Bash, zmienne i instrukcje warunkowe

Zmienne możemy traktować jak etykiety, które przechowują ciągi znaków. Definiujemy je przy pomocy znaku `=` **bez spacji po obu jego stronach**:

```
dog="Azor"
```

Do zmiennych w poleceniach odwołujemy się poprzedzając je symbolem `$`. Okazuje się, że to, czy nazwę zmiennej umieścimy pomiędzy apostrofami, czy też cudzysłowami, ma znaczenie – cudzysłów umożliwia specjalną interpretację znaku `$`.

```
echo $dog
⟶Azor
echo $dogek
⟶
echo ${dog}ek
⟶Azorek
echo 'Jestem $LOGNAME, a to jest pies $dog.'
⟶Jestem $LOGNAME, a to jest pies $dog.
echo "Jestem $LOGNAME, a to jest pies $dog."
⟶Jestem boraks, a to jest pies Azor.
```

Powyżej pojawiła się, zawierająca nazwę użytkownika, zmienna `LOGNAME` – jedna ze *zmiennych środowiskowych*. Są to zmienne, których zawartość jest dostępna dla wszystkich procesów uruchomionych w powłoce. Ich nazwy zapisujemy dużymi literami. Przykłady innych zmiennych środowiskowych:

`HOSTNAME`	nazwa (adres) maszyny, na którą się zalogowaliśmy,
`HOME`	ścieżka do katalogu domowego użytkownika,
`PATH`	ścieżki (oddzielone dwukropkami), w których szukane są programy powłoki,
`PWD`	ścieżka do aktualnego katalogu.

Wszystkie zmienne środowiskowe możemy wypisać komendą `env` (choć nie jest to jej główne przeznaczenie – patrz `man env`). W praktyce opłaca się nam często zmodyfikować zmienną `PATH` w pliku `~/.bashrc` w celu szybkiego uruchamiania naszych programów (skryptów) z danego katalogu:

```
PATH=$PATH:$HOME/scripts
```

Jeżeli w katalogu `scripts` znajdował się program o nazwie `cmd`, to od tej pory będziemy go mogli uruchomić w dowolnym miejscu wpisując po prostu `cmd`.

0.1 Instrukcje sterujące

W bash-u występują dwie instrukcje warunkowe `if then else` oraz `case`. Przyjrzyjmy się ich składni.

```
if [ warunek ] ; then
```

```
    polecenia
elif [ warunek ] ; then
    polecenia
else
    polecania
fi
```

Instrukcja `else` jest opcjonalna i nie musi być używana. To samo dotyczy `elif`, która zastępuje `else if (warunek)`, jak to piszemy używając *C* *C++*. Istotną różnicą jest że w bashu nie używa się bloków `{}`, dlatego instrukcja `if` musi być zakończona przez `fi`. Przykład sprawdzający pierwszy czy argument skryptu jest równa zeru:

```
#!/bin/bash
if [ $1 -eq 0 ] ; then
    echo "$i jest równe zeru"
else
    echo "$1 jest różne od zera"
fi
```

Gdy musimy dla kolejnych wartości zmiennej wykonywać różne akcje, instrukcja `if` staje się niewygodna w użyciu. Do takich rozwiązań służy instrukcja `case` o następującej składni:

```
case zmienna in
    "wzorzec") polecenie ;;
    "wzorzec") polecenie ;;
    "wzorzec") polecenie ;;
    *) polecenie wykonywane domyślnie
esac
```

W instrukcji `case` ilość wzorców może być dowolna. Przykład wypisujący odpowiednią informację zależną od pierwszego argumentu programu:

```
#!/bin/bash
case $1 in
    "0") echo "Jest OK" ;;
    "1") echo "Jest lepiej" ;;
    "2") echo "Jest jeszcze lepiej" ;;
    "3") echo "Jest super" ;;
    *) echo "Przesadzasz!"
esac
```

W bash-u istnieją kilka rodzajów pętli: `for`, `while`, `until` oraz `select`. Omówimy tutaj pierwsze trzy z nich. Pętla `while` wykonuje swoje działanie dopóki warunek jest spełniony:

```
while [ warunek ] ; do
    polecenie
done
```

Przykład użycia, wypisanie 10 razy zmiennej `i`:

```
#!/bin/bash
i=1;
while [ $i -le 10 ] ; do
    echo "$i"
    i=$[i + 1]
done
```

Pętla `until` różni się od poprzedniej tym że kończy swoje działanie w momencie gdy warunek stanie się prawdziwy:

```
until [ warunek ] ; do
    polecenie
done
```

Przykład użycia, wypisanie 10 razy zmiennej `i`:

```
#!/bin/bash
i=0
until [ $i -ge 10 ] ; do
    echo "$i"
    i=$[i + 1]
done
```

Pętla `for` jest najciekawsza z bash-owych pętli, ponieważ może być używana na dwa sposoby. Po pierwsze możemy jej używać tak jak w `C\C++`, kiedy znamy ograniczenie na ilość iteracji:

```
for ((inicjalizacja zmiennej; warunek zakończenia; zmiana zmiennej)) ;
do
    polecenia
done
```

Przykład, wypisanie 10 razy wartości zmiennej `i`:

```
#!/bin/bash
for (( i=1; $i <= 10; i++ )) ; do
    echo " Iteracja nr: $i"
done
```

Drugie to iterowanie się po pewnej liście zbiorze elementów:

```
for zmienna in lista ; do
    polecenia
done
```

Przykład, wypisanie wszystkich małych liter alfabetu angielskiego:

```
for i in {a..z} ; do
    echo $i
done
```

Aby przerwać działanie pętli uzywamy komendy `break`. Natomiast aby przeskoczyć do kolejnej iteracji służy polecenie `continue`.

Podstawowe warunki logiczne w bashu

Polecenie `test warunek_1 operator warunek2` sprawdza wartość wyrażenia. Polecenie `test` może być zapisane za pomocą nawiasów kwadratowych, które tak naprawdę są aliasem do tego polecenia:

`[ wyrażenie1 operator wyrażenie2 ]`

Operacje arytmetyczne:

`-eq`	równy
`-ne`	różny
`-lt`	mniejszy niż
`-le`	mniejszy lub równy
`-gt`	większe niż
`-ge`	większe lub równy

Operacje operujące na stringach:

`=`	równy
`!=`	różny
`<`	pierwszy string alfabetycznie przed drugim
`>`	pierwszy string alfabetycznie za drugim
`-n`	string ma długość większą niż 0
`-z`	string ma zerową długość

Łączenie wyrażeń odbywa się przez alternatywę `||` lub koniunkcję `&&`:

```
warunek1 || warunek2 || warunek3 ...
```

```
warunek1 && warunek2 && warunek3 ...
```

Wyrażenia są sprawdzane po kolei, aż do momentu kiedy da się już określić wartość logiczną całego wyrażenia. Dlatego alternatywa wykonuje się do momentu gdy jakieś wyrażenie zwróci `TRUE` albo do końca i jeśli wszystkie wyrażenia dadzą `FALSE` to cała alternatywa też będzie fałszywa. Natomiast koniunkcja odwrotnie, przerywana jest w momencie gdy jakieś wyrażenie zwróci `FALSE`, a jeśli nie to wykonuje się do końca i jeśli wszystkie wyrażenia dadzą `TRUE` to koniunkcja będzie prawdziwa.

Wczytywanie danych ze standardowego wejścia

Prostą komendą czytającą pojedynczą linię wejścia jest
`read -opcje nazwa_zmiennej_do_której_czytamy`.

Przykład, wczytywanie napisów i łączenie ich w jeden:

```
#!/bin/bash
while read zmienna
do
napis=$napis" "$zmienna
done
echo $napis
```

Ewaluacja wyrażeń arytmetycznych w bashu

Istnieje wiele sposobów na wyliczanie wartości wyrażenia w bash-u. Zwykłe wpisanie ich w instrukcji `echo` jednak nie wystarczy. Umieszczenie wyrażenia pomiędzy nawiasami: `$((wyrażenie))` wykona je, ale jest ograniczone tylko do liczb całkowitych.

```
echo "2+2"
2+2
echo $((7+7))
```

14

Aby wykonać wyrażenia zmiennoprzecinkowe używamy polecenia `bc -l`:

```
echo "2+2"| bc
4
echo "3+5.5"| bc -l
8.5
echo "2/5"| bc
0
echo "2/5"| bc -l
.40000000000000000000
```

Używając parametru `scale` możemy określić liczbę liczb po przecinku, nie musimy wtedy używać `-l`. Należy jednak pamiętać, że ważna jest kolejność wyrażeń!

```
echo "scale = 4; 11 * 110 / 70" | bc
17.2857
echo "scale = 4; 11 / 70 * 110" | bc
17.2810
echo "scale = 4; (11 / 70) * 110" | bc
17.2810
```