LABORATORIUM SOP

PRZYPOMNIENIE z UKOS *info.txt

1. Zalogować się na szuflandię do swojego katalogu domowego.
2. Wyświetlić bieżącą ścieżkę.
3. W katalogu domowym utworzyć katalogi SOP_BASH oraz SOP_C.
4. Stworzyć pliki SOP_BASH\bash_info.txt oraz SOP_C\C_info.txt
5. Wpisać do pliku bash_info.txt swoje imię i nazwisko, semestr i nazwę przedmiotu
6. Wyświetlić zawartość katalogu domowego w postaci listy.

Wyświetlić zawartość katalogu domowego w postaci drzewa.

HELLO SOP BASH WORLD sop1.sh

W katalogu SOP_BASH tworzymy plik sop1.sh, w którym umieścimy pierwszy skrypt:

```
touch sop1.sh
```

Następnie za pomocą dowolnego edytora ASCII (vi, gedit, itp.) wpisujemy do niego zawartość:

```
#!/bin/bash

#To jest komentarz.

echo "Hello SOP Bash world"
```

Znak **#** (hasz) oznacza komentarz, wszystko co znajduje się za nim w tej samej linii, jest pomijane przez interpreter. Pierwsza linia skryptu zaczynająca się od znaków: **#!** ma szczególne znaczenie - wskazuje na rodzaj shella w jakim skrypt ma być wykonany, tutaj skrypt zawsze będzie wykonywany przez interpreter poleceń **/bin/bash**, niezależnie od tego jakiego rodzaju powłoki w danej chwili używamy.

```
echo "Hello World"
```

wydrukuje na standardowym wyjściu (stdout) czyli na ekranie napis: **Hello World**.

Aby móc uruchomić skrypt należy mu jeszcze nadać atrybut wykonywalności za pomocą polecenia:

```
chmod +x sop1.sh
```

Jeśli katalog bieżący w którym znajduje się skrypt nie jest dopisany do zmiennej **PATH**, to nasz skrypt możemy uruchomić w ten sposób:

```
./sop1.sh
```

podobnie można podać również pełną ścieżkę:

```
/home/user/sop1.sh
```

ECHO

`sop2*.sh`
`sop2*_out.txt`

Polecenie echo służy do wydrukowania na standardowym wyjściu (**stdout** - domyślnie jest to ekran) napisu.

Składnia:

```
echo "TEKST"
```

Parametry:

-n nie jest wysyłany znak nowej linii

-e włącza interpretacje znaków specjalnych

- **\a** czyli alert, usłyszysz dzwonek
- **\b** backspace
- **\c** pomija znak kończący nowej linii
- **\f** escape
- **\n** form feed czyli wysuw strony
- **\r** znak nowej linii
- **\t** tabulacja pozioma
- **\v** tabulacja pionowa
- \\ backslash
- **\nnn** znak, którego kod ASCII ma wartość ósemkowo
- **\xnnn** znak, którego kod ASCII ma wartość szesnastkowo

W katalogu SOP_BASH tworzymy plik sop2.sh, w którym umieścimy skrypt:

```
#!/bin/bash
echo -n "1.BEZ NOWEJ LINI"
echo "2.Ze znakiem nowej linii."      #wydrukuje tekst na ekranie
```

Można też pisać do pliku. W tym wypadku **echo** wydrukuje tekst do pliku, ale zmaże całą jego wcześniejszą zawartość, jeśli plik podany na standardowym wyjściu nie istnieje, zostanie utworzony.

```
echo "jakiś napis" > plik
```

Tutaj napis zostanie dopisany na końcu pliku, nie zmaże jego wcześniejszej zawartości.

```
echo "jakiś napis" >> plik
```

W katalogu SOP_BASH tworzymy plik sop2a.sh, w którym umieścimy skrypt

```
#!/bin/bash
echo -n "1.BEZ NOWEJ LINI" > sop2a_out.txt
echo "2.Ze znakiem nowej linii." > sop2a_out.txt #zapisze tekst do pliku
```

W katalogu SOP_BASH tworzymy plik sop2b.sh, w którym umieścimy skrypt

```
#!/bin/bash
echo -n "1.BEZ NOWEJ LINI" >> sop2b_out.txt
echo "2.Ze znakiem nowej linii." >> sop2b_out.txt #dopisze tekst do pliku
```

Wykonajmy kilka razy skrypt `sop2a.sh` i sprawdźmy zawartość pliku `sop2a_out.txt` .
Wykonajmy kilka razy skrypt `sop2b.sh` i sprawdźmy zawartość pliku `sop2b_out.txt` .

CYTOWANIE `sop3*.sh`

Znaki cytowania służą do usuwania interpretacji znaków specjalnych przez powłokę.

Wyróżnia się nastepujące znaki cytowania:

- **cudzysłów** (ang. *double quote*)
- `" "`

  Między cudzysłowami umieszcza się tekst, wartości zmiennych zawierające spacje.

  Cudzysłowy zachowują znaczenie specjalne trzech znaków:

  - **$** wskazuje na nazwę zmiennej, umożliwiając podstawienie jej wartości
  - \ znak maskujący
  - ` ` odwrotny apostrof, umozliwia zacytowanie polecenia

  W katalogu SOP_BASH tworzymy plik sop3a.sh, w którym umieścimy skrypt

```
#!/bin/bash

x=2
echo "Wartosc zmiennej x to $x"    #wydrukuje Wartość zmiennej x to 2
echo -e "Usłyszysz dzwonek\a"
echo "Polecenie date pokaże: `date`"
```

- **apostrof** (ang. *single quote*)
- `' '`

  Wszystko co ujęte w znaki apostrofu traktowane jest jak łańcuch tekstowy, apostrof wyłącza interpretowanie wszystkich znaków specjalnych, traktowane są jak zwykłe znaki.

  W katalogu SOP_BASH tworzymy plik sop3b.sh, w którym umieścimy skrypt

```
#!/bin/bash

echo '$USER'   #nie wypisze twojego loginu
```

- **odwrotny apostrof** (ang. *backquote*)
- ` ` `

  umożliwia zacytowanie polecenia, bardzo przydatne jeśli chce się podstawić pod zmienną wynik jakiegoś polecenia.

  W katalogu SOP_BASH tworzymy plik sop3c.sh, w którym umieścimy skrypt

```
#!/bin/bash

x=`ls -la $PWD`
echo $x                          #pokaże rezultat polecenia
```

  Alternatywny zapis, który ma takie samo działanie wygląda tak:

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

```
echo $(ls -la $PWD)
```

- **backslash** czyli znak maskujący
- \
  Jego działanie najlepiej wyjaśnić na przykładzie: w celu by na ekranie pojawił się napis $HOME (wyłączyć interpretacje przez powłokę tej zmiennej) piszemy :
  ```
  echo \$HOME                          #i jest napis $HOME
  ```

  W katalogu SOP_BASH tworzymy plik sop3d.sh, w którym umieścimy skrypt
  ```
  echo "$HOME"                         #wydrukuje /home/ja
  ```

  aby wyłączyć interpretacje przez powłokę tej zmiennej, należy wpisać:

  ```
  echo \$HOME                          #i jest napis $HOME
  ```

Słowa zastrzeżone (ang. reserved words)

Mają dla powłoki specjalne znaczenie, wtedy gdy nie są cytowane.

Lista słów zastrzeżonych:

- **!**
- **case**
- **do**
- **done**
- **elif**
- **else**
- **esac**
- **fi**
- **for**
- **function**
- **if**
- **in**
- **select**
- **then**
- **until**
- **while**
- **{**
- **}**
- **time**
- **[**
- **]**

ZMIENNE PROGRAMOWE `sop4*.sh`

(ang. program variables)

To zmienne definiowane samodzielnie przez użytkownika.

```
nazwa_zmiennej="wartość"
```
Na przykład:
```
x="napis"
```
Do zmiennej odwołujemy się poprzez podanie jej nazwy poprzedzonej znakiem **$** i tak dla zmiennej **x** może to wyglądać następująco:
```
echo $x
```
Na co należy uważać? **Nie może być spacji po obu stronach!**
```
x = "napis"
```
ten zapis jest błędny

Pod zmienną możemy podstawić wynik jakiegoś polecenia, można to zrobić na dwa sposoby:

- Poprzez użycie odwrotnych apostrofów:

  ```
  `polecenie`
  ```

  Przykład: `GDZIE_JESTEM=`pwd``

  W katalogu SOP_BASH tworzymy plik sop4a.sh, w którym umieścimy skrypt
  ```
  #!/bin/bash

  GDZIE_JESTEM=`pwd`
  echo "Jestem w katalogu $GDZIE_JESTEM"
  ```

  Wartością zmiennej **GDZIE_JESTEM** jest wynik działania polecenia **pwd**, które wypisze nazwę katalogu w jakim się w danej chwili znajdujemy.

- Za pomocą rozwijania zawartości nawiasów:

  ```
  $(polecenie)
  ```

  Przykład: `GDZIE_JESTEM=$(pwd)`

  W katalogu SOP_BASH tworzymy plik sop4b.sh, w którym umieścimy skrypt
  ```
  #!/bin/bash

  GDZIE_JESTEM=$(pwd)
  echo "Jestem w katalogu $GDZIE_JESTEM"
  ```

ZMIENNE SPECJALNE `sop5*.sh`

(ang. special variables, special parameters)
To najbardziej prywatne zmienne powłoki, są udostępniane użytkownikowi tylko do odczytu (są wyjątki). Kilka przykładów:

- **$0**

  nazwa bieżącego skryptu lub powłoki

  W katalogu SOP_BASH tworzymy plik sop5a.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash

  echo "$0"
  ```

  ./sop5a.sh pokaże nazwę uruchomionego skryptu.

- **$1..$9**

  Parametry przekazywane do skryptu (wyjątek, użytkownik może modyfikować ten rodzaj $-ych specjalnych.

  W katalogu SOP_BASH tworzymy plik sop5b.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash

  echo "$1 $2 $3"
  ```

  Jeśli wywołany zostanie skrypt z jakimiś parametrami to przypisane zostaną zmiennym: od **$1** do **$9**. Zobacz co się stanie jak podasz za małą liczbę parametrów oraz jaki będzie wynik podania za dużej liczby parametrów.

- **$@**

  Pokaże wszystkie parametry przekazywane do skryptu (też wyjątek), równoważne **$1 $2 $3**..., jeśli nie podane są żadne parametry **$@** interpretowana jest jako nic.

  W katalogu SOP_BASH tworzymy plik sop5c.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash

  echo "Skrypt uruchomiono z parametrami: $@"
  ```

  A teraz wywołaj ten skrypt z jakimiś parametrami, mogą być brane z powietrza np.:

  ./sop5c.sh `-a d`

  Efekt będzie wyglądał następująco:

  ```
  Skrypt uruchomiono z paramertami -a d
  ```

- **$?** kod powrotu ostatnio wykonywanego polecenia
- **$$** PID procesu bieżącej powłoki

ZMIENNE ŚRODOWISKA `sop6.sh`

(ang.environment variables)

Definiują środowisko użytkownika, dostępne dla wszystkich procesów potomnych. Można je podzielić na:

- globalne - widoczne w każdym podshellu
- lokalne - widoczne tylko dla tego shella w którym został ustawione

```
x="napis"                  #Deklaracja zmiennej lokalnej:
echo $x                    # wyświetli wartość zmiennej x
```

Aby zainicjować zmienną globalną widoczną piszemy:

```
export x="napis"
```

Teraz zmienna **x** będzie widoczna w podshellach, jak widać wyżej służy do tego polecenie **export**, nadaje ono wskazanym zmiennym atrybut zmiennych globalnych. W celu uzyskania listy aktualnie eksportowanych zmiennych należy wpisać **export**, opcjonalnie **export -p** . Na tej liście przed nazwą każdej zmiennej znajduje się zapis:

```
declare-x
```

To wewnętrzne polecenie BASH-a, służące do definiowania zmiennych i nadawania im atrybutów, **-x** to atrybut eksportu czyli jest to, to samo co polecenie **export**. Ale tu uwaga! Polecenie **declare** występuje tylko w BASH-u, nie ma go w innych powłokach, natomiast **export** występuje w **ksh**, **ash** i innych, które korzystają z plików startowych **/etc/profile**. Dlatego też zaleca się stosowanie polecenia **export**.

```
export -n zmienna
```

spowoduje usunięcie atrybutu eksportu dla danej zmiennej

Niektóre przykłady zmiennych środowiskowych:

```
$HOME          #ścieżka do twojego katalogu domowego
$USER          #twój login
$HOSTNAME      #nazwa twojego hosta
$OSTYPE        #rodzaj systemu operacyjnego
```

itp. dostępne zmienne środowiskowe można wyświetlić za pomocą polecenia:

```
printenv | more
```

W katalogu SOP_BASH tworzymy plik sop6.sh, w którym umieścimy skrypt

```
#!/bin/bash

echo "Scieżka do twojego katalogu domowego: $HOME"
echo "Twoj login: $USER"
echo "Nazwa twojego hosta: $HOSTNAME"
echo "Rodzaj twojego systemu: $OSTYPE"
```

ZMIENNE TALICOWE sop7*.sh

BASH pozwala na stosowanie zmiennych tablicowych jednowymiarowych. Czym jest tablica? To zmienna która przechowuje listę jakichś wartości (rozdzielonych spacjami), w BASH'u nie ma maksymalnego rozmiaru tablic. Kolejne wartości zmiennej tablicowej indexowane są przy pomocy liczb całkowitych, zaczynając od **0**.

- **Deklaracja elementów tablicy.**

  Składnia

  ```
  zmienna=(wartość1 wartość2 wartość3 wartośćn)
  ```

  W katalogu SOP_BASH tworzymy plik sop7a.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash
  tablica=(element1 element2 element3)
  echo ${tablica[0]}
  echo ${tablica[1]}
  echo ${tablica[2]}
  ```

  Zadeklarowana została zmienna tablicowa o nazwie: **tablica**, zawierająca trzy wartości: **element1 element2 element3**. Natomiast polecenie: **echo ${tablica[0]}** wydrukuje na ekranie pierwszy elementu tablicy. W powyższym przykładzie w ten sposób wypisana zostanie cała zawartość tablicy. Do elementów tablicy **odwołuje się** za pomocą **wskaźników**.

- **Odwołanie do elementów tablicy.**

  Składnia:

  ```
  ${nazwa_zmiennej[wskaźnik]}
  ```

  W katalogu SOP_BASH tworzymy plik sop7b.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash
  tablica=(element1 element2 element3)
  echo ${tablica[*]}
  ```

  Poniższy skrypt robi to samo co wcześniejszy.

Wskaźnikiami są indexy elementów tablicy, począwszy od **0** do **n** oraz **@**, *****. Gdy odwołując się do zmiennej nie poda się wskaźnika: **${nazwa_zmiennej}** to nastąpi odwołanie do elementu tablicy o indexie **0**.Jeśli wskaźnikiem będą: **@** lub ***** to zinterpretowane zostaną jako wszystkie elementy tablicy, w przypadku gdy tablica nie zawiera żadnych elementów to zapisy: **${nazwa_zmiennej[wskaźnik]}** lub **${nazwa_zmiennej[wskaźnik]}** są interpretowane jako nic.

Można też uzyskać długość (liczba znaków) danego elementu tablicy:

```
${#nazwa_zmiennej[wskaźnik]}
```

W katalogu SOP_BASH tworzymy plik sop7c.sh, w którym umieścimy skrypt

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${#tablica[0]}
```

Polecenie **echo ${#tablica[0]}** wydrukuje liczbę znaków z jakich składa się pierwszy element tablicy: **element1** wynik to **8**. W podobny sposób można otrzymać liczbę wszystkich elementów tablicy, wystarczy jako wskaźnik podać: **@** lub *****.

W katalogu SOP_BASH tworzymy plik sop7d.sh, w którym umieścimy skrypt

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${#tablica[@]}
```

który po wykonaniu ./sop7d.sh da wynik: **3**.

- **Dodawanie elementów do tablicy.**

  Składnia:

  ```
  nazwa_zmiennej[wskaźnik]=wartość
  ```

  Przykład:

  ```
  #!/bin/bash

  tablica=(element1 element2 element3)
  tablica[3]=element4

  echo ${tablica[@]}
  ```

  Jak wyżej widać do tablicy został dodany **element4** o indexie **3**. Mechanizm dodawania elementów do tablicy, można wykorzystać do tworzenia tablic, gdy nie istnieje zmienna tablicowa do której dodajemy jakiś element, to BASH automatycznie ją utworzy.

  W katalogu SOP_BASH tworzymy plik sop7e.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash
  linux[0]=slackware
  linux[1]=debian
  echo ${linux[@]}
  ```

Utworzona została tablica **linux** zawierająca dwa elementy.

- **Usuwanie elementów tablic i całych tablic.**

  Dany element tablicy usuwa się za pomocą polecenia **unset**.

  Składnia:

```
unset nazwa_zmiennej[wskaźnik]
```

  W katalogu SOP_BASH tworzymy plik sop7f.sh, w którym umieścimy skrypt

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[@]}
unset tablica[2]
echo ${tablica[*]}
```

  Usunięty został ostatni element tablicy.

  Aby usunąć całą tablicę wystarczy podać jako wskaźnik: **@** lub ***.**

  W katalogu SOP_BASH tworzymy plik sop7g.sh, w którym umieścimy skrypt

```
#!/bin/bash
tablica=(element1 element2 element3)
unset tablica[*]
echo ${tablica[@]}
```

  Zmienna tablicowa o nazwie **tablica** przestała istnieć, polecenie: **echo ${tablica[@]}** nie wyświetli nic.

STRUMIENIE DANYCH sop8*.sh

Strumienie danych

Każdy uruchomiony w Linuxie proces skądś pobiera dane, gdzieś wysyła wyniki swojego działania i komunikaty o błędach. Tak więc procesowi przypisane są trzy strumienie danych:

- **stdin** (ang. *standard input*) czyli **standardowe wejście**, skąd proces pobiera dane, domyślnie jest to **klawiatura**
- **stdout** (ang. *standard otuput*) to **standardowe wyjście**, gdzie wysyłany jest wynik działania procesu, domyślnie to **ekran**
- **stderr** (ang. *standard error*) **standardowe wyjście błędów**, tam trafiają wszystkie komunikaty o błędach, domyślnie **ekran**

Linux wszystko traktuje jako plik, niezależnie od tego czy to jest plik zwykły, katalog, urządzenie blokowe (klawiatura, ekran) itd. Nie inaczej jest ze strumieniami, **BASH** identyfikuje je za pomocą przyporządkowanych im liczb całkowitych (od **0** do **2**) tak zwanych **deskryptorów plików**.
I tak:

- **0** to plik z którego proces pobiera dane **stdin**
- **1** to plik do którego proces pisze wyniki swojego działania **stdout**
- **2** to plik do którego trafiają komunikaty o błędach **stderr**

Za pomocą **operatorów przypisania** można manipulować strumieniami, poprzez przypisanie deskryptorów: **0**, **1**, **2** innym plikom, niż tym reprezentującym klawiaturę i ekran.

- **Przełączanie standardowego wejścia**

  Zamiast klawiatury jako standardowe wejście można otworzyć plik:

  ```
  < plik
  ```

  Przykład:

  Najpierw stwórzmy plik **lista** o następującej zawartości:

  ```
  slackaware
  redhat
  debian
  caldera
  ```

  Użyjemy polecenia **sort** dla którego standardowym wejściem będzie nasz plik.

  ```
  sort < lista
  ```

  Wynikiem będzie wyświetlenie na ekranie posortowanej zawartość pliku **lista**:

  ```
  caldera
  debian
  redhat
  slackware
  ```

- **Przełączanie standardowego wyjścia**

  Wynik jakiegoś polecenia można wysłać do pliku, a nie na ekran, do tego celu używa się operatora:

  ```
  > plik
  ```

  W katalogu SOP_BASH tworzymy plik sop8a.sh, w którym umieścimy skrypt

  ```
  #!/bin/bash
  ls -la /usr/bin > ~/wynik
  ```

  Rezultat działania polecenia **ls -la /usr/bin** trafi do pliku o nazwie **wynik**, jeśli wcześniej nie istniał plik o takiej samej nazwie, to zostanie utworzony, jeśli istniał cała jego poprzednia zawartość zostanie nadpisana.

  Jeśli chcemy aby dane wyjściowe dopisywane były na końcu pliku, bez wymazywania jego wcześniejszej zawartości, stosujemy operator:

```
>> plik
```

W katalogu SOP_BASH tworzymy plik sop8b.sh, w którym umieścimy skrypt

```
#!/bin/bash
free -m >> ~/wynik
```

Wynik polecenia **free -m** (pokazuje wykorzystanie pamięci RAM i swap'a) zostanie dopisany na końcu pliku **wynik**, nie naruszając jego wcześniejszej zawartości.

- **Przełączanie standardowego wyjścia błędów**

Do pliku można też kierować strumień diagnostyczny:

```
2> plik
```

W katalogu SOP_BASH tworzymy plik sop8c.sh, w którym umieścimy skrypt

```
#!/bin/bash
echo "Stderr jest skierowane do pliku error"
ls -y 2> ~/error    #błąd
```

W powyższym skrypcie polecenie **ls** jest użyte z błędną opcją **-y**, komunikat o błędzie trafi do pliku **error**.

Za pomocą operatora:

```
<< plik
```

można dopisać do tego samego pliku kilka komunikatów o błędach, dopisanie kolejnego nie spowoduje skasowania wcześniejszej zawartości pliku.

W katalogu SOP_BASH tworzymy plik sop8d.sh, w którym umieścimy skrypt

```
#!/bin/bash
echo "Stderr jest skierowane do pliku error"
ls -y 2>> ~/error             #błąd
cat /etc/shadow 2>> ~/error   #błąd2
```

Jako błąd drugi zostanie potraktowane polecenie **cat /etc/shadow** (zakładając, że zalogowałeś się jako **użytkownik**) ponieważ prawo **odczytu** pliku **/etc/shadow** ma tylko **root**.