Systemy Operacyjne - Linux - Instrukcje 3 & 4 - Skrypty

3.1 Edytor vi

Vi jest najpopularniejszym edytorem tekstu w systemach uniksowych. Można go znaleźć praktycznie w każdym systemie. W systemie Linux (także emulatorze dostępnym tutaj) dostępny jest także edytor vim (vi improved). Po dokumencie można poruszać się za pomocą strzałek jak również klawiszy (k ↑, j ↓, h ←, l →).

→ Uruchomienie edytora **vi**

```
vi [nazwa_pliku]
```

Edytor **vi** działa w dwóch trybach:

- **trybie komend** - umożliwia poruszanie się po pliku, zapis, wstawianie itp.,
- **trybie wpisywania** - umożliwia pisanie i przesuwanie się po tekście.

→ Możliwości przejścia do **trybu wpisywania**

```
i - rozpoczęcie wpisywania w miejscu w którym znajduje się kursor
a - rozpoczęcie wpisywania od następnego znaku za kursorem
A - rozpoczęcie wpisywania na końcu wiersza
c - usunięcie wiersza i rozpoczęcie wpisywania
o - wstawienie wiersza pod tym w którym znajduje się kursor i rozpoczęcie wpisywania
x - usunięcie znaku w miejscu kursora
```

→ Możliwości przejścia do **trybu komend**

Wciśnięcie klawisza **Escape**

→ Komendy dotyczące **zapisu/wyjścia**

```
:q - wyjście (działa tylko w przypadku braku zmian w pliku)
:q! - wyjście z pominięciem zmian
:w [nazwa_pliku]- zapis
:wq [nazwa_pliku] - zapis i wyjście
:x [nazwa_pliku] - zapis i wyjście
ZZ - zapis i wyjście
```

Zadanie 3.1

1. Otworzyć edytor vi **nie podając nazwy pliku**, przejść do trybu pisania.
2. Wpisać trzy wiersze o treściach odpowiednio: raz, dwa, trzy.
3. Przejść do trybu komend, wybrać zapis i wyjście podające nazwę 'pierwszy'.
4. Sprawdzić czy plik został utworzony (komenda *ls*) i wyświetlić jego treść (np. *head nazwa_pliku*).

→ Komendy dotyczące **kopiuj/wytnij/wklej**

dd - wycięcie wiersza (przeniesienie do bufora)
dw - wycięcie od miejsca w którym znajduje się kursor do końca słowa wraz ze spacją
de - wycięcie od miejsca w którym znajduje się kursor do końca słowa bez spacji
d$ - wycięcie od kursora do końca wiersza
p - wklejenie zawartości bufora
:n,m w nazwa_pliku - zapis wierszy o numerze od n do m do podanego pliku
:r nazwa_pliku - wstawienie w dokumencie zawartości podanego pliku
:help - pomoc programu **vi**

Zadanie 3.2

1. Otworzyć edytor vi **podając nazwę 'drugi'**, przejść do trybu pisania.
2. Wpisać trzy wiersze o treściach odpowiednio: cztery, piec, szesc.
3. Dodać pierwszą pustą linijkę i wstawić w jej miejsce zawartość pliku 'pierwszy'.
4. Opusić program za pomocą komendy ZZ i wyświetlić zawartość pliku 'drugi').

3.2 Skrypty

Skrypty są to zwykłe pliki tekstowe zawierające m.in. polecenia powłoki. W skryptach mogą znajdować się także instrukcje warunkowe, pętle, działania arytmetyczne. Skrypty można uruchamiać na **trzy** sposoby:

- Poprzez **określenie powłoki** w której ma on zostać uruchomiony na przykład *sh skryptparametry*,
- Poprzez **nadanie praw do wykonywania** pliku i podaniu nazwy pliku,
- Poprzedzając nazwę kryptu znaku **kropki i spacją**: . *skrypt parametry*.

W skryptach można używać zmiennych, ich deklaracja oraz wykorzystywanie przedstawia się następująco:

→ Wykorzystywanie zmiennych

```
zmienna=5
let suma=$zmienna+6
echo $suma
```

Należy zwrócić uwagę na użycie słowa kluczowego **let** podczas wykonywania operacji na zmiennych, oraz wykorzystaniu znaku **$** podczas odwoływania się do zmiennych. Do kolejnych **parametrów skryptów** można odwołać się w sposób następujący: $0, $1, ..., ${10}, ${11}, przy czym **$0 zawiera nazwę** wywołanego skryptu. Możliwe jest także odwołanie się do wszystkich parametrów skryptu **$@**.

Zadanie 3.3

1. Za pomocą edytora **vi** utworzyć plik 'skrypt1'.
2. Napisać skrypt sumujący dwa podane parametry i wyświetlający wynik.
3. Zapisać skrypt i uruchomić za pomocą polecenia . *skrypt*1 11 15 **lub** *sh skrypt*1 11 15.
4. Spradzić działanie skryptu bez podania parametrów oraz podając parametry tekstowe.

3.3 Instrukcje warunkowe

Do sprawdzania warunków służy instrukcja if. Jej format wygląda następująco:

→ Budowa instrukcji **if**

```
if [ warunki ]
then
        polecenia
else
        polecenia
elif [ warunki ]
then
        polecenia
fi
```

Należy zwrócić uwagę na brak wystąpienia słowa **then** po słowie kluczowych **else**.

→ Warunki logiczne

```
[ -d plik ] - plik istnieje i jest katalogiem
[ -f plik ] - plik istnieje i plikiem zwykłym
[ -r plik ] - plik istnieje i mamy do niego prawo czytania
[ -w plik ] - plik istnieje i mamy do niego prawo pisania
[ -x plik ] - plik istnieje i mamy do niego prawo wykonywania
[ -s plik ] - plik istnieje i mam rozmiar większy niż zero bajtów
[ -z s1 ] - długość łańcucha s1 jest zerowa
[ -n s1 ] - długość łańcucha s1 nie jest zerowa
[ s1 = s2 ] - łańcuchy s1 i s2 są identyczne
[ s1 != s2 ] - łańcuchy s1 i s2 nie są identyczne
[ s1 ] - łańcuch s1 nie jest pusty
```

→ Przykład - usunięcie pliku podanego jako parametr

```
if [ -f $1 -a $2 = 'usun' ] ; then
        rm $1
else
        echo Plik nie istnieje lub nie znana komenda
fi
```

Należy zwrócić uwagę na **znak spacji** pomiędzy nawiasami kwadratowymi a warunkiem, wykorzystanie **operatora -a** oraz wykorzystanie **średnika** w skróconym zapisie warunków (then w jednej linii z warunkiem).

Zadanie 3.4

1. Napisać skrypt sprawdzający czy podano trzy parametry skryptu, jeżeli tak to wyświetlić ich sumę.
2. Zapisać i przetestować utworzony skrypt.

Innym rodzajem instrukcji warunkowej jest instrukcja case. Jej format to:

→ Budowa instrukcji **case**

```
case $zmienna in
        wartosc1) polecenia ;;
        wartosc2) polecenia ;;
        wartosc3) polecenia ;;
        *) polecenia_domyślne
esac
```

Należy zwrócić uwagę na dwukrotne wykorzystanie znaku średnika.

→ Przykład wykorzystania instrukcji **case**

```
echo "Podaj liczbe 1-4: "
read pory_roku
case $pory_roku in
        "1") echo "Wiosna" ;;
        "2") echo "Lato" ;;
        "3") echo "Jesien" ;;
        "4") echo "Zima" ;;
        *) echo "Nieprawidlowa wartosc"
esac
```

Zadanie 3.5

1. Napisać skrypt wczytujący od użytkownika dwie liczby (polecenie *read*) oraz typ działania do wykonania (1 - dodawanie, 2 - odejmowanie, 3 - mnożenie, 4 - dzielenie).
2. Skrypt powinien (w zależności od numeru instrukcji) wyświetlić odpowiedni wynik.

Zadanie 3.6 *

1. Załóżmy, że w systemie nie ma polecenia mv. Napisz skrypt, który umożliwi zastąpienie tego polecenia. W tym celu wykorzystaj polecenie *cp*.

Zadanie 3.7 *

1. Napisz skrypt o nazwie 'czas'.
2. Skrypt ten powinien wyświetlić datę w formacie RRRR-MM-DD (polecenie *data*).
3. W zależności od numeru miesiąca powinna wyświetlać się odpowiednia nazwa pory roku.
4. Zmień atrybuty skryptu tak, aby możliwe było jego wykonanie za pomocą *./czas*.

3.4 Funkcje

Podobnie jak w programach w skryptach można definiować funkcje, które mogą zawierać często powtarzające się operacje. Format definiowania funkcji wygląda następująco:

→ Deklaracja funkcji

```
function nazwa
{
        polecenia
        return wartosc
}
```

Do funkcji można przekazywać parametry podobnie jak przy wywołaniu skryptu. Wewnątrz funkcji widoczne są one jako $1, $2 itd. **Wartośc zwracanych przez funkcje nie można** przypisać bezpośrednio do zmiennych. Są one jednak widoczne jako $? co ilustruje następujący przykład:

→ Przykład deklaracji i wywołania funkcji

```
function suma
{
        let wynik=$1+$2
        return $wynik
}
suma 5 4
echo Wynik: $?
```

→ Inny przykład deklaracji i wywołania funkcji

```
wynik=0
function suma
{
        let wynik=$1+$2
}
suma 5 4
echo Wynik: $wynik
```

Zadanie 3.8

1. Napisz funkcję zwracającą minimum z dwóch podanych parametrów

→ Wczytywanie danych

```
read - wczytuje dane do zmiennej $REPLY
read zmienna - wczytuje dane do zmiennej $zmienna
read < plik - wczytuje dane z pliku
```

3.5 Pętle

Pętle mają podobną składnię jak w popularnych językach programowania:

→ Pętla **while**

```
while [ warunki ]
do
        polecenia
done
```

→ Pętla **for**

```
for i in zbior_wartosci
do
        polecenia
done
```

Zbiorem wartości może być:

- Zdefiniowana przez użytkownika lista wartości, np: 1 2 4 lub ciągi tekstowe: kot pies ptak
- Lista plików i folderów, np: /usr/local/* lub *
- Lista elementów zwróconych przez polecenie zewnętrzne, np: ‘seq 1 2 10‘ (**należy użyć apostrofu znajdującego się pod znakiem tyldy**)

→ Przykład pętli sprawdzającej pliki w katalogu bieżącym

```
for i in * ; do
        if [ -f $i ] ; then
                echo $i jest plikiem
        elif [ -d $i ] ; then
                echo $i jest katalogiem
        else
                echo $i jest czyms innym
        fi
done
```

Zadanie 3.9

1. Zapoznaj się z pomocą polecenia seq (*man seq* lub *seq −−help*).
2. Wyświetl za pomocą seq liczby nieparzyste od 1 do 99.
3. Napisz skrypt z pętlą przechodzącą po tych liczbach i tworzącą katalogi o nazwie ’kat$liczba’.

3.6 Pętla wyboru

Do tworzenia menu przydatna może być pętla select, która działa dopóki nie zostanie wywołane polecenie break lub return. Tworzy ona ponumerowaną listę. Jej format to:

→ Pętla **select**

```
select i in opcje
do
        polecenia
done
```

Pętla ta generuje listę wyboru i **oczekuje** na wybór użytkownika, np:

→ Pętla **select** - przykład nr. 1

```
select i in Raz Dwa Trzy ; do
        echo $i
done
```

Wyświetli: 1) Raz, 2) Dwa, 3) Trzy i będzie oczekiwać na podanie przez użytkownika jednej z wartości 1, 2 lub 3. W przypadku podania wartości nastąpi wykonanie pętli i **ponowne** oczekiwanie na wybór użytkownika (taka pętla będzie zatem działać teoretycznie w nieskończoność). W związku z powyższym z pętlami select często łączy się polecenia **case** w których jedną z opcji jest zakończenie pętli:

→ Pętla **select** z wykorzystaniem instrukcji **case**

```
select i in Kopiuj Przenies Koniec ; do
        case $i in
                "Kopiuj") cp plik plik1 ;;
                "Przenies") mv plik plik2 ;;
                "Koniec") exit ;;
        esac
done
```

Zadanie 3.10

1. Napisz skrypt wykorzystujący pętle select (**bez instrukcji case**)) zawierającą 12 różnych opcji.
2. Pętla powinna się zakończyć gdy użytkownik wybierze opcję nr. 7.

Zadanie 3.11 *

1. Zapisz w dowolnym pliku kilka wyrazów.
2. Napisz pętle wyświetlającą kolejno wyrazy z tego pliku.
3. Po każdym wyrazie pętla powinna odczekać sekundę (polecenie *sleep*).

Zadanie 3.12 *

1. Napisz skrypt wykorzystujący pętle select (**bez instrukcji case**)) w której opcje wyboru będą zawierać polecenie 'Koniec' oraz listę plików z danego folderu.
2. W przypadku wybrania polecenia koniec pętla powinna zakończyć działanie.
3. W przypadku wybrania pliku powinna zostać utworzona kopia pliku o rozszerzeniu .cpy.

3.7 Edytor JOE

Joe jest kolejnym popularnym edytorem dostępnym w wielu systemach Linux. Uruchamiamy go pisząc joe w linii poleceń. Dla edytora tego dostępna jest pomoc man oraz pomoc w programie, którą można przejrzeć wciskając CTRL+k h. Program umożliwia operacje na blokach tekstu, wyrazach liniach, wstawianie plików itp. Podstawowe klawisze uzywane w programie:

- **Poruszanie się po tekście:**
  CTRL+Z – poprzedni wyraz, CTRL+X – następny wyraz, CTRL+A – początek linii, CTRL+E – koniec linii, CTRL+U – poprzedni ekran, CTRL+V – następny ekran, CTRL+KU – początek tekstu, CTRL+KV – koniec tekstu,
- **Operacje na blokach tekstu:**
  CTRL+KB – początek bloku, CTRL+KK – koniec bloku, CTRL+KM – przeniesienie bloku, CTRL+KC – skopiowanie bloku, CTRL+KW – zapisanie bloku do pliku, CTRL+KY – usunięcie bloku,
- **Usuwanie tekstu:**
  CTRL+D – usunięcie znaku, CTRL+Y – usunięcie linii, CTRL+W – usunięcie słowa na prawo od kursora, CTRL+O – usunięcie słowa na lewo od kursora, CTRL+J – usuniecie tekstu do końca linii na prawo od kursora,
- **Operacje na plikach:**
  CTRL+KD – zapisanie pliku, CTRL+KR – wstawienie pliku, CTRL+KE – edycja pliku,
- **Wyjście z programu:**
  CTRL+KX – wyjście z zapisem, CTRL+C – wyjście bez zapisu.

3.8 Zadania utrwalające

- Wyprowadź do pliku lista_etc zawartość katalogu /etc. Wstaw ten plik na początek pliku koniec z poprzedniego ćwiczenia używając vi.
- Napisz skrypt wyświetlający nazwę skryptu oraz cztery jego parametry.
- Stwórz skrypt wyświetlający grupami pliki do których mamy prawo do odczytu, zapisu i wykonania.
- Napisz skrypt, który po 1 minucie od uruchomienia wyświetli: "Minela jedna minuta" (polecenie at).
- Znajdź błędy w skrypcie:

```
if [-d plik ]; then
        mv plik plik2
elif [ -f plik3 ] then
        mw plik3 plik4
fi
```

- Stwórz skrypt, który wyszuka nazwę procesu podaną jako parametr wśród wszystkich procesów obecnych w systemie i wyświetli informacje dotyczące tego procesu.
- W pliku .bash_logout umieść polecenie usuwające wszystkie pliki z katalogu domowego. Wyloguj się i zaloguj ponownie. Jaki efekt?
- Napisz skrypt przeszukujący plik .bash_history i wyświetlający wszystkie wystąpienia polecenia cat.
- Napisz skrypt wyświetlający same nazwy użytkowników aktualnie zalogowanych w systemie (pol. *who*).
- Napisz skrypt wyświetlający rozmiar pliku, podanego jako parametr, w bajtach (polecenie *wc*).