Kurs pisania skryptów w powłoce BASH

1. Wstęp

Skrypt shellowy to nic innego jak bardziej rozbudowana wersja listy pewnych poleceń, które mają zostać wykonane przez system. Zgodnie z ogólnie przyjętą konwencją, każdy powstający skrypt (działający w BASH-u) powinien się zaczynać od następującej linijki:

#!/bin/bash

Linia ta jest o tyle istotnie, że przy jej pomocy definiujemy powłokę, która będzie odpowiedzialna za wykonanie ciągu instrukcji umieszczonych wewnątrz skryptu.
Pomimo tego, że pominiecie tej linii spowoduje, że skrypt będzie działał, zaleca się jej stosowanie, ponieważ składnia rozmaitych powłok może się nieznacznie różnić, a w rezultacie uruchomiony skrypt może wyświetlać różne wyniki (będzie raz działać, a raz nie) w zależności od tego, pod jaką powłoką zostanie uruchomiony.

2. Pierwszy skrypt.

Jak to zazwyczaj bywa podczas nauki pisania programów, pierwszym wykonanym programem przez osobę pragnącą zgłębić tajniki danego jeżyka programowania jest program wyświetlający krótki tekst na monitorze. Aby nie odstępować od tych „powszechnie przyjętych norm", także w Bash-u pierwszym skryptem, który zostanie napisany w tym kursie będzie skrypt, który wyświetli na ekranie monitora króciutki tekst.
Skrypt, który to wykona jest niezwykle prosty i składa się z dwóch linijek kodu:

#!/bin/bash
echo "Moj pierwszy skrypt"

Uruchomienie tego skryptu spowoduje wyświetlenie na ekranie monitora komunikatu o treści:

Moj pierwszy skrypt

a następnie przejście do następnej linii (aby pozostać w tej samej linii należałoby po słowie echo wstawić parametr –n co wyglądałoby tak: echo –n "Mój pierwszy skrypt").
Jak się można łatwo domyśleć polecenie echo odpowiada za wyświetlenie tekstu na ekranie monitora.

Bardzo często stosowaną praktyką wśród programistów jest komentowanie poszczególnych linii powstającego kodu. Także i w BASH-u jest to możliwe, a dokonuje się tego przy użyciu znaku # co w rezultacie wygląda następująco:

#!/bin/bash
echo "Moj pierwszy skrypt" #wyświetlenie tekstu na monitorze

Pewnie dla większości osób czytających ten tekst, a które miały już wcześniej styczność z programowaniem (chociażby w Pascalu) od razu rzuca się w oczy, że ciąg instrukcji nie jest zakończony znakiem średnika, a same źródło skryptu nie znajduje się pomiędzy słowami kluczowymi begin i end. Są to jedne z wielu różnic, które występują pomiędzy „zwykłym programowaniem", a pisaniem skryptów powłoce BASH.

UWAGA
Po zapisaniu skryptu do pliku należy pamiętać o nadaniu temu plikowi odpowiednich praw do jego wykonywania w przeciwnym przypadku skrypt nie zadziała.

3. Zmienne.

W każdym języku programowania potrzebne są zmienne, aby przy ich pomocy wykonać pewne operacje. Także w skryptach basha-a można używać zmiennych, a definiuje się je w następujący sposób:

ZMIENNA=witam # jeżeli w wartości zmiennej nie ma spacji nie trzeba używać znaku cudzysłowie

lub

ZMIENNA=12345 # jeżeli przypisujemy liczby lub cyfry nie używamy znaku cudzysłowie

lub

ZMIENNA="witam was" # jeżeli w wartości zmiennej znajduje się znak spacji, to wartość tą trzeba umieścić w cudzysłowie w przeciwnym wypadku będzie to zinterpretowane jako błędny zapis.

Zmienne można zadeklarować w dowolnym dla programisty momencie, a ponadto nie trzeba nadawać im odpowiedniego typu tak jak to ma miejsce w innych językach programowania (np. pascal, c , c++). Podczas deklarowania zmiennych i przypisywania im pewnych wartości należy zwrócić uwagę na jedną bardzo istotną rzecz, a mianowicie po obu stronach znaku = (który służy do przypisania wartości do zmiennej) nie mogą znajdować się spacje ponieważ spowoduje to powstanie błędu.

ZMIENNA = witam #błędny zapis !!!

Powyższy zapis jest błędny, ponieważ shell interpretuje linię poleceń jako komendę i jej argumenty, które są rozdzielone znakiem spacji, zatem zapis ZMIENNA = witaj zostanie zinterpretowany w ten sposób, że słowo ZMIENNA będzie uważane za polecenie, natomiast znak '=' i słowo 'witaj' będą uważane za parametry do tego polecenia. Zapis ZMIENNA=witaj zostanie zinterpretowane jako polecenie, co jest jak najbardziej poprawne.

Kolejną istotną rzeczą, o której należy wspomnieć przy omawianiu zmiennych, jest fakt, że ich nazwy pisze się dużymi literami (oczywiście jak ktoś nazwę zmiennej napisze małymi literami to się nic nie stanie i skrypt będzie działał poprawnie, ale przy programowaniu skryptów przyjęło się, że nazwy zmiennych są pisane dużymi literami).
Bardzo istotny jest również fakt, że BASH rozróżnia duże i małe litery, zatem odwoływanie się do zmiennej odbywa się przez użycie dokładnie takiej samej nazwy zmiennej przy pomocy jakiej ta zmienna została zadeklarowana (jeżeli nazwa zadeklarowanej zmiennej jest napisana dużymi literami, a odwołanie do niej odbywa się poprzez podanie jej nazwy z małych liter to oczywiście odwołanie to nie przyniesie oczekiwanego rezultatu, ponieważ taka zmienna nie istnieje).

Odwołanie się do wartości zadeklarowanej wcześniej zmiennej odbywa się w następujący sposób:

ZMIENNA=witaj #deklaracja zmiennej
$ZMIENNA #odwołanie się do zmiennej w sposób prawidłowy
$zmienna #odwołanie się do zmiennej w sposób nieprawidłowy, ponieważ nazwa wcześniej zadeklarowanej zmiennej została podana z dużych liter, a podczas odwołania się do niej jej nazwa jest podana małymi literami, w rezultacie zostanie to zinterpretowane jako odwołanie się do nowej zmiennej, której nazwa jest napisana małymi literami.

W Bashu podczas wyświetlania teksu na ekranie monitora można się spotkać z dwoma konstrukcjami:

- echo ”Mój pierwszy skrypt”
- echo ‘Mój pierwszy skrypt’

w obu, przedstawionych powyżej przypadkach efekt będzie taki sam – na ekranie monitora pokaże się napis:

Mój pierwszy skrypt

jednakże różnicę w tych konstrukcjach zobaczymy na przykładzie prostego skryptu:

```
#!bin/bash
ZMIENNA=”Mój pierwszy skrypt”
echo ”$ZMIENNA”
echo ‘$ZMIENNA’
```

Po uruchomieniu powyższego skryptu na ekranie monitora ukaże się następujący rezultat:

Mój pierwszy skrypt
$ZMIENNA

zatem zasada jest bardzo prosta: wewnątrz cudzysłowie nazwy zmiennych które są poprzedzone znakiem $ są zastępowane przez przypisaną im wartość, natomiast wewnątrz apostrofów nie.

Niekiedy zachodzi potrzeba, aby nazwy zmiennych ująć w cudzysłowy. Jest to istotne, gdy wartość zmiennej zawiera spacje lub jest pustym ciągiem. Aby to zobrazować należy prześledzić działanie następującego skryptu:

```
#!bin/bash
ZMIENNA=””
if [ -n $ZMIENNA ]; then # -n sprawdza czy argument nie jest pusty
        echo ” Zmienna o nazwie ZMIENNA nie jest pusta”;
else
        echo ” Zmienna o nazwie ZMIENNA jest pusta”;
fi
```

Uruchomienie powyższego skryptu spowoduje, że na ekranie monitora pojawi się napis:

Zmienna o nazwie ZMIENNA nie jest pusta

Zdziwieni ? Dzieje się tak dlatego, że shell zamienił $ZMIENNA na wartość zmiennej czyli na ciąg pusty co nie jest argumentem, zatem jeżeli nie ma argumentu to skrypt zwraca prawdę.
Aby skrypt działał poprawnie to musiałby wyglądać w następujący sposób:

```
#!bin/bash
ZMIENNA=""
if [ -n "$ZMIENNA" ]; then
        echo " Zmienna o nazwie ZMIENNA nie jest pusta"
else
        echo " Zmienna o nazwie ZMIENNA jest pusta"
fi
```

W rezultacie na ekranie monitora wyświetli się wynik:

Zmienna o nazwie ZMIENNA jest pusta.

Będzie się tak działo ponieważ shell zinterpretuje warunek w następujący sposób: [-n ""] (wcześniej interpretacja wyglądała tak [-n]).
Podoba sytuacja jest w następującym przypadku:

```
ZMIENNA="Ala ma kota"
if [ -n $ZMIENNA ]; then
…
```

Shell zinterpretuje warunek jako [-n Ala ma kota] co jest zapisem błędnym. Aby zapis był poprawny należy go wykonać w następujący sposób

```
ZMIENNA="Ala ma kota"
if [ -n "$ZMIENNA" ]; then
…
```

W tym przypadku shell zinterpretuje warunek jako: [-n "Ala ma kota"] co jest zapisem poprawnym.

Mówiąc o zmiennych należy rozważyć jeszcze jeden przypadek, a mianowicie jak wypisać na ekranie monitora wartość danej zmiennej, a bezpośrednio za tą wartością (bez znaku spacji) wyświetlić jakiś tekst, np. wyświetlić wartość ZMIENNA, a bezpośrednio za tą wartością słowo 'stopni'. Sprawa może wydaje się prosta i na pierwszy rzut oka pewnie większość osób napisałaby taki skrypt w następujący sposób:

```
#!/bin/bash
ZMIENNA=49
echo "$ZMIENNAstopni"
```

Niestety po uruchomieniu powyższego skryptu ekran monitora pozostanie pusty. Dlaczego ? ano dlatego, że shell zinterpretował ten skrypt jako próbę wyświetlenia wartości zmiennej o nazwie ZMIENNAstopni, która przecież nie została zadeklarowana czyli nie istnieje. Aby skrypt dał poprawny rezultat to nazwę zmiennej trzeba oddzielić od tekstu nawiasami klamrowymi {}. Poprawny skrypt powinien więc wyglądać następująco:

```
#!/bin/bash
ZMIENNA=49
echo "${ZMIENNA}stopni"
```

Po jego wywołaniu na ekranie monitora pojawi się napis:

49stopni

Przy pomocy zmiennych możemy również wywoływać polecenia linux'a. Dzieje się tak dlatego, że shell rozwija zmienne. Aby to zrozumieć należy prześledzić działanie poniższego skryptu.

```
#!/bin/bash
Z1="ls"
Z2="-l"
$Z1 $Z2
```

Wywołanie tego skryptu spowoduje wykonanie przez shella następującego polecenia:

ls –l

Do tej pory nie został jeszcze omówiony sposób przekazywania do zmiennej wartości, która została wprowadzone przez użytkownika przy pomocy klawiatury. Aby tego dokonać należy użyć polecenia read, a jego działanie ilustruje poniższy skrypt:

```
#!/bin/bash
echo "Podaj swoje imie:"
read IMIE
echo "Na imie masz: $IMIE"
```

Powyższy skrypt działa następująco: na ekranie monitora pojawi się napis o treści 'Podaj swoje imie:' po czym skrypt będzie oczekiwał aż użytkownik wpisze z klawiatury swoje imię (np. Krzysiek) i przyciśnie klawisz ENTER, a następnie na ekranie monitora pojawi się napis o treści 'Na imie masz: Krzysiek' co zakończy działanie uruchomionego skryptu.

Na zmiennych można wykonywać również proste operacje matematyczne. Odbywa się to w następujący sposób:

```
#!/bin/bash
echo "Podaj pierwsza liczbe:"
read L1
echo "Podaj druga liczbe:"
read L2
SUMA=$(($L1 + $L2))
```

echo ”wynik dodawania wynosi: $SUMA”

4. Parametry.

Do wywoływanego skryptu można przekazywać parametry jego wywołania. Parametrów można używać dokładnie tak samo jak zmiennych jednakże nie można zmienić ich wartości. Każdy z parametrów wywołania posiada swój numer, dzięki czemu programista w bardzo prosty sposób może odwołać się do każdego z nich. Odwołanie się w treści skryptu do parametrów odbywa się w następujący sposób:

S* - wszystkie parametry począwszy od parametru numer 1
$0 – nazwa programu
$1 – parametr nr 1
…
$9 – parametr nr 9
$# - liczba parametrów
$$ - numer procesu

Działanie parametrów zilustruje poniższy skrypt:

```
#!/bin/bash
echo ”nazwa skryptu: $0”
echo ”liczba parametrow: $#”
echo ”wszystkie parametry wywolana: $*”
echo ”parametr pierwszy: $1”
echo ”parametr drugi: $2”
echo ”parametr trzeci: $3”
echo ”numer procesu: $$”
```

Po zapisaniu powyższego skryptu do pliku o nazwie skrypt1 jego wywołanie wygląda następująco:

./skrypt1 123 434 2312

a wynik jego działania wygląda tak:

nazwa skryptu: ./skrypt1
liczba parametrow: 3
wszystkie parametry wywolania: 123 434 2312
parametr pierwszy: 123
parametr drugi: 434
parametr trzeci: 2312
numer procesu: 138

Oczywiście jest to najprostszy przykład użycia parametrów, ponieważ parametry można wykorzystywać w bardziej skomplikowany sposób niż zaprezentowany tutaj.

5. Instrukcja CASE.

Podobnie jak w innych językach programowania tak i w skryptach pisanych pod basha można używać instrukcji CASE. Składnia tej instrukcji wygląda następująco:

```
case $ZMIENNA in
        1) zadanie numer 1;;
        2) zadanie numer 2;;
        *) zadanie w każdym innym przypadku;;
esac
```

Instrukcja case zaczyna się od słowa case, a kończy się zapisaniem tego słowa od końca (esac), a pomiędzy tymi słowami wypisane są wszystkie opcje obsługiwane przez tą instrukcję. Zastosowanie konstrukcji *) obejmuje wszystkie pozostałe możliwości, które nie zostały zdefiniowane w instrukcji case.

Działanie tej instrukcji obrazuje poniższy skrypt:

```
#!/bin/bash
case $1 in
        piatek) echo "weekend jest juz tuz tuz";;
        sobota) echo "juz weekend";;
        niedziela) echo "weekend się juz powoli konczy";;
        poniedzialek) echo "już poniedziałek znowu trzeba isc do pracy";;
        *) echo „srodek tygodnia trzeba ciezko pracowac";;
esac
```

Wywołanie tego skryptu może wyglądać następująco:

a) ./nazwa_skryptu piątek - wyświetlenie tekstu ‘weekend jest już tuz tuz’,
b) ./nazwa_skryptu sobota - wyświetlenie teksu ‘już weekend’
c) ./nazwa_skryptu niedziela - wyświetlenie teksu ‘weekend się już powoli konczy’
d) ./nazwa_skryptu poniedzialek - wyświetlenie teksu ‘już poniedziałek znowu trzeba isc do pracy
e) ./nazwa_skryptu <jakikolwiek inny dzień tygodnia> - wyświetlenie teksu ‘srodek tygodnia trzeba ciezko pracowac’

6. Instrukcja warunkowa IF.

W instrukcji warunkowej if wyróżnia się trzy zasadnicze składnie tego polecenia:

a) w przypadku, gdy zachodzi potrzeba sprawdzenia jakiegoś warunku np. czy dwie liczby są równe składnia wygląda następująco:

```
if warunek; then
        wyrażenie1;
        wyrażenie2;
fi
```

b) w przypadku gdy należy wykonać inny zestaw poleceń, jeżeli zadany warunek kończy się wynikiem negatywnym, składnia polecenia wygląda następująco:

```
if warunek; then
        wyrażenie1;
else
        wyrażenie2;
fi
```

c) w przypadku gdy należy sprawdzić inny warunek jeżeli wcześniejszy warunek nie został spełniony, składnia polecenia if wygląda następująco (ilość wyrażeń elif jest dowolna):

```
if warunek1; then
        wyrażenie;
elif warunek2; then
        wyrażenie2;
fi
```

Instrukcja warunkowa if działa w taki sposób, że polecenia wewnątrz bloku if/fi, if/elif, a następnie elif/fi jeśli zadany warunek jest prawdziwy.

7. Zapis warunków w instrukcji warunkowej IF.

Do testowania warunków używa się operatorów, które w wyniku swojego działania zwracają prawdę lub fałsz w zależności od tego czy sprawdzany warunek jest prawdziwy czy nieprawdziwy. Zazwyczaj warunek jest zapisywany w następującej postaci:

[operand1 operator operand2]

W niektórych przypadkach warunki zapisuje się tylko przy pomocy jednego operandu (drugiego - operand2). Bardzo istotny jest fakt, że pomiędzy nawiasami, operandami operandami operatorem musi istnieć odstęp w postaci spacji. Jeżeli tego odstępu nie ma to wtedy wszystko to co jest napisane między nawiasami kwadratowymi jest traktowane jako jeden operand bez jakichkolwiek operatorów.

```
[$X=$Y] /# błędny zapis !!!
[ $X = $Y ] # zapis prawidłowy
```

Ponadto bardzo istotne jest także używanie znaków cudzysłowu we wszystkich warunkach w których operator przyjmuje postać –n (na początku kursu zostało wyjaśnione dlaczego).

8. Operatory instrukcji warunkowej IF.

Najczęściej używanymi operatorami w instrukcji warunkowej if są:

-n – operand ma niezerową długość (jeden operand)
-z – operand ma zerową długość (jeden operand)
-d – istnieje katalog o nazwie operand (jeden operand)
-f – sprawdza czy operand jest plikiem (jeden operand)

-e – sprawdza czy zbiór (plik, katalog) o nazwie operand (jeden operand) istnieje
-L – sprawdzenie czy plik o nazwie operand jest dowiązaniem symbolicznym (jeden operand)
-eq – sprawdza czy operandy są równymi liczbami (dwa operandy)
-neq – sprawdza czy operandy są różnymi liczbami (dwa operandy)
= - sprawdza czy operandy są jednakowymi ciągami znaków (dwa operandy)
!= - sprawdza czy operandy są różnymi ciągami znaków (dwa operandy)
-lt – sprawdza czy operand 1 jest mniejszy operand2 (dwa operandy które są liczbami całkowitymi)
-le – sprawdza czy operand1 jest równy lub mniejszy od operand2 (dwa operandy które SA liczbami całkowitymi)
-gt – sprawdza czy operand1 jest większy od operand2 (dwa operandy które SA liczbami całkowitymi)
-ge – sprawdza czy operand1 jest równy lub większy od operand2 (dwa operandy które SA liczbami całkowitymi)

Oczywiście nie są to wszystkie typy operatorów, które są dostępne w bashu, ale w zupełności wystarczają do napisania większości skryptów. Oczywiście istnieje możliwość poznania reszty operatorów które są dostępne w bashu. Aby to uczynić należy skorzystać z manuala wpisując polecenie:

man bash

A oto prosty przykład skryptu wykorzystującego instrukcję warunkową if

```
#!/bin/bash
if [ $1 –eq $2 ]; then
        echo ”liczby $1 i $2 sa rowne”;
else
        echo ”liczby $1 i $2 sa rozne”;
fi
if [ -e $3 ]; then
        echo ”plik o nazwie $3 istnieje”;
        if [ -L $3 ]; then
                echo ”i jest dowiązaniem symbolicznym”;
        elif [ -f $3 ]; then
                echo ”i jest zwykłym plikiem”;
        fi
else
        echo ”plik o nazwie $3 nie istnieje”
fi
```

Uruchomienie powyższego skryptu wygląda następująco:

./nazwa_skryptu 1 3 test.txt

Wynik działania skryptu wygląda następująco (zakładając ze plik podany jako trzeci parametr istnieje i że jest zwykłym plikiem):

liczby 1 i 3 sa rozne
plik o nazwie test.txt istnieje

i jest zwykłym plikiem

9. Pętle.

Jak wiadomo pętle stosuje się w celu wykonania pewnych instrukcji dla kolejnych iteracji lub kilku parametrów. W bashu przy tworzeniu skryptów również możemy się posługiwać pętlami. Do najczęściej stosowanych pętli zalicza się pętle for i while, które zostaną omówione.

FOR

Składnia pętli for wygląda następująco:

```
for ZMIENNA in …; do
        instrukcje;
done
```

Przykładem użycia pętli for może być skrypt, który wyświetli, linijka po linijce, wszystkie parametry, z jakimi ten skrypt został uruchomiony niezależnie od liczby tych parametrów. Kod takiego skryptu wygląda następująco:

```
#!/bin/bash
for ZMIENNA in $*; do
        echo “$ZMIENNA”;
done
```

Oczywiście pętle for można używać na wiele różnych sposobów np.:

```
#!/bin/bash
DZIEN1=poniedziałek
DZIEN2=wtorek
DZIEN3=środa
for ZMIENNA in ”$DZIEN1” ”$DZIEN2” ”$DZIEN3”; do
        echo ”Dzisiaj jest: $ZMIENNA”;
done
```

Jak pokazuje powyższy skrypt, pętla for wykonuje instrukcje zawarte wewnątrz pętli na parametrach, które są oddzielone spacjami dlatego też zaleca się umieszczanie tych parametrów cudzysłowach chyba, że parametry te nie zawierają znaku spacji.

WHILE

Składnia pętli while wygląda następująco:

```
while warunek; do
        instrukcje;
done
```

W pętli while warunek zbudowany jest dokładnie tak samo jak w instrukcji warunkowej IF (ta sama konstrukcja te same operatory).

Pętla while działa tak długo dopóki warunek jest prawdziwy w przeciwnym razie pętla kończy swoje działanie. Do sterowania pętlą while istnieją dwie instrukcje:

- break – przerwanie wykonywania pętli
- continue – wymuszenie kolejnej iteracji pętli

Przykładowy skrypt wykorzystujący pętlę while może wyglądać następująco:

```
#!/bin/bash
if [ $1 –gt $2 ]; then
        X=$1;
        while [ $X –gt $2 ]; do
                ROZNICA=$(($X - $2));
                echo ”liczba $X jest wieksza od liczby $2 o $ROZNICA”;
                X=$(($X – 1));
        done
else
        echo ”liczba $1 nie jest wieksza od liczby $2”;
fi
```

Powyższy skrypt działa w następujący sposób: jeżeli pierwszy parametr uruchomienia skryptu jest równy lub mniejszy od parametru drugiego z którym skrypt został uruchomiony to na ekranie monitora wyświetli się napis ‘liczba <wartość pierwszego parametru> nie jest większa od liczby <wartość drugiego parametru>’ w przeciwnym razie na ekranie monitora będzie się wyświetlał napis ‘liczba <wartość aktualna liczby> jest większa od liczby <wartość drugiego parametru> o <różnica między liczbą pierwszą a drugą>’ po czym wartość pierwszej liczby będzie zmniejszana o jeden. Pętla zakończy swoje działanie, kiedy warunek
[$X –gt $2] będzie nieprawdziwy czyli wtedy kiedy zmienna X osiągnie taką samą wartość jak zmienna $2.

UNTIL

Pętla until na dokładnie taką samą składnie jak pętla while i działa tak samo jak pętla while z tą różnica, że warunek znajdujący się w pętli until jest zanegowany. Pętla until działa tak długo dopóki warunek jest nieprawdziwy w przeciwnym razie pętla kończy swoje działanie. Zgodnie z tym składnia pętli until wygląda następująco:

```
until warunek; do
        instrukcje;
done
```

Przykładowy skrypt wykorzystujący pętlę while może wyglądać następująco:

```
#!/bin/bash
X=$1
until [ $X –gt $2 ]; do
        echo ”liczba $X jest mniejsza od liczby $2”;
        X=$(($X + 1));
Done
```

Nietrudno zauważyć, że skrypt ten będzie wypisywał na ekranie monitora tekst ‘liczba <pierwszy parametr skryptu> jest mniejsza od liczby <drugi parametr skryptu>’, po którym następuje zwiększenie o jeden pierwszego parametru skryptu, dopóki pierwszy parametr nie będzie równy lub większy od drugiego parametru skryptu.

10. Użycie znaków globalnych w pętlach

Wspominając o pętlach, których implementację możemy stosować w skryptach shellowych należy również wspomnieć o znaku ‘*’. Otóż wszystkie parametry zawierające znak ‘*’ zastępowane są listą plików, które pasują do zadanego wzorca (sam znak ‘*’ jest zastępowany listą wszystkich plików znajdujących się w bieżącym katalogu za wyjątkiem plików, których nazwa zaczyna się od kropki).
Zasadę działania znaku ‘*’ zobrazuje poniższy skrypt:

```
#!/bin/bash
echo ”Oto zawartość katalogu ${PWD}: ”
echo ${PWD}/*
```

W wyniku wykonania w/w skryptu zostaną wyświetlony wszystkie pliki znajdujące się w katalogu bieżącym. Oczywiście istnieje również możliwość wykorzystania znaku ‘*’ w zakładaniu tzw. maski:

```
#!/bin/bash
echo ”Oto wykaz wszystkich plików z katalogu ${PWD}, których nazwa zaczyna się na litere
a: ”
echo ${PWD}/a*
```

Wykonanie powyższego skryptu spowoduje wypisanie, z katalogu bieżącego, wszystkich plików, których nazwa zaczyna się na literę ‘a’.

11. Podstawianie poleceń

Jednym z bardziej przydatnych i wygodnych mechanizmów basha jest możliwość pobierania danych, które zostały otrzymane w wyniku działania jakiegoś polecenia linuxowego i traktowanie ich tak jakby zostały wprowadzone z klawiatury. Mechanizmem tym jest mechanizm podstawiania poleceń.
Istnieją dwa sposoby podstawiania poleceń:

- rozwijanie zawartości nawiasów
- ujmowanie polecenia w tzw. wsteczne apostrofy.

Rozwijanie zawartości nawiasów odbywa się w następujący sposób: $(polecenie_do_wykonania) – zapis taki zostaje zastąpiony przez wynik wykonania polecenia znajdującego się w nawiasach. Istotny jest fakt, że nawiasy mogą być zagnieżdżone, czyli polecenia również mogą zawierać nawiasy.

Ujmowanie polecenia w tzw. wsteczne apostrofy:
`polecenie_do_wykonania` - zapis taki zostaje zastąpiony przez wynik wykonania polecenia znajdującego się w nawiasach.

Poniżej znajduje się przykład skryptu, który obrazuje zasadę działania mechanizmu podstawiania poleceń:

```
#!/bin/bash
DANE=”$(ls –l)”
echo ”Wynik metody rozwijania zawartosci nawiasow:”
echo ”$DANE”
DANE=`cat test.txt`
echo ”Wynik metody ujmowania polecenia w tzw. wsteczne apostrofy:”
echo ”$DANE”
```

12. Operacje arytmetyczne

Omawiając problematykę pisania skryptów działających w powłoce bash należy również wspomnieć o możliwości wykonywania obliczeń arytmetycznych, do których używa się operatora expr odpowiadającego za wykonanie wszystkich operacji. Jego użycie wygląda w następujący sposób:

WYNIK_DZIALANIA=`expr 13 – 7` # za zmienna WYNIK_DZIALANIA zostanie podstawiony wynik odejmowania liczby 7 od liczby 13.

Zapis operacji matematycznych:

a + b odpowiednik `expr a + b` #dodawanie
a - b odpowiednik `expr a - b` #odejmowanie
a * b odpowiednik `expr a \* b` #mnożenie
a / b odpowiednik `expr a / b` #dzielenie
a % b odpowiednik `expr a % b` #reszta z dzielenia a przez b

Przykładowy skrypt wykonujący podstawowe działania matematyczne:

```
#!/bin/bash
A=$1
B=$2
echo –n ”Wynik dodawania liczby $1 do liczby ${2}: ”
WYNIK=`expr $A +$B`
echo”$WYNIK”
echo –n ”Wynik odejmowania liczby $1 od liczby ${2}: ”
WYNIK=`expr $B - $A`
echo”$WYNIK”
echo –n ”Wynik mnozenia liczby $1 przez liczbe ${2}: ”
WYNIK=`expr $A \* $B`
echo”$WYNIK”
echo –n ”Wynik dzielenia liczby $1 przez liczbe ${2}: ”
WYNIK=`expr $A / $B`
echo”$WYNIK”
echo –n ”Reszta z dzielenia liczby $1 przez liczbe ${2}: ”
WYNIK=`expr $A % $B`
echo”$WYNIK”
```

13. Funkcje

Pisząc skrypty w bashu programista może również tworzyć funkcje, które w bardzo dużej mierze przyczyniają się do skrócenia kodu skryptu i zwiększają jego przejrzystość.
Ogólna postać funkcji wygląda następująco:

```
nazwa_funkcji()
{
        instrukcje do wykonania
}
```

Przykładową funkcję można zapisać w następujący sposób:

```
dodawanie()
{
WYNIK=$(($1 + $2))
echo"$WYNIK"
}
```

Utworzoną funkcję wywołuje się podając jej nazwę i jeśli zachodzi taka potrzeba parametry, z jakimi ma zostać ona uruchomiona, np.:

```
dodawanie 12 69
```

Podobnie jak w przypadku parametrów skryptu przekazywanych w linii poleceń bezpośrednio po nazwie pliku zawierającego kod skryptu, tak i w przypadku funkcji, parametry funkcji umieszczane są w tablicy $*, a odwoływanie się do poszczególnych parametrów odbywa się poprzez konstrukcję $1, $2, $3 … itd, jednakże parametry te, na czas działania funkcji, zastępują parametry skryptu przekazane w linii poleceń. Po zakończeniu działania funkcji można wykonywać działania na parametrach skryptu, ponieważ wartość tych parametrów nie uległa zmianie.
Wewnątrz funkcji można oczywiście używać wszystkich konstrukcji, które zostały omówione w niniejszym kursie.
Przykładowy skrypt zawierający funkcje wygląda następująco:

```
#!/bin/bash
dzielenie_liczb()
{
WYNIK=`expr $1 / $2`
echo "wynik dzielenia wykonanego przez funkcje wynosi: $WYNIK"
}
WYNIK=`expr $1 / $2`
echo "wynik dzielenia wykonanego poza funkcja (przed jej wykonaniem) wynosi $WYNIK"
dzielenie_liczb 10 5 #wywołanie funkcji wraz z parametrami
WYNIK=`expr $1 / $2`
echo "wynik dzielenia wykonanego poza funkcja (po jej wykonaniu) wynosi $WYNIK"
```

Przykładowe wywołanie powyższego skryptu wygląda następująco:

```
./nazwa_skryptu 20 4
```

W wyniku wykonania powyższego skryptu na ekranie monitora pojawi się następujący wynik jego działania:

wynik dzielenia wykonanego poza funkcja (przed jej wykonaniem) wynosi 5
wynik dzielenia wykonanego przez funkcje wynosi: 2
wynik dzielenia wykonanego poza funkcja (po jej wykonaniu) wynosi 5

Powyższy skrypt ma za zadanie udowodnić, że parametry skryptu nie są modyfikowane w wyniku wywołania wewnątrz skryptu funkcji wraz z jej parametrami.

14. Zakończenie

I to by było na, tyle. W kursie tym podana została podstawowa składnia pomocna w pisaniu prostych jak i tych bardziej skomplikowanych skryptów. Oczywiście przedstawiony materiał nie wyczerpuje w całości poruszonego zagadnienia, a jedynie stanowi wstęp do dalszej pracy podczas tworzenia skryptów działających w powłoce bash.