Wprowadzenie do Linuxa

Marek Grochowski

4 stycznia 2019

Spis treści

1 Wstęp

1.1 Trochę historii UNIX-a i .. wolnego oprogramowania

1969 pierwszy UNIX z powłoką (*ang. shell*), edytorem tekstu, pisany w kodzie maszynowym na komputery architektury PDP-7 i PDP 9, Ken Thompson, Dennis Richie, Bell Labs, firma AT&T, New Jersey, USA
1972 druga edycja UNIX-a zawierająca potoki (*ang. pipe*)
1973 jądro systemu w języku C (Dennis Richie) - UNIX staje się przenośny
1975 wprowadzenie UNIX-a (szóstej wersji) na uczelnie do zastosowań naukowych, m.in. do Kalifornijskiego Uniwersytetu Berkeley
1977 powstaje BSD (Berkeley Software Distribution) - m.in. edytor ex (Bill Joy), kompilator Pascala
1983 System V - pierwszy komercyjny UNIX (AT&T)
Blokada źródeł UNIX-a, początki ruchu na rzecz wolnego oprogramowania (*ang. open source*)
1983 rozpoczęcie pracy nad GNU (Gnu is Not Unix), Richard Stallman (MIT), wszystko przez drukarkę Xerox
1983 Richard Stallman (MIT) tworzy Free Software Fundation - celem jest stworzenie wolnego systemu operacyjnego
1984 wydanie 4.2BSD zawierający np. TCP/IP (początki internetu)
W międzyczasie mnożą się komercyjne jak i darmowe odmiany UNIX-a - brak standardu
BSD + System V = Solaris (Sun)
1988 specyfikacja POSIX.1 w odpowiedzi Single UNIX Specification
1989 pierwsza wersja licencji GNU GPL (Ogólna Publiczna Licencja)
1990 na zamówienie MS powstaje Xenix - pierwszy UNIX dla PC
1991 Linus Torvalds i jądro Lunixa
1992-1994 procesy sądowe AT&T i Novel wstrzymują rozwój kodu z Berkley, jednak na bazie BSD powstają FreeBSD i NetBSD
1994 powstaje Red Hat Linux (Linux rozpowszechniany w dystrybucjach)
Drzewo genealogiczne UNIX-a

źródło: Wikipedia

1.2 Budowa i własności systemu UNIX

Główne cechy systemu Unix:

- wielozadaniowość - system z podziałem czasu, pozwala na uruchamianie wielu procesów jednocześnie
- wielodostępowość - umożliwia pracę z wieloma użytkownikami

Budowa Unixa:

- **jądro** (*ang. kernel*) - niskopoziomowe oprogramowanie obsługujące sprzęt, dostarczające określone usługi dla programów użytkowych (realizuje system plików, planuje przydziału pracy procesora, zarządza pamięcią i urządzeniami zewnętrznymi, inicjuje działanie systemu, zapewnia mechanizmy komunikacji, dostarcza zestawu wywołań systemowych)
- **powłoka** (*ang. shell*) - interpreter poleceń, pozwala na komunikację użytkownika z urządzeniami i procesami, uruchamianie programów i nadzorowanie ich pracą - najpowszechniejsze powłoki to: sh, ksh, csh, tcsh, bash
- biblioteki systemowe
- oprogramowanie

Pliki w Unixie:

- plik jest ciągiem bajtów
- pliki są zorganizowane w postaci drzewa wyrastającego z korzenia `/`. Do każdego pliku możemy dostać się podając ścieżkę od korzenia (ścieżka bezwzględna) lub względem bieżącego katalogu (ścieżka względna)
- katalogi też są plikami, zawierają informację o innych plikach (katalogach), które się w nich znajdują. Każdy katalog zawiera plik o nazwie `.` (kropka) będący odniesieniem do tego katalogu oraz plik o nazwie `..` (dwie kropki) będący odnośnikiem do katalogu położonego wyżej
- urządzenia zewnętrzne (drukarki, terminale, dyski itp.) oraz kanały komunikacji międzyprocesorowej reprezentowane są za pomocą plików "specjalnych" umieszczonych w katalogu `/dev`)
- działające procesy również dostępne są w postaci plików w katalogu `/proc`
- nazwy plików są dowolne, nie dłuższe niż 255 znaków. Pliki których nazwy rozpoczynają się od kropki (np. `.tcshrc`) są plikami ukrytymi
- każdy plik jest własnością określonego użytkownika oraz jest skojarzony z pewną grupą użytkowników. Dla każdego pliku istnieją jasno określone uprawnienia dostępu (odczytu, edycji i uruchamiania) dla każdego użytkownika.
- wszystko w UNIXie jest plikiem

Struktura katalogów:

`/` korzeń drzewa katalogów

`/bin` katalog zawierający najważniejsze polecenia systemowe (np. `/bin/ls`, `/bin/cp`, etc.)

`/home` katalogi domowe użytkowników (np. `/home/marek` to katalog domowy użytkownika `marek`)

`/lib` najważniejsze biblioteki (np. `/lib/libc.so` - biblioteka języka C, `/lib/modules/` - moduły jądra, itp.)

`/root` katalog domowy administratora systemu

`/mnt` najczęściej używane miejsce do montowania nośników (systemów plików), np. dyskietek, dysków, płyt cdrom

`/etc` globalne pliki konfiguracyjne (np. `/etc/passwd` - lista użytkowników (kont))

`/dev` pliki urządzeń (np. `/dev/lp1` - drukarka, `/dev/hda1` - pierwsza partycja głównego dysku)

`/proc` pseudosystem plików z informacjami o procesach (np. `/proc/cpuinfo` - inf. dotyczące procesora)

Ścieżka do pliku:

- bezwzględna - od korzenia drzewa
  przykład: `/usr/share/local/`
- względna - poczynając od bieżącego katalogu
  przykład: `../../usr/share/local/`

1.3 Rozpoczynamy pracę

Przed rozpoczęciem pracy w systemie Unix należy posiadać konto, czyli przydzielony identyfikator i hasło dostępu. Każdy użytkownik ma określone prawa dostępu do zasobów systemu. Zasady te ustala administrator (*ang. root*) czyli super użytkownik mający (nieomal) nieograniczoną władzę nad systemem.

Serwery dostępne dla studentów WFAiIS:

- `ferm7.fizyka.umk.pl` - (ferm) serwer aplikacji dostępny dla studentów (dostęp wyłącznie z sieci lokalnej)
- `polon7.fizyka.umk.pl` - (polon) serwer aplikacji dostępny dla studentów (dostęp wyłącznie z sieci lokalnej)
- `ameryk.fizyka.umk.pl` - serwer dostępowy (ssh, scp, poczta) dostępny z internetu

Regulamin sieci LAN można znaleźć pod adresem http://wwwold.fizyka.umk.pl/fizyka/?q=node/141

Zdalna sesja w trybie tekstowym.
Logowanie do powłoki linuksowej z systemu Windows możliwe jest za pomocą programu `Putty`, który obsługuje bezpieczny protokół `ssh`. Po uruchomieniu programu należy w odpowiednim miejscu podać adres serwera `ferm.fizyka.umk.pl`. Rozpoczynamy pracę logując się do systemu podając identyfikator (*ang. login*) oraz hasło (*ang. password*) po czym terminal powinien przywitać nas linią zachęty w postaci:

```
student@ferm:~$
```

Korzystając z powłoki linuksowej (dostępnej np. w środowisku Cygwin) zalogujemy się za pomocą polecenia `ssh`.

```
$ ssh identyfikator@ferm.fizyka.umk.pl
```

Po zakończeniu pracy wydajemy polecenie `logout` lub `exit`.

Zdalna sesja w trybie graficznym:
Chcąc uruchomić graficzne aplikacje na zdalnym serwerze można skorzystać z programu `Cygwin` (środowisko linuksowe w systemie Windows). Połączenie X-serwera ze zdalnym serwerem uzyskujemy wydając w powłoce Cygwina polecenie:

```
$ X -query ferm.fizyka.umk.pl :8
```

Praca w trybie graficznym jest również możliwa za pośrednictwem VNC. W tym celu należy najpierw po zalogowaniu na wybrany serwer za pomocą ssh wydać komendę:

```
$ vncserwer
```

Polecenie uruchomi pulpit identyfikowany za pomocą liczby całkowitej. Przy pierwszym uruchomieniu zostaniemy poproszeniu o podanie hasła, które będzie używane przy łączeniu za pomocą aplikacji klienckiej. Teraz połączenie do pulpitu możliwe jest za pomocą dowolnej aplikacji klienckiej VNC (np. vncwiewer, RealVNC, itp.), gdzie w polu adresu należy podac adres serwera wraz z numerem pulpitu podanym przy uruchomieniu serwera, np:. `ferm.fizyka.umk.pl:13`
Zmiana hasła:
Zmiana hasła na serwerze `ferm` dokonywana jest poprzez formularz dostępny na stronie UCI pod adresem http://www.uci.umk.pl/studenci/konto/korzystanie/. Na serwerach wydziałowych hasło jest uaktualniane raz na dobę (około godz. 2).

Zasady nadawania hasła:

- co najmniej 9 znaków, w tym przynajmniej jedna duża litera i znak specjalny lub cyfra
- prZeMIesZanE DUżE i małe litery oraz cyfry i znaki specj@1ne
- nie podawać swoich danych osobistych, daty urodzenia itp.
- nie należy stosować słów które można znaleźć w słowniku wyrazów polskich lub angielskich
- nie używać prostych sekwencji np.: qwerty, 123456

Hasło powinno stanowić pozornie przypadkowy ciąg znaków ale powinno dać się łatwo zapamiętać. Przykłady haseł: `t@Jn3|ha51o` albo `s2Um1_d0kola=la5`

2 Podstawowe polecenia

2.1 Anatomia polecenia

Ogólna postać poleceń wydawanych w powłoce wygląda tak:

```
polecenie [-opcje]... [argumenty]...
```

gdzie `polecenie` jest nazwą programu (polecenia), który chcemy uruchomić (program powinien znajdować się w jednym z katalogów zawartych w zmiennej systemowej `$PATH`, jeśli tak nie jest to musimy podać pełną ścieżkę do danego polecenia, np. `/bin/ls`)

`-opcje` to ciąg znaków poprzedzony myślnikiem. Opcje modyfikują działanie programu (polecenia). Na przykład `ls -t` wyświetli listę plików w kolejności posortowanej względem czasu modyfikacji.

`argumenty` to elementy na których operuje polecenie (np. nazwy plików, ciągi znaków). Na przykład `ls /bin` wyświetli listę plików w katalogu `/bin`.

Notacja stosowana w dokumentacji zakłada, że zawartość nawiasów kwadratowych `[]` jest opcjonalna zaś `...` (wielokropek) oznacza, że poprzednia część polecenia może się powtarzać wielokrotnie Przykład:

```
ls [-la] [katalog]...
```

oznacza, że polecenie `ls` może być modyfikowane za pomocą opcji `-a` lub `-l` i argumentem tego plecenia może być katalog lub lista katalogów oddzielona znakiem spacji.

```
$ ls -l /usr
$ ls -a /usr /home /etc
$ ls -l -a /etc
$ ls -la /home /etc
```

2.2 Gdzie szukać pomocy?

Pomoc na temat sposobu użycia danego polecenia możemy otrzymać uruchamiając je z dodatkową opcją `-h` lub `--help`.
Przykład:
```
$ ls ---help
$ man -h
```

2.2.1 Podręcznik systemowy

W systemie znajduje się podręcznik `man` zawierający opis wszystkich dostępnych poleceń i programów, opis funkcji systemowych oraz zainstalowanych bibliotek i wiele innych przydatnych informacji.

man wyświetla strony podręcznika (manuala) dotyczące danego polecenia
Postać: `man [rozdzial] [opcje] nazwa`
Otrzymujemy opis składni i wszystkich opcji danego polecenia
Przykład:
`$ man ls`
wyświetli opis polecenia `ls` zawarty w podręczniku man.
Program `man` oferuje wiele skrótów klawiszowych ułatwiających przeglądanie zawartości podręcznika oraz wyszukiwanie wyrażeń. Szczegółową pomoc na ten temat otrzymamy wciskając przycisk `h`.

whatis przeszukuje podręcznik (opisy poleceń) w poszukiwaniu danej nazwy.
Postać: `apropos nazwa...`
Przykład:
`$ whatis ls less`
Wyświetli krótki opis poleceń `ls` i `less`.

apropos przeszukuje opisy poleceń podręcznika `man` w poszukiwaniu danego słowa (wyrażenia regularnego).
Postać: `apropos słowo_szukane`
Przykład:
`$ apropos grep`
Wyświetli opisy zawierające słowo `grep`.
Możliwe jest stosowanie wyrażeń regularnych (więcej informacji w rozdziale dotyczących narzędzi tekstowych).
Przykład:
`$ apropos '^l(..)?s$'`
Wyświetli wpisy które rozpoczynają się od litery `l`, kończą na literą `s` a pomiędzy nimi mogą wystąpić dowolne dwa znaki lub brak znaku. Więc do tego wzoru pasuje zarówno opis polecenia `ls` jak i `less`

info podręcznik GNU
Postać: `info [temat]`
Pomiędzy tematami i zagadnieniami w podręczniku `info` można poruszać się poprzez odnośniki. Najważniejsze skróty klawiszowe: `n` - przejście do następnego rozdziału, `p` - przejście do poprzedniego rozdziału, `u` - wyjście do rozdziału nadrzędnego (np. do spisu rozdziałów), `Enter` - przejście do treści wskazanej w menu przez kursor. Pełną listę możliwych poleceń otrzymamy wciskając `?`.

Przykład:
`$ info`
wyświetli spis najważniejszych tematów opisanych w podręczniku
`$ info coreutils`
wyświetli rozdział dotyczący podstawowych narzędzi dostarczonych z systemem
`$ info ls`
opis polecenia `ls`

help pomoc dotycząca poleceń wbudowanych w powłokę
Postać: `help [komenda]`
Powłoka zawiera wiele wbudowanych poleceń. Aby poznać ich listę wystarczy uruchomić polecenie `help` nie podając żadnych argumentów. O wszystkich poleceniach powłoki można też dowiedzieć się z podręcznika `man bash`. Polecenie `type` pozwala przekonać się o tym czy dane polecenie jest wbudowanym poleceniem powłoki.

```
$ type cd
cd jest wewnętrznym poleceniem powłoki
$ type date
date jest /bin/date
```

2.3 Ćwiczenia

1. Dowiedź się do czego służą polecenia: `alias`, `echo`, `rm`, `test`, `w`, `[`
2. Które z poleceń z poprzedniego ćwiczenia jest wbudownym poleceniem powłoki Bash. W jakim katalgu znajdują się pozostałe polecenia?

2.4 Zarządzanie plikami

Opis narzędzi służących do zarządzania plikami można znaleźć w dokumentacji systemowej pod hasłem `fileutils`.

2.4.1 Najważniejsze polecenia

ls wyświetla zawartość katalogu
Postać: `ls [opcje] [plik...]`
Przykład:
`$ ls`
wyświetli zawartość bieżącego katalogu
`$ ls /bin`
wyświetli listę plików w katalogu `/bin`
Polecenie `ls` może być uruchamiane z wieloma parametrami (zobacz `man ls`).
Najczęściej używanymi są:
`-l` wyświetla dokładne informacje o plikach (rodzaj pliku, uprawnienia, nazwę właściciela grupę, rozmiar, datę modyfikacji)
`-a` wyświetla wszystkie pliki, także pliki ukryte (ich nazwa zaczyna się od kropki)
`-s` wyświetla dodatkowo rozmiar plików
`-R` rekurencyjne wyświetlanie zawartości katalogów (wraz z podkatalogami)
`-d` wyświetla katalogi a nie ich zawartość
`-t` posortowanie wyniku według czasu modyfikacji pliku

`-S` posortowanie wyniku według rozmiaru plików
`-r` odwrócenie kolejności sortowania
`-i` wyświetla numer i-węzła plików
Przykład:
`$ ls -la /etc /home`
wyświetli dokładną informację o wszystkich plikach w katalogach `/etc` i `/home`

mkdir tworzy katalog
Postać: `mkdir [-p] katalog...`
Przykład:
`$ mkdir nowykatalog`
utworzy katalog o nazwie `nowykatalog`
Najważniejsze opcje:
`-p` pozwala tworzyć "gałęzie" katalogów
Przykład:
`$ mkdir -p kat1/kat2/kat3/kat4`
utworzy cztery puste katalogi (jeden wewnątrz drugiego)

rmdir usuwa puste katalogi
Postać: `rmdir [-p] katalog...`
Przykład:
`$ rmdir nowykatalog`
usunie pusty katalog o nazwie nowykatalog
Przykład:
`$ rmdir -p kat1/kat2/kat3/kat4`
usunie całą "gałąź" pustych katalogów

cd zmienia bieżący katalog
Postać: `cd [katalog]`
Przykład:
`$ cd /usr/bin`
spowoduje przejście do katalogu `/usr/bin`
`$ cd ~`
spowoduje powrót do katalogu domowego
`$ cd ..`
przejście do katalogu położonego wyżej
`$ cd -`
powrót do ostanio odwiedzonego katalogu
`$ cd`
powrót do katalogu domowego

rm usuwa pliki
Postać: `rm [opcje] plik...`
Przykład:
`$ rm dane.txt`
usunie plik o nazwie `dane.txt`
`$ rm *.txt`

usunie wszystkie pliki z rozszerzeniem `.txt`
Najważniejsze opcje:
`-f` nie pytaj o potwierdzenie podczas usuwania
`-r` usuń rekurencyjnie, przydatne przy usuwaniu katalogów wraz z zawartością
`-i` pytaj o potwiedzenie przy usuwaniu każdeg pliku
Przykład:
`$ rm -fr katalog`
usunie cały `katalog`

cp kopiuje pliki i katalogi
Postać:
`cp plik1 plik2`
`cp plik... katalog`
`cp -r katalog1... katalog2`
Przykład:
`$ cp /etc/passwd ~/kopia_dane.txt`
tworzy kopię pliku `/etc/passwd` o nazwie `kopia_dane.txt` w katalogu domowym użytkownika
`$ cp * jakis_katalog/`
stworzy kopie plików z bieżącego katalogu w katalogu `jakis_katalog` (katalog docelowy musi istnieć)
`$ cp /etc/hosts .`
skopiuje plik `hosts` z katalogu `/etc` do bieżącego katalogu
Najważniejsze opcje:
`-r` kopiowanie rekurencyjne, pozwala kopiować katalogi z całą zawartością
Przykład:
`$ cp -r /usr/src .`
kopiuje katalog `/usr/src` do bieżącego katlogu
`$ cp -r /usr/src nowy_katalog`
kopiuje katalog `/usr/src` do bieżącego katlogu zmieniając jego nazwę na `nowy_katalog`

mv przenosi pliki
Postać:
`mv plik1 plik2`
`mv plik... katalog`
Przykład:
`$ mv dane.txt nowedane.txt`
zmienia nazwę pliku `dane.txt` na `nowedane.txt`
`$ mv *.c programy/`
przeniesie wszystkie pliki z bieżącego katalogu posiadające rozszerzenie `*.c` do katalogu `programy`

pwd wyświetla bieżący katalog
Postać: `pwd`
Przykład:
`$ pwd`

`/home/student`
Opcja `-P` powoduje wypisanie bieżącego katalogu z pominięciem dowiązań symbolicznych.

ln tworzy dowiązanie (sztywne lub symboliczne) do plików
Postać:
`ln [opcje] plik nazwa_dowiazania`
`ln [opcje] plik... katalog`
Przykład:
`$ ln dane.txt lndane.txt`
tworzy dowiązanie sztywne do pliku `dane.txt` o nazwie `lndane.txt`
`$ ln /etc/* tmp/`
tworzy dowiązania sztywne w katalogu `tmp` dla wszystkich plików z katalogu `/etc`
Uwaga: każdy plik istnieje dopóki nie usuniemy wszystkich jego dowiązań. Najważniejsze opcje polecenia `ln`:
`-s` tworzy dowiązanie symboliczne. W przeciwieństwie do dowiązania sztywnego dowiązanie symboliczne może być tworzone dla katalogów oraz dla plików położonych w obrębie innego systemu plików.

O liczbie dowiązań do pliku informuje wynik polecenia `ls -l` (druga kolumna).

```
$ ln -s /etc etc_link
$ ln /etc/passwd passwd_link
$ ls -l

lrwxrwxrwx 1 student stud   5 03-06 20:14 etc_link -> /etc/
-rw-r--r-- 2 student stud 465 2009-04-02  passwd_link
```

touch zmienia datę modyfikacji pliku lub tworzy pusty plik
Postać: `touch [opcje] plik...`
Przykład:
`$ touch nowyplik`

file wyświetla informację o zawartości pliku
Postać: `file [opcje] plik...`
Przykład:

```
$ file main.c index.html /etc/hosts
main.c:     ASCII C program text
index.html: HTML document text
/etc/hosts: ASCII text
```

du wyświetla rozmiar zajętej przestrzeni dyskowej
Postać: `du [opcje] plik...`
Najważniejsze opcje:
`-b` w bajtach
`-k` w kilobajtach
`-m` w megabajtach
`-h` w czytelnej formie
`-s` tylko objętość całkowita dla każdego argumentu

`-c` podsumowanie dla wszystkich plików
Przykład:
`$ du -ms dokumenty`
wyświetli zajętość w megabajtach katalogu `dokumenty`
`$ du -h -s -c *`
wyświetli rozmiar wszystkich plików i katalogów w bieżącym katalogu w czytelnej postaci oraz z podsumowaniem.

Inne przydatne polecenia: stat, mkfifo, lsof, shred, mknod, dd, find, rename

2.4.2 Znaki specjalne powłoki - dopasowanie nazw plików

***** dopasowanie dowolnego ciągu znaków

? dopasowanie pojedynczego znaku

[**lista**] dopasowanie jednego ze znaków z podanej listy

[^**lista**] dopasowanie jednego znaku nie należącego do listy

{ciag1,ciag2,...} rozwinięcie napisu z użyciem wszystkich kombinacji ciągów znaków

Przykłady:
```
$ ls *.txt
$ cp /etc/p*d ~
$ rm plik?.txt
$ ls /etc/[abc]*
$ ls /bin/*[a-g]
$ rm *.[^a-z]
$ mkdir katalog_{1,2,3}
$ rmdir plik_[1-4]
$ echo {Ala,Ula,Ola}" ma "{psa,kota,rybkę}.
```

2.4.3 Ćwiczenia

1. Obejrzyj zawartość katalogów `/etc`, `/proc`, `/dev`, `/home`, `/dev`, `/lib`.
2. Utwórz w swoim katalogu domowym katalogi według poniższego schematu. Spróbuj dokonać tego za pomocą jednego polecenia.

```
.
|-- katalog
|   `-- katalog
|-- Moj nowy katalog
|-- nowy_katalog
`-- raz
`-- dwa
|-- cztery
`-- trzy
```

3. Do katalogu `katalog` przekopiuj plik `/etc/passwd`
4. Do katalogu `raz/dwa/trzy` skopuj wszystkie pliki z katalogu `/etc` w których nazwach występuje litera 'a', 'b' lub 'p'.

5. W katalogu `nowy_katalog` utwórz pusty plik o nazwie `plik_testowy`
6. W katalogu `katalog` utwórz dowiązanie do pliku `nowy_katalog/plik_testowy` o nazwie `link`
7. Za pomocą edytora tekstu (np. `nano`) zmień treść pliku `nowy_katalog/plik_testowy` i zapisz w nim kilka linijek tekstu. Następnie go usuń i sprawdź zawartość pliku `katalog/link`.
8. W katalogu `nowy_katalog` utwórz dowiązanie symboliczne o nazwie `link_symb` do pliku `katalog/link`. Zmień zawartość pliku `nomy_katalog/link_symb` i zmień jego zawartość. Następnie usuń plik `katalog/link`. Na co wskazuje teraz utworzone dowiązanie symboliczne?
9. Zmień nazwę katalogu `nowy_katalog` na `stary_katalog`
10. Przenieś katalog `raz` do katalogu `stary_katalog` zmieniając jego nazwę na `jeden`
11. Usuń wszystkie utworzone w tym zadaniu katalogi i pliki.
12. Jaka jest różnica między wynikiem polecenia `ls` a `ls *` ?
13. Ile dowiązań ma pusty katalog? Ile dowiązań ma katalog głowny `/` ?

2.5 Narzędzia tekstowe

Opis narzędzi służących do wyświetlania i modyfikowania zawartości plików tekstowych można znaleźć w dokumentacji systemowej pod hasłem `textutils`.

2.5.1 Najważniejsze polecenia

cat wyświetla zawartość strumienia wejściowego lub zawartość plików
Postać: `cat [opcje] [plik...]`

Przykład:
`$ cat /etc/passwd`
wyświetli zawartość pliku `/etc/passwd`. Polecenie `cat` może też posłużyć do tworzenia plików tekstowych

```
$ cat > pliktekstowy
to jest tekst
który zostanie umieszczony
w pliku o nazwie pliktekstowy
Aby zakończyć wciśnij Ctrl+ D
```

lub do łączenia kilku plików w jedną całość - rezultat można przekierować do pliku:
`$ cat pliktekstowy dane.txt > nowy.txt`

more wyświetla zawartość pliku strona po stronie
Postać: `more [opcje] plik`
Przykład:
`$ more /etc/passwd`
wyświetli zawartość pliku `passwd`
`$ ls /bin | more`
pozwala przejrzeć listę plików w katalogu `/bin`

less wyświetl zawartość pliku strona po stronie
Postać: `less [opcje] plik`
Jest to ulepszona wersja polecenia `more` pozwalająca poruszać się po pliku zarówno w przód jak i w tył.
Przykład:
`$ less /etc/passwd`

Programy `more` i `less` posiadają wiele funkcji dostępnych za pomocą skrótów klawiszowych o których możemy się dowiedzieć wciskając `h`. Inne przydatne funkcje uzyskamy wciskając: `q` - wyjście z programu, `/`*wyrażenie* - poszukuje *wyrażenia* w pliku, `n` - szuka następnego wystąpienia.

head wyświetla początek pliku
Postać: `head [opcje] plik...`
Przykład:
`$ head /etc/passwd`
wyświetli 10 pierwszych linii pliku `passwd`
Najważniejsze opcje:
`-n liczba` wyświetli określoną liczbę początkowych linii
`-c liczba` wyświetli określona liczbę początkowych znaków
Przykład:
`$ head -c 10 /etc/passwd`
wyświetli 10 pierwszych znaków pliku `passwd`
`$ ls | head -n 3`
wyświetli nazwy trzech pierwszych plików z bieżącego katalogu

tail wyświetla koniec pliku
Postać: `tail [opcje] plik...`
Działanie i opcje takie same jak w poleceniu `head` z tą różnicą, że wyświetlane jest zakończenie pliku.
Przykład:
`$ tail -n 4 /etc/passwd`
wyświetli cztery ostatnie linie pliku `passwd`

cmp porównuje pliki znak po znaku
Postać: `cmp [opcje] plik1 plik2`
Polecenie wyświetla pozycje pierwszego napotkanego znaku (bajtu) różniącego oba pliki.
Przykład:
`$ cmp plik1.txt plik2.txt`
`plik1.txt plik2.txt różnią się: bajt 30 linia 2`
Najważniejsze opcje:
`-c` wypisuje różniące się znaki

diff znajduje różnice pomiędzy plikami taktowymi
Postać: `diff [opcje] plik1 plik2`
Przykład:
`$ diff plik1.txt plik2.txt`

Wynikiem działania jest wyświetlenie fragmentów tekstu, które są różne w obu plikach wraz z informacją jak należy zmienić pierwszy z plików aby otrzymać drugi z użyciem 3 operacji: zamień (`c`), usuń (`d`), dodaj (`a`) fragment tekstu.
Przykładowo komunikat `1,10c2,5` oznacza, że należy zamienić linie od 1 do 10 w pierwszym pliku na tekst który występuje w liniach od 2 do 5 w drugim pliku. `3a5` oznacza, że w linii trzeciej pierwszego pliku należy dodać 5 linię z drugiego pliku
Wyjście programu `diff` tworzy łatkę, którą można zaaplikować za pomocą polecenia `patch` na drugim pliku aby jego zawartość uczynić identyczną z zawartością pliku pierwszego.
Przykład:

```
$ diff plik1 plik2 > patch.txt
$ patch plik1 patch.txt
```

patch aplikuje łatkę z programu `diff` na pliku tekstowym
Postać: `patch plikoryginalny plikzłatką`

wc liczy ilość znaków, słów i linii w pliku
Postać: `wc [opcje] plik...`
Najważniejsze opcje:
`-c` drukuje liczbę znaków/bajtów w pliku
`-w` drukuje liczbę wyrazów w pliku
`-l` drukuje ilość linii w pliku
Przykład:

```
$ wc -c dane.txt
```

wyświetli ilość bajtów zajętych przez plik
Przykład:

```
$ wc -l *.txt
```

wyświetli liczbę linii we wszystkich plikach o rozszerzeniu .txt znajdujących się w bieżącym katalogu.

```
$ ls /bin/ | wc -l
```

zwróci liczbę plików w katalogu `/bin/`.

sort wypisuje posortowaną zawartość pliku tekstowego
Postać: `sort [opcje] plik...`
Przykład:

```
$ sort dane.txt > posortowane.txt
```

spowoduje posortowanie linii zawartych w pliku `dane.txt` i przesłanie wyniku do pliku `posortowane.txt`
Niektóre opcje:
`-r` odwraca kolejność sortowania
`-u` usuwa duplikaty
`-f` wyłącza rozróżnianie małych i dużych liter
`-n` sortowanie liczb (standardowo dane sortowane traktowane są jako ciągi znaków)
Przykład:

```
$ du . | sort -n
```

wyświetli listę plików w bieżącym katalogu posortowaną według rozmiaru
`+liczba` pozwala pominąć przy sortowaniu określoną liczbę pól (pola standardowo są rozdzielone białymi znakami (przestarzała wersja))
`-k poz1[,poz2]` pozwala specyfikować względem którego pola (kolumny) chcemy sortować
`-t separator` używa podanego znaku jako separatora pól (kolumn)
Przykład:

```
$ ls -l | sort +4 -n
```

wyświetli posortowaną listę plików według piątej kolumny otrzymanej za pomocą polecenia `ls -l`
`$ sort -k 5 -t : /etc/passwd`
Wyświetli posortowana listę użytkowników (piąta kolumna pliku `passwd`, gdzie kolumny sa oddzielone dwukropkami).

grep wyświetla linie pasujące do wzorca
Postać: `grep [opcje] wzorzec [plik...]`

Przykład:
`$ grep student /etc/passwd`
wyświetli linie z pliku `/etc/passwd` zawierającą słowo "student"
Często stosuje się to polecenie jako filtr w strumieniu, np:
`$ ls /bin | grep z | wc -l`
wyświetli liczbę plików z katalogu `bin` zawierających w nazwie literę "z"
Najważniejsze opcje:
`-v` wyświetlane są wiersze w których wzorzec nie pojawia się
`-l` wyświetli tylko nazwę pliku w którym znaleziono wzorzec
`-i` nie rozróżnia dużych i małych liter we wzorcu
`-A n` wyświetla także n kolejnych linii
`-B n` wyświetla także n poprzedzających linii

cut Wypisuje wybrane fragmenty linii
Postać: `cut [opcja]... [plik]...`
Niektóre opcje:
`-b N` wypisuje tylko podane bajty
`-f N` wypisuje tylko podane kolumny (standardowo separatorami kolumn są białe znaki)
`-d znak` użyj podanego znaku jako separatora kolumn
Przykład:
`$ cut -c 1 /etc/passwd`
wyświetli tylko pierwszy znak z każdej linii.
`$ cut -c 4-7 plik`
wyświetli znaki od 4-go do 7-go.
`$ cut -f 2- plik`
Wyświetli linie bez pierwszej kolumny
`$ cut -d : -f 5 /etc/passwd`
wyświetli imiona i nazwiska użytkowników (5 kolumna pliku `passwd`, gdzie kolumny oddzielone są dwukropkiem).

paste łączy linie plików
Postać: `paste pliki...`

Przykładowo:
`$ paste plik1 plik2`
wypisze na standardowym wyjścuiu połoczone zawratości obu plików.

tr Zamienia znaki wczytane ze standardowego wejścia.
Postać: `tr łańcuch1 łańcuch2`
`tr -d łańcuch`
`tr -s łańcuch`

Najważniejsze opcje:
`-d` usuń podane w łańcuchu znaki
`-s` usuń wielokrotne wystąpienia tych samych znaków
Przykład:
`$ echo $PATH | tr : ' '`
wyświetla wartość zmiennej $PATH zastępując dwukropki spacjami.
`$ echo Witaj swiecie | tr ai ia`
w podanym haśle zamienia literę 'i' na 'a' oraz literę 'a' na 'i'
`$ echo Witaj swiecie | tr [a-z] [A-Z]`
zamienia małe litery na duże
`$ cat plik | tr -d ' '`
usuwa spacje z pliku
`$ cat plik | tr -s ' '`
usuwa powtórzenia spacji w pliku

Inne przydatne polecenia i narzędzia (textutils): nano, emacs, vi, vim, awk, join, paste, tac, nl, od, split, csplit, uniq, comm, ptx, tsort, tr, fold

2.5.2 Przekierowanie standardowego wejścia i wyjścia programu

`polecenie > plik`
przekierowanie wyjścia programu do pliku (zawartość pliku zostanie nadpisana)

`polecenie >> plik`
przekierowanie wyjścia programu z dopisywaniem do pliku

`polecenie 2> plik`
przekierowanie wyjścia diagnostycznego do pliku

`polecenie >& plik`
przekierowanie wyjścia standardowego i diagnostycznego do pliku

`polecenie < plik`
przekierowanie wejścia programu z pliku

`polecenie << słowo`
przekierowanie wejścia programu z klawiatury do momentu wystąpienia danego `słowa`

Przykłady:

```
$ ls /etc > lista
$ head -n 3 /etc/passwd >≥ lista
$ haed -n 3 /etc/passwd >& lista
$ cat < lista
$ cat < lista > nowalista
$ cat << KONIEC > tekst
To jest pewien tekst
KONIEC
```

2.5.3 Potoki

`polecenie1 | polecenie2`
połączenie wyjścia programu 1 z wejściem programu 2

Przykłady:
```
$ cat /etc/passwd | wc -l
$ grep Marek /etc/passwd | cut -f 5 -d : | sort | head -n 1 > wybraniec
```

tee czyta standardowe wejście i przesyła je na standardowe wyjście oraz do pliku.
Postać: `tee [-a] plik`
Najważniejsze opcje:
`-a` dopisuje zawartość strumienia wyjściowego do pliku (bez tej opcji zawartość pliku zostałaby nadpisana)
Przykład:
`$ grep Marek /etc/passwd | tee plik1.txt | wc -l`
zapisze linie z pliku `/etc/passwd` zawierające słowo `marek` w pliku `plik1.txt`, zaś na ekranie wyświetlona zostanie ilość tych linii.

2.5.4 Podstawianie wyników polecenia

Uruchomienie polecenia w postaci `$(polecenie)` powoduje podstawienie standardowego wyjścia polecenia w miejsce wywołania. Identyczny działenie można uzyskać róznież umieszczając polecenie pomiędzy znakami `` ` `` (pochyłe "uszy"). Pozwala to miedzy innymi na zapisanie wyniku programu w zmiennej, np,:
`$ a=$(ls /bin)`
lub uzycie wyjścia polecenia jako argumentów innego polecenia, np,:
`$ echo `ls /bin``
Chcąc usunąc pliki, których nazwy zawarte są w pliku tekstowym, można to osiągnąc poleceniem:
`$ rm $(cat lista_plikow.txt)`

2.5.5 Ćwiczenia

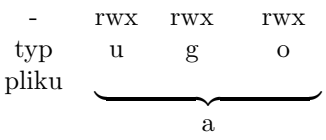
1. Korzystając z polecenia `cat` utwórz krótką notatkę tekstowa w pliku `tekst.txt`.
2. Korzystając z polecenia `cat` skopuj plik `tekst.txt` pod nazwą `tekst2.txt`
3. Korzystając z polecenia `cat` oraz pliku `/etc/passwd` utwórz plik `users.txt` zawierający posortowaną listę użytkowników (imiona i nazwiska). Wyświetl użytkowników których nazwiska kończą się literami `ski`.
4. Wyświetl plik (katalog) z twojego katalogu domowego o największej objętości
5. Wyświetl całkowitą liczbę plików znajdujących się w katalogach wymienionych w zmiennej `$PATH`
6. Polecenie `ps -A` wyświetla listę wszystkich uruchomionych procesów. Wykorzystaj to polecenie do stworzenia pliku `procesy.txt` zawierającego listę posortowanych i niepowtarzających się nazw działających procesów
7. Połącz wszystkie pliki tekstowe `*.txt` z bieżącego katalogu w jeden plik o nazwie `calosc.txt`

2.6 Uprawnienia

W systemie Unix/Linux każdy plik posiada właściciela i grupę do której jest przyporządkowany oraz zestaw uprawnień definiujących dostęp właściciela, grupy oraz wszystkich pozostałych użytkowników. Uprawnienia te dotyczą możliwości odczytu (*ang. read*), zapisu (*ang. write*) oraz wykonywania (*ang. execute*) plików. Informacje o uprawnieniach plików uzyskamy wydając polecenie `ls -l`
Przykład:
`$ ls -la`

```
drwxr-xr-x  3 marek users 4096 sty  5 03:40 .
drwxr-xr-x  9 marek users 4096 sty  4 23:34 ..
-rwxr-xr-x  1 marek users 7312 sty  4 23:54 a.out
-rw-r--r--  1 marek users  572 sty  4 23:54 main.c
drwxr-xr-x  2 marek users 4096 sty  5 03:40 wynik
```

W kolejnych kolumnach otrzymujemy następujące informacje:
10 znaków określających typ i uprawnienia pliku, ilość dowiązań do pliku, nazwa właściciela, nazwa grupy, rozmiar w bajtach, data i godzina modyfikacji oraz nazwa pliku.



Pierwszy znak oznacza typ pliku: `d` to katalog, `-` to zwykły plik.
Następne 9 znaków określa uprawnienia pliku (`r` - odczyt,`w` - zapis, `x` - wykonywanie) odpowiednio dla trzech zbiorów użytkowników: dla właściciela, dla grupy oraz dla wszystkich innych. Z powyższego przykładu wynika więc, że plik `main.c` może być odczytywany przez wszystkich ale jego zawartość może zmieniać tylko właściciel `marek`. Plik `a.out` może zostać uruchomiony przez wszystkich (nadane uprawnienie `x`). Plik `wynik` jest katalogiem (d) i wszyscy mogą odczytywać jego zawartość (wyświetlać listę plików w katalogu) oraz mają prawa wykonywania (x), czyli mają dostęp do katalogu ale tylko właściciel może zmieniać zawartość tego katalogu (tworzyć lub usuwać pliki w tym katalogu).
Poniższe polecenia pozwalają na zmianę uprawnień plików:

chmod zmienia prawa dostępu do pliku

Postać: `chmod [opcje] prawa plik...`
Polecenie służy do ustawiania praw odczytu (r), zapisu (w) i wykonywania (x) pliku. Prawa te można nadać jednemu z trzech zbiorów użytkowników: właścicielowi pliku (u), grupie (g) i całej reszcie (o). Można też zmienić prawa wszystkim użytkownikom (a). Możliwe są trzy operacje: (+) dodanie uprawnień, (-) cofnięcie uprawnień lub (=) zastąpienie starych uprawnień nowymi.
Składnia zmiany uprawnień powinna wyglądać tak: `[[ugoa...][+-=][rwx...],...]`

Przykład:
`$ chmod a+r dane.txt`
nadanie prawa do odczytu dla wszystkich
`$ chmod u-x *.sh`
cofnięcie prawa do wykonywania przez właściciela plików o rozszerzeniu `.sh`

`$ chmod ug=r dane.txt`
nadanie prawa do odczytu przez właściciela i grupę
`$ chmod ugo=rwx dane.txt`
nadanie wszystkim uprawnień do odczytu, zapisu i wykonywania pliku - równoważne poleceniu
`$ chmod a=rwx dane.txt`
Najważniejsze opcje:
`-v` wyświetla komunikat o każdym zmienionym uprawnieniu
`-R` zmień uprawnienia katalogów i całej ich zawartości
Przykład:
`$ chmod -R a+r`
zezwoli wszystkim na możliwość czytania wszystkich plików w domowym katalogu

Innym sposobem nadawania uprawnień jest zapis numeryczny, np:
`$ chmod 703 dane.txt`
Pierwsza cyfra określa uprawnienia użytkownika, druga - grupy a trzecia - reszty. Wartość (od 0 do 7) oznacza rodzaj uprawnień: wykonywanie (1), zapis (2) lub odczyt (4). W celu nadania kilku uprawnień należy zsumować odpowiednie cyfry. W powyższym przykładzie 7 oznacza nadanie praw do odczytu, zapisu i wykonywania (4+2+1) dla właściciela, 0 oznacza, że grupa nie posiada żadnych uprawnień a 3 odpowiada nadaniu prawa do zapisu i wykonywania (2+1) dla pozostałych użytkowników.

chown zmienia właściciela i grupę pliku
Postać: `chown [opcje] użytkownik[:grupa] plik...`

chgrp zmienia grupę użytkowników pliku
Postać: `chgrp [opcje] grupa plik...`

Uwaga: polecenia `chown` oraz `chgrp` mogą byż niedostępne dla zwykłego użytkonika.

2.6.1 Ćwiczenia

1. Sprawdź ustawienia plików `/etc/passwd`, `/etc/shadow`.
2. Sprawdź uprawnienia katalogów domowych pozostałych użytkowników systemu.
3. Ustaw uprawnienia swojego katalogu domowego (łącznie z całą zawartością) tak abyś tylko ty miał możliwość przeglądania zawartości.
4. Utwórz katalog `~/public_html` i umieść w nim plik o nazwie `index.html`. Ustaw uprawnienia tak aby możliwe było wyświetlenie twojej strony domowej. Serwer WWW na serwerze `ferm` jest tak skonfigurowany, że strona znajdująca się w katalogu `public_html` jest dostępna pod adresem `http://www.fizyka.umk.pl/~identyfikator`.

2.7 Procesy

System UNIX pozwala na uruchomienie wielu procesów działających równocześnie.
Wydanie polecenia zazwyczaj uruchamia proces na pierwszym planie, chcąc uruchomić kolejny program użytkownik musi zaczekać na zakończenie działającego procesu.
Istnieje możliwość uruchomienia wielu procesów działających w tle. Powłoka uruchomi proces w tle

jeśli na końcu polecenia dodamy `&`.
Przykład:
`$ sleep 100 &`
P olecenie `sleep` uruchamia proces który nic nie robi i kończy działanie po określonej liczbie sekund.
Do działającego procesu można wysłać sygnał aby zakończył swoje działanie albo się zatrzymał.
W przypadku działającego na pierwszym planie programu można:
- zakończyć działanie procesu wciskając `Ctrl+C`
- zatrzymać proces wciskając `Ctrl+Z`, proces można wówczas przywrócić do działania w tle lub na pierwszym planie za pomocą poleceń powłoki `bg` i `fg`.

2.7.1 Najważniejsze polecenia

ps podaje informacje o działających procesach
Postać: `ps [opcje]`

Opcje polecenia `ps` moga byc podawane w różnej formie: poprzedzone myślnikiem (w stylu UNIX) np. `ps -l`, bez myślinika (styl BSD), np. `ls aux` lub poprzedzone dwoma myślnikami (styl GNU), np. `ps --help`. W razie niepewności zajrzyj do dokumentacji `man ps`.
Najważniejsze opcje:
`-l` wyświetla więcej informacji o procesach
`-f` format ekstra-pełny
`-A` lub `e` wszystkie procesy
`-U numer_użytkownika` lub `U`, lub `--user` procesy uruchomione przez danego użytkownika
`u` format zorientowany na użytkownika
`f` wyświetl drzewo procesów

Przykład:

```
$ ps
PID TTY          TIME CMD
3873 pts/2    00:00:00 su
3875 pts/2    00:00:00 bash
3893 pts/2    00:00:00 tcsh
3899 pts/2    00:00:00 vim
3904 pts/2    00:00:00 ps
```

PID (*ang. Process ID*) jest liczbą jednoznacznie identyfikującą proces w systemie. TTY to nazwa terminala z którego zostało uruchomione polecenie `CMD`.

```
$ ps -l
```

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1002  3873  3211  0  76   0 -   710 wait   pts/2    00:00:00 su
0 S  1002  3875  3873  0  75   0 -  1271 wait   pts/2    00:00:00 bash
0 S  1002  3893  3875  0  75   0 -   908 rt_sig pts/2    00:00:00 tcsh
0 T  1002  3899  3893  0  78   0 -   997 finish pts/2    00:00:00 vim
0 R  1002  3945  3893  0  76   0 -   581 -      pts/2    00:00:00 ps
```

Oznaczenia: UID - numer użytkownika, PPID - numer procesu rodzica (procesu z którego wywodzi się dany proces), PRI - priorytet, NI - wartość parametru NICE ustawianego poleceniem `nice`

top lista procesów w czasie rzeczywistym
Postać: `top [opcje]`
Polecenie `top` prezentuje listę działających procesów wraz z najważeniejszymi informacjami na temat obciążenia systemu (zajętość pamięci i obciążenie CPU). Program udostępnia wiele skrutów klawiszoych pozwalających na sortowanie i filtrowanie listy procesów:

- `h` wyświetla listę skrótów klawiszowych
- `k` zabija wskazany proces
- `r` zmienia parametr NICE wskazanego proesu
- `f` wybór pól prezentujących informacje o procesach
- `q` wyjście z programu

Polecenie:
`$ top -u username`
wyświetla listę procesów wskazanego użytkonika.

htop lista procesów w czasie rzeczywistym
Postać: `htop [opcje]`
Jest to bardizej przyjazna dla użytkownika wersja programu `top`

pstree wyświetla drzewo procesów
Postać: `pstree [opcje]`
Najważniejsze opcje:
`-p` dodaje numery PID
`-H PID` wyróżnia podany proces (`-h` aktualny proces)

kill zabija proces
Postać: `kill [-sygnal] PID...`
Polecenie wysyła sygnał do procesu o numerze PID. Jeżeli nie sprecyzujemy rodzaju sygnału wówczas `kill` wysyła sygnał `SIGINT` przerywający działanie procesu.
Opcja `-l` wyświetla listę sygnałów jakie można przesłać do procesów (więcej na ten temat w dokumentacji `man 7 signal`)
Najważniejsze sygnały:
`2 SIGINT` przerwanie procesu (ten sygnał jest wysyłany do procesu gdy wciśniemy `Ctrl+C`)
`9 SIGKILL` sygnał "zabicia" procesu
`19 SIGSTOP` zawieszenie procesu (`Ctrl+Z`)
`18 SIGCONT` wznowienie zatrzymanego procesu
`$ kill -9 3899`
wysyła sygnał `KILL` o numerze 9 do podanego procesu
`$ kill -9 -1`
wysyła sygnał `KILL` do wszystkich procesów (uwaga: spowoduje wylogowanie, gdyż zamknięty zostanie także proces powłoki)
`$ kill -SIGSTOP 4008`
zawiesza działanie procesu
`$ kill -18 4008`
wznawia działanie zawieszonego procesu
Wbudowanw w powłokę Bash polecenie `kill` umożliwia dodatkowo wystyłanie sygnałów do procesów identyfikowanych za pomocą numeru zadania JID (zobacz polecenie jobs)

killall zabija procesy o podanej nazwie
Postać: `killall [-s sygnal] nazwa ...`
Przykład:
`$ killall sleep`

pgrep wyświetla numery PID procesów pasujących do wzorca
Postać: `pgrep [opcje] wyrażenie`
Najważniejsze opcje:
`-u użytkownik` zawęża wyniki tylko do procesów danego użytkownika.
`-l` dodaj informację o nazwie procesu
Przykład:
`$ pgrep -u marek sleep`
wyświetli numery procesów uruchomionych przez użytkownika `marek` zawierających w nazwie `sleep`.
`$ kill -9 $(pgrep -u marek sleep)`
wyśle syknał `SIGKILL` do procesów uruchomionych przez użytkownika `marek` zawierających w nazwie `sleep`.

pkill wysyła sygnał do procesów o nazwach pasujących do wzorca
Postać: `pgrep [-sygnał] [opcje] wyrażenie`
Polecenie to rozszerza działanie `pgrep` o możliwość wysłania sygnału do pasujących procesów.
Przykład:
`$ pkill -9 -u marek sleep`
wyśle sygnał `SIGKILL` do procesów uruchomionych przez użytkownika `marek` zawierających w nazwie `sleep`.

jobs podaje status procesów uruchomionych w bieżącej powłoce
Postać: `jobs [-l]`
Przykład:

```
$ jobs
[1]-  Done                    sleep 100
[2]+  Stopped                 vim
```

W nawiasie kwadratowym podany jest numer zadania `JID` (*ang. Job ID*), obok stan procesu (zatrzymany, uruchomiony, itp.).
Opcja `-l` wyświetla dodatkowo `PID` procesu.

bg uruchamia zawieszone zadanie w tle
Postać: `bg [JID...]`

Gdy nie podano numery zadania to wznawiane jest ostatio zawieszone zadanie.
Przykład wznowienia zadania o numerze JID 2:
`$ bg %2`

fg uruchamia zatrzymane zadanie na pierwszym planie
Postać: `fg [JID]`
Przykład:

`$ fg %2`
spowodowałoby przeniesienie programu `vim` z poprzedniego przykładu na pierwszy plan

nice uruchamia program z zadanym priorytetem
Postać: `nice -priorytet [opcje] polecenie`
Przykład:
`$ nice +19 emacs`
uruchomi program `emacs` z parametrem `NICE` równym 19. Im większa wartość `NICE` tym mniej zasobów będzie pochłaniało wykonanie procesu, czas jego trwania ulegnie wydłużeniu dając pierwszeństwo procesom z mniejszym priorytetem.

renice pozwala zwiększyć priorytet działającego procesu
Postać: `renice priorytet [PID] [-u użytkownik]`
Opcja `-u` pozwala zmienić priorytet procesom uruchomionym przez danego użytkownika.
Przykład:
`$ renice 19 4343 -u student`
zwiększy priorytet do 19 procesu o numerze `PID` 4343 oraz wszystkim procesom uruchomionym przez użytkownika `student` .

at uruchamia proces o zadanym czasie
Postać: `at [-f plik] CZAS`
Polecenie lub lista poleceń do uruchomienia wczytywana jest ze standardowego wejścia, lub po opcji `-f` możemy podać nazwę pliku w który zawarta jest lista poleceń do uruchomienia. `CZAS` może być wyrażony w wielu formach (po szczegóły zajrzyj do podręcznika `man at`). Wynik działania polecenia wysyłany jest do skrzynki pocztowej użytkownika. Przykład:
`$ at 10:21 -f plik`
spowoduje uruchomienie poleceń zawartych w pliku `plik` o godzinie 10.21
`$ echo ls | at now +1minutes`
spowoduje uruchomienie polecenia `ls` dokładnie za minutę

atq wyświetla listę zadań ustawionych do wykonania za pomocą polecenia `at`
Postać: `atq`

atm usuwa z kolejki zadanie o podanym numerze
Postać: `atm numer_zadania`

crontab program zarządzający tabelami demona `cron`, który służy do wykonywania zaplanowanych w czasie operacji
Postać: `crontab [-e|-l|-r]`
Opcje:
`-e` edycja tabeli zadań
`-r` usunięcie bieżącej tabeli zadań
`-l` wyświetla bieżącą tabele zadań
Tabela zadań powinna zawierać w każdej linii wpis następującej postaci:
`MINUTY GODZINY DNI MIESIACE DNI_TYGODNIA polecenie`
gdzie `MINUTY` to liczba z zakresu 0-59, `GODZINY` - liczba z zakresu 0-23, `DNI` - liczba z zakresu 1-31, `MIESIACE` od 1 do 12, `DNI_TYGODNIA` od 1 do 7. Użycie gwiazdki `*` zamiast liczy oznacza dowolną wartość.
Przykładowa tabela zadań:

```
30   18   *   *       *      rm -f ~/tmp
30   0    1   1,6,12  *   mail student@fizyka.umk.pl < wiadomosc.txt
```

Powyższy zapis oznacza, że codziennie o godzinie 18:30 wykonywana jest komenda usuwająca katalog ∼/tmp, zaś 30 minut po północy w pierwszy dzień stycznia, czerwca i grudnia wysyłany jest za pomocą poczty elektronicznej plik `wiadomosc.txt` do podanego użytkownika. Więcej szczegółów w podręczniku demona `cron` i polecenia `crontab`.

nohup uruchamia polecenie, które nie zostanie przerwane w momencie wylogowania
Postać: `nohup polecenie`
Przykład:
`$ nohup sleep 1000 &`

fuser identyfikuje procesy używające plików lub gniazd sieciowych
Postać: `fuser [opcje] plik...`
Polecenie `fuser` wyświetla listę numerów procesów PID, które wykorzystują w danym momencie wskazany plik, system plików lub gniazdo sieciowe.
Najważniejsze opcje:
`-k` zabij procesy zamiast wypisywać ich numery PID
`-u` podaj nazwę właściciela procesu wraz z numerem PID
`-v` wyświetl więcej szczegółów na temat procesów
Przykład:
`$ fuser /home`
wypisze numery procesów używających plików w katalogu `/home`

Inne przydatne polecenia: fuser, iotop, atd, pidof, lsof

2.7.2 Katalog `/proc`

W systemach UNIXopodobnych występuje katalog `/proc` zawierający informacje o procesach. Każdy proces jest tu reprezentowany w postaci katalogu o nazwie odpowiadającej numerowi PID. Wewnątrz każdego z nich można znaleźć pliki opisujące status danego procesu, np.: `cmdline` zawiera pełną linie uruchomionego polecenia, `stat` zawiera informacje o stanie procesu (z tego pliku korzysta `ps`), `status` - informacje o stanie procesu w bardziej przystępnej formie, itp.
Przykład:
`$ cat /proc/1231/status`
wyświetli inf. o stanie procesu 1231.

2.7.3 Ćwiczenia

1. Polecenie `xclock -update 1` uruchamia w środowisku graficznym zegar z sekundnikiem. Uruchom zegar na pierwszym planie, zawieś jego działanie za pomocą `Crtl+Z` i wznów działanie w tle.
2. Wyślij sygnał `SIGSTOP` do procesu wyświetlającego zegar.
3. Uruchom w tle polecenie `xclock -update 1`
4. Wyślij sygnał `SIGCont` do wstrzymanego procesu wyświetlającego zegar.
5. Wyświetl wszystkie uruchomione przez siebie procesy.
6. Zabij procesy związane z wyświetlaniem zegarów.

2.8 Informacje o systemie i użytkownikach

date podaje datę i czas systemowy
Postać: `date [opcje] [format]`

cal wyświetla kalendarz
Postać: `cal [opcje]`

printenv wyświetla zmienne środowiskowe
Postać: `printenv [zmienna]`
Domyślnie polecenie wyświetli listę wszystkich zmiennych środowiskowych.
Przykład:
`$ printenv PATH`
wyświetli wartość przypisaną do zmiennej `PATH`.

tty wyświetla nazwę terminala
Postać: `tty`

whoami kim jestem
Postać: `whoami`

id informacje o użytkowniku - GID, UID itp.
Postać: `id [opcje] [użytkownik]`

groups nazwy bieżących grup
Postać: `groups [użytkownik]`

finger informacje o użytkowniku.
Postać: `finger [użytkownik]`

who lista zalogowanych użytkowników
Postać: `who [opcje]`

uname informacje o systemie
Postać: `uname [opcje]`

hostname nazwa hosta
Postać: `hostname [opcje]`

df infromacje o zajęctości zamontowanych dysków
Postać: `df [opcje]... [plik]...`
Polecenie wypisuje listę wszystkich zamontowanych systemów plików wraz z informacją o ich zajętości i miejscu zamontowania.
`$ df`

```
Filesystem     1K-blocks      Used Available Use% Mounted on
udev             3997676         0   3997676   0% /dev
tmpfs             806444      3368    803076   1% /run
/dev/sda3       95596964  87020120   3677672  96% /
tmpfs            4032208         0   4032208   0% /sys/fs/cgroup
/dev/sdb1         507904     38664    469240   8% /boot/efi
/dev/sda5      546370624 491924808  26621944  95% /home
```

Najważniejsze opcje:
`-m` zajętość w MB
`-k` zajętość w KB
`-h` zajętość w formie czytelnej (wartość z odpoiednim przyrostkiem B, KB, MB, GB)
`-T` wyświetla inforamcje o typie systemu plików (np. ext4, NTFS, ...)

Jeżeli argumentem polecenia jest plik lub katalog to wyświetlana jest zajętość dysku na którym ten plik rezyduje.

```
$ df -h -T /
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/sda3      ext4   92G   83G  3,6G  96% /
```

free informacje o zajętości pamięci w systmie
Postać: `free [opcje]`
Polecenie wyświetla informację o całkowitej zajętej pamięci fizycznej oraz pamięci wymiany. Opcje `-b`, `-k`, `-m`, `-g` pozwalają określić w jakich jednostkach wyświetlane są wartości (B, kB, MB i GB odpowiednio). Opcja `-h` wypisze wartości w czytelnej formie dodając odpowiedni przyrotek okreslający rozmiar.
Przykład:

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:           7,7G        3,0G        1,4G        219M        3,3G        4,2G
```

Inne przydatne polecenia: users, w, dnsdomianname, chfn

2.9 Wyszukiwanie plików

which wyszukuje położenie programu w katalogach ze zmiennej `$PATH`
Postać: `which polecenie`
Przykład:

```
$ which find
/usr/bin/find
```

whereis wyszukuje (wszystkie) położenia plików binarnych, źródłowych i stron podręcznika danego polecenia
Postać: `whereis polecenie...`
Przykład:

```
$ whereis find
find: /usr/bin/find /usr/man/man1/find.1.gz
```

find szuka plików w drzewie katalogów

Postać: `find [katalog] [wyrażenie]`

Argumentem polecenia jest `katalog` w którym chcemy odnaleźć plik określony za pomocą `wyrażenia`.

Najważniejszymi opcjami stosowanymi w wyrażeniu są:

`-name nazwa_pliku` znajdź plik o podanej nazwie

`-iname nazwa_pliku` znajdź plik o podanej nazwie (nie rozróżnia wielkości liter)

`-group nazwa_grupy` plik należy do danej grupy

`-user nazwa_użytkownika` właścicielem pliku jest użytkownik

`-type [f|d|l|b]` typ pliku: f - zwykły plik, d- katalog, l - link, b - plik blokowy

`-atime [+|-]liczba` plik był otwierany określoną liczbę godzin temu

`-mtime [+|-]liczba` plik był modyfikowany określoną liczbę dni temu

`-size [+|-]liczba[c|k|M|G]` plik o określonym rozmiarze (c - bajty, k - kilobajty, M - megabajty, G - gigabajty

Znak + lub - przed `liczba` oznacza poszukiwanie odpowiednio większej lub mniejszej wartości

Przykład:

```
$ find dane/ -name plik.txt
```

szuka pliku `plik.txt` w katalogu `dane`

```
$ find ~-name '*.jpg' -user kazik
```

znajdzie wszystkie pliki o rozszerzeniu .jpg w domowym katalogu należące do użytkownika `kazik`

```
$ find . -mtime -2
```

znajdzie pliki w bieżącym katalogu które były modyfikowane w ciągu ostatnich dwóch dni

```
$ find /usr -iname '[a-d]*' -user root -type f -size -2M
```

wyszuka w katalogu `/usr` pliki o nazwie zaczynającej się od liter a, b, c lub d, których właścicielem jest `root` i które maja rozmiar nie większy od 2 megabajtów

W momencie znalezienia pliku spełniającego dane wyrażenie można wykonać określoną akcje (standardowo jest to `-print` czyli wyświetlenie lokalizacji pliku

Dodając opcje `-exec` możemy wykonać dowolne polecenie. Takie polecenie musi być zakończone znakami \;

```
$ find . -name '*.txt' -exec echo znalazłem \;
```

wyświetli komunikat `znalazłem` dla każdego znalezionego pliku

Aby wykonać polecenie na znalezionym pliku należy dodać {}, w to miejsce zostanie wstawiona jego nazwa

```
$ find . -name '*.txt' -exec rm -f '{}' \;
```

spowoduje usuniecie wszystkich znalezionych plików

Dużo więcej na temat polecenia `find` można znaleźć w dokumentacji `man`

locate wyszukiwanie plików o podanej nazwie

Postać: `locate [opcje]... [plik...]` Polecenie przeszukuje bazę danych wcześniej zindeksowanych plików w poszukiwaniu informacji o położeniu pliku, którego nazwa (lub ścieżka) zawiera podany wzorzec. Należy jednak pamiętać, że w zależności od tego jak dawno temu przeprowadzane było indeksowanie plików, lista plików może być nieaktualna i może zawierać wpisy o plikach, które już zostały usunięte lub może nie wynajdywać plików, które zostały utworzone przed uaktualnieniem bazy danych.

2.10 Inne narzędzia

echo wyświetla linię tekstu

Postać: `echo [opcje] ciąg_znaków`

Przykład:

```
$ echo witaj świecie
witaj świecie
```

Polecenie `echo` pomaga zobaczy co zostanie wstawione w miejsce znaków specjalnych *, ?, []

```
$ echo rm -f unix.*
rm -f unix.aux unix.dvi unix.gz unix.log unix.out unix.pdf unix.tex
```

printf wypisuje sformatowany tekst

Postać: `printf format [argumety]...`

yes wyświetla w nieskończoność dany ciąg znaków

Postać: `yes [ciąg_znaków]`

expr oblicza wartość wyrażenia matematycznego

Postać: `expr wyrażenie`

Liczby i operatory muszą być oddzielone spacjami

```
$ expr 2 + 2
4
$ expr 2 '*' 3
6
```

Znak * z powodu swojego specjalnego znaczenia musi być zawarty w cudzysłowie

seq wyświetla sekwencję liczb

Postać: `seq [opcje] [początek] [krok] koniec`

Przykład:

```
$ seq 3 2 10
3 5 7 9
```

xargs buduje i uruchamia polecenia powłoki na podstawie tekstu ze standardowego wejścia

Postać: `xarg [polecenie]`

Przykład:

```
$ find ~-name '*.mp3' | xargs du -sm
```

uruchomi polecenie `du -sm` podając jako argument nazwy plików przekazane w strumieni wyjściowym przez polecenie `find`.

bc kalkulator o dowolnej precyzji

Postać: `bc [plik]`

Kalkulator `bc` wykonuje obliczenia arytmetyczne dostarczone w strumieniu wejściowym. Kalkulator wspiera wszystkie operatory arytmetyczne, logiczne i oparatory relacji w takiej samej postaci jak w języku C. Dodatkowo znak ^ pełni rolę operatora potęgowania.

Przykład:

```
$ echo "2 ^ 10" | bc
1024
```

Domyślnie obliczenia realizowane są z dokładnością do liczb całkowitych.

Precyzję (ilość cyfr po przecinku) określamy za pomocą zmiennej `scale`.

```
$ echo "1/3" | bc
0
$ echo "scale=20; 1/3" | bc
.33333333333333333333
```

2.11 Edytor strumieniowy sed

sed Edytor strumieniowy

Postać: `sed [-n] [-e skrypt] [opcja]... [plik]...`

Odczytuje kolejne linie ze strumienia wejściowego (lub pliku), dokonuje edycji zgodnie z podanym skryptem i wynik wyświetla na standardowym wyjściu.
Najważniejsze opcje:
`-n` hamuje normalne wyjście (wyświetlanie tylko linii wskazanych w skrypcie komendą `p`)
`-e` wykonają podany skrypt (pojedyncze polecenie). Jeśli podajemy tylko jedną komendę ta opcja nie jest wymagana.

Składnia skryptu:
`[adres[,adres]] funkcja [argumenty]`
`adres` to numer linii pliku (`$` oznacza numer ostatniej linii) lub wyrażenie regularne umieszczone pomiędzy znakami `/`
. Określa on zakres linii strumienia na których będę dokonywane operacje. Na przykład `1,3` pasuje do pierwszych trzech linii, `/bash/` pasuje do wszystkich linii zawierających wyrażenie `bash`, zaś `/begin/,/end/` dotyczy wszystkich kolejnych linii z których pierwsza zawiera słowo `begin` a ostatnia słowo `end`. `funkcja` do wyboru mamy wiele możliwości edycji strumienia. Najważniejsze to:
`a tekst` dodaj podany tekst przed następną linią
`c tekst` zamień linię podanym tekstem
`d` usuń linię
`i tekst` wstaw podany tekst
`p` wyświetl bufor (aktualnie edytowaną linię)
`s/wyrażenie/łańcuch/` zastępuje podanym `łancuchem` pierwsze znalezione w buforze `wyrażenie`
`s/wyrażenie/łańcuch/g` zastępuje podanym `łancuchem` wszystkie znalezione w buforze `wyrażenia`
`=` wyświetla numer linii
Przykłady:
`$ sed -n '1p' plik`
wyświetli pierwsza linię pliku
`$ sed -n '3,$p' plik`
wyświetli wszystkie linie od 3-ciej to końca pliku
`$ sed '3,$d' plik`
usunie wszystkie linie od 3-ciej do końca pliku
`$ sed -n '/Marek/p' /etc/passwd`
wyświetla linie zawierające słowo `Marek` z pliku `/etc/passwd`
`$ sed '/UNIX/c Linux' plik`
Zamienia linie w których występuje słowo `UNIX` zwrotem `Linux`
`$ sed -n '/UNIX/=' plik`
wyświetla numery linii w których występuje wyrażenie `UNIX`
`$ sed 's/UNIX/Linux/g plik`
zamienia wszystkie wystąpienia słowa `UNIX` na `Linux`

`$ sed -n 's/UNIX/Linux/g plik`
tak jak wyżej ale wyświetlane są wyłącznie linie w których nastąpiła zmiana

2.12 Wyrażenia regularne

Wybrane metaznaki wyrażeń rozszerzonych wyrażeń regularnych (POSIX ERE, *ang. Extended Regular Expressions*)
`[lista]` pasuje do pojedynczego znaku z danej listy
`[^lista]` pasuje do znaku nie podanego na liście
`.` (kropka) pasuje do dowolnego pojedynczego znaku
`\w` jest równoważne `[0-9a-zA-Z]` lub `[[:alnum:]]`, czyli zastępuje dowolna literę lub cyfrę
`\W` oznacza to samo co `$[^[[:alnum:]]]`
`^` i `$` to odpowiednio początek i koniec linii
`\<` oraz `\>` początek i koniec słowa
Po wyrażeniu regularnym mogą stać operatory powtórzenia:
`?` poprzedzający element pasuje zero lub jeden raz, np. `miark?a` pasuje do `miarka` ale też `miara`
`*` poprzedzający element pasuje zero lub więcej razy, np `W*in` pasuje zarówno do słowa `Windows` jak i do `Linux`
`+` poprzedzający element pasuje jeden lub więcej razy,
`{n}` poprzedzający element pasuje dokładnie `n` razy
`|` operator LUB, np. `Fizyka|fizyka` pasuje do `fizyka` oraz `Fizyka`
`()` grupowanie, np. `fizy(ka|cy)` pasuje zarówno do `fizyka` i `fizycy`.
Uwaga: w podstawowych wyrażeniach regularnych (POSIX BRE *ang. Basic Regular Expressions*) stosowanych w większości narzędzi UNIXowych metaznaki `?`, `+`, `{}`, `()`, `|` tracą swoje szczególne znaczenie; zamiast nich należy użyć `\?`, `\+`, `\{\}`, `\(\)`, `\|`.
Przykłady:
`grep 'bash$' /etc/passwd`
linie w których występuje wyraz rozpoczynający się literą `a` lub `A`
`grep '^From: ' /var/mail/$USER`
lista odebranej poczty (linie rozpoczynające się słowem `From:`)
`grep -v '^$' plik`
wszystkie linie, które nie są puste
`grep '[0-9]\{9\}' plik`
dziewięciocyfrowe ciągi liczb, np. numery telefonów
`grep '(.\+) plik'`
psuje do dowolnego ciągu składającego się przynajmniej z jednego znaku zawartego w nawiasach

2.13 Narzędzia sieciowe

host podaje informacje o hoście
Postać: `host [opcje] adres`
Standardowo `host` tłumaczy nazwy domen na adresy IP i na odwrót. Przykład:
```
$ host ferm
158.75.5.47
$ host 127.0.0.1
localhost
```

ping wysyła pakiet testowy do wybranego hosta
Postać: `ping adres`

Pakiet próbny jest wysyłany aż nie przerwiemy procesu za pomocą Ctrl-C. Przykład:

```
$ ping www.google.pl
$ ping 127.0.0.1
```

traceroute wyświetla trasę pokonywana do danego hosta

Postać: `traceroute [opcje] nazwa_hosta`

Przykład:

```
$ traceroute www.google.com
traceroute: Warning: www.google.com has multiple addresses; using 64.233.183.99
traceroute to www.l.google.com (64.233.183.99), 30 hops max, 38 byte packets
1 phys-to-torman (158.75.5.190) 0.242 ms 0.297 ms 0.231 ms
2 * * *
3 war-b2-pos11-0.telia.net (213.248.68.53) 8.753 ms 8.797 ms 8.856 ms
4 ffm-bb1-pos6-3-2.telia.net (213.248.96.21) 33.833 ms 33.679 ms 33.705 ms
5 ffm-b2-link.telia.net (213.248.69.93) 34.974 ms 34.810 ms 34.968 ms
6 google-111945-ffm-b2.c.telia.net (213.248.69.86) 33.758 ms 34.057 ms 33.071 ms
7 72.14.238.119 (72.14.238.119) 51.698 ms 40.944 ms 41.185 ms
MPLS Label=162573 CoS=0 TTL=1 S=1
8 64.233.175.246 (64.233.175.246) 44.573 ms 43.413 ms 44.210 ms
9 216.239.43.42 (216.239.43.42) 43.819 ms 45.157 ms 44.443 ms
10 216.239.43.34 (216.239.43.34) 44.962 ms 44.667 ms 64.233.183.99 (64.233.183.99)
44.965 ms
```

write wysyła wiadomość tekstową do użytkownika

Postać: `write uzytkownik[@adres]`

talk program do interaktywnej rozmowy z użytkownikiem

Postać: `talk uzytkownik[@adres]`

mesg zablokowanie możliwości komunikacji poleceniami `talk` i `write`

Postać: `mesg [n|y]`

mail wysyłanie poczty elektronicznej

Postać: `mail uzytkownik[@adres] [-s temat] [-c adres innego adresata]`

Przykład:

```
$ mail grochu@ferm
```

Chcąc wysłać treść zawartą w pliku `plik_tekstowy` można jego zawartość umieścić w strumieniu wejściowym programu `mail`, np:

```
$ cat plik_tekstowy.txt | mail grochu@ferm
$ mail grochu@ferm < plik_tekstowy.txt
```

telnet połączenie ze zdalnym komputerem

Postać: `telnet uzytkownik[@adres]`

Ze względów bezpieczeństwa dziś rzadko używany, wyparty przez szyfrowane połączenie `ssh`.

ssh szyfrowane połączenie ze zdalnym komputerem

Postać: `ssh uzytkownik[@adres]`

Pozwala na uruchomienie zdalnie powłoki, np: `$ ssh unix@158.75.5.136`
lub uruchomienie polecenia na zdalnej maszynie, np: `$ ssh unix@158.75.5.136 ls`
wyświetla listę plików w katalogu domowym zdalnej maszyny

ftp połączenie z serwerem FTP pozwalającym na przesyłanie plików
Postać: `ftp uzytkownik[@adres]`
W Internecie można znaleźć wiele publicznie dostępnych serwerów FTP (logowanie jako użytkownik `anonymous`). Na stronie `archie.icm.edu.pl` znajduje się wyszukiwarka ułatwiająca przeszukiwanie takich serwerów.

sftp szyfrowane połączenie z serwerem FTP pozwalającym na przesyłanie plików
Postać: `sftp uzytkownik[@adres]`
Polecenia dostępne po zalogowaniu można zobaczyć wpisując polecenie `help`
Najważniejsze polecenia to:
`get nazwa_pliku` - pobranie pliku
`put nazwa_pliku` - wysłanie pliku
`quit` - rozłączenie

scp szyfrowane kopiowanie plików z serwerów `ssh` i `sftp`
Postać: `scp [-r] uzytkownik[@adres]:plik_zrodlowy plik_docelowy`
`scp [-r] plik_zrodlowy uzytkownik[@adres]:plik_docelowy`
Składnia i działanie podobne do polecenia `cp` z tą różnicą, że kopiowanie odbywa się pomiędzy maszyna lokalna i zdalna przy użyciu szyfrowanego protokołu.
Przykłady:
`$ scp unix@158.75.5.136:paczka.tar.gz paczka.tar.gz`
pobranie pliku `paczka.tar.gz`
`$ scp paczka.tzr.gz unix@158.75.5.136:.`
wysłanie pliku `paczka.tar.gz`
`$ scp -r unix@158.75.5.136:/gry .`
pobranie całej zawartości (rekurencyjnie) z katalogu `gry` do bieżącego katalogu

wget program do pobierania zasobów stron www i serwerów ftp
Postać: `wget [-rkx] adres_strony_lub_pliku`
Program oferuje wiele możliwości (patrz `man wget`)
Jedna z ciekawych opcji jest możliwość pobierania całych witryn internetowych z zachowaniem hierarchii katalogów i plików (tworzenie tzw. mirrorów).
Przykład:
`$ wget -m http://www.phys.uni.torun.pl/~grochu/unix/materialy/index.html`

lynx przeglądarka stron WWW
Postać: `lynx [opcje] [URL]`

mutt klient poczty elektronicznej
Postać: `mutt [opcje]`

2.14 Archiwa, kompresja danych

Najczęściej używane programy służące do kompresji danych:

gzip kompresuje pliki
Postać: `gzip [-r] plik`
Skompresowane pliki automatyczne uzyskują rozszerzenie `.gz`
Opcja `-r` pozwala skompresować wszystkie pliki w podanym katalogu (każdy plik kompresowany jest osobno).
Przykład:
`$ gzip archiwum.tar`
utworzy skompresowany plik o nazwie `archiwum.tar.gz`

zip kompresuje pliki i katalogi
Postać: `zip [-r] nazwa_archiwum pliki_do_spakowania`
Tworzy plik o podanej nazwie dodając rozszerzenie `.zip` zawierający skompresowaną zawartość podanych plików.
Opcja `-r` pozwala skompresować zawartość całego katalogu.
Przykład:
`$ zip dokumenty *.txt`
utworzy plik `dokumenty.zip` zawierający skompresowaną zawartość wszystkich plików posiadających rozszerzenie `*.txt`
`$ zip -r konto ~`
w pliku `konto.zip` powinna znaleźć się zawartość całego katalogu domowego

bzip2 kompresuje pliki
Postać: `bzip2 pliki_do_spakowania`
Skompresowane pliki otrzymują rozszerzenie `.bz2`

Aby rozpakować plik utworzony za pomocą jednego z powyższych algorytmów należy wykonać:

gunzip rozpakowanie pliku `*.gz` oraz `*.tgz`
Postać: `gunzip [-r] plik`

unzip rozpakowanie pliku `*.zip`
Postać: `unzip plik`

bunzip2 rozpakowanie pliku `*.bz2`
Postać: `bunzip2 plik`

Polecenia `gzip` i `bzip2` kompresują pojedyncze pliki dlatego chcąc skompresować kilka plików w jedną całość należy utworzyć archiwum za pomocą programu `tar`.

tar narzędzie do archiwizowania danych
Postać: `tar [opcje] pliki`

Tworząc archiwum pierwszą nazwa pliku podaną jako argument musi być nazwa tego archiwum.

Najważniejsze operacje i opcje:
`c` - stwórz archiwum
`x` - rozpakuj archiwum
`l` - wyświetl zawartość archiwum
`f` - zapisz do pliku (standardowo `tar` pracuje na strumieniach wejściowym i wyjściowym)
`z` - skompresuj archiwum za pomocą programu `gzip`
`j` - skompresuj archiwum za pomocą `bzip2`
`v` - wyświetla dodatkowe komunikaty

Przykłady tworzenia archiwum:
`$ tar fcv arch.tar kat/`
utworzy archiwum o nazwie `arch.tar` zawierające zawartość katalogu `kat`
`$ tar fcvz arch.tar.gz *`
stworzy spakowane (`gzip`) archiwum o nazwie `arch.tar.gz` zawierające wszystkie pliki i katalogi z bieżącego katalogu
`$ tar fvcj arch.tar.bz2 plik1.txt plik2.txt`
utworzy archiwum skompresowane za pomocą `bzip2` o nazwie `arch.tar.bz2` zawierające dwa pliki o nazwach `plik1.txt plik2.txt`

Otwieranie archiwum:
```
$ tar fx arch.tar
$ tar fxvz arch.tar.gz
$ tar fxvj arch.tar.bz2
```

3 Powłoka bash

Powłoka jest programem, który udostępnia użytkownikowi środowisko pracy. W skład powłoki wchodzą: linia komend, zestaw wbudowanych poleceń, narzędzia obsługi zadań, narzędzia sprawdzające pisownię wpisywanych poleceń. Obecnie domyślną powłoką użytkową na serwerach studenckich naszego wydziału jest powłoka `bash`. Lista dostępnych w systemie powłok znajduje się w pliku `/etc/shells`.

3.1 Konfiguracja powłoki i środowiska

Plikiem konfiguracyjnym powłoki Bash jest plik `~/.bashrc`. W pliku tym możemy zdefiniować zmienne, aliasy i funkcje, które pozwalają dostosować środowisko oraz działanie programów do swoich potrzeb. Po zmianie zawartości pliku `.bashrc` nowe ustawienia będą dostępne w bieżącej powłoce po wykonaniu komendy:
`$ source ~/.bashrc`

Komenda `source` odczytuje zawartość podanego pliku i wykonuje zawarte w nim komendy.

3.2 Zmienne powłoki i środowiska

Zachowanie powłoki można skonfigurować ustawiając wartości specjalnym zmiennych. Oto lista kilku z nich:

`PS1` definiuje zawartość znaku zachęty

`HISTSIZE` ilość zapamiętywanych poleceń w historii

`IFS` zmienna definiująca separator pól dla komendy `read` (domyślna wartość to białe znaki: spacja, tabulacja, nowa linia)

`RANDOM` losowa wartość całkowita

Listę wszystkich zmiennych oraz funkcji zadeklarowanych w bieżącej powłoce uzyskujemy za pomocą polecenia `set`. Więcej o tworzeniu i manipulowaniu zmiennymi powłoki znajduje się w rozdziale 4.4.
Zmienne można podzielić na **zmienne powłoki** (lokalne), które są dostępne wyłącznie w danej instancji powłoki oraz **zmienne środowiskowe**, które są dziedziczone przez procesy potomne.
Zmienne środowiskowe definiuje się poprzez wyeksportowanie zmiennych powłoki za pomocą komendy `export`, np.:

```
$ ZMIENNA=42
$ export ZMIENNA
```

lub krócej

```
$ export ZMIENNA=42
```

Zachowanie wielu programów zależne jest od ustawień konkretnych zmiennych środowiskowych. Aktualną listę zmiennych środowiskowych można uzyskać za pomocą polecenia `printenv` lub `env`. Przykład typowych zmiennych środowiskowych:

`PATH` lista katalogów oddzielonych dwukropkiem. Katalogi te są przeszukiwane przez powłokę w poszukiwaniu programów do uruchomienia

`USER` nazwa użytkownika

`HOME` katalog domowy użytkownika

`EDITOR` domyślny edytor tekstu

`LD_LIBRARY_PATH` lista lokalizacji w których system poszukuje bibliotek

`LANG`, `LANGUAGE`, `LC_ALL` ustawienia lokalizacji (wersji językowej)

3.2.1 Lokalizacja

Aktualne ustawienia lokalizacji można sprawdzić za pomocą polecenia `locale`.

locale wyświetl ustawienia lokalizacji
Postać: `locle [opcje]`
Polecenie wyświetla wartości zmiennych odpowiedzialnych za lokalizację środowiska, takich jak, np,: `LANG`, `LANGUAGE`, `LC_ALL`.
Najważniejsze opcje:
`-a` wyświetli listę wszystkich zainstalowych w systemie lokalizacji

Ustawienie lokalizacji (np. języka polskiego) sprowadza się do ustawienia wartości zmiennych środowiskowych takich jak:

`LANG` podstawowa zmienna odpowiedzialna za ustawienia języka, używana gdy nie są zdefiniowane zmienne `LC_*`

`LC_CTYPE` kodowanie znaków używane do prezentacji tekstu

`LC_NUMERIC` formatowanie wartości liczbowych

`LC_TIME` formatowanie czasu i daty

`LC_COLLATE` określa sposób sortowania alfabetycznego

`LC_MESSAGES` język komunikatów systemu

`LC_ALL` nadpisuje wszystkie pozostałe ustawienia lokalizacji

Przykład ustawienia lokalizacji polskiej:
```
$ export LC_ALL=pl_PL.utf8
$ querty
bash: qwerty: nie znaleziono polecenia
```
Zmiana na inną lokalizację:
```
$ export LC_ALL=hu_HU.utf8
$ querty
bash: qwerty: parancs nem található
```
Dodanie wpisu o polskiej lokalizacji do pliku konfiguracyjnego powłoki:
```
$ echo export LC_ALL=pl_PL.utf8 >> ~/.bashrc
```

3.3 Aliasy

Alias to „przezwisko" jakie możemy nadać dowolnej komendzie. Pozwala to nadać krótką lub wygodną w zapisie nazwę poleceniom powłoki które posiadają złożoną składnię lub są na tyle często używane, że wygodniej jest je zastąpić krótszymi w zapisie aliasami. Należy pamiętać, że alias ma pierwszeństwo przed wszystkimi innymi poleceniami, więc istnieje możliwość zastąpienia dowolnego polecenia innym poprzez utworzenie odpowiedniego aliasu.

alias ustawia lub wyświetla aliasy
Postać: `alias [nazwa[=wartość]]` Przykład:
`$ alias lt=ls -l -a -t`
definiuje alias (polecenie) o nawie `lt`, które uruchomi polecenie wyświetlające listę wszystkich plików posortowanych względem czasu modyfikacji
`$ alias`
wyświetli wszystkie zdefiniowne aliasy
`$ alias lt`
wyświetli polecenie przypisane do aliasu o nazwie `lt`

unalias usuwa alias o podanej nazwie z powłoki
Postać: `unalias nazwa`
Przykład: `$ unalias lt`
usunie alias o nazwie `lt`.

4 Skrypty - wstęp do programowania w powłoce Bash

Skrypty to pliki tekstowe zawierające ciągi instrukcji i poleceń, które są uruchamiane linia po linii. W odróżnieniu od zwykłych plików tekstowych, skrypty możemy uruchamiać tak jak zwykłe programy. W systemach UNIX/GNU Linux skrypty pozwalają na zautomatyzowanie wielu czynności a każdy skrypt może zostać dodany do repertuaru dostepnych w systemie programów.

4.1 Struktura skryptu

- w pierwszej linii powinna znajdować ścieżka do powłoki w której ma być interpretowany skrypt poprzedzona znakami `#!` (tzw. *a*ha-bang lub *h*ashbang). W przypadku skryptów w powłoce Bash w pierwszea linia powinna wyglądać tak:
  `#!/bin/bash`

- w każdej kolejnej linii możemy umieścić:
  - dowolne plecenie powłoki lub instrukcję uruchamiającą program
  - instrukcję uruchamiającą inny skrypt
  - instrukcję sterującą (np. pętle `while`,`for`, warunek `if`, itp.)

- skrypt powinien kończyć się instrukcją `exit`, któej argumentem jest liczba całkowita dodatnia o wartości 0 gdy skrypt kończy się powodzeniem. Każda watość większa od 0 powinna być uzywana w przypadku gdy skrypt z róznych przyczyn nie kończy się powodzeniem

- tekst zawarty po znaku `#` aż do końca linii jest komentarzem i nie jest interpretowany przez powłokę

Przykład prostego skryptu:

```
#!/bin/bash
# To jest skrypt ktory wyswietla komunikat
echo "Witaj świecie"
exit 0
```

4.2 Uruchamianie skryptu

Ciąg instrukcji zawartych w pliku tekstowym możemy uruchomić w powłoce Bash wydając polecenie:
`$ bash skrypt.sh`
Jeżeli skrypt zawiera poprawny *h*ashbang w pierwszej linii i jezeli plikowi nadamy uprawnienia do wykonywania
`$ chmod a+x skrypt.sh`
wówczas skrypt możemy uruchomić podając jego nazwę poprzedzoną ścieżką do pliku. Przykładowo, gdy `skrypt.sh` znajduje się w bieżącym katalogu polecenie
`$ ./skrypt`
uruchomi wszystkie instrukcje zawarte w pliku. Skrypt możemy umieścić też w jednym z katalogów ze zmiennej `$PATH` i wówczas do uruchomienia skryptu wystarczy podać jego nazwę.
Aby dodać bieżący katalog do zmiennej `$PATH` należy wydać polecenie
`$ export PATH=".:$PATH"`

4.3 Wykrywanie błędów w skrypcie

W przypadku wystąpienia błędu podczas interpretowania skryptu powłoka przerywa jego działanie wyświetlając stosowny komunikat. Podczas poszukiwania przyczyn powstawania błedu warto uruchomić skrypt poleceniem:
`$ bash -x skrypt.sh`
Opcja `-x` powoduje, że każda instrukja skryptu przed uruchomieniem jest wypisywana na standardowe wyjście diagnostyczne.

4.4 Zmienne

Zmienne definiuje się używając składni `zmienna=wartosc` lub w przypadku zmiennych liczbowych `let zmienna=liczba`, np.
```
$ napis="Ala ma kota"
$ let wynik=10
```
Nazwa zmiennej może składać się z dowolnych liter, cyfr (cyfra nie może być pierwszym znakiem nazwy zmiennej) oraz znaku podkreślenia.
Wartość umieszczoną w zmiennej wydobywamy umieszczając $ przed nazwą zmiennej ewentualnie otaczając nazwę zmiennej klamrami, np.:
```
$ echo ${napis}
$ Ala ma kota
$ echo $HOME
$ /home/student
$ echo ${wynik}
$ 10
```

Tablice W powłoce Bash mamy do dyspozycji tablice jednowymiarowe. Nie muszą one być deklarowane. Do poszczególnych elementów tablicy odwołujemy sie poprzez nawiasy kwadratowe `${zmienna[indeks]}`, gdzie `index` jest liczbą całowitą dodatnią.
```
$ kolor[0]=bialy
$ kolor[1]=czarny
$ kolor[5]=zielony
$ echo Kolor pierwszy to ${kolor[1]}
```
Kolor pierwszy to `czarny`
```
$ echo Wszystkie kolory: ${kolor[*]}
Wszystkie kolory: bialy czarny zielony
```
Tablice indeksowane są liczbami całkowitymi począwszy od zera. Zmienną tablicową można zainicjować ciągiem wartości podanych w nawiasach `zmienna=(wartosc1 wartosc2 ... wartoscN)`, np.
```
$ dzien=(poniedzialek wtorek sroda czwartek piatek sobota niedziela)
$ echo ${dzien[6]}
sobota
$ echo Dni tygodnia: ${dzien[*]}
Dni tygodnia: poniedzialek wtorek sroda czwartek piatek sobota niedziela
```
Liczbę elementów tablicy uzyskujemy wyrażeniem `${#zmienna[*]}`
```
$ echo Ilosc dni tygodnia = ${#dzien[*]}
Ilosc dni tygodnia = 7
```
Wyrażenie `${#zmienna[indeks]}` zwraca ilość znaków zawartych w elemencie tablicy o podanym indeksie.

```
$ echo Slowo ${dzien[1]} zawiera ${#dzien[1]} znakow
Slowo wtorek zawiera 6 znakow
```

Polecenie `unset zmienna` usuwa podaną zmienną. Chcąc usunąc wybrany element tablicy nalezy wykonać `unset tablica[index]`.

```
$ unset kolor
$ unset dzien[4]
```

Zmienne `$*`, `$#`, `$0`, `$1`. Zmienna `$*` zawiera listę wszystkich argumentów z jakimi został wywołany skrypt, zmienna `$#` podaje liczbę tych argumentów, zmienna `$0` zawiera nazwę skryptu, zaś zmienne `$1`, `$2`, `$3`, itd. zawierają kolejne argumenty.
Przykład skryptu i nazwie `argumenty.sh`, który wyświetli swoją nazwę, liczbę argumentów oraz pierwsze dwa argumenty:

```
#!/bin/bash
echo Nazwa skryptu=$0
echo Podales $# argumentow
echo Oto one: $*
echo Argument 1 = $1
echo Argument 2 = $2
exit 0
```

Przykładowe działanie:

```
$ ./argumenty.sh
```

```
Nazwa skryptu=./argumenty.sh
Podales 0 argumentow
Oto one:
```

```
$ ./argumenty.sh Ala ma kota
```

```
Nazwa skryptu=./argumenty.sh
Podales 3 argumentow
Oto one: Ala ma kota
Argument 1 = Ala
Argument 2 = ma
```

4.5 Operacje arytmetyczne i warunki logiczne

Powłoka `bash` pozwala na wykonywanie prostych operacji arytmetycznych na liczbach całkowitych za pomocą instrukcji `let`.
Przykład:

```
$ let suma=2+2
$ echo $suma
4
$ let liczba=$suma*2
$ echo $liczba
8
```

Składnia polecenia `let` pozwala na używanie zmiennych liczbowych bez konieczności poprzedzania ich znakiem `$`.

```
$ let liczba=suma*suma+3
```

```
$ let suma++
```
Dostępne operatory oraz priorytet ich wykonywania są takie same jak w języku C. Dodatkowo, dostepny jest operator ** realizujący potęgowanie.
```
$ let x=2**10
$ echo $x
1024
```
Równoważne użyciu polecenia let jest zastosowanie ((wyrażenie)).
Przykłady:
```
$ a=$((1+2))
$ a=$(($a*$a))
$ a=$((a+1))
$ ((a++))
```

Proste operacje arytmetyczne można także wykonywać za pomocą instrukcji expr. Obliczenia o precyzji zmiennopozycyjnej (dla liczb rzeczywistych) można wykonywać za pomocą kalkulatorów bc lub dc.
Wyrażenia warunkowe realizowane są za pomocą polecenia [wyrażenie] lub polecenie test. Wartością zwracaną polecenia jest kod (status programu) 0 w przypadku gdy wyrażenie jest prawdziwe lub 1 gdy wyrażenie jest fałszywe.
Uwaga: wyrażenie [jest poleceniem, dlatego wszystkie argumenty muszą byc oddzielone spacją.
Porównywanie napisów odbywa się za pomocą argumentów ==, !=, < i >.
```
$ [ $SHELL == /bin/bash ] && echo Uzywasz powloki Bash
$ test $USER != root && echo Nie jestes administratorem
```
Porównując liczby całkowite należy skorzystać z operatorów -eq ((*ang. equal*) - równe), -ne ((*ang. not equal*) - nie równe), -lt ((*ang. less then*) - mniejsze niż), -gt ((*ang. greater than*) - większe niż), -le ((*ang. less equal*) - mniejsze równe) i -ge ((*ang. greater equal*) - większe równe).
```
$ [ $RANDOM -lt 16384 ] && echo Reszka || echo Orzel
$ test $(cat /etc/passwd | wc -l) -gt 100 && echo Uzytkownikow jest wiecej niz 100
```
Wyrażenie warunkowe może tez sprawdzać atrybuty plików:
```
 $ [ -e /etc/passwd ] && echo Plik /etc/passwd istnieje
```

-e plik	plik istnieje
-f plik	plik istnieje i jest zwykłym plikiem
-d plik	plik istnieje i jest katalogiem
-r plik	użytkownik posiada prawo do czytania pliku
-w plik	użytkownik posiada prawo do zmiany zawartości pliku
-x plik	użytkownik posiada prawo do wykonywania pliku
-o plik	użytkownik jest właścicielem pliku

```
$ test -d /etc/passwd && echo Plik /etc/passwd jest katalogiem
```
Dostępne mamy też operatory && (AND), || (OR) oraz zaprzeczenie ! (NOT) pozwalające łączyć wyrażenia warunkowe w bardziej złożone warunki.
W takim przypadku złożone wyrażenie logiczne należy umieścić w dodatkowych nawiasach [].
```
$ [[ $(($RANDOM%2)) -eq 1 && ! $USER == root ]] && echo Warunek jest spelniony
$ test ! -w /etc/passwd || $USER != root && echo Nie masz uprawnien do modyfikcaji
/etc/passwd lub nie jestes root-em.
```
Wyrażenia warunkowe znajdują zastosowanie wraz z instrukcją if oraz pętlą while oraz until.

4.6 Instrukcje sterujące

Warunek `if`. Składnia warunku:

```
if wyrażenie;
then
     instrukcje
fi
```

Gdy *wyrażenie* jest prawdziwe (zobacz wyrażenia warunkowe) wówczas wykonywane są *instrukcje* pomiędzy słowem `then` i `fi`.
Przykład:

```
#!/bin/bash
if [ $# -eq 0 ];
then
   echo "Nie podałeś żadnych argumentów "
fi
exit 0
```

Bardziej rozbudowane wyrażenie warunkowe:

```
if wyrażenie;
then
     instrukcje
else
     instrukcje 2
fi
```

Instrukcje zawarte w bloku rozpoczynającym się od `else` są wykonywane gdy *wyrażenie* nie jest spełnione. Np.:

```
#!/bin/bash
if [ $# -eq 0 ];
then
   echo "Nie podałeś żadnych argumentów. "
   echo "Podaj liczbe calkowita "
   echo "Sprobuj: $0 liczba"
   exit 1
fi
if [ $1 -lt 0 ];
then
   echo Liczba $1 jest mniejsza od zera
else
  echo Liczba $1 jest wieksza lub równa 0
fi
exit 0
```

Przykład - skrypt o nazwie `rzut.sh` wyświetlający słowo Orzeł lub Reszka z prawdopodobieństwem 1/2.

```
#!/bin/bash
if [ $(($RANDOM%2)) -eq 1 ];
then
   echo "Reszka"
else
  echo "Orzeł"
fi
```

Przykład - skrypt o nazwie `podglad.sh` który dla danego w argumencie pliku wyświetla jego zawartość za pomocą `less`, zaś jeżeli podany argument jest nazwą pliku wówczas wyświetlana jest zawartość tego katalogu.

```
#!/bin/bash
if [ $# -lt 1 ];
then
   echo "Podaj plik lub katalog jako argument."
   echo "Uzycie: $0 plik"
   exit 1
fi
if [ -f $1 ];
then
  less $1
else
  if [ -d $1 ];
  then
     ls -l $1
  else
    echo "Blad: $1 nie jest plikiem ani katalogiem"
  fi
fi
```

Pętla `for`. Pętla `for` wykonuje zadane instrukcje tyle razy ile jest elementów na podanej *liście*.

`for` *zmienna* `in` *lista*;
`do`
    instrukcje
`done`

W każdej iteracji kolejny element z *listy* jest podstawiany do *zmiennej*.
Przykład uzycia w linii komend:
`$ for f in *; do echo "Plik $f zajmuje $(du -sb $f) bajtow"; done`
dla każdego pliku w bieżącym katalogu wyświetli komunikat o ilości zajmowanych bajtów. Przykład - skrypt zamieniający w nazwach plików duże litery na małe.

```
#!/bin/bash
if [[ $# -eq 0 || $1 == "-h" || $1 == "--help" ]];
then
  echo "Uzycie: $0 [-h] plik..."
  echo "Zamienia w nazwach podanych plikaow duze litery na male (np. Plik.TXT na plik.txt)."
  echo "Opcja -h wyswietla pomoc."
```

```
  exit 1
fi
for plik in $*
do
  if [ -e $plik ];
  then
    nowy_plik=$(echo $plik | tr '[A-Z]' '[a-z]')
    if [ $plik != $nowy_plik ];
    then
      echo "Zamieniam: $plik na $nowy_plik"
      mv $plik $nowy_plik
    fi
  else
    echo "Blad: $plik - nie ma takiego pliku"
  fi
done
```

Powłoka `Bash` umożliwia także użycie pętli `for` znanej z języka C.

`for ((` *wyrażenie1* `;` *warunek* `;` *wyrażenie2* `))`
`do`
instrukcje
`done`

Wszystkie *instrukcje* są wykonywane dopóki *warunek* jest spełniony. Początkowe *wyrażenie1* jest uruchomione tylko raz przed rozpoczęciem pętli, zazwyczaj służy do zainicjowania zmiennych. Końcowe *wyrażenie2* jest wykonywane na końcu każdej iteracji, zazwyczaj używane jest do zwiększenia (lub zmniejszenia) pewnego licznika.
Przykład - skrypt wyznaczający silnię:

```
#!/bin/bash
if [[ $# -eq 0 || $1 == "-h" || $1 == "--help" ]];
then
  echo "Uzycie: $0 [-h] liczba"
  echo "Oblicza silnie podanej liczby."
  echo "Opcja -h wyswietla pomoc."
  exit 1
fi
silnia=1;
for (( i=2 ; i<=$1 ; i++ ))
do
  let silnia=silnia*i;
done
echo "Silnia wynosi $silnia"
```

Przykład - wielokrotne losowanie kostką:

```
#!/bin/bash
if [[ $1 == "-h" || $1 == "--help" ]];
then
```

```
  echo "Rzut kostka"
  echo "Uzycie: $0 [-h] liczba"
  echo "Wyswietla wynik rzutu kostka powtorzenoge zadana liczbe razy."
  echo "Opcja -h wyswietla pomoc."
  exit 1
fi
ile=1
if [ $# -gt 0 ]; then ile=$1 ;fi
for (( i=1 ; i<=ile ; i++ )) do
  wynik=$(($RANDOM%6+1))
  echo $wynik
done
```

Pętla `while`. Składnia pętli `while`:

`while` *warunek*
`do`
    instrukcje
`done`

Podane *instrukcje* są wykonywana dopóki *warunek* jest prawdziwy.
Przykład - stoper, odlicza sekundy od rozpoczęcia działania skryptu:

```
#!/bin/bash
if [[ $1 == "-h" || $1 == "--help" ]];
then
  echo "Stoper - po prostu uruchom bez argumentow."
  echo "Ctrl+C konczy odliczanie."
  echo "Opcja -h wyswietla pomoc."
  exit 1
fi
let s=0
while true; do
  echo $s
  sleep 1
  let s++
done
```

Instrukcja `true` zwraca zawsze wartość logiczną prawda. Analogicznie `false` daje odpowiedź negatywną.

Instrukcja `case`. Instrukcja `case` pozwala na wykonanie wybranych instrukcji w zależności od wartości przyjmowanej przez pewną zmienną. Działanie bardzo podobne do instrukcji `if` jednak często wygodniejsze w użyciu, zwłaszcza gdy mamy więcej niż dwie możliwości do wyboru.

`case` *zmienna* `in`
    wartość_1`)`
        instrukcje 1

```
        ;;
    wartość_2)
        instrukcje 2
        ;;
    ...
    *)
        instrukcje
        ;;
esac
```

Przykład - skrypt wyświetla menu:

```
#!/bin/sh
while true
  do
     clear
     echo "============================"
     echo "[1] Wyświetl dzisiejszą datę"
     echo "[2] Wyswietl listę plików w bierzącym katalogu"
     echo "[3] Pokarz kalendarz"
     echo "[4] Pokarz listę zalogowanych uzytkownków"
     echo "[5] Zakończ"
     echo "============================"
     echo -n "Wybierz liczbę [1-5]: "
     read akcja
     case $akcja in
       1) echo "Dzisiejsza data: `date`"
          ;;
       2) echo "Lista plików w katalogu `pwd`"
          ls -l
          ;;
       3) cal  ;;
       4) echo "Lista zalogowanych" ; who ;;
       5) echo "Do widzenia"
          exit 0 ;;
       *) echo "Blad!!! Proszę wybrać wartość 1,2,3,4, lub 5";
  esac
  echo  "Wciśnij klawisz Enter"
  read
done
```

Instrukcja `exit`**.** Instrukcja `exit` kończy działanie skryptu. Liczba całkowita umieszczona po instrukcji `exit` jest zwracana do powłoki jako wynik działania skryptu. W przypadku poprawnego wykonania skrypt powinien kończyć się wyrażeniem `exit 0`. Gdy skrypt nie został wykonany poprawnie wówczas po słowie `exit` wstawiamy dowolną liczbę różną od zera (wartość zwracanej liczby może w ten sposób sygnalizować rodzaj błędu który spowodował niepoprawne wykonanie skryptu). Wartość zwracana po słowie `exit` umieszczana jest w zmiennej `$?`.

Instrukcja `read`. Instrukcja `read` pozwala wczytać linie tekstu do podanej zmiennej.
Przykład:

```
#!/bin/bash
echo Podaj imie i nazwisko
read dane
echo Witaj $dane
```

Możliwe jet też czytanie tekstu z pliku linia po linii, np.:

```
#!/bin/bash
echo Podaj nazwe pliku do wyswietlenia
read plik
if [ ! -r $plik];
then
  echo "Nie moge czytac z pliku $plik"
  exit 1
fi
cat $plik | while read linia
do
  echo $linia
done
```

Funkcje. W powłoce `bash` możliwe jest definiowanie funkcji, czyli wyodrębnionych podprogramów oznaczonych pewną unikatową nazwą.

```
nazwa_funkcji( )
{
    instrukcje do wykonania
    return
}
```

Funkcja wykonywana jest w momencie gdy pojawi się wywołanie jej nazwy.
Przykład:

```
#!/bin/sh
pomoc()
{
  echo "Wyswietla liste i liczbe zalogowanych uzytkonikow"
  echo "Opcje:"
  echo "-h pomoc"
  echo "-l liczba zalogowanych użytkoników"
  echo "-w lista zalogowanych użytkoników"
  return
}
# lista niepowtarzajacych sie nazw zalogowanych uzytkownikow
lista()
{
  users=( ‘who | cut -f 1 -d ’ ’ | sort | uniq ‘ )
```

```
  return
}
case $1 in
  "-h")  pomoc ;;
  "-l") lista
        echo "Liczba zalogowanych uzytkonikow = ${#users[*]} "
        ;;
  "-w") lista
        echo "Lista uzytkownikow: ${users[*]}" ;;
     *) pomoc ;;
esac
```

Do funkcji możemy przekazać argumenty dodając je przy wywołaniu po nazwie funkcji. Kolejne argumenty umieszczone są w zmiennych `$1`, `$2`, `$3`, itd. Ilość argumentów dana jest poprzez `$#`, lista wszystkich argumentów zawarta jest w zmiennej `$*` a zmienna `$0` zawiera nazwę funkcji.

```
#!/bin/sh
funkcja()
{
  echo "Argumenty funkcji $*"
  echo "Ilość argumentów funkcji $#"
  return 0
}
echo "Argumenty skryptu $*"
echo "Ilość argumentów skryptu $#"
echo Uruchamiam funkcje z argumentami: raz dwa trzy
funkcja raz dwa trzy
echo Uruchamiam funkcje z argumentami będącymi nazwami plików z bierzącego katalogu
funkcja *
exit 0
```

Zmienne użyte wewnątrz funkcji maja zakres globalny, tzn. ich wartość jest dostępna poza funkcją (np. zmienna `users` z pierwszego przykładu w tym paragrafie). Chcąc ograniczyć czas życia zmiennej wyłącznie do obszaru zdefiniowanego przez funkcję należy zadeklarować zmienną poprzedzając ja instrukcją `local`. Używanie zmiennych lokalnych może uchronić przed wieloma trudnymi do wykrycia błędami, więc gdzie tylko jest to możliwe, wewnątrz funkcji należy je stosować.
Przykład:

```
#!/bin/sh
funkcja()
{
  local zmienna="Zmienna lokalna"
  echo "Jestem wewnątrz funkcji"
  echo "zmienna=$zmienna"
  return 0
}
zmienna="Zmienna globalna"
echo "Zmienna=$zmienna"
funkcja
echo "Zmienna=$zmienna"
exit 0
```

4.7 Przykłady

Tutaj znajdują się przykładowe skrypty w `Bash`.

5 Indeks polecen

`alias` - ustawia lub wyświetla aliasy
`atm` - usuwa z kolejki zadanie o podanym numerze
`at` - uruchamia proces o zadanym czasie
`bc` - kalkulator o dowolnej precyzji
`bg` - uruchamia zawieszone zadanie w tle
`bzip2` - kompresuje pliki
`cal` - wyświetla kalendarz
`cat` - wyświetla zawartość strumienia wejściowego lub zawartość plików
`cd` - zmienia bieżący katalog
`chgrp` - zmienia grupę użytkowników pliku
`chmod` - zmienia prawa dostępu do pliku
`chown` - zmienia właściciela i grupę pliku
`cmp` - porównuje pliki znak po znaku
`cp` - kopiuje pliki i katalogi
`cut` - Wypisuje wybrane fragmenty linii
`date` - podaje datę i czas systemowy
`df` - infromacje o zajęctości zamontowanych dysków
`diff` - znajduje różnice pomiędzy plikami taktowymi
`du` - wyświetla rozmiar zajętej przestrzeni dyskowej
`echo` - wyświetla linię tekstu
`expr` - oblicza wartość wyrażenia matematycznego
`fg` - uruchamia zatrzymane zadanie na pierwszym planie
`file` - wyświetla informację o zawartości pliku
`find` - szuka plików w drzewie katalogów
`finger` - informacje o użytkowniku.
`free` - informacje o zajętości pamięci w systmie
`ftp` - połączenie z serwerem FTP pozwalającym na przesyłanie plików
`fuser` - identyfikuje procesy używające plików lub gniazd sieciowych
`grep` - wyświetla linie pasujące do wzorca
`groups` - nazwy bieżących grup
`gzip` - kompresuje pliki
`head` - wyświetla początek pliku
`help` - pomoc dotycząca poleceń wbudowanych w powłokę
`hostname` - nazwa hosta
`host` - podaje informacje o hoście
`htop` - lista procesów w czasie rzeczywistym
`id` - informacje o użytkowniku - GID, UID itp.
`info` - podręcznik GNU
`jobs` - podaje status procesów uruchomionych w bieżącej powłoce
`killall` - zabija procesy o podanej nazwie
`kill` - zabija proces
`less` - wyświetl zawartość pliku strona po stronie
`ln` - tworzy dowiązanie (sztywne lub symboliczne) do plików

`locale` - wyświetl ustawienia lokalizacji
`locate` - wyszukiwanie plików o podanej nazwie
`ls` - wyświetla zawartość katalogu
`lynx` - przeglądarka stron WWW
`mail` - wysyłanie poczty elektronicznej
`man` - wyświetla strony podręcznika (manuala) dotyczące danego polecenia
`mkdir` - tworzy katalog
`more` - wyświetla zawartość pliku strona po stronie
`mutt` - klient poczty elektronicznej
`mv` - przenosi pliki
`nice` - uruchamia program z zadanym priorytetem
`nohup` - uruchamia polecenie, które nie zostanie przerwane w momencie wylogowania
`paste` - łączy linie plików
`pgrep` - wyświetla numery PID procesów pasujących do wzorca
`ping` - wysyła pakiet testowy do wybranego hosta
`pkill` - wysyła sygnał do procesów o nazwach pasujących do wzorca
`printenv` - wyświetla zmienne środowiskowe
`printf` - wypisuje sformatowany tekst
`ps` - podaje informacje o działających procesach
`pstree` - wyświetla drzewo procesów
`pwd` - wyświetla bieżący katalog
`renice` - pozwala zwiększyć priorytet działającego procesu
`rmdir` - usuwa puste katalogi
`rm` - usuwa pliki
`sed` - Edytor strumieniowy
`seq` - wyświetla sekwencję liczb
`sftp` - szyfrowane połączenie z serwerem FTP pozwalającym na przesyłanie plików
`sort` - wypisuje posortowaną zawartość pliku tekstowego
`ssh` - szyfrowane połączenie ze zdalnym komputerem
`tail` - wyświetla koniec pliku
`talk` - program do interaktywnej rozmowy z użytkownikiem
`tar` - narzędzie do archiwizowania danych
`tee` - czyta standardowe wejście i przesyła je na standardowe wyjście oraz do pliku.
`telnet` - połączenie ze zdalnym komputerem
`top` - lista procesów w czasie rzeczywistym
`touch` - zmienia datę modyfikacji pliku lub tworzy pusty plik
`traceroute` - wyświetla trasę pokonywana do danego hosta
`tr` - Zamienia znaki wczytane ze standardowego wejścia.
`tty` - wyświetla nazwę terminala
`unalias` - usuwa alias o podanej nazwie z powłoki
`uname` - informacje o systemie
`wc` - liczy ilość znaków, słów i linii w pliku
`wget` - program do pobierania zasobów stron www i serwerów ftp
`whatis` - przeszukuje podręcznik (opisy poleceń) w poszukiwaniu danej nazwy.
`whereis` - wyszukuje (wszystkie) położenia plików binarnych, źródłowych i stron podręcznika danego polecenia
`whoami` - kim jestem
`who` - lista zalogowanych użytkowników
`write` - wysyła wiadomość tekstową do użytkownika
`xargs` - buduje i uruchamia polecenia powłoki na podstawie tekstu ze standardowego wejścia

`yes` - wyświetla w nieskończoność dany ciąg znaków
`zip` - kompresuje pliki i katalogi