

# Wprowadzenie do systemu UNIX

## Interpretery i skrypty cz. 1.

### Wprowadzenie

Zajęcia te mają na celu przedstawienie środowiska interpreterów poleceń w systemach uniksowych. Interpretery służą do wykonywania poleceń wydawanych przez użytkownika systemu, zarówno *interaktywnie* (a więc wpisywanymi w czasie rzeczywistym przez użytkownika z oczekiwaniem natychmiastowego uzyskania rezultatów), jak i *wsadowo* (ang. *batch* – jako tzw. *skrypt*, czyli tekstowy plik z zawartymi w nim poleceniami). Możemy je podzielić na trzy grupy na podstawie rodzaju wykonywanych operacji:

- **interpretery logujące** – wykonujące operację logowania użytkownika i przygotowania dla niego środowiska (instrukcje do wykonania w trakcie uruchamiania takiego interpretera znajdują się w pliku `$HOME/.bash_profile` lub `$HOME/.profile`,
- **interpretery interaktywne** – wykonujące w czasie rzeczywistym komendy wpisywane przez użytkownika (uruchomieniem takiego terminala zarządza się, w przypadku interpretera *Bash*, przez zawartość pliku `$HOME/.bashrc`),
- **interpretery wsadowe** – przetwarzające skrypty użytkownika.

Środowisko interpretera systemowego pozwala też na programowanie w podstawowym zakresie. Możliwe jest definiowanie w nim własnych funkcji, aliasów na polecenia już istniejące oraz zmiennych. Istnieją zmienne trzech typów:

- **zmienne specjalne** – deklarowane w momencie uruchomienia powłoki, posiadają przypisane na stałe funkcje:
  - `$#` – liczba argumentów, z którymi został uruchomiony interpreter (powłoka) lub aktualnie interpretowany skrypt,
  - `$0` – nazwa aktualnie interpretowanego skryptu lub interpretera,
  - `$1` – pierwszy argument skryptu lub wywołania interpretera (analogicznie działają `$2`, `$3`, `$4` itd.),
  - `$*` – lista wszystkich argumentów skryptu lub wywołania interpretera,
  - `$?`  – kod wyjścia (błędu) zwrócony przez ostatnią wywołaną funkcję,
  - `$$`  – PID bieżącego procesu powłoki lub skryptu,
- **zmienne środowiskowe** – przechowują dane pozwalające na sterowanie środowiskiem (interpreterem):
  - `$HOME` – adres katalogu domowego bieżącego użytkownika,
  - `$PATH` – lista ścieżek dostępu do katalogów przechowujących programy, których wykonanie odbywać ma się bez wskazywania do nich ścieżki dostępu,
  - `$HISTFILE` – wskaźnik do pliku przechowującego historię wydanych przez użytkownika poleceń,

- `$HISTSIZE` – liczba wpisów przechowywanych w `HISTFILE`,
- `$SHELLOPTS` – parametry powłoki,
- `$PS1` – formatowanie postaci znaku zachęty (ang. *prompt*),
- `$PS2` – formatowanie postaci znaku zachęty w linii z kontynuacją wydawanego wieloliniowego polecenia,
- `$TMOUT` – czas nieaktywności, po którym następuje zakończenie sesji,
- `$IGNOREEOF` – ustawienie liczby wywołań skrótu **Ctrl-d** (*End of File*), które zostaną zignorowane przed wylogowaniem użytkownika,
- `$LANG` – określenie języka sesji,

- **zmienne programowe** – definiowane przez użytkownika.

`ALIAS` to krótka nazwa zdefiniowana przez użytkownika na wybrane przez niego polecenie, którego nie chce on wpisywać za każdym razem w pełnej formie. Aliasy tworzone są przy użyciu komendy **alias** i usuwane poleceniem **unalias**. Przykład tworzenia aliasu został wskazany poniżej:

```
alias lu='ls -alh'
```

URUCHAMIANIE PROGRAMÓW w interpreterze następuje poprzez podanie ich nazwy oraz ewentualnych jego argumentów. Po wydaniu komendy, polecenie wyszukiwane jest w trzech obszarach:

1. lista poleceń wbudowanych interpretera, np. **echo**, **cd**,
2. lista aliasów,
3. lista programów znajdujących się w katalogach wskazywanych przez zmienną `$PATH`.

Katalog bieżący nie jest domyślnie przeszukiwany przy próbie uruchomienia programu bez podawania ścieżki dostępu do niego. Program z bieżącego katalogu można wywołać przez:

```
./program
```

Aby pominąć sprawdzanie aliasów przy wykonywaniu polecenia, można użyć odwróconego ukośnika (`\` – ang. *backslash*):

```
\program
```

INTERPRETERY pozwalają także na tworzenie samodzielnie zdefiniowanych funkcji i zmiennych użytkownika.

Zmienne tworzone są przy wykorzystaniu składni: `zmienna=wartosc`

Do wyświetlania aktualnie istniejących zmiennych służy polecenie **set**, wyświetlenia wskazanej zmiennej – polecenie **echo**, a usuwanie zmiennych zrealizować można z użyciem komendy **unset**.

W ten sposób stworzone zmienne dostępne są w aktualnym środowisku, nie zostaną one jednak przekazane do środowiska potomnego (uruchomionego interpretera). Eksportu zmiennej, tak by była ona dziedziczona, dokonuje się poleceniem **export**. Do wyświetlenia listy eksportowanych zmiennych można użyć polecenia **env**.

Oczywiście, program uruchomi się tylko wtedy, gdy wywołujący go użytkownik posiada uprawnienie do jego wykonania.

Definiowanie funkcji nie wchodzi w zakres zajęć, jednak można się z nim zapoznać tutaj: <https://ryanstutorials.net/bash-scripting-tutorial/bash-functions.php>

Domyślnie zmienne mogą być modyfikowane. Do zablokowania możliwości edycji zmiennej o danej nazwie służy polecenie **readonly**. Sam atrybut „tylko do odczytu” domyślnie nie jest eksportowany do środowisk potomnych, można to zmienić poprzez wydanie komendy:

```
export readonly ZMIENNA=wartosc
```

## Zadania

1. Zapoznaj się ze składnią komend **set**, **unset** i **env**. Pracując z wykorzystaniem powłoki *Bash*, sprawdź wartości ustawionych zmiennych środowiskowych.

- Czy zmienna środowiskowa **noclobber** jest zdefiniowana? Sprawdź wpływ tej zmiennej środowiskowej na sposób tworzenia, nadpisywania i dopisywania informacji do plików z wykorzystaniem mechanizmów przekierowywania strumieni danych (operatory `>`, `>>`, `>|`).
- Powrót do ustawień pierwotnych. Uwaga: ustawienie wartości zmiennej środowiskowej **SHELLOPTS** następuje poleceniem:  

```
set -o noclobber
```

2. W pliku o nazwie `wyjscia.c` zapisz następujący kod źródłowy programu:

```
#include <stdio.h>
int main () {
    fprintf (stdout, "Standardowe wyjscie\n");
    fprintf (stderr, "Standardowe wyjscie diagnostyczne\n");
    return 0;
}
```

Skompiluj program poleceniem: `gcc -o wyjscia wyjscia.c`. Uruchom program `wyjscia` tak, aby wypisywane przez niego komunikaty zostały zapisane odpowiednio do plików `std.txt` oraz `err.txt`, a potem tak, aby oba komunikaty pojawiły się w jednym pliku o nazwie `razem.txt`.

3. Korzystając z komendy **set**, sprawdź, ile ostatnio wydanych komend pamiętanych jest w *shellu*. Sprawdź, jakie komendy zostały ostatnio wydane. Zmień wartość zmiennej środowiskowej, odpowiadającej za tę wartość, na **20000**.
4. Sprawdź, w jakim pliku pamiętana jest historia poleceń. Sprawdź listę historii poleceń komendą **history**. Wykonaj raz jeszcze ostatnie polecenie rozpoczynające się od konkretnego znaku (składnia: `!polecenie`) oraz polecenie o konkretnym numerze.
5. Istnieje zmienna środowiskowa, której wartość oznacza dopuszczalny czas pozostawienia terminala w bezczynności. Po upływie tego czasu system automatycznie zrywa sesję i odłącza użytkownika od systemu. Uruchom interpreter interaktywny i sprawdź działanie zmiennej **TMOUT**. W jakich jednostkach podawany jest czas bezczynności?

6. Uruchom powłokę *Bash* ponownie. Zmienna środowiskowa **IGNOREEOF**, jeśli jest ustawiona (poleceniem `export IGNOREEOF`), zapobiega kończeniu pracy interpretera poleceń (wylogowywaniu się) przez naciśnięcie kombinacji klawiszy **Ctrl-d**. Niestety, pięciokrotne naciśnięcie **Ctrl-d** powoduje odłączenie użytkownika od systemu. Nadanie zmiennej wartości powoduje, że po tylukrotnym, ile wynosi wartość zmiennej, naciśnięciu **Ctrl-d** użytkownik zostanie od systemu odłączony. Sprawdź działanie zmiennej.
7. Zmienna środowiskowa **PATH** przechowuje ścieżki dostępu do katalogów, w których interpreter poszukuje programów do wykonania. Ze względów bezpieczeństwa nie ma w niej katalogu bieżącego. Zmodyfikuj wartość zmiennej **PATH** tak, aby ścieżka dostępu do katalogu bieżącego znalazła się na końcu ścieżek w tej zmiennej.
8. Postać podstawowego znaku zachęty linii komand definiuje zmienna środowiskowa **PS1**. Znak kontynuacji pojawia się, gdy linia z poleceniem nie została zakończona (na przykład, z powodu niezamknięcia cudzysłowu) lub ostatnim znakiem w linii poprzedniej jest odwróconego ukośnika (`\` – ang. *backslash*). Znak kontynuacji jest zdefiniowany wartością zmiennej środowiskowej **PS2**. Sprawdź wartości obu zmiennych. Ustaw wartość zmiennej **PS1** tak, aby znak zachęty zawierał informacje o nazwie użytkownika, nazwie hosta, czasie w formacie 24-godzinnym oraz ścieżki dostępu do bieżącego katalogu. Zmień również postać **PS2**.
9. Zdarza się, iż wielokrotnie wykonujemy komendę o długiej liście opcji. Komendzie takiej można nadać krótką nazwę, czyli tzw. *alias*.
  - Sprawdź listę już zdefiniowanych aliasów.
  - Dodaj własny, o nazwie **ll** i postaci:
 

```
ls -al --color=always | less
```
  - Dlaczego, pomimo wymuszenia kolorowania nazw plików, pojawiają się one bez kolorów? Popraw alias, aby działał poprawnie.
10. Aby wprowadzone w środowisku interpretera polecenia zmiany były aktywne, należy je zapisać w odpowiednich plikach konfiguracyjnych, różnych dla różnych rodzajów interpretera (logujący, interaktywny).
  - Utwórz lub zmodyfikuj pliki konfiguracyjne tak, aby zmieniony znaczek zachęty **PS1** był zdefiniowany tylko w interpreterze logującym, alias obowiązywał we wszystkich interpreterach, zaś zmienne **IGNOREEOF** oraz **TMOUT** były zdefiniowane tylko w interpreterze interaktywnym.
  - Zmodyfikuj wartość zmiennej **PATH** tak, aby zawierała ona ścieżkę dostępu do katalogu bieżącego, a nowa wartość zmiennej była dostępna w interpreterze logującym i interaktywnym.
  - W trakcie odłączania się od systemu, ekran powinien być czyszczony (polecenie **clear**).

Użyj wpisu w pliku `~/.bash_logout`.

### Zadanie sprawdzające

Poniższe zadanie wykorzystuje kompleksowo wiedzę z tego laboratorium. Postaraj się wykonać je samodzielnie w domu. Jeżeli masz z nim problemy, przestuduj ponownie materiały źródłowe, rozwiąż wcześniejsze zadania i podejmij kolejną próbę.

Podaj pełną komendę zwracającą konkretną wartość (nie należy np. liczyć wierszy „ręcznie”):

1. Jak „doświadczalnie” sprawdzić, jaki jest znak kontynuacji (**PS2**)?
2. Ile jest w systemie zmiennych, które nie są eksportowane do środowiska potomnego?
3. Jak uruchomić nowy interpreter interaktywny?

### Podsumowanie komend

Na zajęciach przedstawione zostały następujące komendy:

Grupa	Komenda
Zmienne	set, unset, env, export, readonly
Alias	alias, unalias
Historia	history, !!, !n, !-n

### Przydatna składnia: brace expansion

W powłoce tekstowej możliwe jest wykorzystanie składni *brace expansion*, pozwalającej na tworzenie ciągów alfanumerycznych o podanych parametrach.

Do utworzenia ciągu z postępowaniem jeden możemy skorzystać z wyrażenia `{1..20}`, które zwróci ciąg liczb od 1 do 20 (włącznie). Jest to tak samo możliwe dla liter, zatem składnia `{a..k}` jest również poprawna.

Można także specyfikować *skok* dokonywany podczas tworzenia ciągu, zatem wyrażenie:

```
echo {1..20..3}
```

Spowoduje wypisanie ciągu znaków:

```
1 4 7 10 13 16 19
```

Ciągi mogą być również generowane malejąco, np. `{19..1..3}` spowoduje zwrócenie powyższej listy liczb w odwrotnej kolejności.

### Przydatna składnia: przekierowania

Typowym przekierowaniem, pozwalającym na przekazanie zawartości pliku tekstowego na standardowe wejście programu, jest:

```
program < plik.txt
```

Można jednak skorzystać z innych metod. Poniższa składnia pozwala na przesłanie wielowierszowego wejścia do programu (DELIM oznacza początek i koniec bloku tekstu i jest oznaczeniem deklarowanym w pierwszej linii po <<):

```
program << DELIM
Tutaj
jest
wiele
linii
DELIM
```

Użycie operatora <<- pozwala na usunięcie *białych znaków* z początku wejścia:

```
program <<- DELIM
Te linie
    zostaną przekazane
        bez tabulatorów
DELIM
```

Można także przekierowywać tekst na wejście programu jakby został on przekazany przez standardowe wejście. Służy do tego składnia:

```
program <<< "Tekst na wejście"
```

I jest ona analogiczna do:

```
echo "Tekst na wejście" | program
```

Nie jest to jednak forma często spotykana.