3. Strumienie i potoki

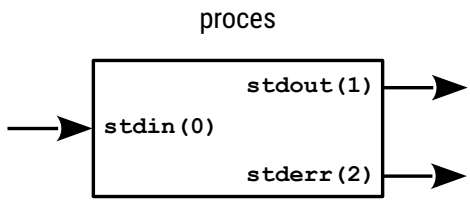
3.1. Strumienie

Strumień (ang. *stream*) jest abstrakcyjnym łączem znakowym (tzn. łączem zdolnym transmitować pojedyncze znaki/bajty, w odróżnieniu od łącz blokowych, które transmitują w jednym cyklu większe porcje danych, tak, jak czynią to na przykład urządzenia dyskowe albo sieciowe), służącym do komunikacji pomiędzy pracującym procesem a światem zewnętrznym. W czasie pracy procesu strumień jest kojarzony z pewnym wybranym zasobem fizycznym, na przykład urządzeniem, plikiem czy też połączeniem sieciowym. Strumień identyfikowany jest przez nazwę oraz nieujemną liczbą całkowitą, która pełni rolę tak zwanego *deskryptora pliku* (ang. *file descriptor*) bądź według innej terminologii, uchwytu pliku (ang. *file handle*). Dana ta jest udostępniana procesowi w momencie otwarcia strumienia (na przykład z użyciem funkcji `open()`, która zwraca deskryptor poprzez swój wynik), a następnie proces używa jej do reprezentowania plików obsługiwanych przez funkcje systemowe takie jak na przykład `read()`, `write()` i `close()`.

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

– **stdin** (od ang. *standard input*, standardowe wejście, deskryptor 0) – strumień domyślnie powiązany z podstawowym urządzeniem wejściowym (klawiatura

terminala)
- **stdout** (ang. *standard output*, standardowe wyjście, deskryptor 1) – strumień domyślnie powiązany z podstawowym urządzeniem wyjściowym (ekran terminala)
- **stderr** (ang. *standard error output*, standardowe wyjście diagnostyczne, deskryptor 2) – strumień domyślnie powiązany z podstawowym urządzeniem wyjściowym (ekran terminala)



Rys. 3..1: Współpraca procesu ze strumieniami standardowymi

Strumienie domyślnie wykorzystywane są w sposób następujący:
- dane wprowadzane przez użytkownika przez klawiaturę trafiają do strumienia 0
- dane wyprowadzane przez proces trafiają poprzez strumień 1 na ekran bądź inne urządzenie wizualizujące
- dane diagnostyczne (na przykład 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.

3.2. 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. Na przykład 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 w systemie jako plik o nazwie **`/dev/null`**, a 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 też powodu plik **`/dev/null`** 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 jako 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.txt
```

spowoduje, że w pliku *ls-l.txt* 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 >; na przykład:

```
ls -l hidden_dir 2> error.txt
```

spowoduje, że w pliku *error.txt* zostanie umieszczona treść komunikatu o błędzie, który pojawi się, jeśli katalog *hidden_dir* 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; jeśli nie istnieje, to zostanie utworzony; na przykład:

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

spowoduje, że do pliku *ls-l.txt* zostanie dopisana długa lista plików znajdują-

cych 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 »; na przykład:

```
ls /root 2>> error.txt
```

spowoduje, że do pliku *error.txt* zostania dopisana treść nieuniknionego komunikatu o błędzie, który pojawi się, jeśli nie jesteś 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) odczytywać je ze swojego 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; na 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 standar-

dowe wyjście diagnostyczne. Ponieważ operatory dotyczące różnych strumieni można ze sobą dowolnie łączyć w jednej linii poleceń, tym samym istnieje możliwość rozdzielenia tych strumieni, na przykład 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 do niczego potrzebne, można je przekierować do pliku **/dev/null**, na przykład:

```
cat in1 in2 2> /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**, na przykład:

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

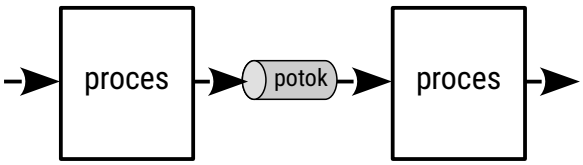
3.3. Potoki

Potok (ang. *pipeline* lub w skrócie *pipe*) 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 inny pracujący **proces**. Operatorem, który spina dwa procesy w potok, jest znak **|** (ang. *bar*, chociaż spotyka się również określenie *pipe*). 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 buforowane przez proces **less** i prezentowane na ekranie w wygodny sposób, umożliwiając użytkownikowi



Rys. 3..2: Współpraca procesów poprzez potok

ich swobodne przeglądanie i przeszukiwanie (w tym miejscu warto być może przypomnieć, że wyjście z przeglądarki **less** następuje po naciśnięciu klawisza **q**)

who | less
jak wyżej, ale w odniesieniu do listy zalogowanych użytkowników

who | sort | less
jak wyżej, ale przed wyświetleniem na konsoli lista zostanie posortowana alfabetycznie według nazw użytkowników

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

3.4. 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 swoich 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.

Dla celów dalszych rozważań załóżmy istnienie pliku *studenci.txt*, który mógłby

być częścią listy ocen z pewnej trudnej wejściówki, a który zawiera następujące cztery wiersze tekstu:

```
Waldemar Zaskoczony 3
Roksana Pewna 3,5
Euzebia Nadzwyczajna 05
Waldemar Zaskoczony 3
```

Założymy również, że każdy prezentowany potok rozpoczynać będziemy od polecenia **cat**, którego zadaniem będzie jedynie „karmienie" potoku danymi, natomiast ewentualne przekształcenia zawartości pliku odbywać się będą w kolejnych ogniwach.

cat (od ang. *concatenate*)
zasadniczo nie wpływa na treść przesyłanych w nim danych, chociaż można sprawić, że będzie je subtelnie modyfikował zgodnie z podanymi mu przełącznikami, na przykład:

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

Na przykład – ponumerowanie wszystkich linii tekstu z potoku:

```
$ cat studenci.txt | cat -n
     1 Waldemar Zaskoczony 3
     2 Roksana Pewna 3,5
     3 Euzebia Nadzwyczajna 05
     4 Waldemar Zaskoczony 3
```

head
wyprowadza na standardowe wyjście początkową porcję danych pojawiających się na standardowym wejściu (domyślnie jest to 10 pierwszych linii); może też odczytywać dane z plików o nazwach podanych jako argumenty; 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

Na przykład – wyprowadzenie jedynie pierwszych dwóch linii tekstu z potoku:

```
$ cat studenci.txt | head -n 2
Waldemar Zaskoczony 3
Roksana Pewna 3,5
```

tail
wyprowadza na standardowe wyjście końcową porcję danych pojawiających się na standardowym wejściu (domyślnie 10 ostatnich linii); może odczytywać dane z plików o nazwach podanych jako argumenty; jego zachowanie mogą modyfikować następujące przełączniki:

-c *x* zamiast 10 ostatnich linii prześlij *x* ostatnich znaków
-n *x* zamiast 10 ostatnich linii prześlij *x* ostatnich linii

Na przykład – wyprowadzenie jedynie ostatniej linii z potoku:

```
$ cat studenci.txt | head -n 2 | tail -1
Roksana Pewna 3,5
```

sort
służy do sortowania danych wejściowych, które domyślnie sortowane są leksykograficznie (słownikowo); sortowanie danych odbywa się wierszami; najważniejsze przełączniki to:

-n wymusza sortowanie numeryczne (próbuje traktować klucze sortowania jak liczby, nie jak tekst)
-b ignoruje spacje na początkach linii
-d wymusza tak zwany *tryb książki telefonicznej*, tzn. ignoruje w czasie sortowania znaki, które nie są literami albo cyframi
-f ignoruje wielkość liter (utożsamia litery małe i wielkie)
-t *x* zmienia domyślny separator kolumn (tzn. umownych pól wewnątrz wierszy) na znak *x* (domyślnie separatorami są spacje i tabulacje)
-r wymusza odwrotny porządek sortowania
-i ignoruje znaki niedrukowalne
-k *key* 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/kolumn (ciągów znaków rozdzielanych domyślnie białymi znakami), a znaki wewnątrz każdego z osobna pola również są numerowane i także od 1; najprostsza postać definicji klucza to:
col_beg.char_beg,col_end.char_end
gdzie **col_beg** to numer pola zawierającego początek klucza, a **char_beg**

to numer znaku w polu, od którego zaczyna się klucz; analogicznie, **col_end** i **char_end** określają położenie końca klucza i jeśli są pominięte wraz z poprzedzającym przecinkiem, to zakłada się, że klucz kończy się na ostatnim znaku linii, na przykład:

```
sort -k 2.1
```

pomija w sortowaniu pierwszą kolumnę.

Poniżej podajemy kilka elementarnych przykładów prezentujących różne zachowania filtra . Oto jak działa posortowanie linii z potoku bez wskazania klucza:

```
$ cat studenci.txt | sort
Euzebia Nadzwyczajna 05
Roksana Pewna 3,5
Waldemar Zaskoczony 3
Waldemar Zaskoczony 3
```

Posortowanie linii z potoku alfabetycznie według nazwiska:

```
$ cat studenci.txt | sort -k 2
Euzebia Nadzwyczajna 05
Roksana Pewna 3,5
Waldemar Zaskoczony 3
Waldemar Zaskoczony 3
```

Posortowanie linii z potoku alfabetycznie według oceny:

```
$ cat studenci.txt | sort -k 3
Euzebia Nadzwyczajna 05
Waldemar Zaskoczony 3
Waldemar Zaskoczony 3
Roksana Pewna 3,5
```

Posortowanie linii z potoku numerycznie według oceny:

```
$ cat studenci.txt | sort -k 3 -n
Waldemar Zaskoczony 3
Waldemar Zaskoczony 3
```

```
Roksana Pewna 3,5
Euzebia Nadzwyczajna 05
```

uniq (od ang. *unique*)
usuwa **powtarzające się, sąsiednie linie**, wczytywane z wejścia albo z plików o nazwach podanych w argumentach; używany z następującymi przełącznikami:

- **-d** wyprowadza tylko linie powtarzające się (czyli jest to odwrócenie zachowania domyślnego)
- **-u** wyprowadza tylko linie unikalne
- **-c** poprzedza wyprowadzane linie licznikami powtórzeń

Ze względu na sposób jego działania użycie filtra **uniq** ma sens tylko i wyłącznie wtedy, gdy napływające do niego wiersze tekstu są już wstępnie posortowane – tylko w takim przypadku może mieć miejsce faktyczne usuwanie bądź zliczanie powtórzeń. Dlatego też filtr ten najczęściej pojawia się w potokach dopiero po filtrze **sort**.
Na przykład odnalezienie w potoku powielonej linii:

```
$ cat studenci.txt | sort | uniq -d
Waldemar Zaskoczony 3
```

Policzenie liczby wystąpień każdej linii w posortowanym potoku:

```
$ cat studenci.txt | sort | uniq -c
      1 Euzebia Nadzwyczajna 05
      1 Roksana Pewna 3,5
      2 Waldemar Zaskoczony 3
```

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

Na przykład zebranie statystyk przetwarzanego potoku:

```
$ cat studenci.txt | wc
      4      12      86
```

tr (od ang. *translate*)
tr *[przełącznik...] zbiór1 [zbiór2]*
zamienia znaki ze wskazanego zbioru *zbiór1* na wymienione znaki ze zbioru *zbiór2*; wymaga podania co najmniej jednego zbioru (w takim przypadku uznaje się, że drugi zbiór jest pusty, co oznacza żądania usuwania znaków wymienionych z pierwszym zbiorze); w pełnej wersji **tr** używa złożonej składni do zapisywania uogólnionych postaci zbiorów, my poprzestaniemy na postaci prostszej, w której składowymi zbiorów są pojedyncze znaki; najczęściej stosowane przełączniki to:

- **-c** traktuje pierwszy zbiór jakby był swoim własnym dopełnieniem (czyli wszystkimi znakami różnymi od podanych *explicite*)
- **-s** usuwa powtarzające się, sąsiednie znaki

Na przykład zamiana przecinków na kropki:

```
$ cat studenci.txt | tr , .
Waldemar Zaskoczony 3
Roksana Pewna 3.5
Euzebia Nadzwyczajna 05
Waldemar Zaskoczony 3
```

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, na przykład **-c 1-10** wybiera pierwsze 10 znaków z każdego wiersza
- **-f** określa numery wycinanych pól, na przykład **-f1,3-5,10** wyświetla pola 1,3,4,5 i 10

-d *del* ustawia znak ***del***(ewentualnie znaki) użyte w pliku do separowania pól pól (separatorem domyślnym jest tabulator)
uwaga: przełączniki **-c** *i* **-f** *wykluczają się wzajemnie i nie mogą być użyte więcej niż raz; co więcej, przełączników tych nie można użyć do zmiany kolejności znaków/pól, na przykład* **-f3,2** *wywoła taki skutek jak* **-f2,3***; do zamiany kolejności pól najczęściej używa się bardziej uniwersalnego narzędzia o nazwie* **awk** *(nazwa pochodzi od nazwisk*

twórców pierwszej wersji tego programu, Alfreda V. Aho, Petera Weinbergera i Briana Kernighana i odczytuje się ją jako jedno słowo awk, wymawiane jak pierwsza sylaba w słowiek awkward; rozpoznanie możliwości ***awk*** *pozostawia się czytelnikowi)*

Na przykład wybranie z potoku wyłącznie nazwisk i ocen:

```
$ cat studenci.txt| cut -d ' ' -f2,3
Zaskoczony 3
Pewna 3,5
Nadzwyczajna 05
Zaskoczony 3
```

grep

Pochodzenie nazwy tego niezwykle elastycznego narzędzie nie jest do końca jasne. Być może jest to skrót od angielskich słów *global regular expression print*, ale nie jest też wykluczone, że *grep* wziął swoją nazwę od nieużywanego już interaktywnie, prymitywanego wierszowego edytora **ex** (zdarza się jeszcze, że używa się go jako komponentu skryptów do wykonania automatycznie złożonych modyfikacji plików). Edytor **ex** do wyszukania łańcucha w tekście używał polecenia o następującej postaci:

```
:g/wyrażenie regularne/p
```

a ponieważ w angielszczyźnie wyrażenia regularna są często określane mianem *re* (od słów *regular expression*), to w efekcie otrzymujemy wyglądający już bardzo znajomo ciąg znaków **g/re/p**. Podstawowa składnia uruchomienia polecenia **grep** prezentuje się następująco:

```
grep [przełącznik...] wyrażenie
```

Użyty jako filtr **grep** 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 wyrażeniem regularnym, jednak ten wariant jego użycia zostanie omówiony w następnym rozdziale; na razie uznajemy, że wyrażenie jest po prostu łańcuchem znaków do wyszukania; należy mieć jednak na uwadze, że pewne znaki mogą być tutaj traktowane*

specjalnie (są to: **.** *(kropka),* **^** *(caret),* **$** *(dolar),* ***** *(gwiazdka),* **[]** *(nawiasy kwadratowe),* **<>** *(nawiasy ostre),* **** *(backslash); jeśli zachodzi konieczność wyszukania ich jako zwykłych znaków, należy je poprzedzać (zacytować) znakiem* **** *(backslash); ponadto,* **grep** *potrafi również przeszukiwać pliki o nazwach wymienionych w argumentach, ale ten wariant pozostawiamy do omówienia w kolejnym rozdziale.*
Najczęściej używane przełączniki to:

- **-v** wyprowadza linie nie zawierające szukanego wzorca (jest to odwrócenie zachowania domyślnego)
- **-c** wyprowadza liczbę odnalezionych wyrażeń
- **-i** ignoruje wielkość liter przy wyszukiwaniu (utożsamia litery małe i wielkie)
- **-n** wyprowadza numery linii zawierających wyszukiwany wzorzec

Na przykład wybranie z potoku ocen pana Zaskoczonego:

```
$ cat studenci.txt| grep -i zaskoczony | cut -d ' ' -f3
3
3
```

3.5. Źródła uzupełniające

1. Manual polecenia **cat**: `https://man7.org/linux/man-pages/man1/cat.1.html`
2. Manual polecenia **cut**: `https://man7.org/linux/man-pages/man1/cut.1.html`
3. Manual polecenia **grep**: `https://man7.org/linux/man-pages/man1/grep.1.html`
4. Manual polecenia **head**: `https://man7.org/linux/man-pages/man1/head.1.html`
5. Manual polecenia **less**: `https://man7.org/linux/man-pages/man1/less.1.html`
6. Manual polecenia **sort**: `https://man7.org/linux/man-pages/man1/sort.1.html`
7. Manual polecenia **tail**: `https://man7.org/linux/man-pages/man1/tail.1.html`
8. Manual polecenia **tr**: `https://man7.org/linux/man-pages/man1/tr.1.html`

9. Manual polecenia **uniq**: `https://man7.org/linux/man-pages/man1/uniq.1.html`
10. Manual polecenia **wc**: `https://man7.org/linux/man-pages/man1/wc.1.html`
11. Opis mechanizmu potoków w powłoce Bash: `https://www.gnu.org/software/bash/manual/bash.html#Pipelines`
12. Opis mechanizmu przekierowań w powłoce Bash: `https://www.gnu.org/software/bash/manual/bash.html#Redirections`

3.6. Zadania do samodzielnego wykonania

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 programu **cp** w poniższym kontekście?
   ```
   cp plik1 katalog2/plik2
   ```
5. Jak przy pomocy programu **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. Tak zachowuje się też 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 licz-

nikami początkowych/końcowych znaków; sprawdź w manualu, jak osiąga się ten efekt.

12. Przeprowadź eksperyment pokazujący, jak polecenie postaci

```
tail -f nazwa_pliku
```

prezentuje w czasie rzeczywistym fakt przyrastania danych w 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. Skonstruuj eksperyment ilustrujący na czym polega sortowanie w trybie numerycznym (opcja **`-n`** polecenia **`sort`**).
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 tekście filtr o nazwie **`tee`**?
20. Do czego służy nieomówiony w tekście filtr o nazwie **`pv`**?
21. Pewien plik zawiera listę studentów. Każda linia zawiera kolejno nazwisko, imię i ocenę, rozdzielone znakiem spacji. Sporządź nową wersję tego pliku tak, aby w każdej linii znalazły się kolejno ocena i nazwisko Wskazówka: użyj nieomówiony w instrukcji filtr **`paste`**.
22. Tak otrzymany nowy plik posortuj według nazwisk.
23. Pewien plik zawiera numery PESEL, po jednym w linii. Jak wykryjesz powtarzające się numery? Jak je policzysz?
24. Pewien zestaw pięciu plików tekstowych zawiera listy obecności sporządzane na pięciu kolejnych wykładach. Jak wyszukać studentów, którzy byli na wszystkich wykładach?