

[SOP] Ćwiczenia III:

▼ Polecenia:

- `mv` - przemieszczanie plików między katalogami lub zmiana nazwy plików; w zależności od drugiego argumentu (`mv file1.txt renamed.txt`)
- `rm` - usuwanie plików i katalogów
 - `rm -d` - usuwanie pustych katalogów
 - `rm -r` - usuwanie rekurencyjne
 - `rm -f` - nie patrz na komunikaty po prostu usuń
- `rmdir` - usuwanie katalogów
- `for`
- `while`
- `grep` - przeszukuje linie w pliku na podstawie schematu
- `egrep` - przeszukuje linie w pliku na podstawie wyrażenia regularnego
- `"^a.{6}"` - . oznacza
- `*` - wszystkie pliki i katalogi
- `*.txt` - wszystkie pliki z rozszerzeniem txt

▼ Wyrażenia:

- `-f FILE` - jeśli istnieje i jest zwykłym plikiem
- `-d FILE` - jeśli istnieje i jest katalogiem
- `-e FILE` - `plik` istnieje
- `-r FILE` - `plik` istnieje i przyznano uprawnienia do odczytu
- `-w FILE` - `plik` istnieje i przyznano uprawnienia do zapisu
- `-x FILE` - `plik` istnieje i przyznano uprawnienia do wykonywania
- `FILE1 -ot FILE2` - `plik1` jest starszy niż `plik2`
- `FILE1 -nt FILE2` - `plik1` jest nowszy niż `plik2`
- `-z STRING` - długość `STRING` wynosi zero
- `-n STRING` - długość `STRING` jest niezerowa
- `EXPRESSION1 -a EXPRESSION2` - zarówno `EXPRESSION1` jak i `EXPRESSION2` są prawdziwe
- `EXPRESSION1 -o EXPRESSION2` - albo `EXPRESSION1` albo `EXPRESSION2` ma wartość true
- `STRING1 = STRING2` - ciągi są równe
- `STRING1 != STRING2` - ciągi nie są równe
- `INTEGER1 -eq INTEGER2` - `INTEGER1` jest równy `INTEGER2`
- `INTEGER1 -ge INTEGER2` - `INTEGER1` jest większy lub równy `INTEGER2`
- `INTEGER1 -gt INTEGER2` - `INTEGER1` jest większy niż `INTEGER2`
- `INTEGER1 -le INTEGER2` - `INTEGER1` jest mniejsza lub równa `INTEGER2`

- `INTEGER1 -lt INTEGER2` - `INTEGER1` jest mniejszy niż `INTEGER2`
- `INTEGER1 -ne INTEGER2` - `INTEGER1` nie jest równy `INTEGER2`

▼ Skrypt 1:

```
#!/bin/bash

# zmienną lepiej dawać w cudzysłów jeśli się nie wie czy będzie podana
if [ -n $1 ] # negacja -> -z; czy podany string nie ma długości zero
then
    echo "It is nonzero string"
else
    echo "It is empty string"
fi
```

▼ Skrypt 2:

```
#!/bin/bash

if [ "$1" \> "$2" ] # bez znaku ucieczki / zostało by to odczytane jako strumień
then
    echo "$1 is after $2"
elif [ "$1" \< "$2" ] # bez znaku ucieczki / zostało by to odczytane jako strumień
then
    echo "$1 is before $2"
elif [ "$1" = "$2" ] # negacja - !=
then
    echo "$1 is equal to $2"
fi
```

▼ Skrypt 3:

```
#!/bin/bash

# czy coś jest plikiem lub katalogiem czy coś nie istnieje
if [ -f "$1" ] # czy istnieje i jest plikiem regularnym
then
    echo "it is regular file"
elif [ -d "$1" ]
then
    echo "it is a dir"
else
    echo "it is something else or doesn't exist"
fi
```

▼ Skrypt 4:

```
#!/bin/bash

if [ -e $1 ]
then
    echo "it exists"
else
    echo "it doesn't exist"
fi
```

▼ Skrypt 5:

```
#!/bin/bash

if [ -r $1 ] # czy plik istnieje i ma uprawnienia do czytania
then
    echo "it has read perm"
fi

if [ -w $1 ] # czy plik istnieje i ma uprawnienia do edycji
then
    echo "it has write perm"
fi

if [ -x $1 ] # czy plik istnieje i ma uprawnienia do odpalania
then
    echo "it has exec perm"
fi
```

▼ Skrypt 6:

```
#!/bin/bash

if [ $1 -nt $2 ] # negacja -> -ot; czy jest nowsze niż
then
    echo "$1 is newer than $2"
else
    echo "$1 is older than $2"
fi
```

▼ Skrypt 7:

```
#!/bin/bash

if [ $1 -gt $2 -a $2 -lt 10 ] # jedno wyrażenie testujące -a to AND
# if [ $1 -gt $2 ] && [ $2 -lt 10 ]
then
    echo "success?"
else
    echo "fail?"
```

```

fi

if [ $1 -eq $2 -o $2 -ge 5 ]
#if [ $1 -eq $2 ] || [ $2 -ge 5 ]
then
    echo "nice?"
else
    echo "bad?"
fi

```

▼ Skrypt 8:

```

#!/bin/bash

text=$(head -5 $1)
for word in $text
do
    echo "next word is $word"
done

```

```

#!/bin/bash

for file in $(cat $1) # każdy biały znak traktuje jako separator
do
    echo "next word is $wone"

```

```

#!/bin/bash

for file in *.sh
do
    #echo $(cat $file)
    echo "next bash script is $file"
done

```

```

#!/bin/bash

for arg in $@
do
    "echo next arg is $arg"
done

```

▼ Skrypt 9:

```

#!/bin/bash

i=0
while [ "$i" -lt 5 ]
do

```

```
echo "value i is $i"
((i+=1)) # działa też przy użyciu let lub expr
done
```

▼ Skrypt 10:

```
#!/bin/bash

i=0
#head -5 $1 | while read -r line
text=$(head -5 $1)
while read -r line # podział po znaku nowej linii
do
    echo "next line is $line"
    ((i+=1)) # podwatek - poza nim wartość się nie zmieni
    echo i value is $i
done <<< "$text" #< $1
# < $1 # strumień skierowany do done bo cała pętla while zaczyna się na while
a kończy na done
echo "final i value is $i"
```

▼ Wyrażanie regularne:

- ▼  dopasowuje **dowolny znak** oprócz nowej linii:

```
#!/bin/bash

fruits_file=`cat fruit.txt | grep App.e` # Apple
echo "$fruits_file"
```

- ▼  dopasowuje **początek łańcucha**:


```
#!/bin/bash

fruits_file=`cat fruit.txt | grep ^B`
echo "$fruits_file" # wszystkie słowa zaczynające się na literę B
```

- ▼  dopasowuje **koniec łańcucha**:

```
#!/bin/bash

fruits_file=`cat fruit.txt | grep e$`
echo "$fruits_file" # wszystkie słowa kończące się na literę e
```

- ▼  dopasowuje zero lub więcej wystąpień, czyli **dowolną liczbę razy, znaku w łańcuchu**:

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

```
fruits_file=`cat fruit.txt | grep ap*le`  
echo "$fruits_file" # wszystkie słowa mające w sobie sekwencję ap_le
```

- ▼  używane do **unikania następującego po nim znaku**:

```
#!/bin/bash  
  
fruits_file=`cat fruit.txt | grep "\ "`  
echo "$fruits_file" # wszystkie słowa mające w sobie jedną spację
```

- ▼  używane do **dopasowywania lub wyszukiwania zestawu wyrażeń regularnych**:

```
#!/bin/bash  
  
fruits_file=`cat fruit.txt | grep -E "(fruit)"`  
echo "$fruits_file" # wszystkie słowa mające w sobie słowo fruit
```

- ▼  dopasowuje dokładnie **jeden znak w łańcuchu** lub strumieniu:

```
#!/bin/bash  
  
fruits_file=`cat fruit.txt | grep -E Ch?`  
echo "$fruits_file" # wszystkie słowa mające w sobie Ch
```