4. Przeadresowanie wejścia-wyjścia

Programy, które przetwarzają dane i generują jakieś rezultaty, potrzebują kanałów komunikacyjnych, aby pobrać dane wejściowe i przesłać wyniki swojej pracy. Mamy trzy kanały komunikacyjne stdin, stdout, stderr. Kanały te traktowane są przez system operacyjny jako pliki, które w szczególności mogą być urządzeniami zewnętrznymi. System Linux i Unix pozwala otworzyć jednocześnie do 20 plików, które są poddane przetwarzaniu. Każdy z nich ma swój deskryptor, służący jego identyfikacji: stdin to 0, stdout – 1, tsderr -2, pozostałe pliki to zakres od 3-19.

Przykłady

1. Standardowe strumienie danych są używane wtedy, gdy nie jest określony specjalny sposób wejścia i wyjścia:
   `sort`
   polecenie tak zapisane oczekuje danych wejściowych z klawiatury i wynik swojej pracy poda na ekran. Jeśli napiszemy:
   `sort plik`
   to dane wejściowe zostaną pobrane z pliku, a wynik znowu zostanie wypisany na ekranie. Jeśli jednak napiszemy:
   `sort –o plik_out plik`
   to dane wejściowe zostaną pobrane z pliku, a wynik zostanie również umieszczony w pliku.

Omawiane systemy operacyjne oferują mechanizm zwany kierowaniem strumieni lub przeadresowaniem wejścia-wyjścia. Do takich przeadresowań służą operatory przeadresowania.

Operatory przeadresowania	
Konstrukcja	**Czynność**
Proste przekierowania	
`polecenie > plik`	Wysłanie wyniku **`polecenia`** do **`pliku`** (nadpisanie).
`polecenie < plik`	Pobranie z pliku danych wejściowych dla **`polecenia.`**
`polecenie >> plik`	Wysłanie wyniku **`polecenia`** do **`pliku`** (dopisanie).
`polecenie << ogr`	Przekazanie na standardowe wejście danych do ogranicznika **`ogr`**.
Przekierowania z użyciem deskryptorów plików	
`polecenie >&n`	Wysłanie **`polecenia`** do **`pliku`** o deskryptorze **`n`**.
`polecenie m>&n`	Wysłanie **`polecenia`**, który standardowo zostałby przesłany do **`pliku`** o deskryptorze **`m`**, do **`pliku`** o deskryptorze **`n`**.
`polecenie >&-`	Zamknięcie standardowego wyjścia.
`polecenie <&n`	Pobranie danych wejściowych dla **`polecenia`** z **`pliku`** o deskryptorze **`n`**.
`polecenie m<&n`	Pobranie danych wejściowych, które standardowo zostałby pobrane z **`pliku`** o deskryptorze **`m`**, z **`pliku`** o deskryptorze **`n`**.
`polecenie >&-`	Zamknięcie standardowego wejścia.
Przekierowanie wielokrotne	
`polecenie 2> plik`	Przekierowanie standardowego wyjścia błędów do **`pliku`**.
`polecenie 2>&1`	Przekierowanie standardowego wyjścia błędów, jak i standardowego wyjścia do **`pliku`**.
`(polecenie > p1)2>p2`	Przekierowanie standardowego wyjścia błędów do pliku **`p1`**, a standardowego wyjścia do pliku do pliku **`p2`**.

Polecenie cat

Polecenie **`cat`** (skrót od *concatenate* – łączyć) służy do łączenia plików. Choć wiele źródeł wymienia to jako zastosowanie główne, w większości polecenie jest wykorzystywane wyłącznie z powodu drugiej oferowanej funkcji – odczytywania krótkich plików.

`cat -opcje plik`

`-b` -numeruje tylko linie zawierające treść i ignoruje puste

`-n` -numeruje tylko linie (również puste)

Chociaż polecenie **cat** jest łatwe w użyciu, nie nadaje się do plików o długości przekraczającej dwadzieścia linii – i to jego jest główną wadą. Każdy dłuższy plik wymaga przewijania. Co więcej, jeśli plik ma więcej linii, niż zdoła wyświetlić terminal wirtualny, możemy nie zobaczyć początku pliku. Z tego powodu wiele osób woli przeglądać pliki przy pomocy innych poleceń.

Polecenie more

Polecenie **more** jest bardziej skomplikowane od **cat** i lepiej sprawdza się w przypadku plików o długości powyżej dwudziestu linii. Jego główną zaletą jest to, że wyświetla całą informację na ekranie, a więc nie grozi nam, że nie zobaczymy części pliku.

```
more plik
```

Jeżeli podstawowy komunikat nie jest wystarczający, możemy rozszerzyć go przy pomocy opcji **-d**

Gdy wyniki są pokazywane na ekranie, na dole listy pojawia się podświetlona wiadomość –*More* –. Aby przejść do następnej niewidocznej linii, wciskamy klawisz Enter, do następnego ekranu wyników – spację, aby powrócić do wiersza poleceń – q. Podczas przeglądania pliku możemy wprowadzić *numer z klawiatury* by określić liczbę linii przeskoku do przodu i Enter.

Podczas stosowania **more** możemy dojść do wniosku, że ma ono jedną poważną wadę – nie pozwala przewijać tekstu do tyłu, dlatego po obejrzeniu całego tekstu ekran zostaje wyczyszczony, chyba że wyjdziemy z polecenia **more** i uruchomimy je ponownie.

Przykłady

2. Operator **>** pozwala skierować strumień wyjściowy do zwykłego pliku podanego jako parametr. Jeśli wyświetlimy **plik1** za pomocą **cat**, możemy działanie tego polecenia umieścić w pliku **plik2**.
   ```
   cat plik1 > plik2
   ```
   Możemy tez łączyć wyniki:
   ```
   cat plik1 plik2 plik3 > plik123
   ```
   Wydanie powyższego polecenia spowoduje utworzenie **pliku123** jeśli go nie ma i wpisanie do niego danych, lecz jeśli jest dane zostaną nadpisane i możemy stracić wcześniejszą pracę.
   Znaki spacji wokół znaku **>** są opcjonalne i równoważne.
3. Operator **<** oznacza otwarcie pliku jako standardowego wejścia. Polecenie:
   ```
   sort < plik1
   ```
   spowoduje pobieranie znaków z **pliku1** a nie z klawiatury. Wyjście zostanie skierowane na ekran, ponieważ nie podano innej opcji. **Plik1** jest otwierany przez system operacyjny.
   ```
   sort plik1
   ```
   w tym wypadku **plik1** jest otwierany przez samo polecenie **sort**.
   Strumienie można kierować w jednym wierszu poleceń, a miejsce wystąpienia znaków **< >** nie jest ważne:
   ```
   tr “[a-z]” “[A-Z]” < plik1 > plik2
   tr “[a-z]” “[A-Z]” > plik2 < plik1
   < plik1 tr “[a-z]” “[A-Z]” > plik2
   ```
4. Użycie operator **>** wiąże się z ryzykiem skasowania starych danych. Jeśli więc chcemy zachować to co jest w pliku i dołożyć nowe rzeczy należy użyć operatora **>>**
   ```
   cat plik4 >>plik123
   ```
5. Operator **<<[-]org** pozwala na czytanie standardowego wejścia, aż do wystąpienia ogranicznika. Symbol ten wykorzystuje jako wejście dane zawarte wewnątrz polecenia powłoki (skryptu), dzięki czemu nie trzeba tworzyć oddzielnego pliku wejściowego. Opcja **–** powoduje usunięcie z podanego ograniczenia **org** oraz z tekstu wejściowego poprzedzających znaków tabulacji.
   Mamy następujący skrypt:
   ```
   mail $* <<stop
   Napisz do mnie list
   .
   stop
   ```

Słowo stop występujące po **<<** działa jak sygnał końca pliku, wiec wynikiem skryptu będzie wysłanie komunikatu: Napisz do mnie list.

6. Operator **2>** służy do przekierowania standardowego wyjścia błędów.

7. Operator **2>&1** służy do przekierowania standardowego wyjścia błędów oraz standardowego wyjścia do jednego pliku.
   spell raport > raport_sp 2>&1
   Uwaga: najpierw stdout kierowane jest do pliku a dopiero później operatorem **2>&1** stderr łączone jest z stdout.

Zmienna noclobber

Gdy kierujemy strumień wyjściowy do pliku, który już istnieje, to jego wcześniejsza zawartość zostanie skasowana. Aby się to nie stało (gdyż powłoka nie ostrzega, że nadpisujemy plik) powłoka oferuje zmienną **noclobber** w celu ochrony danych istniejących w pliku. Zmienną tą w powłokach Korna i Bash ustawia się za pomocą polecenia **set**:

```
set –o noclobber
```

Aby ponownie zezwolić na nadpisanie należy wpisać:

```
set +o noclobber
```

Jeśli nie chcemy użyć tego zapisu to nadpisanie możemy uzyskać za pomocą operatora **>|**.

Ćwiczenia

1. Polecenie **date** wyświetla na monitorze aktualną datę. Przeadresuj polecenie **date** do pliku *data.txt* w katalogu *zadanie*. Zobacz czy data znajduje się w pliku *data.txt* za pomocą polecenia **more**.
2. Do pliku *data.txt* dopisz listę plików i katalogów z katalogu *zadanie*.
3. Posługując się poleceniem **cat** utwórz dwa pliki: *plik1* i *plik2*.
4. Utwórz plik *razem.txt*, który będzie zawierał treść *plik1*, *plik2*.
5. Użyj polecenia **cat**, aby wyświetlić zawartość: *plik1*, *plik2*, *plik3*. Z jakich strumieni wyjściowych pochodzą informacje wyświetlone na ekranie?
6. Powtórz ćwiczenie 5, tylko tak aby komunikat o błędach był przesłany do pliku *bledy*.
7. Powtórz ćwiczenie 6, tylko tak aby dodatkowo wynik połączenia plików był przesłany do pliku *dobre*.
8. Powtórz ćwiczenie 7, ale dodatkowo ustaw zmienną blokującą nadpisywanie. Co trzeba zmienić w poleceniu z ćwiczenia 7, aby nie wyświetlał się komunikat o błędzie.
9. Wyłącz zmienną blokującą nadpisywanie.
10. Powtórz ćwiczenie 5, tylko tak aby wszystko zostało zapisane w pliku *wszystko*.
11. Używając znaków **<** i **>** oraz polecenia **tr** prześlij zawartość pliku *wszystko* do pliku *całość* z jednoczesną zmianą małych liter na wielkie.