O'REILLY®

Bash

Leksykon kieszonkowy

PRZEWODNIK DLA UŻYTKOWNIKÓW
I ADMINISTRATORÓW SYSTEMÓW

Helion

Arnold Robbins

Tytuł oryginału: Bash Pocket Reference, Second Edition

Tłumaczenie: Patryk Wierzchoń

ISBN: 978-83-283-2820-4

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

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

Printed in Poland.

Spis treści

Zmienne

W tej sekcji opisano:

- przypisywanie zmiennych,
- podstawianie zmiennych,
- zmienne pośrednie,
- wbudowane zmienne powłoki,
- pozostałe zmienne powłoki,
- tabele,
- specjalne znaki zachęty.

Przypisywanie zmiennych

Nazwy zmiennych składają się z dowolnej liczby liter, cyfr i podkreślników. Powłoka rozróżnia wielkość znaków i nie pozwala na rozpoczynanie nazw zmiennych od cyfr. Wartości przypisuje się znakiem `=`. **Nie wolno** wstawiać spacji pomiędzy nazwą zmiennej a jej wartością. Możesz przypisać wiele wartości do wielu zmiennych w jednej linii, rozdzielając deklaracje spacjami:

```
imie=Arnold nazwisko=Boczek liczbadzieci=4 liczbazwierzat=1
```

Zgodnie z konwencją nazwy zmiennych wykorzystywanych lub ustawianych przez powłokę zapisuje się wielkimi literami. Możesz jednak wykorzystywać taki zapis w swoich skryptach, jeśli nazwa Twojej zmiennej nie pokrywa się ze specjalną zmienną powłoki.

Domyślnie powłoka traktuje wartości zmiennych jako łańcuchy znaków, nawet jeśli zawierają tylko cyfry. Jednak gdy wartość przypisana jest do zmiennej typu `integer` (stworzonej za pomocą `declare –i`), Bash zinterpretuje przypisaną wartość jako wyrażenie arytmetyczne (zobacz „Wyrażenia arytmetyczne”). Na przykład:

```
$ i=5+3 ; echo $i
5+3
$ declare -i jj ; jj=5+3 ; echo $jj
8
```

Operator += umożliwia dodanie prawej strony deklaracji do istniejącej wartości. Dla zmiennych typu `integer` wyrażenie po prawej stronie deklaracji jest rozpatrywane jako wyrażenie arytmetyczne, które jest obliczane i dodawane do wartości. Natomiast nowe wartości zmiennych tablicowych zostają dopisane do tabel. Na przykład:

```
$ imie=Arnold                Zmienna typu string
$ imie+=" Boczek" ; echo $imie
Arnold Boczek
$ declare -i jj ; jj=3+5     Zmienna typu integer
$ echo $jj
8
$ jj+=2+4 ; echo $jj
14
$ zwierzeta=(czarna ruda)    Zmienna tablicowa
$ echo ${zwierzeta[*]}
czarna ruda
$ zwierzeta+=(chmura zosia)
$ echo ${zwierzeta[*]}
czarna ruda chmura zosia
```

Podstawianie zmiennych

W poniższych wyrażeniach nie wstawia się spacji. Dwukropek (`:`) jest opcjonalny — jeśli został zastosowany, *zmienna* musi być ustawiona i różna od `null`. Formy podstawiania zmiennych uwzględniają wartość opcji `nocasematch`.

W powłokach nieinteraktywnych z włączoną opcją `set -u` stosowanie nieustawionej zmiennej w podstawieniach `#`, `##`, `%`, `%%`, `//`, `/#`, `/%`, `^`, `^^`, `,` i `,,` spowoduje wyjście z powłoki.

Rozważ tekst zacytowany w apostrofach w podstawieniu zmiennej `${zmienna ↪:=a'specjalny-tekst'b}`. W tym przykładzie `'specjalny-tekst'` rozpoznany będzie jako cytowany. Jednak w trybie POSIX, gdzie podstawianie zmiennej odbywa się w cudzysłowie, apostrofy nie będą rozpoznane jako wewnętrzne, zagnieżdżone cytowanie. Istnieją też wyjątki: apostrofy oznaczają cytowanie, jeżeli stosowane są w podstawieniach `#`, `##`, `%`, `%%`, `//`, `/#`, `/%`, `^`, `^^`, `,` i `,,`.

zmienna=wartość	Przypisz *wartość* do *zmiennej*.
`${`*zmienna*`}`	Użyj wartości zmiennej. Klamry są opcjonalne, jeśli *zmienna* jest oddzielona od następującego po niej tekstu, ale są obowiązkowe dla tabel.
`${`*zmienna*`:-`*wartość*`}`	Użyj *zmiennej*, jeśli posiada przypisaną wartość. W przeciwnym razie użyj *wartości*.
`${`*zmienna*`:=`*wartość*`}`	Użyj *zmiennej*, jeśli posiada przypisaną wartość. W przeciwnym razie użyj *wartości* i przypisz ją do *zmiennej*.

`${zmienna:?wartość}`	Użyj *zmiennej*, jeśli posiada przypisaną wartość. W przeciwnym razie wyświetl *wartość* i zakończ (jeśli powłoka nie jest interaktywna). Jeśli nie podano *wartości*, wyświetl `parameter null or not set` do `stderr`.
`${zmienna:+wartość}`	Użyj *wartości*, jeśli *zmienna* ma przypisaną wartość. W przeciwnym razie nie rób nic.
`${#zmienna}`	Użyj długości *zmiennej*.
`${#*},${#@}`	Użyj liczby parametrów.
`${zmienna#wzorzec}`	Użyj wartości *zmiennej* po usunięciu z niej tekstu z lewej pasującego do *wzorca*. Usuwa najkrótszy pasujący tekst.
`${zmienna##wzorzec}`	Tak jak *#wzorzec*, ale usuwa najdłuższy pasujący tekst.
`${zmienna%wzorzec}`	Użyj wartości *zmiennej* po usunięciu z niej tekstu z prawej pasującego do *wzorca*. Usuwa najkrótszy pasujący tekst.
`${zmienna%%wzorzec}`	Tak jak *%wzorzec*, ale usuwa najdłuższy pasujący tekst.
`${zmienna^wzorzec}`	Zamień znaki w zmiennej na wielkie litery. Jeśli pierwsza litera w wartości zmiennej pasuje do wzorca, zamieniana jest na wielką. *Zmienna* może mieć wartość * albo @; w takim wypadku modyfikowane są parametry. *Zmienna* może być też tabelą indeksowaną przez * albo @; w tym wypadku podstawienie wykonywane jest dla wszystkich elementów tabeli.
`${zmienna^^wzorzec}`	Tak jak *^wzorzec*, ale sprawdza każdy znak w ciągu znaków.
`${zmienna,wzorzec}`	Tak jak *^wzorzec*, ale zamienia pasujące znaki na małe litery. Sprawdza dopasowanie tylko dla pierwszego znaku w ciągu znaków.
`${zmienna,,wzorzec}`	Tak jak *,wzorzec*, ale sprawdza każdy znak w ciągu znaków.
`${zmienna@a}`	Użyj flag reprezentujących atrybuty zmiennej (jak dla `declare`). Zmienna może być tabelą indeksowaną przez * albo @; w tym wypadku przekształcenie wykonywane jest dla wszystkich elementów.
`${zmienna@A}`	Łańcuch znaków w formie polecenia lub przypisania wartości, który po sprawdzeniu odtwarza *zmienną* i jej wartości. Zmienna może być tabelą indeksowaną przez * albo @; w tym wypadku przekształcenie wykonywane jest dla wszystkich elementów.
`${zmienna@E}`	Wartość *zmiennej* z sekwencją ucieczki `$'...'` (zobacz sekcję „Sekwencje ucieczki”). Zmienna może być tabelą indeksowaną przez * albo @; w tym wypadku przekształcenie wykonywane jest dla wszystkich elementów.
`${zmienna@P}`	Wartość *zmiennej* ze sprawdzonymi sekwencjami ucieczki dla tekstu zachęty (zobacz sekcję „Specjalne znaki zachęty”). Zmienna może być tabelą indeksowaną przez * albo @; w tym wypadku przekształcenie wykonywane jest dla wszystkich elementów.
`${zmienna@Q}`	Wartość *zmiennej* zacytowana w taki sposób, że można wprowadzić wartości jako wejście. Zmienna może być tabelą indeksowaną przez * albo @; w tym wypadku przekształcenie wykonywane jest dla wszystkich elementów.

`${!prefiks*}, ${!prefiks@}`	Lista zmiennych, których nazwy zaczynają się od *prefiksu*.
`${zmienna:poz}`, `${zmienna:poz:zakres}`	Zaczynając od pozycji *poz* (liczonej od 0), w *zmiennej* wydobądź *zakres* znaków albo pozostałe znaki, jeżeli nie określono *zakresu*. *poz* i *zakres* mogą być wyrażeniami arytmetycznymi. Jeśli *zakres* jest liczbą ujemną, powłoka policzy znaki od końca. Jeżeli zmienna to * albo @, interpretacja wykonywana jest na parametrach. Jeżeli *poz* wynosi 0, na liście wyników uwzględniony zostanie parametr $0. Analogicznie zmienna może być tabelą indeksowaną przez * lub @.
`${zmienna/wzorzec/zast}`	Użyj wartości *zmiennej*, pierwsze dopasowanie do *wzorca* jest zastępowane przez *zast*.
`${zmienna/wzorzec}`	Użyj wartości *zmiennej*, pierwsze dopasowanie do *wzorca* jest usuwane.
`${zmienna//wzorzec/zast}`	Użyj wartości *zmiennej*, każde dopasowanie do *wzorca* jest zastępowane przez *zast*.
`${zmienna/#wzorzec/zast}`	Użyj wartości *zmiennej*, każde dopasowanie do *wzorca* jest zastępowane przez *zast*. Dopasowanie musi wystąpić na początku wartości.
`${zmienna/%wzorzec/zast}`	Użyj wartości *zmiennej*, każde dopasowanie do *wzorca* jest zastępowane przez *zast*. Dopasowanie musi wystąpić na końcu wartości.
`${!zmienna}`	Użyj wartości *zmiennej* jako nazwy innej zmiennej (odwołanie pośrednie).

Przykłady

```
$ z=za d=dol pusta=                  Przypisz wartości do trzech zmiennych (ostatnia to null)
$ echo ${z}korzenic                  Klamra wymagana
zakorzenic

$ echo ${z-$d}                       Wyświetl wartość z lub d; z posiada wartość, więc wypisz ją
za

$ echo ${tmp-`date`}                 Jeśli zmienna tmp nie jest ustawiona, wykonaj polecenie date
Tue Feb 2 22:52:57 EST 2016

$ echo ${pusta="brak danych"}        Zmienna pusta jest ustawiona, więc zostanie wyświetlona
                                      (pusta linia)
$ echo ${pusta:="brak danych"}       Zmienna pusta jest ustawiona jako null, wyświetl tekst
brak danych

$ echo $pusta                        Pusta ma teraz nową wartość
brak danych

# Z nazwy bieżącej lokalizacji usuń najdłuższy ciąg znaków kończący się /,
# co usuwa ścieżkę i zostawia jej końcówkę.
$ koniec=${PWD##*/}

# Słynny wyraz
$ wyraz=konstantynopolitanczykowianeczka
```

```
# Zmień wielkość pierwszej litery
$ echo ${wyraz^[k-n]}
Konstantynopolitanczykowianeczka
# Zmień wielkość wszystkich pasujących znaków
$ echo ${wyraz^^[r-t]}
konSTanTynopoliTanczykowianeczka
```

Zmienne pośrednie

Zmienne pośrednie to zmienne przechowujące nazwy innych zmiennych. Wszystkie działania (odwołania, przypisanie wartości i zmiany atrybutów) wykonane na zmiennych pośrednich wykonywane są tak naprawdę na zmiennej, której nazwę przechowuje zmienna pośrednia. Tworzy się je poleceniem `declare –n`, usuwa poprzez `unset -n`, a testuje za pomocą `test -R`. Na przykład:

```
$ powitanie="witaj swiecie"          Zwyczajne przypisanie zmiennej
$ declare -n wiadomosc=powitanie     Deklaracja zmiennej pośredniej
$ echo $wiadomosc                    Dostęp przez zmienną pośrednią
witaj swiecie                         Wartość to $powitanie
$ wiadomosc="zegnaj"                 Przypisanie wartości przez zmienną pośrednią
$ echo $powitanie                    Demonstracja zmiany
zegnaj
```

Bash posiada również specjalną składnię, która umożliwia pośrednie odwołanie do zmiennej, ale nie można w ten sposób przypisywać wartości:

```
$ tekst=powitanie   Przypisanie zmiennej
$ echo ${!tekst}    Użycie aliasu
zegnaj
```

Jeżeli użyjemy zmiennej pośredniej jako zmiennej kontrolnej w pętli `for`, warunki pętli są traktowane jak nazwy zmiennych i zmienna kontrolna odwołuje się do każdej z nich:

```
$ declare -n nr        Utwórz zmienną pośrednią
$ i=1                  Prosty licznik
$ for nr in v1 v2 v3   Rozpocznij pętlę
> do
>   nr=$((i++))        Każda zmienna dostaje unikalną wartość
> done
$ echo $v1 $v2 $v3     Pokaż wyniki
1 2 3
```

Zamiana istniejącej zmiennej na zmienną pośrednią blokuje atrybuty `-c`, `-i`, `-l` i `-u` (sprawdź sekcję poświęconą `declare`).

Wbudowane zmienne powłoki

Powłoka automatycznie ustawia zmienne wbudowane, które używane są w skryptach powłoki. Zmienne wbudowane mogą wykorzystywać wzorce podstawiania z poprzednich sekcji. Miej na uwadze, że znak $ nie jest częścią nazwy zmiennej, chociaż wykorzystuje się go zawsze w odwołaniach do zmiennej. Poniższe zmienne są dostępne w każdej powłoce kompatybilnej z Bourne:

`$#` Liczba argumentów w wierszu poleceń.

`$-` Obecnie działające opcje (podane w wierszu poleceń lub wraz z poleceniem `set`). Powłoka ustawia kilka opcji automatycznie.

`$?` Status wyjściowy ostatniego polecenia.

`$$` Numer procesu powłoki.

`$!` Numer procesu ostatniego polecenia w tle.

`$0` Pierwsze słowo, tj. nazwa polecenia. Jeżeli zostało ono wyszukane w zmiennej środowiskowej PATH, zmienna będzie przechowywała całą ścieżkę.

`$`*n* Poszczególne argumenty wiersza poleceń (parametry). Powłoka Bourne dopuszcza bezpośrednie odwołanie do maksymalnie dziewięciu parametrów (*n* = 1 ... 9). Bash dopuszcza *n* większe od 9 przy zastosowaniu składni `${`*n*`}`.

`$*, $@` Wszystkie argumenty z wiersza poleceń (`$1 $2 ...`).

`"$*"` Wszystkie argumenty z wiersza poleceń jako jeden ciąg znaków (`"$1 $2..."`). Wartości są rozdzielone pierwszym znakiem z `$IFS`.

`"$@"` Wszystkie argumenty z wiersza poleceń zacytowane pojedynczo (`"$@"`, `"$@"...`).

Bash automatycznie ustawia poniższe zmienne dodatkowe[2]:

`$_` Zmienna tymczasowa, w której zapisywana jest ścieżka wykonywanego skryptu lub programu. Później przechowuje ostatni argument poprzedniego polecenia. Podczas sprawdzania maili przechowuje pasujący plik `MAIL`.

`BASH` Pełna ścieżka użyta do wywołania tej instancji powłoki.

`BASHOPTS` Lista aktywnych opcji powłoki rozdzielonych przecinkami, tylko do odczytu. Każda z opcji na liście jest prawidłowa dla `shopt -s`. Jeśli ta zmienna istnieje w środowisku w momencie uruchomienia Basha, opcje zawarte w niej zostaną ustawione przed wykonaniem plików startowych.

`BASHPID` Identyfikator procesu bieżącej powłoki. W niektórych przypadkach wartość może być inna niż w `$$`.

`BASH_ALIASES` Tabela asocjacyjna, w której każdy element zawiera alias zdefiniowany poleceniem `alias`. Dodanie elementu tworzy nowy alias.

[2] Nie wszystkie zmienne są zawsze ustawiane. Na przykład zmienne `COMP*` przyjmują wartości tylko wtedy, gdy włączone są programowalne funkcje uzupełniania.

BASH_ARGC	Zmienna tablicowa, której elementy zawierają liczbę argumentów dla odpowiadającej funkcji lub wywołania skryptu. Ustawiana tylko w rozszerzonym trybie debug (`shopt -s extdebug`). Nie może być nieustawiona.
BASH_ARGV	Zmienna podobna do BASH_ARGC. Każdy element jest jednym z argumentów przekazywanych do funkcji lub skryptu. Działa ona na zasadzie stosu, z którego wartości są pobierane przy każdym wywołaniu. Ostatni element to ostatni argument ostatniej funkcji lub wywołania skryptu. Ustawiana tylko w rozszerzonym trybie debug. Nie może być nieustawiona.
BASH_CMDS	Tabela asocjacyjna. Każdy element odwołuje się do polecenia w wewnętrznej tabeli haszującej zarządzanej przez polecenie `hash`. Indeks to nazwa polecenia, a wartość to jego pełna ścieżka. Dodanie elementu do tabeli spowoduje dodanie polecenia do tabeli haszującej.
BASH_COMMAND	Polecenie, które wykonywane jest obecnie albo zaraz zostanie wykonane. W obsłudze pułapki jest to polecenie, które było wykonywane w momencie wywołania pułapki.
BASH_EXECUTION_STRING	Argument przekazany do opcji `-c`.
BASH_LINENO	Zmienna tablicowa odpowiadająca BASH_SOURCE i FUNCNAME. Dla każdego numeru funkcji `i` (zaczynając od zera) `${FUNCNAME[i]}` zostało wywołane w pliku *${BASH_SOURCE[i]}* w linii `${BASH_LINENO[i]}`. Pierwsza w tabeli znajduje się ostatnio wywołana funkcja. Zmienna nie może być nieustawiona.
BASH_REMATCH	Tablica przypisywana za pomocą operatora =~ w konstrukcji `[[ ]]`. Indeks zerowy to tekst, który spełniał dopasowanie do całego wzorca. Kolejne pozycje to tekst, który pasował do podgrup wzorca znajdujących się w nawiasach. Zmienna jest tylko do odczytu.
BASH_SOURCE	Zmienna tablicowa zawierająca nazwy plików źródłowych. Każdy element odpowiada pozycjom w FUNCNAME i BASH_LINEO. Zmienna nie może być nieustawiona.
BASH_SUBSHELL	Ta zmienna jest inkrementowana za każdym razem, gdy tworzona jest podpowłoka lub środowisko podpowłoki.
BASH_VERSINFO[0]	Główny numer wersji lub rewizji Basha.
BASH_VERSINFO[1]	Drugorzędny numer wersji Basha.
BASH_VERSINFO[2]	Numer poprawki.
BASH_VERSINFO[3]	Numer kompilacji.
BASH_VERSINFO[4]	Status wersji.
BASH_VERSINFO[5]	Typ maszyny; dane takie same jak w zmiennej $MACHTYPE.
BASH_VERSION	Tekst opisujący wersję powłoki.
COMP_CWORD	Zmienna wykorzystywana przez programowalne uzupełnianie. Indeksuje COMP_WORDS, wskazując bieżącą pozycję kursora.
COMP_KEY	Zmienna wykorzystywana przez programowalne uzupełnianie. Zawiera znak lub ostatni ze znaków z sekwencji, która wywołała funkcję uzupełniania.
COMP_LINE	Zmienna wykorzystywana przez programowalne uzupełnianie. Bieżący wiersz poleceń.

`COMP_POINT`	Zmienna wykorzystywana przez programowalne uzupełnianie. Pozycja kursora jako indeks znaku w `$COMP_LINE`.
`COMP_TYPE`	Zmienna wykorzystywana przez programowalne uzupełnianie. Znak opisujący typ uzupełniania. Normalne uzupełnianie oznaczane jest tabulacją, `?` oznacza listę uzupełnień po naciśnięciu klawisza *Tab* dwa razy, `!` oznacza listę alternatyw w uzupełnianiu częściowym, `@` oznacza uzupełnienie, jeśli zmodyfikowano wyraz, a `%` to uzupełnienie menu.
`COMP_WORDBREAKS`	Zmienna wykorzystywana przez programowalne uzupełnianie. Znaki, które podczas uzupełniania traktowane są przez **readline** jako separatory wyrazów.
`COMP_WORDS`	Zmienna wykorzystywana przez programowalne uzupełnianie. Tabela zawiera pojedyncze słowa z wiersza poleceń.
`COPROC`	Tablica, która przechowuje deskryptory plików służących do komunikacji z nienazwanym procesem współbieżnym. Więcej informacji znajduje się w sekcji „Procesy współbieżne”.
`DIRSTACK`	Zmienna tablicowa zawierająca stos lokalizacji wyświetlany przez `dirs`. Zmiana istniejących elementów powoduje zmianę stosu, ale dodawanie i usuwanie elementów może odbywać się tylko przy użyciu `pushd` i `popd`.
`EUID`	Zmienna tylko do odczytu zawierająca efektywny identyfikator bieżącego użytkownika (UID).
`FUNCNAME`	Zmienna tablicowa zawierająca nazwy funkcji. Każdy element odpowiada elementom z tablic `BASH_SOURCE` i `BASH_LINENO`.
`FUNCNEST`	Wartość większa od zera, która określa maksymalny poziom zagnieżdżenia wywoływania funkcji. Jeżeli został przekroczony, bieżące polecenie zostanie anulowane.
`GROUPS`	Zmienna tablicowa zawierająca listę numerycznych identyfikatorów grup, do których należy bieżący użytkownik.
`HISTCMD`	Numer historii bieżącego polecenia.
`HOSTNAME`	Nazwa bieżącego hosta.
`HOSTTYPE`	Tekst opisujący system hosta.
`LINENO`	Bieżący numer linii skryptu lub funkcji.
`MACHTYPE`	Opis systemu hosta w formacie GNU **procesor-firma-system**.
`MAPFILE`	Domyślna tabela dla poleceń `mapfile` i `readarray`. Więcej informacji znajdziesz w sekcji poświęconej poleceniu `mapfile`.
`OLDPWD`	Poprzednia lokalizacja robocza.
`OPTARG`	Wartość argumentu ostatniej opcji przetworzonej przez `getopts`.
`OPTIND`	Numeryczny indeks `OPTARG`.
`OSTYPE`	Ciąg znaków opisujący system operacyjny.

`PIPESTATUS`	Zmienna tablicowa zawierająca statusy wyjściowe poleceń w najbardziej pierwszoplanowym potoku. Pamiętaj, że potok może zawierać tylko jedno polecenie.
`PPID`	Numer procesu rodzica powłoki.
`PWD`	Bieżąca lokalizacja robocza (ustawiana poleceniem `cd`).
`RANDOM[=`*n*`]`	Wygeneruj nową losową liczbę przy każdym odwołaniu; zaczyna od *n*, jeżeli zostało podane.
`READLINE_LINE`	Do użytku z `bind -x`. Zmienna przechowuje zawartość bufora edycji.
`READLINE_POINT`	Do użytku z `bind -x`. Indeks punktu wstawienia dla `$READLINE_LINE`.
`REPLY`	Domyślna odpowiedź; używana przez `select` i `read`.
`SECONDS[=`*n*`]`	Liczba sekund od uruchomienia powłoki. Jeżeli podano *n*, polecenie zwróci *n*+liczba sekund.
`SHELLOPTS`	Lista rozdzielonych przecinkiem opcji powłoki (dla `set -o`), tylko do odczytu. Jeżeli istnieje w środowisku w momencie uruchomienia powłoki, Bash włączy najpierw opcje zdeklarowane w niej, zanim wykona pliki uruchomieniowe.
`SHLVL`	Zwiększane o 1 przy każdym uruchomieniu Basha.
`UID`	Identyfikator bieżącego użytkownika. Tylko do odczytu.

Wiele z tych zmiennych wspiera programowalne uzupełnianie (zobacz sekcję „Programowalne uzupełnianie") lub Bash Debuggera (zobacz *http://bashdb.sourceforge.net/*).

Inne zmienne powłoki

Zmienne przedstawione poniżej nie są ustawiane automatycznie przez powłokę, jednak wiele z nich może wpływać na jej zachowanie. Zazwyczaj ustawia się je w plikach *.bash_profile* lub *.profile*, w których możesz zdefiniować je zgodnie z Twoimi potrzebami. Można przypisać im wartości, korzystając ze składni:

```
zmienna=wartość
```

Na poniższej liście wymieniono, jakich wartości oczekuje się przy definiowaniu zmiennych:

`BASH_COMPAT`	Jeśli wartością jest liczba dziesiętna lub całkowita (na przykład 4,3 lub 43), która odpowiada poziomowi kompatybilności powłoki, włącza dany poziom kompatybilności (np. 4,3 lub 43 odpowiadają `shopt -s compat43`). Jeśli zmienna nie ma wartości lub wartością jest pusty ciąg tekstowy, włączona zostanie kompatybilność bieżącej powłoki. Polecenie `shopt` nie zmienia wartości tej zmiennej. Wartość może być dziedziczona ze środowiska.
`BASH_ENV`	Jeżeli jest ustawiona przy uruchomieniu, określa nazwę pliku, w którym znajdują się polecenia inicjalizacyjne. Zanim wartość zostanie zinterpretowana jako nazwa pliku, przechodzi przez interpretację parametru, podstawienie polecenia i interpretację arytmetyczną.

`BASH_LOADABLES_PATH`	Jedna ścieżka lub więcej, rozdzielone dwukropkami, w których powłoka szuka dynamicznie ładowanych poleceń wbudowanych określonych przez polecenie `enable`.
`BASH_XTRACEFD=`*`n`*	Deskryptor plików, do którego Bash zapisuje wyjście śledzenia (z `set -x`).
`CDPATH=`*`lokalizacje`*	Lokalizacje przeszukiwane przez `cd`. Pozwala na stosowanie skrótów podczas zmiany lokalizacji. Domyślnie nieustawiona.
`CHILD_MAX=`*`n`*	Określa maksymalną liczbę procesów podrzędnych, dla których powłoka zapamięta statusy wyjściowe. Maksymalna liczba to 8192, minimalna zależy od systemu.
`COLUMNS=`*`n`*	Szerokość kolumn ekranu; używana w trybach edycji linii i na listach `select`. Domyślnie bieżąca szerokość terminala.
`COMPREPLY=(`*`słowa...`*`)`	Tabela, z której Bash odczytuje możliwe uzupełnienia wygenerowane przez funkcje uzupełnień.
`EMACS`	Jeżeli wartość zaczyna się od `t`, Bash zakłada, że włączony jest bufor Emacs, i wyłącza edycję linii.
`ENV=`*`plik`*	Nazwa pliku wykonywanego podczas uruchamiania powłoki w trybie POSIX lub gdy Bash jest wywoływany jako */bin/sh*. Przydatna do przechowywania aliasów i definicji funkcji, np. `ENV=$HOME/.shellrc`.
`EXECIGNORE=`*`lista_wzorców`*	Lista wzorców rozdzielona dwukropkami; opisuje nazwy plików, które mają być ignorowane podczas wyszukiwania plików wykonywalnych. Wykorzystywana do ignorowania wykonywalnych plików współdzielonych bibliotek. Pod uwagę brana jest też wartość opcji `extglob`.
`FCEDIT=`*`plik`*	Edytor używany przez polecenie `fc`. Domyślna wartość to */bin/ed* dla powłoki w trybie POSIX. Inaczej domyślny edytor znajduje się w zmiennej `$EDITOR` lub jest nim `vi`, jeżeli zmienna jest nieustawiona.
`FIGNORE=`*`lista_wzorców`*	Lista sufiksów rozdzielona dwukropkami, która określa nazwy plików ignorowane podczas uzupełniania nazw plików dla biblioteki **readline**.
`GLOBIGNORE=`*`lista_wzorców`*	Lista rozdzielonych dwukropkami wzorców określających zbiór nazw plików, które mają być ignorowane przy porównywaniu ze wzorcem. Wartości opcji `nocasematch` i `extglob` są brane pod uwagę.
`HISTCONTROL=`*`lista`*	Lista rozdzielonych dwukropkami wartości, które kontrolują, w jaki sposób zapisywane są polecenia w pliku historii. Przyjmowane wartości to `ignoredups`, `ignorespace`, `ignoreboth` oraz `erasedups`.
`HISTFILE=`*`plik`*	Plik, w którym przechowywana jest historia poleceń. Domyślna wartość to *~/.bash_history*.
`HISTFILESIZE=`*`n`*	Maksymalna liczba linii w pliku historii. Może się różnić od liczby poleceń. Jeżeli wynosi zero, w pliku nie będą przechowywane żadne polecenia, jeśli wartość jest ujemna lub nienumeryczna — nie ma limitu. Wartość domyślna to 500.

`HISTIGNORE=`*`lista`*	Lista rozdzielonych dwukropkami wzorców, które muszą pasować do całego wiersza poleceń. Pasujące wiersze **nie są** zapisywane w historii. Znak & we wzorcu bez znaku ucieczki oznacza poprzednią linię historii. Wartość opcji `extglob` jest uwzględniana.
`HISTSIZE=`*`n`*	Maksymalna liczba poleceń przechowywanych w historii. Jeżeli wartość wynosi zero, historia nie przechowuje żadnych poleceń. Wartość ujemna lub nienumeryczna oznacza brak limitu. Wartość domyślna to `500`.
`HISTTIMEFORMAT=`*`format`*	Tekst formatujący *strftime(3)* używany przy wyświetlaniu znaczników czasu przy poleceniach dla polecenia `history`. Jeśli zmienna ma wartość (nawet `null`), powłoka zapisze w historii polecenia ze znacznikami czasu.
`HOME=`*`lokalizacja`*	Lokalizacja domowa; ustawiana przez `login` (z pliku */etc/passwd*).
`HOSTFILE=`*`plik`*	Nazwa pliku sformatowanego tak samo jak */etc/hosts*, z którego powinna korzystać powłoka przy uzupełnianiu nazw hosta.
`IFS='`*`znaki`*`'`	Separatory wejścia. Domyślnie spacja, tabulator, znak nowej linii.
`IGNOREEOF=`*`n`*	Wartość numeryczna wskazująca, ile następujących po sobie znaków końca pliku należy wpisać, aby Bash zakończył działanie. Jeśli wartość jest nienumeryczna lub wynosi `null`, domyślnie zostaje ustawione `10`. Dotyczy tylko powłok interaktywnych.
`INPUTRC=`*`plik`*	Inicjalizacja pliku dla biblioteki *readline*. Nadpisuje domyślną wartość *~/.inputrc*.
`LANG=`*`ustawienia`*	Domyślna wartość ustawień lokalnych. Używana, jeśli nie ustawiono żadnej ze zmiennych `LC_*`.
`LC_ALL=`*`ustawienia`*	Bieżące ustawienia lokalne; nadpisuje `LANG` i pozostałe zmienne `LC_*`.
`LC_COLLATE=`*`ustawienia`*	Ustawienia lokalne używane do porównywania znaków (kolejność sortowania).
`LC_CTYPE=`*`ustawienia`*	Ustawienia lokalne dla funkcji klas znaków (zobacz sekcję „Metaznaki nazw plików”).
`LC_MESSAGES=`*`ustawienia`*	Ustawienia lokalne używane do tłumaczenia ciągów `$"..."`.
`LC_NUMERIC=`*`ustawienia`*	Ustawienia lokalne zapisu ułamków dziesiętnych.
`LC_TIME=`*`ustawienia`*	Ustawienia lokalne formatu daty i czasu.
`LINES=`*`n`*	Wysokość ekranu; używana dla list `select`. Domyślnie wysokość bieżącego terminala.
`MAIL=`*`plik`*	Domyślny plik do sprawdzania przychodzących maili; ustawiane przez `login`.
`MAILCHECK=`*`n`*	Liczba sekund przerwy między sprawdzeniem maili. Domyślnie to 60 sekund (minuta).
`MAILPATH=`*`pliki`*	Jeden lub kilka rozdzielonych dwukropkami plików do sprawdzania przychodzących maili. Do każdego pliku możesz określić komunikat, który wyświetli powłoka, jeśli plik zmieni swój rozmiar. Komunikaty są oddzielone od plików znakiem ?. Domyślny komunikat to `You have mail in $_` (Masz wiadomość w `$_`). `$_` zastępowane jest nazwą pliku. Możesz na przykład przypisać wartość w taki sposób: `MAILPATH="$MAIL?Kot w worku!:/etc/motd?Nowa Wiadomość Logowania"`.

`OPTERR=`*`n`*	Gdy wartość wynosi `1` (domyślnie), Bash wyświetla komunikaty błędów z wbudowanego polecenia `getopts`.
`PATH=`*`lokalizacje`*	Jedna ścieżka lub kilka rozdzielonych dwukropkami, w których wyszukuje się polecenia do wykonania. Wbudowana wartość domyślna to: `/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:.`. W wielu systemach wartość domyślna to: `/bin:/usr/bin`.
`POSIXLY_CORRECT=`*`tekst`*	Gdy zostanie ustawiona podczas uruchamiania powłoki lub w trakcie jej działania, Bash wejdzie w tryb POSIX, blokując zachowania i funkcjonalności niezgodne z POSIX.
`PROMPT_COMMAND=`*`polecenie`*	Jeśli zmienna ma przypisaną wartość, Bash wykona zadeklarowane polecenie za każdym razem przed wyświetleniem znaku zachęty.
`PROMPT_DIRTRIM=`*`n`*	Wskazuje, ile elementów adresu ma zostać skróconych dla specjalnych znaków zachęty `\w` lub `\W` (zobacz sekcję „Specjalne znaki zachęty”). Skrócone elementy zastępowane są tyldą.
`PS0=`*`tekst`*	Tekst wyświetlany przez powłoki interaktywne po odczytaniu polecenia, ale przed wykonaniem go.
`PS1=`*`tekst`*	Pierwszy znak zachęty; domyślnie `'\s-\v\$ '`.
`PS2=`*`tekst`*	Drugi znak zachęty (używany w poleceniach wieloliniowych); domyślnie `>`.
`PS3=`*`tekst`*	Znak zachęty w pętlach `select`; domyślnie `#?`.
`PS4=`*`tekst`*	Znak zachęty dla śledzenia wykonywania (`bash -x` albo `set -x`); domyślnie `+`. Powłoki działające jako root nie dziedziczą tej zmiennej ze środowiska.
`SHELL=`*`plik`*	Nazwa domyślnej powłoki użytkownika (np. */bin/sh*). Bash ustawia wartość tej zmiennej, jeśli nie ma jej w środowisku podczas uruchomienia.
`TERM=`*`tekst`*	Typ terminala.
`TIMEFORMAT=`*`tekst`*	Format wyjścia słowa kluczowego `time`. Szczegóły w *bash(1)*.
`TMOUT=`*`n`*	Jeśli nie wpisano żadnego polecenia po *n* sekundach, zakończ pracę powłoki. Wpływa również na polecenie `read` i pętlę `select`.
`TMPDIR=`*`lokalizacja`*	*`Lokalizacja`*, w której przechowywane są pliki tymczasowe tworzone przez powłokę.
`auto_resume=`*`lista`*	Umożliwia używanie prostych ciągów znaków do uruchamiania zatrzymanych procesów. Z wartością `exact` ciąg musi być taki sam jak nazwa procesu. Z wartością `substring` może pasować do fragmentu nazwy.
`histchars=`*`znaki`*	Dwa lub trzy znaki sterujące rozwijaniem historii w stylu `csh`. Pierwszy znak określa zdarzenie wywołujące historię, drugi to znak „szybkiego podstawienia”, trzeci określa rozpoczęcie komentarza. Wartość domyślna to `!^#`. Zobacz sekcję „Historia w stylu powłoki C”.

Tablice

Bash udostępnia dwa rodzaje tablic: **tablice indeksowane**, w których indeksy to liczby od zera w górę, i **tablice asocjacyjne**, w których indeksami są ciągi znaków.

Tablice indeksowane

Bash umożliwia tworzenie tablic jednowymiarowych. Indeks pierwszego elementu wynosi zero. Powłoka nie ma ograniczeń odnośnie liczby elementów. Tablice tworzy się przy użyciu następującej składni:

```
wiadomosc=(czesc jak sie masz)
```

w której określone w nawiasach wartości stają się elementami tablicy. Pojedyncze elementy można również przypisać następująco:

```
wiadomosc[0]=czesc      #Trudniejszy sposób
wiadomosc[1]=jak
wiadomosc[2]=sie
wiadomosc[3]=masz
```

Tablic indeksowanych nie trzeba deklarować bezpośrednio. Każde odwołanie do indeksowanej zmiennej może utworzyć tablicę.

Odwołując się do tablic, używaj składni `${...}`. Nie jest to jednak konieczne wewnątrz `((...))` (forma `let` wykonująca automatyczne cytowanie). Pamiętaj, że znaki `[ i ]` nie są opcjonalne.

Ujemne indeksy liczy się na zasadzie odejmowania od wartości ostatniego indeksu powiększonej o jeden.

```
$ a=(0 1 2 3 4 5 6 7 8)  Utwórz tablicę indeksowaną
$ echo ${a[4]}           Użyj dodatniego indeksu
4
$ echo ${a[-2]}          Użyj indeksu: 8+1−2 = 7
7
```

Podstawienie tablic

Dostępne są następujące podstawienia dla tablic i ich elementów:

`${`*nazwa*`[`*i*`]}`	Użyj elementu *i* tablicy *nazwa*; *i* może być dowolnym wyrażeniem arytmetycznym, tak jak opisano w sekcji „Wyrażenia arytmetyczne”.
`${`*nazwa*`}`	Użyj elementu tablicy *nazwa* z indeksem 0.
`${`*nazwa*`[*]}`, `${`*nazwa*`[@]}`	Użyj wszystkich elementów tablicy *nazwa*.
`${#`*nazwa*`[*]}`, `${#`*nazwa*`[@]}`	Użyj liczby elementów w tablicy.

Tablice asocjacyjne

Bash umożliwia korzystanie z tablic asocjacyjnych, w których indeksami są ciągi znaków zamiast liczb (tak jak w `awk`). W tym wypadku znaki `[` i `]` działają jak cudzysłów. Tablice asocjacyjne muszą być deklarowane za pomocą opcji `-A` w poleceniach `declare`, `local` i `readonly`. Specjalna składnia umożliwia dodanie wielu elementów jednocześnie:

```
dane=([jan]=30 [maria]=25)  Dodanie wartości do tablicy asocjacyjnej
wiadomosc=([0]=jak [2]=leci) Dodanie wartości do tablicy indeksowanej
```

Do pobrania wartości użyj zapisu `${dane[jan]}` i `${dane[maria]}`.

Do pobrania indeksów tablic asocjacyjnych wykorzystuje się te same specjalne interpretacje co dla tablic indeksowanych.

Specjalne znaki zachęty

Bash przetwarza wartości zmiennych `PS0`, `PS1`, `PS2` i `PS4` pod kątem poniższych specjalnych sekwencji ucieczki:

`\a`	Znak dzwonka (BEL) ASCII (07 ósemkowo).
`\A`	Aktualna godzina w formacie dwudziestoczterogodzinnym HH:MM.
`\d`	Data w formacie „dzień tygodnia miesiąc dzień".
`\D{`*`format`*`}`	Data w *`formacie`* określonym zgodnie z *strftime(3)*.
`\e`	Znak ESCAPE ASCII (033 ósemkowo).
`\h`	Nazwa hosta do pierwszej kropki.
`\H`	Pełna nazwa hosta.
`\j`	Bieżąca liczba procesów.
`\l`	Główna nazwa terminala powłoki.
`\n`	Znak nowej linii.
`\r`	Znak powrotu karetki.
`\s`	Nazwa powłoki (główna nazwa `$0`).
`\t`	Bieżący czas w formacie dwudziestoczterogodzinnym HH:MM:SS.
`\T`	Bieżący czas w formacie dwunastogodzinnym HH:MM:SS.
`\u`	Nazwa bieżącego użytkownika.
`\v`	Wersja powłoki.
`\V`	Pełny numer wersji Basha (wersja plus numer poprawki).

\w	Bieżąca lokalizacja z adresem ze zmiennej $HOME skróconym do ~. Zobacz też opis zmiennej PROMPT_DIRTRIM.
\W	Główna nazwa bieżącej lokalizacji z adresem ze zmiennej $HOME skróconym do ~. Zobacz też opis zmiennej PROMPT_DIRTRIM.
\!	Numer historii polecenia (przechowywany w historii).
\#	Numer bieżącej komendy (licznik poleceń wykonywanych przez daną powłokę).
\$	Jeżeli efektywny UID wynosi 0, wyświetl #. W przeciwnym razie wyświetl $.
\@	Bieżąca godzina w formacie dwunastogodzinnym a.m./p.m.
nnn	Znak reprezentowany przez wartość ósemkową *nnn*.
\\	Ukośnik.
\[	Rozpocznij sekwencję niewyświetlanych znaków, np. określających wyróżnienie lub kolory w emulatorze terminala.
\]	Zakończ sekwencję niewyświetlanych znaków.

Zmienne PS0, PS1, PS2 i PS4 są przetwarzane pod kątem sekwencji ucieczki, zmiennych, poleceń i wyrażeń arytmetycznych. Na początku przetwarzane są sekwencje ucieczki. Następnie, jeśli opcja powłoki promptvars jest włączona przez polecenie shopt (domyślnie), wykonywane są podstawienia.

W trybie POSIX przetwarzanie odbywa się inaczej. Wartości PS1 i PS2 przetwarzane są pod kątem interpretacji parametrów, ! zastępowany jest numerem historii bieżącej komendy, a !! oznacza wykrzyknik.

Skorowidz

O'REILLY®

Poznaj Basha — Twoje narzędzie pozwalające w pełni kontrolować system!

Bash jest podstawową powłoką dla systemów operacyjnych GNU/Linux i Mac OS X, dostępną również dla systemu Solaris oraz dla systemów z rodziny BSD. Może być też z łatwością skompilowany dla każdego innego systemu Unix, a nawet dla OpenVMS. Każdy administrator komputerów pracujących pod kontrolą Uniksa lub pokrewnego systemu powinien umieć posługiwać się Bashem.

Jeśli pisanie skryptów powłoki sprawia Ci problemy, sięgnij po to zwięzłe i praktyczne kompendium do nauki Basha w wersji 4.4, podstawowej powłoki dla systemów operacyjnych z rodziny Unix. Układ książki ułatwia szybkie przeglądanie i wyszukiwanie interesującej nas treści. Równocześnie zawarte w niej informacje pozwolą każdemu początkującemu administratorowi na sprawne rozpoczęcie pracy z powłoką Bash.

Arnold Robbins — programista i autor książek technicznych. Pierwszy raz zetknął się z systemami uniksowymi w 1980 roku, kilka lat później zaczął pisać skrypty powłoki. Obecnie jest opiekunem implementacji gawk i związanej z nim dokumentacji.

W leksykonie przedstawiono:

- zwięzłą historię i przegląd funkcjonalności Basha w wersji 4.4
- sposób wywoływania powłoki oraz składnię poleceń
- funkcje, zmienne, wyrażenia arytmetyczne i inne elementy języka
- wykorzystanie programowalnego uzupełniania
- zasady kontroli zadań i wykonywania poleceń

Helion

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

ISBN 978-83-283-2820-4

9 788328 328204

Informatyka w najlepszym wydaniu

cena: 29,90 zł