Wstęp do użytkowania systemu operacyjnego LINUX

Magdalena Kuich

Kontakt: mkuich@fuw.edu.pl

Materiały: www.fuw.edu.pl/~mkuich/tik2022/

Organizacja zajęć - informacje ogólne

- Zajęcia odbywają się w trybie stacjonarnym
- Obecność na zajęciach nie jest obowiązkowa, obecność na kolokwiach jest!
- Rozkład zajęć:

temat	liczba zajęć	zaliczenie
LINUX	1 (2) zajęcia	–
LATEX	2 (+1) zajęcia	kolokwium
PYTHON	3 (+1) zajęcia	kolokwium
Mathematica	4 (+1) zajęcia	kolokwium
kolokwium poprawkowe	1 zajęcia	1 wybrane kolokwium

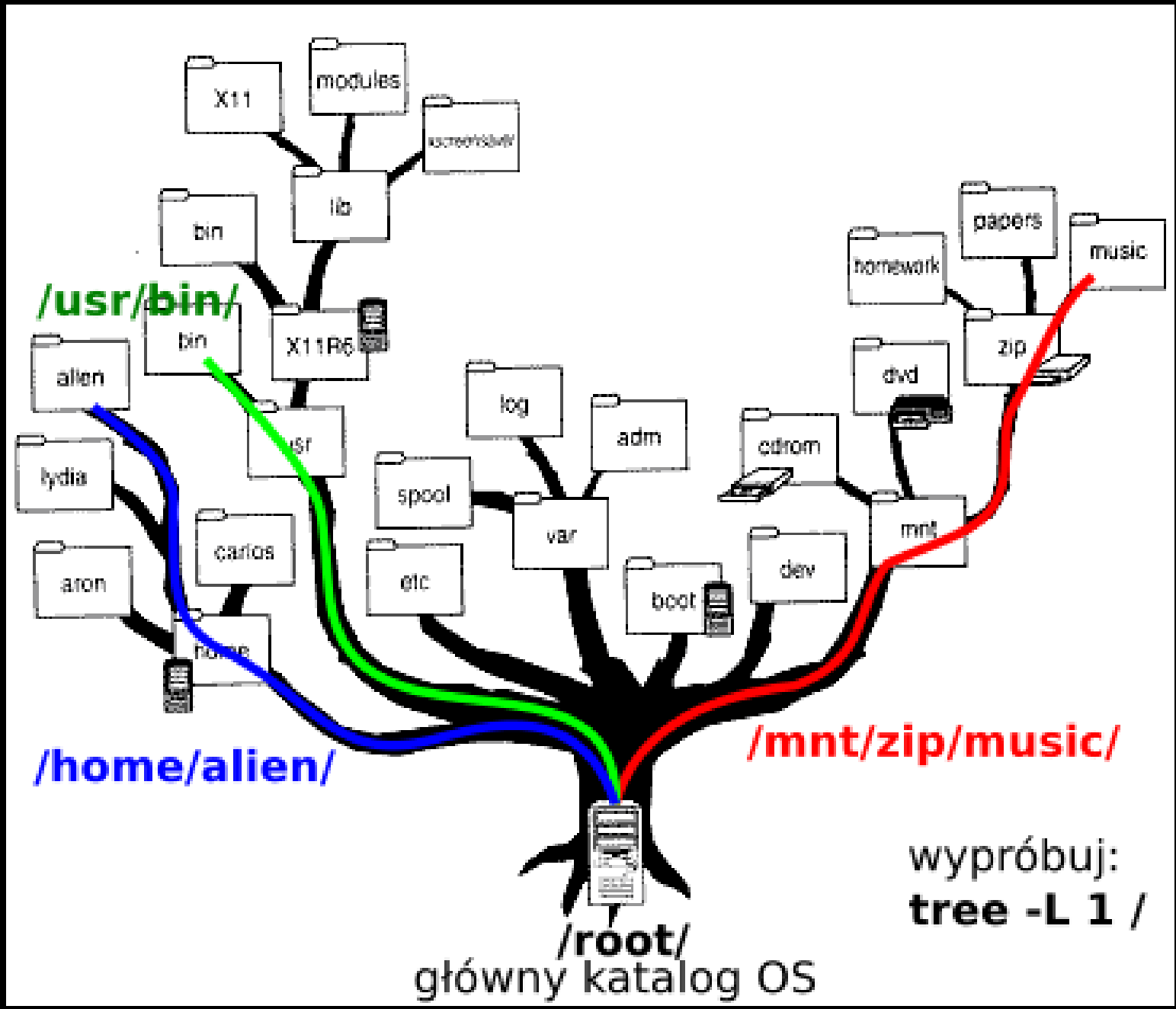
	Październik					Listopad					Grudzień				Styczeń				
PN		3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30
WT		4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17	24	31
ŚR		5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	18	25	
CZW		6	13	20	27	3	10	17	24	1	8	15	22 PT	29	5	12	19	26	
PT		7	14	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	
SB	1	8	15	22	29	5	12	19	26	3	10	17	24	31	7	14	21	28	
N	2	9	16	23	30	6	13	20	27	4	11	18	25	1	8	15	22	29	

Zaliczenie przedmiotu: 50% punktów ze wszystkich kolokwiów i testu z wykładu

LINUX OS 1

- Oprogramowanie jest otwarte i wolne - każdy może z niego korzystać za darmo i rozwijać na własną rękę
- Interfejsy:
  - → graficzny - zarządzamy komputerem poprzez odpowiednie menadżery, różne środowiska graficzne (Xfce, KDE, GNOME, Unity, Mate ...)
  - → wiersza poleceń - zarządzamy komputerem za pomocą komend wysyłanych w terminalu/konsoli
  - → w trybie graficznym mamy możemy korzystać z emulatora trybu tekstowego - terminala
- Najbardziej znane dystrybucje (rodzaje):
  - → Mint LINUX
  - → Ubuntu
  - → **Fedora**
  - → CentOS
  - → Debian
  - → Arch Linux
- LTS. - Long Term Support, warto wybierać właśnie te dystrybucje

Hierarchiczny system plików



Katalogi i serwery

- Każdy użytkownik w systemie Linux ma przypisany swój **katalog domowy** np.: `/dmj/2022/ab123456/`
  - → dla wygody wprowadzono skrót: `~/` = `/dmj/2022/ab123456/`
  - → to podstawowe miejsce pracy na systemie Lixux
  - → miejsce przeznaczone na nasze dane, indywidualne pliki konfiguracyjne itp.
  - → pojemność katalogu domowego na studenckich serwerach wydziałowych jest ograniczona do ∼5 GB
- Pracując na serwerach `okwf`, tj. `tempac` lub `primus`, mamy do dyspozycji także **katalog roboczy**:
  - → jest on podpięty (podlinkowany) w katalogu domowym: `~/_work_/`
  - → pojemność katalogu roboczego na studenckich serwerach wydziałowych wynosi ∼2 GB
- Studenckie serwery wydziałowe:
  - → `tempac.okwf.fuw.edu.pl` - jest dostępny do logowania (SSH) ze świata, umożliwia dostęp do zawartości własnych kont spoza Wydziału
  - → `primus.okwf.fuw.edu.pl` - dostępny do logowania z sieci Wydziału, można na nim uruchamiać dłuższe zadania obliczeniowe

Komunikacja zdalna - `ssh`

- `ssh` - *secure shell*, to protokół komunikacyjny stosowany w sieciach TCP/IP. Służy do zdalnego i szyfrowanego łączenia terminalowego z komputerami.
- Poprzez `ssh` głównie pracujemy w trybie tekstowym (tj. w terminalu), ale możemy przesyłać także tryb graficzny, jeśli wywołamy połączenie z opcją `-X` lub `-Y`.
- Łącząc się przez `ssh`, musimy podać login użytkownika, serwer (host) oraz opcje (np. transmisja grafiki lub nr portu)
- Domyślnie łączmy się przez port 22, ale można wymusić łączenie przez inne porty.
- Chcąc połączyć się z pracownią komputerową na wydziale z komputera spoza sieci wydziałowej:
  `ssh -X ab123456 (at) tempac.okwf.fuw.edu.pl`
  - → `ab123456` - mój login
  - → `tempac.okwf.fuw.edu.pl` - nazwa host'a
- Chcąc połączyć się z serwerem `primus`:
  `ssh -X ab123456 (at) primus.okwf.fuw.edu.pl`
- Linux: terminal
  Windows: Putty, Google Chrome Secure Shell

Terminal i program

- W terminalu możemy uruchamiać programy, wykonywać polecenia na plikach i katalogach, tworzyć i usuwać pliki i katalogi, edytować pliki tekstowe, etc.
  - → każde poprawne polecenie wpisane w terminalu i zatwierdzone poprzez wciśnięcie przycisku `Enter` uruchamia jakiś program/programik/skrypt, np.:
    `gedit` - edytor tekstu
  - → uruchomiony program "zamraża" terminal, czyli uniemożliwia wpisywanie kolejnych poleceń
  - → aby "odblokować" terminal można zastosować:
    `Ctrl+C` - zamyka aktualnie uruchomiony program
    `Ctrl+Z` - zawiesza aktualnie uruchomiony program i przywraca terminal. Można wtedy wpisać w terminalu komendę
    `bg` - *background*, która wprowadza zawieszony program do pracy w "tle", a aby przywrócić program z pracy w tle można wtedy wpisać w terminalu komendę
    `fg` - *foreground*.
  - → Można też uruchomić program, wpisując znak & (ampersand) po nazwie programu, co sprawia, że program działa w "tle" i okno terminala jest nadal aktywne:
    `gedit&`

Dokumentacja podstawowych poleceń i procesy

- `man` pozwala na przeglądanie dokumentacji wszystkich programów
  - → `man nazwa_polecenia`, np. `man gedit`
  - → z man'a wychodzimy przyciskiem `q`
  - → w man'ie szukamy przyciskiem `/`
  - → następny wynik wyszukiwania jest dostępny po kliknięciu `n`
- `ps -e` lub `ps -ef` wyświetlają listę wszystkich procesów działających na danym serwerze (domyślnie - chronologicznie)
  - → każdy proces ma przypisane numer procesu (PID)
  - → każdy proces jest wywołany przez użytkownika
  - → każdy proces można wyłączyć znając jego PID i korzystając z polecenia `kill` np. `kill 618087`
- `top` wyświetlają listę procesów działających na danym serwerze wraz ze statystykami systemu (domyślnie - pod względem obciążenia)
  - → z man'a wychodzimy przyciskiem `q`
- `echo` pozwala na wyświetlanie napisów przekazanych jako argument
  `echo "Hello"`
- `$` - służy do odwoływania się do zmiennej w powłoce `echo $HOME`
  mamy kilka zmiennych systemowych, np.: HOME, USER, PATH
  możemy także sami definiować zmienne i się do nich odwoływać:

```
x=10
echo $x
```

```
napis="Hello"
echo $napis
```

Katalogi i pliki - podstawowe polecenia 1

- `tree` - wyświetla strukturę drzewa katalogów
- `pwd` - *print working directory*, wyświetla aktualny katalog roboczy
- `cd` - *change directory* pozwala zmienić katalog roboczy
  - → `cd ścieżka_do_katalogu` - wchodzimy do katalogu
  - → `cd ..` - wychodzimy o poziom wyżej
  - → `cd` - wychodzimy do katalogu domowego
- `ls ścieżka_do_katalogu` - *list content*, wyświetla zawartości katalogu
  - → `ls -a` - pozwala na wyświetlenie plików „ukrytych”
  - → `ls -l` - wyświetla szczegółowe dane plików
  - → `ls -R` - listuje katalogi „rekurencyjnie”
- `mkdir nazwa_katalogu` - tworzy katalogu o nazwie nazwa_katalogu
- `touch nazwa_pliku` - tworzy pusty plik o nazwie nazwa_pliku

Zadania 4–11

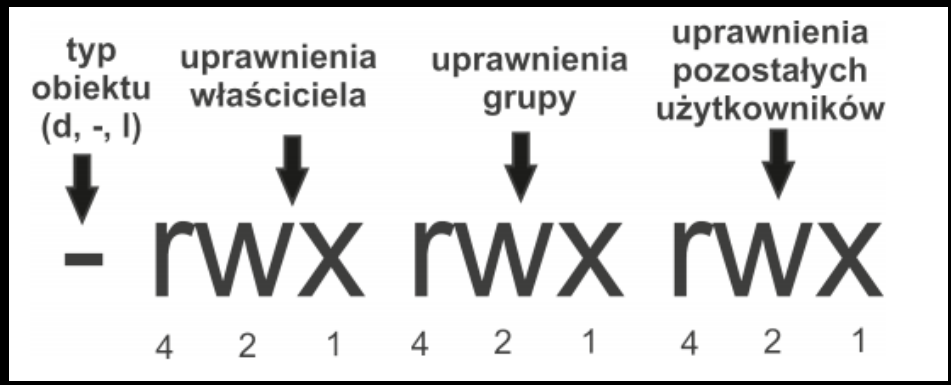
Katalogi i pliki - podstawowe polecenia2

- `rmdir nazwa_katalogu` usuwa pusty katalog o nazwie nazwa_katalogu
- `rm nazwa_pliku` - usuwa plik o nazwie nazwa_pliku
  - → `rm -r nazwa_katalogu` - usuwa z zawatrością
  - → `rm -f` - "forsuje" usuwanie plików/argumentów
  - → `rm -i` - wymusza zapytanie o usunięcie plików/argumentów
- `cp ścieżka1 ścieżka2` - *copy*, pozwala skopiować argument1 w miejsce wskazane przez argument2
  - → `cp -r katalog1 katalog2` - kopiowanie katalogów z zawartością
  - → `cp katalog/plik ./` - kopiowanie pliku, do katalogu w którym się znajdujemy
    `.` lub `./` - oznaczają "tu gdzie jesteś"
- `mv ścieżka1 ścieżka2` - *move*, pozwala przenieść argument1 w miejsce wskazane przez argument2

Zadania 12–15

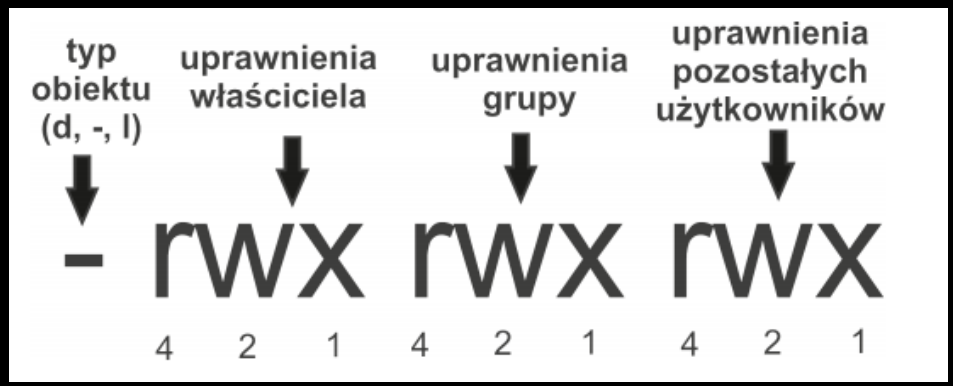
Prawa dostępu

- Każdy plik w systemie linux ma określone prawa dostępu.
- Istnieją trzy podstawowe prawa (poniżej w zapisie symbolicznym):
  - → `r` - *read*, uprawnia do przeczytania pliku
  - → `w` - *write*, uprawnia do zapisu i modyfikowania pliku
  - → `x` - *execute*, uprawnia do wykonania/uruchomienia argumentu (najczęściej skryptu lub programu)
- Każdy z tych atrybutów można ustawić dla właściciela pliku (u-user), innych z grupy (g-group) lub wszystkich innych użytkowników (oothers). Każdy użytkownik może należeć do wielu grup! Aby poznać swoje grupy można skorzystać z polecenia `id`.
- Dla katalogów artybut `x` pozwala na wejście do katalogu lub dowolnego podkatalogu, a `r` na listowanie zawartości.



Prawa dostępu

- Typ obiektu:
  - → `d` - *directory*, katalog
  - → `-` - plik
  - → `l` - link
- Prawa dostępu można opisać z pomocą liczb całkowitych z zakresu 0-7. W takim zapisie mamy odpowiednie przyporządkowania:
  - → `x` = 1 - *execute*, uprawnia do wykonania
  - → `w` = 2 - *write*, uprawnia do zapisu i modyfikowania pliku
  - → `r` = 4 - *read*, uprawnia do przeczytania pliku
- Konkretne prawa dostępu uzyskuje się dodając do siebie 1, 2 i 4. Np:
  - → 1+2 = 3 : eXecute + Write
  - → 1+4 = 5 : eXecute + Read
  - → 1+2+4 = 7 : eXecute + Write + Read



Nadawanie uprawnień dostępu - polecenie `chmod`

- Polecenie `chmod` pozwala na ustawienie praw dostępu dla pliku lub katalogu
- `chmod` używamy w postaci: `chmod <przywileje> nazwa_pliku`, np.:
  - → `chmod go=rx plik.txt` - ustawia uprawnienia do odczytu i wykonywania dla grupy oraz pozostałych użytkowników, odbiera wcześniej istniejące uprawnienia
  - → `chmod u+x,o+r plik.txt` - dodaje prawa już istniejących: do wykonywania dla właściciela i prawa odczytu dla innych
  - → `chmod a+rwx plik.txt` - nadaje wszystkim wszystkie możliwe uprawnienia'plik.txt',
- W zapisie numerycznym przywileje określają 3 cyfry - po kolei dla właściciela, grupy i wszystkich innych, np.:
  - → `chmod 744 nazwa_pliku` - ustawia pełen prawa dla właściciela i prawa odczytu dla innych
  - → `chmod -R 777 /home/user` - wszyscy będą mogli zmieniać zawartość katalogu `/home/user` oraz jego podkatalogów, jak też czytać go i wykonywać zawarte w nim pliki
- Opcja `-R` pozwala (jak zwykle) działać rekurencyjnie na podkatalogach

Domyślne ustawienia dostępu - polecenie `umask`

- `umask` (*user mask*) bez argumentów wyświetla odwrotność liczb określających uprawnienia; np.: maska `0002`:
  - → właściciel ma pełne prawa do zasobu (0002: od 7 odjęto 0, co daje 7)
  - → grupa (0002: od 7 odjęto 0, co daje 7)
  - → pozostali mają odczyt i wykonanie (0002: od 7 odjęto 2 co daje 5)
  - → pierwsze "0" od lewej strony to bity specjalne
- `umask -S` wyświetla ustawienia w formie symbolicznej
- `umask <maska>` ustawia domyśle prawa dostępu do nowych plików i katalogów; np.: `umask 0022`:
  - → właściciel ma pełne prawa do zasobu (0002: od 7 odjęto 0, co daje 7)
  - → grupa (0002: od 7 odjęto 2, co daje 5)
  - → pozostali mają odczyt i wykonanie (0002: od 7 odjęto 2 co daje 5)
- `umask` może przyjmować także ustawienia w formie symbolicznej; np.:
  - → `umask u-w` odbiera właścicielowi domyślne uprawnienia zapisu
  - → `umask u+w` dodaje właścicielowi domyślne uprawnienia zapisu

Zadania 16–20

Wzorce

- Znak $*$ zastępuje dowolną liczbę dowolnych znaków
- Znak ? zastępuje dokładnie jeden dowolny znak.
- Używając [] można określić zakres znaków, które mogą się pojawić. Np:
  - → `[abc]` - zastępuje a lub b lub c.
  - → `[a-c]` - zastępuje od a do c
  - → `[0-9]` - zastępuje dowolną cyfrę
  - → `[!a-c]` - zastępuje dowolny znak poza wymienionymi
  - → `{fraza1,fraza2}` - jeden z ciągów znaków oddzielonych przecinkami.

Zadania 21–22

Wyświetlanie plików i zliczanie słów

- Polecenia służące do wyświetlania całej zawartości plików:
  - → `less`, np.: `less plik.txt`
  - → `more`, np.: `more plik.txt`
  - → `cat`, np.: `cat plik.txt`
- Polecenia służące do wyświetlania początków lub końców plików:
  - → `head`, np.: `head plik.txt` - drukuje początek pliku (domyślnie pierwszych 10 wierszy)
  - → `tail`, np.: `tail plik.txt` - drukuje koniec pliku (domyślnie ostatnich 10 wierszy)
- `wc` - *word count*, drukuje liczbę linii, słów i znaków w tekście
  - → można wywołać go bez opcji lub z opcjami `-l`, `-w` lub `c`
  - → np.: `wc plik.txt`

Zadania 23–25

Standardowe wejście/wyjście i operatory przekazania

- Znaki, które wpisujemy z klawiatury trafiają w systemie do tzw. „**standardowego wejścia**". Program odpowiada na tzw. „**standardowe wyjście**", które wyświetlane jest na ekranie.
- Operator $<$ podaje na std wejście zawartość pliku.
- Operator |, tzw. *pipe*, pozwala przekierować std wyjście na std wejście
- Dane ze standardowego wyjścia można zapisać do pliku używając znaków $>$ lub $>>$
  - $\rightarrow$ $>$ tworzy nowy plik i zapisuje do niego wynik działania programu. Jeżeli plik już istnieje zostaje zastąpiony!
  - $\rightarrow$ $>>$ działa podobnie, ale jeżeli plik już istnieje, to wynik zostaje dopisany.

Zadania 26–28

Wyszukiwanie i sortowanie

- `find <ścieżka> <warunki>`, gdzie:
  - → `<ścieżka>` - w tym katalogu i jego podkatalogach zostanie dokonane przeszukanie
  - → `<warunki>` - zestaw warunków precyzujących jakie pliki mają być wyszukane.
    - → `-name wzorzec` - pozwala sprecyzować nazwę (działają znaki specjalne `*`,`?` i `[]`). Wzorzec trzeba podać w „”
    - → `-size n[ck]` - rozmiar: `c` - w bajtach, `k` - w kilobajtach. (`+n` -rozmiar większy niż, `-n` - mniejszy niż)
    - → `-type` - wyszukiwanie “po typie” plików.
- `grep <wzorzec> <plik/wejście>` - przeszukuje dane na wejściu w poszukiwaniu linii ze wzorcem
  - → wzorzec trzeba podać w „”, (działają znaki specjalne `*`,`?` i `[]`)
- `du` - oblicza i wypisuje informacje o ilości miejsca na dysku zajmowanego przez pliki w poszczególnych katalogach
- `sort` - sortuje dane wg wybranych kryteriów

Zadania 29–31

Historia poleceń

- `.bash_history` jest plikiem przechowującym listę poleceń wykonanych w powłoce przez użytkownika
  - → zazwyczaj plik z historią jest *ukryty* w katalogu domowym użytkownika (“.” oznacza, że plik jest ukryty)
  - → aby go “wylistować” należy skorzystać z opcji umożliwiającej wyświetlanie plików ukrytych
  - → można go także znaleźć poprzez zmienną: `$HISTFILE`

- Polecenia `history` wyświetla w terminalu ostatnie (domyślnie 5000) komend wykonanych w powłoce przez użytkownika
  - → `history 13` wyświetli ostatnie 13

Zadania 32–34

Skrypty – bonus

- W bash'u możemy pisać proste programy - skrypty
  - → skrypt = plik tekstowy, w którym znajdują się polecenia interpretowane przez powłokę bądź interpreter
  - → w bash'u - rozszerzenie "sh"; np.: `nazwa_skryptu.sh`
  - → w bash'u nagłówek: `#!/bin/bash`
  - → kolejne polecenia w nowych liniach lub w tej samej linii oddzielone ";"

```
#!/bin/bash
echo "Hello "
echo "world"
```

```
#!/bin/bash
echo "Hello" ; echo "world"
```

  - → uruchamianie poprzez ścieżkę do skryptu; np.:
    `./skrypt.sh` - skrypt znajdujący się w katalogu, w którym stoję
    `<katalog_w_ktorym_jest_skrypt>/skrypt.sh` - skrypt w innym katalogu
  - → skrypty mogą przyjmować argumenty, piszemy je po spacji i nazwie skryptu

Mój skrypt "skrypt.sh":

```
#!/bin/bash
echo "Hello world"
echo "My name is $1"
```

Uruchomienie:

```
./skrypt.sh M
```

Efekt:

```
Hello world
My name is M
```

Pętle – bonus

- Uogólniony przykład pętli for:

```
for <iterator i warunek iterowania się pętli> ;
do polecenie 1 ;
polecenie 2 ;
...  ;
done
```

Kilka przydatnych przykładów:

```
for i in {1..5} ; do touch plik$i.txt ; done
```

```
for (( j=1; j<=5; j++ )); do echo "Liczba $j">>plik$c.txt ;
done
```

```
for file in $(ls plik*) ; do cat $file ; done
```