Programowanie w BASH'u
Skrypt do ćwiczeń

1. Wstęp

Interpreter poleceń umożliwia wykonywanie poleceń pobieranych zarówno z klawiatury jak i z pliku. Jeżeli polecenia te wykonywane są z pliku to plik taki nazywa się skryptem.

Skrypty mogą zawierać dowolne polecenia systemu oraz dodatkowe elementy takie jak zmienne, instrukcje sterujące i programotwórcze. Interpreter poleceń umożliwia także przekazywanie do skryptów parametrów zewnętrznych nazywanych parametrami wywołania.

Skrypty edytuje się w dowolnym edytorze tekstowym (vi, mcedit, itp.). Aby skrypt mógł zostać wykonany należy mu nadać prawo wykonywalności (x). Uruchamia się go natomiast wywołując nazwę skryptu z ewentualnymi parametrami np.: `./skrypt p1 p2 p3 ...`

Dla każdego skryptu należy określić w jakiego rodzaju powłoce ma być wykonany, dlatego konieczne jest umieszczenie takiej informacji w pierwszej linii skryptu. Linia ta może mieć następującą składnię: `#!/bin/bash`, gdzie po znakach `#!` wpisuje się ścieżkę do interpretera poleceń, w którym skrypt został napisany.

Jeśli istnieje konieczność dopisania komentarzy to wpisuje się je po znaku `#`. Wszystko co zostanie wpisane po tym znaku w tej samej linii, jest pomijane przez interpreter.

Przykłady

```
#!/bin/bash
# to jest skrypt, który wypisze na ekranie: „Hello World !”
echo „Hello World !”
```

2. Pisanie na standardowe wyjście

Polecenie echo umożliwia wpisanie na standardowym wyjściu dowolnego tekstu. Napis ten może być zwykłym ciągiem tekstowym, może zawierać wartości parametrów lub zmiennych, może również zawierać wynik wykonania jakiegoś polecenia.

Parametry

`-n` nie wysyła znaku nowej linii
`-e` włącza interpretacje znaków specjalnych w tekście takich jak:

- `\a` czyli alert - bell
- `\b` backspace
- `\c` pomija znak nowej linii
- `\f` escape
- `\r` znak nowej linii

- `\t` tabulacja pozioma
- `\v` tabulacja pionowa
- `\\` backslash
- `\xnnn` znak o podanym szesnastkowo kodzie ASCII

Przykłady

```
#wyświetlenie napisu bez przejścia do nowej linii (dwa warianty)
echo –n „jakiś tekst bez znaku nowej linii”
echo –e „jakiś tekst bez znaku nowej linii\c”
#przejście do nowej linii
echo –e „\r”
#zapisane tekstu do pliku
echo „ten tekst trafi do pliku” > plik
#wyświetlenie tekstu oraz wartości zmiennej
echo „mój katalog domowy to: $HOME”
#wyświetlenie tekstu oraz wyniku polecenia
echo „aktualna data: `date`”
```

3. Zmienne

Zmienne umożliwiają przechowywanie i poźniejsze wykorzystanie wartości zdefiniowanych przez użytkownika lub będących wynikiem wykonania jakiegoś polecenia.

Zmienną definiujemy używając składni: `zmienna=wartość` (uwaga, nie może być spacji po obu stronach znaku =), gdzie element `wartość` może być określony jako:

- `'czysty tekst'` tekst zawarty pomiędzy znakami apostrofów nie jest interpretowany nawet jeśli zawiera jakieś znaki sterujące, nazwy zmiennych lub polecenia,
- `` `polecenie` `` lub `$(polecenie)` poprawne składniowo polecenie powłoki,
- `"tekst, zmienne, polecenia"` może zawierać zwykły tekst, zmienne lub poprawne składniowo polecenie powłoki

Do zmiennych odwołać się można podając jej nazwę poprzedzoną znakiem `$`.

Przykłady

```
x='jakiś napis'
echo „zmienna x zawiera tekst: $x”
data=`date`
echo „aktualna data to: $data”
katalog=$(pwd)
echo „jestem w katalogu: $katalog”
y=$(ls –al $katalog)
echo $y #wyświetlenie zawartości katalogu bieżącego
echo „mój katalog domowy to: $HOME”
echo „napis \$HOME”
```

```
z=abc
echo $z    #abc
echo $zd   #
echo ${z}d #abcd

a=2
b=3
c=$a + $b
echo $c    #2 + 3
```

Tak zdefiniowane zamienne są zmiennymi lokalnymi, co oznacza, ze mają zasięg tylko w ramach bieżącego skryptu. Oprócz zmiennych lokalnych można zdefiniować także zmienne globalne widoczne w innych skryptach i podshellach. Zmienne globalne definiuje się tak samo jak zmienne lokalne należy je tylko poprzedzić słowem `export`.

Przykłady

```
export x='jakiś napis'
export data=`date`
```

W systemie istnieje już kilka predefiniowanych zmiennych systemowych takich jak np:

- `$HOME` ścieżka katalogu domowego,
- `$PWD` katalog bieżący,
- `$HOSTNAME` nazwa hosta,
- `$OSTYPE` rodzaj systemu operacyjnego

Szczegółową listę zmiennych globalnych można znaleźć wydając polecenie `export` lub `printenv`.

Zmienne lokalne są usuwane automatycznie po zakończeniu skryptu zmienne globalne pozostają w systemie. Aby usunąć zmienną globalną lub jawnie usunąć zmienną lokalną należy użyć polecenia `unset zmienna`. Można także spowodować aby zmienna globalna stała się zmienna lokalną (i została usunięta automatycznie) w tym celu należy wydać polecenie `export –n zmienna`.

4. Parametry

Do skryptów można przekazywać parametry zewnętrzne (parametry wywołania), które mogą być wykorzystane wewnątrz skryptu do sterowania przetwarzaniem. Wewnątrz skryptu istnieją następujące, predefiniowane, zmienne:

- `$0` nazwa bieżącego skryptu,
- `$1, $2 ... $9` kolejne parametry przekazane do skryptu,
- `$@` wszystkie parametry przekazane do skryptu
- `$#` liczba parametrów przekazanych do skryptu
- `$?` kod powrotu ostatnio zakończonego polecenia

- `$$` PID bieżącej powłoki

Parametry `$1 ... $9` można przesuwać w lewo za pomocą polecenia `shift`. Po wydaniu tego polecenia parametr `$1` przyjmie wartość parametru `$2`, `$2` będzie miał wartość parametru `$3`, ..., `$8` będzie maił wartość parametru `$9`, a `$9` będzie miał wartość pustą.

Skrypt o nazwie „ nasz_skrypt"

```
#!/bin/bash
echo „nazwa skryptu: $0"
echo „liczba argumentów wywołania: $#"
echo „wszystkie argumenty wywołania: $@"
echo „argument1: $1, argument2: $2, argument3: $3"
```

Przykładowe wywołanie:

```
nasz_skrypt a b –xyz
```

Wynik działania:

```
nazwa skryptu: nasz_skrypt
liczba argumentów wywołania: 3
wszystkie argumenty wywołania: a b –xyz
argument1: a, argument2: b, argument3: -xyz
```

5. Tablice

Interpreter poleceń BASH umożliwia także na stosowanie zmiennych w postaci jednowymiarowych tablic. Deklarowanie i odwołanie do elementów tablicy jest podobne do tworzenia zwykłych zmiennych, dodatkowym elementem jest tylko indeks elementu w tablicy. Tablice indeksowane są od 0.

Przykłady

```
#utworzenie trzyelementowej tablicy i nadanie jej elementom wartości
#element1, element2, element3
tablica1=(element1 element2 element3)
#utworzenie tablicy dwuelementowej i nadanie jej elementom wartości
tablica2[0]="abc"
tablica2[1]="xyz"
#wypisanie drugiej komórki z tablica1 i tablica2
echo ${tablica1[1]}
echo ${tablica2[1]}
#wypisanie wszystkich elementów tablica1
echo ${tablica1[*]}
#wyświetlenie ilości elementów w tablica1
echo ${#tablica1[*]}
#wyświetlenie ilości znaków w drugiej komórce tablica1
echo ${#tablica1[1]}
#usuniecie drugiej komórki z tablica1
unset tablica1[1]
```

```
#usuniecie całej tablica2
unset tablica2[*]
```

6. Pobieranie danych

Do pobierania danych ze standardowego wejścia służy polecenie `read`. Polecenie to umożliwia pobranie danych i zapisanie ich w odpowiednich zmiennych.

Przykłady

```
#pobranie danych do zmiennej wpis
echo -ne "wpisz coś:\a"
read wpis
echo "$wpis"

#pobranie danych do kilku zmiennych. Jeśli danych będzie więcej niż
#zmiennych ostatnia zmienna będzie zawierała resztę danych
echo "wpisz trzy wartości:"
read a b c
echo "wartość zmiennej a to: $a"
echo "wartość zmiennej b to: $b"
echo "wartość zmiennej c to: $c"

#pokaże znak zachęty i nie przejdzie do nowej linii
read -p "wisz:" odp
echo "$odp"

#pobranie danych do tablicy
echo "podaj elementy zmiennej tablicowej:"
read -a tablica
echo "${tablica[*]}"
```

7. Obliczanie wartości wyrażeń

Do obliczania wartości wyrażeń służy polecenie `expr`. Polecenie to ma następującą składnię: `expr wartość1 operator wartość2`. W poleceniu można stosować następujące operatory: `+,-,*,/,=,%,^,<,>,&,|`, itd. W BASH'u wyrażenia możemy obliczać stosując także notację `$[wyrażenie]` lub `$((wyrażenie))`.

Przykłady

```
max=0
max=expr $max + 1
echo $max #1

a=2
b=3
c=$[$a^$b+1]
d=$(($a^$b+1))
echo $c, $d #7, 7
```

8. Instrukcje sterujące

8.1. Instrukcja warunkowa if

Instrukcja ta sprawdza czy warunek jest prawdziwy, jeśli tak to wykonywane jest polecenie lub polecenia znajdujące się po słowie kluczowym `then`. Instrukcję należy zakończyć słowem kluczowym `fi`. W sytuacji gdy test warunku zakończy się wynikiem negatywnym można wykonać alternatywny zestaw poleceń, które umieszczamy po słowie kluczowym `else`. Za pomocą tej instrukcji można także testować większą ilość warunków, jeśli pierwszy warunek nie będzie prawdziwy, sprawdzony zostanie następny. W takim przypadku następne warunki należy umieścić po słowie kluczowym `elif`.

Składnia

```
if warunek
then
polecenie(a)
fi
```

```
if warunek
then
polecenie(a)
else
polecenie(a)
fi
```

```
if warunek
then
polecenie(a)
elif warunek
polecenie(a)
elif warunek
polecenie(a)
fi
```

Do konstruowania warunków służy polecenie `test wyrażenie1 operator wyrazenie2` lub notacja oparta o zapis symboliczny w nawiasie kwadratowym: `[ wyrażenie1 operator wyrazenie2 ]`. Uwaga pomiędzy nawiasami a treścią warunków muszą być spacje. Konstrukcja ta zwraca wartość 0 (true) jeśli warunek jest spełniony oraz 1 (false) jeśli warunek nie jest spełniony. Wartość testu warunku przechowywana jest także w zmiennej `$?`. Jeśli operatorem nie jest operator porównania `wyrażenie1` nie występuje.

Przykłady operatorów:

- `-a` plik istnieje
- `-b` plik istnieje i jest blokowym plikiem specjalnym
- `-` plik istnieje i jest plikiem znakowym
- `-e` plik istnieje
- `-h` plik istnieje i jest linkiem symbolicznym
- `=` sprawdza czy wyrażenia są równe
- `!=` sprawdza czy wyrażenia są różne
- `-n` wyrażenie ma długość większą niż 0
- `-d` wyrażenie istnieje i jest katalogiem
- `-z` wyrażenie ma zerową długość
- `-r` można czytać plik

- `-w` można zapisywać do pliku
- `-x` można plik wykonać
- `-f` plik istnieje i jest plikiem zwykłym
- `-p` plik jest łączem nazwanym
- `-N` plik istnieje i był zmieniany od czasu jego ostatniego odczytu
- `plik1 -nt plik2` plik1 jest nowszy od pliku2
- `plik1 -ot plik2` plik1 jest starszy od pliku2
- `-lt` mniejsze niż
- `-gt` większe niż
- `-ge` większe lub równe
- `-le` mniejsze lub równe
- więcej przykładów operatorów można znaleźć w pomocy polecenia bash

W przypadku wyrażeń logicznych można używać spójników logicznych takich jak: `-a` (and), `-o` (or) oraz `!` (not).

Przykłady

```
#sprawdzenie czy w katalogu domowym znajduje sie plik .bashrc i
wypisanie odpowiedniego #komunikatu
if [ -e ~/.bashrc ]
then
echo "w twoim katalogu domowym jest plik.bashrc"
else
echo "w twoim katalogu domowym nie ma pliku .bashrc"
fi

#sprawdzenie czy w katalogu bieżącym jest plik plika oraz czy
#użytkownik ma prawo do jego odczytu a następnie utworzenie kopii
#tego pliku
if [ -r plika –a –e plika ]
then
cp plika plika.bak
fi

#sprawdzenie czy parametr $1 jest większy lub równy od $2
if [ $1 –gt $2 ]
then
echo "\$1 jest większy od \$2"
fi
```

8.2. Warunkowe wykonanie polecenia

Polecenia mogą być wykonywane pod warunkiem sukcesu lub błędu wykonania poprzedniego polecenia. Istnieją dwie możliwości warunkowego wykonania poleceń:

- `polecenie1 && polecenie2` polecenie2 zostanie wykonane pod warunkiem poprawnego wykonania polecenia1 – kod powrotu 0
- `polecenie1 || polecenie2` polecenie2 zostanie wykonane jeśli polecenie1 zakończy się błędem – kod powrotu <> 0

8.3. Instrukcja case

Pozwala na dokonanie wyboru spośród kilku wzorców. Najpierw sprawdzana jest wartość zmiennej po słowie kluczowym case i porównywana ze wszystkimi wariantami po kolei. Jeśli dopasowanie zakończy się sukcesem wykonane zostanie polecenie lub polecenia przypisane do danego wzorca. W przeciwnym wypadku użyte zostanie polecenie domyślne oznaczone symbolem gwiazdki.

Składnia

```
case zmienna in
"wzorzec1") polecenie(a)1 ;;
"wzorzec2") polecenie(a)2 ;;
"wzorzec3") polecenie(a)3 ;;
*) polecenie(a)_domyślne
esac
```

Przykłady

```
echo "podaj cyfrę dnia tygodnia"
read d
case "$d" in
"1") echo "poniedziałek" ;;
"2") echo "wtorek" ;;
"3") echo "środa" ;;
"4") echo "czwartek" ;;
"5") echo "piątek" ;;
"6") echo "sobota" ;;
"7") echo "niedziela" ;;
*) echo "nic nie wybrałeś"
esac
```

8.4. Pętla for

Wykonuje polecenia zawarte wewnątrz pętli, na każdym składniku listy.

Składnia

```
for zmienna in lista
do
polecenie(a)
done
```

Przykłady

```
#wypisanie na ekranie kolejnych elementów listy jeden, dwa, trzy
for x in jeden dwa trzy
do
echo "to jest $x"
done

#wypisanie na ekranie kolejnych elementów listy – plików z
#rozszerzeniem html
for x in *html
do
echo "to jest plik $x"
done

#wypisanie na ekranie kolejnych wierszy będących wynikiem wykonania
polecenia
for x in `cat /etc/passwd`
do
echo "to jest wiersz $x"
done
```

8.5. Pętla while

Pętla jest wykonywana tak długo dopóki spełniony jest warunek. Jeśli warunek jest spełniony to wykonane zostanie polecenie lub lista poleceń zawartych wewnątrz pętli, gdy warunek stanie się fałszywy pętla zostanie zakończona. Wykonywanie pętli można zakończyć wcześniej umieszczając w niej polecenie `break`. Polecenie `continue` wznawia wykonywanie przerwanej petli.

Składnia

```
while warunek
do
polecenie(a)
done
```

Przykłady

```
x=1;
while [ $x -le 10 ]
do
echo "napis pojawił się po raz: $x"
x=$[x + 1]
done
```

8.6. Pętla until

Pętla jest wykonywana tak długo dopóki nie jest spełniony warunek Jeśli warunek jest fałszywy wykonywane jest polecenie lub lista poleceń zawartych wewnątrz pętli, miedzy słowami kluczowymi `do` a `done`. Pętla `until` kończy swoje działanie w momencie gdy warunek stanie się prawdziwy. Wykonywanie pętli można zakończyć wcześniej umieszczając w niej polecenie `break`. Polecenie `continue` wznawia wykonywanie przerwanej petli.

Składnia

```
until warunek
do
polecenie(a)
done
```

Przykłady

```
x=1;
until [ $x -ge 10 ]
do
echo "napis pojawił się po raz: $x"
x=$[x + 1]
done
```

9. Funkcje

Jeżeli w skrypcie często pojawia się grupa powtarzających się poleceń wtedy można zastosować funkcje. Stosowanie funkcji poprawia czytelność kodu oraz pozwala na łatwiejsze eliminowanie błędów. Do utworzonej funkcji odwołać się można za pomocą jej nazwy, wykonane zostaną wtedy wszystkie polecenia zdefiniowane wewnątrz funkcji.

Składnia

```
function nazwa_funkcji
{
polecenie(a)
}
```

Przykłady

```
function napis
{
echo „to jest funkcja wyświetlająca ten napis”
}
#jakieś instrukcje ...
napis
```

Funkcje mogą się znajdować w innym pliku, co uczyni skrypt bardziej przejrzystym i wygodnym. W takim przypadku, aby wykonać funkcję znajdującą się w innym pliku należy dodać ten plik jako nagłówkowy. Plik nagłówkowy dołącza się podając jego nazwę (ścieżkę) poprzedzona kropką i spacją. Pliki nagłówkowe najlepiej dołączać na samym początku skryptu.

Przykłady

```
#plik nagłówkowy o nazwie moje_funkcje
function napis
{
echo „to jest funkcja wyświetlająca ten napis”
}

#plik skryptu
. moje_funkcje
#jakieś instrukcje ...
napis
```

10. Zadania do samodzielnego wykonania

- Napisać skrypt wyświetlający zawartość katalogu podanego jako pierwszy parametr wywołania. Jeśli parametr nie zostanie podany należy wyświetlić informacje o poprawnej składni polecenia.
- Napisz skrypt wyświetlający w kolejnych liniach argumenty przekazane do skryptu. Skrypt napisać w trzech wariantach: pierwszy z wykorzystaniem pętli `for`, drugi – pętli `while` i trzeci – petli `until`.
- Wyświetl zawartość plików podanych jako kolejne parametry wywołania pod warunkiem że pliki te istnieją. Jeśli nie został podany żaden plik należy wyświetlić komunikat: „Nie ma czego wyświetlać”. Treści plików należy rozdzielić linią składającą się z samych gwiazdek.
- Napisać skrypt który sprawdzi czy wszystkie pliki podane jako kolejne parametry wywołania istnieją. Jeśli wszystkie pliki istnieją należy wyświetlić komunikat „Wszystkie podane przez Ciebie pliki istnieją” a jeśli choćby jednego pliku brakuje to należy wyświetlić komunikat „Lista podanych plików jest niekompletna”.
- Napisz skrypt kopiujący plik wskazany pierwszym argumentem wywołania do wszystkich katalogów wskazanych kolejnymi argumentami.
- Napisz skrypt umożliwiający wykonanie trzech poleceń: `who, ps , ls`. Wybór polecenia powinien odbywać się za pomocą menu.

- Wyświetl informacje o procesach użytkowników podanych jako kolejne wiersze w pliku będącym pierwszym parametrem wywołania. Informacje należy wyświetlić w dwóch kolumnach: nazwa użytkownika i PID procesu.
- Napisać skrypt zmieniający nazwy plików na duże litery w katalogu podanym jako pierwszy parametr wywołania.
- Napisz skrypt wyświetlający informacje o plikach zwykłych w następującym formacie: `właściciel atrybuty rozmiar nazwa` przy czym pole właściciel może przyjmować trzy wartości: mój – jeśli jesteś właścicielem pliku, admin jeśli właścicielem jest administrator oraz inni jeśli właścicielem są pozostali użytkownicy.
- Napisz skrypt który odwróci kolejność wierszy z pliku podanego jako pierwszy parametr wywołania i wynik tej operacji zapisze do pliku o tej samej nazwie z rozszerzeniem `.rev`.
- Napisać program wsadowy, który poinformuje użytkownika ile z plików podanych jako kolejne parametry wywołania istnieje a ile nie istnieje a następnie wyświetli wszystkie pliki istniejące i nieistniejące w 2 kolumnach

```
Znalezione pliki: 5    Brakujące pliki: 3
-------------------    ------------------
plik1                  plik3
plik4                  plik7
plik9                  plik2
plik5
plik6
```

- Rozszerzyć powyższy skrypt tak aby lista plików pobierana była z kolejnych wierszy pliku podanego jako pierwszy parametr wywołania.
- Napisać program wsadowy do wykonywania podstawowych operacji na plikach (tworzenie, edycja, wyświetlanie, kopiowanie, usuwanie, zmiana nazwy, porównywanie, przeszukiwanie). Program uruchamiany jest z dwoma parametrami: nazwą pliku i operacją do wykonania. Reszta potrzebnych informacji powinna być pobierana od użytkownika w trakcie wykonania określonych operacji (np. przy zmianie nazwy pliku program powinien poprosić o podanie nowej nazwy). Program powinien zawierać obsługę błędów.
- Napisać program baza.bat obsługujący prostą bazę danych. Baza danych powinna zawierać następujące informacje: imię (nie może być puste), nazwisko (nie może być puste), płeć (k/m) i wiek w latach (1-100) . Dane powinny być przechowywane w kolejnych wierszach pliku „dane.txt" w formacie:

```
nazwisko1 imie1 płeć1 wiek1
nazwisko2 imie2 płeć2 wiek2
nazwisko3 imie3 płeć3 wiek3
```

  Zakładamy, że jeśli przy pierwszym uruchomieniu programu plik dane.txt nie istnieje musi zostać automatycznie utworzony. Program powinien być obsługiwany za pomocą menu i powinien umożliwiać: dodanie osoby (ze sprawdzeniem poprawności wpisanych danych), usunięcie osoby (z potwierdzeniem), wyświetlenie posortowanych danych, wyszukiwanie danych według wpisanego ciągu. Program nie powinien wyświetlać żadnych komunikatów systemowych tylko te zdefiniowane w programie. Pary nazwisko - imię w bazie nie mogą się powtarzać - program powinien to sprawdzić przy dodawaniu nowego wpisu.
- Przetestuj działanie i możliwości zastosowania polecenia `dialog`. Zastosuj program w trzech dowolnych skryptach.