# 4. SKRYPTY POWŁOKI – PODSTAWOWE INFORMACJE

## 4.1. Charakterystyka skryptów

Cechy skryptów:

- skrypty to zestawy poleceń zapisane w plikach tekstowych,
- wykonanie skryptu polega na realizacji poleceń zapisanych w kolejnych wierszach pliku,
- wiersz, który zaczyna się od znaku „#" traktowany jest jako komentarz i nie jest wykonywany,
- pierwsza linia skryptu powinna mówić o tym jakiego dla jakiego rodzaju powłoki został napisany skrypt (np. #!/bin/bash),
- jeżeli napisany skrypt ma być programem powłoki, to musi mieć nadane prawo do wykonania,
- rozszerzenie pliku zawierającego skrypt powłoki to *sh* (chociaż może go nie być).

Przykładowy skrypt powłoki o nazwie *kopia.sh*:

```
echo ″Mój katalog domowy″ > wykaz.txt
date >> wykaz.txt
ls -al >> wykaz.txt
echo ″Koniec″ >> wykaz.txt
```

*powyższy skrypt utworzy plik „wykaz.txt" (lub zastąpi jego zawartość, jeżeli taki plik istnieje), zapisze w nim tekst „Mój katalog domowy", bieżącą datę, zawartość bieżącego katalogu oraz tekst „Koniec"*



Skrypt powłoki można uruchomić poprzedzając jego nazwę kropką i spacją („. ") lub przekierowując jego zawartość do polecenia *bash*, np.

```
$ . kopia.sh
$ bash < kopia.sh
```

Skrypt powłoki ma dla użytkownika status *programu powłoki*, wtedy kiedy użytkownik ma nadane prawo do jego wykonywania (będzie to wyjaśnione dalej), np.

```
$ chmod u+x kopia.sh
```

Uruchamianie programu jest możliwe poprzez podanie jego nazwy, np.

```
$ kopia.sh
```

Taki sposób uruchomienia jest możliwy tylko wtedy, kiedy zmienna PATH zawiera ścieżkę dostępu do katalogu bieżącego. Jeżeli PATH nie przeszukuje bieżącego katalogu, to taki skrypt należy uruchamiać poprzedzając jego nazwę względną ścieżką dostępu (czyli „./"), tzn.

```
$ ./kopia.sh
```

## 4.2. Parametry skryptu

*Parametr* skryptu to informacja przekazywana do skryptu w momencie jego uruchamiania, np.

```
$ skrypt.sh parametr1 parametr2
```

Największą zaletą tworzenia skryptów z parametrami jest możliwość podstawiania w ich miejsce różnych wartości. Przykładowo, skrypt o nazwie *dopisz.sh*, który do pliku o nazwie podanej jako pierwszy parametr dopisuje zawartość pliku o nazwie podanej jako drugi parametr, może być wykonany z takimi parametrami:

```
$ dopisz.sh list1.txt list2.txt
```

co oznacza, że dopisze do zawartości pliku *list1.txt* zawartości pliku *list2.txt*. Można go uruchomić także z innymi parametrami, np.

```
$ dopisz.sh rozdzial3.txt rozdzial4.txt
```

co oznacza, że dopisze do pliku *rozdzial3.txt* zawartości pliku *rozdzial4.txt*.
Uogólniając, można powiedzieć, że wywołanie skryptu ma następującą postać:

```
$ dopisz.sh parametr1 parametr2
```

gdzie *parametr1* oraz *parametr2* są nazwami plików.
Parametry podawane przy uruchomieniu skryptu nazywane są *parametrami aktualnymi*.

Aby skrypt mógł wykonać postawione zadanie musi mieć możliwość odwołania się do wartości parametrów podanych przy jego uruchamianiu, tzn. pisząc skrypt jego autor musi podać, w którym miejscu skrypt odwołuje się do parametru pierwszego, w którym do parametru drugiego, itd.

Możliwość odwołania się do wartości parametrów daje wykorzystanie symboli, w miejsce których w czasie wykonywania skryptu wstawiane są wartości parametrów aktualnych:

- $1 - symbolizuje pierwszy parametr aktualny skryptu (tzn. w czasie wykonania skryptu w jego miejsce wstawiana jest wartość pierwszego parametru aktualnego), $2 - symbolizuje drugi parametr, itd. aż do $9,
- $0 - symbolizuje nazwę skryptu,
- $# - symbolizuje liczbę parametrów aktualnych,



- $* - symbolizuje łącznie wszystkie parametry aktualne, z którymi został uruchomiony skrypt.

Parametry występujące w tekście skryptu i symbolizujące parametry aktualne (tzn. $1, $2, …, $9) nazywane są *parametrami formalnymi*.

Przykładowo, wspomniany wyżej skrypt *dopisz.sh* ma następującą postać:

```
cat $2 >> $1
```

Jego wywołanie poprzez podanie polecenia:

```
$ dopisz.sh list1.txt list2.txt
```

będzie realizowane jako:

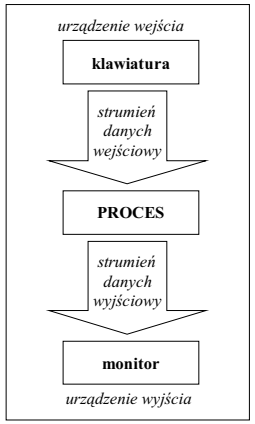
```
cat list2.txt >> list1.txt
```

# 5. OPERACJE WEJŚCIA - WYJŚCIA

## 5.1. Strumień danych oraz urządzenia wejścia i wyjścia

- dostarczanie danych do procesów oraz generowanie danych przez procesy polega na przesyłaniu tzw. *strumieni danych*,
- dane do procesu trafiają, jako *strumień danych wejściowych* wygenerowany przez *urządzenie wejścia*,
- z procesu dane wychodzą, jako wygenerowany przez proces *strumień danych wyjściowych* skierowany do *urządzenia wyjścia*,
- urządzenia wejścia to np. klawiatura, plik; urządzenia wyjścia to np. monitor, plik, drukarka. W przypadku tych urządzeń rozróżnia się, tzw.
  - *standardowe wejście* - urządzenie, z którego proces domyślnie otrzymuje strumień danych wejściowych (najczęściej klawiatura),
  - *standardowe wyjście* - urządzenie, do którego proces domyślnie kieruje strumień danych wyjściowych (najczęściej monitor).

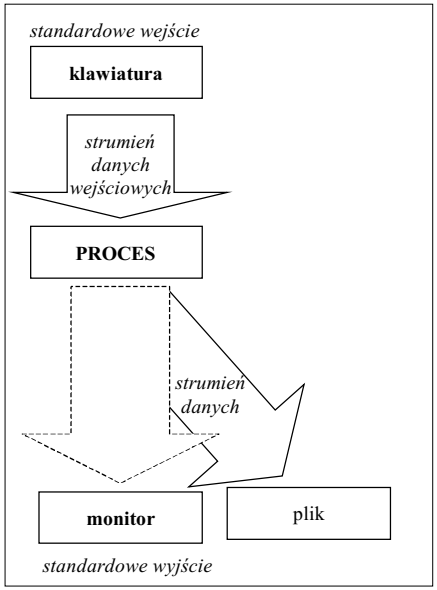





**Rys. 8. Przepływ strumieni danych pomiędzy procesem a urządzeniami wejścia i wyjścia**

## 5.2. Przekierowanie wejścia i wyjścia

- *przekierowanie wejścia* oznacza zmianę urządzenia wejściowego z domyślnego na inne, wskazane przez użytkownika. Służy do tego operator „<";
- *przekierowanie wyjścia* oznacza zmianę urządzenia wyjściowego z domyślnego na inne, wskazane przez użytkownika. Służą do tego operator „>" lub operator „>>". Różnica w działaniu tych operatorów jest widoczna w przypadku przekierowania wyjścia do pliku, tzn.:
  - operator „>" powoduje *zastąpienie* bieżącej zawartości pliku przekierowanym strumieniem danych,
  - operator „>>" powoduje *dopisanie* (na końcu pliku) do zawartości pliku przekierowanego strumienia danych.




**Rys. 9. Przekierowanie wyjścia**

Przykładowe przekierowanie wyjścia:

```
$ ls > spis.txt
$ cat zaproszenie.txt > teksty.txt
$ cat ogloszenie.txt >> teksty.txt
$ sort lista.txt > lista_posortowana.txt
$ echo „Uniwersytet Ekonomiczny” > uczelnia.txt
```

Przykładowe przekierowanie wejścia:

```
$ sort
ola
ala
basia
[Crtl]+[D]
ala
basia
ola
```



*standardowo polecenie sort pobiera z podanego pliku (standardowe wejście polecenia sort) strumień danych i wyświetla go posortowanego wierszami na monitorze (standardowe wyjście polecenia sort). W tym przypadku plik nie został podany, dlatego polecenie sort będzie pobierało strumień danych z klawiatury, aż do naciśnięcia kombinacji [Ctrl]+[D]. Następnie wyświetli je na monitorze posortowane.*

```
$ write nowakj < komunikat.txt
```

### 5.3. Standardowe wyjście błędów

Standardowe wyjście błędów jest to strumień danych, który zostaje wygenerowany kiedy polecenie zakończyło się niepowodzeniem. Do przekierowania standardowego wyjście błędów służy operator „2>”, n p.

```
$ cat mojplik.txt 2> error.txt
```

*- jeżeli mojplik.txt np. nie istnieje to informacja o błędzie zostanie skierowana do pliku error.txt*

```
$ cat mojplik.txt > nowyplik.txt 2>> error.txt
```

*- jeżeli nie zaistnieje błąd to utworzony zostanie tylko nowyplik.txt; jeżeli błąd zaistnieje, to komunikat o błędzie zostanie dopisany do pliku error.txt*

```
$ find / -name "nazwa_pliku" 2> /dev/null
```

*- błędy z tego polecenia nie zostaną wyświetlone*

### 5.4. Identyfikatory standardowych wyjść „&”1 i „&2”

W operacjach przekierowania strumienia:

- standardowe wyjście jest identyfikowane przez „&1”,
- standardowe wyjście błędów jest identyfikowane przez „&2”.

Można używać odpowiednich przekierowań, np. „2>&1”, „>&2”

```
$ cat mojplik.txt > nowyplik.txt 2>&1
```

*- jeżeli mojplik.txt istnieje, to jego zawartość zostanie skierowana do pliku nowyplik.txt, a jeżeli nie istnieje, to komunikat o błędzie zostanie skierowany na standardowe wyjście i ostatecznie do pliku nowyplik.txt*

### 5.5. Tworzenie potoków

*Potok* jest to skierowanie standardowego wyjścia jednego procesu do standardowego wejścia innego procesu. Operatorem tworzenia potoku jest pionowa kreska „|”.

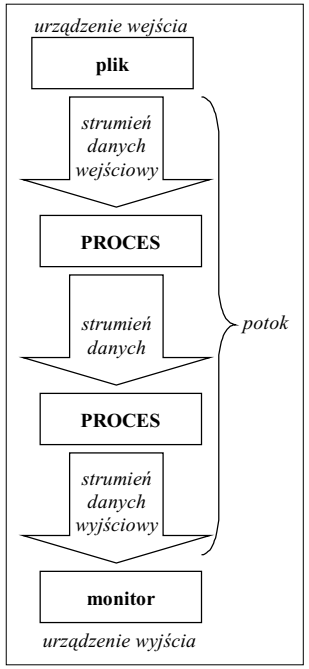
Tworzenie potoku można przedstawić następująco:

*proces* | *proces* | ... | *proces*

Tworzenie potoków można równocześnie łączyć z przekierowaniem wejścia i wyjścia, co pozwala na zaawansowane przetwarzanie strumieni danych., np.

*proces* | *proces* | ... | *proces* > *urządzenie wyjścia*



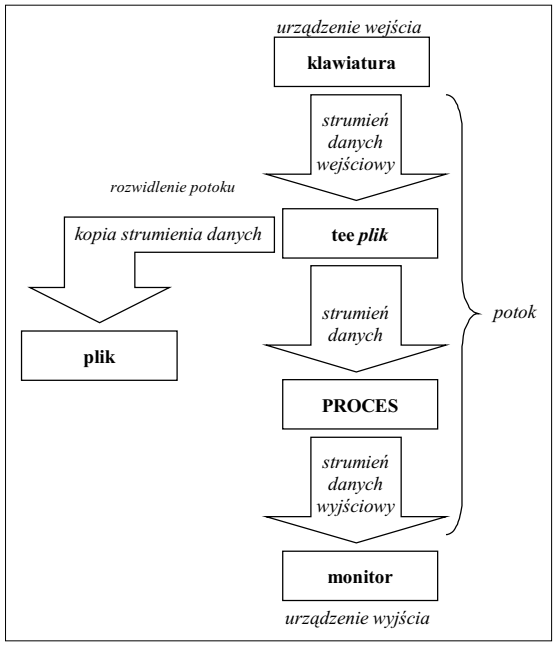



**Rys. 10. Potok danych**

Przykładowe tworzenie potoków:

```
$ ls | sort
$ finger | grep ”Jacek” | sort > lista.txt
```

Potoki mogą być także *rozwidlane*. Do rozwidlania potoku służy polecenie *tee*, które robi kopię strumienia danych i zapisuje ją w pliku, a „oryginalny” strumień danych przepuszcza dalej do standardowego wyjścia lub kolejnego procesu.




**Rys. 11. Rozwidlenie potoku danych**

Przykładowe rozwidlenia potoków:

```
$ sort lista.txt | tee lista_posortowana.txt
$ sort lista.txt | tee pliczek.txt | cat -n
```

### 5.6. Filtry

Filtry to polecenia odczytujące dane (z pliku lub strumienia), wykonujące na nich określone działania i wysyłające wynik na standardowe wyjście.

- **more, less** - powoduje wyświetlanie strumienia - ekran po ekranie, np.

```
$ ls -al | more
$ ls -al | less
```



- **fmt** - powoduje wyświetlanie strumienia danych sformatowanego w wiersze o określonej szerokości (ale bez dzielenia wyrazów), np.

```
$ fmt -10 lista.txt
```

*wyświetlenie pliku lista.txt w wierszach o długości do 10 znaków (chyba, że pojedynczy wyraz jest dłuższy, wtedy pojawi się cały)*

- **tee** - kopiuje standardowe wyjście do pliku, np.

```
$ cat mojplik.txt | tee nowyplik.txt
$ sort lista.txt | tee listaPosortowana.txt
```

- **head** - wyświetla zadaną liczbę pierwszych wierszy pliku (domyślnie 10), np.

```
$ head -3 plik.txt
$ cat plik.txt | head -3
```

- **tail** - wyświetla zadaną liczbę ostatnich wierszy pliku (domyślnie 10), np.

```
$ tail -5 plik.txt
```

- **wc** - wyświetla informację o liczbie wierszy, słów i znaków znajdujących się w pliku, np.

```
$ wc plik.txt
$ cat plik.txt | wc
```

Opcja „-w” zlicza tylko słowa, opcja „-c” zlicza tylko znaki.

- **sort** - sortuje zawartość strumienia, np.

```
$ sort plik.txt
$ cat plik.txt | sort > posortowanyPlik.txt
```

- **tac** – wyświetla zawartość pliku wierszami „od końca” (tzn. ostatnie wiersze wyświetla jako pierwsze), np.

```
$ tac plik.txt
```

- **grep** - wyszukuje wiersze zawierające ciągi znaków zgodne ze wzorcem, np.

```
$ grep „abcd” plik1.txt plik2.txt plik3.txt
$ cat plik1.txt | grep „abcd”
$ finger | grep ”Jacek” | sort
$ cat plik1.txt | grep -E -i „^a”
```

*- opcja „-E” sprawia, że grep interpretuje wyrażenia regularne*

*- opcja „-i” ignoruje wielkość liter.*

- **diff** - porównuje każdy wiersz dwóch plików znak po znaku, np.

```
$ diff plik1.txt plik2.txt
```

- **comm** - porównuje dwa posortowane pliki wiersz po wierszu i standardowo wyświetla wyniki w trzech kolumnach (wiersze występujące tylko w pierwszym pliku, wiersze występujące tylko w drugim pliku, wiersze wspólne dla obu plików). Istnieje możliwość wyłączenia wyświetlania poszczególnych kolumn, np.

```
$ comm lista1.txt lista2.txt  -1 -3
```

*wyświetli wiersze występujące tylko w pliku drugim (pierwsza i trzecia kolumna została pominięta)*



- **cut** - wybiera określone pola lub kolumny z pliku, np.

```
$ cut -f1,3 plik1.txt
```

*- określenie, które pola (1 i 3) mają być pobrane z pliku*

*- uwaga: domyślnym ogranicznikiem pól jest tabulacja*

```
$ cut -f1,3 -d”;” plik1.txt
```

*- definiuje ogranicznik (tutaj średnik), który ma być użyty w pliku*

```
$ cut -c1-4 plik1.txt
```

*określenie, które znaki (od 1 do 4) mają być pobrane z pliku*

- **sed** - edytor strumieniowy, np.

```
$ sed -n „3 p” lista.txt
```

*wyświetlenie (p) tylko (-n) trzeciego wiersza z pliku lista.txt*

```
$ sed „/abc/ d” lista.txt
```

*wyświetlenie pliku lista.txt bez wiersza (d) zawierającego ciąg znaków „abc” (ciąg musi znajdować się pomiędzy znakami /.../)*

- **tr** – zamienia lub kasuje znaki w strumieniu danych, np.

```
$ cat imiona.txt | tr [a-z] [A-Z]
```

*- zamienia wszystkie litery w pliku imiona.txt z małych na duże*

```
$ cat imiona.txt | tr [:lower:] [:upper:]
```

*- tak jak w poprzednim przykładzie*

```
$ cat imiona.txt | tr abc 123
```

*zamienia w pliku imiona.txt odpowiednio literę „a” na cyfrę „1”, „b” na „2”, „c” na „3”*

```
$ cat imiona.txt | tr -d „b” -s „a”
```

*usuwa z pliku imiona.txt wszystkie litery „b” i powtarzające się obok siebie litery „a”*

- **paste** - łączy wiersze z różnych plików (zobacz też polecenie *join*), np.

```
$ paste -d”;” student.txt ocena.txt
```

*połączenie wierszy z dwóch plików i wstawienie pomiędzy nimi średnika*

- **uniq** - eliminuje powtarzające się wiersze wśród danych wejściowych

```
$ uniq -c lista.txt
```

*wyświetla informację (-c) ile razy dany wiersz występuje w pliku*

- **split** – dzieli zawartość pliku (strumienia danych) na porcje (liczbę bajtów, liczbę wierszy) i zapisuje je w oddzielnych plikach

```
$ split lista.txt -l 1
```

*podzieli plik lista.txt na pojedyncze wiersze i zapisze je w oddzielnych plikach*

- **hexdump** – zamienia zawartość pliku (strumienia danych) na kody w systemie szesnastkowym.

- **sum, cksum, sha1sum, md5sum** – tworzy sumy kontrolne (odpowiednio: sumę BSD, sumę CRC, wartość funkcji haszującej SHA1 i MD5) dla plików i strumieni danych.