Narzędzia dla programistów
Sprawozdanie
Skrypty powłoki

Grupa №11

Ćwiczenie 3

Skrypty powłoki

1 Tworzenie skryptów

1. Wszystkie polecenia oraz edyckja plikow będą wykonane za pomocą terminala i edytora *Vim(VI Improved)*.

   (a) Poleceniem `mkdir` stworzyłem katalog o nazwie ***cw3***. Polecenie `mkdir` przyjmuje jako argument/y nazwe/y katalogu/katalogów.

   ```
   $ mkdir cw3
   ```

   (b) Uzywając polecenie `touch` w katalogu ***cw3*** został stworzony plik o nazwie ***pkt1***.

   ```
   $ touch ./cw3/pkt1
   ```

2. Po wejsciu uzywając polecenia `vim ./cw3/pkt1` zapisałem pierwszą linie ***#!/bin/bash***, ktora jest instrukcją dla systemu uniksowego, aby uruchomić interpreter którego sciezka i nazwa jest podana po znaku `!`.

   ```
   $ vim ./cw3/pkt1
   ```

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

3. Dopisałem drugą linie skryptu: `echo "Test polecenia skryptu"` w celu sprawdzania czy utworzony skrypt zostanie uruchomiony. Zapisałem plik poleceniem `:w` i wyszedłem uzywając polecenia `:q`.

   ```
   1 #!/bin/bash
   2 echo "Test polecenia skryptu"
   ```

4. Zmieniłem uprawnienia skryptu: `chmod 744 ./cw3/pkt1`, zeby móc uruchomić skrypt.

   ```
   $ chmod 744 ./cw3/pkt1
   ```

   Jeśli użyć polecenia `file` i dodać jako parametr nazwe pliku. Otrzymujemy cos takiego:

   ```
   pkt1: Bourne-Again shell script, ASCII text executable
   ```

   Ostatnie słowo oznacza, że możemy wykonać ten plik. Po sprawdzaniu, mozemy uruchomic plik uzywając polecenia `./cw3/pkt1` lub `bash pkt1`(jeśli jesteście w katalogu który zawiera plik pkt1).

   ```
   $ ./cw3/pkt1
   ```

   I mamy następujący wynik

   ```
   $ ./cw3/pkt1
   Test polecenia skryptu
   $
   ```

2 Polecenia warunkowe

1. Utworzyłem nowy scrypt o nazwie ***pkt2*** w katalogu ***cw3***. Wpisałem mu pierwszą linie interpretera(`#!/bin/bash`) i zmieniłem uprawnienia jak wcześniej(`chmod 744 pkt2`). Uzyłem polecenie `echo` oraz zdanie, które za pomocą strzałki przeadresuje do pliku ***pkt2***.

```
$ echo "#!/bin/bash" > pkt2
$ chmod 744 pkt2
```

2. W skrypcie uzyłem polecenie `test` z parametrem `f` żeby sprawdzic, czy istnieje plik *pkt1*.
Opis dzialania skryptu: przy uruchomianiu skrypt przyjmuje absolutną sciezkę (która zawiera sie w zmiennej `LOCAL`) do katalogu który zawiera plik o nazwie *pkt1*. W przypadku nieprawidłowej ścieżki lub braku pliku skrypt się skompiluje i nic nie zwróci, inaczej wyswietli się kominukat "*Plik pkt1 istnieje*".

```
1 #!/bin/bash
2 LOCAL=$1
3 if [ -f $LOCAL/pkt1 ]; then
4         echo "Plik pkt1 istnieje"
5 fi
```

3. Do skyptu *pkt2* dopisałem linie, która sprawdza czy plik *dane1* istnieje, a jeżeli nie istnieje to utworzy plik o takiej nazwie z zawartoscią "*Linia tekstu w danych1*".

```
1  #!/bin/bash
2  LOCAL=$1
3  if [ -f $LOCAL/pkt1 ]; then
4          echo "Plik pkt1 istnieje"
5  fi
6  if [ -f $LOCAL/dane1 ]; then
7          echo "Plik dane1 istnieje";
8  else
9          echo "Linia tekstu w danych1" > $LOCAL/dane1
10         echo "Plik dane1 nie istnial. Plik zostal stworzony"
11 fi
```

Opis dzialania calego skryptu: jak wcześniej skrypt przyjmuje jeden argument(ścizkę do katalogu w którym skrypt będzie sprawdzać czy istnieją pliki).

   (a) Pierwsze sprawdzenie: skrypt sprawdza czy istnieje plik *pkt1* w katalogu ścieżke do ktorego skrypt przyjmowal. Zmienna `LOCAL` zawiera ścieżke. Żeby uzyć wartość, która jest w tej zmiennej używamy symbola `$` i `nazwa_zmiennej`. Jeśli plik *pkt1* istnieje w tym miejscu wyswietli się komunikat "*Plik pkt1 istnieje*". W przeciwnym przypadku nic się nie wyswietli.

   (b) Drugie sprawdzenie: wszystko działo tak samo, tylko w przypadku braka pliku, skrypt stworzy plik o nazwie *dane1* z zawartoscią "*Linia tekstu w danych1*" oraz wyswietli się komunikat "*Plik dane1 nie istnial. Plik zostal stworzony*"

Używane funkcje i polecenia: `if, test, then, echo, fi`.

3 Zmienne

1. Uworzyłem skrypt *pkt3*, który wczytuje dwie liczby wprowadzone z klawiatury, następnie wyswietla inforamcje, która z nich jest wieksza.

```
$ echo "#!/bin/bash" > pkt3
$ chmod 744 pkt2
```

Skrypt końcowy

```
#!/bin/bash

(if [ $# -eq 1 ]; then
        echo "Please enter the second argument"
        exit 1
elif [ $# -eq 2 ]; then
        exit 0
else
    echo "Enter two arguments"
    exit 1
fi) \
&& \
(echo -e "Arguments were entered successfully\nOutput:"
if [ $1 -eq $2 ]; then
        echo "num1=num2"
elif [ $1 -gt $2 ]; then
        echo "num1>num2"
else
        echo "num1<num2"
fi)
```

Opis dzialania:

(a) Caly skrypt jest podzielony na dwie czesci(dwa osobne skrypty). Podzielone za pomocą nawiasów i połoczone podwójnym symbolem `&`, który oznacza że druga cześć będzie uruchomiona wtedy i tylko wtedy kiedy pierwsza cześć zwróci wejsciowy kod `zero`(prawde).

(b) Początkowa linia: ścieżka do interpretera i jego nazwa `#!/bin/bash`.

(c) Pierwsza czesc skryptu: skrypt(za pomocą parametra: `-eq` i zmiennej $#) sprawdza ile argumentów wpisał użytkownik(przy uruchomianiu skrypta). `-eq`(z angielskiego equal, co oznacza równość). $# oznacza ilosci podanych argumentów.

Jesli ilosc argumentów jest równa 1 skrypt wypisze wiadomość "*Please enter the second argument*", co oznacza że tylko jedna liczba została wpisana przez użytkownika.

Jesli ilosc argumentów jest równa 2 skrypt nie wypisze wiadomości tylko zwróci kod wyjsciowy `0`.

Jesli ilosc argumentów nie jest równa `1` i `2` czyli kiedy użytkownik wpisał więcej niż 2 liczby lub nie wpisał żadnej będzie zwrócony kod wyjsciowy `1`.

Kod wyjsciowy `zero` oznacza, że ten program wykonał się dobrze i skrypt może iść dalej.

2. Jeśli pierwsza czesc wykonała sie dobrze zacznie dzialanie druga czesc skryptu.

(a) Po uruchomianiu drugiej cześci wyswietli się wiadomość *"Arguments were entered successfully"*. Za pomocą parametrów: `-eq`, `-gt` skrypt sprawdza czy `pierwsza liczba` jest równa `drugiej`, lub wieksza od drugiej. `-gt` oznacza(greater than).

Jeśli pierwsza liczba jest równa drugiej wyświetli się komunikat `num1=num2`.

Jeśli pierwsza liczba jest większa od drugiej wyświetli komunikat `num1>num2`.

We wszystkich innych przypadkach będzie wyświetlony komunikat `num1<num2`.

4 Polecenia iteracyjne

1. (a) Utworzyłem skrypt *pkt4*. W skrypcie wykorzystałem pętle `for`.

```
#!/bin/bash
echo -e "\n\t\tWitam w programie!\n\tOpis: Wpisz katalog\
(absolute path)\n\t\
typy plikow ktorego chcesz wiedzec.\n"

read -p "Absolute path: " location

dir=0
fil=0
slin=0
blck=0
chrk=0
sckt=0

for elem in $(ls $location)
do
    if [ -d $location/$elem ];then
        echo -e "$elem: directory"
        dir=$[$dir+1]
    elif [ -h $location/$elem ];then
        echo -e "$elem: symbolic link"
        slin=$[$slin+1]
    elif [ -b $location/$elem ];then
        echo -e "$elem: block-special file"
        blck=$[$blck+1]
    elif [ -c $location/$elem ];then
        echo -e "$elem: character-special file"
        chrk=$[$chrk+1]
    elif [ -S $location/$elen ];then
        echo -e "$elem: socket"
        sckt=$[$sckt+1]
    elif [ -f $location/$elem ];then
        echo -e "$elem: file"
        fil=$[$fil+1]
    else
        echo -e "$elem: noname type"
    fi
done

echo "directory count: $dir"
echo "file count: $fil"
echo "symbolic link count: $slin"
echo "block-special file count: $blck"
echo "character-special file count: $chrk"
echo "socket count: $sckt"
```

(b) Opis dzialania skryptu: użytkownik wpisuje(za pomocą polecenia `read` z parametrem `-p`, który pozwala wpisać wynik wczytania do zmiennej `location`) absolutną ścieżkę do katalogu w którym chce sprawdzić czy nazwy są nazwami plików, urządzeń, katalogów, linków. Sprawdzanie wykonane za pomocą polecenia `test` z parametrami `-e(katalog), -h(link symboliczny) -b(plik blokowy), -c(plik znakowy), -e(socket), -f(plik)`.

(c) Skrypt sprawdza i zapisuje ilość(np. katalogów) w zmienną `dir`. Skrypt konczy się wyświetleniem ilości (katalogów, plików i itp.)

Używane funkcje i polecenia: `if, then, elif, else, echo, for, read`.

2. (a) Utworzyłem skrypt `pkt5` z pętla `while`, który wczyta tekst do wyświetlenia oraz liczbe, a następnie wyświetlia ten tekst zadaną liczbe razy.

```
#!/bin/bash

echo -e "\n\t\tWitam w programie!\n\n\tOpis: Wpisz wiersz i liczbe\
\n\tile razy chcesz go wyswietlic\n"
echo -e "\tOption:\n\tJesli chcesz przerwac petle\n\twpisz 0 jako\
liczbe powtorow\n"
num=1

while [ $num -ne 0 ]
do
        read -p "Wiersz: " line;
        read -p "Liczba: " n;
    if [ $n -eq 0 ]; then
        num=$[$num-1]
    fi
        for (( count=0; count<n; count++ ))
        do
                echo "$[$count+1]: "$line
        done
done
```

(b) Opis dzialania skryptu: w skrypcie stworzona zmienna `num` za pomoca której działa pętla `while`. W przypadku jeśli użytkownik wpisuje 0 jako liczbe powtorów skrypt się skonczy.
Uzywane funkcje i polecenia: `echo, while, read, if, then, for`.

3. (a) Utworzyłem skrypt `pkt6` z instrukcją `case` i pętla `while`, który w zależnosci od wpowadzonego przez uzytkownika tekstu podejmie naspępującą czynność:

- Jeśli użytkownik podaje tekst `a`, poleceniem `cat /dev/null > daty.log` stworza się pusty plik o nazwie `daty.log`
- Jeśli użytkownik podaje tekst `b`, poleceniem `date` bieżacą date i godzina dołacza się do pliku `daty.log`.
- Jeśli użytkownik podaje tekst `c`, wyświetla się na ekranie zawartość pliku `daty.log`.
- Jeśli użytkownik podaje tekst `x`, to skrypt zakończy działanie.
- W pozostałych przypadkach wartość traktowana jest jak liczba, która mówi ile razy wypisać na ekran biezacą date i godzine.

```
#!/bin/bash

echo -e "\n\t\tWitam w programie!\n\tOpis:
\ta) cat /dev/null > daty.log
\tb) echo date >> daty.log
\tc) cat daty.log
\tx) break
\t[0-9]) echo 'Wpisana liczba'"


while [ 1 -ne 0 ]
do
        read -p "Wybierz opcje: " line;
        case $line in
                a) echo -e "\n\tWybrales\las opcje 'a'. Polecenie
        'cat' zapisalo zawartosc pliku '/dev/null' do
        pliku daty.log w biezacym katalogu.\n"  && cat \
        /dev/null > daty.log;;
                b) echo -e "\n\tWybrales\las opcje 'b'. Poleceniem
        'echo' wpisales(dodales) wynik polecenia 'date'
        do pliku daty.log.\n" && echo $(date) >> daty.log;;
```

Pierwsza czesc skryptu

```
                c) echo -e "\n\tWybrales\las opcje 'c'. Polecenie
        cat wyswietla zawartosc pliku 'daty.log'.\n\tWynik:
        $(cat daty.log)\n";;
                x) echo -e "\n\tWybrales\las opcje 'x' co oznacza
        zakonczenie petli. Bye!\n" && break;;
                [0-9]) for (( count=0; count< $line; count++ )); \
        do cat daty.log; done;;
        esac
done
```

Druga czesc skryptu

(b) Przykładowe użycie sktyptu:

```
$ ./pkt6.sh

Wybierz opcje: a

Wybrales\las opcje 'a'. Polecenie 'cat'
zapisalo zawartosc pliku '/dev/null' do
pliku daty.log w biezacym katalogu.
```

Używane funkcje i polecenia: `echo`, `while`, `do`, `read`, `case`, `cat`, `break`.

4. **Zadanie dla zaawansowanych** Utworzyłem skrypt, który wyjście polecenia `date` przesyła do polecenia `read`, aby wyświetlić date w formacie *rok-miesiac-dzien godzina:minuty:sekundy*. Bez uzycia parametra `"+%Y-%m-%d %H:%M:%S"` polecenia date.

```
#!/bin/bash

data="$(date | awk -F' ' '{print $4"-"$3"-"$2,$5,$6}')"

echo $data
```

Opis działania skryptu: w tym skrypcie użyłem `AWK`. `AWK` to taki program który jest używany dla wyszukiwania, modyfikacji tekstu.

`AWK` ma taką postać:

```
$ awk 'opcja' 'warunek{akcja}'
```

Kilka opcji które przyjmuje `AWK`:

- -F(separator)
- -f(wczytanie danych z pliku)
- -o(wypisac wynik do pliku)

Jakie moga być akcji:

- print - wyswietlic za pomoca standartowego wyjscia
- printf - formatowane wyswietlenie za pomoca standartowego wyjscia
- lenght - zwraca dlugosc wiersza.

W tym skrypcie uzywałem opcje `-F` i akcje `print`.

(a) Po pierwsze w skrypcie wykona się polecenie `date` i dane wyjsciowe idą do następnego polecenia `awk`. Symbol | oznacza że dane wyjściowe z poprzedniego polecenia będa używane jako dane wejściowe dla następnego polecenia.

```
$ date
Wed 07 Dec 2022 06:33:53 PM CET
```

(b) Po wykonaniu polecenia `date` dane idą do polecenia `awk`. Patrząc na wyjsciowa linie widzimy dane rozdzielane spacjami i za pomocą opcji `-F` polecenia `awk` możemy logicznie rozdzielic tą linie na kolumny, czyli

- `Wed` kolumna pierwsza(argument pierwszy)
- `07` kolumna druga(argument drugi)
- itp.

W takim razie kazda kolumna jest argumentem. Polecenie `awk` pozwala wyświetlac te kolumny jak chcemy(np. w odwrotnej kolejności lub wyświetliac nie wszystkie kolumny) Jak wyglada wynik używajac polecenia date i awk

```
$ date
Wed 07 Dec 2022 06:33:53 PM CET
$ ./pkt7.sh
2022-Dec-07 06:33:54 PM
```

Uzywane funkcje i polecenia: `date, echo, awk`.

5 Obsługa edytora VI/VIM

1. Przeszedłem do katalogu `cw3`. Poleceniem `vim pkt1` otworzyłem plik `pkt1`.
2. Poruszałem się po pliku używając klawiszę [h] znak w lewo, [j] linia w dol, [k] linia w gore, [l] znak w prawo.
3. Wrociłem na początek pliku poleceniem `gg`.
4. Przeszedłem do ostatniej linii w pliku poleceniem `G`.
5. Usunałem bieżacą linie poleceniem `dd`.
6. Dodałem nową linie poniżej stosując polecenie `o`.
7. Wpowadziłem dowolny tekst komentarza np. # `to jest linia komentarza`, wyszedłem z trybu wprowadzania klawiszem *Escape*([⎋])
8. Wstawiłem wycięta linie poleceniem `p`.
9. Zapisałem plik poleceniem `:w`[↵].
10. Przeszedłem do trzeciej linii poleceniem `3G`.
11. Zakonczyłem prace edytora VIM przez `:q`[↵].

6 Zakonczenie cwiczenia

Usunałem utworzone pliki i katalogi poleceniem: `rm -rf ~/cw3`.