Środowisko pracy informatyka

Piotr Fulmański
Cezary Obczyński

Piotr Fulmański[1]
Cezary Obczyński[2]

Wydział Matematyki i Informatyki
Uniwersytet Łódzki
Banacha 22, 90-238, Łódź
Polska

e-mail:
1: fulmanp@math.uni.lodz.pl
2: czacza@math.uni.lodz.pl

Data ostaniej modyfikacji: **6 listopada 2008**

Spis treści

Przedmowa

Tytułem wstępu nic nie bedzie ;)

Rozdział 1

UNIX

1.1 Czym jest UNIX?

Unix Time-Sharing System (pisane również jako UNIX, choć nie jest to skrót – nazwa „UNIX" jest kalamburem określenia MULTICS, który był wzorem dla Uniksa) – system operacyjny napisany w 1969 r. w Bell Labs (UNIX System Laboratories) przez Dennisa Ritchie i Kena Thompsona. Rozwijany później w bardzo dynamiczny sposób, co zaowocowało powstaniem wielu odmian i implementacji.

Od roku 2007 właścicielem znaku UNIX jest The Open Group. Tylko systemy w pełni spełniające **Single UNIX Specification** mogą używać nazwy UNIX. Pozostałe systemy nazywane są UNIX-o podobnymi[1].

Jako system operacyjny UNIX jest najbardziej znany z tego, iż teoretycznie wszystko jest plikiem oraz z założeń projektowych będących współczesną wersją brzytwy Ockhama[2] czyli reguły KISS[3] (ang. *Keep It Simple, Stupid*, czyli *to ma być proste, głupku*), mówiących żeby tworzyć program

[1]Ponieważ jednak wszystko to co będzie napisane dalej jest na ogół prawdziwe w każdym systemie czy to typu UNIX czy UNIX-o podobnym, dlatego aby zaoszczędzić sobie pisania będziemy pisać po prostu UNIX.

[2]Brzytwa Ockhama – wprowadzona przez Williama Ockhama (ok. 1285-1349) zasada: istnień nie należy mnożyć ponad potrzebę (łac. Non sunt multiplicanda entia sine necessitate), tłumaczona także tradycyjnie jako: Bytów nie mnożyć, fikcyj nie tworzyć, tłumaczyć fakty jak najprościej. W praktyce tłumaczy się to jako: proste rozwiązanie jest najlepsze albo nie wymyślaj nowych czynników jeżeli nie istnieje taka potrzeba, a jeżeli już, to udowodnij najpierw ich istnienie.

[3]Reguła ta posiada także polski odpowiednik: BUZI (Bez Udziwnień Zapisu, Idioto), atrakcyjny przez to, że nie tylko oddaje sens akronimu, ale i on sam kojarzy się z angielskim pierwowzorem (kiss – całować; pocałunek, buziak).

robiący dobrze jedną rzecz.

1.2 Interpreter jako łącznik

Interpreter poleceń (ang. *command processor*[4]) jest częścią systemu operacyjnego (programem) odpowiedzialną za tłumaczenie poleceń systemowych wprowadzanych przez użytkownika w trybie konwersacyjnym. Oznacza to, iż program ten czyta linie tekstu wprowadzone przez użytkownika i interpretuje je zależnie od kontekstu jako polecenia systemu operacyjnego lub język programowania.

Interpreter poleceń często nazywany jest także **powłoką** (ang. *shell*). W pierwotnym znaczeniu shell był programem stanowiącym interfejs dla użytkownika, czyli pełnił rolę warstwy pośredniczącej między użytkownikiem a systemem operacyjnym (jego jądrem). W znacznie szerszym kontekście (który nas nie będzie interesował) oznaczać będzie każde oprogramowanie zbudowane „nad" jakąś częścią systemu komputerowego pozwalające na jego wykorzystanie[5].

Iterfejsy użytkownika traktowane z punktu widzenia użytkownika systemu operacyjnego, podzielić możemy na dwie kategorie: tekstowe (ang. *command line*) oraz graficzne (ang. *graphical*) dostarczające odpowiednio tekstowego (CLI – command line interface) lub graficznego (GUI – graphical user interface) interfejsu użytkownika. Bez względu na sposób interakcji z użytkownikiem (tekstowy czy graficzny) zasadniczym celem każdego z nich jest wywołanie (uruchomienie) innego, wybranego przez użytkownika, programu i jego kontrolowanie.

Zwolennicy interfejsu tekstowego, nie bez racji, twierdzą, że wiele operacji można wykonać znacznie szybciej w trybie tekstowym niż graficznym. Graficzny interfejs przewyższa natomiast tekstowy jeśli chodzi o intuicyjność i łatwość obsługi (brak konieczności pamiętania składni wielu poleceń). W praktyce odpowiednio elastyczną pracę uzyskuje się wykorzystując zarówno jeden jak i drugi interfejs.

Na zakończenie wypada stwierdzić, że terminy interpreter, powłoka i shell zwykle traktowane są jako synonimy i w większości przypadków oznaczają „narzędzie" tekstowe.

[4] Alternatywne nazwy: *command line interpreter*, *command line shell*, *command language interpreter*.

[5] Na przykład, choć może to nie jest najlepszy przykład, przeglądarka internetowa stanowi swojego rodzaju powłokę nad oprogramowaniem renderującym wygląd strony.

I jeszcze jedna uwaga: kontynuując temat interpreterów skupimy się wyłącznie na interpreterach UNIX-o podobnych.

1.3 Przegląd interpreterów

Pierwszym interpreterem napisanym dla systemu UNIX był sh, czyli powłoka Bourne'a (ang. Bourne shell). Program ten odznacza się dużą szybkością działania i z tego względu jest wciąż powszechnie wykorzystywany do uruchamiania skryptów. Jego funkcjonalność jest niestety mocno ograniczona podczas pracy interaktywnej ze względu na brak licznych udogodnień, które oferują nowocześniejsze powłoki.

Znacznie wygodniejszy jest drugi klasyczny interpreter csh (ang. C shell). Program csh oferuje m.in. historię poleceń, aliasy, i sterowanie pracami. Składnia złożonych poleceń jest bardzo zbliżona do języka C (w którym został napisany) i różni się znacząco od składni stosowanej w sh.

Trzecim klasycznym interpreterem jest ksh (ang. Korn shell), stanowiący rozszerzenie programu sh o wiele nowych funkcji. Z założenia, ksh przeznaczony był dla administratorów i raczej nie zdobył uznania zwykłych użytkowników ze względu na skomplikowaną składnię.

Żaden z wymienionych programów nie jest dostępny w klasycznej wersji w systemie Linux.

Obecnie największą popularność cieszą się nowoczesne interpretery łączące duże możliwości z łatwością obsługi. Należy tu wymienić powłoki zsh i bash (ang. Bourne Again shell), powstałe z rozwinięcia sh, oraz powłokę tcsh. Powłoka zsh staje się powoli standardem systemu UNIX, zaś bash jest od samego początku podstawowym i domyślnym interpreterem Linuxa.

W dalszej części opisujemy powłokę bash.

1.4 Pliki konfiguracyjne

Powłoka bash może wykorzystać kilka plików konfiguracyjnych w zależności od sposobu uruchomienia.

Powłoka uruchomiona po zalogowaniu użytkownika (ang. login shell) odczytuje kolejno zawartość następujących plików:

1. `/etc/profile`
2. `~/.bash_profile`
3. `~/.bash_login`

4. `~/.profile`

Każda powłoka interaktywna odczytuje zawartość plików:

1. `/etc/bashrc`

2. `~/.bashrc`

1.5 Edycja poleceń

Powłoka bash daje bogate możliwości edycji poleceń. Pozwala łatwo modyfikować wpisane polecenie poprzez przemieszczanie się kursorami i w wierszu, usuwanie niepoprawnych znaków i wstawianie nowych. Jest to szczególnie przydatne do poprawiania długich poleceń złożonych z wielu wyrazów, w których błąd popełniono na samym początku. Opisany poniżej mechanizm historii pozwala przywołać takie polecenie, poprawić i ponownie wykonać.

Interpreter umożliwia ponadto uzupełnianie nazw poleceń i plików podczas edycji wiersza polecenia. Po wprowadzeniu początkowego fragmentu nazwy polecenia można wcisnąć `[Tab]`, aby powłoka odnalazła i uzupełniła nazwę. Poszukiwania prowadzone są wśród wbudowanych poleceń oraz w katalogach wymienionych w ścieżce poszukiwań programów `PATH`. Podobnie odbywa się uzupełnianie nazw plików z tym, że poszukiwania dotyczą tylko jednego katalogu wskazanego w początkowym fragmencie nazwy ścieżkowej pliku (domyślnie jest to katalog bieżący). W obydwu przypadkach, jeśli bash odnajdzie kilka pasujących nazw, to wyświetla ich listę dając użytkownikowi możliwość wyboru.

Po zakończeniu edycji użytkownik zatwierdza polecenie wciskając `[Enter]`.

Interpreter może przechowywać listę ostatnio wydanych poleceń i udostępniać je użytkownikowi do powtórnego wykorzystania. Mechanizm ten nosi nazwę historii poleceń[6]. Powłoka bash udostępnia historię poleceń, jeśli ustawiona jest opcja `history`. Liczbę pamiętanych poleceń określa zmienna `HISTSIZE` ustawiana domyślnie na wartość 500. Po uruchomieniu, bash zaczyna pracę z pustą listą historii, ale może ją zainicjować z pliku określonego zmienną `HISTFILE`. Domyślnie jest to plik `~/.bash_history`. Maksymalną liczbę linii w pliku określa zmienna `HISTFILESIZE`.

Aktualną listę historii można wypisać poleceniem `history`. Każde z zapamiętanych poleceń poprzedzone jest swoim numerem. Posługując się tym

[6] Oryginalna powłoka Bourne'a sh nie ma wbudowanego mechanizmu historii.

numerem lub początkowym fragmentem nazwy, można przywołać konkretne polecenie w następujący sposób:

- **!identyfikator** — ogólna postać odwołania do historii poleceń;
- **!!** — przywołuje ostatnie polecenie;
- **!n** — przywołuje polecenie o numerze n;
- **!-n** — przywołuje polecenie położone n pozycji od końca listy czyli od ostatniego polecenia;
- **!string** — przywołuje polecenie rozpoczynające się ciągiem znaków string.

1.6 Konfiguracja interpretera

Sposób działania interpretera jest w dużym stopniu ustalany przez parametry i opcje.

1.6.1 Parametry

Parametry służą do przechowywania różnych wartości. Mogą być reprezentowane przez nazwę w postaci liczbę naturalną, jeden ze znaków specjalnych lub ciągu znaków.

- Parametry reprezentowane przez liczbę naturalną noszą nazwę **parametrów pozycyjnych** i przechowują argumenty wywołania powłoki.
- **Parametry specjalne** mają postać pojedyńczych znaków specjalnych a ich wartości są automatycznie ustawiane i modyfikowane dynamicznie przez powłokę.
- Parametry reprezentowane przez nazwę określane są jako **zmienne**. Ustawianie wartości zmiennych odbywa się za pomocą jednego z trzech poleceń:

```
zmienna=[wartość]

typeset zmienna=[wartość]

declare zmienna=[wartość]
```

Jeśli wartość zostanie pominięta, to zmiennej zostanie przypisany zerowy ciąg znaków. Usunięcie zmiennej umożliwia polecenie:

```
unset zmienna
```

1.6.2 Parametry pozycyjne

Parametry pozycyjne reprezentowane są przez kolejne liczby naturalne. Parametr `$0` przechowuje nazwę powłoki lub nazwę skryptu. Parametry `$1`, `$2`, `$3`,... przechowują kolejne argumenty. Odwołania do parametrów o wyższych numerach, składających się z kilku cyfr, wymagają użycia nawiasów klamrowych: `${10}`, `${11}`,... Wszystkie parametry pozycyjne można jednocześnie przesunać w lewo o dowolną liczbę pozycji za pomocą polecenia: `shift [n]`. Gdy argument nie jest podany, domyślnie następuje przesunięcie wszystkich parametrów o jedną pozycję. W rezultacie parametr `$1` jest usuwany (bezpowrotnie), parametr `$2` przesuwa się na `$1`, parametr `$3` na `$2` itd. W przypadku gdy parametr `n` jest podany przesuwanie argumentów następuje o `n` pozycji, tzn. `n` pierwszych argumentów jest usuwanych a pozostałe zajmują pozycje zaczynając od `$1`. Jak widać z powyższego opisu, parametr `$0` nigdy nie podlega przesunięciu i zawsze przechowuje nazwę.

Początkowe ustawienie parametrów pozycyjnych można zmienić poleceniem: `set arg1 arg2 arg3 ...` Polecenie zmienia na raz wszystkie parametry pozycyjne, usuwając stare wartości i przypisując części z nich nowe wartości podane w poleceniu. Nie ma możliwości zmiany tylko wybranych parametrów.

1.6.3 Parametry specjalne

Każda powłoka ustawia automatycznie kilka parametrów specjalnych, których wartości zmieniają się w trakcie jej działania (patrz tabela 1.1). Parametr `$?` umożliwia sprawdzenie, czy jakieś polecenie zakończyło się pomyślnie. Parametr `$$` zawiera identyfikator PID bieżącej powłoki, który można wykorzystać np. do stworzenia pliku tymczasowego o unikalnej nazwie. Powłoka ustawia również trzy parametry specjalne związane z parametrami pozycyjnymi. Parametr `$#` określa liczbę ustawionych parametrów pozycyjnych z wyjatkiem parametru `$0`. Parametr `$*` przechowują całą listę parametrów pozycyjnych w postaci jednego ciągu znaków `"$1 $2 $3 ..."`, natomiast `$@` przechowuje tę listę w postaci oddzielnych ciągów `"$1" "$2" "$3"` ... Do parametrów tych można odwołać się pisząc np. prosty skrypt

Parametr	Znaczenie
#	Liczba parametrów pozycyjnych.
*	Lista parametrów pozycyjnych w postaci jednego ciągu znaków.
@	Lista parametrów pozycyjnych w postaci oddzielnych ciągów znaków.
?	Status zakończenia ostatnio uruchomionego polecenia.
$	PID bieżącej powłoki.
!	PID procesu ostatnio uruchomionego w tle.
-	Lista opcji bieżącej powłoki.

Tablica 1.1. Parametry specjalne.

```
#!/bin/bash
echo "liczba argumentow=$#"
echo "lista parametrow>$*<"
echo "lista parametrow>$@<"
echo "PID biezacej powloki:$$"
echo "Status zakonczenia ostatniego procesu:$?"
echo "argument nr 1: $1"
echo "argument nr 2: $2"
echo "argument nr 3: $3"
```

1.6.4 Parametry — zmienne

Zmienne możemy podzielić na dwie kategorie:

- zmienne środowiska;
- zmienne lokalne powłoki.

Polecenie `set` pozwoli nam uzyskać listę wszystkich zmiennych. Odwołanie do wartości dowolnego parametru następuje za pomocą znaku `$`:

```
$parametr
```

```
${parametr}
```

Nawiasów użyć musimy zawsze, gdy w poleceniu bezpośrednio za nazwą parametru chcemy umieścić inne znaki lub gdy nazwa liczbowa składa się z więcej niż jednej cyfry.

```
x=Ala
```

```
echo $x
echo $x123
echo ${x}123
```

Zmienne środowiska

Każdy tworzony proces, w tym również każda powłoka, otrzymuje nowe środowisko w postaci tablicy zawierającej ciągi znaków `zmienna=wartość`. Każdy taki ciąg oznacza utworzenie nowej zmiennej środowiska. Ten początkowy zestaw zmiennych środowiska jest dziedziczony po procesie macierzystym, którym najczęściej jest powłoka uruchamiająca nowy program. W trakcie działania procesu środowisko może być modyfikowane poprzez usuwanie i dodawanie nowych zmiennych oraz przez zmiany ich wartości. Wszystkie modyfikacje znajdują odzwierciedlenie we wspomnianej tablicy. Zmiany te jednak nie przenoszą się do procesu macierzystego. Zasada ta dotyczy również procesów potomnych danego procesu utworzonych przed modyfikacją środowiska.

Zmienne środowiska wyróżnia więc to, że nie są związane tylko z lokalną powłoką, ale są dziedziczone przez wszystkie procesy potomne tworzone przez powłokę. Są wykorzystywane przez różne procesy a nie tylko przez proces powłoki. Nie można ich jednak traktować jako zmienne globalne, gdyż nie stają się widoczne dla wszystkich procesów.

Całe środowisko można wyświetlić przy pomocy polecenia `printenv` lub `env`. Do wypisania wartość pojedyńczej zmiennej służy polecenie

```
echo $zmienna
```

Dodanie do środowiska nowej zmiennej polega na wyeksportowaniu zmiennej lokalnej, której trzeba wcześniej przypisać właściwą wartość

```
zmienna=wartość
export zmienna
```

Zmienną środowiska można usunąć poleceniem `unset`, podobnie jak zmienną lokalną.

Najważniejsze zmienne środowiska zdefiniowane w każdej powłoce zebrano w tabeli 1.2.

Zmienne lokalne powłoki

Zmienne lokalne mają znaczenie wyłącznie dla bieżącej powłoki i tylko w niej są widoczne. Nie podlegają dziedziczeniu i nie mogą być wykorzystane

Zmienna	Znaczenie
CDPATH	Ścieżka poszukiwań katalogów.
HOME	Katalog domowy zalogowanego użytkownika.
IFS	Wejściowy separator pola.
LANG	Język lokalny.
LD_LIBRARY_PATH	Ścieżka poszukiwań bibliotek dynamicznych.
LOGNAME	Nazwa zalogowanego użytkownika.
MAIL	Skrzynka pocztowa zalogowanego użytkownika.
MAILCHECK	Częstotliwość sprawdzania skrzynki pocztowej.
MANPATH	Ścieżka poszukiwań dokumentacji systemowej.
PATH	Ścieżka poszukiwań programów.
PWD	Katalog bieżący.
SHELL	Powłoka uruchamiana po zalogowaniu użytkownika.
TERM	Typ terminala.
TZ	Strefa czasowa.
USER	Nazwa zalogowanego użytkownika.

Tablica 1.2. Najważniejsze zmienne środowiska.

przez inne procesy. Do definiowana i usuwania zmiennych wykorzystujemy przedstawione wcześniej sposoby.

1.7 Opcje

Dodatkową możliwość konfiguracji interpretera zapewniają opcje. Można je ustawić podając odpowiednie argumenty przy wywołaniu powłoki albo w trakcie jej działania posłużyć się poleceniem:

```
set -o [opcja ...]
```

Polecenie wydane bez argumentów wypisuje aktualne ustawienia poszczególnych opcji. Usuwanie opcji umożliwia polecenie:

```
set +o [opcja ...]
```

Większość opcji ma swoje jednoliterowe odpowiedniki, których można użyć przy uruchamianiu powłoki.

Zapis	Znaczenie
`?`	Zastępuje jeden dowolny znak.
`*`	Zastępuje dowolny ciąg znaków.
`[ ]`	Zastępuje jeden znak z listy podanej wewnątrz nawiasów.
`[ - ]`	Zastępuje jeden znak z zakresu podanego wewnątrz nawiasów.
`[^ ]` albo `[! ]`	Zastępuje dowolny znak oprócz tych podanych wewnątrz nawiasów.

Tablica 1.3. Znaki specjalne stosowane we wzorcach nazw plików.

1.8 Interpretacja znaków specjalnych przez powłokę

1.8.1 Metaznaki i generowanie nazw plików

Każdy interpreter dopuszcza stosowanie w nazwach plików specjalnych znaków, które mogą zastępować jeden lub więcej dowolnych znaków. Otrzymuje się w ten sposób wzorce pasujące do nazw kilku plików. Zestaw znaków specjalnych powłoki bash przedstawiono w tabeli 1.9. Przykład.

```
ls *.[hc]
```

1.8.2 Cytowanie

Cytowanie stosowane jest w celu usunięcia specjalnego znaczenia niektórych znaków i słów. Zapobiega interpretacji tych znaków przez powłokę, przywracając ich podstawowe znaczenie. Niektóre sposoby cytowania dopuszczają pewne wyjątki.

- `\` – Zapobiega interpretacji następnego znaku.
- `" "` – Zapobiegają interpretacji wszystkich znaków zawartych wewnątrz cudzysłowów, z wyjątkiem znaków `$` i ‘‘.
- `’ ’` – Zapobiegają interpretacji wszystkich znaków zawartych wewnątrz apostrofów.
- ‘ ‘ – Wykonuje polecenie zawarte wewnątrz znaków i wstawia w to miejsce strumień wyjściowy tego polecenia.

Przykład.

```
#‘ - ,,lewy’’ apostrof
echo "‘date‘ $pwd"
```

1.8.3 Grupowanie poleceń

Interpreter umożliwia jednoczesne wydanie kilku poleceń w jeden linii. Sposób grupowania decyduje o tym, w jaki sposób i w jakiej kolejności powłoka wykona te polecenia. Poniżej przedstawiono kilka możliwości połączenia poleceń w jednej linii oraz efekty takiego użycia.

- `cmd ; cmd ; cmd` – Powłoka grupuje polecenia na pierwszym planie wykonując je sekwencyjnie.
- `cmd & cmd & cmd &` – Powłoka grupuje polecenia w tle wykonując je asynchronicznie.
- `cmd | cmd | cmd` – Powłoka grupuje polecenia wykonując je w potoku.
- `(cmd ; cmd)` – Powłoka grupuje polecenia i wykonuje w nowej powłoce.
- `{ cmd ; cmd ; }` – Powłoka grupuje polecenia i wykonuje w bieżącej powłoce, tworzy jeden strumień wyjściowy dla całej grupy poleceń i zwraca jeden status zakończenia.

Przykłady ilustrujące grupowanie znaleźć można w części 1.10.

1.8.4 Przekierowanie strumieni

Powłoka związuje z każdym tworzonym procesem trzy strumienie danych:

- standardowy strumień wejściowy, tzw. stdin,
- standardowy strumień wyjściowy, tzw. stdout,
- standardowy strumień błędów (diagnostyczny), tzw. stderror.

Każdy ze strumieni związany jest z plikiem zwykłym lub z plikiem urządzenia wejścia/wyjścia. Domyślnym przywiązaniem dla wszystkich strumieni jest plik terminala. Każdy plik otwarty przez proces otrzymuje kolejny numer zwany deskryptorem pliku. Deskryptory 0, 1 i 2 zarezerwowane są dla plików związanych ze standardowymi strumieniami. Przed wykonaniem dowolnego polecenia można zmienić domyślne przywiązanie strumieni, czyli przekierować wejście i wyjście z polecenia.

- `[d]<plik` – Przekierowanie wejścia powoduje otwarcie pliku do czytania z deskryptorem `d`; jeżeli `d` zostało pominięte przyjmowane jest 0.
- `[d]>plik` – Przekierowanie wyjścia powoduje otwarcie pliku do pisania z deskryptorem `d`; jeżeli `d` zostało pominięte przyjmowane jest 1.
- `[d]>>plik` – Przekierowanie wyjścia powoduje otwarcie pliku do dopisywania z deskryptorem `d`; jeżeli `d` zostało pominięte, przyjmowane jest 1.
- `[d]<>plik` – Przekierowanie powoduje otwarcie pliku do czytania i pisania z deskryptorem `d`.
- `&>plik lub >&plik` – Przekierowanie powoduje skierowanie strumienia wyjściowego i strumienia błędów do tego samego pliku.
- `[d1]>&d2` – Przekierowanie powoduje zduplikowanie deskryptora pliku wyjściowego; deskryptor `d1` staje się kopią deskryptora `d2` i wskazuje na ten sam plik.

Operatory przekierowania można umieścić w linii polecenia zarówno przed jak i za poleceniem. Przekierowania wykonywane są w kolejnosci występowania licząc od lewej do prawej.

Przykłady użycia

Przykład 1 Polecenie `ls` wypisuje pliki i katalogi znajdujące się w bieżącym katalogu. Wywołanie postaci

```
ls > lista.txt
```

powoduje zapisanie efektu działania polecenia `ls` a więc listy plików i katalogów w pliku `lista.txt`. Jeśli plik o takiej nazwie istniał to zostanie on usunięty i utworzony nowy. Wywołanie postaci

```
ls >> lista.txt
```

powoduje dopisanie efektu działania polecenia `ls` a więc listy plików i katalogów do pliku `lista.txt`. Jeśli plik o takiej nazwie nie istniał to zostanie on utworzony.

Znak	Znaczenie
~	katalog domowy użytkownika,

Tablica 1.4. Inne znaki specjalne.

Przykład 2 Utworzenie kopii pliku. Wykorzystamy polecenie `cat`, którego standardowe wyjście (ekran) przekierujemy do pliku `list_kopia.txt`.

```
cat lista.txt > lista_kopia.txt
```

Przykład 3 Łączenie przekierowań. Polecenie `sort` sortuje wiersze podane na jego wejściu. Poniższe wywołanie powoduje, że polecenie to pobiera wiersze z pliku `przyklad.txt` a wynik dzialania zapisuje w pliku `przyklad_sort.txt`

```
sort < przyklad.txt > przyklad_sort.txt
```

Kolejne przykłady ilustrujące użycie strumieni znaleźć można w części 1.10.

1.8.5 Inne

Interpreter korzysta jeszcze z kilku innych znaków specjalnych (tabela 1.4).

1.9 System plików

Prymitywne systemy plikowe posiadały jedną listę plików wchodzących w skład systemu plików. W miarę postępu systemy plików zostały unowocześnione i współczesne systemy plików oprócz głównej listy (**katalogu głównego**) zawierają także podlisty (**podkatalogi**) które również mogą zawierać dalsze podlisty (podkatalogi). W ten sposób rozwinęło się pojęcie **katalogów nadrzędnych** oraz **katalogów podrzędnych**.

Katalog, w którym znajduje się plik jest nadrzędny (bezpośrednio) względem tego pliku. Katalog jest podrzędny (bezpośrednio) względem innego katalogu w którym się znajduje.

W ten sposób powstało pojęcie drzewiastej hierarchii systemu plików, w której każdy katalog może zawierać podkatalogi, które mogą zawierać dalsze podkatalogi. Powstaje w ten sposób struktura, która wygląda jak „odwrócone drzewo”.

Systemy, które mają taką strukturę nazywają się hierarchicznymi systemami plików. W takich systemach plik identyfikowany jest nie na postawie jego nazwy ale całej ścieżki dostępu. Format ścieżki zależy od systemu operacyjnego. W systemach typu UNIX katalog jest reprezentowany przez ukośnik (ang. *slash* – „/").

```
+- [boot]
+- [etc]
+- [home]
|   +- [fulmanp]
|       +- [zajecia]
|       |   +- srodowisko.pdf
|       |
|       +- [dokumenty]
|           +- wazneCos.txt
|
+- [usr]
+- [var]
```

Dla powyższej struktury plików pełna ścieżka do pliku `srodowisko.pdf` wygląda następująco:

```
/home/fulmanp/zajecia/srodowisko.pdf
```

1.9.1 Struktura systemu plików

Dla systemu GNU/Linux powstały dwa formalne standardy określające układ katalogów w głównym systemie plików: FSSTND i bazujący na nim FHS.

Filesystem Hierarchy Standard (FHS)

Filesystem Hierarchy Standard (FHS) czyli Standard Hierarchii Systemu Plików definiuje rozmieszczenie plików i katalogów w systemach operacyjnych z rodziny Linux i Unix.

Proces rozwoju standardu dla hierarchii plików w systemie plików rozpoczął się w sierpniu 1993 próbą ustalenia struktury katalogów dla systemu Linux. FSSTND (Filesystem Standard), czyli hierarchia katalogów dla Linuksa, został wydany 14 lutego 1994 (z poprawkami z 9 października 1994 i 28 marca 1995).

Na początku roku 1996 rozpoczęto tworzenie szerszej wersji FSSTND adresowanej nie tylko dla systemu Linux, ale także dla innych systemów Uniksowych (przy pomocy społeczności skupionej przy rozwoju BSD). Nazwa standardu została zmieniona na Filesystem Hierarchy Standard lub FHS.

FHS jest zarządzany przez Free Standards Group – niedochodową grupę składającą się z twórców oprogramowania i sprzętu komputerowego (m.in.: HP, Red Hat, IBM i Dell). Nie wszyscy twórcy dystrybucji Linuksa dokładnie przestrzegają tego standardu np. w większości dystrybucji katalog /srv/ nie jest stosowany. Niektóre systemy Linuksowe i Uniksowe odchodzą od reguł zapisanych w FHS (np. GoboLinux). Mac OS X łączy standard FHS z łatwymi do zapamiętania nazwami katalogów, np. `/Library/` czy `/Applications/`.

Wszystkie pliki i katalogi znajdują się w katalogu głównym `/`, nawet jeśli są przechowywane na innych dyskach fizycznych. Niektóre z tych katalogów mogą być obecne tylko po instalacji konkretnego oprogramowania np. X Window System.

- `/bin/` Podstawowe pliki wykonywalne (ang. *binaries*) dostępne dla wszystkich użytkowników (np. `cat`, `ls`, `cp`).
- `/boot/` Pliki bootloadera (np. `kernel`, `initrd`).
- `/dev/` Pliki urządzeń (ang. *device*; np. `/dev/null`).
- `/etc/` Pliki konfiguracyjne.
  - `/etc/opt/` Pliki konfiguracyjne dla katalogu `/opt/`.
  - `/etc/X11/` Pliki konfiguracyjne X w wersji 11.
  - `/etc/sgml/` Pliki konfiguracyjne dla SGML.
  - `/etc/xml/` Pliki konfiguracyjne dla XML.
- `/home/` Katalogi domowe użytkowników (ang. *home directory*).
- `/lib/` Biblioteki (ang. *libraries*) dla programów z katalogów `/bin/` i `/sbin/`.
- `/mnt/` Punkt montowania (ang. *mount point*) innych niż natywny systemów plików.
- `/media/` Punkty montowań dla nośników wymienialnych (ang. *media*; np. CD-ROMów).

- `/opt/` Statycznie kompilowane aplikacje.
- `/proc/` Wirtualny system plików `proc` informujący o stanie systemu i poszczególnych procesów, w większości pliki tekstowe (np. `uptime`, `network`).
- `/root/` Katalog domowy użytkownika `root`.
- `/sbin/` Pliki wykonywalne do zarządzania systemem (ang. *system binaries*; np. `init`, `route`, `ifup`).
- `/tmp/` Pliki tymczasowe (ang. *temporary files*), których stan nie jest gwarantowany po zamknięciu systemu.
- `/srv/` Dane specyficzne dla miejsca (ang. *site-specific*), które są udostępniane (ang. *served*) przez system.
- `/usr/` Drugorzędowa hierarchia dla danych współdzielonych użytkownika (ang. *user shareable*), dane tylko do odczytu.
  - `/usr/bin/` Jak w hierarchi pierwszorzędowej, ale nie wymagane do uruchomienia, czy naprawy systemu.
  - `/usr/include/` Standardowe pliki nagłówkowe (ang. *include files*).
  - `/usr/lib/` Jak w hierarchi pierwszorzędowej.
  - `/usr/sbin/` Jak w hierarchi pierwszorzędowej, ale nie wymagane do uruchomienia, czy naprawy systemu (np. demony różnych usług sieciowych).
  - `/usr/share/` Dane niezależne od architektury, a więc między nimi współdzielone (ang. *shared*).
  - `/usr/src/` Kod źródłowy (ang. *source code*; np. źródła kernela z jego nagłówkami)
  - `/usr/X11R6/` X w wersji 11 wydaniu 6.
  - `/usr/local/` Trzeciorzędowa hierarchia danych lokalnych (ang. *local data*), specyficzna dla tego hosta.
- `/var/` Pliki często ulegające zmianom (ang. *variable files*), takie jak: logi, bazy danych, tymczasowe pliki e-mail.
  - `/var/lock/` Pliki blokady (ang. *lock*) zasobów będących w użyciu.

Polecenie	Działanie
`cd`	Zmiana katalogu.
`ls`	Wyświetlenie zawartości katalogu.
`pwd`	Bieżące położenie w systemie plików.
`mkdir`	Utworzenie katalogu.
`rmdir`	Usunięcie katalogu.
`mv`	Przeniesienie pliku.
`rm`	Usunięcie pliku.
`cp`	Kopiowanie pliku.
`touch`	Zamiana daty ostatniej modyfikacji lub dostępu do pliku bądź utworzenie pustego pliku.

Tablica 1.5. Polecenia pozwalające na manipulowanie plikami i katalogami.

- `/var/log/` Logi różnych aplikacji.
- `/var/mail/` Skrzynki pocztowe (ang. *mailboxes*) użytkowników.
- `/var/run/` Informacje o działaniu systemu (ang. *system runing*) od ostatniego jego uruchomienia (np. aktualnie zalogowani użytkownicy, uruchomione demony).
- `/var/spool/` Miejsce przechowania oczekujących zadań (ang. *spooling*; np. kolejki wydruku, nieprzeczytane e-maile).
- `/var/spool/mail/` Przestarzałe miejsce na skrzynki pocztowe (ang. *mailboxes*) użytkowników.
- `/var/tmp/` Pliki tymczasowe (ang. *temporary files*) których stan nie jest gwarantowany po zamknięciu systemu. Preferowane zamiast `/tmp` w trybie wielu użytkowników (multi-user).

1.9.2 Manipulowanie plikami i katalogami

Najważniejsze polecenia pozwalające na manipulowanie plikami i katalogami wymienione zostały w tabeli 1.5. Pozwalają one na elementarne zarządzanie systemem plików: tworzenie, przenoszenie i usuwanie plików a także poruszanie się po systemie plików. W tabeli 1.6 wymieniono inne polecenia dotyczące systemu plików.

ls

- `-l` – long;
- `-a` – all;

Polecenie	Działanie
`chmod`	Zmiana uprawnień.
`chgrp`	Zmiana grupy do której należy plik.
`chown`	Zmiana właściciela pliku.
`df`	wypisywanie wolnej przestrzeni zamontowanych systemów plików.
`du`	wyświetla ilość miejsca zajmowanego przez pliki/katalogi.
`file`	ustalanie typu danych zawartych w pliku.
`find`	przeszukiwanie systemu plików.

Tablica 1.6. Inne polecenia związane z systemem plików.

rm

- `-r` – rekursywne usuwanie wszystkich plików i podkatalogów z bieżącego katalogu;

find

Poniższe polecenie wyszukuje plików o nazwie `myfile` w katalogu `/home/fulmanp`.

```
find /home/fulmanp -name myfile -type f
```

Polecenie

```
find local /tmp -name mydir -type d
```

wyszukuje katalogi o nazwie `mydir` w podkatalogu `local` bieżącego katalogu i w katalogu `/tmp`.

1.9.3 Uprawnienia do plików i katalogów

Użytkownicy, grupy i własność

Systemu typu UNIX pozwalają na pracę wielu użytkowników w tym amym czasie. **Użytkownik** (ang. *usuer*) jest pojęciem pozwalającym na logiczny podział zasobów i zarządzanie nimi (w szczególności uprawnieniami). *Uzytkownik* może być związany zarówno z fizycznie istniejącą osobą (np. użytkownik `fulmanp`, którym jest Piotr Fulmański) jak i pewnym „typem operacji systemowych" (np. użytkownik `www` z którym związana jest obsługa lokalnego serwera www. Z punktu widzenia systemu UNIX nie jest istotne kim lub czym użytkownik jest. Istotne natomiast jest to, jakie zasoby (pliki, programy, urządzenia itd.) należą do którego z użytkowników i co on może

z nimi zrobić. Identyfikacja każdego z użytkowników odbywa się za pomocą tzw. UID (ang. *User ID*) a tzw. *username* lub *login* jak np. `fulmanp` czy `www` są tylko aliasami dla UID ułatwiającymi ludziom pracę w systemi.

Użytkownicy mogą zostać zorganizowani za pomocą nadrzędnej struktury jaką jest grupa (ang. *group*). Każdy użytkownik może należeć do jednej lub wielu grup. Istnienie grup uzasadnione jest chęcią kontrolowania identycznych uprawnień przypisanych pewnej ilości użytkownikom a związanych z wybranym zasobem. W takiej sytuacji zmiana uprawnień dla grupy skutkuje zmianą uprawnień dla wszystkich użytkowników wchodzących w skład grupy.

Każdy plik w systemie UNIX jest (jednocześnie) własnością:

- pewnego użytkownika
- pewnej grupy.

Użytkownik będący właścicielem posiada prawo modyfikowania uprawnień.

Uprawnienia

Z każdym plikiem w systemie UNIX związany jest zbiór uprawnień, mówiących kto i co może z plikiem zrobić. Uprawnienia to:

- `read` – prawo do czytania,
- `write` – prawo do pisania (modyfikowania),
- `execute` – prawo do wykonania.

O ile znaczenie tych uprawnień dla plików wydaje się dość oczywiste, o tyle dla katalogów wymaga już wyjaśnienia

- `r` – możliwość przeglądania zawartości katalogu,
- `w` – możliwość zmiany zawartości katalogu,
- `x` – możliwość przejścia do katalogu (polecenie cd).

Zbiór uprawnień dzieli się na trzy grupy:

- uprawnienia dla właściciela (ang. **o***wner*),
- uprawnienia dla grupy której własnością jest plik (ang. *owner* **g***roup*,
- uprawnienia dla pozostałych (ang. **o***ther*).

Umownie występowanie danego prawa oznaczane jest przez wyświetlenie odpowiedniej litery natomiast jego brak sygnalizowany jest znakiem `-`. Całość zapisywana jest jako jeden ciąg. Na przykład, zapis

```
rwxr-x--x
```

oznacza, że

- aktywnymi uprawnieniami dla właściciela są: `r`, `w` oraz `x`,
- aktywnymi uprawnieniami dla grupy są: `r` oraz `x`,
- aktywnym uprawnieniem dla pozostałych jest `x`.

Przykładowym wynikiem zastosowania polecenia

```
ls -l
```

jest

```
$ ls -l
drwxr-xr-x 3 fulmanp fulmanp   4096 2008-01-02 23:23 materialy
-rwxr-xr-x 1 fulmanp fulmanp    163 2007-10-14 00:19 preface.tex
-rw-r--r-- 1 fulmanp fulmanp  51024 2007-10-15 14:01 srodowisko.dvi
-rw-r--r-- 1 fulmanp fulmanp 192775 2008-10-07 14:04 srodowisko.pdf
-rwxr-xr-x 1 fulmanp fulmanp   2471 2008-10-06 09:06 srodowisko.tex
drwxr-xr-x 2 fulmanp fulmanp   4096 2008-10-03 11:56 TEST
-rwxr-xr-x 1 fulmanp fulmanp  53065 2008-10-07 16:58 unix.tex
```

Polecenie `chmod`

Aby ustanowić lub zmodyfikować uprawnienia należy użyć polecenia `chmod`. Składnia polecenia jest następująca:

```
chmod [opcje] uprawnienia plik(i)
```

Uprawnienia określamy podając co chcemy zrobić, z jakimi uprawnieniami i w stosunku do kogo. Na przykład zapis

```
chmod a-x dane.txt
```

oznacza, że wszystkim (a jak ang. *all*) a więc właścicielowi, grupie i innym osobom zabieramy (znak `-`) prawo `x`. W ogólności mamy do dyspozycji następujące symbole:

- `u` – oznaczajacy właścicila,

- `g` – oznaczajacy grupę,
- `o` – oznaczajacy pozostałych użytkowników,
- `a` – oznaczajacy wszystkie powyższe grupy (czyli właściciela, grupę i pozostałych użytkowników).

O przyznawaniu lub odbieraniu prawa decydują symbole

- `+` – nadanie prawa,
- `-` – odebranie prawa,
- `=` – ustanowienie dokładnie takiego prawa.

Spojrzmy na ponższe przykłady

```
$ ls -l
-rw-r--r-- 1 fulmanp fulmanp      0 2008-10-07 17:19 test.txt
$ chmod a-r test.txt
$ ls -l
--w------- 1 fulmanp fulmanp      0 2008-10-07 17:19 test.txt
$ chmod g+rwx test.txt
$ ls -l
--w-rwx--- 1 fulmanp fulmanp      0 2008-10-07 17:19 test.txt
$ chmod g=w test.txt
$ ls -l
--w--w---- 1 fulmanp fulmanp      0 2008-10-07 17:19 test.txt
```

Dziwne liczby

Podany powyżej sposób jest dobry do zmian pojedyńczych uprawnień, np. odebranie prawa do czytania dla grupy. Jeśli jednak za jednym razem chcemy nadać konkretny zestaw uprawnień dotyczący wszystkich trzech grup tj. właściciela, grupy i pozostałych użytkowników, wówczas o wiele wygodniej jest skorzystać z możliwości określania uprawnień za pomocą liczb.

W takim przypadku posługujemy się umową, że prawu `r` odpowiada liczba 4, prawu `w` odpowiada 2 a prawu `x` odpowiada 1. Jeśli chcemy np. ustanowić następujące prawa

```
rwxr-x-w-
```

dla pliku `test.txt` wówczas jako uprawnienia podajemy trzy liczby (dla każdej z grup osobno), z których każda jest sumą uprawnień, które mają zostać ustanowione

Polecenie	Działanie
`cat`	Otwiera plik i kopiuje jego zawartość na stdout.
`cut`	Z każdej linii pliku wybiera zadane pola.
`csplit`	Podział pliku według wzorca.
`split`	Podział pliku na części o zadanej ilości wierszy.
`wc`	Zliczanie wierszy, wyrazów i znaków w pliku.

Tablica 1.7. Polecenia dotyczące plików i katalogów.

```
      owner group other
      rwx   r-x   -w-
      |||   | |    |
      ||1   4 1    2
      |2
      4

suma: 7     5     2
```

A więc polecenie przyjmie postać

```
chmod 752 test.txt
```

Polecenie `chgrp`

1.10 Najczęściej wykorzystywane polecenia

Polecenia nie stanowią o funkcjonalności powłoki, bo jak wyjaśniono wcześniej są samodzielnymi programami przez nią wywoływanymi. Jednak właśnie pracując z wierszem poleceń możemy przekonać się o wielkiej potędze ukrytej w tekstowych poleceniach. Dlatego teraz przedstawimy kilka z nich podając ich możliwe zastosowania.

Najważniejsze: pomoc do każdego polecenia po wpisaniu `man nazwaPolecenia`.

1.10.1 Polecenia związane z plikami i katalogami

cat

Można wykorzystać do łączenia plików:

```
cat plik1 plik2 plik3 > sumaPlikow
```

cut

Polecenie `cut` umożliwia wybranie z każdego wiersza elementów z określonych pól. Najprościej wyjaśnić to na przykładzie. Załóżmy, że mamy następujący plik o nazwie `cut.txt`:

```
123;456;789
ab;cde;efg
ABC;DEF;GHIJ
```

Poniżej przedstawion przykłady użycia polecenia `cut` do wybrania określonych pól z każdego wiersza, przy założeniu, że pola rozdzielone są znakiem `;`

```
$ cut -f1 -d';' cut.txt
123
ab
ABC
$ cut -f1,3 -d';' cut.txt
123;789
ab;efg
ABC;GHIJ
$ cut -f1-3 -d';' cut.txt
123;456;789
ab;cde;efg
ABC;DEF;GHIJ
```

Wybierać można też poszczegolne kolumny znaków

```
$ cut -c1  cut.txt
1
a
A
$ cut -c1-2  cut.txt
12
ab
AB
$ cut -c1-6  cut.txt
123;45
ab;cde
ABC;DE
$ cut -c1,4-6  cut.txt
```

```
1;45
acde
A;DE
```

csplit

Polecenie `csplit` dzieli plik podany jako argument wejściowy na części w oparciu o zadane kryteria podziału. Można na przykład podzielić plik na fragmenty zawierające po 10 linii i każdy z takich fragmentów zapisać w osobnym pliku. `csplit` wyświetla rozmiar każdego z utworzonych plików.

Załóżmy, że mamy następujący plik o nazwie `f.txt`:

```
linia 1
linia 2
Rozdzial
linia 4
linia 5
Rozdzial
linia 7
linia 8
linia 9
linia 10
Rozdzial
linia 12
linia 13
```

Wówczas polecenie

```
csplit f.txt 5 {1}
```

spowoduje podział pliku `f.txt` na 3 części

- Pierwszy zawierający linie od 1 do 4. 4 a nie 5, gdyż wybierane są wszystkie linie począwszy od pierwszej do linii spełniającej kryterium ale bez tej linii (w tym przypadku kryterium jest ilość linii).
- `{1}` powoduje, że powtórzone zostanie kryterium wyboru, czyli wybrane zostaną linie od ostatniego punktu podziału do linii 10 (ostatnim punktem podziału była linia 5, a 5+5 równa się 10). W efekcie wybranych zostanie 5 linii.
- Reszta pliku.

Polecenie	Działanie
`cut`	Wybieranie kolumn.
`diff`	Porównywanie plików.
`grep`	Wybieranie wierszy.
`head`	Wybieranie wierszy początkowych.
`sed`	Edycja strumienia danych (napis po napisie).
`tail`	Wybieranie wierszy końcowych.

Tablica 1.8. Polecenia wyboru.

Poniższe polecenie spowoduje podział pliku `f.txt` na 4 części według dopasowania do wzorca, którym w tym przypadku jest tekst `Rozdzial`. Dopasowanie to zostanie powtórzone jeszcze 2 razy.

```
csplit  -f chap f.txt /Rozdzial/ {2}
```

Otrzymane pliki można połączyć w jeden używając polecenia

```
cat chap0[0-3] > razem.txt
```

split

`split` - jest to program konsoli Uniksa służący do dzielenia plików na mniejsze części. Bez podania parametrów dzieli plik tak, aby każda część miała maksymalnie 1000 linii. Pliki wyjściowe nazywane są tak, że do nazwy podanej jako argument dodaje się `aa` dla pierwszego pliku, `ab` dla drugiego itd. Bez podania podstawowego członu nazwy, polecenie `split` stosuje domyślnie `x`.

Polecenie

```
split -b4m plik.bin
```

spowoduje podział pliku `plik.bin` na części o rozmiarze 4 MB (`k` – kB, `m` – MB, nic – B).

Podział pliku na fragmenty po 4 linie uzyskujemy poleceniem

```
split -l 4 f.txt
```

1.10.2 Polecenia wyboru

```
#pierwsze 10 wierszy
head file
```

```
#pierwsze 5 wierszy
head -n 5 file
lub
head --lines=5 file
#wszystkie z wyjątkiem ostatnich 5 wierszy
head --lines=-5 file

#ciagle wypisywanie nowych wierszy
tail -f file
```

grep

Najbardziej przydatne opcje

- `-l` – nazwa pliku ze znalezionym wzorcem
- `-n` – numer wiersza ze wzorcem
- `-i` – nieistotna wielkość liter
- `-v` – wszystko co nie pasuje do wzorca

Przykłady użycia

Przykład 1 Wyświetl wszystkie wiersze zawierające literę *k*.

```
grep k samochody.txt
```

Polecenie `grep` wyświetli tylko wiersze zawierające małą literę *k*. Aby `grep` nie zwracał uwagi na wielkość liter, musimy użyć parametru `-i`:

```
grep -i k samochody.txt
```

Przykład 2 Wyświetl wszystkie wiersze zawierające napis *Skoda*.

```
grep Skoda samochody.txt
```

Jeżeli chcemy, aby `grep` wyświetlał wiersze, które nie zawierają zadanego wzorca, musimy posłużyć się parametrem `-v`.

```
grep -v Skoda samochody.txt
```

Przykład 3 Wyświetl wszystkie wiersze, które nie zawierają samochodu Skoda.

```
grep Skoda samochody.txt
```

Jeżeli chcemy, aby `grep` wyświetlał wiersze, które nie zawierają zadanego wzorca, musimy posłużyć się parametrem `-v`.

```
grep -v Skoda samochody.txt
```

Jeżeli wynik filtrowania chcesz uporządkować (posortować) alfabetycznie, możesz posłużyć się poleceniem `sort`.

```
grep {v Skoda samochody.txt | sort
```

Przykład 3 Wyświetl wszystkie wiersze, które nie zawierają samochodu Skoda.

```
grep Skoda samochody.txt
```

Jeżeli chcemy, aby `grep` wyświetlał wiersze, które nie zawierają zadanego wzorca, musimy posłużyć się parametrem `-v`.

```
grep -v Skoda samochody.txt
```

Jeżeli wynik filtrowania chcesz uporządkować (posortować) alfabetycznie, możesz posłużyć się poleceniem `sort`.

```
grep -v Skoda samochody.txt | sort
```

Przykład 4 Ile razy w pliku `samochody.txt` występuje samochód *SUZUKI*.

```
grep -c SUZUKI samochody.txt
```

Przykład 5 Informację o numerach wierszy, w których wystąpił wzorzec, uzyskamy za pomocą parametru `-n`.

```
grep -n Skoda samochody.txt
```

Metaznak	Znaczenie
.	Jeden dowolny znak.
[...]	Dowolny znak z wymienionych w nawiasie.
[^...]	Dowolny znak z nie wymienionych w nawiasie.
^	Na początku wiersza.
$	Na końcu wiersza.
\<	Na początku słowa.
\>	Na końcu słowa.

Tablica 1.9. Metaznaki i ich znaczenie.

```
#-f wskazuje kolumny
# , - kolejne numery, np. -f1,5
# - - zakres, np. -f1-5
#-d ogranicznik
#-c - znaki, np -c1-15,55- znaki od 1 do 15 i od 55 do konca
cut -f1,5 -d:


#s - zastepuje pierwsze wystapienie
#g - dziala globalnie
sed -e "s/shell/SHELL/g" file
#zastepowane wartosci mozna tez pobrac z pliku
sed -f plikZWzorcami file
#a plik wyglada np tak:
s/shell/SHELL/
s/c language/C Language/
#zmiana tylko w pierwszych pieciu wierszach
sed -e "1,5s/shell/SHELL/" file
#tylko wiersz z dopasowanym napisem
#dopasowywany napis jest w ukosnikach
sed -e "/unix/ s/shell/SHELL/" file
```

1.11 Wyrażenia regularne

W większości poleceń można użyć dopasowywania do wzoraca, którym może być wyrażenie bardziej skomplikowane niż pojedyńcze słowo. Do konstruowania bardziej złożonych wzorcy służą tzw. **wyrażenia regularne** budowane przy pomocy **metaznaków** (tabela 1.9).

Przykład 1 Pokazać wszystkie samochody, których nazwy składają się z co najmniej 4 znaków.

```
grep .... samochody.txt
```

Przykład 2 Pokazać wszystkie samochody, w nazwach których występuje łańcuch *OR*.

```
grep OR samochody.txt
```

Przykład 3 Wydrukuj wszystkie nazwy samochodów, w których występuje łańcuch składający się z litery *S* i po niej litery różnej od literu *U*.

```
grep S[^U] samochody.txt
```

Przykład 4 Pokazać wszystkie nazwy samochodów, które zaczynają się na literę *S*, a po niej nie występuje litera *U*.

```
grep ^S[^U] samochody.txt
```

Przykład 5 Pokazać wszystkie nazwy samochodów, które kończą się na literę *a* (bez względu na wielkość liter).

```
grep -i a$ samochody.txt
```

Przykład 6 Pokaż wszystkie osoby zamieszkałe w miejscowości Sierpc (spis osób jest w pliku `osoby.txt`).

```
grep -E '\<Sierpc\>' osoby.txt
```

Przykład 7 Pokaż wszystkie osoby zamieszkałe w miejscowości Sierpc lub Lipno (spis osób jest w pliku `osoby.txt`).

```
grep -E '\<(Sierpc|Lipno)\>' osoby.txt
```

1.12 Skrypty

Plik tekstowy zawierający listę poleceń dla interpretera nosi nazwę skryptu. Skrypty umożliwiają zapisanie często używanych sekwencji poleceń i zastąpienie ich jednym poleceniem — poleceniem uruchomienia skryptu.

1.12.1 Komentarz

Wszystkie linie rozpoczynających się od znaku `#`, traktowane są przez powłokę jako komentarz i w związku z tym są ignorowane.

1.12.2 Argumenty wywołania

Argumenty wywołania skryptu przekazywane są do powłoki jako parametry pozycyjne. Parametr `$0` zawsze wskazuje nazwę skryptu, natomiast parametry `$1`, `$2`, ... przechowują kolejne argumenty. Pamiętajmy, że nazwa skryptu nie podlega przesunięciu w wyniku działania polecenia `shift`. Poniższy przykład wypisuje dwa pierwsze argumenty przekazane do skryptu poprzedzne nazwą skryptu.

```
#!/bin/bash
echo $0 $1 $2
```

1.12.3 Metody uruchamiania skryptów

Skrypt zawiera listę poleceń, które powinień wykonać interpreter. Istnieje kilka sposobów uruchomienia skryptu. Jednym ze sposobów jest jawne wywołanie nowego interpretera z pierwszym argumentem w postaci nazwy skryptu:

```
sh skrypt arg ...
```

```
bash skrypt arg ...
```

Znacznie wygodniej jest jednak posługiwać się wyłącznie nazwą skryptu jako nazwą nowego polecenia. W tym celu należy wcześniej ustawić prawo wykonywania skryptu:

```
chmod +x skrypt
skrypt arg ...
```

Ponieważ każda powłoka stosuje inną składnię, musimy więc w jakiś sposób dokonać wyboru interpretera, który wykona skrypt. Domyślnie uruchamiana jest powłoka sh. Użytkownik może wskazać dowolny interpreter umieszczając jego pełną ścieżkę dostępu w pierwszej linii skryptu po znakach `#!`, np.: `#!/bin/bash`. W powyższych przypadkach uruchamiany jest nowy proces interpretera, który wykonuje polecenia zawarte w skrypcie, po czym kończy swoje działanie. Sterowanie wraca do procesu interpretera,

w którym wydano polecenie. Nie są w nim widoczne zmiany dokonane w skrypcie takie, jak modyfikacja zmiennych czy zmiana bieżącego katalogu.

Skrypt może być również wykonany przez bieżący interpreter i wtedy wszystkie zmiany pozostają widoczne po zakończeniu skryptu. Służy do tego polecenie . (w powłokach sh i bash) lub `source` (w powłoce bash):

```
. skrypt arg ...

source skrypt arg ...
```

Typowym zastosowaniem tej metody jest ponowne odczytanie plików konfiguracyjnych powłoki po wprowadzeniu modyfikacji:

```
. .bashrc

source .cshrc
```

Skrypt można wreszcie uruchomić zamiast bieżącego interpretera, posługując się poleceniem `exec`:

```
exec skrypt arg ...
```

W tym przypadku proces interpretera, który ma wykonać skrypt, zastępuje bieżący interpreter. Po zakończeniu działania skryptu nie istnieje już proces, w którym wydano polecenie. Sterowanie przechodzi więc do procesu nadrzędnego (macierzystego) lub następuje wylogowanie z systemu.

1.13 Język powłoki

1.13.1 Elementarz

Tworzenie zmiennych

```
#!/bin/bash
echo Teraz wstawimy zmienna
x=2
echo Nasza zmienna ma wartosc $x
```

Pobieranie danych od użytkownika

```
#!/bin/bash
echo "Podaj swoje imie:"
read imie
echo "Masz na imie $imie, lepiej nie mozna bylo :-)"
```

Proste operacje na zmiennych

```
#!/bin/bash
x=2
y=4
z=$((x+y))
echo "Twoja pierwsza liczba to: $x"
echo "Twoja druga liczba to: $y"
tekst="zmienna tekstowa"
echo "Suma tych liczb wynosi $z"
echo "Natomiast zmienna tekstowa: $tekst"
```

1.13.2 Pętle i polecenia sterujące

Zestaw poleceń powłoki obejmuje typowych pętle i polecenia sterujące. Ponieważ w większości poleceń tego typu można podawać warunki których spełnienie lub nie decyduje o dalszej akcji i warunki te zapisywane są w taki sam sposób niezależnie od polecenia, dlatego informacje o nich umieszczono w osobnym punkcie: punkt .

W poleceniu **if** wykonywana jest gałąź, dla której ostatnie polecenie z listy warunków zwraca status zerowy.

```
if lista_warunków then
   lista_poleceń
elif lista_warunków then lista_poleceń
...
else
   lista_poleceń
fi
```

Przykład.

```
#!/bin/bash
if [ $1 = fajnie ]
then
  echo "arg nr 1 = fajnie"
else
  echo "arg nr 1 != fajnie"
fi
```

```
#!/bin/bash
echo -n "Podaj liczbe wieksze od zera: "
read liczba
```

```
if [ $liczba -gt 2 ]
then
  echo "Podales dobra liczbe. $liczba jest wieksza od zera"
fi
```

```
#!/bin/bash

echo -n "Podaj liczbe z przedziaulu (0,10):"
read liczba

if [ $liczba -gt 0 ] && [ $liczba -lt 10 ]
then
  echo -e "\tPodales dobra liczbe. $liczba nalezy do przedziau (0,10)"
else
  echo -e "\tOszukales. $liczba nie nalezy do tego przedzialu"
fi
```

Pętla **for** może przyjąć jedną z dwóch postaci. Postać pierwsza:

```
for zmienna [in słowo ...]
do
   lista_poleceń
done
```

W takim przypadku wykonuje się raz dla każdego słowa z listy. Jeśli lista zostanie pominięta, pętla wykonuje się raz dla każdego ustawionego parametru pozycyjnego.

Postać druga:

```
for ((wyrażenie1; wyrażenie2; wyrażenie3))
do
   lista_poleceń
done
```

Przykład.

```
#!/bin/bash
for f in *
do
  echo $f
done
```

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8
do
  echo "Liczba $i"
done
```

```
#!/bin/bash
echo "ten skrypt ma $# parametrow"
j=0
echo "parametry to: $@"
for i in $@
do
  j=`expr $j + 1`
echo "param $j: " $i
done
```

Pętla **while** wykonywana jest dopóki ostatnie polecenie z listy warunków zwraca status zerowy[7].

```
while lista_warunków
do
   lista_poleceń
done
```

Przykłady.

```
#!/bin/bash
liczba=1
while [ $liczba -le 10 ]
do
  echo "Liczba ma wartosc: $liczba"
  liczba=$[liczba+1]
done
```

Pętla **until** wykonywana jest dopóki ostatnie polecenie z listy warunków zwraca status niezerowy.

```
until lista_warunków
do
   lista_poleceń
done
```

[7] Czyli kończy się powodzeniem.

Przykłady.

```
#!/bin/bash

liczba=1
until [ $liczba -ge 10 ]
do
  echo -e "Liczba ma wartosc: $liczba \a"
  liczba=$[liczba+1]
done
```

Polecenie **case** stara się dopasować słowo kolejno do każdego wzorca i wykonuje listę poleceń związaną z pierwszym pasującym wzorcem.

```
case słowo in
wzorzec [|wzorzec] ...) lista_poleceń;;
...
esac
```

Przykłady.

```
#!/bin/bash

echo -n "Podaj cyfre dla dnia tygodnia: "
read cyfra

case "$cyfra" in
  "1") echo -e "\tPoniedzialek" ;;
  "2") echo -e "\tWtorek" ;;
  "3") echo -e "\tSroda" ;;
  "4") echo -e "\tCzwartek" ;;
  "5") echo -e "\tPiatek" ;;
  "6") echo -e "\tSobota" ;;
  "7") echo -e "\tNiedziela" ;;
  *) echo "To nie to"
esac
```

Pętla **select** jest narzędziem ułatwiającym utworzenie często wykorzystywanej w skryptach funkcjonalności jaką jest menu. Wygeneruje ona z listy słów następujących po `in` ponumerowane menu, w którym każdej pozycji odpowiada kolejna liczba począwszy od 1. Poniżej menu znajduje się znak zachęty gdzie wpisujemy cyfrę odpowiadająca wybranej przez nas pozycji w menu. Jeśli nic nie wpiszemy i wciśniemy `ENTER`, menu będzie wyświetlone ponownie. To co wpisaliśmy zachowywane jest w zmiennej `REPLY`. Gdy odczytane zostaje `EOF` (ang. *End Of File*) czyli znak końca pliku (`CTRL+D`)

to select kończy pracę. Pętla działa dotąd dopóki nie wykonane zostaje polecenie `break` lub `return`.

```
#!/bin/bash

select liczba in A B C D
do
  echo "Wybrano $REPLY"
  echo "Koniec pokazu"
  break
done
```

```
#!/bin/bash

echo "Wybierz cos:"
select x in A B C D
do
  case "$x" in
    "A") echo "Wybrales A" ;;
    "B") echo "Wybrales B" ;;
    "C") echo "Wybrales C" ;;
    "D") echo "Wybrales D" ;;
    *) echo "Nic nie wybrales"
  esac
  break
done
```

Pętle for, while i until mogą być przerywane za pomocą poleceń: `break [n]` i `continue [n]`. Polecenie `break` przerywa wykonywanie pętli. Polecenie `continue` przerywa wykonywanie bieżącej iteracji pętli i wznawia następną iterację. Obydwa polecenia przerywają n poziomów zagnieżdżonych pętli, gdy podany jest argument. Domyślnie przyjmowane jest $n = 1$.

Zakończenie działania powłoki umożliwia polecenie: `exit [status]`. Argument oznacza status zakończenia zwracany przez powłokę. Domyślnie zwracany jest status zerowy oznaczający zakończone powodzeniem działanie aplikacji.

1.13.3 Sprawdzanie warunków

Polecenia sterujące if, while i until wykorzystują jako warunek działania status zakończenia innego, zasadniczo dowolnego, polecenia systemu. W większości przypadków sprawdzamy jednak „zwykłe” warunki np. porównujemy dwie liczby. Takich możliwości dostarcza wbudowany w powłokę

mechanizm sprawdzania warunków w postaci „naturalnej” oraz polecenie `test`, które można wywołać na dwa sposoby:

```
test warunek
```

```
[ warunek ]
```

We wszystkich przypadkach składnia specyfikowania warunku jest podobna. Istnieje możliwość sprawdzania atrybutów plików, porównywania ciągów znaków i liczb całkowitych. Liczby rzeczywiste traktowane są jak ciągi znaków.

Testowanie atrybutów pliku

- `-r plik` – sprawdza, czy użytkownik posiada prawo do czytania,
- `-w plik` – sprawdza, czy użytkownik posiada prawo do pisania,
- `-x plik` – sprawdza, czy użytkownik posiada prawo do wykonywania,
- `-f plik` – sprawdza, czy plik to plik zwykły,
- `-d plik` – sprawdza, czy plik to katalog,
- `-c plik` – sprawdza, czy plik to plik specjalny znakowy,
- `-b plik` – sprawdza, czy plik to plik specjalny blokowy,
- `-p plik` – sprawdza, czy plik to plik FIFO,
- `-s plik` – sprawdza, czy plik to plik o niezerowej długości.

Testowanie ciągów znaków

- `s1` – sprawdza, czy ciąg jest niezerowy,
- `s1 = s2` – sprawdza, czy ciągi są identyczne,
- `s1 != s2` – sprawdza, czy ciągi są różne,
- `-z s1` – prawda jeśli długość napisu `s1` wynosi zero,
- `-n s1` – prawda jeśli długość napisu `s1` jest różna od zera.

Porównywanie liczb całkowitych

- `n1 -eq n2` – sprawdza czy $n1 = n2$,
- `n1 -ne n2` – sprawdza czy $n1 \neq n2$,
- `n1 -gt n2` – sprawdza czy $n1 > n2$,
- `n1 -ge n2` – sprawdza czy $n1 \geq n2$,
- `n1 -lt n2` – sprawdza czy $n1 < n2$,
- `n1 -le n2` – sprawdza czy $n1 \leq n2$.

Wypisywanie kolejno argumentów wywołania skryptu.

```
#!/bin/bash

while [ $# -gt 0 ]
do
    echo $1
    shift
done
```

Warunki można ze sobą łączyć korzystając z następujących operatorów logicznych:

- `-a` – operator AND,
- `-o` – operator OR,
- `( )` – grupowanie warunków.

1.14 Przykładowe skrypty

Skrypt 1

Skrypt kopiuje wszystkie pliki posiadające zadane rozszerzenie `ext` do katalogu `dir` pod warunkiem, że ich rozmiar przekroczył wartość `size`. Na zakończenie wypisuje łączny rozmiar skopiowanych plików oraz sumę rozmiarów wszystkich plików o zadanym rozszerzeniu.

Wywołanie

```
nazwaSkryptu ext size dir
```

gdzie `ext` jest rozszerzeniem pliku, `size` - granicznym rozmiarem a `dir` katalogiem do którego zostaną przeniesione pliki.

```
#!/bin/bash
if [ $# -ne 3 ]
then
  echo "Zla ilosc parametrow"
  echo "Wywolanie:"
  echo "nazwaSkryptu ext size dir"
  echo "ext - rozszerzenie"
  echo "size - rozmiar (prog)"
  echo "dir - katalog docelowy"
else
  flaga=0
  size=0
  sizeBig=0
  totalSize=0
  if [ -e $3 ]
  then
    rm -rf $3
  fi
  mkdir $3
  for f in `ls -l | grep .$1$ | sed -e "s/[ ]\+/ /g" | cut -f5,8 -d' '`
  do
    if [ $flaga -eq 0 ]
    then
      size=$f
      flaga=1
    else
      if [ -f $f ]
      then
        if [ $size -gt $2 ]
        then
          bigSize=`expr $bigSize + $size`
          cp $f $3
        fi
        totalSize=`expr $totalSize + $size`
      fi
      flaga=0
    fi
  done
fi

echo "Laczny rozmiar plikow typu $1 wynosi " $totalSize
echo "Laczny rozmiar plikow przekraczajacych prog $2 wynosi " $bigSize
```

1.15 Zadania do samodzielnego rozwiązania

1. Wypisać listę użytkowników aktualnie zalogowanych.
2. Sprawdzić nazwę katalogu bieżącego.
3. Pokazać pomoc o poleceniu `ls`.
4. Wyświetlić listę procesów.
5. Jak zmienić hasło do swojego konta?
6. Wypisać wszystkie pliki w kolumnie, z informacją o czasie ostatniej modyfikacji, posortowane w/g tego czasu.
7. Wypisać wszystkie pliki z wszystkich katalogów zaczynające się od „ba".
8. Wyświetlić wszystkie pliki z bieżącego katalogu posortowane w/g rozmiaru.
9. Utworzyć katalog o nazwie `spi`. Wejść do katalogu, wyjść z niego. Usunąć nowo utworzony katalog.
10. Utworzyć katalog o nazwie `spi2` a w nim katalog `spi3`, a w nim katalog o nazwie `spi4`. Usunąć wszystkie katalogi jednym poleceniem.
11. Utworzyć katalog o nazwie `poziom1` a w nim katalog `poziom2`. W katalogu `poziom2` utworzyć plik `skasuj_mnie`. Usunąć katalogi `poziom1` i `poziom2` jednym poleceniem, wraz z plikiem `skasuj_mnie`.
12. Utworzyć katalog o nazwie `katalog ze spacjami`. Skasować nowo utworzony katalog.
13. Wyjaśnić różnicę pomiędzy ścieżką absolutną i względną.
14. Utworzyć plik zawierający polecenia utworzenia katalogów `kat1` i `kat2` w katalogu domowym. Zapisać plik na dysku. Wykonać zapisany plik. Sprawdzić czy katalogi zostały utworzone a następnie wykonać ponownie ten sam plik. Przekierować standardowy strumień błędów do pliku `bledy.txt`. Sprawdzić działanie przekierowania.

15. Utworzyć plik zawierający polecenie wylistowania zawartości pliku `/etc/passwd` i przekierować wynik (standardowy strumień wyjścia) do pliku `copypass.txt`.

16. Wylistować w formie długiej zawartość katalogu nadrzędnego w stosunku do katalogu domowego i wyświetlić tylko te wiersze, które zawierają ciąg „zu".

17. Znaleźć wszystkie pliki i katalogi w katalogu bieżącym zawierające w sobie literę „a".

18. Znaleźć wszystkie pliki i katalogi na dysku zawierające w swojej nazwie ciąg znaków „nowa wiadomość".

19. Znaleźć wszystkie pliki na dysku, których nazwa zaczyna się literą „a" i kończy literą „r" i ich nazwa jest dłuższa niż 3 znaki, wyświetlić metodą pozwalającą przeglądać wyniki strona po stronie z opcją cofania.

20. Znaleźć wszystkie pliki na dysku zaczynające się literą „a" i kończące literą „s" lub lierą „r". Zignorować wszystkie błędy podczas wyszukiwania (błędy nie wypisują się na ekran).

21. Napisać skrypt, który pobierze od użytkownika liczbę i wyświetli na ekranie jej kwadrat.

22. Sprawdź środowisko zdefiniowane po zalogowaniu się do systemu. Zorientuj się jaki jest Twój przypisany interpreter poleceń. Sprawdź jaka zmienna wskazuje nazwę (Twojego) użytkownika systemu i jak jest ta nazwa. Sprawdź jaka zmienna wskazuje nazwę Twojego katalogu domowego. Utwórz skrypt, który wyświetli te informacje: nazwę użytkownika, nazwę katalogu domowego, nazwę przypisanego interpretera poleceń.

23. Napisać polecenie, które skopiuje wszystkie pliki należące do użytkownika o rozmiarze większym od zadanego rozmiaru do katalogu o nazwie `bigFiles`.

24. Wykorzystując program `grep` wyświetl wiersz Twojego konta z pliku `/etc/passwd`. Następnie użyj polecenia `cut` aby z wybranego wiersza wyselekcjonować samo imię.

25. Napisz skrypt, który sprawdzi aktualny dzień i godzinę (polecenie `date`) i jeśli wypada on w czasie Twoich ćwiczeń, wyświetli komunikat: „*Miłej pracy.*" a jeśli nie jest to czas ćwiczeń to wyświetli „*Odpoczywasz?*".

26. Napisz polecenie, które wybierze, posortuje i wydrukuje nazwy wszystkich użytkowników systemu.

27. Napisz skrypt, który będzie kopiował plik podany jako pierwszy argument do wszystkich katalogów podanych jako kolejne argumenty wywołania.

28. Napisać skrypt, który w plikach o zadanym rozszerzeniu zamieni litery z małych na duże.

29. Napisz skrypt/polecenie, które spowoduje, że zostaną utworzone kopie wszystkich plików o zadanym rozszerzeniu, ale ich nazwa (łącznie z rozszerzeniem) będzie pisana dużymi literami (przydatne może okazać się polecenie `tr`).

30. Co robi następujące polecenie

```
ls -l | sed -e "s/[aeio]/u/g"
```

Rozdział 2

LaTeX

2.1 Podstawy

2.1.1 Historia

Zanim powiemy czym jest LaTeX, parę słów należy poświęcić systemowi TeX stworzonemu przez Donalda E. Knutha a przeznaczonemu zasadniczo do składu tekstów oraz wzorów matematycznych. Knuth rozpoczął pracę nad TeXem w 1977 roku. Zasadniczym jego motywem była chęć odwrócenia tendencji do pogarszania się jakości typograficznej, co uwidaczniało się także w jego własnych książkach i artykułach. Chodziło o to aby stworzyć narzędzie do cyfrowego składu publikacji, które nadzoruje za użytkownika proces składu dokumentu, dbając o przestrzeganie wszelkich zasad poprawnego składania tekstu.

W używanej obecnie postaci TeX został udostępniony w roku 1982, a niewielkie rozszerzenie, dotyczące ośmiobitowego kodowania znaków, pojawiło się w roku 1989. TeX ma opinię programu nadzwyczaj stabilnego, pracującego na różnego rodzaju platformach sprzętowych oraz praktycznie wolnego od błędów. Numery wersji TeXa zbiegają do liczby π, a obecny wynosi 3,14159.

LaTeX jest systemem składu dokumentów (zestawem instrukcji (poleceń, makrodefinicji, makr)) umożliwiającym złożenie i wydrukowanie dokumentów spełniających bardzo wysokie standardy typograficzne na najwyższym poziomie. Choć można go wykorzystać do przygotowywania dowolnego rodzaju dokumentów, poczynając od prostych listów, a kończąc na pokaźnych objętościowo książkach to jednak zasadniczym jego przeznaczenime jest tworzenie publikacji naukowych i technicznych o wysokiej jakości typograficznej.

Pierwszą wersję LaTeXa opracował Leslie Lamport. Do formatowania dokumentu LaTeX używa programu TeX. Pielęgnacją dzisiejszych wersji LaTeXa zajmuje się Frank Mittelbach. Najnowsza wersja pakietu, zawierająca wiele ulepszeń oraz unifikacji dla odróżnienia od pozostałych nazywa się LaTeX2ε.

Słowo TeX należy wymawiać [tech]. Zgłoska [ch] ma związek z tym, że znak X przypomina grecką literę „chi" (χ). *Tex* jest też pierwszą sylabą greckiego słowa *texnologia* (technologia). W sytuacjach, w których nie można zapisywać nazwy TeX z charakterystycznym obniżeniem litery E, należy zamiennie używać wersji TeX.

Słowo LaTeX należy wymawiać [la-tech] (bardziej właściwa dla języka polskiego) albo [lej-tech] (ta wersja częściej słyszana jest u obcokrajowców). Jeżeli nie można zapisać symbolu LaTeX, to zamiennie należy użyć zapisu LaTeX. LaTeX2ε wymawiamy [la-tech dwa i], a zamienną wersją zapisu jest LaTeX2e.

Ponieważ w niniejszym dokumencie jedynie bardzo krtótkie wprowadzamy w zagadnienia związane z systemem LaTeXdlatego po więcej informacji odsyłamy do bardzo bogadtch źródeł dostępnych zasadniczo w Internecie.

2.1.2 Filozofia pracy

Praca z LaTeXem zasadniczo różni się od podejścia stosowanego w procesorach tekstu typu WYSIWYG[1], takich jak MS Word albo OpenOffice. Oto zamiast tzw. *formatowania wizualnego* mamy tutaj do czynienia z tzw. *formatowaniem logicznym.*

Używając programów typu WYSIWYG, autor decyduje interakcyjnie (to znaczy na bieżąco podczas pisania) o **wyglądzie graficznym dokumentu**. Przez cały czas pracy nad dokumentem widzi na ekranie, jak tekst wygląda i jednocześnie jak będzie wyglądał po wydrukowaniu.

Używając LaTeXa, zwykle nie można oglądać dokumentu w jego ostatecznej postaci i jednocześnie dopisywać tekstu. W każdej chwili można go „podejrzeć" ale wymaga to jego **przetworzenia**. W procesie przetwarzania, na podstawie poleceń LaTeXa zawartych w tekście, nadawany jest graficzny charakter dokumentu (dobierane są automatycznie odpowiednie czcionki, tworzone odpowiednie wcięcia, odstępy itd.).

[1] ang. *What you see is what you get* (dostaniesz to, co widzisz).

2.1.3 Pliki źródłowe, polecenia LaTeXa i komentarze

2.2 Tworzymy pierwsze dokumenty

2.2.1 Struktura pliku źródłowego

Najprostszy dokument

```
\documentclass{article}
% kodowanie: latin2, utf8 lub cp1250
\usepackage[OT4]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{polski}
\begin{document}
Male jest piekne.
\end{document}
```

Klasy dokumentu

article

report

book

letter

Opcje wpływające na sposób działania klasy

10pt, 11pt, 12pt Ustalenie stopnia pisma dla tekstu głównego dokumentu. Domyślną wartością jest `10pt`.

a4paper, letterpaper, ... Ustalenie wymiarów papieru. Wartością domyślną jest `letterpaper`. Inne dopuszczalne wartości to: `a5paper`, `b5paper`, `executivepaper` i `legalpaper`.

fleqn Składanie wyeksponowanych wzorów matematycznych od lewego marginesu zamiast domyślnego centrowania.

leqno Umieszczanie numerów wzorów matematycznych na lewym marginesie zamiast domyślnie na prawym.

titlepage, notitlepage Pierwsza z opcji powoduje, że LaTeXskłada tytuł (instrukcja `maketitle`) oraz streszczenie (instrukcja `abstract`) na odzielnej stronie, druga – że skład tekstu zaczyna się na stronie tytułowej. W klasie `article` tytuł i streszczenie nie są domyślnie składane na oddzielnych stronach, podczas gdy w stylach `report` i `book` – są.

onecolumn, twocolumn Skład jedno- lub dwułamowy (dwukolumnowy).

oneside, twoside Druk na jednej lub na dwóch stronach kartki papieru. W klasach `article` i `report` domyślną opcją jest `oneside`, natomiast w klasie `book` – `twoside`. Włączenie opcji `oneside` powoduje, że LaTeX nie wyrównuje wysokości kolejnych stron, dopuszczając pewną ich zmienność.

openright, openany Wybranie pierwszej opcji powoduje, że tytuły rozdziałów będą umieszczane na stronach nieparzystych. W klasie `article` opcja nie ma znaczenia, gdyż w tej klasie nie jest zdefiniowane pojęcie rozdziału. W klasie `report` domyślną wartością jest `openany`, a w klasie `book` – `openright`.

Dodatkowe pakiety

```
\usepackage{polski}
\usepackage{latexsym}
\usepackage{amsmath}
\usepackage{amscd}
\usepackage{amssymb}
```

Tytuł, autor, data modyfikacji

```
\author{Srodowisko pracy informatyka}
\title{\LaTeX -- zajecia I}
\date{\today}
```

Generowanie strony tytułowej i spisu treści

```
\maketitle
\tableofcontents
```

Trochę bardziej rozbudowany najprostszy dokument

```
%klasa dokumentu
\documentclass{article}
%kodowanie znakow
\usepackage[OT4]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[MeX]{polski}
%pakiety z symbolami matematycznymi
\usepackage{latexsym}
\usepackage{amsmath}
\usepackage{amscd}
\usepackage{amssymb}
```

```
\author{Srodowisko pracy informatyka}
\title{\LaTeX — zajecia I}
\date{\today}

% === tutaj zaczyna sie dokument ===
\begin{document}
\maketitle
\tableofcontents
Male jest piekne.
\end{document}
```

2.2.2 Logiczny podział treści

Treści przekazywane w dokumencie możemy uczynić przystępniejszymi nadając jemu odpowiednią strukturę logiczną. I tak treść podzielić możemy na rozdziały, podrozdziały, podrozdziały podrozdziałów, akapity itd. Dodatkowo mamy możliwość użycia list numerowanych, wypunktowanych i nazywanych.

2.2.3 Wyrażenia matematyczne

W LaTeXu występuje specjalny tryb przeznaczony do składania wyrażeń matematycznych. Wzory wpisuje się między znakami `$` i `$`, między parami znaków `\\(` i `\\)` albo między `\begin{math}` oraz `\end{math}`.

Składając większe wzory, powinniśmy je wyróżniać, to znaczy wstawiać między akapitami, w osobnym wierszu. Takie wzory umieszcza się między parami znaków `\\[` i `\\]`. Do uzyskiwania wzorów numerowanych stosujemy otoczenie `equation`.

Poniższy przykład przedstawia różnicę w wyglądzie wzorów złożonych wewnątrz akapitu i w wersji wyróżnionej:

To będzie wewnątrz $\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$ akapitu a następne będzie wyróżnione

$$\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$$

Wybrane symbole matematyczne i podstawowe wyrażenia.

- $\leq$
- $\geq$

- $\subset$
- $\supset$
- $\subseteq$
- $\supseteq$
- $\in$
- $\equiv$
- α
- β
- γ
- δ
- ε
- λ
- φ
- ϕ
- ω
- $\frac{1}{2}$
- $a^{bc}d$
- $a_{bc}d$
- $x^2 + 5 = y^2 + \log_5 4$
- $\sum_{i=1}^{5} =?$
- $\lim_{n\to\infty}(n^2 - 5)$
- $\int f(x)dx$
- $\int_5^6 f(x)dx$
- $\|x + y\|$

- $x \neq y$
- $\sqrt{5}$
- $x_1, \ldots, x_n$
- $x_1 + \cdots + x_n$

2.2.4 Kroje i wielkość pisma

Kroje pisma

1. `\texttt{...}` lub `{\tt ...}` – `pismo o stałej szerokości znaków`
2. `\textbf{...}` lub `{\bf ...}` – **pismo pogrubione**
3. `\textsc{...}` – KAPITALIKI
4. `\textsl{...}` – *pismo ukośne*
5. `\textit{...}` lub `{\it ...}` – *pismo pochyłe*
6. `{\em ...}` – *pismo wyróżnione*
7. `\textit{\textbf{...}}` – ***pismo pogrubione pochyłe***

Wielkość pisma

1. `{\tiny ...}` – najmniejszy dostępny rozmiar fontu
2. `{\scriptsize ...}` – większy niż tiny
3. `{\footnotesize ...}` – większy niż scriptsize
4. `{\small ...}` – większy niż footnotesize
5. `{\normalsize ...}` – większy niż small
6. `{\large ...}` – większy niż normalsize
7. `{\Large ...}` – większy niż large
8. `{\LARGE ...}` – większy niż Large
9. `{\huge ...}` – większy niż LARGE

10. {\Huge ...} – największy dostępny rozmiar fontu

2.2.5 Spis treści

```
\begin{thebibliography}{99}
\bibitem{cokt} A. B. Cos, A. Ktos, {\em O czyms},
Wydawnictwo Nieznany, Pcim, 1999.
\bibitem{ully} E. Ulomny, A. Lysy, {\em Bla, bla, bla},
Wydawnictwo DNC, Raciazek, 2001.
\end{thebibliography}
```

Bibliografia

[1] Lowell Jay Arthur, Ted Burns, *UNIX. Programowanie w shellu*, MIKOM, Warszawa, listopad 1998.