

Python w Elektronicznej Sieci #6: Programowanie w bash'u

Projekt „Matematyka dla Ciekawych Świata”,
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2020-03-31

1 Bash jako narzędzie programowania

1.1 Zmienne

Określanie typów zmiennych w bashu odbywa się na podstawie wartości znajdującej się w zmiennej. Zasadniczo wszystkie zmienne są napisami, a interpretacja typu ma miejsce przy ich użyciu (a nie przy tworzeniu). Obsługiwane są liczby całkowite oraz napisy, bash nie posiada wbudowanej obsługi liczb zmiennoprzecinkowych.

```
zmiennaA=-91
zmiennaB="qa z"
zmiennaC=98.6 # to będzie traktowane jako napis a nie liczba
```

Zwróć uwagę na brak spacji pomiędzy nazwą zmiennej a znakiem równości w operacji przypisania - jest to wymóg składniowy.

Odwołanie do zmiennej odbywa się z użyciem znaku dolara, po którym występuje nazwa zmiennej. Nazwa może być ujęta w klamry, ale nie musi (jest to przydatne gdy nie chcemy dawać spacji pomiędzy nazwą zmiennej a np. fragmentem napisu). Rozwijaniu ulegają nazwy zmiennych znajdujące się w napisach umieszczonych w podwójnych cudzysłowach.

```
echo $zmiennaA ${zmiennaA}AA
echo "$zmiennaA ${zmiennaA}AA"
echo '$zmiennaA ${zmiennaA}AA'
```

1.2 Podstawowe operacje

Aby wykonać działania arytmetyczne należy umieścić je wewnątrz $(())$

Dodawanie, mnożenie, odejmowanie zapisuje się i działają one tak jak w normalnej matematyce, dzielenie zapisuje się przy pomocy ukośnika i jest ono zawsze dzieleniem całkowitym:

```
a=12; b=3; x=5; y=6

e=$(( ($a + $b) * 4 - $y ))
c=$(( $x / $y ))

echo $e $c $z
```

Zauważ zachowanie przy odwołaniu do niezainicjalizowanej zmiennej z.

Operacje logiczne obsługiwane są komendą test lub operatorem $[]$ wynik zwracany jest jako kod powrotu. Należy zwrócić uwagę na escapowanie odwrotnym ukośnikiem nawiasów i na to że spacje mają znaczenie. Negację realizuje $!$, należy pamiętać jednak że wynikiem negacji dowolnej liczby jest FALSE.

```
[ \ ( $a -ge 0 -a $b -lt 2 \) -o $z -eq 5 ]; z=$?
```

```
echo $z
```

Wartość zmiennej `z` jest wynikiem warunku: ((`a` większe równe od zera) AND (`b` mniejsze od dwóch)) OR (`z` równe 5). Bash stosuje logikę odwróconą 0 oznacza prawdę, coś nie zerowego to fałsz.

Jako operacje podstawowe powinniśmy patrzeć także na wykonanie innych programów i pobieranie ich standardowego wyjścia i/lub kodu powrotu. Pobieranie standardowego wyjścia możemy realizować za pomocą ujęcia polecenia w *backquotes* (```) lub operatora `$()` (pozwala on na zagnieżdżanie takich operacji). Natomiast kod powrotu ostatniej komendy znajduje się w zmiennej `?` (używaliśmy tego już przy obliczaniu wyrażeń logicznych).

```
a=`cat /etc/issue`  
b=$(cat /etc/issue; cat /etc/resolv.conf)
```

```
echo $a  
echo $b  
echo "$b"
```

Zwróć uwagę na różnicę w wypisaniu zmiennej zawierającej znaki nowej linii objętej cudzysłowami i nie objętej nimi.

Bash nie obsługuje liczb zmiennoprzecinkowych ani operacji bitowych, nieobsługiwane operacje można wykonać za pomocą innego programu np:

```
a=`echo 'print(3/2)' | python3`  
b=$(echo '3/2' | bc -l)  
echo $a $b
```

1.3 Uruchamianie kodu z pliku

Dłuższe fragmenty kodu bashowego często wygodniej jest pisać w pliku tekstowym niż bezpośrednio w linii poleceń. Plik taki może zostać wykonany przy pomocy polecenia: `./nazwa_pliku` pod warunkiem że ma prawo wykonalności (powinien także zawierać w pierwszej linii komentarz określający program używany do interpretacji tekstowego pliku wykonywalnego, w postaci: `#!/bin/bash`). Może też być wykonany za pomocą wywołania: `bash nazwa_pliku`.

Przydatną alternatywą dla powyższych metod wykonania kodu zawartego w pliku jest włączenie go do aktualnej sesji basha przy pomocy `./nazwa_pliku`. W odróżnieniu od poprzednich metod pozwala to na korzystanie z funkcji i zmiennych zdefiniowanych w tym pliku w kolejnych poleceniach.

1.4 Pętle i warunki

1.4.1 Pętla for

W bashu możemy korzystać z kilku wariantów pętli `for`. Jednym z najczęściej używanych jest przypadek iterowania po plikach¹:

```
for nazwa in /tmp/* ; do  
    echo $nazwa;  
done
```

Możliwe jest też iterowanie po wartościach całkowitych zarówno w stylu „shellowym” jak i w stylu C

1. Dokładniej: iteracja odbywa się po liście napisów rozdzielanej spacjami - zobacz rezultat `echo /tmp/*`

```
for i in `seq 0 20`; do
    echo $i;
done

for (( i=0 ; $i<=20 ; i++ )) ; do
    echo $i;
done
```

1.4.2 Pętla while

Często używana jest pętla while w połączeniu z instrukcją read co umożliwia przetwarzanie jakiegoś wejścia (wyniku komendy lub pliku) linia po linii (także z podziałem linii na słowa):

```
cat /etc/fstab | while read slowo reszta; do
    echo $reszta;
done
```

Powyższa pętla wypisze po kolei wszystkie wiersze pliku /etc/fstab przekazanego przez stdin (przy pomocy komendy cat)² z pominięciem pierwszego słowa (które wczytywane było do zmiennej slowo).

Słowa domyślnie rozdzielane są przy pomocy dowolnego ciągu spacji lub tabulatorów, separator można zmienić za pomocą zmiennej IFS, np:

```
IFS=":"
while read a b; do echo $a; done < /etc/group
unset IFS # przywracamy domyślne zachowanie read poprzez usunięcie zmiennej IFS
```

1.4.3 Instrukcja if

Poznane wcześniej obliczanie wartości wyrażeń logicznych najczęściej stosowane jest w instrukcji warunkowej if³.

```
# instrukcja if - else
if [ "$xx" = "kot" -o "$xx" = "pies" ]; then
    echo "kot lub pies";
elif [ "$xx" = "ryba" ]; then
    echo "ryba"
else
    echo "coś innego"
fi
```

Zauważ że spacje wokół i wewnątrz nawiasów kwadratowych przy warunku są istotne składniowo, zawartość nawiasów kwadratowych to tak naprawdę argumenty dla komendy test. Oprócz typowych warunków logicznych możemy sprawdzać np. istnienie plików, czy też ich typ (link, katalog, etc). Szczegółowy opis dostępnych warunków które mogą być użyte w tej konstrukcji znajduje się w man test.

Jako warunek może wystąpić dowolne polecenie wtedy sprawdzany jest jego kod powrotu 0 oznacza prawdę / zakończenie sukcesem, a wartość nie zerowa fałsz / błąd

```
if grep '^root:' /etc/passwd > /dev/null; then
    echo /etc/passwd zawiera root-a;
fi
```

2. Takie rozwiązanie nazywane jest *martwym kotem* i powinno go się unikać. Lepszym rozwiązaniem jest przekazywanie pliku przez przekierowanie strumienia wejściowego przy pomocy < plik, który w tym przypadku powinien znaleźć się za kończącym pętlę słowem kluczowym done.
3. Może być też stosowane np. w pokazanej wcześniej pętli while

Istnieje możliwość skróconego zapisu warunków z użyciem łączenia instrukcji przy pomocy `&&` (wykonaj gdy poprzednia zwróciła zero – true) lub `||` (wykonaj gdy poprzednia zwróciła nie zero – false):

```
[ -f /etc/issue ] && echo "jest plik /etc/issue"

grep '^root:' /etc/passwd > /dev/null && echo /etc/passwd zawiera root-a;
```

1.4.4 Instrukcja case

Instrukcja `case` służy do rozważania wielu przypadków opartych na równości zmiennej z podanymi napisami.

```
case $xx in
    kot | pies)
        echo "kot lub pies"
        ;;
    ryba)
        echo "ryba"
        ;;
    *)
        echo "cos innego"
        ;;
esac
```

1.5 Definiowanie funkcji

W bashu każda funkcja może przyjmować dowolną ilość parametrów pozycyjnych (w identyczny sposób obsługiwane są argumenty linii poleceń dla całego skryptu). Ilość parametrów znajduje się w zmiennej `$#`, lista wszystkich parametrów w `$@`, a do kolejnych parametrów możemy odwoływać się z użyciem `$1`, `$2`, itd.

```
f1() {
    echo "wywołano z $# parametrami, parametry to: $@"

    [ $# -lt 2 ] && return;

    echo -e "drugi: $2\npierwszy: $1"

    # albo kolejnych w pętli
    for a in "$@"; do echo $a; done

    # lub z użyciem polecenia shift
    for i in `seq 1 $#`; do
        echo $1
        shift # powoduje zapomnienie $1
              # i przenumerowanie argumentów pozycyjnych o 1
              # wpływa na wartości $@ $# itp
    done

    # funkcja może zwracać tylko wartość numeryczną -- tzw kod powrotu
    return 83
}
```

Zwróć uwagę że w nawiasach po nazwie funkcji nie podajemy przyjmowanych argumentów, natomiast puste nawiasy te są elementem składniowym i muszą wystąpić. Jeżeli zapisujesz definicję funkcji w jednej linii, np. `abc() { echo "abc"; }` to pamiętaj, że spacje po otwierającym i przed kończącym nawiasem klamrowym są obowiązkowe, podobnie jak średniki występujące po każdej instrukcji w ciele funkcji.

Wywołanie funkcji nie różni się niczym od wywołania programów czy instrukcji wbudowanych (możemy używać przekierowań strumieni wejścia, wyjścia, czy też przechwycić wyjście do zmiennej). Powyższą funkcję możemy wywołać np. w następujący sposób: `f1 a "b c" d`

1.6 Wbudowane przetwarzanie napisów w bash'u

Wbudowane przetwarzanie napisów w bashu opiera się na odwołaniach do zmiennych w postaci `${}`:

- `${zmienna:-"napis"}` zwróci napis gdy zmienna nie jest zdefiniowana lub jest pusta
- `${zmienna:="napis"}` zwróci napis oraz wykona podstawienie `zmienna="napis"` gdy zmienna nie jest zdefiniowana lub jest pusta
- `${zmienna:+"napis"}` zwróci napis gdy zmienna jest zdefiniowana i nie pusta
- `${#str}` zwróci długość napisu w zmiennej `str`
- `${str:n}` zwróci pod-napis zmiennej `str` od `n` do końca
- `${str:n:m}` zwróci pod-napis zmiennej `str` od `n` o długości `m`
- `${str/"n1"/"n2"}` zwróci wartość `str` z zastąpionym pierwszym wystąpieniem `n1` przez `n2`
- `${str//"n1"/"n2"}` zwróci wartość `str` z zastąpionymi wszystkimi wystąpieniami `n1` przez `n2`
- `${str#"ab"}` zwróci wartość `str` z obcięciem "ab" z początku
- `${str%"fg"}` zwróci wartość `str` z obcięciem "fg" z końca

W napisach do obcięcia możliwe jest stosowanie shellowych znaków uogólniających, czyli `*`, `?`, `[abc]`, itd operator `#` i `%` dopasowują minimalny napis do usunięcia, natomiast operatory `##` i `%%` dopasowują maksymalny napis do usunięcia.

Możliwe jest także korzystanie z wyrażeń regularnych. Polecenie `expr match $x 'wr1\(wr2\)wr3'` zwróci na `stdout` (wypisze) część `$x` pasującą do wyrażenia regularnego `wr2`, wyrażenia regularne `wr1` i `wr2` pozwalają na określanie części napisu do odrzucenia. Alternatywną składnią jest `expr $x : 'wr1\(wr2\)wr3'`

Możliwe jest też sprawdzanie dopasowań wyrażeń regularnych poprzez (zwróć uwagę na brak cytowania wyrażenia regularnego):

```
[[ "$z" =~ ^([^\=]*)= ]] && echo "OK"
```

Możliwe jest także zaawansowane formatowanie napisów, konwertowanie liczb na napisy, w tym wypisywanie w różnych systemach liczbowych przy pomocy `printf`⁴:

```
printf "0o%o %d 0x%x\n" 0xf 010 3
```

2 Literatura dodatkowa

Więcej informacji o podstawach działania w systematach „unixowych” oraz pełniejsza lista poleceń zamieszczona została pod adresem http://vip.opcode.eu.org/#Systemy_unix-owate.

4. Instrukcja `printf` ma składnię opartą na tej funkcji z C, interpretuje ona także liczby zmiennoprzecinkowe.

3 Zadania

Zadanie 3.0.1

Napisz pętlę, która wypisze wszystkie pliki nieukryte z bieżącego katalogu w postaci linków HTML, czyli: dla pliku o nazwie ABC powinna wypisać `ABC`. Przedstaw zarówno rozwiązanie z użyciem pętli `for`, jak i pętli `while`.

Zadanie 3.0.2

Napisz warunek, który sprawdzi czy `/tmp/abc` istnieje i jest katalogiem.

Zadanie 3.0.3

Napisać funkcję przyjmującą dwa argumenty - liczbę i napis; funkcja ma wypisać napis tyle razy ile wynosi podana liczba.

Zadanie 3.0.4

Napisać funkcję przyjmującą jeden argument - liczbę kotów i wypisującą:

- "Ala ma kota" dla ilości kotów równej 1
- "Ala ma x koty" lub "Ala ma x kotów" gdzie dobrana jest poprawna forma, a pod x podstawiona podana w argumencie ilość kotów.

Dla uproszczenia należy założyć że podana ilość kotów jest w zakresie od 1 do 9.

4 Zadania dodatkowe

Zadanie 4.0.1

Wyświetl z `/etc/passwd` linie w których UID (3 pole) ma wartość ≥ 1000 nie korzystając z AWK.

Zadanie 4.0.2

Napisz polecenie które skopiuje wszystkie pliki (nie katalogi ani linki symboliczne) z katalogu `/etc` do `/tmp`

Zadanie 4.0.3

Napisz funkcję która przyjmuje dwa argumenty - napis wyszukiwany i napis go zastępujący oraz dokonuje rekurencyjnego wyszukania i zamiany tych napisów w wszystkich plikach w bieżącym katalogu.

Wskazówka: dla uproszczenia możesz przyjąć że napisy te składają się jedynie z liter i cyfr.

Zadanie 4.0.4

Napisz polecenie które przekopiuje wszystkie pliki zawierające słowo `hostname` z katalogu `/etc` (wraz z jego podkatalogami) do katalogu `/tmp/etc` zachowując strukturę katalogów (czyli plik `/etc/a/b` kopiowany jest do katalogu `/tmp/etc/b`). Przyjmij że katalog `/tmp/etc` nie istnieje.

5 Praca domowa

5.1 Instrukcja wysyłania rozwiązań

Rozwiązania zadań domowych należy przesłać na adres licealisci.pracownia@icm.edu.pl wpisując jako temat wiadomości g2.x PD6, gdzie x to numer grupy, np. g2.1 PD6 dla grupy nr. 1, itd. Zadania domowe są nie obowiązkowe, jednak zachęcamy do ich robienia i wysyłania rozwiązań (nawet niekompletnych).

Termin nadsyłania zdań domowych to 2020-04-07 godzina 16⁰⁰. Jeżeli wysłałeś rozwiązania w terminie, ale nie były one w 100% poprawne i dostałeś od sprawdzającego możliwość wysłania poprawki masz na to dodatkowe 4 dni.

Na ten adres można także nadsyłać ewentualne pytania do zadań (zarówno domowych jak i innych zamieszczonych w skrypcie), w tym wypadku także prosimy o umieszczenie w temacie wiadomości g2.x, gdzie x to numer grupy.

5.2 Zadania domowe

Zadanie 5.2.1 — 1 pkt

Napisz polecenie które dla wszystkich plików z rozszerzeniem .TXT w bieżącym katalogu (bez podkatalogów) dokona zmiany ich nazwy zmieniając rozszerzenie na .txt, zachowując podstawową część nazwy bez modyfikacji. W rozwiązaniu nie korzystamy z polecenia rename.

Zadanie 5.2.2 — 2 pkt

Napisz polecenie które wyszuka i przekopiuje do katalogu /tmp pliki z katalogu /etc (wraz z jego podkatalogami), które (w swojej treści) zawierają napis nameserver.

Zadanie 5.2.3 — 3 pkt

Plik /proc/cmdline zawiera informację o opcjach przekazanych do jądra podczas startu. Kolejne opcje rozdzielane są spacją, a nazwę opcji od jej argumentu rozdziela znak równości. Napisz polecenie które wypisze argument opcji root. Dla pliku /proc/cmdline postaci:

```
BOOT_IMAGE=/vmlinuz-4.9-amd64 root=UUID=cad866ab-aabd-4686-8376-e4b9f1c2ae9e rw  
polecenie powinno wypisać:  
UUID=cad866ab-aabd-4686-8376-e4b9f1c2ae9e
```