

5. Interpreter poleceń

Wstęp

System operacyjny musi zapewnić użytkownikom możliwość wydawania poleceń. Może w tym celu zastosować interfejs tekstowy lub graficzny. Interfejs tekstowy, nazywany interpreterem poleceń, umożliwia wpisanie nazwy polecenia, opcji i argumentów oraz oglądanie wyników niektórych poleceń. Interpretery są stosowane w różnej formie w wielu systemach operacyjnych. W niektórych przypadkach stanowią jedyny interfejs użytkownika (np. w MS DOS), w innych mogą być stosowane równolegle z interfejsem graficznym (np. Unix, Linux, Windows). Systemy Unix i Linux oferują szeroki wybór różnych interpreterów poleceń, określanych również jako powłoki.

W wykładzie 5 omawiamy działanie programu **bash**, który jest podstawowym interpreterem systemu Linux. Opisujemy możliwości edycji polecenia przez użytkownika oraz sposób wartościowania i wykonywania polecenia przez powłokę. Przedstawiamy sposoby konfiguracji interpretera w celu dostosowania jego działania do potrzeb konkretnego użytkownika. Omawiamy parametry i opcje powłoki oraz zestaw znaków, posiadających specjalne znaczenie.

5.1. Przegląd interpreterów

W systemach UNIX i Linux **interpreter** poleceń określany jest również jako **powłoka** (ang. *shell*). Nazwa wywodzi się stąd, że program ten pełni rolę warstwy pośredniczącej między użytkownikiem a systemem operacyjnym. W dalszej części podręcznika będziemy posługiwać się zamiennie obydwojema określeniami.

Pierwszym interpreterem napisanym dla systemu UNIX był **sh**, czyli powłoka Bourne'a (ang. *Bourne shell*). Program ten odznacza się dużą szybkością działania i z tego względu jest wciąż powszechnie wykorzystywany do uruchamiania skryptów. Jego funkcjonalność jest niestety mocno ograniczona podczas pracy interaktywnej ze względu na brak licznych udogodnień, które oferują nowocześniejsze powłoki.

Znacznie wygodniejszy jest drugi klasyczny interpreter **cs**h (ang. *C shell*). Program **cs**h oferuje m.in. historię poleceń, aliasy, i sterowanie pracami. Składnia złożonych poleceń jest bardzo zbliżona do języka C (w którym został napisany) i różni się znacząco od składni stosowanej w **sh**.

Trzecim klasycznym interpreterem jest **ksh** (ang. *Korn shell*), stanowiący rozszerzenie programu **sh** o wiele nowych funkcji. Z założenia, **ksh** przeznaczony był dla administratorów i raczej nie zdobył uznania zwykłych użytkowników ze względu na skomplikowaną składnię.

Żaden z wymienionych programów nie jest dostępny w klasycznej wersji w systemie Linux.

Obecnie największą popularność cieszą się nowoczesne interpretery łączące duże możliwości z łatwością obsługi. Należy tu wymienić powłoki **zsh** i **bash** (ang. *Bourne Again shell*), powstałe z rozwinięcia **sh**, oraz powłokę **tcsh**. Powłoka **zsh** staje się powoli standardem systemu UNIX, zaś **bash** jest od samego początku podstawowym i domyślnym interpreterem Linuxa.

W dalszej części tego wykładu opisujemy powłokę **bash**.

5.2. Pliki konfiguracyjne

Powłoka sh

W przypadku najprostszej powłoki **sh**, pliki konfiguracyjne odczytuje wyłącznie powłoka logowania (ang. *login shell*), czyli powłoka uruchomiona po zalogowaniu użytkownika. Kolejność plików jest następująca:

- /etc/profile
- ~/.profile
- /etc/login
- ~/.login

Powłoka bash

Powłoka **bash** może wykorzystać kilka plików konfiguracyjnych, odczytywanych w zależności od sposobu uruchomienia. Powłoka uruchomiona po zalogowaniu użytkownika (ang. *login shell*) odczytuje kolejno zawartość następujących plików:

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile

Każda powłoka interaktywna z wyjątkiem powłoki logowania odczytuje zawartość plików:

- /etc/bashrc
- ~/.bashrc

Powłoka zsh

Powłoka **zsh** daje jeszcze większą swobodę konfiguracji. Powłoka uruchomiona po zalogowaniu użytkownika odczytuje kolejno zawartość następujących plików:

- /etc/zshenv
- ~/.zshenv
- /etc/zprofile
- ~/.zprofile
- /etc/zshrc
- ~/.zshrc
- /etc/zlogin
- ~/.zlogin

Każda kolejna powłoka odczytuje zawartość plików:

- /etc/zshenv
- ~/.zshenv

Każda powłoka interaktywna odczytuje ponadto:

- /etc/zshrc
- ~/.zshrc

5.3. Działanie powłoki

Edycja poleceń

Powłoki **bash** i **zsh** dają bogate możliwości edycji poleceń. Pozwala łatwo modyfikować wpisane polecenie poprzez przemieszczanie się kursorami \leftarrow i \rightarrow w wierszu, usuwanie niepoprawnych znaków i wstawianie nowych. Jest to szczególnie przydatne do poprawiania długich poleceń złożonych z wielu wyrazów, w których błąd popełniono na samym początku. Opisany poniżej mechanizm historii pozwala przywołać takie polecenie, poprawić i ponownie wykonać.

Interpreter umożliwia ponadto uzupełnianie nazw poleceń i plików podczas edycji wiersza polecenia. Po wprowadzeniu początkowego fragmentu nazwy polecenia można wcisnąć **[Tab]**, aby powłoka odnalazła i uzupełniła nazwę. Poszukiwania prowadzone są wśród wbudowanych poleceń oraz w katalogach wymienionych w ścieżce poszukiwań programów **PATH**. Podobnie odbywa się uzupełnianie nazw plików z tym, że poszukiwania dotyczą tylko jednego katalogu wskazanego w początkowym fragmencie nazwy ścieżkowej pliku (domyślnie jest to katalog bieżący). W obydwu przypadkach, jeśli interpreter odnajdzie kilka pasujących nazw, to wyświetla ich listę dając użytkownikowi możliwość wyboru. Po zakończeniu edycji użytkownik zatwierdza polecenie wciskając **[Enter]**.

Wartościowanie i wykonywanie poleceń

Każde polecenie wydane przez użytkownika jest analizowane przez interpreter, rozwijane do pełnej postaci i dopiero wykonywane. Interpreter wykonuje kolejno następujące operacje:

- podstawia wartości zmiennych (interpretacja znaku **\$**),
- podstawia wyniki poleceń (interpretacja znaków **`**),
- dokonuje podziału na nazwę polecenia i argumenty (interpretacja separatorów),
- wykonuje przekierowanie strumieni danych,
- rozwija nazwy plików,
- rozpoznaje i wykonuje polecenie.

Jeśli interpreter rozpozna nazwę funkcji lub wbudowanego polecenia, to sam je wykonuje. Jeżeli rozpozna nazwę programu lub skryptu, to uruchamia go jako nowy proces potomny. W przypadku skryptu nowym procesem jest powłoka potomna, która wykonuje polecenia ze skryptu.

Każde polecenie zwraca swój status zakończenia. Przyjęto konwencję, że zerowy status oznacza pomyślne zakończenie, zaś wartość niezerowa jest zwracana, gdy polecenia nie udało się wykonać. Jeśli polecenie zostało przerwane przez sygnał o numerze **n**, to zwracana jest wartość **128 + n**.

Historia wydawanych poleceń



Interpreter może przechowywać listę ostatnio wydanych poleceń i udostępniać je użytkownikowi do powtórnego wykorzystania. Mechanizm ten nosi nazwę historii poleceń.

Oryginalna powłoka Bourne'a sh nie ma wbudowanego mechanizmu historii.

Powłoka **bash** udostępnia historię poleceń, jeśli ustawiona jest opcja **history** (patrz punkt 2.5). Liczbę pamiętanych poleceń określa zmienna **HISTSIZE** ustawiana domyślnie na wartość 500. Po uruchomieniu, **bash** zaczyna pracę z pustą listą historii, ale może ją zainicjować z pliku określonego zmienną **HISTFILE**. Domyślnie w powłoce **bash** jest to plik `~/.bash_history`. Maksymalną liczbę linii w pliku określa zmienna **HISTFILESIZE**.

Aktualną listę historii można wypisać poleceniem **history**. Każde z zapamiętanych poleceń poprzedzone jest swoim numerem. Posługując się tym numerem lub początkowym fragmentem nazwy, można przywołać konkretne polecenie w następujący sposób:

- !**identyfikator - ogólna postać odwołania do historii poleceń,
- !!** - przywołuje ostatnie polecenie,
- !n** - przywołuje polecenie o numerze n,
- !-n** - przywołuje polecenie położone n pozycji od końca listy czyli od ostatniego polecenia,
- !string** - przywołuje polecenie rozpoczynające się ciągiem znaków string.

Istnieje też drugi sposób korzystania z historii poleceń, który w większości przypadków jest znacznie wygodniejszy. Powłoka **bash** umożliwia przeglądanie listy w linii poleceń przy użyciu kursorów  oraz . Przywołane polecenie można potem modyfikować jak opisano w punkcie 1.3.

Alias

Istotne ułatwienie do wielokrotnego posługiwania się złożonymi poleceniami wnoszą **aliasy**. Umożliwiają zastąpienie wielocłónowego polecenia pojedynczym wyrazem. Nową nazwą można się posługiwać tak, jak każdym innym poleceniem, dodając opcje i argumenty. Nie ma jednak możliwości wstawienia zmiennych argumentów do tekstu samego aliasu.

Mechanizm aliasów jest dostępny w większości interpreterów (z wyjątkiem sh), ale tylko w tych egzemplarzach, które zostały uruchomione w trybie interaktywnym. W związku z tym aliasy nie są na ogół dostępne w skryptach.

Każda uruchomiona powłoka przechowuje niezależnie własną listę aliasów zdefiniowanych przy pomocy wbudowanego polecenia:

```
alias [-p] [nazwa=wartość ...]

alias [nazwa ...]
```

Polecenie wywołane bez argumentów lub z opcją **-p** wypisuje pełną listę aliasów. Jeśli podane są tylko nazwy aliasów, powłoka wypisuje ich wartości. Dla argumentów w postaci nazwy z przypisaną wartością tworzone są nowe aliasy. Jeżeli przypisywana wartość składa się z kilku wyrazów (np. polecenie z opcjami i argumentami), to należy podać ją w cudzysłowach bądź apostrofach.

Przykład

```
alias ll="ls -l"
alias l='ls -FC'
```

Aliaszy zwykle ustawiane są w plikach konfiguracyjnych odczytywanych przez powłoki interaktywne, czyli w plikach: `/etc/bashrc` i `~/bashrc` lub `/etc/zshrc` i `~/zshrc`.

Usuwanie aliasów umożliwia polecenie:

```
unalias [-a] [nazwa ...]
```

5.4. Konfigurowanie powłoki

Sposób działania interpretera jest w dużym stopniu ustalany przez **parametry** i **opcje**.

Parametry

Parametry są instancjami przechowującymi wartości. Mogą być reprezentowane przez nazwę w postaci łańcucha znaków, liczbę naturalną lub jeden ze znaków specjalnych.

Parametry reprezentowane przez nazwę określane są jako **zmienne**.

Parametry reprezentowane przez liczby naturalne noszą nazwę **parametrów pozycyjnych** i przechowują argumenty wywołania powłoki.

Parametry specjalne mają postać pojedynczych znaków specjalnych a ich wartości są automatycznie ustawiane i modyfikowane dynamicznie przez powłokę.

Odwołanie do wartości dowolnego parametru następuje za pomocą znaku \$ w następujący sposób:

```
$parametr
```

```
${parametr}
```

Postać z nawiasami jest konieczna, gdy w poleceniu bezpośrednio za nazwą parametru chcemy umieścić inne znaki oraz gdy nazwa liczbowa składa się z więcej niż jednej cyfry.

Przykład

```
$ x=abc
$ echo $x
abc
$ echo $xy
$ echo ${x}y
abcy
```

Ustawianie wartości zależy od typu parametru. Ustawianie wartości zmiennych odbywa się za pomocą jednego z trzech poleceń:

```
zmienna=[wartość]
```

```
typeset zmienna=[wartość]
```

```
declare zmienna=[wartość]
```

Jeśli wartość zostanie pominięta, to zmiennej zostanie przypisany zerowy łańcuch znaków.

Usunięcie zmiennej umożliwia polecenie:

```
unset zmienna
```

Zmienne można podzielić na dwie kategorie:

1. **zmienne środowiska**,
2. **zmienne lokalne powłoki**.

Listę wszystkich zmiennych, zarówno lokalnych jak i środowiska, uzyskamy poleceniem:

```
set
```

Zmienne środowiska

Każdy tworzony proces, w tym również każda powłoka, otrzymuje nowe **środowisko** w postaci tablicy zawierającej ciągi znaków `zmienna=wartość`. Każdy taki ciąg oznacza utworzenie nowej **zmiennej środowiska**.

Ten początkowy zestaw zmiennych środowiska jest **dziedziczony** po procesie macierzystym, którym najczęściej jest powłoka uruchamiająca nowy program.

W trakcie działania procesu środowisko może być modyfikowane poprzez usuwanie i dodawanie nowych zmiennych oraz przez zmiany ich wartości. Wszystkie modyfikacje znajdują odzwierciedlenie we wspomnianej tablicy. Zmiany te jednak nie przenoszą się do procesu macierzystego. Zasada ta dotyczy również procesów potomnych danego procesu utworzonych przed modyfikacją środowiska.

Zmienne środowiska wyróżnia więc to, że nie są związane tylko z lokalną powłoką, ale są dziedziczone przez wszystkie procesy potomne tworzone przez powłokę.

Są wykorzystywane przez różne procesy a nie tylko przez proces powłoki. Nie można ich jednak traktować jako zmienne globalne, gdyż nie stają się widoczne dla wszystkich procesów.

Całe środowisko można wyświetlić przy pomocy poleceń:

```
printenv
```

lub

```
env
```

Wartość pojedynczej zmiennej wypisze polecenie:

```
echo $zmienna
```

Dodanie do środowiska nowej zmiennej polega na wyeksportowaniu zmiennej lokalnej, której trzeba wcześniej przypisać właściwą wartość:

```
zmienna=wartość
```

```
export zmienna
```

Można to również zrobić w jednym etapie wydając polecenie (w powłokach **bash** i **zsh**):

```
export zmienna=wartość
```

Zmienną środowiska można usunąć poleceniem **unset**, podobnie jak zmienną lokalną.

W tablicy 5.1 zebrano najważniejsze zmienne środowiska definiowane w każdej powłoce i wyjaśniono ich znaczenie dla powłoki i innych procesów.

Tablica 5.1 Najważniejsze zmienne środowiska

Nazwa zmiennej	Znaczenie
LOGNAME	Nazwa zalogowanego użytkownika.
USER	Nazwa zalogowanego użytkownika.
HOME	Katalog domowy zalogowanego użytkownika.
LANG	Język lokalny.
PATH	Ścieżka poszukiwań programów.
CDPATH	Ścieżka poszukiwań katalogów.
MANPATH	Ścieżka poszukiwań dokumentacji systemowej.
LD_LIBRARY_PATH	Ścieżka poszukiwań bibliotek dynamicznych.
PWD	Katalog bieżący.
SHELL	Powłoka uruchamiana po zalogowaniu użytkownika.
TERM	Typ terminala.
MAIL	Skrzynka pocztowa zalogowanego użytkownika.
MAILCHECK	Częstotliwość sprawdzania skrzynki pocztowej.
IFS	Wejściowy separator pola.
TZ	Strefa czasowa.
PS1	Podstawowy komunikat gotowości powłoki wyświetlany, gdy powłoka jest gotowa do przyjęcia kolejnego polecenia.
PS2	Wtórny komunikat gotowości powłoki wyświetlany, gdy powłoka oczekuje na dokończenie polecenia w następnej linii.

Zmienne PS1 i PS2 mogą być również ustawione jako zmienne lokalne. Definiują one sposób, w jaki powłoka zgłasza gotowość do dalszej pracy. Jeśli powłoka jest gotowa do przyjęcia kolejnego polecenia, to wyświetla **komunikat gotowości** zawierający ciąg znaków przypisany zmiennej PS1. Jeśli powłoka wymaga jeszcze dodatkowych danych wejściowych, żeby dokończyć polecenie, to wyświetla ciąg znaków przypisany zmiennej PS2. Podstawowe komunikaty zawierają tylko jeden znak, odpowiednio \$ lub >. Stąd wywodzi się inna nazwa takiego komunikatu - **znak zachęty** powłoki. Najczęściej stosuje się jednak dłuższe komunikaty, które mogą zawierać nazwę komputera, nazwę użytkownika, nazwę bieżącego katalogu czy numer polecenia.

Przykład

Poniżej przedstawiono przykładowe wartości zmiennych PS1 i PS2 oraz ich efekt w postaci komunikatów gotowości powłoki:

```
herkules.apw 1 $ echo $PS1
%m.%n %! $
herkules.apw 2 $ echo $PS2
>
```

Zmienne lokalne powłoki

Zmienne lokalne mają znaczenie wyłącznie dla bieżącej powłoki i tylko w niej są widoczne. Nie podlegają dziedziczeniu i nie mogą być wykorzystane przez inne procesy. Sposób definiowania i usuwania zmiennych opisano powyżej. W tabelicy 5.2 opisane zostały wybrane zmienne powłoki.

Tabela 5.2 Wybrane zmienne lokalne powłoki bash

Nazwa zmiennej	Znaczenie
BASH	Pełna nazwa ścieżkowa interpretera poleceń.
HISTFILE	Nazwa pliku przechowującego historię wydawanych poleceń.
HISTFILESIZE	Maksymalna liczba linii w pliku historii poleceń.
HISTSIZE	Liczba pamiętanych poleceń.

Parametry specjalne

Każda powłoka ustawia automatycznie kilka parametrów specjalnych, których wartości zmieniają się w trakcie jej działania. W tabelicy 5.3 przedstawiamy znaczenie tych parametrów.

Tabela 5.3 Parametry specjalne powłoki bash

Parametr	Znaczenie
#	Liczba parametrów pozycyjnych.
*	Lista parametrów pozycyjnych w postaci jednego ciągu znaków.
@	Lista parametrów pozycyjnych w postaci oddzielnych ciągów znaków.
?	Status zakończenia ostatnio uruchomionego polecenia.
\$	PID bieżącej powłoki.
!	PID procesu ostatnio uruchomionego w tle.
-	Lista opcji bieżącej powłoki.

Parametr **\$?** umożliwia sprawdzenie w skrypcie, czy jakieś polecenie zakończyło się pomyślnie. Parametr **\$\$** zawiera identyfikator PID bieżącej powłoki, który można wykorzystać np. do stworzenia pliku tymczasowego o unikalnej nazwie.

Parametry pozycyjne

Parametry pozycyjne reprezentowane są przez kolejne liczby naturalne. Parametr **\$0** przechowuje nazwę powłoki lub nazwę skryptu. Parametry **\$1**, **\$2**, **\$3** ... przechowują kolejne argumenty. Odwołania do parametrów o wyższych numerach, składających się z kilku cyfr, wymagają użycia nawiasów klamrowych: **\${10}**, **\${11}**... Powłoki **bash** i **zsh** nie ograniczają więc możliwości bezpośredniego dostępu do wszystkich parametrów, podczas gdy w powłoce **sh** istnieje ograniczenie tylko do pierwszych 9 parametrów: **\$1**, ... **\$9** a dostęp do pozostałych otrzymuje się po przesunięciu.

Wszystkie parametry pozycyjne można jednocześnie przesunąć w lewo o dowolną liczbę pozycji za pomocą polecenia:

```
shift [n]
```

Gdy argument nie jest podany, domyślnie następuje przesunięcie wszystkich parametrów o **jedną pozycję**. W rezultacie parametr **\$1** jest usuwany (bezpowrotnie), parametr **\$2** przesuwa się na **\$1**, parametr **\$3** na **\$2** itd. W przypadku gdy parametr **n** jest podany przesuwanie argumentów następuje o **n** pozycji, tzn. **n** pierwszych argumentów jest usuwanych a pozostałe zajmują pozycje zaczynając od **\$1**.

Parametr \$0 nigdy nie podlega przesunięciu i zawsze przechowuje nazwę.

Powłoka ustawia również trzy parametry specjalne związane z parametrami pozycyjnymi. Parametr **\$#** określa liczbę ustawionych parametrów pozycyjnych z wyjątkiem parametru **\$0**. Parametr **\$*** przechowują całą listę parametrów pozycyjnych w postaci jednego ciągu znaków "**\$1 \$2 \$3 ...**", natomiast **\$@** przechowuje tę listę w postaci oddzielnych ciągów "**\$1**" "**\$2**" "**\$3**" ...

Początkowe ustawienie parametrów pozycyjnych można zmienić poleceniem:

```
set arg1 arg2 arg3 ...
```

Polecenie zmienia na raz wszystkie parametry pozycyjne, usuwając stare wartości i przypisując części z nich nowe wartości podane w poleceniu. Nie ma możliwości zmiany tylko wybranych parametrów.

Opcje

Dodatkową możliwość konfiguracji interpretera zapewniają opcje. Można je ustawić podając odpowiednie argumenty przy wywołaniu powłoki albo w trakcie jej działania posłużyć się poleceniem:

```
set -o [opcja ...]
```

Polecenie wydane bez argumentów wypisuje aktualne ustawienia poszczególnych opcji.

Usuwanie opcji umożliwia polecenie:

```
set +o [opcja ...]
```

Większość opcji ma swoje jednoliterowe odpowiedniki, których można użyć przy uruchamianiu powłoki. Niektóre opcje są domyślnie ustawiane przez powłokę. W tablicy 4.4 podano nazwy, skróty literowe oraz znaczenie wybranych opcji.

Tablica 5.4 Wybrane opcje powłoki bash

Nazwa opcji (skrót literowy)	Znaczenie
history	Powłoka udostępnia mechanizm historii. W powłokach interaktywnych jest ustawiona domyślnie.
ignoreeof	Ignorowanie przez powłokę znaku końca pliku (ctrl d).
monitor (-m)	Powłoka udostępnia sterowanie pracami.
noclobber (-C)	Powłoka nie nadpisuje istniejących plików w wyniku przekierowania strumieni (>, >&). Można wymusić nadpisanie poprzez użycie > .
notify (-b)	Powłoka natychmiast informuje o zakończeniu procesu w tle. Gdy opcja nie jest ustawiona, wypisuje informacje po wydaniu następnego polecenia.
noglob (-f)	Powłoka nie rozwija nazw plików zawierających znaki specjalne.
emacs	Powłoka używa stylu edytora emacs do edycji linii poleceń. Dopuszczane jest przemieszczanie się kursorami i wstawianie znaków.
vi	Powłoka używa stylu edytora vi do edycji linii poleceń. Opcja ta wyklucza się wzajemnie z opcją emacs.
xtrace (-x)	Powłoka wypisuje każde polecenie w postaci rozwiniętej przed jego wykonaniem. Umożliwia w ten sposób śledzenie wykonywania poleceń.

5.5. Interpretacja znaków specjalnych

Rozwijanie nazw plików

Każdy interpreter dopuszcza stosowanie w nazwach plików specjalnych znaków, które mogą zastępować jeden lub więcej dowolnych znaków. Otrzymuje się w ten sposób wzorce pasujące do nazw kilku plików. W Tabl. 5.5 przedstawiono zestaw znaków specjalnych powłoki **bash**.

Tablica 5.5 Znaki służące do rozwijania nazw plików

Znak	Znaczenie
?	zastępuje jeden dowolny znak,
*	zastępuje dowolny ciąg znaków,
[]	zastępuje jeden znak z listy podanej wewnątrz nawiasów,
[-]	zastępuje jeden znak z zakresu podanego wewnątrz nawiasów,
[^] lub [!]	zastępuje dowolny znak oprócz tych podanych wewnątrz nawiasów.

Przykład

```
ls *.?  
ls *.[ch]
```

Cytowanie

Cytowanie stosowane jest w celu usunięcia specjalnego znaczenia niektórych znaków i słów. Zapobiega interpretacji tych znaków przez powłokę, przywracając ich podstawowe znaczenie. Niektóre sposoby cytowania dopuszczają pewne wyjątki.

Tablica 5.6 Znaki służące do cytowania

Znak	Znaczenie
\	zapobiega interpretacji następnego znaku,
" "	zapobiegają interpretacji wszystkich znaków zawartych wewnątrz cudzysłowów, z wyjątkiem znaków \$ i ` ,
' '	zapobiegają interpretacji wszystkich znaków zawartych wewnątrz apostrofów,
` `	wykonuje polecenie zawarte wewnątrz znaków i wstawia w to miejsce strumień wyjściowy tego polecenia.

Przykład

```
$ echo \$HOME  
$HOME  
  
$ echo "Katalog bieżący: `pwd`"  
Katalog bieżący: /tmp  
  
$ echo 'Znaki specjalne powłoki: $ ` ` * \'  
Znaki specjalne powłoki: $ ` ` * \
```

Grupowanie poleceń

Interpreter umożliwia wydanie kilku poleceń w jednej linii. Sposób zgrupowania zdecydowanie o tym, w jaki sposób i w jakiej kolejności powłoka wykona te polecenia. Poniżej przedstawiono kilka możliwości połączenia poleceń w jednej linii oraz efekty takiego użycia (skrót **cmd** oznacza dowolne polecenie).

Tablica 5.7 Znaki służące do grupowania poleceń

Znak	Znaczenie
<code>cmd ; cmd ; cmd</code>	powłoka grupuje polecenia na pierwszym planie wykonując je sekwencyjnie,
<code>cmd & cmd & cmd &</code>	powłoka grupuje polecenia w tle wykonując je asynchronicznie,
<code>cmd cmd cmd</code>	powłoka grupuje polecenia wykonując je w potoku,
<code>(cmd ; cmd)</code>	powłoka grupuje polecenia i wykonuje w nowej powłoce,
<code>{ cmd ; cmd ; }</code>	powłoka grupuje polecenia i wykonuje w bieżącej powłoce, tworzy jeden strumień wyjściowy dla całej grupy poleceń i zwraca jeden status zakończenia.

Przykład

```
ls -lR / | more
sleep 50 & ps -f
sleep 50 ; ps -f
```

Przekierowanie strumieni

Powłoka łączy z każdym tworzonym procesem trzy strumienie danych:

1. standardowy strumień wejściowy **stdin**,
2. standardowy strumień wyjściowy **stdout**,
3. standardowy strumień błędów (diagnostyczny) **stderr**.

Każdy ze strumieni związany jest z plikiem zwykłym lub z plikiem urządzenia wejścia/wyjścia. Domyślnym przywiązaniem dla wszystkich strumieni jest plik terminala.

Każdy plik otwarty przez proces otrzymuje kolejny numer zwany **deskryptorem** pliku. Deskryptory **0**, **1** i **2** zarezerwowane są dla plików związanych ze standardowymi strumieniami.

Przed wykonaniem dowolnego polecenia można zmienić domyślne przywiązanie strumieni, czyli przekierować wejście i wyjście z polecenia. Przekierowania strumieni w powłocie **bash** obejmują następujące operacje wymienione w Tabl. 5.8.

Tablica 5.8 Znaki służące do przekierowania strumieni danych

Znak	Znaczenie
[d]<plik	przekierowanie wejścia powoduje otwarcie pliku do czytania z deskryptorem d ; jeżeli d zostało pominięte przyjmowane jest 0,
[d]>plik	przekierowanie wyjścia powoduje otwarcie pliku do pisania z deskryptorem d ; jeżeli d zostało pominięte przyjmowane jest 1,
[d]>>plik	przekierowanie wyjścia powoduje otwarcie pliku do dopisywania z deskryptorem d ; jeżeli d zostało pominięte, przyjmowane jest 1,
[d]<>plik	przekierowanie powoduje otwarcie pliku do czytania i pisania z deskryptorem d ,
&>plik lub >&plik	przekierowanie powoduje skierowanie strumienia wyjściowego i strumienia błędów do tego samego pliku,
[d1]>&d2	przekierowanie powoduje zduplikowanie deskryptora pliku wyjściowego; deskryptor d1 staje się kopią deskryptora d2 i wskazuje na ten sam plik.

Jeżeli numer deskryptora **d** zostanie pominięty, to powłoka domyślnie przyjmie deskryptor **0** (**stdin**) lub **1** (**stdout**) w zależności od rodzaju przekierowania.

Operatory przekierowania można umieścić w linii polecenia zarówno przed jak i za poleceniem. Przekierowania wykonywane są w kolejności występowania licząc od lewej do prawej.

Przykład

```
ps -ef >procesy.txt
ls -lR >pliki.txt 2>bledy.log
```

Inne

Interpreter korzysta jeszcze z kilku innych znaków specjalnych, które zostały już wprowadzone we wcześniejszej części tego podręcznika. Dla przypomnienia przytaczamy je również w tym miejscu.

Tablica 5.9 Inne znaki specjalne

Znak	Znaczenie
spacja, tabulator, znak nowej linii	separatory rozdzielające poszczególne elementy polecenia,
~	katalog domowy użytkownika,
\$	podstawienie wartości parametru (zmiennej),
!	odwołanie do mechanizmu historii.

Bibliografia

- [1] Glass G., Ables K.: Linux dla programistów i użytkowników, Wydawnictwo Helion 2007
(rozdziały: 5, 6)

Nowe pojęcia, definicje i wyrażenia

Termin	Objaśnienie
alias	skrótowa nazwa innego, zwykle wielocłonowego polecenia
historia poleceń	mechanizm przechowywania listy ostatnio wydanych poleceń i udostępniania ich użytkownikowi do powtórnego wykorzystania
parametr	instancja powłoki przechowująca jakąś wartość, reprezentowana przez nazwę w postaci ciągu znaków, liczbę naturalną lub jeden ze znaków specjalnych
powłoka	program interpretera poleceń użytkownika
skrypt	plik tekstowy zawierający listę poleceń dla interpretera poleceń
środowisko	tablica zawierająca ciągi znaków zmienna=wartość , definiujące zmienne środowiska
zmienne środowiska	zmienne procesu podlegające dziedziczeniu przez wszystkie procesy potomne

Zadania do wykładu 5

Zadanie 1

Zmodyfikuj (lub utwórz) lokalne pliki konfiguracyjne powłoki **bash** w celu :

- a) dodania do ścieżki poszukiwań katalogu ~/bin ,
- b) dodania aliasa na polecenie wyświetlania listy wszystkich własnych procesów.

Zadanie 2

Sprawdź jak praktycznie objawia się różnica pomiędzy zmiennymi środowiska i zmiennymi lokalnymi powłoki.

Zadanie 3

Zaproponuj przykłady wyrażeń ilustrujących działanie następujących znaków specjalnych:

- 1. " " (cudzysłów),
- 2. ' ' (apostrof),
- 3. ` ` (odwrotny apostrof),
- 4. \.

Sprawdź jak zastosowanie powyższych znaków wpływa na interpretację znaków \$, ` ` , *. Wyniki przedstaw w tabeli.