza się do wypisania wartości zmiennej `MY_VAR`. Skrypt nie ustawia wcześniej jej wartości i polega całkowicie na zastanym stanie środowiska. Jeśli środowisko nie obejmuje tej zmiennej, `echo_myvar` wypisze pusty ciąg wartości zmiennej. W przeciwnym razie `echo_myvar` wypisze właściwą wartość zmiennej.

Z kolei skrypt `set_myvar` ustawia zmienną `MY_VAR` i wywołuje skrypt `echo_myvar`. Jednak za pierwszym razem ten ostatni wypisuje wartość pustą. To dlatego, że skrypt `set_myvar` nie utrwalił zmiennej w środowisku — póki co jedynie ustawił zmienną lokalną względem bieżącej (swojej) powłoki.

Za drugim razem skrypt `set_myvar` dokonuje eksportu zmiennej `MY_VAR`. Teraz wywołany skrypt `echo_myvar` wypisuje właściwą wartość zmiennej.

Ustawianie zmiennych środowiskowych w powłoce C

Prezentowane dotąd przykłady ustawiania zmiennych środowiskowych dotyczyły powłoki Bourne'a i zgodnych z nią w tym zakresie powłok `ksh` i `bash`. W powłoce C tradycyjnie już stosuje się odmienną składnię utrwalania zmiennych w środowisku. Tu służy do tego polecenie wbudowane `setenv`. Wywołuje się je następująco:

```
setenv zmienna wartość
```

Zauważ brak znaku równości, charakterystycznego dla przypisania wartości do zmiennej.

Znakomitym źródłem przykładów obsługi zmiennych środowiskowych w powłoce C są skrypty `/etc/csh.login` i `/etc/csh.cshrc` — to systemowe pliki inicjalizacyjne dla powłoki C (i `tcsh`). Będzie o nich jeszcze mowa w podrozdziale „Dostosowywanie własnego konta”. Oto przykład zaczerpnięty z tych plików:

```
setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin"  
setenv MAIL "/var/spool/mail/$USER"
```

W `csh` i `tcsh` nie stosuje się osobnego wywołania `export`.

Wczytywanie wartości do bieżącego środowiska

Przy uruchamianiu skryptu bieżąca powłoka, np. bash, uruchamia powłokę potomną (najczęściej wywołując polecenie `sh`), która ma wykonać kod skryptu. Wywołania powłoki potomnej można uniknąć, uruchamiając skrypt w środowisku bieżącym. Służy do tego polecenie `source`.

Składnia polecenia prezentuje się następująco:

```
source nazwa_skryptu
```

Polecenie `source` uruchamia wskazany skrypt w kontekście bieżącej powłoki, nie angażując powłoki potomnej. Polecenie to jest zwykle wykorzystywane do wczytywania plików startowych powłoki.

Polecenie `source` jest też dostępne w postaci alternatywnej — w postaci kropki:

```
. nazwa_skryptu
```

Efekt powinien być identyczny jak przy wywołaniu polecenia `source`.

Dostosowywanie własnego konta

Uruchamiana powłoka odczytuje zawartość pliku inicjalizacyjnego (plików inicjalizacyjnych), definiującego pierwotne środowisko. Pliki te są same w sobie skryptami powłoki. Niektóre z nich, jak `/etc/profile`, przechowywane są w katalogach systemowych. Większość użytkowników nie ma możliwości ich modyfikowania. Pozostałe jednak znajdują się w indywidualnych katalogach domowych poszczególnych użytkowników. Te można modyfikować zgodnie z własnymi potrzebami.

Wszelkie pliki systemowe należy zazwyczaj pozostawić nietknięte, ponieważ ustalają one wartości domyślne właściwe dla całego systemu. W ustawienia te mogą ingerować administratorzy systemu.

Natomiast pliki przechowywane w katalogach domowych użytkowników służą właśnie do dostosowywania środowiska pracy do indywidualnych upodobań i potrzeb. Można je uzupełniać własnymi poleceniami (trzymając się składni powłoki).

Każda powłoka wyróżnia zestaw plików, w których szuka ustawień inicjalizacyjnych. Niemal wszystkie tego rodzaju pliki są przechowywane w katalogu domowym; nazwy wszystkich, z jednym wyjątkiem, zaczynają się od kropki.

Pliki o nazwach zaczynających się od kropki to pliki ukryte. Pliki takie nie są ujmowane w wypisie zawartości katalogu, o ile polecenia generującego ten wypis nie uzupełni się specjalną opcją; również symbole wieloznaczne, jak ``, nie obejmują plików o takich nazwach.*

Pliki inicjalizacyjne muszą mieć odpowiednie nazwy, inaczej powłoka nie zdoła ich odnaleźć i wczytać. Na przykład w powłoce bash jednym z plików inicjalizacyjnych jest `.bashrc`. Plik

o dokładnie takiej nazwie powinien znajdować się w katalogu domowym użytkownika; do tego użytkownik musi mieć prawo odczytu tego pliku.

Naleciałości historyczne powodują, że większość powłok korzysta dziś z kilku plików inicjalizacyjnych. Ich funkcje w znacznej mierze się pokrywają, przez co dostosowania powłoki bash można dokonywać zarówno w pliku `.bash_profile`, jak i `.bash_login`.

Nie próbuj ustawiać zmiennych środowiskowych, jeśli nie znasz ich znaczenia i wpływu na działanie programów. Takie próby mogą zachwiać stabilnością aplikacji, które oczekują konkretnych wartości poszczególnych elementów środowiska.

Procedury inicjalizacyjne poszczególnych powłok omówimy sobie w kolejnych punktach.

Rozruch powłoki Bourne'a

Jeśli powłoką logowania jest powłoka Bourne'a, jej inicjalizacją steruje między innymi plik przechowywany w katalogu domowym użytkownika, pod nazwą `.profile`. Powłoka wczytuje polecenia zawarte w tym pliku, wciągając tym samym zawarte tam ustawienia do swojego środowiska (i środowiska wszystkich powłok potomnych).

Powłoka Bourne'a podejmuje taką inicjalizację tylko wtedy, kiedy zostanie uruchomiona jako powłoka logowania, to znaczy uruchomiona dla użytkownika który właśnie zalogował się do systemu. Przy uruchamianiu powłok potomnych plik `.profile` jest ignorowany.

Rozróżnienie pomiędzy powłoką logowania a pozostałymi powłokami jest istotne właśnie ze względu na sposób inicjalizacji powłoki.

Rozruch powłoki Korn

Kiedy w roli powłoki logowania występuje powłoka Korn, ustawienia pierwotne należy zapisywać w pliku `.profile` przechowywanym w katalogu domowym. W tym aspekcie rozruch przebiega podobnie, jak w powłoce Bourne'a.

Jeśli `ksh` nie jest uruchamiana jako powłoka logowania, następuje odczyt kodu z pliku wskazywanego zmienną środowiskową `ENV`. Plik ten pełni rolę analogiczną do roli pliku `.cshrc` w powłoce C.

Jeśli zmienna `ENV` wskazuje nie konkretny plik, a cały katalog, powłoka szuka w nim pliku o nazwie `ksh_env`. Jeśli zaś `ENV` wskazuje konkretny plik, powłoka podejmuje wczytywanie tego pliku i uruchomienie zawartych w nim poleceń w kontekście bieżącej powłoki.

Rozruch powłoki C

Kiedy powłoka C uruchamia się jako powłoka logowania, podejmuje odczyt poleceń z plików `/etc/csh.cshrc` i `/etc/csh.cshlogin`. Następnie podejmuje poszukiwanie pliku o nazwie `.cshrc` w katalogu domowym użytkownika i wczytuje i uruchamia polecenia zapisane w tymże pliku.

Do tego wszystkiego rozruch jest uzupełniany odczytem i wykonaniem poleceń z pliku *.login* z katalogu domowego użytkownika. Z kolei przy wylogowywaniu użytkownika powłoka logowania odczytuje i uruchamia zawartość pliku */etc/csh.logout* i pliku *.logout* z katalogu domowego użytkownika.

Kiedy C startuje jako powłoka potomna (nie powłoka logowania), odczytuje polecenia z pliku */etc/csh.cshrc* i pliku *.cshrc* z katalogu domowego użytkownika.

Rozruch powłoki T C

Rozruch powłoki T C przebiega podobnie jak rozruch powłoki C. Różnica polega na tym, że pliki startowe mogą mieć nazwę *.cshrc* albo *.tcshrc*. Powłoka T C rozpoznaje plik *.cshrc* i uwzględni zawarte w nim polecenia, co ma ułatwić migrację do tej powłoki z powłoki C.

Rozruch powłoki bash

Bash przy uruchamianiu wykonuje czynności charakterystyczne dla powłok C, Bourne'a i Korn'a, korzystając również z podobnych konwencji nazewnictwa plików startowych.

Jeśli powłoka bash startuje jako powłoka logowania, w pierwszej kolejności odczytuje i uruchamia polecenia z pliku */etc/profile*. Następnie podejmuje próby odszukania i uruchomienia szeregu plików z katalogu domowego użytkownika:

- *.bash_profile*
- *.bash_login*
- *.profile*

Pliki są wyszukiwane zgodnie z podaną tu kolejnością. Uruchamiany jest zaś pierwszy znaleziony plik. Ma to ułatwić przenosiny z powłoki ksh czy csh.

Przy zamykaniu powłoki logowania bash szuka w katalogu domowym użytkownika pliku o nazwie *.bash_logout* i wykonuje zapisane w nim polecenia.

Uruchomienie powłoki bash jako powłoki logowania polega na wywołaniu jej z opcją `--login`, jak tutaj:

```
$ bash --login
```

Aby pominąć odczyt plików startowych, należy wywołać powłokę bash z opcją `-noprofile`.

Kiedy powłoka bash jest uruchamiana jako powłoka potomna, jej rozruch polega na przeszukaniu jednej z dwóch lokalacji systemu plików. Otóż jeśli powłoka startuje w trybie interaktywnym, szuka i uruchamia plik *.bashrc* z katalogu domowego użytkownika. W innych przypadkach rozruchem steruje plik wskazywany przez zmienną środowiskową `BASH_ENV`. Zmienna ta odgrywa w procesie rozruchu rolę analogiczną do roli zmiennej `ENV` w powłoce ksh.

Opcja `--norc` wymusza na powłoce `bash` zaniechanie uruchamiania pliku `.bashrc`. Z kolei opcja `--rcfile` służy do wskazywania powłoce nieinteraktywnej (i niepełniającej roli powłoki logowania) pliku startowego innego niż plik `.bashrc` z katalogu domowego użytkownika.

Pliki sterujące rozruchem poszczególnych powłok zostały wymienione w tabeli 4.2.

Tabela 4.2. Pliki inicjalizacji (rozruchu) wybranych powłok

Powłoka	Rozruch	Logowanie	Wylogowanie
bash	<code>.bashrc</code> (dla powłoki potomnej, interaktywnej), <code>\$BASH_ENV</code> (dla powłoki potomnej, nieinteraktywnej)	<code>/etc/profile</code> , potem <code>.bash_profile</code> ewentualnie <code>.bash_login</code> , ewentualnie <code>.profile</code>	<code>.bash_logout</code>
csch	<code>/etc/csh.cshrc</code> a potem <code>.cshrc</code>	<code>/etc/csh.cshrc</code> , potem <code>/etc/csh.login</code> , potem <code>.cshrc</code> , potem <code>.login</code>	<code>/etc/csh.logout</code> , potem <code>.logout</code>
ksh	<code>\$ENV</code>	<code>.profile</code>	Brak
sh	Brak	<code>/etc/profile</code> , potem <code>.profile</code>	Brak
tcsh	<code>/etc/csh.cshrc</code> oraz <code>.tcshrc</code> bądź <code>.cshrc</code>	<code>/etc/csh.cshrc</code> , potem <code>/etc/csh.login</code> , potem <code>.tcshrc</code> , ewentualnie <code>.cshrc</code> , potem <code>.login</code>	<code>/etc/csh.logout</code> , potem <code>.logout</code>

Nie należy ślepo i całkowicie polegać na możliwościach dostosowywania powłoki do własnych upodobań. To co prawda świetna sprawa, ale nie wolno pisać skryptów, opierając się na indywidualnych ustawieniach środowiska. Taki skrypt, przeniesiony do innego systemu albo choćby uruchomiony z konta innego użytkownika, może działać niezgodnie z oczekiwaniami.

Ponieważ inicjalizacja powłoki i jej środowiska następuje w ramach jej procedury rozruchowej, nie zawsze wiadomo, co powoduje rzeczony problem. Niekiedy dopiero po wielu godzinach bezskutecznych poszukiwań wychodzi na jaw, że sam skrypt jest zupełnie w porządku, a problem tkwi w nietypowych ustawieniach wczytanych z plików startowych powłoki.

Obsługa argumentów wiersza poleceń

Kolejnym obszarem interakcji skryptów z otoczeniem jest wiersz poleceń. Za jego pośrednictwem możemy przekazywać rozmaite opcje i argumenty nie tylko do poleceń, ale i do skryptów powłoki. Oczywiście samo przekazanie zestawu opcji i argumentów do wywołanego polecenia tudzież skryptu to ta prosta część zadania. Trudniej jest obsłużyć tak otrzymane wartości we wnętrzu skryptu.

W pierwszej kolejności należy kolejno wszystkie przekazane elementy wiersza wywołania poddać stosownej analizie.

Wczytywanie argumentów wywołania w powłoce Bourne'a

Kiedy powłoka Bourne'a uruchamia skrypt, przekazane do tego skryptu argumenty i opcje umieszcza w specjalnych zmiennych powłoki, do których skrypt może się swobodnie odwoływać. Na przykład zmienna \$1 przechowuje pierwszy element wiersza wywołania (pierwszy za nazwą skryptu). Może to być opcja (np. -v) albo argument, na przykład nazwa pliku do przetworzenia — cokolwiek to jest, skrypt może się do tego odwołać za pośrednictwem \$1.

Odwzorowanie kolejności elementów wiersza wywołania w nazwach zmiennych (jak \$1) zostało zaznaczone w nazwach tych zmiennych, określanych mianem parametrów pozycyjnych.

Listę zmiennych specjalnych, związanych z obsługą elementów wiersza polecenia w powłoce Bourne'a wymienia tabela 4.3.

Tabela 4.3. Zmienne specjalne powłoki Bourne'a

Zmienna	Wartość
\$0	Nazwa skryptu podana w wierszu polecenia.
\$1	Pierwszy element wiersza polecenia.
\$2	Drugi element wiersza polecenia.
\$3	Trzeci element wiersza polecenia.
\$4	Czwarty element wiersza polecenia.
\$5	Piąty element wiersza polecenia.
\$6	Szósty element wiersza polecenia.
\$7	Siódmy element wiersza polecenia.
\$8	Ósmy element wiersza polecenia.
\$9	Dziewiąty element wiersza polecenia.
\$#	Liczba wyodrębnionych elementów wywołania.
\$*	Komplet elementów wywołania w kolejności zgodnej z kolejnością w wierszu polecenia (elementy są tu oddzielone spacjami).

Nazwa skryptu, pod jaką został wywołany, zapisywana jest w parametrze pozycyjnym \$0.

Kolejne elementy wywołania są wyodrębniane do kolejnych parametrów pozycyjnych, od \$1 do \$9. Nie można wprost odwołać się do elementów o numerach większych od 9 (np. \$10). Nie oznacza to jednak, że w wywołaniu można przekazywać tylko dziewięć opcji czy argumentów. Pozostałe argumenty i opcje wywołania można wyodrębnić z ciągu \$*, zawierającego komplet elementów wywołania. Liczbę tych elementów można odczytać z \$#.

Odwołania do parametrów pozycyjnych o wyższych numerach możliwe są w powłoce Korn, która rozpoznaje i obsługuje odwołania postaci \${10} i tak dalej.

\$0 nie wchodzi w skład ciągu \$.*

spróbuj sam Wypisywanie argumentów wywołania

Poniższy skrypt wypisuje na wyjściu przekazane doń argumenty wywołania:

```
# Badanie argumentów wywołania.

echo "Skrypt: $0"

echo "Liczba elementów wywołania: $#"

```

Zapiszmy go pod nazwą `args`. Po uruchomieniu skryptu generowane przez niego wyjście będzie zależać od przekazanych do skryptu argumentów, jak tu:

```
$ sh args arg1 arg2
Skrypt: args
Liczba elementów wywołania: 2
1-szy argument: arg1
2-gi argument: arg2
Komplet argumentów: [arg1 arg2]
```

Spróbujmy uruchomić skrypt z nieco większą liczbą argumentów:

```
$ sh args arg1 2 3 4 5 6 7 8 9 10
Skrypt: args
Liczba elementów wywołania: 10
1-szy argument: arg1
2-gi argument: 2
Komplet argumentów: [arg1 2 3 4 5 6 7 8 9 10]
```

Jak to działa?

Skrypt `args` na początku wypisuje wartość `$0`, czyli wartość zmiennej przechowującej nazwę skryptu występującą w wywołaniu. Potem następuje wypisanie liczby wyodrębnionych argumentów wywołania, a następnie wartości dwóch pierwszych argumentów (w osobnych wierszach). Wreszcie w ostatnim komunikacie skrypt wypisuje w nawiasie prostokątnym kompletną listę argumentów wywołania.

Zasada działania skryptu zdaje się nieskomplikowana, ale nie wszystko jest tak do końca oczywiste — można się o tym przekonać, uruchamiając skrypt `args` z odpowiednio spreparowanymi zestawami argumentów. Pierwsza próba może polegać na wywołaniu bez jakiegokolwiek argumentu:

```
$ sh args
Skrypt: args
Liczba elementów wywołania: 0
1-szy argument:
2-gi argument:
Komplet argumentów: []
```

Mamy tu zerową liczbę argumentów i szereg odwołań do wartości pustych.

Jeśli skrypt zostanie wywołany z jednym argumentem, wyjście powinno prezentować się tak:

```
$ sh args arg1
Skrypt: args
Liczba elementów wywołania: 1
1-szy argument: arg1
2-gi argument:
Komplet argumentów: [arg1]
```

Wartość przechowywana jako \$0 to nazwa skryptu widniejąca w wierszu polecenia. Stąd w \$0 może zamiast właściwej nazwy pliku skryptu pojawić się równie dobrze pełna bądź względna ścieżka dostępu do pliku skryptu, albo jeszcze inny wariant nazwy skryptu:

```
$ sh /home/ericfj/beginning_shell_scripting/scripts/args arg1 arg2
Skrypt: /home/ericfj/beginning_shell_scripting/scripts/args
Liczba elementów wywołania: 2
1-szy argument: arg1
2-gi argument: arg2
Komplet argumentów: [arg1 arg2]
```

Tu w wywołaniu skryptu został on zidentyfikowany przez bezwzględną ścieżkę dostępu i w takiej też postaci jego nazwa występuje we wnętrzu skryptu, widoczna jako \$0.

W wywołaniu mogą też wystąpić argumenty puste, ale liczone jako osobne argumenty wywołania:

```
$ sh args arg1 "" " " " arg4
Skrypt: args
Liczba elementów wywołania: 4
1-szy argument: arg1
2-gi argument:
Komplet argumentów: [arg1 " " " arg4]
```

Pierwszym argumentem wywołania jest tu niewyróżniający się niczym ciąg `arg1`. Drugi argument jest jednak ciągiem pustym (`""`). Trzeci argument to z kolei ciąg niepusty, ale zawierający wyłącznie znaki spacji; wreszcie czwarty argument znów jest najzwyczajniejszym ciągiem, niepustym i złożonym z widocznych znaków — `arg4`.

Kiedy w skrypcie następuje odwołanie `$*`, ujęte pomiędzy znakami cudzysłowu (jak w skrypcie `args`, gdzie odwołanie `$*` zostało osadzone w ciągu tekstowym), powstaje ciąg obejmujący również spacje składające się na ciąg argumentu trzeciego (`$3`).

Ponieważ spacja pełni rolę separatora elementów w ciągu `$*`, a argumenty mogą zawierać spacje albo nawet w całości składać się tylko ze spacji, odwołania mogą być niekiedy mylące:

```
$ sh args "1 2 3 4 5 6 7 8 9"
Skrypt: args
Liczba elementów wywołania: 1
1-szy argument: 1 2 3 4 5 6 7 8 9
2-gi argument:
Komplet argumentów: [1 2 3 4 5 6 7 8 9]
```

Do skryptu przekazaliśmy jeden tylko argument ("1 2 3 4 5 6 7 8 9"), ale z racji spacji pomiędzy znakami w ciągu argumentu, kiedy skrypt `args` wypisuje wartość `$*`, można omyłkowo uznać, że do skryptu przekazano aż 9 argumentów.

Jak widać, przy manipulowaniu argumentami wywołania skryptu należy zachować pewną ostrożność.

Obsługa argumentów wiersza polecenia jest szczególnie uciążliwa w systemach Windows i Mac OS X, gdzie często stosuje się nazwy katalogów zawierających spacje (jak choćby w nazwie `C:\Program Files`).

Jak dotąd ograniczaliśmy obsługę argumentów wywołania do wypisywania ich na wyjściu skryptu. Spróbujmy teraz zrobić z nich lepszy użytek.

spróbuj sam Używanie argumentów wywołania

Skryptem `myls`, prezentowanym w rozdziale 3., naśladowaliśmy działanie polecenia `ls`. Skrypt ten możemy teraz rozbudować o możliwość przekazywania w wywołaniu nazwy katalogu do przejrzania. Zapiszmy nową wersję skryptu pod nazwą `myls3`:

```
# Zakładamy, że $1 (pierwszy argument wywołania)
# wskazuje katalog do przeszukania.

cd $1
for filename in *
do
    echo $filename
done
```

Po uruchomieniu skryptu będzie można się przekonać, że zawartość wypisywanej na wyjściu listy nazw plików jest uzależniona od argumentu wywołania, określającego nazwę katalogu:

```
$ sh myls3 /usr/local
bin
etc
games
include
lib
libexec
man
sbin
share
src
```

Jak to działa?

Skrypt `myls3` uzupełnia pierwowzór z rozdziału 3. o obsługę wiersza polecenia — pierwszy argument wywołania jest tu uznawany za nazwę katalogu, w którym należy wyszukać pliki do wypisania. Realizacja postulatu wyrażonego wywołaniem odbywa się przez przejście do wskazanego katalogu poleceniem `cd` i podjęcie zwykłej pętli wypisującej nazwy plików z bieżącego katalogu.

W ten sposób można za pośrednictwem argumentów wywołania wskazywać katalogi i pliki wyznaczone do przetwarzania w skrypcie powłoki.

Wczytywanie argumentów wywołania w powłocie C

W powłokach C i T C do obsługi argumentów wywołania również stosuje się specjalne zmienne, podobnie jak w powłokach sh, bash i ksh. Tyle że w powłokach csh i tcsh w miejsce \$# stosuje się odwołanie \$#argv.

W csh i tcsh specjalna zmienna \$#argv przechowuje liczbę argumentów wywołania skryptu.

Trzeba zaznaczyć, że tcsh rozpoznaje też odwołanie postaci \$# i można tam stosować zarówno zapis \$#, jak i \$#argv.

Usamodzielnianie skryptów powłoki

Jak dotąd uruchamianie skryptów wymagało od nas jawnego wywołania powłoki (sh) z nazwą skryptu przeznaczonego do uruchomienia. Nie przypomina to bynajmniej wywołań zwykłych poleceń, jak choćby ls. W przypadku samodzielnych poleceń wywołanie sprowadza się do podania nazwy polecenia.

Rzecz jasna istnieje sposób, który pozwala zamienić nasz niepełnowartościowy w tym sensie skrypt na pełnowartościowe polecenie wykonywalne — tak, aby użytkownicy wywoływali go, wpisując jego nazwę w wierszu polecenia, i nawet nie domyślali się, że to skrypt powłoki, a nie program wykonywalny.

Transformacja owa składa się z dwóch etapów:

- oznaczenia pliku skryptu jako pliku wykonywalnego;
- uzupełnienia kodu skryptu o specjalny wiersz identyfikujący powłokę mającą uruchomić skrypt.

Etapom tym przyjrzymy się w osobnych punktach.

Nadawanie skryptowi atrybutu wykonywalności

W systemach uniksowych i uniksopodobnych wszystkie skrypty powłoki, polecenia, programy binarne i w ogóle wszystkie pliki, które da się uruchomić, powinny być oznaczone jako *pliki wykonywalne*. Plik wykonywalny to plik posiadający atrybut uprawniający użytkownika do jego uruchamiania. Uprawnienia pliku modyfikuje się poleceniem chmod.

Zanim przystąpisz do zmiany zestawu uprawnień, powinieneś jednak sprawdzić, jakie uprawnienia definiuje plik obecnie. Wykaz uprawnień można wypisać poleceniem `ls`, wywołanym z opcją `-l`, jak tu:

```
$ ls -l myls3
-rw-rw-r-- 1 ericfj engineering 124 Oct 12 22:39 myls3
```

Każda trójka symboli (np. `rw-`) reprezentuje uprawnienia pewnej kategorii użytkowników. Pierwsza dotyczy użytkownika będącego właścicielem pliku (tu jest nim `ericfj` i dotyczą go uprawnienia `rw-`), druga obejmuje użytkowników grupy skojarzonej z plikiem (tu `engineering`, z uprawnieniami `rw-`), trzecia zaś dotyczy całej reszty użytkowników (tu mają oni uprawnienia `r--`). Symbol `r` w tej trójce oznacza możliwość odczytu pliku. Symbol `w` oznacza uprawnienia do zapisywania (modyfikacji) pliku. A myślnik oznacza brak uprawnień.

Trójka `rw-` oznacza więc uprawnienia do odczytu i zapisu pliku, a `r--` to uprawnienia wyłącznie do odczytu. Jak widać, ten plik nie definiuje w ogóle uprawnień do uruchamiania.

Dodanie takich uprawnień wymaga wywołania polecenia `chmod`. Oto przykład:

```
$ chmod u+x myls3
```

Argument w postaci `u+x` to jeden z możliwych sposobów wyrażenia chęci modyfikacji zestawu uprawnień. Znak `u` oznacza tu użytkownika, a konkretnie właściciela pliku. Znak `+` oznacza uzupełnienie zestawu uprawnień, a `x` to symbol uprawnienia do uruchamiania.

*Uprawnienia można też podawać w specjalnym zapisie ósemkowym, np. 0666.
Po szczegóły odsyłamy do dokumentacji systemowej dla polecenia `chmod`.*

Sprawdźmy poleceniem `ls`, czy faktycznie skrypt zyskał uprawnienie do uruchamiania:

```
$ ls -l myls3
-rwxrw-r-- 1 ericfj engineering 124 Oct 12 22:39 myls3
```

Jak widać, właściciel pliku ma teraz komplet uprawnień (`rwx`), a więc prawo do odczytywania, zapisywania i uruchamiania pliku. Plik skryptu stał się niniejszym plikiem wykonywalnym (choć na razie tylko dla jego właściciela).

Użytkownicy powłoki `csh` czy też `tsh` powinni po takiej zmianie uprawnień wywołać polecenie `rehash`, odświeżające wewnętrzną listę plików wykonywalnych:

```
$ rehash
```

Po oznaczeniu skryptu atrybutem wykonywalności należy jeszcze odpowiedzieć powłoce (i wszelkim innym powłokom), w *jaki sposób* ma dokonać uruchomienia skryptu.

Magiczny wiersz #!

Pliki wykonywalne — czyli, ogólnie mówiąc, polecenia — pochodzą z wielu źródeł i mają najróżniejsze postaci. Większość poleceń to programy skompilowane, binarne. Są one pisane w języku programowania C i potem kompilowane do postaci wykonywalnej, odpowiedniej dla procesora właściwego dla danej platformy.

Polecenia można by z grubsza podzielić na:

- skompilowane, wykonywalne (bezpośrednio) programy binarne,
- archiwa wykonywalne języka Java (plik *.jar*),
- skrypty w językach takich jak Python, Perl, Tcl, Ruby czy Lua,
- skrypty powłoki.

Przy próbie uruchomienia pliku wykonywalnego powłoka przede wszystkim próbuje określić typ pliku i na jego podstawie zastosować odpowiednią metodę wywołania. Jeśli na przykład plik jest skompilowanym, wykonywalnym plikiem binarnym, powłoka odczytuje z niego pierwszych kilka bajtów, zawierających specjalny kod (znany też jako *liczba magiczna pliku* — od pliku */etc/magic*, zawierającego listę znanych kodów). Ten kod determinuje sposób uruchomienia pliku.

Taka sama procedura dotyczy również skryptów powłoki. Powłoka musi w pierwszej kolejności wykryć, że dany plik wykonywalny jest skryptem. Jest to o tyle proste, że wszelkie skrypty są zwykłymi plikami tekstowymi, więc bajty pliku zawierają kody znaków drukowalnych.

Kiedy już powłoka stwierdzi, że ma do czynienia ze skryptem, powinna znaleźć odpowiedni sposób jego uruchomienia — słowem, wybrać program, który ma zająć się interpretacją i wykonywaniem poleceń zapisanych w skrypcie. Jeśli na przykład w systemie działa powłoka bash, a skrypt jest przeznaczony dla powłoki sh, powłoka bash powinna wybrać i wywołać program powłoki sh i przekazać mu zadanie uruchomienia skryptu.

Przyjęło się, że jeśli skrypt zaczyna się od wiersza ze znakami `#!`, to ów specjalny komentarz wskazuje program interpretera właściwego dla skryptu. Z poniższego wynikałoby więc, że skrypt ma zostać uruchomiony w powłoce sh:

```
#!/bin/sh
```

Trzeba przy tym pamiętać, że ów specjalny wiersz musi być pierwszym wierszem pliku. A znak `#` musi być pierwszym znakiem pierwszego wiersza. Jeśli z jakichkolwiek przyczyn specjalne znaczenie takiego wiersza będzie nieznanne dla powłoki, w której nastąpi wywołanie pliku skryptu, powłoka ta zignoruje wiersz całkowicie — wszak `#` jest poza tym wyjątkiem zapowiedzią komentarza.

Składnia wiersza `#!` prezentuje się następująco:

```
#!/pełna/ścieżka/do/interpretera
```

Interpreter to program podejmujący interpretację poszczególnych poleceń skryptu i uruchamiający te polecenia. Dla skryptów powłoki Bourne'a interpreterem jest `sh` (instalowany jako */bin/sh*). Interpreterami skryptów powłoki są same powłoki. Nie jest to regułą w przypadku innych języków skryptowych — choćby w języku Tcl interpreter może być równocześnie namiastką powłoki (`tclsh`) albo interpreterem z obsługą trybu graficznego (`wish`).

Podsumowując, powłoka obsługująca wywołanie skryptu analizuje wiersz `#!` i wywołuje wymieniony tam program interpretera.

Listę konwencjonalnych lokalizacji instalacji interpreterów i powłok wymienia tabela 4.4.

Tabela 4.4. Typowa lokalizacja plików powłok i interpreterów języków skryptowych

Interpreter (powłoka)	Wiersz #!
ash	#!/bin/ash
bash	#!/bin/bash
csh	#!/bin/csh
ksh	#!/bin/ksh
perl	#!/usr/bin/perl ewentualnie #!/usr/local/bin/perl
python	#!/usr/bin/python ewentualnie #!/usr/local/bin/python
sh	#!/bin/sh
tcsh (Tcl)	#!/usr/bin/tcsh ewentualnie #!/usr/local/bin/tcsh
tcsh	#!/bin/tcsh ewentualnie #!/usr/local/bin/tcsh
wish (Tcl)	#!/usr/bin/wish ewentualnie #!/usr/local/bin/wish
zsh	#!/bin/zsh

Identyczny mechanizm wykorzystywany jest również w skryptach języków skryptowych, takich jak Perl czy Python. Na przykład skrypt języka Perl zaczynałby się od wiersza:

```
#!/usr/bin/perl
```

Istnieje jednak problem w postaci ryzyka niedostępności wskazanego interpretera w danym systemie, albo dostępności w innej lokalizacji. Jeśli dany interpreter nie wchodzi w skład standardowej dystrybucji danego systemu, to jeśli zostanie zainstalowany dodatkowo, będzie znajdować się najprawdopodobniej w katalogu `/usr/local/bin` (albo jeszcze innym), ale nie w `/bin` czy `/usr/bin`. Z kolei w systemie Linux niemal wszystkie istniejące pakiety stanowią część danej dystrybucji (choć niekoniecznie są instalowane domyślnie) i niemal zawsze są instalowane w katalogach `/bin` bądź `/usr/bin`. Stąd ścieżka występująca w wierszu `#!` nie wszędzie wskazuje istniejący program interpretera.

spróbuj sam Usamodzielnianie skryptu

Począwszy od omawianego poprzednio skryptu `mys3` zaczniemy tworzyć skrypty nadające się do samodzielnego wywołania, a to za sprawą wiersza `#!` wskazującego interpreter właściwy dla uruchomienia skryptu. Zapiszmy niniejszym uzupełniony skrypt pod nazwą `mys4`:

```
#!/bin/sh

# Zakładamy, że $1 (pierwszy argument wywołania)
# wskazuje katalog do przeszukania.

cd $1
for filename in *
do
    echo $filename
done
```

Właściwy kod skryptu nie różni się niczym od kodu ze skryptu `mys3`, różnica ogranicza się więc do pierwszego wiersza (który został tu wyróżniony pogrubieniem czcionki). Można więc z powodzeniem skopiować kod ze skryptu `mys3`.

Teraz trzeba jeszcze oznaczyć skrypt atrybutem wykonywalności:

```
$ chmod u+x mys4
```

W powłokach C i T C, po umieszczeniu w którymś z katalogów ścieżki przeszukiwania nowego pliku wykonywalnego, należy jeszcze odświeżyć wewnętrzną listę poleceń wykonywalnych, przechowywaną przez powłokę. Służy do tego polecenie `rehash`:

```
$ rehash
```

Po wywołaniu polecenia `chmod` (i ewentualnie `rehash`) można już uruchamiać skrypt jako samodzielne polecenie w bieżącym katalogu:

```
$ ./mys4 /usr/local  
bin  
etc  
games  
include  
lib  
libexec  
man  
sbin  
share  
src
```

Efekt powinien być taki sam jak po analogicznym uruchomieniu polecenia `mys3`.

Jak to działa?

Opatrzenie skryptu atrybutem wykonywalności i uzupełnienie go o wiersz `#!` pozwala na uruchamianie skryptu jako samodzielnego polecenia. Zapis `./mys4` z poprzedniego przykładu informuje powłokę, aby wyszukała plik wykonywalny `mys4` w katalogu bieżącym.

Jeśli ów katalog bieżący wchodzi w skład ścieżki przeszukiwania, można wywołać skrypt `mys4` jeszcze prościej:

```
$ mys4  
bin  
etc  
games  
include  
lib  
libexec  
man  
sbin  
share  
src
```

Wygodę tę można też osiągnąć, kopiując plik skryptu do jednego z katalogów wchodzących w skład ścieżki przeszukiwania (czyli do jednego z katalogów wymienionych w ciągu zmiennej środowiskowej `PATH`). W ten właśnie sposób można uzupełniać system o własne polecenia.

Podsumowanie

Skrypty powłoki nie są bytami niezależnymi od reszty elementów systemu, a w szczególności muszą działać z uwzględnieniem otoczenia, czyli środowiska. Tym samym pozwalają na odwoływanie się do tego środowiska użytkownikowi.

W tym rozdziale udało się omówić:

- Zmienne środowiskowe, przechowujące ustawienia poszczególnych elementów środowiska systemowego. Do tych ustawień zaliczamy wartości odzwierciedlające rodzaj i typ platformy, położenie katalogów poleceń oraz ustawienia indywidualne dla użytkowników, na przykład wskazania domyślnej powłoki i katalogu domowego użytkownika.
- Zmienne środowiskowe są zwykłymi zmiennymi powłoki. Można się do nich odwoływać za pośrednictwem nazwy uzupełnionej (z przodu) znakiem \$. Ale ustawienie zmiennej środowiskowej, aby na trwałe weszło do środowiska, musi zostać uzupełnione operacją eksportu zmiennej.
- Wszystkie dane umieszczane w wierszu polecenia, w tym opcje i argumenty wywołania skryptu, są w nim dostępne za pośrednictwem specjalnych zmiennych zwanych parametrami pozycyjnymi (\$1, \$2 itd.). Można z nich robić dowolny użytek we wnętrzu skryptu.
- Aby skrypt powłoki stał się samodzielnym poleceniem, trzeba oznaczyć plik skryptu atrybutem wykonywalności i umieścić w pierwszym wierszu skryptu specjalny zapis #!, wskazujący program interpretera właściwy do wykonania skryptu.

W następnym rozdziale zajmiemy się plikami — wszak znakomita większość skryptów powłoki w ten czy inny sposób odwołuje się do plików zewnętrznych.

Zadania

1. Napisz skrypt, który w obliczu braku zmiennej SHELL informuje o tym użytkownika i kończy działanie. Działanie skryptu przetestuj w obu możliwych przypadkach: przy ustawionej i nieustawionej zmiennej SHELL.
2. Napisz skrypt, który przeglądając kolejne argumenty wywołania, będzie je wypisywał na wyjściu. Skrypt ma działać poprawnie niezależnie od liczby argumentów wywołania i wypisywać jedynie te z nich, które mają niepuste wartości. Na przykład argument "", choć przekazany do skryptu, jako pusty nie powinien być wypisywany na wyjściu. Skrypt powinien ponadto wypisywać łączną liczbę argumentów wywołania.
3. Napisz skrypt, który wypisywałby wszystkie przekazane doń argumenty, ale taki, który działałby bez modyfikacji w powłokach bash, ksh, sh i csh (uwaga: najtrudniej uzyskać zgodność z powłoką C).
4. Napisz skrypt, który przyjmując za pośrednictwem argumentów wywołania dowolną liczbę nazw katalogów i wypisuje zawartość każdego z tych katalogów. Wypis zawartości kolejnego katalogu powinien być poprzedzony jego nazwą.