man – wyświetlenie pomocy

pwd – gdzie jestem

cd – przejście do katalogu głównego
cd *dir* – przejście do katalogu *dir*
cd .. – cofnięcie się do katalogu nadrzędnego
cd *../dir* – cofnięcie się do katalogu nadrzędnego i przejście do katalogu *dir* (ścieżka **względna**)
cd */dir_1/dir_2* – przejście od katalogu głównego, przez katalog *dir_1* do katalogu *dir_2* (ścieżka **bezwzględna**)

ls *dir* – wyświetlenie zawartości katalogu *dir*
ls -a – wyświetlenie całej zawartości, także tej ukrytej
ls -l – wyświetlenie zawartości razem ze wszystkimi danymi
ls -la – połączenie ls -a i ls -l

mkdir *dir* – stworzenie katalogu o nazwie *dir*
rmdir *dir* – usunięcie pustego katalogu o nazwie *dir*
mv *dir_1 dir_2* – przeniesienie katalogu *dir_1* do katalogu *dir_2*
touch *file* – utworzenie pliku o nazwie *file*
cat > *file* – edycja pliku *file*
find *dir* -name *file* – wyszukanie pliku o nazwie *file* w katalogu *dir*
cp *file dir* – skopiowanie pliku *file* do katalogu *dir*
mv *file_1 file_2* – zmiana nazwy pliku *file_1* na *file_2*
rm *file* – usunięcie pliku *file*
rm -r *dir_1/dir_2* – usunięcie katalogu *dir_2* znajdującego się w katalogu *dir_1* razem z zawartością
ln -s *dir link* – utworzenie linku symbolicznego *link* do katalogu *dir*

***** - dowolny ciąg znaków
ls /etc/a*b – wyświetlenie zawartości katalogu *etc*, która zaczyna się na ‘a’ i kończy na ‘b’
? – jeden dowolny znak
cp ??? *dir* – skopiowanie wszystkich trzy znakowych plików do katalogu *dir*
[] – dokładnie jeden spośród znaków w nawiasach
cp [m-p]?? – skopiowanie plików, które zaczynają się znakami od ‘m’ do ‘p’ i kończą dwoma dowolnymi

sudo – pozwala na wykonanie poleceń, do których są wymagane uprawnienia administratora
sudo cat /etc/passwd – wylistowanie użytkowników oraz ich podstawowych danych
sudo cat /etc/shadow – wylistowanie zaszyfrowanych haseł i informacji uzupełniających użytkowników
sudo cat /etc/group – wylistowanie grup
id *user* lub **groups *user* –** wyświetlenie grup, do których należy użytkownik *user*
users lub **w** lub **who** lub **whoami –** wyświetlenie aktualnie zalogowanych użytkowników
sudo chgrp *group file_1 file_2* – zmiana grupy plików *file_1* i *file_2* na *group*
sudo chown *owner file* – zmiana właściciela pliku file na *owner*
sudo chown -r *user*:*group dir* – zmiana właściciela i grupy wszystkich plików w katalogu *dir*

sudo useradd *user* – utworzenie nowego użytkownika o nazwie *user*
-s – nadanie użytkownikowi powłoki
-g – przypisanie użytkownika do grupy
-m – utworzenie nowego katalogu domowego dla użytkownika
sudo useradd -s /bin/bash -g *group* -m *user* – użytkownik *user* należący do grupy *group* mający powłokę */bin/bash*
sudo chsh *user* – zmiana powłoki użytkownika *user*
sudo chfn *user* – zmiana nazwy oraz innych informacji użytkownika *user*
finger *user* – wyświetlenie informacji o użytkowniku *user*
last *number user* – wyświetlenie *number* ostatnich logowań użytkownika *user*
last 10 student – wyświetla 10 ostatnich logowań użytkownika student
last -n 10 – wyświetla 10 ostatnich logowań i restartów systemu
sudo passwd *user* – nadanie lub zmiana hasła użytkownika *user*

su *user* – logowanie na konto innego użytkownika *user*
ssh *user@pc* – logowanie na konto *user* na innym komputerze o nazwie hosta lub adresie IP *pc*
exit – powrót na swoje konto

chmod *perm file* (*dir*) – nadanie określonych uprawnień dla pliku *file* (lub katalogu *dir*)
r – uprawnienia do czytania zawartości pliku (przeszukania zawartości katalogu)
w – uprawnienia do zmiany zawartości pliku (katalogu)
x – uprawnienia do wykonywania pliku (przejścia do katalogu)
1 (0 + 0 + 1) –x
2 (0 + 2 + 0) -w-
3 (0 + 2 + 1) -wx
4 (4 + 0 + 0) r--
5 (4 + 0 + 1) r-x
6 (4 + 2 + 0) rw-
7 (4 + 2 + 1) rwx
chmod 641 *file* – właściciel rw, grupa r, pozostali użytkownicy x
a – wszyscy użytkownicy
u – właściciel pliku
g – grupa pliku
o – inni użytkownicy
- - odebranie prawa
+ - dodanie prawa
= - ustalenie nowych praw niezależnie od poprzednich
chmod u-w,g+w,o=rw *file*
umask *perm* – zmiana domyślnych praw nowo utworzonych plików
umask 133 – nowe pliki będą miały uprawnienia 644 (bo 777-644=133)

ps – wyświetlanie procesów
-l – wyświetla więcej informacji
-A lub **-e** lub **aux** – wyświetla wszystkie procesy
kill -s 9 *PID* – natychmiastowe zakończenie procesu o numerze *PID*
killall -s 9 *name* – natychmiastowe zakończenie procesu o nazwie *name*
fuser *file* – wyświetla wszystkie procesy używające pliku *file*
Ctrl + Z lub ***command* &** lub **bg *process* –** pozostawienie programu uruchomionego w tle
fg *process* – przeniesienie programu na pierwszy plan
jobs – wyświetlenie listy działających zadań
nohup *process* – utworzenie pliku *nohup.out*, do którego przekierowany jest wynik programu *process*
top – pokazuje procesy oraz użycie przez nie pamięci i procesora
nice *priority command* – nadanie priorytetu procesowi
+ *priority* – mniejszy priorytet
- *priority* – większy priorytet (wymagana komenda **sudo**)
sudo renice *priority PID* – zmiana priorytetu procesu o numerze *PID*
ps -eu student | grep wireshark – sprawdzenie PID programu *wireshark*
> - przekierowanie strumienia wyjściowego (**stdout**)
< - przekierowanie strumienia wejściowego (**stdin**)
2> - przekierowanie strumienia błędów (**stderr**)
>> oraz **2> -** dopisanie zawartości strumienia zamiast nadpisania
ls -l *dir* > *file* – przekierowanie zawartości katalogu *dir* do pliku *file*
ls l 2> *file* – przekierowanie błędu do pliku *file*
bash < *file_1* >> *file_2* 2> /dev/null – wykonanie poleceń z pliku *file_1,* wynik operacji będzie w *file_2*, a błędy zostaną zignorowane
ls -l *dir* > *file_1* 2> *file_2* – zawartość katalogu *dir* zostanie przekierowana do pliku *file_1*, błędy do pliku *file_2*
ls *dir* &> *file* lub **ls -l *dir* > *file 2> &l* –** zarówno zawartość katalogu *dir* jak i błędy zostaną przekierowane do pliku *file*
***command* &> /dev/null –** całkowite stracenie wyniku zastosowanej komendy

ls -l *dir* | sort | uniq – potokowanie, zostanie wyświetlona posortowana zawartość katalogu *dir* bez duplikatów
ls -R *dir* – zostanie wyświetlona zawartość katalogu *dir* oraz wszystkich jego podkatalogów (rekurencyjnie)

cat *file* > /dev/print cat – wydrukowanie zawartości pliku *file*
cat *file* > /dev/dsp cat – odegranie pliku dźwiękowego *file*
cat *file_1 file_2 file_3* > *file_4* – przekierowanie zawartości plików *file_1*, *file_2* i *file_3* do pliku *file_4*

head -n 5 *file* – wyświetlenie pierwszych 5 linii pliku *file*
tail -n 10 *file* – wyświetlenie ostatnich 10 linii pliku *file*
more – umożliwia swobodne przewijanie danych do przodu
less – umożliwia swobodne przewijanie danych do przodu i do tyłu

tr – usuwanie tekstu ze źródła lub zastępowanie znaków
tr -d 'std' < *file_1* > *file_2* – wycina wszystkie znaki '*s*', '*t*' i '*d*' z pliku *file_1* i zapisuje do pliku *file_2*
tr ',' '\n' < *file_1* > *file_2* – zamienia wszystkie przecinki w pliku *file_1* na znak nowej linii i zapisuje w pliku *file_2*
echo 'how are you?' | cut -c 1,5,9 – wyświetla pierwszy, piąty i dziewiąty znak zdania '*how are you?*'
cat *file* | cut -c 1-4 – wyświetla tylko od pierwszego do czwartego znaku każdej linii pliku *file*
grep *pattern file* – wyświetla linie z pliku *file* pasujące do wzorca *pattern*
ls -l | grep student – wyświetla tylko te linie w katalogu, w których występuje słowo '*student*'

nano *script* – utworzenie skyptu o nazwie *script* i otworzenie go w edytorze tekstowym *nano*
```
#! /bin/bash #komentarz
echo ”Hello world!”
```
Ctrl + S – zapisanie skryptu
Ctrl + X – wyjście z edytora
bash *script* – wykonanie skryptu, w tym przypadku wyświetlenie '*Hello world!*'
lub
./*script*
lub
chmod +x *script* – nadanie wszystkim uprawnień wykonywania do skryptu *script*
nohup *./script* – utworzenie pliku *nohup.out*, do którego przekierowany jest wynik skryptu *script*
cat nohup.out – wyświetlenie wyniku skryptu
nohup bash -c 'date&&cal' – dodanie do *nohup.out* daty i kalendarza
echo ”*text*” – wyświetlenie tekstu
-n - nie jest wysyłany znak nowej linii
```
echo -n ”Hello”
echo ”world” –
```
skrypt wyświetli *Helloworld*
-e - włączona jest interpretacja znaków specjalnych (*a*, *b*, *c*, *f*, *n*, *r*, *t*, *v*, \\, *xnnn*)
read *variable* – odczytanie zmiennej *variable* z klawiatury (zmienna **programowa**)
-p – pokaże znak zachęty bez kończącego znaku nowej linii
```
read -p ”Podaj:” variable
echo ”$variable”
```
-a – podane wartości są przypisywane do zmiennej tablicowej
```
read -a array
echo ”{array[1]}”
```
– wyświetla element podanej tablicy o indeksie *1*
-e – nie trzeba podawać nazwy zmiennej, wiersz trafia do *$REPLY*
```
read -e
echo ”$REPLY”
```
echo ”$*variable*” – wyświetlenie zmiennej *variable*
dir=`pwd` lub dir=$(pwd)
```
echo ”$dir”
```
– wykonanie polecenia *pwd* i wyświetlenie katalogu, z którego korzysta użytkownik (odwrotne apostrofy)
Zmienne środowiskowe:
$HOME – ścieżka do katalogu domowego
$USER – login użytkownika

$HOSTNAME – nazwa hosta, na który zalogowany jest użytkownik
$OSTYPE – nazwa systemu operacyjnego
echo ”$HOME”
printenv | more – wyświetla wszystkie zmienne środowiskowe
echo ”$0” **–** wyświetla nazwę skryptu (**zmienna specjalna**)
$1…$9 – parametry przekazane do skryptu przez użytkownika
echo ”$1”
bash *script* hello – wyświetla *hello*
echo ”$2 $4”
bash *script* my name is john – wyświetla *name john*
$@ - wyświetla wszystkie podane parametry
$? – wyświetla kod powrotu ostatnio podanego polecenia
$$ - wyświetla *PID* procesu bieżącej powłoki

array=(element_1 element_2 element_3) – zadeklarowanie tablicy
echo ${array[0]} lub **echo ${array} –** wyświetla pierwszy element tablicy
echo ${array[*]} lub **echo ${array[@]}** – wyświetla całą tablicę
echo ${#array[0]} – wyświetla długość pierwszego elementu tablicy
echo ${#array[@]} – wyświetla liczbę elementów tablicy
array[1]=element_4 – zmienia drugi element tablicy na *element_4*
array[3]=element_5 – dodaje *element_5* jako czwarty element tablicy
unset *array*[*index*] – usuwa element z tablicy
unset array[*] – usuwa całą tablicę

Słowa zastrzeżone:
!, case, do, done, elif, else, esac, fi, for, function, if, in, select, then, until, while, {, }, time, [,], test

“ ” – dowolny tekst i zmienne (znaki specjalne: **$**, ****, ` `)
‘ ‘ – łańcuch tekstowy (nie interpretuje znaków specjalnych)
` ` - cytowanie polecenia
x=`ls -la $pwd`
echo $x
\ - wyłącza interpretację znaków specjalnych
echo ”$HOME” – wyświetla katalog domowy użytkownika
echo \$HOME – wyświetla napis *$HOME*

expr – wykonuje obliczenie i wyświetla wynik
expr 2 + 2 – wyświetla 4
expr 1 == 2 – wyświetla 0
Wartość **1** oznacza prawdę, a **0** fałsz
a=2
a=`expr $a + 1`
echo $a **–** wyświetla *3*
Operatory, które są jednocześnie znakami specjalnymi dla powłoki (np. **<** lub *****) należy poprzedzić znakiem ****
$((…)) – alternatywny zapis dostępny dzięki powłoce *bash*
echo $(((12+43)/3)) – wyświetla *18*

test – sprawdza podane warunki
a=1
test $a -eq 1
echo $? **–** wyświetla 0
Wartość **0** oznacza prawdę, a **1** fałsz
Przykładowe operatory:
-a – plik istnieje
= – wyrażenia są równe

!= – wyrażenia są różne
-lt – mniejsze niż
-gt – większe niż
-ge – większe lub równe niż
-le – mniejsze lub równe niż
-n – wyrażenie ma długość większą niż 0
-z – wyrażenie ma zerową długość
-d – wyrażenie istnieje i jest katalogiem
-f – wyrażenie istnieje i jest plikiem
-r – można czytać plik
-w – można zapisywać do pliku
-x – można wykonać plik
***file_1* -nt *file_2* –** plik *file_1* jest nowszy od pliku *file_2*
***file_1* -ot *file_2* –** plik *file_1* jest starszy od pliku *file_2*
[*expression_1 operator expression_2]* - alternatywny sposób zapisu warunków

```
a=1
[ $a -eq 1 ]
echo $?
```

– wyświetla 0
((*expression*)) – alternatywny sposób zapisu warunków dostępny dzięki powłoce *bash*

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

– wyświetla 1

If *condition*
then
command
fi

If *condition*
then
command_1
else
command_2
fi

If *condition_1*
then
command_1
elif *condition_2*
then
command_2
…
else
command_3
fi

```
if [ -e ~/.bashrc ]
then
echo “Masz plik .bashrc”
else
echo “Nie masz pliku .bashrc”
fi
```

Ta sama instrukcja zapisana w jednej linii:

```
if [ -e ~/.bashrc ]; then echo ”Masz plik .bashrc”; else echo “Nie masz pliku .bashrc”; fi
```

case *variable* in
“*condition_1*”) *command_1* ;;
“*condition_2*”) *command_2* ;;
“*condition_3*”) *command_3* ;;
…
) *default_command
esac

```
echo “Podaj cyfrę dnia tygodnia:”
read day
case “$day” in
“1”) echo “Poniedziałek” ;;
“2”) echo “Wtorek” ;;
…
```

```
*) echo “Nic nie wybrałeś”
esac
```
for *variable* in *list*
do
command
done
```
for x in jeden dwa trzy
do
echo $x
done
```
– wyświetla *jeden dwa trzy* w osobnych liniach

Podobna instrukcja z zachowaniem składni zbliżonej do języka C umożliwiona dzięki powłoce *bash*:
```
for ((x=1;x<4;x++)); do echo $x; done;
```
select *variable* in *list*
do
command
done

Pętla *select* generuje ponumerowane menu z listy *list*, gdzie każdej pozycji odpowiada kolejna liczby od 1 wzwyż Użytkownik może podawać liczby, a skrypt wykona polecenie odpowiedniej pozycji z listy, pętla działa dopóki nie odczyta EOF (*End Of File*), który użytkownik może podać za pomocą **CTRL+D**
```
select myselection in fred wilma pebbles barney betty
do
case $myselection in
fred) echo ”Fred was the selection” ;;
wilma) echo ”Wilma was the seleciton” ;;
pebbles) echo ”Pebbles was the selection” ;;
barney) echo ”Barney was the selection” ;;
betty) echo ”Betty was the selection” ;;
esac
done
```

while *condition*
do
command
done
```
x=1
while [ $x -le 10 ]; do
echo $x
x=$[x + 1]
done
```

until *condition*
do
condition
done
```
x=1
until [ $x -ge 10 ]; do
echo $x
x=$[x + 1]
done
```

break – kończy działanie pętli, w której się znajduje
```
for x in jeden dwa trzy
do
echo $x
if [ ”$x” = ”dwa” ]
then
break
fi
done
```
– wyświetla *jeden*

continue – wymusza przejście do kolejnej iteracji pętli
```
for x in jeden dwa trzy
do
if [ ”$x” = ”dwa” ]
then
continue
fi
echo $x
done
```
– wyświetla *jeden* i *trzy*

function *function_name*
{
command
}
Funkcje można umieścić w osobnym pliku nagłówkowym:
nano functions
```
#! /bin/bash
function fun1
{
echo ”a”
}
```
Użycie funkcji w osobnym skrypcie:
```
. functions
fun1
```
– wyświetla *a*

sed – edytor do filtrowania informacji, nie modyfikuje pliku, tylko wysyła wynik na standardowe wyjście
sed ‘s/ala/Ala/’ *file_1* > *file_2* – szuka w pliku *file_1* słowa *ala*, zamienia je na *Ala* i przekazuje wynik do pliku *file_2*
-n – nie wyświetla wynikowych wierszy
sed -n ‘s/ala/Ala/’ *file* – komenda wykona swoje działanie, ale nie wyświetli wynikowego tekstu
-e – wymagane przed każdym poleceniem jeśli jest ich kilka
sed -e ‘s/ala/Ala/’ -e ‘s/kota/psa/’ *file* – jeśli początkowy tekst to *ala ma kota* to po zmianie będzie *Ala ma psa*
-f – wczytuje polecenia z innego pliku
sed -f *file_1 file_2* – jeśli w pliku *file_1* są polecenia to zostaną one zastosowane do pliku *file_2*, a wynik będzie wyświetlony na ekranie
Polecenia programu *sed*:
a *text* – dopisuje tekst na początku wiersza
b *label* – przejdź do polecenia zaczynającego się **:*label***
c *text* – zastępuje wiersze tekstem
d – usuwa wiersze
p – wyświetla wiersze
g – zastępuje bieżący wiersz zawartością bufora
h – kopiuje wiersz do bufora
i *text* – wstawia tekst przed wybranymi wierszami
r *file* – wysyła na wyjście plik *file*
s/*expression*/*text*/*flag* – zastępuje ciąg znaków odpowiadający *expression* na tekst
Znaczniki:
brak – zastępuje pierwsze wystąpienie w wierszu
g – zastępuje wszystkie wystąpienia wzorca
p – wyświetla wiersz po wykonaniu zastąpienia
w – zapisuje wiersz do pliku po wykonaniu zastąpienia
n – zastępuje n-te wystąpienie wzorca
& - odwołanie do *expression*
sed ‘s/.*/Greeting: &/’ *file* – jeśli w pliku *file* jest tekst *Hello world!*, to na wyjściu będzie *Greeting: Hello world!*
t *label* – przechodzi do wiersza gdzie jest *label*
w *file* – zapisuje bieżący wiersz do pliku *file*
y/*abc*/*xyz* – zastępuje znaki *abc* odpowiednio znakami *xyz*
= – wypisuje numer wiersza na wyjściu
:*label* – etykieta
‘{ *command_1*; *command_2*; }’ – wykonuje kilka poleceń na raz
!*command* – wykonaj polecenie dla wierszy, które nie odpowiadają wzorcowi

nano skrypt

```
#! /bin/sed -f
s/a/A/g; s/e/E/g
```

chmod +x skrypt

./skrypt *file* – zamienia litery *a* na *A* i litery *e* na *E* tekstu w pliku *file*

sed '1,5 s/a/A/g' *file_1* > *file_2* – zamienia litery *a* na *A* pliku *file_1*, ale tylko w wierszach od 1 do 5, wynik przekazuje do pliku *file_2*

sed '/ala/,/kota/ s/a/A/g' *file* – robi to samo, tylko, że zaczynając na wierszu zawierającym tekst *ala*, a kończąc na tym, który zawiera tekst *kota*

. – dopasuj jeden dowolny znak

$ - dopasuj poprzedzające wyrażenie do końca wiersza

^ - dopasuj poprzedzające wyrażenie do początku wiersza

*** -** dopasuj zero lub więcej wystąpień poprzedzającego znaku

\ - pomija specjalne znaczenie operatora (np. *****)

[] – dopasuj dowolny znak ujęty w nawiasach

[-] – dopasuj znaki z przedziału

[^] – dopasuj znak, którego nie ma w nawiasach

grep 'Ala' *file* – znajduje wyraz *Ala*

grep 'A.' *file* – znajduje dwuliterowe wyrazy zaczynające się na *A*

grep 'kota$' *file* – znajduje linie kończące się wyrazem *kota*

grep '^Ala' *file* – znajduje linie zaczynające się wyrazem *Ala*

grep 'A*a' *file* – znajduje wszystkie wyrazy zaczynające się na *A* i kończące się na *a*

grep '[0-9][0-9]*' *file* – znajduje dowolny ciąg cyfr

Można zapamiętać tekst w pierwszym argumencie aby go użyć w innym argumencie tej samej komendy

\(*text*\) – zostanie zapamiętany *text*

\1 – aby użyć zapamiętanego tekstu

echo Ala ma kota i papuge | sed 's/\(Ala ma\) [^]*/\1 psa/' – wynikiem komendy będzie *Ala ma psa i papuge*

awk – język do przeszukiwania wzorców i przetwarzania tekstów

BEGIN {*command*} – polecenie jest wykonywane przed rozpoczęciem przetwarzania pliku

END {*command*} – polecenie jest wykonywane po zakończeniu przetwarzania pliku

print – wyświetla informacje

awk 'BEGIN {print "poczatek"}; {print $0}; END {print "koniec"}' *file* – wyświetla *początek* zawartość pliku *file koniec*

To samo w formie skryptu:

```
#! /usr/bin/awk -f
BEGIN {print "początek"}
{print $0}
END {print "koniec"}
```

Po dodaniu praw wykonywania skrypt wywołujemy za pomocą ***./script file***

Zmienne w *awk* są automatycznie przypisywane podczas odczytu danych; plik jest dzielony podanym separatorem

awk -F: '{print $1, $3}' *file* – wyświetla pierwsze i trzecie pole z pliku *file*, które są oddzielone separatorem *:*

$0 – oznacza cały wiersz

-F – określa separator

FS – zmienia separator kolejnych pól, domyślnie spacja

FS="\t+" – zmiana separatora na tabulator, który występuje co najmniej raz

RS – zmienia separator kolejnych rekordów (linii), domyślnie znak nowej linii

RS=":" – zmiana separatora na dwukropek

NR – zmienna oznaczająca numer aktualnie przetwarzanej linii

{ print NR, "> ", $0 } – wyświetli numer linii oraz *>* przed każdym rekordem

```
{x = 3}
```

– własna zdefiniowana zmienna

```
{ zm[1] = 10
zm[2] = 20
print zm[2] }
```

– zmienne tablicowe (indeksowane od 1)

awk '{ print "Wpisano: ", $0 }' – wyświetla liczby, które podaje użytkownik dopóki nie zakończy programu (**CTRL+D**)
***/expression/* -** wyrażenie regularne, na podstawie, którego można wyszukiwać dane
awk '/ala/ { print $0 }' *file* – wyświetla linie pliku *file*, w których jest słowo *ala*
awk '$1 == "Chapter1" { print $0 }' *file* – wyświetla linie, których pierwszym polem jest *Chapter1*
Możliwe operatory to: ==, <, <=, >, >=, !=, ~, !~
~ – zwraca prawdę jeżeli lewy operand zawiera wyrażenie regularne zapisane w drugim operandzie
***pattern_1* && *pattern_2* –** iloczyn logiczny (i, and)
***pattern_1* || *pattern_2* –** suma logiczna (lub, or)
***pattern_1*, *pattern_2* –** zakres, prawdziwy dla wszystkich linii znajdujących się pomiędzy wyrażeniami
! *pattern* – operator NOT
awk '($0 !~ /^S/) && ($2>100) {print $0}' *file* – wyświetla linie nie zaczynające się na *S* i mające w drugim polu liczbę większą od 100
Operatory matematyczne: +, -, *, /, ^, % oraz większość operatorów z języka C (+=, -=, ++, --, …)

```
BEGIN { a = 0 }
{ a = a + $1
print "Dodaje liczbe ", $1
}
END { print "Suma: ", a }
```

while (*condition*) {
command
}

```
{
count=10
while (count != 0) {
print count
count--
}
}
```

do {
command
} while (*condition)*

```
{
count=1
do {
print count
}  while (count != 1)
}
```

– kod zostanie wykonany dla każdej linii pliku

for (*variable* in *list)* {
command
}
lub tak jak w języku C:

```
{
for (x=1;x <= 4;x++) {
print x
}
}
```