Temat: Automatyzacja pracy - skrypty - pętle w Linux.

Cel ogólny lekcji: Przygotowanie uczniów do efektywnego wykorzystania mechanizmów automatyzacji pracy w systemie operacyjnym Linux poprzez skrypty i pętle, umożliwiające wykonywanie powtarzalnych zadań oraz elastyczne zarządzanie różnymi parametrami w skryptach.

Cele szczegółowe:

1. Zrozumienie istoty automatyzacji pracy oraz zdolność do identyfikowania sytuacji, w których zastosowanie skryptów może przynieść korzyści w systemie Linux.
2. Zapoznanie z różnymi typami pętli w języku Bash, w tym pętlą for, while i until, w celu automatyzacji iteracyjnych procesów.
3. Opanowanie składni pętli for, while i until, w tym zrozumienie warunków i logiki sterującej ich działaniem.
4. Rozwijanie umiejętności tworzenia, testowania i modyfikowania skryptów w Bashu z wykorzystaniem różnych rodzajów pętli.
5. Zrozumienie funkcji instrukcji warunkowych takich jak if, break i continue w kontekście pętli while, umożliwiających bardziej zaawansowaną kontrolę nad przebiegiem skryptów.
6. Nauka wykorzystywania skryptów w Bashu do manipulowania, porównywania oraz analizy różnych parametrów w systemie operacyjnym Linux.
7. Rozwijanie umiejętności analitycznego myślenia poprzez analizę wyników działania skryptów oraz przewidywanie efektów ich działania.
8. Zrozumienie znaczenia funkcji w skryptach Bash oraz zdolność do ich tworzenia, aby zwiększyć czytelność, modularyzację kodu i efektywność działań.
9. Zapoznanie się z podstawowymi operatorami arytmetycznymi w Bash, takimi jak dodawanie, odejmowanie, mnożenie, dzielenie i reszta z dzielenia.
10. Opanowanie tworzenia i deklarowania funkcji w Bashu, w tym nauka składni oraz struktury ich definicji.
11. Umiejętność przekazywania argumentów do funkcji i efektywnego ich wykorzystywania wewnątrz funkcji do wykonywania operacji.
12. Praktyczne wykorzystanie funkcji w celu skrócenia kodu, zwiększenia czytelności i umożliwienia wielokrotnego użycia określonej logiki.
13. Poznanie technik obsługi parametrów linii poleceń i ich rola w skryptach oraz funkcjach.
14. Opanowanie tworzenia skryptów, które wykorzystują operacje arytmetyczne do przeprowadzania obliczeń na liczbach.

15. Zdobycie praktycznych umiejętności poprzez tworzenie skryptów, które integrują funkcje i operacje arytmetyczne, aby skutecznie rozwiązywać konkretne zadania.
16. Zrozumienie różnicy między zmiennymi lokalnymi a globalnymi w kontekście funkcji oraz ich wpływu na przekazywanie danych.
17. Rozwinięcie zdolności do optymalizacji kodu, poprzez wykorzystanie funkcji do automatyzacji powtarzalnych działań oraz operacji arytmetycznych do złożonych obliczeń.

Dzięki osiągnięciu tych celów, uczniowie będą zdolni tworzyć, analizować i modyfikować skrypty w języku Bash, co pozwoli im na efektywną automatyzację zadań w środowisku Linux oraz zwiększy ich umiejętności programistyczne i myślenie analityczne. Uczestnictwo w lekcji pozwoli uczniom zdobyć praktyczną wiedzę z zakresu automatyzacji pracy oraz programowania w systemie operacyjnym Linux.

Wykonaj i przetestuj działanie wszystkich skryptów poniżej. W zszycie zapisz nowe informacje.

1. Pętle.

Pętle stosuje się w celu wykonania pewnych instrukcji dla kolejnych iteracji lub kilku parametrów. W bashu przy tworzeniu skryptów możemy się posługiwać pętlami. Do najczęściej stosowanych pętli zalicza się pętle for i while.

a) FOR

Składnia pętli for wygląda następująco:

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

1. 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 skryptu:

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

Opis działania tego skryptu:

Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.

W pętli for, wyrażenie $* odnosi się do wszystkich argumentów przekazanych do skryptu podczas jego uruchamiania.

Pętla for przechodzi przez każdą wartość w $* (czyli każdy argument podany przy uruchamianiu skryptu) i przypisuje ją do zmiennej ZMIENNA.

Wewnątrz pętli, echo "$ZMIENNA" wyświetla wartość zmiennej ZMIENNA.

Pętla for kontynuuje przetwarzanie kolejnych argumentów, wykonując te same instrukcje dla każdej z nich.

Po zakończeniu pętli, skrypt się zamyka.

Podsumowując, ten skrypt wyświetla każdy argument przekazany przy uruchamianiu skryptu w nowej linii.

Utwórz plik skryptu o nazwie skrypt_a1, nadaj mu atrybut wykonywalności.

Nadaj prawa wykonywalności skryptowi (chmod +x nazwa_skryptu.sh)

chmod +x skrypt_a1

Uruchomisz skrypt w następujący sposób:

./skrypt_a1 pierwszy_drugi trzeci

Skrypt wyświetli:

```
“pierwszy_drugi”
“trzeci”
```

Skrypt wykorzystuje pętlę for do wyświetlania kolejnych wartości przekazywanych jako argumenty podczas uruchamiania skryptu.

2. Wykonaj 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
```

Oto opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.

2. Skrypt tworzy trzy zmienne DZIEN1, DZIEN2 i DZIEN3, z którymi są powiązane nazwy dni tygodnia.

3. W pętli for, wyrażenie ”$DZIEN1 ”$DZIEN2 ”$DZIEN3 tworzy jedną ciągłą sekwencję z trzech dni tygodnia, oddzielając je spacją.

4. Pętla for przechodzi przez każdą wartość w sekwencji i przypisuje ją do zmiennej ZMIENNA.
5. Wewnątrz pętli, echo ”Dzisiaj jest: $ZMIENNA” wyświetla komunikat z odpowiednim dniem tygodnia.
6. Pętla for kontynuuje przetwarzanie kolejnych dni, wykonując te same instrukcje dla każdego z nich.
7. Po zakończeniu pętli, skrypt się zamyka.

W powyższym skrypcie, 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.

Podsumowując, ten skrypt wyświetla komunikat z nazwą każdego dnia tygodnia po kolei.

Utwórz folder cw1516, a w nim utwórz plik skryptu o nazwie skrypt_a2, nadaj mu atrybut wykonywalności.

Nadaj prawa wykonywalności skryptowi (chmod +x nazwa_skryptu.sh)

```
chmod +x skrypt_a2
./skrypt_a2
```

Po uruchomieniu skryptu, powinien on wyświetlić:

Dzisiaj jest: poniedziałek

Dzisiaj jest: wtorek

Dzisiaj jest: środa

To jest przykład wykorzystania pętli for do iteracji przez listę zmiennych i wykonania działań związanych z każdą zmienną.

b) 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

Skrypt wykorzystujący pętlę while:

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

```
if [ $1 -gt $2 ]; then
   X=$1;
   while [ $X -gt $2 ]; do
      ROZNICA=$(($X - $2));
      echo "liczba $X jest większa od liczby $2 o $ROZNICA";
      X=$(($X - 1));
   done
else
   echo "liczba $1 nie jest większa od liczby $2";
fi
```

Opis działania skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt sprawdza, czy pierwszy argument $1 jest większy od drugiego argumentu $2 za pomocą warunku [$1 -gt $2].
3. Jeśli warunek jest spełniony (czyli jeśli pierwsza liczba jest większa od drugiej):
   a. Skrypt przypisuje wartość $1 do zmiennej X.
   b. Następnie skrypt wchodzi w pętlę while z warunkiem [$X -gt $2].
   c. Wewnątrz pętli:
      1) Oblicza różnicę między $X a $2 i przypisuje ją do zmiennej ROZNICA.
      2) Wypisuje komunikat o różnicy między liczbami.
      3) Zmniejsza X o 1.
   d. Pętla działa, dopóki X jest większe od $2.
4. Jeśli warunek początkowy nie jest spełniony (czyli pierwsza liczba nie jest większa od drugiej), skrypt wypisuje komunikat o tym fakcie.
5. Skrypt kończy się.

Utwórz plik skryptu o nazwie skrypt_b, nadaj mu atrybut wykonywalności.

Nadaj prawa wykonywalności skryptowi (chmod +x nazwa_skryptu.sh)

```
chmod +x skrypt_b
```

Uruchomisz skrypt w następujący sposób:

```
./skrypt_b 10 5
```

Skrypt wyświetli:

liczba 10 jest większa od liczby 5 o 5

liczba 9 jest większa od liczby 5 o 4

liczba 8 jest większa od liczby 5 o 3

liczba 7 jest większa od liczby 5 o 2

liczba 6 jest większa od liczby 5 o 1

Podsumowując, ten skrypt porównuje dwie liczby podane jako argumenty i jeśli pierwsza liczba jest większa od drugiej, wypisuje różnicę między nimi w ramach pętli while. W przeciwnym przypadku, skrypt informuje, że pierwsza liczba nie jest większa od drugiej.

c) 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
```

Skrypt wykorzystujący pętlę while:

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

Oto opis działania tego skryptu:

1) Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2) Skrypt przypisuje wartość pierwszego argumentu $1 do zmiennej X.
3) W pętli until, warunek [$X -gt $2] sprawdza, czy X jest większe od drugiego argumentu $2.
4) Pętla until działa, dopóki warunek [$X -gt $2] nie jest spełniony (czyli dopóki X jest mniejsze lub równe $2).
5) Wewnątrz pętli:
    a. Skrypt wypisuje komunikat "liczba [wartość_X] jest mniejsza od liczby [wartość_2]".
    b. Zmienna X jest zwiększana o 1 za pomocą X=$(($X + 1)).

6) Pętla until działa, aż X będzie większe od $2.
7) Po zakończeniu pętli, skrypt się zamyka.

Utwórz plik skryptu o nazwie skrypt_c, nadaj mu atrybut wykonywalności.

Aby uruchomić skrypt, postępuj podobnie jak wcześniej, nadając prawa wykonywalności skryptowi (chmod +x nazwa_skryptu.sh) i uruchamiając go (./nazwa_skryptu.sh 7 10), gdzie 7 to pierwsza liczba, a 10 to druga liczba. Skrypt wyświetli komunikaty w zależności od wyniku pętli.

Uruchomisz skrypt w następujący sposób:

```
chmod +x skrypt_c
./skrypt_c 7 10
```

Skrypt wyświetli:

"liczba 7 jest mniejsza od liczby 10"
"liczba 8 jest mniejsza od liczby 10"
"liczba 9 jest mniejsza od liczby 10"
"liczba 10 jest mniejsza od liczby 10"

Podsumowując, ten skrypt wykonuje pętlę until, która wypisuje komunikat o tym, że pierwsza liczba jest mniejsza od drugiej liczby. Pętla będzie działać, dopóki wartość pierwszej liczby X nie osiągnie wartości większej od drugiej liczby $2. 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.

Zgłoszenie1

2. Użycie znaków globalnych w pętlach

a) 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 '*'

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.

Opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt wykorzystuje polecenie echo do wypisania komunikatu: "Oto zawartość katalogu [ścieżka_bieżącego_katalogu]:".
3. Skrypt używa zmiennej ${PWD} do uzyskania pełnej ścieżki bieżącego katalogu.
4. Następnie skrypt używa /* (gwiazdka) aby zwrócić zawartość katalogu, a więc wszystkie pliki i foldery w bieżącym katalogu.
5. Wszystkie elementy katalogu są wyświetlane za pomocą drugiego polecenia echo ${PWD}/*.
6. Skrypt kończy się.

Podsumowując, ten skrypt wyświetla komunikat z pełną ścieżką bieżącego katalogu, a następnie wyświetla wszystkie pliki i foldery znajdujące się w tym katalogu.

Utwórz plik skryptu o nazwie skrypt2a nadaj mu atrybut wykonywalności.

Aby uruchomić skrypt, otwórz terminal, przejdź do katalogu, w którym znajduje się ten skrypt, i wykonaj komendę:

```
chmod +x skrypt2a
./skrypt2a
```

Skrypt wyświetli zawartość bieżącego katalogu w postaci listy plików i folderów.

b) 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 s:
echo ${PWD}/s*
```

Skrypt bashowy również służy do wyświetlania wykazu plików z bieżącego katalogu, których nazwa zaczyna się od litery "s". Oto opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt używa polecenia echo do wypisania komunikatu: "Oto wykaz wszystkich plików z katalogu [ścieżka_bieżącego_katalogu], których nazwa zaczyna się na literę s:".
3. Skrypt używa zmiennej ${PWD} do uzyskania pełnej ścieżki bieżącego katalogu.
4. Następnie skrypt używa /${PWD}/s* (ścieżka/bieżący_katalog/s*), aby zwrócić wykaz plików, których nazwa zaczyna się od litery "s".
5. Wszystkie pliki, których nazwa zaczyna się od litery "s", są wyświetlane za pomocą drugiego polecenia echo ${PWD}/s*.

6. Skrypt kończy się.

Podsumowując, ten skrypt wyświetla komunikat z pełną ścieżką bieżącego katalogu oraz wykaz plików, których nazwa zaczyna się od litery "s".

Utwórz plik skryptu o nazwie skrypt2b, nadaj mu atrybut wykonywalności.

Aby uruchomić skrypt, otwórz terminal, przejdź do katalogu, w którym znajduje się ten skrypt, i wykonaj komendę:

```
chmod +x skrypt2b
./skrypt2b
```

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

Zgłoszenie2

3. 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: $(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.

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”
```

Skrypt wykonuje dwie operacje - wykorzystuje komendy systemowe ls -l i cat do przypisania zawartości do zmiennej DANE, a następnie wyświetla tę zawartość przy użyciu różnych metod rozwijania zmiennych. Opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt używa komendy ls -l do uzyskania listy plików i folderów w bieżącym katalogu wraz z ich szczegółowymi informacjami (-l). Wynik komendy jest przypisywany do zmiennej DANE za pomocą DANE=”$(ls -l)”.
3. Skrypt używa polecenia echo do wypisania komunikatu: "Wynik metody rozwijania zawartości nawiasów:".
4. Skrypt wyświetla zawartość zmiennej DANE przy użyciu echo ”$DANE”.
5. Skrypt używa komendy cat do odczytania zawartości pliku o nazwie test.txt. Wynik komendy jest przypisywany do zmiennej DANE za pomocą DANE=cat test.txt``.
6. Skrypt używa polecenia echo do wypisania komunikatu: "Wynik metody ujmowania polecenia w tzw. wsteczne apostrofy:".
7. Skrypt ponownie wyświetla zawartość zmiennej DANE przy użyciu echo ”$DANE”.
8. Skrypt kończy się.

Podsumowując, ten skrypt wykonuje dwie operacje:

Wyświetla wynik komendy ls -l przy użyciu zmiennych nawiasów (ang. command substitution) i wyświetla go jako tekst.

Odczytuje zawartość pliku test.txt przy użyciu polecenia cat i również wyświetla tę zawartość jako tekst.

Utwórz plik skryptu o nazwie skrypt3, nadaj mu atrybut wykonywalności.

chmod +x skrypt3

Aby uruchomić skrypt, otwórz terminal, przejdź do katalogu, w którym znajduje się ten skrypt, i wykonaj komendę:

./skrypt3

Skrypt wyświetli wyniki obu operacji w terminalu.

zgłoszenie3

4. Operacje arytmetyczne

Wykonywanie 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

Srypt 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"
```

Skrypt wykonuje operacje arytmetyczne na dwóch zmiennych przekazanych jako argumenty przy uruchomieniu. Oto opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt przypisuje wartość pierwszego argumentu $1 do zmiennej A i wartość drugiego argumentu $2 do zmiennej B.
3. Skrypt używa polecenia echo -n do wypisania komunikatu bez przechodzenia do nowej linii: "Wynik dodawania liczby [wartość_A] do liczby [wartość_B]:".

4. Skrypt wykorzystuje polecenie expr do wykonania dodawania wartości A i B, wynik jest przypisywany do zmiennej WYNIK.
5. Skrypt używa echo $WYNIK do wyświetlenia wyniku dodawania.
6. Skrypt powtarza kroki 3-5 dla kolejnych operacji arytmetycznych: odejmowania, mnożenia, dzielenia i reszty z dzielenia.
7. Skrypt kończy się.

Podsumowując, ten skrypt wykonuje różne operacje arytmetyczne (dodawanie, odejmowanie, mnożenie, dzielenie i reszta z dzielenia) na dwóch zmiennych przekazanych jako argumenty podczas uruchomienia skryptu.

Utwórz plik skryptu o nazwie skrypt4, nadaj mu atrybut wykonywalności.

chmod +x skrypt4

Aby uruchomić skrypt, otwórz terminal i wykonaj komendę:

./skrypt4 liczba1 liczba2

Zamień liczba1 i liczba2 na rzeczywiste liczby, na których chcesz wykonać operacje arytmetyczne.

./skrypt4 5 6

”Wynik dodawania liczby 5 do liczby 6: ”11”

”Wynik odejmowania liczby 5 od liczby 6: ”1”

”Wynik mnozenia liczby 5 przez liczbe 6: ”30”

”Wynik dzielenia liczby 5 przez liczbe 6: ”0”

”Reszta z dzielenia liczby 5 przez liczbe 6: ”5”

Skrypt wyświetla wyniki poszczególnych operacji.

Zgłoszenie4

5. Funkcje

Można tworzyć funkcje, które w bardzo dużej mierze przyczyniają się do skrócenia kodu skryptu i zwiększają jego przejrzystość:

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

Przykładowa funkcja:

```
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.

a) Przykładowy skrypt zawierający funkcje:

```
#!/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 $1 $2 #wywołanie funkcji wraz z parametrami
WYNIK=`expr $1 / $2`
echo ”wynik dzielenia wykonanego poza funkcja po jej wykonaniu wynosi $WYNIK”
```

Skrypt bashowy definiuje funkcję dzielenie_liczb, a następnie dwukrotnie wykonuje dzielenie tych samych liczb poza funkcją oraz z użyciem funkcji. Oto opis działania tego skryptu:

1. Skrypt zaczyna się od deklaracji #!/bin/bash, która wskazuje, że skrypt będzie interpretowany przy użyciu powłoki Bash.
2. Skrypt definiuje funkcję dzielenie_liczb, która przyjmuje dwie liczby jako argumenty.
   a. Wewnątrz funkcji skrypt używa polecenia expr do wykonania dzielenia wartości pierwszego argumentu przez wartość drugiego argumentu. Wynik jest przypisywany do zmiennej WYNIK.

b. Skrypt wykorzystuje polecenie echo do wypisania komunikatu z wynikiem dzielenia wykonanego w funkcji.

3. Skrypt poza funkcją również wykonuje dzielenie tych samych liczb, używając polecenia expr. Wynik jest przypisywany do zmiennej WYNIK.
4. Skrypt używa polecenia echo do wypisania komunikatu z wynikiem dzielenia wykonanego poza funkcją przed wykonaniem funkcji.
5. Skrypt wywołuje funkcję dzielenie_liczb, przekazując jej argumenty ($1 i $2).
6. Skrypt znowu wykonuje dzielenie tych samych liczb poza funkcją, używając polecenia expr. Wynik jest przypisywany do zmiennej WYNIK.
7. Skrypt używa polecenia echo do wypisania komunikatu z wynikiem dzielenia wykonanego po wykonaniu funkcji.
8. Skrypt kończy się.

Podsumowując, skrypt definiuje funkcję do dzielenia liczb i dwukrotnie wykonuje to dzielenie: raz poza funkcją i raz z użyciem funkcji.

Utwórz plik skryptu o nazwie skrypt5, nadaj mu atrybut wykonywalności.

chmod +x skrypt5

Wywołanie powyższego skryptu wygląda następująco:

./skrypt5 liczba1 liczba2

Zamień liczba1 i liczba2 na rzeczywiste liczby, na których chcesz wykonać dzielenie. Skrypt wyświetli wyniki obu dzieleń w terminalu.

./skrypt5 20 4

W wyniku wykonania 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: 5

wynik dzielenia wykonanego poza funkcja po jej wykonaniu wynosi 5

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

Warto zaznaczyć, że wynik dzielenia jest zaokrąglany w dół do najbliższej całkowitej liczby mniejszej lub równej wynikowi rzeczywistego dzielenia.

Zgłoszenie5

Przedstawiony materiał nie wyczerpuje w całości poruszonego zagadnienia, a jedynie stanowi wstęp do dalszej pracy podczas tworzenia skryptów działających w powłoce bash.

Podsumowanie:

Po wykonaniu wszystkich czynności z powyższej instrukcji przeczytaj ponownie z zrozumieniem cel ogólny i cele szczegółowe, które znajdują się na pierwszej stronie instrukcji. Jeżeli one zostały niezrealizowane to powtarzaj wykonie tej instrukcji w szkole lub/i w domu do momentu zrealizowania.