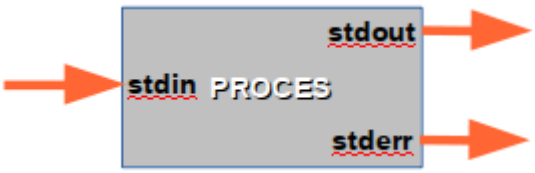
Temat zajęć	Strumienie, potoki, filtry
Zakres materiału	Operatory: `> >> < << 2> 2>>` Polecenia: `cat, cut, grep, head, sort, tail, tr, uniq, wc`

Strumienie

Strumień (ang. *stream*) jest abstrakcyjnym łączem znakowym, służącym do komunikacji pomiędzy pracującym procesem, a światem zewnętrznym. W czasie pracy procesu strumień jest kojarzony z pewnym wybranym zasobem fizycznym, np. urządzeniem, plikiem, połączeniem sieciowym, itp. Strumień identyfikowany jest przez nazwę oraz numer, który jest tożsamy z tzw. uchwytem plikowym (ang. *file handle*), używanym do reprezentowania plików otwieranych i obsługiwanych przez funkcje systemowe takie jak `open`, `read`, `write` i `close`.

Każdy proces w chwili uruchomienia otrzymuje od systemu trzy otwarte strumienie reprezentujące podstawowe źródła i ujścia danych. Są to:

- **stdin** (ang. *standard input*, standardowe wejście, numer 0) – strumień domyślnie powiązany z podstawowym urządzeniem wejściowym (klawiatura terminala)
- **stdout** (ang. *standard output*, standardowe wyjście, numer 1) – strumień domyślnie powiązany z podstawowym urządzeniem wyjściowym (ekran terminala)
- **stderr** (ang. *standard error output*, standardowe wyjście diagnostyczne, numer 2) - strumień domyślnie powiązany z podstawowym urządzeniem wyjściowym (ekran terminala)



Strumienie domyślnie wykorzystywane są w sposób następujący:

- dane wprowadzane przez użytkownika trafiają przez klawiaturę do strumienia 0
- dane wyprowadzane przez proces trafiają poprzez strumień 1 na ekran bądź inne urządzenie wizualizujące
- dane diagnostyczne (np. komunikaty o błędach) trafiają przez strumień 2 na ekran bądź inne urządzenie wizualizujące

Zasadniczo użytkownik nie jest w stanie rozpoznać, które z prezentowanych na ekranie danych dotarły do niego przez strumień 1 (*stdout*), a które przez strumień 2 (*stderr*), ale istniejące pomiędzy nimi rozróżnienie pozwala odseparować te informacje od siebie, o ile użytkownik sobie tego zażyczy.

Przekierowania

Przekierowaniem (eng. *redirection*) nazywamy działanie, w wyniku którego pewien strumień danych zostaje skojarzony z innym urządzeniem/plikiem, niż przypisanym domyślnie przez system. Przekierowanie określone w linii poleceń obowiązuje od momentu uruchomienia procesu aż do jego zakończenia. Zauważ, że proces ma możliwość wykrycia, że wykorzystywany przez niego strumień jest przekierowany i zmodyfikować swoje działanie zależnie od sytuacji. Np. polecenie `ls` może kolorować nazwy plików, gdy są one wyświetlane na ekranie i nie robić tego, gdy wynik działania narzędzia jest kierowany do pliku.

Przy tej okazji wspomnijmy też o ciekawym urządzeniu, które reprezentowane jest systemie jako plik o nazwie `/dev/null`, które zachowuje się jak *czarna dziura*. Oznacza to, że wszelkie dane kierowane do tego urządzenia (pliku) przepadają w nim bez najmniejszego śladu, a w przypadku próby odczytu plik ten zachowuje się tak, jakby był pusty, natychmiast sygnalizując sytuację końca danych. Z tego powodu jest często wykorzystywany jako śmietnik dla danych, które nie są nikomu potrzebne lub jako wzorzec pustego pliku.

Do definiowania przekierowań używa się następującego zbioru operatorów:

- **`>`** : przekierowanie standardowego wyjścia; w wyniku działania tego operatora dane wyprowadzane przez proces na strumień 1 zostaną **skierowane** do pliku/urządzenia wymienionego jak prawy argument operatora; jeżeli plik określony jako ujście danych już istnieje, jego zawartość zostanie usunięta i zastąpiona danymi wprowadzonymi przez proces; takiej operacji nie towarzyszy żadne ostrzeżenie, więc wskazana jest ostrożność; przykład:

  ```
  ls -l > ls-l
  ```

  spowoduje, że w pliku `ls-l` zostanie umieszczona długa lista plików znajdujących się w bieżącym katalogu.

- **`2>`** : przekierowanie wyjścia diagnostycznego; w wyniku działania tego operatora dane wyprowadzane przez proces na strumień 2 zostaną **skierowane** do pliku/urządzenia wymienionego jako prawy argument operatora zgodnie z zachowaniem opisanym wcześniej dla operatora `>`; przykład:

  ```
  ls -l tytyryty 2> error
  ```

  spowoduje, że w pliku `error` zostanie umieszczona treść komunikatu o błędzie, który pojawi się, jeśli katalog `tytyryty` jest niedostępny.

- **`>>`** : przekierowanie standardowego wyjścia; w wyniku działania tego operatora dane wyprowadzane przez proces na strumień 1 zostaną **dopisane** do pliku wymienionego jak prawy argument operatora; jeżeli plik określony jako ujście danych już istnieje, jego dotychczasowa zawartość pozostanie nienaruszona, a jeśli nie istnieje, to zostanie założony; przykład:

  ```
  ls -l .. >> ls-l
  ```

  spowoduje, że do pliku `ls-l` zostanie dopisana długa lista plików znajdujących się w katalogu nadrzędnym

- **`2>>`** : przekierowanie wyjścia diagnostycznego; w wyniku działania tego operatora dane wyprowadzane przez proces na strumień 2 zostaną **dopisane** do pliku wymienionego jako prawy argument operatora zgodnie z zachowaniem opisanym dla operatora `>>`; przykład:

  ```
  ls /root 2>> error
  ```

spowoduje, że do pliku `error` zostania dopisana treść nieuniknionego komunikatu o błędzie, który pojawi się, jeśli nie jesteśmy użytkownikiem *root*

- `<` : przekierowanie standardowego wejścia; w wyniku działania tego operatora dane z pliku o nazwie podanej jako prawy argument zostaną **wprowadzone** do procesu na jego strumień 0 dokładnie w taki sposób, jakby zostały odczytane z klawiatury; znane ci już polecenie `less` może wyświetlać dane z pliku o wskazanej nazwie albo też (jeśli nie podano żadnych nazw plików) odczytywane je ze standardowego wejścia; z tego powodu oba podane poniżej postaci są równoważne:

  ```
  less dane.txt
  ```

  ```
  less <dane.txt
  ```

- `<<` : wielowierszowe przekierowanie standardowego wejścia; w wyniku działania tego operatora wszystkie kolejne wiersze wprowadzane z klawiatury zostaną **wprowadzone** do procesu na jego strumień 0, a działanie to zostanie zakończone z chwilą napotkania na wejściu wiersza zawierającego taki sam ciąg znaków, jak podany w prawym argumencie operatora; przykład:

  ```
  cat <<STOP
  Wlazł kotek na płotek
  I mruga
  STOP
  ```

  oczywiście, przydatność takiego wariantu w zwykłej pracy konsolowej jest znikoma, jednak może oddać nieocenione usługi w przypadku użycia wewnątrz skryptu

Wiele z poleceń wyprowadza wyniki swojego działania na standardowe wyjście oraz równolegle dodatkowe informacje o błędach i/lub ostrzeżeniach na standardowe wyjście diagnostyczne. Ponieważ operatory dotyczące różnych strumieni można ze sobą dowolnie łączyć w jednej linii poleceń, tym samym istnieje możliwość rozdzielenia tych strumieni, np. w taki sposób::

```
cat in1 in2 2> err
```

Takie polecenie spowoduje wyświetlenie zawartości plików `in1` i `in2` oraz zapisanie informacji o błędach do pliku `err`.

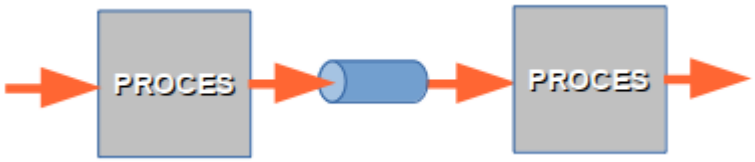
W celu zignorowania danych, które z jakichkolwiek względów nie są potrzebne, można je przekierować do pliku `/dev/null`.

W przypadku, gdy standardowy strumień wyjściowy został już przekierowany, a życzymy sobie, aby dane ze strumienia diagnostycznego trafiły w to samo miejsce, można użyć zapisu `2>&1`, np.:

```
cat in1 in2 in3 in4 in5 > out 2>&1
```

Potoki

Potok (ang. *pipe* lub *pipeline*) to łańcuch współpracujących procesów skonstruowany w taki sposób, aby każdy z nich (oprócz pierwszego) czytał dane w wyjścia procesu poprzedniego oraz aby każdy z nich (oprócz ostatniego) pisał swoje wyniki na wejście procesu następnego. Dane przesyłane pomiędzy procesami są w przezroczysty sposób buforowane przez system operacyjny, sprawiając wrażenie, że każdy z procesów pracuje absolutnie autonomicznie w takt napływających danych. Można stwierdzić, że potok jest specyficzną formą przekierowania, w której źródłem/ujściem danych nie jest plik/urządzenie, a pracujący **proces**. Operatorem, który spina dwa procesy w potok, jest znak | (ang. *bar*). Jego lewy argument jest nadajnikiem danych, a prawy odbiornikiem. Zauważ, że w potokach, w których uczestniczą więcej niż dwa procesy, wszystkie oprócz pierwszego i ostatniego są zarówno odbiornikami jak i nadajnikami.



Oto kilka prostych przykładów:

- `ls -al | less`

  wyniki produkowane przez polecenie `ls` są tu opracowywane przez `less` i prezentowane na ekranie w wygodny sposób

- `who | less`

  jak wyżej, ale w odniesieniu do listy zalogowanych użytkowników

- `who | sort | less`

  jak wyżej, ale lista będzie posortowana alfabetycznie według nazw użytkowników

Nie każdy proces jest zdolny pracować w potoku. Programy, które nie wykorzystują standardowego wejścia jako źródła przetwarzanych danych oraz standardowego wyjścia do wyprowadzania wyników, nie nadają się do tego celu (np. edytory). Oznacza to, że najczęściej w potokach wykorzystuje się tylko te narzędzia, które były tworzone specjalnie do tego celu. Są to tzw. filtry.

Filtry

Filtrem nazywamy program, którego podstawowym zadaniem jest wpływanie na postać danych odbieranych ze standardowego wejścia (niekiedy również z pliku, którego nazwę filtr dostaje w linii poleceń) i przekazywanie ich w zmienionej formie na standardowe wyjście (niekiedy również do pliku, którego nazwę filtr dostaje w linii poleceń). Pełnię swoim możliwości filtr ujawnia, gdy zostanie umiejętnie zastosowany jako element potoku. Poniżej przedstawiamy krótki opis najczęściej używanych i najbardziej przydatnych filtrów używanych standardowo w systemie Linux.

- `cat`

  zasadniczo nie wpływa na treść przesyłanych w nim danych, chociaż można sprawić, że będzie je subtelnie modyfikował zgodnie z podanymi mu przełącznikami:

  - `-b` numeruje niepuste linie
  - `-n` numeruje wszystkie linie
  - `-s` zastępuje wiele następujących po sobie pustych linii jedną
  - `-v` prezentuje znaki niedrukowalne w czytelny sposób, niezakłócający pracy terminala

- `head`

  wyprowadza na standardowe wyjście początkową porcję danych pojawiających się na standardowym wejściu (domyślnie jest to 10 pierwszych linii); może odczytywać dane z pliku o nazwie podanej jako argument; jego zachowanie mogą modyfikować następujące przełączniki:

  - `-c` *`x`* zamiast 10 pierwszych linii prześlij *x* pierwszych znaków
  - `-n` *`x`* zamiast 10 pierwszych linii prześlij *x* pierwszych linii

- `tail`

  wyprowadza na standardowe wyjście końcową porcję danych pojawiających się na standardowym wejściu (domyślnie 10 ostatnich linii); może odczytywać dane z pliku o nazwie podanej jako argument; jego zachowanie mogą modyfikować następujące przełączniki:

  - `-c` *`x`* zamiast 10 ostatnich linii prześlij *x* ostatnich znaków
  - `-n` *`x`* zamiast 10 ostatnich linii prześlij *x* pierwszych linii
  - `-f` pracuje w nieskończoność, próbując czytać znaki z końca rosnącego pliku

- `sort`

  służy do sortowania danych wejściowych, które domyślnie sortowane są *leksykograficznie*; sortowanie danych odbywa się liniami; najważniejsze przełączniki to:

  - `-n` wymusza sortowanie numeryczne
  - `–b` ignoruje spacje na początkach linii
  - `-d` wymusza tzw. tryb *książki telefonicznej*
  - `-f` ignoruje wielkość liter
  - `-t x` zmienia domyślny separator kolumn na znak x (domyślnie są spacje i tabulacje)
  - `-r` wymusza odwrotny porządek sortowania
  - `-i` ignoruje znaki niedrukowalne
  - `-k` *`KLUCZ`* wskazuje położenie danych według których będzie odbywać się sortowanie; w najprostszym przypadku sort zakłada, że każda z linii składa się z ponumerowanych od 1 pól (ciągów znaków rozdzielanych białymi znakami), a znaki wewnątrz pola również są numerowane i także od 1; najprostsza postać określenia *`KLUCZA`* to:
    `P1.Z1,P2.Z2`

gdzie `P1` to pole zawierające początek klucza, a `Z1` numer znaku w polu od którego zaczyna się klucz; analogicznie, `P2` i `Z2` określają położenie końca klucza i jeśli są pomięte wraz z poprzedzającym przecinkiem, to zakłada się, że klucz kończy się na ostatnim znaku linii, np.:

```
sort -k 2.1
```

pomija w sortowaniu pierwszą kolumnę.

- `uniq`

  usuwa **powtarzające się**, **sąsiednie** linie wczytywane z wejścia; używany z następującymi przełącznikami:

  - `-d` wyprowadza tylko linie powtarzające się
  - `–u` wyprowadza tylko linie unikalne
  - `–c` poprzedza wyprowadzane linie licznikami powtórzeń

- `wc`

  od ang. *word counter*; wyprowadza na wyjście wartości liczników podających, ile w danych wejściowych było linii, słów i znaków; przełączniki służą do wybrania podzbioru prezentowanych wartości i tak:

  - `-l` wyprowadź tylko licznik linii
  - `-w` wyprowadź tylko licznik słów
  - `-c` wyprowadź tylko licznik znaków

- `tr [przełącznik...] zbiór1 [ zbiór2 ]`

  od ang. *translate*; zamienia znaki ze wskazanego zbioru na wymienione w poleceniu; wymaga podania co najmniej jednego zbioru (w takim przypadku uznaje się, że drugi zbiór jest pusty); oryginalnie `tr` używa dość złożonej składni do zapisywania uogólnionych postaci zbiorów, my poprzestaniemy na razie na postaci prostszej, w której składowymi zbiorów są pojedyncze znaki; najczęściej stosowane przełączniki to:

  - `-c` traktuje pierwszy zbiór jakby był jego dopełnieniem (czyli wszystkimi znakami różnymi od podanych)
  - `-s` usuwa powtarzające się, sąsiednie znaki

- `cut`

  wycina wskazane fragmenty wierszy podawanych na wejście, a rezultat wycinania (zależny od przełączników) wysyła na wyjście; najczęściej używa się następujących przełączników:
  - `-c` określa początkowe i końcowe kolumny wycinanych fragmentów, np. `-c 1-10` wybiera pierwsze 10 znaków
  - `-f` określa numery wycinanych pól, np. `-f1,3-5,10` wyświetla pola 1,3,4,5 i 10
  - `-d` ustawia separator (separatory) pól (domyślnie jest nim tabulator)

  uwaga: przełączniki -c i -f mogą być użyte wielokrotnie

- `grep [przełącznik...] wyrażenie`

  wyszukuje w strumieniu danych odbieranym ze standardowego wyjścia ciągu znaków opisanego argumentem wyrażenie i reaguje stosownie do użytych przełączników; w najprostszym przypadku wyprowadza na swoje standardowe wyjście tylko te wiersze, które zawierają poszukiwany ciąg; **uwaga:** argument *wyrażenie* może być tzw. *wyrażeniem regularnym* (ang. *regular expression* lub w skrócie *regexp*), jednak ten wariant jego użycia zostanie omówiony w następnym ćwiczeniu; na razie uznajemy, że wyrażenie jest po prostu łańcuchem znaków do wyszukania; należy mieć jednak na uwadze, że pewne znaki mogą być traktowane specjalnie (są to: `.` (kropka), `^` (caret), `$` (dolar), `*` (gwiazdka), `[]` (nawiasy kwadratowe), `<>` (nawiasy ostre), `\` (backslash); jeśli zachodzi konieczność wyszukania ich jako zwykłych znaków, należy je poprzedzać znakiem `\` (backslash); ponadto, `grep` potrafi również przeszukiwać pliki o nazwach wymienionych w argumentach; najczęściej używane przełączniki to:
  - `-v` wyprowadza linie nie zawierające szukanego wzorca
  - `-c` wyprowadza liczbę liczbę odszukanych wyrażeń
  - `-i` ignoruje wielkość liter przy wyszukiwaniu
  - `-n` wyprowadza numery linii zawierających dany wzorzec

Zadania do wykonania w czasie zajęć

1. Program `cat` uruchomiony bez argumentów kopiuje na ekran znaki wprowadzane z klawiatury. Sprawdź, czy `cat` wykonuje to kopiowanie znak po znaku czy linia po linii?
2. Jak zakończyć działanie programu `cat` w powyższej sytuacji? Podaj dwa dostępne sposoby. Który z nich jest lepszy?
3. Jak `cat` prezentuje znaki niedrukowalne?
4. Jak użyć programu `cat`, aby wystąpił jako zamiennik program `cp` w poniższym kontekście?

   ```
   cp plik1 katalog2/plik2
   ```

5. Jak przy pomocy program `cat` uzyskać efekt identyczny z tym, który uzyskamy używając programu `touch` z nazwą nieistniejącego pliku?
6. Zbadaj, na który ze strumieni trafia komunikat emitowany przez polecenie `cd`, w którym użyto nazwy nieistniejącego katalogu. Jak się tego dowiedzieć?
7. Wiele poleceń systemu Linux uznaje, że jeśli w linii poleceń jako nazwa pliku wystąpi znak – (łącznik), to reprezentuje on standardowe wejście. Jednym z nich jest polecenie `cat`. Zakładając, że `a` i `b` są nazwami istniejących niepustych plików, sprawdź, jaki efekt wywoła polecenie następującej postaci:

   ```
   cat a – b > c
   ```

8. Jak ukryć komunikat o błędzie produkowany przez poniższe polecenie?

   ```
   cat /etc/shadow
   ```

9. Dlaczego nie istnieje operator `2<`?
10. Czy istnieje operator `>&1`?
11. Filtry `head` i `tail` potrafią posługiwać się *kilobajtami* i *megabajtami* jako licznikami początkowych/końcowych znaków; sprawdź w manualu jak osiąga się ten efekt.
12. Przeprowadź eksperyment pokazujący, jak `tail -f nazwa_pliku` prezentuje w czasie rzeczywistym fakt przyrastania pliku. Wskazówka: posłuż się więcej niż jednym oknem terminala
13. Wyjaśnij, co to znaczy, że domyślne sortowanie stosowane przez filtr `sort` jest *leksykograficzne*. W jakich sytuacjach sortowanie takie będzie nieprzydatne?
14. Na czym polega sortowanie w trybie książki telefonicznej?
15. Czym jest *słowo* w sensie używanym przez filtr `wc`?
16. Posortuj plik `/etc/passwd` według identyfikatorów UID. Zapisz wynik do pliku.
17. Ile jest różnych (niepowtarzalnych) identyfikatorów GID w pliku `/etc/passwd`?
18. Sporządź własną kopię pliku `/etc/group`, w którym wszystkie znaki przestankowe zostaną zamienione na spacje.
19. Do czego służy nieomówiony w instrukcji filtr o nazwie `tee`?
20. Pewien plik zawiera listę studentów. Każda linia zawiera kolejno nazwisko, imię i ocenę. Sporządź nową wersję tego pliku tak, aby w każdej linii było kolejno nazwisko i ocena.
21. Tak otrzymany nowy plik posortuj według nazwisk.
22. Pewien plik zawiera numery PESEL, po jednym w linii. Jak wykryjesz powtarzające się numery? Jak je policzysz?
23. Pewien zestaw pięciu plików tekstowych zawiera listy obecności sporządzane na pięciu kolejnych wykładach. Jak wyszukać studentów, którzy byli na wszystkich wykładach?

24. Ściągnij spakowany plik będący fragmentem prawdziwej kroniki pewnego serwera, zawierającej informacje o poprawnych logowaniach (jest to nikły procent zawartości tego pliku) oraz o znacznie liczniejszych próbach włamań na różne konta. Zapoznaj się ze strukturą tego pliku i sposobem, w jaki obrazowane są różne błędne sytuacje. Użyj poznanych filtrów, w celu znalezienia odpowiedzi na pytania zadane w kolejnych punktach. Uwaga: unikaj tworzenia plików tymczasowych – użyj ich dopiero wtedy, gdy nie będziesz miał innego wyjścia.
25. Z ilu różnych adresów IP próbowano się włamywać?
26. Z ilu różnych adresów IP logowano się poprawnie?
27. Na jakie konto najczęściej próbowano się włamywać?
28. Na jakie konto najczęściej logowano się poprawnie?
29. Na ile różnych kont logowano się poprawnie?
30. Na ile różnych kont próbowano się włamywać?
31. Którego dnia miało miejsce najwięcej prób włamań?
32. W jakie dni nikt nie logował się poprawnie?
33. **Dla zaawansowanych:** z jakich krajów włamywano się najczęściej? Wskazówka: istnieje polecenie o nazwie `whois`, przy pomocy którego można zapytać o dane administracyjne skojarzone z pewnym adresem IP, np.

    ```
    whois 82.145.72.70
    ```

    wyprowadza (przytoczono tylko początek ilustrujący skąd można pobrać daną o kraju):

    ```
    % This is the RIPE Database query service.
    % The objects are in RPSL format.
    %
    % The RIPE Database is subject to Terms and Conditions.
    % See http://www.ripe.net/db/support/db-terms-conditions.pdf
    % Note: this output has been filtered.
    % To receive output for a database update, use the "-B" flag.
    % Information related to '82.145.72.0 - 82.145.75.255'
    % Abuse contact for '82.145.72.0 - 82.145.75.255' is 'abuse@man.szczecin.pl'

    inetnum:        82.145.72.0 - 82.145.75.255
    netname:        TUNET
    descr:          Szczecin University of Technology
    country:        PL
    admin-c:        BB1418-RIPE
    tech-c:         AP2028-RIPE
    tech-c:         PM5262-RIPE
    status:         ASSIGNED PA
    mnt-by:         ACI-MNT-POLAND
    source:         RIPE # Filtered
    ```