Politechnika Łódzka

Instytut Informatyki Stosowanej

ADMINISTRACJA I BEZPIECZEŃSTWO SYSTEMÓW SIECIOWYCH

Monitorowanie i zarządzanie serwerami sieciowymi

Laboratorium – ćwiczenie 3
System Linux –skrypty i funkcje

Prowadzi: dr inż. Piotr Urbanek
purbanek@iis.p.lodz.pl

Spis treści

Warunki w instrukcjach sterujących

Podstawą większości instrukcji sterujących jest sprawdzanie jakiegoś warunku. Do sprawdzania warunków służy polecenie test. (Należy zatem pamiętać, aby **nie nadawać skryptom nazwy *test*** !).

Składnia:

test wyrażenie1 ***operator*** *wyrażenie2*

Przykład:

```
# a=1
# test $a -eq 1
# echo $?
0
# a=5
# test $a -eq 1
# echo $?
1
```

Wyjaśnienie:

Polecenie test zwraca wartość 0 (true), jeśli warunek jest spełniony lub wartość 1 (false) w przeciwnym wypadku (**inaczej niż w przypadku expr!**). Wartość zwracana umieszczana jest w zmiennej specjalnej $?

Poniżej wyszczególniono kilka przykładowych operatorów polecenia test:

-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

Aby poprawić czytelność zapisu w instrukcjach sterujących dostępny jest alternatywny sposób zapisu:

[wyrażenie1 ***operator*** *wyrażenie2]*

Uwaga: W przypadku tego drugiego zapisu (z którego korzystaliśmy we wcześniejszych przykładach) należy pamiętać o pojedynczych spacjach pomiędzy nawiasami, a treścią warunku. Przykład:

Przykład:

```
# a=1
# [ $a -eq 1 ]
# echo $?
0
# a=5
# [ $a -eq 1 ]
# echo $?
1
```

Powłoka *bash* udostępnia także zapis warunku zgodny ze składnią języka C. Ma on następującą podać:

((warunek_c))

Przykład:

```
# a=1
# (((($a==1)||($a==3)))
# echo $?
0
# a=3
# (((($a==1)||($a==3)))
# echo $?
0
# a=5
# (((($a==1)||($a==3)))
# echo $?
1
```

Instrukcje sterujące

Instrukcja warunkowa if

Instrukcja if sprawdza czy warunek jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub polecenia znajdujące się po słowie kluczowym then. Instrukcja kończy się słowem fi.

Składnia:

```
if warunek
then polecenie
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
echo "Masz plik .bashrc"
fi
```

Uwaga: Wszystkie instrukcję sterujące powinny być zapisywane z zachowaniem podanego podziału na linie. Można jedynie dopisywać dodatkowe znaki końca linii w celu poprawy czytelności zapisu (co uczyniono w przykładzie po słowie 'then'). Jeśli chcemy zapisać polecenie w jednej linii, musimy końce linii zastąpić znakiem ';'. Powyższy przykład będzie wówczas miał postać:

```
if [ -e ~/.bashrc ]; then echo "Masz plik .bashrc"; fi
```

Instrukcja if umożliwia również dokonanie wyboru dwuwariantowego. Składnia takiej instrukcji przedstawiona jest poniżej:

Składnia:

```
if warunek
then polecenie1
else polecenie2
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
echo "Masz plik.bashrc"
else
echo "Nie masz pliku .bashrc"
fi
```

Możliwe jest również testowanie większej ilości warunków (wybór wielowariantowy). Służy do tego słowo kluczowe elif (ang. else if).

Składnia:

```
if warunek
then polecenie1
elif warunek
then polecenie2
else polecenie 3
fi
```

Przykład:

```
#!/bin/bash
if [ -x /opt/kde/bin/startkde ]
then echo "Masz KDE w katalogu /opt"
elif [ -x /usr/bin/startkde ]
then echo "Masz KDE w katalogu /usr"
elif [ -x /usr/local/bin/startkde ]
then echo "Masz KDE w katalogu /usr/local"
else echo "Nie wiem gdzie masz KDE"
fi
```

Instrukcja case

Instrukcja case pozwala na dokonanie wyboru wielowariantowego. Wartość zmiennej zmienna porównywana jest z poszczególnymi wzorcami. Jeśli wzorzec ma taką samą wartość jak zmienna wówczas wykonywane są polecenia przypisane do tego wzorca. Jeśli nie uda się znaleźć dopasowania wykonywane jest polecenie domyślne oznaczone symbolem * (gwiazdka). Dlatego warto zawsze to polecenie domyślne umieszczać w instrukcji case, co będzie zabezpieczeniem przed błędami popełnionymi przez użytkownika.

Składnia:

```
case zmienna in
"wzorzec1") polecenie1 ;;
"wzorzec2") polecenie2 ;;
"wzorzec3") polecenie3 ;;
*) polecenie_domyślne
esac
```

Przykład:

```
#!/bin/bash
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
```

Pętla for

Wykonuje polecenia zawarte wewnątrz pętli, na każdym składniku listy (iteracja).

Składnia:

```
for zmienna in lista
do
polecenie
done
```

Przykład:

```
for x in jeden dwa trzy
do
echo "To jest $x"
done
```

Do zmiennej x w każdym przebiegu pętli for przypisywany jest jeden z elementów listy. Następnie wykonywane jest polecenie echo "To jest $x"

Pętla for może być również przydatna, gdy chcemy wykonać jakąś operację na wszystkich plikach w danym katalogu. Poniższy skrypt obrazuje taką sytuację:

```
#!/bin/bash
for x in *html
do
echo "To jest plik $x"
done
```

Skrypt ten wypisuje wszystkie plik z rozszerzeniem html znajdujące się w bieżącym katalogu.

Powłoka 'bash' pozwala na zapis pętli 'for' z zachowaniem składni zbliżonej do języka C. Na przykład:

```
# for((a=0;a<5;a++)); do echo $a; done;
0
1
2
3
4
```

Pętla select

Pętla **select** wygeneruje z listy słów po **in** proste ponumerowane menu, gdzie każdej pozycji odpowiada kolejna liczba od 1 wzwyż. Poniżej menu znajduje się znak zachęty PS3. Umożliwia to wprowadzenie cyfry odpowiadającej wybranej przez nas pozycji w menu. W przypadku, gdy zostanie naciśnięty klawisz ENTER, menu zostanie wyświetlone ponownie. Wartość wprowadzona przez użytkownika zostanie zapisana w zmiennej REPLY. Gdy odczytane zostaje **EOF** (ang. End Of File) czyli znak końca pliku (CTRL+D) pętla **select** zakończy pracę. Pętla działa dotąd dopóki nie wykonane zostaje polecenie **break** lub **return**.

Składnia:

```
select zmienna in lista
do
polecenie
done
```

Przykład:

```
select myselection in fred wilma pebbles barney betty
do
case $myselection in
fred)
echo Fred was the selection
;;
wilma)
echo Wilma was the selection
;;
pebbles) echo Pebbles was the selection
;;
barney)
echo Barney was the selection
;;
betty)
echo Betty was the selection
;;
esac
done
```

Efekt działania tego skryptu przedstawiony jest poniżej:

```
$./select
1) fred
2) wilma
3) pebbles
4) barney
5) betty
#? 3
Pebbles was the selection
#? 5
Betty was the selection
#? 6
#? 4
Barney was the selection
# Ctrl-C
$
```

Pętla while

W pierwszym kroku sprawdzany jest warunek czy jest spełniony. Jeśli tak, wówczas wykonywane są polecenia zawarte wewnątrz pętli. Po sprawdzeniu, że warunek jest fałszywy, wykonanie pętli zostaje przerwane.

Składnia:

```
while warunek
do
polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
while [ $x -le 10 ]; do
echo "Napis pojawił się po raz: $x"
x=$[x + 1]
done
```

Na początku sprawdzany jest warunek, czy zmienna x o wartości początkowej 1 jest mniejsza lub równa 10. Warunek ten jest prawdziwy w związku z czym wykonywane są polecenia zawarte wewnątrz pętli. Przy każdym przebiegu pętli wartość zmiennej x zwiększa jest o 1. Gdy wartość x przekroczy 10, wykonanie pętli zostanie przerwane.

Pętla until

Pętla until sprawdza, czy warunek jest spełniony. Jeśli nie jest 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.

Składnia:

```
until warunek
do
polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
until [ $x -ge 10 ]; do
echo "Napis pojawił się po raz: $x"
x=$[x + 1]
done
```

Zmienna x, ma wartość 1. Sprawdzamy, czy wartość zmiennej x jest większa lub równa 10. Jeśli nie to wykonywane są polecenia zawarte wewnątrz pętli. Gdy zmienna x osiągnie wartość 10, pętla zostanie zakończona.

Modyfikowanie działania instrukcji sterujących

Break

Polecenie **break** kończy działanie pętli, w której się znajduje. Proszę przyjrzeć się poniższemu skyptowi:

```
#!/bin/bash
for imie in Tomek Jacek Henio
do
  echo $imie
  echo "znowu"
done
echo "wszystko zrobione"
```

Po wykonaniu powyższego skryptu uzyskamy:

```
Tomek
znowu
Jacek
znowu
Henio
znowu
wszystko zrobione
```

A teraz wersja z wykorzystaniem polecenia break:

```
#!/bin/bash
for imie in Tomek Jacek Henio
do
  echo $imie
  if [ "$imie" = "Jacek" ]
  then
     break
  fi
  echo "znowu"
done
echo "wszystko zrobione"
```

Po wykonaniu powyższego skryptu uzyskamy:

```
Tomek
znowu
Jacek
wszystko zrobione
```

Continue

Polecenie **continue** wymusza w pętli przejście do kolejnej iteracji. Przykład:

```
for imie in Tomek Jacek Henio
do
  echo $imie
  if [ "$imie" = "Jacek" ]
  then
     continue
  fi
  echo "znowu"
  done
echo "wszystko zrobione"
```

Po wykonaniu powyższego skryptu uzyskamy:

```
Tomek
znowu
Jacek
Henio
znowu
wszystko zrobione
```

Polecenia **break** i **continue** przyjmują również argument liczbowy (**break N**, **continue N**), pozwalający określać liczbę warstw pętli, przez które mają "przebić" się te instrukcje (na przykład wyjście jednocześnie z N warstw pętli). Trzeba jednak pamiętać, że tego rodzaju operacje prowadzą do powstania nieczytelnego kodu, dlatego nie zalecane jest ich użycie.

Funkcje

Funkcje są to podprogramy, dzięki czemu często wykorzystywane instrukcje można zgrupować, zamiast pisać je po kilka razy. Aby odwołać się do funkcji należy tylko podać jej nazwę. Ogólna składnia funkcji ma postać:

```
function nazwa_funkcji
{
polecenie1
polecenie2
polecenie3
}
```

Na przykład:

```
#!/bin/bash
function napis
{
echo "Przykładowy napis"
}
napis
```

Możliwe jest również umieszczanie funkcji w osobnym pliku. Dzięki temu nasz skrypt będzie bardziej przejrzysty. Jak z tej możliwości skorzystać? Należy utworzyć plik nagłówkowe do których odwołujemy się zgodnie z poniższą składnią:

```
. ~/plik_z_funkcjami
nazwa_funkcji
```

Obowiązkowo należy pamiętać o podaniu kropki i spacji przed nazwą pliku nagłówkowego. Zdefiniujmy funkcję napis w pliku nagłówkowym funkcje.

```
#!/bin/bash
function napis
{
echo -e 'Wywołałeś funkcję z pliku "funkcje".\a'
}
```

Aby z tej funkcji skorzystać, należy utworzyć skrypt i umieścić wewnątrz odpowiednie wywołanie funkcji. Zaimplementujemy taki skrypt:

```
#!/bin/bash
echo "Testowanie pliku nagłówkowego"
. funkcje
napis
```

Zadania

1. Podaj ogólną składnie polecenia ‘case’.

2. Co to jest znak EOF?

3. Jaki operator oznaczony jest za pomocą -ge ?

4. Czy możliwe jest umieszczenie funkcji w oddzielnym pliku ?

5. Podaj ogólną składnię polecenia ‘for’.

6. Napisz skrypt wyświetlający bieżący czas. Skrypt powinien działać do momentu przerwania przy pomocy Ctrl-C

7. Napisz skrypt wyświetlający wszystkie katalogi w bieżącym katalogu.

8. Korzystając z wcześniejszego skryptu napisz skrypt wyświetlający katalogi w bieżącym katalogu i jego podkatalogach. (Należy przerobić wcześniejszy skrypt na procedurę, która będzie rekursywnie wywoływała samą siebie).

9. Napisz skrypt wykonujący cztery podstawowe działania arytmetyczne. Skrypt ma pytać jakie działanie użytkownik chce wykonać a następnie o operandy tego działania. Skrypt powinien działać w nieskończonej pętli (szczegóły dotyczące działań arytmetycznych można odnaleźć w manualu funkcji expr lub bash).

10. Napisz skrypt, który będzie sprawdzał czy w wywołaniu wystąpiły 3 argumenty. Jeśli nie to ma prosić o ich podanie.

11. Napisz skrypt, który sprawdzi aktualny czas systemowy, a następnie w zależności od wartości godziny wypisze odpowiedni komunikat:

- <00-06 – „Dobranoc”
- <06-12 – „Milego dnia”
- <12-18 – „Dzien dobry”
- <18-24 – „Dobranoc”

12. Napisz skrypt obliczający silnię podanej liczby.

13. Napisz skrypt wyświetlający dwójkową reprezentację liczby podanej w systemie dziesiętnym.