Powrót (Systemy operacyjne)

Skrypty w BASHu

ćwiczenia laboratoryjne

Ważne komendy (przypomnienie)

- `clear` - czyści ekran terminala.
- `read` - wczytuje dane ze standardowego wejścia oraz umieszcza je w zmiennej.
- `sleep` - pauzuje przez pewien czas.

Budowa skryptu BASH

```
#!/bin/bash

#instrukcje do wykonania

exit 0
```

Uruchomienie: `./nazwa`

- *nazwa* musi mieć rozszerzenie: **.sh*
- skrypt można uruchomić jeżeli ma uprawnienia do wykonania.

Uruchomienie w bieżącej powłoce: `.` *nazwa* lub `source` *nazwa*

Podanie argumentów podczas uruchomienia: `./nazwa argument1 argument2 ... argumentN`

Przypisanie wartości zmiennej

- *zmienna=wartość*
- `let` *zmienna=wartość* - typu całkowitego.
- *zmienna*=(*wartość1 wartość2 ... wartośćN*) - typu tablicowego.
- *zmienna*=([*index1*]=*wartość1* [*index2*]=*wartość2* ... [*indexN*]=*wartośćN*) - typu tablicowego (przypisanie wartości do konkretnych pozycji).

Odniesienie się do wartości zmiennej: `$`*zmienna*.

Odniesienie się do argumentów

- `$0` - polecenie uruchomienia skryptu.
- `$1` - pierwszy argument (*argument1*).
- `$2` - drugi argument (*argument2*).
- `$3` - trzeci argument (*argument3*).
- `$`*N* - *n*-ty argument.
- `$*` - wszystkie argumenty.
- `$#` - liczba argumentów.

Składnia operacji arytmetycznych: `$((`*wyrażenie*`))` lub `$[`*wyrażenie*`]`.
Można je wykonywać jedynie na liczbach całkowitych.

Operacje na zmiennych typu napisowego

- *napis1napis2* - konkatenacja.
- `${#`*zmienna*`}` - długość napisu, którego wartość znajduje się w zmiennej.
- `${`*zmienna*`:`*pozycja*`}` - wartość napisu od *pozycja* (indeksowany od 0).
- `${`*zmienna*`:`*pozycja1*`:`*pozycja2*`}` - wartość napisu od *pozycja1* do *pozycja2* (indeksowany od 0).

Operacje na zmiennych typu całkowitego

- *wartość1*+*wartość2* - dodawanie.
- *wartość1*-*wartość2* - odejmowanie.
- *wartość1***wartość2* - mnożenie.
- *wartość1*/*wartość2* - dzielenie.
- *wartość1*%*wartość2* - reszta z dzielenia.
- *wartość1****wartość2* - potęga

- `0`*wartość* - liczba w systemie ósemkowym.
- `0x`*wartość* - liczba w systemie szesnastkowym.
- *podstawa*`#`*wartość* - liczba w innym systemie liczbowym.

Operacje na zmiennych typu tablicowego

- `${`*zmienna*`[*]}` - wszystkie elementy tablicy.
- `${`*zmienna*`[`*indeks*`]}` - odniesienie się do wartości znajdującej się w tablicy na pozycji *indeks*.
- `${!`*zmienna*`[*]}` - wszystkie indeksy.
- `${#`*zmienna*`[*]}` - liczba elementów tablicy.

Wartość wyjścia standardowego polecenia: `$(`*polecenie*`)` lub `` `polecenie` ``.

Operatory logiczne

- *wyrażenie1* `&&` *wyrażenie2* - spełniony jeżeli wyrażenie po prawej oraz po lewej jest spełnione (dwa wyrażenia muszą być spełnione).
- *wyrażenie1* `||` *wyrażenie2* - spełniony jeżeli wyrażenie po prawej lub po lewej jest spełnione (przynajmniej jedno wyrażenie musi być spełnione).
- `!` *wyrażenie* - neguje wartość wyrażenia.

na liczbach całkowitych:

- *wartość1* `-eq` *wartość2* - spełniony, jeżeli oba wartości są takie same.
- *wartość1* `-ne` *wartość2* - spełniony, jeżeli oba wartości są różne.
- *wartość1* `-lt` *wartość2* - spełniony jeżeli wartość po lewej stronie jest mniejsza niż wartość po prawej stronie.
- *wartość1* `-le` *wartość2* - spełniony jeżeli wartość po lewej stronie jest mniejsza lub równa wartości po prawej stronie.
- *wartość1* `-gt` *wartość2* - spełniony jeżeli wartość po lewej stronie jest większa niż wartość po prawej stronie.
- *wartość1* `-ge` *wartość2* - spełniony jeżeli wartość po lewej stronie jest większa lub równa wartości po prawej stronie.

na napisach:

- *wartość1* `=` *wartość2* - spełniony, jeżeli oba wartości są takie same.
- *wartość1* `!=` *wartość2* - spełniony, jeżeli oba wartości są różne.
- *wartość1* `<` *wartość2* - spełniony, jeżeli pierwszy napis jest prędzej leksykograficznie.
- *wartość1* `>` *wartość2* - spełniony, jeżeli pierwszy napis jest później leksykograficznie.
- `-z` *wartość* - spełniony jeżeli wartość na długość równą 0.
- `-n` *wartość* - spełniony jeżeli wartość na długość różną od 0.

na plikach i katalogach:

- `-e` *wartość* - spełniony, jeżeli ścieżka istnieje.
- `-f` *wartość* - spełniony, jeżeli wartość jest plikiem i istnieje.
- `-d` *wartość* - spełniony, jeżeli wartość jest katalogiem i istnieje.
- `-c` *wartość* - spełniony, jeżeli wartość jest plikiem specjalnym znakowym i istnieje.
- `-b` *wartość* - spełniony, jeżeli wartość jest plikiem specjalnym blokowym i istnieje.
- `-h` *wartość* - spełniony, jeżeli wartość jest dowiązaniem symbolicznym i istnieje.
- `-S` *wartość* - spełniony, jeżeli wartość jest gniazdem i istnieje.
- `-p` *wartość* - spełniony, jeżeli wartość jest potokiem nazwanym i istnieje.
- `-r` *wartość* spełniony jeżeli plik/katalog można odczytać.
- `-w` *wartość* spełniony jeżeli plik/katalog można zapisać.
- `-x` *wartość* spełniony jeżeli plik/katalog można wykonynać.
- *wartość1* `-nt` *wartość2* - spełniony jeżeli wartość po lewej stronie jest nowsza od wartości po prawej stronie.
- *wartość1* `-ot` *wartość2* - spełniony jeżeli wartość po lewej stronie jest starsza od wartości po prawej stronie.

Instrukcja warunkowa *if*

```
if [[ warunek1 ]]
then
 # instrukcje wykonywane, jeżeli warunek1 jest spełniony
elif [[ warunek2 ]]
then
 # instrukcje wykonywane, jeżeli poprzedni warunek nie został spełniony natomiast warunek2 jest spełniony
# ...
else
 # instrukcje wykonywane, jeżeli żadne z wcześniejszych wyrażeń nie został spełniony
fi
```

Instrukcja warunkowa *case*

```
case wyrażenie in
wartość1)
 # instrukcje wykonywane, jeżeli wyrażenie -eq wartość1
 ;;
wartość2)
 # instrukcje wykonywane, jeżeli wyrażenie -eq wartość2
 ;;
# ...
*)
 # instrukcje wykonywane, jeżeli żaden z wcześniejszych warunków nie został spełniony
 ;;
esac
```

Pętla *while* - pętla w która jest wykonywana dopóki warunek jest spełniony.

```
while [[ warunek ]]
do
 # instrukcje do wykonania w pętli, jeżeli warunek jest spełniony
done
```

Pętla *until* - pętla w która jest wykonywana dopóki warunek nie jest spełniony.

```
until [[ warunek ]]
do
 # instrukcje do wykonania w pętli, jeżeli warunek jest nie spełniony
done
```

Pętla *for* (wersja 1) - pętla w której określamy liczbę przebiegów.

```
for (( wyrażenie1; wyrażenie2; wyrażenie3 ))
do
 # instrukcje do wykonania w pętli
done
```

- *wyrażenie1* - instrukcja wykonywana przed pierwszym przebiegiem pętli. Przeważnie jest to inicjalizacja zmiennej, która będzie służyła jako "licznik" przebiegów pętli.
- *wyrażenie2* - warunkiem zakończenia pętli.
- *wyrażenie3* - instrukcja wykonywana po każdym przejściu pętli (także ostatnim). Przeważnie jest to instrukcja, zwiększająca "licznik" o odpowiednią wartość.

Pętla *for* (wersja 2) - pętla wykonywana dla każdej wartości z listy.

```
for zmienna in lista
do
 # instrukcje do wykonania w pętli dla każdego argumentu
done
```

- *lista* - napis (słowa oddzielone spacjami) lub tablica

Pętla *select* pętla na podstawie, której tworzone jest proste menu. Z listy argumentów tworzone jest ponumerowane menu, każdej pozycji odpowiada kolejna liczba od zaczynając 1.

```
select zmienna in lista
do
 # instrukcje do wykonania w pętli dla każdego argumentu
done
```

- *lista* - napis (słowa oddzielone spacjami)

Definicja funkcji

```
function nazwa
{
 # instrukcje do wykonania we funkcji
}
```

- wynik jest zwracany przy pomocy instrukcji `return`. Możemy zwrócić jedynie liczbę całkowitą jednobajtową (tzn. z zakresu 0-255).

Wywołanie funkcji: *nazwa argument1 argument2 ... argumentN*

Deklaracja zmiennej lokalnej: `local` *nazwa*
Taka zmienna jest widoczna tylko w bloku w którym się znajduje (np. w bloku funkcji).

Status wyjścia ostatniego polecenia: `$?`
Może zostać użyta m.in. do pobrania wyniku, który zwraca funkcja. Musi zostać użyta w następnej linii, która wywołuje funkcje.

Zadania

Zadanie 1. Napisz skrypt, który wyświetli sumę 3 liczb całkowitych, które zostały pobrane jako argumenty.

Zadanie 2. Napisz skrypt, który wyświetli, pobierze dwa napisy jako argument oraz wyświetli informacje, który jest dłuższy i który jest wcześniej leksykograficznie.

Zadanie 3. Napisz skrypt, który wyświetli wyświetli informacje na temat tego, czy argument, będący ścieżką do pliku lub katalogu oraz wyświetli informację jakie prawa dostępu (odczyt, zapis, wykonanie) zawiera użytkownik.

Zadanie 4. Napisz skrypt, który utworzy katalog nazwany dzisiejszą datą, a następnie przekopiuje do niego wszystkie utworzone dziś pliki, znajdujące się w Twoim katalogu domowym.

Zadanie 5. Napisz skrypt, który wyświetli wszystkie pobrane argumenty w odwrotnej kolejności, np. argumenty `1 2 3`, powinny zostać wyświetlone jako `3 2 1`. Wykorzystaj tablicę.

Zadanie 6. Napisz skrypt, który będzie zawierał funkcję wyświetlającą odwrócony napis, który zostanie pobrany jako argument. Przykładowo napis *kot* powinien zostać wyświetlony jako *tok*.

Zadanie 7. Napisz skrypt, który jako argument pobierze ścieżkę do pliku. Następnie jeżeli plik ma rozszerzenie **.txt* otworzy go w dowolnym edytorze tekstu, jeżeli ma rozszerzenie **.sh* uruchomi skrypt w przeciwnym razie wyświetli odpowiedni komunikat informujący, że plik jest nieznany. Jeżeli żaden argument nie zostanie pobrany wyświetl komunikat informujący, że brakuje argumentu.

Zadanie 8. Napisz skrypt, który wyświetli informacje odnośnie tego jaka jest ścieżka do pliku terminala osoby, której login zostanie pobrany jako argument.

Zadanie 9. Napisz skrypt, który policzy ile plików i katalogów, które znajdują się w katalogu głównym oraz nazwa zawiera podsłowo, które zostało podane jako argument.

Zadanie 10. Napisz skrypt, który wyliczy średnią zaokrąglaną do części całkowitej z dowolnej ilości podanych argumentów. Rozwiąż to zadanie bez korzystania z instrukcji sterujących.

Zadanie 11. Napisz skrypt, który w sposób rekurencyjny wyliczy rozmiary wszystkich plików znajdujących się w folderze, którego ścieżka została podana jako argument.

Zadanie 12. Napisz skrypt, który będzie tworzył archiwum w następujący sposób:

- przeszuka folder o nazwie podanej jako argument i jego podkatalogi oraz znajdzie pliki **.txt*.
- umieści znalezione pliki **.txt* w archiwum (**.tar*, albo **.zip*) o nazwie *użytkownik*_backup_*data* (gdzie *użytkownik* to nazwa aktualnego użytkownika, natomiast *data* to aktualna data), dane na temat użytkownika i daty, powinny zostać pobrane odpowiednim poleceniem, natomiast archiwum powinno zostać umieszczone w folderze `~/backups`.

Zadanie 13. Utwórz dwa skrypty oraz potok nazwany (FIFO).
Pierwszy skrypt powinien odpowiadać za rolę serwera, który po odebraniu słowa wysyła do klienta informację ile samogłosek zawiera słowo (powinien działać w nieskończonej pętli).
Drugi skrypt powinien pełnić rolę klienta, który pobiera napis od użytkownika oraz wysyła go do serwera, następnie czeka aż serwer wyśle obliczenia na temat długości, wyświetli komunikat odnośnie jak długi jest napis i zakończy działanie.
Skrypty odpowiadające z klient i serwer powinni się komunikować poprzez potok nazwany (FIFO).

Zadanie 14. Napisz skrypt, który utworzy zmienną wyeksportowaną o nazwie *test* o wartości pobranej jako argument. Uruchom skrypt w podpowłoce (czyli tak jak zawsze) i bieżącej powłoce (przy drugim sposobie nie umieszczaj `exit 0` na końcu skryptu). Sprawdź czy zmienna została utworzona po pierwszym i drugim uruchomieniu.