GNU Bash

Streszczenie

Artykuł omawia powłokę GNU Bash, będącą podstawową powłoką w systemach GNU, takich jak GNU/Linux. Zwrócona została uwaga na rozszerzone względem standardowej powłoki Sh możliwości Basha. Omówione są funkcje powłoki, jej sposób uruchamiania i elementy języka skryptowego. Zaprezentowane są niektóre możliwości biblioteki GNU Readline zwiększającej wygodę pracy z powłoką. Na koniec przedstawione są zaawansowane narzędzia wspomagające konfigurowanie Basha.

Spis treści

1. Wstęp

W systemie GNU/Linux, podobnie jak w systemach Unix, podstawowym interfejsem pomiędzy użytkownikiem a systemem jest powłoka czyli z angielskiego shell. Podstawową powłoką systemu GNU przygotowywanego przez Free Software Foundation jest GNU Bash. Ponieważ system GNU/Linux przejął większość narzędzi GNU najczęściej używa Basha jako domyślnej powłoki systemu. Ponieważ wielu użytkowników Basha traktuje go najczęściej po prostu jako wersję Sh, nie zdaje sobie sprawy z wielu jego zaawansowanych możliwości i rozszerzeń względem oryginalnego Sh. Artykuł omawia te możliwości, a także prezentuje specyficzne narzędzia wspomagające konfigurowanie samej powłoki.

2. Czym jest GNU Bash?

Nazwa Bash jest skrótem od „Bourne–Again SHell' i nawiązuje do nazwiska autora oryginalnej powłoki uniksowej Sh, Steva Bourne'a. Nawiasem mówiąc, nazwa jest również pewną grą słów, gdyż nazwisko Bourne wymawia się podobnie do angielskiego słowa „born", co można tłumaczyć jako „narodzony/na/ne". W związku z czym nazwę Bash można odczytywać jako „Ponownie narodzony SH", lub też jako, „Ponownie narodzony shell Bourne'a", co wskazuje na pochodzenie Basha.

Autorem Basha jest Brian Fox, pracownik Free Software Foundation. Po nim opiekę nad Bashem i jego rozbudową przejął Chet Ramey. Bash jest dzisiaj w każdej dystrybucji systemu GNU/Linux (może poza niektórymi dystrybucjami miniaturowymi). W Sieci jest rzecz jasna dostępny na serwerze Free Software Foundation i jego mirrorach. W chwili pisania artykułu dostępna była wersja Bash-2.04.

Jak już powiedziano, Bash jest powłoką opartą o Sh. Jednak poza tym, że większość funkcji dziedziczy właśnie po tej powłoce, zawiera również szereg niezwykle pożytecznych rozszerzeń zapożyczonych z powłoki Korna (ksh), autorstwa Davida Korna i powłoki C (Csh), autorstwa Billego Joy'a. Free Software Foundation zaprojektowało Basha w taki sposób, by był zgodny ze specyfikacją „IEEE POSIX Shell and Utilities", opracowywaną przez Grupę Roboczą 1003.2 IEEE.

Warto wspomnieć, że choć Bash był zaprojektowany z myślą o uniksopodobnym systemie GNU, to w tej chwili pracuje, podobnie jak całe oprogramowanie GNU, na praktycznie każdej platformie uniksowej, Linuksie a nawet w środowiskach DOS, Windows i OS/2.

3. Czym jest standard POSIX?

Nazwa POSIX, oznaczająca „Portable Operating System Interface (to uniX)" została pierwotnie wymyślona przez Richarda M. Stallmana na oznaczenie rodziny otwartych standardów określających systemy oparte na Uniksie. W tej chwili POSIX jest standardem oficjalnie opracowanym przez IEEE i specyfikuje wiele aspektów takich systemów. Standaryzacja obejmuje na przykład wywołania funkcji biblioteki standardowej C jak również szereg podstawowych aplikacji użytkowych takich jak powłoki czy narzędzia administracyjne.

Zachowanie powłoki systemowej zostało wyspecyfikowane we wspomnianym powyżej standardzie „POSIX Shell and Utilities". Standard obejmuje między innymi najważniejsze aspekty składni i poleceń interpretera (powłoki) i podaje jakie polecenia (np. *cd*, *exec*) powinny być wbudowane w powłokę. Wyspecyfikowane są również programy i funkcje używane przez skrypty powłoki, takie jak *sed*, *tr*, *awk*, czy funkcje wbudowane takie jak *test* czy *kill*. Standard obejmuje także specyfikację interfejsów takich jak `system()` czy `getenv()`, jak również specyfikację narzędzi pomocniczych takich jak `yacc`, któremu w przypadku GNU odpowiada GNU Bison.

W Bashu zostały zaimplementowane funkcje zgodnie ze wspomnianą powyżej specyfikacją. Poza tym ma on szereg funkcji nie objętych standardem, lub względem standardu udoskonalonych. Możliwe jest wymuszenie pracy zgodnej z POSIXem w trakcie startu powłoki, poprzez wywołanie jej z opcją `-posix`. Tryb zgodności z POSIXem można również włączyć w trakcie pracy powłoki poprzez ustawienie opcji `set -o posix` lub zmiennych powłoki `$POSIXLY_CORRECT`, lub `POSIX_PEDANTIC`.

4. Podstawowe funkcje Basha

Jak już powiedziano, Bash jest oparty na trzech popularnych powłokach uniksowych, czyli Sh, ksh, i Csh. W związku z tym nie ma większego sensu szczegółowe omawianie jego funkcji, lecz jedynie wskazanie jakie funkcje zostały zapożyczone z tych powłok i jakie Bash ma względem nich rozszerzenia czy udoskonalenia.

4.1. Funkcje Sh

Większość funkcji Bash dziedziczy po powłoce Bourne'a – Sh. Bash tak samo obsługuje przekierowania strumieni we/wy, czy rozszerzanie zmiennych i parametrów (ang. *expansion*). Posiada wszystkie funkcje sterujące i warunkowe takie jak *for*, *while*, czy *if* i pozostałe występujące w Sh. Gramatyka Basha jest pełną implementacją gramatyki Sh włącznie z funkcjami powłoki *function*. Standardowe zmienne określające zachowanie powłoki, takie jak `PS1`, `IFS` czy `PATH` są interpretowane identycznie jak w Sh.

Oprócz podobieństw są jednak pewne różnice w funkcjach. Tam gdzie standard POSIX odbiega od oryginalnego Sh, Bash jest zgodny z tym pierwszym. W Bashu można znaleźć szereg funkcji będących rozszerzeniem możliwości Sh, które w Sh nie występują. Przykładem może być polecenie wbudowane *command* czy rozbudowane polecenie *test*. W Bashu zaimplementowano nowe polecenie *$(...)* będące odpowiednikiem konstrukcji `...`. Poza tym są również możliwości definiowania lokalnych zmiennych w funkcjach powłoki, wyłączania, włączania i podstawiania nowych wersji poleceń wbudowanych.

Trzeba przypomnieć, że pisząc skrypty powłoki trzeba pamiętać o różnicach pomiędzy Bashem a Sh. Jeżeli skrypt wykorzystuje zaawansowane możliwości Basha musi się zaczynać od sekwencji `#!/bin/bash`.

4.2. Funkcje Csh

Powłoka C jest źródłem mechanizmu historii linii poleceń powłoki dostępnego przez polecenie: *! polecenie*. Z tej powłoki przeniesiono również mechanizm rozszerzania zawartości nawiasów klamrowych (ang. *brace expansion*), mechanizm podstawiania katalogu domowego użytkowników przy pomocy tyldy (`użytkownik`) a także polecenia *pushd*, *popd* i `dirs` Przykład użycia *brace expansion* jest pokazany poniżej:

```
$ mkdir /tmp/new-{alfa,beta,gamma}-dir
$ ls /tmp
new-alfa-dir
new-beta-dir
new-gamma-dir
$ echo ba-{b,c,d}-sh
ba-b-sh ba-c-sh ba-d-sh
```

4.3. Funkcje ksh

Standard POSIX, a za nim Bash przejął od ksh szereg usprawnień, wśród których jednym z najważniejszych jest kontrola zadań. Występujące w Bashu polecenie *jobs* jest przejęte z ksh.

Bash ma również zmienne `RANDOM` i `REPLY`. Pierwsza z nich zwraca liczbę pseudo losową gdy się wyświetli jej wartość, lub inicjalizuje generator gdy się jej przypisze nową wartość. Zmienna `REPLY` jest domyślną zmienną przechowującą wartość zwracaną przez polecenie *read*.

Wbudowane polecenie *typeset* pozwala na nadawanie atrybutów zmiennym powłoki. Przykładowo można nadawać atrybut „tylko do odczytu" (*readonly*).

Z powłoki Korna został także przejęty mechanizm operacji arytmetycznych (ang. *arithmetic evaluation*). Zmienne powłoki mogą być używane jako operandy operatorów arytmetycznych, oraz przechowywać wyniki operacji. Zaimplementowano prawie wszystkie operatory arytmetyczne występujące w języku ANSI C, wraz z kolejnością wykonywania działań. Wykonywanie operacji arytmetycznych jest możliwe przy pomocy polecenia *$((...)*. Przykład zastosowania może być następujący:

```
$ echo $((2 + 3 * 6))
20
$ A=5 ; B=3 ; echo C=$(($A * $B + 10))
C=25
```

Zamiast polecenia *$((...))* można użyć odpowiadającego mu polecenia *$[...]*. Obydwa polecenia można zagnieżdżać.

Innym rozszerzeniem wziętym z ksh jest rozbudowany mechanizm aliasów. Pozwalają one nadawać poleceniom nowe nazwy, np. `alias del=rm`, a także domyślne parametry wywołania, np. `alias ls="/bin/ls -lF"`.

5. Uruchamianie Basha

Istnieje szereg różnych możliwości uruchamiania powłoki Bash. Najczęściej powłoka jest uruchamiana automatycznie po zalogowaniu się, jeżeli została ustawiona jako domyślna. Wiele programów, a szczególnie skrypty powłoki są uruchamiane po uprzednim stworzeniu nowego procesu powłoki. W praktyce każdy program uruchamiany ręcznie z interaktywnej sesji Basha jest uruchamiany poprzez nową kopię powłoki. W niektórych z tych sytuacji użytkownik ma możliwość podania parametrów z jakimi wywoływany jest Bash. W związku z tym, na pracę nowo uruchomionego procesu Bash mają wpływ z jednej strony opcje z jakimi został wywołany, a z drugiej pliki konfiguracyjne.

Podstawowe wywołanie powłoki ma postać:

```
bash [opcje] [plik]
```

Argument `plik` jest opcjonalny i oznacza nazwę pliku, który powłoka ma zinterpretować, czyli mówiąc popularnie skryptu. Opcje z jakimi można wywołać powłokę są szczegółowo opisane w podręczniku `bash(1)`. Wśród nich na uwagę zasługują między innymi:

- `-c napis` lista poleceń do wykonania jest czytana z `napisu`,
- `-r` lub `- -restricted` uruchomiona powłoka pracuje w trybie ograniczonym (ang. *restricted shell*),
- `-i` powłoka będzie pracowała jako interaktywna,

- `- -login` powłoka będzie pracowała jak powłoka z której użytkownik się logował (ang. *login shell*),
- `- -noediting` wyłączenie biblioteki GNU Readline,
- `- -noprofile` pominięcie plików `profile`,
- `- -norc` pominięcie plików `rc`,
- `- -rcfile plik` podanie *explicite* pliku `rc`,
- `- -posix` wymuszenie trybu zgodności z POSIX.

Zachowanie powłoki można konfigurować poprzez szereg różnych plików startowych. To, które z nich są uruchamiane zależy od tego czy powłoka ma być interaktywna czy powłoką logowania. Kolejność uruchamiania tych plików jest następująca:

- interaktywna powłoka logowania – czytane są kolejno pliki: `/etc/profile`, `/.bash_profile`, `/.bash_login` i `/.profile`,
- powłoka interaktywna, nie będąca powłoką logowania – czytany jest tylko plik `/.bashrc`,
- powłoka nieinteraktywna – czytany jest plik, którego nazwa jest zawarta w zmiennej `$BASH_ENV`,
- jeżeli Bash jest uruchamiany jako Sh (na przykład poprzez link symboliczny) to uruchamiane są pliki `/etc/profile` i `/.profile`.

Po zakończeniu pracy Bash czyta plik `/.bash_logout`.

6. Nowe możliwości

Po omówieniu sposobów uruchamiania warto przedstawić najważniejsze cechy Basha, które sprawiają, że jest niezwykle wygodnym narzędziem, szczególnie w porównaniu do standardowego Sh.

6.1. Nowe polecenia i zmienne

W Bashu wprowadzono lub ulepszono kilka poleceń wbudowanych, które znacznie zwiększają jego możliwości.

Przede wszystkim dostępne są polecenia pozwalające na manipulowanie poleceniami wbudowanymi powłoki. *enable* pozwala na arbitralne włączanie i wyłączanie (z opcją `-n`) poleceń wbudowanych, na przykład:

```
$ type kill
kill is a shell builtin
$ enable -n kill
$ type kill
kill is /bin/kill
$ enable kill
$ type kill
kill is a shell builtin
```

Jak widać, polecenie *enable* pozwoliło na zablokowanie wbudowanego polecenia *kill* i wymuszenie uruchamiania standardowego programu.

Użyte w powyższym przykładzie polecenie *type* jest ciekawym uzupełnieniem zewnętrznego polecenia *which*. *type* pozwala na wyświetlenie rodzaju wpisanego polecenia. Dobrze prezentuje to poniższy przykład:

```
$ type bash
bash is /bin/bash
$ type type
type is a shell builtin
```

Polecenie *ulimit* umożliwia kontrolę zasobów przydzielonych użytkownikowi powłoki w systemach operacyjnych, które to umożliwiają. Poniżej jest przedstawione modyfikowanie ograniczenia na ilość procesów tworzonych przez daną powłokę:

```
$ ulimit -a
core file size (blocks)     0
data seg size (kbytes)      unlimited
file size (blocks)          unlimited
max locked memory (kbytes)  unlimited
max memory size (kbytes)    unlimited
open files                  1024
pipe size (512 bytes)       8
stack size (kbytes)         8192
cpu time (seconds)          unlimited
max user processes          256
virtual memory (kbytes)     unlimited
$ ulimit -u 2
$ bash
bash: fork: Resource temporarily unavailable
```

Jak widać zmniejszenie tego limitu, w tym przykładzie do nieco nierozsądnej wartości, spowoduje zablokowanie możliwości uruchamiania kolejnych procesów. Odpowiednie wykorzystanie tego mechanizmu może przyczynić się do poprawienia bezpieczeństwa systemu w niektórych sytuacjach.

Polecenia *popd*, *pushd* i *dirs* pozwalają, podobnie jak w Csh na używanie stosu katalogów. Przykład użycia:

```
$ cd /
$ pushd /bin ; pwd
/bin /
/bin
$ pushd /usr/bin ; pwd
/usr/bin /bin /
/usr/bin
$ pushd /usr/local/bin ; pwd
/usr/local/bin /usr/bin /bin /
/usr/local/bin
$ popd ; pwd
/usr/bin /bin /
/usr/bin
```

```
$ popd ; pwd
/bin /
/bin
$ popd ; pwd
/
/
```

Jak widać *pushd* pozwala na przechodzenie do kolejnego, nowego katalogu i umieszczanie go na stosie, a *popd* przejście do następnego katalogu ze stosu i usunięcie go. `dirs` wyświetla aktualną zawartość stosu.

6.2. Podstawianie procesów

Ciekawą możliwością uruchamianie procesów przez Bash jest tak zwane podstawianie procesów (ang. *process substitution*). Jest ono podobne do podstawiania poleceń, uruchamianego poprzez polecenie `$(...)` lub `‘...‘`. Jednak Bash zamiast wstawiania w linię poleceń tego, co pojawia się na standardowym wyjściu uruchamianego w tle procesu, otwiera do niego potok. Przykład:

```
cmp <(ls /tmp/a) <(ls /tmp/b)
/dev/fd/63 /dev/fd/62 differ: char 1, line 1
```

Bash uruchamia 1. proces i otwiera do niego potok, następnie uruchamia drugi i jego wyjście również kieruje do polecenia, w tym przypadku `cmp` a następnie wykonuje samo polecenie wykorzystując dane zwrócone przez uruchomione w tle procesy. Ta metoda uruchamiania procesów tle nie podlega ograniczeniom polecenia `‘...‘`.

6.3. Prezentacja systemu plików

W związku z istnieniem w uniksowych systemach plików niezwykle użytecznego mechanizmu linków symbolicznych czyli dowiązań, pojawiają się dwie możliwości poruszania się po katalogach, które są tylko dowiązaniami symbolicznymi. Problem prezentuje poniższy przykład:

```
$ cd /tmp
$ mkdir a
$ mkdir a/b
$ ln -s a/b c
$ cd c
$ pwd
/tmp/c
$ /bin/pwd
/tmp/a/b
```

Jak widać po przejściu do katalogu `/tmp/c` będącego jedynie dowiązaniem symbolicznym do fizycznie istniejącego katalogu `/tmp/a/b`, wbudowane polecenie Basha *pwd* podaje *logiczny* katalog bieżący, „podążający” za dowiązaniami w sposób niewidoczny dla użytkownika. Tymczasem rzeczywisty, *fizyczny*, katalog bieżący jest inny. Podobnie będą się zachowywały wszystkie mechanizmy dopełniania (ang. *completition*) ścieżek, nazw plików i katalogów. Najczęściej takie zachowanie powłoki jest wygodne. Niekiedy jednak może stanowić problem i wtedy można je odłączyć przy pomocy polecenia `set -o physical`.

6.4. Konfigurowanie znaku zachęty

Jedną z bardzo często używanych możliwości Basha jest możliwość zmiany wyglądu standardowego znaku zachęty, czyli prompta. Bash używa 4 różnych promptów w zależności od sytuacji. Ich wygląd jest określany przez zawartość zmiennych `PS1`, `PS2`, `PS3` i `PS4`. Najczęściej widoczny jest oczywiście 1. prompt i to jego wygląd najczęściej jest modyfikowany przez twórców dystrybucji GNU/Linuksa.

Bash pozwala na modyfikowanie znaku zachęty w dużym zakresie. Po pierwsze, pozwala na użycie specjalnych znaków z backslashem wyświetlających różne parametry sesji i systemu. Po drugie, zmienne `PS` mogą zawierać dowolne zmienne powłoki, wyrażenia arytmetyczne, czy wywołania innych poleceń. Wybrane znaki, poprzedzane backslashem są podane poniżej:

d data,

h nazwa hosta,

H pełna nazwa domenowa hosta,

j liczba zadań pracujących w powłoce,

n znak nowej linii,

r znak powrotu karetki,

l nazwa urządzenia terminalowego danej sesji,

t godzina w formacie 24 godzinnym,

T godzina w formacie 12 godzinnym,

u nazwa użytkownika,

w aktualny katalog roboczy,

W ostatni człon aktualnego katalogu roboczego,

! numer danego polecenia w całej historii poleceń,

numer danego polecenia w aktualnej sesji,

$ znak identyfikujący sesję, `#` w przypadku administratora, `$` w przypadku użytkownika.

Kilka bardziej i mniej typowych przykładów zmiany prompta jest pokazane poniżej:

```
bash-2.01$ export PS1="\$ "
$ export PS1="[\u@\h \w]"
[gjn@enterprise ~/Job/LaU/8-Bash]export PS1="[\u@\h \W] "
[gjn@enterprise 8-Bash] export ="(\t)\h\$ "
(06:53:18)enterprise$ export PS1="---\u@\h---\w----------\t---\n\$ "
---gjn@enterprise---~/Job/LaU/8-Bash----------06:55:02---
```

Drugi znak zachęty jest wyświetlany w sytuacjach takich jak pominięcie zamykającego apostrofu lub cudzysłowu:

```
---gjn@enterprise---~/Job/LaU/8-Bash----------07:00:52---
$ export PS2="--->? "
---gjn@enterprise---~/Job/LaU/8-Bash----------07:01:17---
$ echo "Napis
--->? dalszy ciąg
--->? i jeszcze coś."
Napis
dalszy ciąg
i jeszcze coś.
---gjn@enterprise---~/Job/LaU/8-Bash----------07:01:40---
```

W czasach, gdy było poniżej 5 dystrybucji GNU/Linuksa każda miała swój łatwy do rozpoznania, oryginalny znak zachęty, konfigurowany w sposób podany powyżej. Było rzecz jasna przedmiotem sporu zwolenników każdej dystrybucji, który z nich jest lepszy. Obecnie, gdy dystrybucji są dziesiątki, użytkownicy przestali zwracać uwagę na znaki zachęty i znaleźli sobie nowe tematy sporne.

6.5. Historia linii poleceń

Niezwykle przydatną cechą Basha jest zachowywanie historii linii poceń, pozwalającej na szybkie powtarzanie poleceń podanych w danej i poprzednich sesjach. Historia ta jest zapisywana domyślnie w pliku `/.bash_history`. Inny plik można podać przy pomocy zmiennej `HISTFILE`, a maksymalny rozmiar tego pliku przez zmienną `HISTSIZE`. Zmienna `HISTCONTROL` pozwala na ograniczanie poleceń, które są wpisywane do historii.

Polecenie wbudowane `history` pozwala na zarządzanie plikiem historii. Możne je wywoływać z szeregiem opcji, między innymi:

n opcjonalna liczba pozwala na wyświetlenie wybranej liczby linii z pliku historii,

-c umożliwia wyczyszczenie historii,

-d m usuwa wpis numer `m` z historii,

-r wczytuje zawartość pliku historii,

-w zapisuje aktualną historię sesji do pliku.

W trakcie pracy interaktywnej, historia poleceń danej sesji jest przechowywana w pamięci. Dopiero po zakończeniu pracy jest dopisywana do pliku historii. Rzecz jasna każdy z równocześnie pracujących egzemplarzy powłoki ma wspólną tylko tę część historii, która była w pliku podczas startu powłoki, reszta jest przechowywana w pamięci. Dlatego mając dwie równocześnie pracujące sesje, nie można w jednej mieć dostępnej aktualnej historii drugiej. Przykład pokazujący użycie polecenia jest zaprezentowany poniżej. Wykorzystuje on znaki formatujące prompt tak, by wyświetlał informacje związane z historią.

```
$ export PS1="(\#/\!) \$ "
(1/501) $ history 5
  497  dalszy ciąg
  498  i jeszcze coś."
  499  fc
  500  fc
  501  history 5
```

```
(2/502) $ history -c
(3/3) $ history -r
(4/504) $
```

Należy wspomnieć, że zapisywanie historii danej sesji powinno się czasami wyłączać w związku z bezpieczeństwem. Nie zawsze użytkownik chce by ktoś nieuprawniony, taki jak włamywacz, czytał historię jego sesji, która może zawierać potencjalnie poufne informacje, takie jak na przykład maszyny na których otwierało się sesje. Historię można wyłączyć poleceniem `set +o history`, a włączyć ponownie zmieniając znak `+` na `-` w poprzednim poleceniu.

Bash ma również bardzo zaawansowany mechanizm (ang. *history expansion*) dostępu do znajdujących się w historii. To zagadnienie jest szczegółowo opisane w części *HISTORY EXPANSION* w podręczniku *bash(1)*. Można tu jedynie podać przykłady kilku najbardziej przydatnych poleceń:

!n uruchomienie `n`-tego polecenia z historii,

!-n uruchomienie polecenia o `n` wcześniejszego z historii,

!! uruchomienie poprzedniego polecenia z historii,

!napis uruchomienie 1. wcześniejszego polecenia z historii, zaczynającego się od `napisu`,

!?napis uruchomienie 1. wcześniejszego polecenia z historii, zawierającego `napis`,

^**napis1**^**napis2**^ uruchomienie poprzedniego polecenia z historii, ale zmieniając występujący w nim `napis1` na `napis2`.