Systemy operacyjne

Skrypt do ćwiczeń laboratoryjnych

Cezary Sobaniec

v1.6
2024-04-09

Politechnika Poznańska
Instytut Informatyki

Spis treści

1 Podstawy użytkowania systemu Unix

1.1 Środowisko pracy

1. Zaloguj się do systemu podając nazwę użytkownika i hasło. Uruchom emulator terminala (pozycja *Terminal Program* z menu). Wykonaj komendę `ls` (ang. *list*).
2. Zmień hasło komendą `passwd`, podając stare hasło i dwa razy nowe.
3. Przejdź do konsoli tekstowej stosując kombinację **Ctrl–Alt–F1** [1]. Zaloguj się do systemu. Wykonaj komendę `ls`. Przejdź do innych konsoli wciskając **Alt–F1**, **Alt–F2**, **Alt–F3**, **Alt–F4**. Wyloguj się z systemu wykonując komendę `exit` lub `logout`. Wróć do trybu tekstowego wciskając **Alt–F7**.
4. Porównaj efekty wykonania następujących komend:

```
# ls
# ls -l
# ls -a
# ls -l -a
# ls -a -l
# ls -la
```

[1] Uruchamiając system Linux pod kontrolą VirtualBox należy użyć kombinacji **Host-F1**, gdzie klawiszem **Host** domyślnie jest **Right-Ctrl**.

```
# ls -al
```

5. Sprawdź nazwę katalogu bieżącego komendą `pwd` (ang. *print working directory*). Zmień bieżący katalog komendą `cd` (ang. *change directory*):

```
# cd /
# cd /usr
# cd local
# cd bin
# cd ..
# cd man
# cd .
# cd
```

6. Wykonaj komendę `ls` stosując zarówno przełączniki jak i argumenty:

```
# ls -l /usr
# ls -l -a /tmp
# ls -l ..
```

1.2 Pomoc systemowa

1. Przywołaj pomoc systemową opisującą komendę `ls`:

```
# man ls
```

Zapoznaj się z podstawowymi komendami sterującymi przeglądarką pomocy systemowej:

`q`	wyjście,
`Enter`, `j`, `k`	przewijanie liniami,
`Spacja`, `Ctrl-f`, `Ctrl-b`, `Ctrl-d`, `Ctrl-u`	przewijanie stronami,
`g`, `G`	przejście na początek/koniec pliku,
`/`	rozpoczęcie wyszukiwania tekstu,
`n`, `N`	przejście do następnego/poprzedniego wystąpienia słowa.

2. Uzyskaj dostęp do stron pomocy w języku polskim ustawiając zmienną środowiskową LANG:

```
# export LANG=pl_PL.UTF-8
# man ls
```

gdzie pl oznacza język polski, PL oznacza Polskę a UTF-8 jest standardem kodowania znaków narodowych. W przypadku nieczytelnych polskich znaków ustaw wartość zmiennej LANG bez wskazywania kodowania:

```
# export LANG=pl_PL
```

i wybierając z menu okna terminala komendę *Settings/Encoding*, zmień kodowanie na ISO 8859-2.

3. Przećwicz wypisywanie komunikatów w językach narodowych próbując usuwać nieistniejący plik:

```
# rm abcd
rm: nie można usunąć `abcd': Nie ma takiego pliku ani katalogu
# export LANG=de_DE.UTF-8
# rm abcd
# export LANG=fr_FR.UTF-8
# export LANG=ru_RU.UTF-8
# export LANG=zh_TW.UTF-8
```

4. Odwołaj się do różnych sekcji pomocy systemowej (1 – polecenia, 2 – funkcje systemowe, 3 – funkcje biblioteczne, 4 – pliki specjalne, 5 – formaty plików, 6 – gry, 7 – różne, 8 – polecenia administracyjne), np.:

```
# man sleep
# man 3 sleep
# whatis sleep
```

Zapis sleep(3) oznacza stronę pomocy systemowej dla hasła sleep w sekcji 3.

5. Znajdź informacje o komendzie służącej do tworzenia nowych katalogów. Wykorzystaj komendę `apropos`:

```
# apropos directory
# apropos "make.*director"
```

Zapis „`.*`” jest fragmentem *wyrażenia regularnego*, oznaczającego dowolny ciąg znaków (również pusty).

6. Przetestuj działanie komendy `whereis`:

```
# whereis ls
```

7. Przetestuj alternatywny do `man` system pomocy `info` na przykładzie opisu programu `gawk`:

```
# info gawk
```

Zastosuj następujące komendy:

q	wyjście,
Tab	przejście do następnego odnośnika na stronie,
Enter	aktywowanie odnośnika i przejście do nowej strony,
l	(ang. *last*) powrót do poprzedniej strony,
n, **p**	(ang. *next*, *previous*) przejście do następnej/poprzedniej strony w sekwencji,

`u` (ang. *up*) przejście do strony nadrzędnej w hierarchii,
`t` (ang. *top*) przejście do strony głównej,
`/` rozpoczęcie wyszukiwania tekstu,
`i` aktywowanie skorowidza dokumentu.

Uruchom alternatywną przeglądarkę dokumentów `info`:

```
# pinfo gawk
```

8. Na podstawie dokumentacji ze strony hier(7) pomocy systemowej zapoznaj się z opisem znaczenia następujących katalogów: `/etc`, `/bin`, `/usr`, `/tmp`, `/var`, `/home`, `/dev` (struktura katalogów w systemach Unix jest ustandaryzowana).
9. Przygotuj przykładową stronę pomocy systemowej do wydruku i obejrzyj ją na ekranie:

```
# man -t ls > out.ps
# evince out.ps
```

1.3 Interpreter poleceń

1. Przećwicz mechanizm przywoływania i edycji ostatnio wykonywanych komend stosując klawisze kursorów ↑ i ↓.
2. Wyświetl historię ostatnio wykonywanych poleceń komendą `history`. Wykonaj *n*-te polecenie z historii, np.:

```
# !120
```

3. Przećwicz interaktywne przeszukiwanie ostatnio wykonywanych poleceń dostępne po wciśnięciu kombinacji **Ctrl–r**.
4. Przećwicz mechanizm automatycznego uzupełniania nazw programów i plików:

```
# mk Tab Tab d Tab
```

Znajdź w ten sposób wszystkie programy zaczynające się na pr i x. Uzupełnianie nazw plików przećwicz w swoim katalogu domowym:

```
# ls p Tab
```

Spróbuj wyświetlić zawartość katalogu `/usr/share/doc/HTML/en`, na każdym etapie w maksymalnym stopniu wykorzystując automatyczne uzupełnianie nazw katalogów.

5. Przećwicz przerywanie działania poleceń:

```
# sleep 10
Ctrl–c
```

6. Przećwicz przewijanie zawartości okna terminala kombinacjami **Shift–PgUp**,

Shift–PgDn, **Shift–↑** i **Shift–↓** i (działa również w trybie tekstowym).

7. Zapoznaj się z skrótami klawiszowymi umożliwiającymi edycję linii poleceń:

   Ctrl–a, **Ctrl–e**
   przejście na początek/koniec linii,

   Ctrl–f, **Ctrl–b**
   zmiana pozycji kursora o jeden znak,

   Ctrl–k usunięcie tekstu do końca wiersza,

   Ctrl–w usunięcie poprzedniego słowa.

8. Przetestuj odświeżenie ekranu kombinacją **Ctrl–l** i komendą `clear`.

9. Sprawdź działanie kombinacji **Ctrl–d**.

10. Zastosuj nazwy uogólnione (ang. *glob patterns*) w odwołaniach do plików, stosując znaki specjalne: `*` (dowolny ciąg znaków), `?` (pojedynczy znak) i `[]` (pojedynczy znak ze wskazanego zbioru). W celu wykonania ćwiczenia utwórz komendą `touch` puste pliki o różnych nazwach:

```
# touch a.txt b.txt c.txt
# touch a.dat b.dat ab.dat
# ls *.txt
# ls a*
# ls *.*
# ls *
# ls [ab].txt
# ls ?.dat
# ls *.???
```

    Uwaga: nazwy uogólnione to inny mechanizm niż wyrażenia regularne!

11. Wyświetl listę jednoznakowych programów z katalogu `/usr/bin`. Następnie wyświetl analogiczną listę trzyznakowych programów z tego katalogu. Zastosuj przełącznik `-d` komendy `ls` (zobacz dokumentacja `man`). Wyświetl programy dwuliterowe złożone z liter „a”, „b” i „c”. Wyświetl programy zawierające przynajmniej jedną wielką literę i jedną cyfrę.

2
System plików

2.1 Katalogi i pliki

1. Utwórz przykładowe podkatalogi w swoim katalogu domowym:

   ```
   # mkdir x1
   # mkdir x2 x3
   # mkdir -p x1/x4/x5
   ```

   Wyświetl strukturę katalogów korzystając z komend `ls` i `tree`:

   ```
   # ls -lR
   # tree
   # tree x2
   ```

   Usuń katalogi komendą `rmdir`:

   ```
   # rmdir x1
   # rmdir -p x2
   ```

   Uwaga: W dalszych przykładach argumenty będące katalogami będą oznaczane znakiem „/” na końcu nazwy katalogu. Znak ten można pominąć, ale jest to poprawnie interpretowane przez interpreter poleceń. Oto przykładowe równoważne zlecenia:

   ```
   # ls -l x1
   # ls -l x1/
   ```

2. Utwórz przykładowe pliki komendą `touch` (tworzy pusty plik) lub edytorem `mce-`

dit:

```
# touch a.txt
# touch b.csv c.dat
# mcedit ab.txt
```

3. Przetestuj komendę `cp` służącą do kopiowania katalogów:

```
# cp a.txt b.txt
# cp -i a.txt b.txt
# cp a.txt b.txt x1/
# cp a.txt x1/d.txt
# cp *.txt x2/
# cp -v *.txt x2/
```

Zwróć uwagę na kontekst wykonywania komendy `cp`:

```
# cp a.txt ab
```

Sprawdź jak zachowa się powyższe zlecenie w przypadku, gdy: a) `ab` nie istnieje, b) `ab` jest plikiem zwykłym, c) `ab` jest katalogiem.

4. Przećwicz usuwanie plików komendą `rm`:

```
# rm a.txt b.txt
# rm *.txt
# rm -i *.txt
```

Komenda `rm` umożliwia również usuwanie całych struktur katalogów, łącznie z plikami znajdującymi się w środku:

```
# rm -r x1/
```

Sprawdź co powoduje przełącznik `-f` komendy `rm` (`man rm`).

5. Przećwicz kopiowanie całych struktur katalogowych:

```
# cp -r x1/ x5
```

Sprawdź działanie komendy w przypadku, gdy `x5` nie istnieje oraz gdy `x5` jest katalogiem. Sprawdź działanie przełączników `-T` i `-t` komendy `cp`.

6. Zmień nazwę wybranego pliku:

```
# mv a.txt e.txt
```

Zmień lokalizację pliku:

```
# mv a.txt x2/
```

Wykonaj te akcje jednocześnie:

```
# mv a.txt x2/e.txt
```

Sprawdź działanie przełączników `-v`, `-i`, `-t`, `-T` w przypadku komendy `mv`. Sprawdź działanie komendy `mv` w odniesieniu do katalogów:

```
# mv x1/ x2/
# mv x2/ ac
# cd x3
# mv a.txt ..
# mv ../a.txt .
```

(*) 7. Używając komendy `rename(1)` zmień fragmenty nazw plików na wielu plików jednocześnie:

```
# touch abc.txt abc2.txt test-abc.txt abc-abc.txt
# rename abc ABC *.txt
# rename .txt .dat *.txt
```

(*) 8. Jak usunąć plik o nazwie `-v`?

```
# ls > -v
# rm -v
rm: missing operand
Try `rm --help' for more information.
```

(*) 9. Zapoznaj się z komendą `mmv(1)`:

```
# mmv "*.txt" "#u1.TXT"
```

2.2 Prawa dostępu

1. Zinterpretuj następujące prawa dostępu:

```
-rwxr-xr-x  1 daniel   users         164 Feb 21 17:19 test2
-rw-rw----  1 kamil    students    24250 May 27  2002 dane.txt
-r-----r--  1 marek    users       28014 Feb 21 17:43 przyklad.jpg
-rwxrwxrwx  1 daniel   students     4563 Mar  8 04:43 pomoc.html
-r-xr--r--  1 witek    users        4611 Mar  8 04:42 pasjans
```

   a) Które pliki mogą być modyfikowane przez użytkownika daniel należącego do grup users i students?

   b) Które pliki mogą być czytane przez użytkownika marek będącego członkiem grupy students?

   c) Kto może wykonywać program `pasjans`?

2. Zaobserwuj zmiany w prawach dostępu do wybranego pliku po wykonaniu następujących zleceń dla komendy `chmod`:

```
# ls -l a.txt
# chmod u+x a.txt
# ls -l a.txt
# chmod go-rwx a.txt
# chmod u=rw,g=r,o= a.txt
# chmod u=rwx,go-wx a.txt
```

3. Dla wybranych plików: dodaj prawo zapisu dla grupy, odejmij prawo zapisu dla właściciela, dodaj prawo do wykonywania dla wszystkich użytkowników, ustaw prawa użytkownika na `rwx`, usuń wszystkie prawa dla grupy i pozostałych użytkowników, ustaw prawa dla wszystkich użytkowników na `rw`.
4. Zweryfikuj swoje prawa dostępu do plików `/etc/passwd`, `/etc/shadow`, `/var/mail/xxx`, gdzie `xxx` to twoja nazwa użytkownika.
5. Sprawdź jakie prawa są wymagane do tego, aby wyświetlić zawartość pliku:

   ```
   # ls -l a.txt
   # cat a.txt
   ```

   Sprawdź jakie prawa są wymagane do tego, aby zapisać dane do pliku:

   ```
   # echo "Ala ma kota" > a.txt
   ```

   Sprawdź jakie prawa są wymagane do tego, aby wyświetlić zawartość katalogu:

   ```
   # ls -l x1/
   ```

   Sprawdź jakie prawa są wymagane do tego, aby przejść do katalogu.

   Sprawdź jakie prawa są wymagane do utworzenia pliku w katalogu.

   Sprawdź jakie prawa (i do czego!) są wymagane do zmiany praw dostępu do pliku.
6. Dodaj prawo do zapisu dla grupy dla całej struktury katalogów (działanie rekurencyjne):

   ```
   # chmod -R g+w x1/
   ```

   Komendę wykonaj kilkukrotnie z dodatkowym przełącznikiem `-c`. Przećwicz działanie prawa `x` w odniesieniu do struktur katalogowych:

   ```
   # chmod -R -c g+X x1/
   # ls -l x1/
   # chmod -R -c g+x x1/
   # ls -l x1/
   ```
7. Ustaw prawa dostępu do wybranych plików na takie, jak w przykładzie z punktu 1 korzystając z notacji numerycznej. Oto przykładowe zlecenie zmiany praw dostępu:

   ```
   # chmod 764 a.txt
   ```
8. Wykonaj polecenie `umask` i sprawdź w jaki sposób ma to wpływ na prawa dostępu do nowo tworzonych plików, np.:

   ```
   # umask
   # umask 077
   # touch g.txt
   # ls -l g.txt
   # rm g.txt
   ```

```
# umask 007
# touch g.txt
# ls -l g.txt
```

9. Zapoznaj się z dokumentacją do poleceń `chown` i `chgrp`. Zmień grupę, do której należy wybrany plik na jedną z tych, które są wymieniane poleceniem `id`.
10. Sprawdź prawa dostępu do plików `/usr/bin/passwd`, `/usr/bin/write` i katalogu `/tmp`.
11. Zapoznaj się z komendą `stat`.

2.3 Wyszukiwanie plików

Komenda `locate`

1. Znajdź za pomocą programu `locate` wszystkie pliki, które zawierają w nazwie słowo `print`:

   ```
   # locate print
   ```

   Ogranicz listę plików do tych znajdujących się w podkatalogach katalogu `/usr`:

   ```
   # locate "/usr/*print*"
   ```

   Znajdź wszystkie pliki, których nazwa brzmi dokładnie `print`.
2. Znajdź wszystkie podkatalogi katalogu `/usr` o nazwie `bin`.
3. Znajdź trzyznakowe *programy* znajdujące się w podkatalogach katalogu `/usr`. Znajdź dwuznakowe programy składające się z małych liter „a”, „b”, „c”, „d”, „e”.
4. Policz ile jest dwuznakowych programów w podkatalogach katalogu `/usr`, których nazwy zawierają tylko małe litery.
5. Policz wszystkie pliki, których nazwy kończą się na `.TXT`, następnie wszystkie kończące się na `.txt` i ostatecznie wszystkie, które mają rozszerzenie `.txt` pisane z użyciem dowolnej kombinacji wielkich i małych liter. Sprawdź czy suma liczebności pierwszych dwóch grup jest równa liczebności trzeciej grupy.
6. Znajdź wszystkie programy, których nazwa zawiera co najmniej jedną cyfrę.

Komenda `find`

1. Korzystając z programu `find`: wyszukaj w podkatalogach katalogu `/usr/share` pliki z rozszerzeniem `.TXT`:

   ```
   # find /usr/share -name "*.TXT"
   ```

   Przeszukując te same katalogi, znajdź pliki dwuliterowe składające się z wielkich liter.

2. Znajdź wszystkie podkatalogi katalogu `/usr/share`:

   ```
   # find /usr/share -type d
   ```

   Znajdź dwuliterowe katalogi, których nazwy składają się z wielkich liter. Znajdź wszystkie podkatalogi katalogu `/usr` o nazwie `bin`.

3. Zastosuj negację wyszukując w podkatalogach katalogu `/usr/share` pliki nie zawierające kropki w nazwie:

   ```
   # find /usr/share ! -name "*.*"
   ```

   Wyszukaj w katalogu `/usr` pliki specjalne, a więc pliki, które nie są plikami zwykłymi i nie są katalogami. Sprawdź czy wśród tych plików są inne pliki niż dowiązania symboliczne (posiadające „`l`” z przodu).

4. Wyszukaj w katalogu `/usr` pliki o zerowym rozmiarze:

   ```
   # find /usr -size 0
   ```

   Wyszukaj pliki o rozmiarze od 100 do 200 bajtów. Wyszukaj największy plik w podkatalogach `/usr`.

5. Wyświetl informację szczegółową o każdym znalezionym pliku:

   ```
   # find /usr -size 0 -exec ls -l {} \;
   # find /usr -size 0 -ls
   ```

   Skopiuj wszystkie pliki o rozmiarach 256 bajtów do podkatalogu `pliki` w swoim katalogu domowym.

6. Korzystając z kryteriów czasowych (`-mtime`, `-atime`, `-ctime`) wyszukaj: a) pliki z katalogu `/usr`, które zostały zmodyfikowane w przeciągu ostatniego tygodnia, b) pliki, do których nie było dostępu przez ostatni miesiąc, c) pliki ze swojego katalogu domowego, których status zmienił się dzisiaj. Przetestuj również działanie przełączników `-mmin`, `-amin` i `-cmin`.

7. Korzystając z kryterium `-perm` wyszukaj w katalogu `/usr` pliki z ustawionym bitem SUID. Wyszukaj w swoim katalogu domowym pliki, do których mają prawo zapisu użytkownicy inni niż właściciel.

8. Wyszukaj swoje pliki w katalogu `/tmp`.

9. Stosując alternatywę kryteriów (`-o`) zlokalizuj pliki pasujące do różnych wzorców, a więc np. pliki tymczasowe: `*.bak` i `*~`. Rozbuduj zlecenie tak, aby jego efektem było usunięcie tych plików.

10. Dodaj prawo „x” dla pozostałych użytkowników dla wszystkich plików ze swojego katalogu domowego, które mają ustawione to prawo dla grupy.

2.4 Dowiązania

1. Utwórz dowiązanie twarde do wybranego pliku. Zmodyfikuj zawartość pliku oryginalnego i sprawdź zawartość pliku-dowiązania (i odwrotnie). Zmień prawa dostępu do dowiązania i porównaj je z oryginalnym plikiem. Sprawdź nr i-węzła obu plików za pomocą polecenia `ls -i`. Utwórz dowiązanie twarde do katalogu. Usuń dowiązanie lub oryginalny plik obserwując zmianę licznika dowiązań (druga kolumna wyników `ls -l`).
2. Utwórz dowiązanie symboliczne do pliku. Sprawdź i zmień prawa dostępu do dowiązania. Usuń plik oryginalny i wyświetl programem `more` zawartość pliku-dowiązania. Utwórz dowiązanie symboliczne do katalogu. Utwórz dowiązanie symboliczne do nieistniejącego pliku. Sprawdź na co wskazują dowiązania symboliczne: `/dev/cdrom`, `/dev/mouse` i `/usr/X11`.
3. Sprawdź czy można utworzyć dowiązanie twarde do dowiązania symbolicznego.

2.5 Blokowanie dostępu do plików

1. Przetestuj mechanizm zakładania blokad uruchamiając dwa konfliktowe zlecenia. Pierwsze z nich powoduje zajęcie pliku na 10 sekund:

   ```
   # flock -e test.txt sleep 10
   ```

   Komenda `flock` powoduje założenie blokady na wskazanym pliku (`test.txt`) i wykonanie wskazanej komendy (`sleep 10`). Podczas wykonywania polecenia uruchom w innym oknie zlecenie:

   ```
   # flock -e test.txt cat test.txt
   ```

2. Sprawdź zgodność blokad *shared* i *exclusive*.
3. Skonstruuj zlecenie bezpiecznie dopisujące do pliku listę procesów użytkownika.

3
Edytory

3.1 Edytor `vi`

1. Wstawianie tekstu: komendy `i`, `a` i `R` zakończone wciśnięciem `Esc`. Przećwicz nawigację po przykładowym dokumencie: komendy `j`, `k`, `h`, `l`, `0`, `$`, `G`. Przemieszczanie się do następnego/poprzedniego słowa: `w`, `e`, `b`.
2. Usuwanie tekstu: komendy `x`, `X` i `dd`. Wycofywanie ostatnich operacji: komenda *undo* `u` i *redo* `Ctrl-r`.
3. Wyszukiwanie tekstu komendami `/` i `?`. Przejście do następnego/poprzedniego wystąpienia wzorca realizują komendy n i N. Posługując się wyrażeniami regularnymi wyszukaj w tekście:
   - linie zaczynające się od znaku „#”,
   - puste linie,
   - niepuste linie niebędące komentarzem,
   - dowolne liczby,
   - liczby heksadecymalne języka C,
   - ciągi spacji dłuższe niż 1 znak.
4. Kopiowanie tekstu: komendy `p`, `P` i `yy`.
5. Parametryzowanie komend poprzez znaczniki, np.: `d0`, `dG`, `d$`. Komenda `c` i `y` w połączeniu ze znacznikami. Parametryzowanie numeryczne: `10G`, `5i`,

d5d, **d2w**, **d3k**, **d10h**, **d/x**.

6. Operacje na plikach: zapisywanie **:w**, wstawianie zewnętrznego pliku **:r**. Opuszczanie edytora **:q**, **:q!**, z zapisem: **:x**, **ZZ**.
7. Konfiguracja edytora: komendy **:set number**, **:set autoindent**. Konfiguracja w pliku `~/.exrc`.
8. Rozszerzenia edytora `vim`: zaznaczanie tekstu **V**, **v**, **Ctrl-v** i jedna z komend: **d**, **y**, **c**. Podświetlanie składni **:syntax on**.

3.2 Zaawansowane funkcje edytora vim

```
:help g
:1,10s/old/new/
:/^#/ s/old/new/
:g/^$/d
:v/^#/d
ge
gI
gJ
g$
g0
:e .
:Sex
:sh
:tabnew
:gt
```

3.2.1 Zamiana tekstów

1. Zamień wszystkie wystąpienia słowa „edytor” na „editor”:

   ```
   :% s/edytor/editor/g
   ```

   Wykonaj analogiczną zmianę tylko w: (a) pierwszych 10 liniach, (b) liniach, które są komentarzem.
2. Umieść wszystkie liczby w pliku w nawiasach klamrowych, np.:

   ```
   142 --> {142}
   ```
3. Wykonaj działanie odwrotne do przedstawionego powyżej, a więc usuń nawiasy klamrowe, jeśli otaczają liczbę.
4. Zamień otoczenie słów znakami „*” otoczeniem znacznikami HTML `<b>`, np.:

   ```
   To jest *kot*.  -->  To jest <b>kot</b>.
   ```

5. Zaimplementuj obsługę formatowania odnośników w zapisie Wiki na format HTML:

```
[tytuł|url]  -->  <a href="url">tytuł</a>
```

3.3 Edytor mcedit

https://midnight-commander.org/wiki/doc/editor/hotkeys

3.4 Edytor emacs

1. Wczytaj dokument *Tutorial* komendą **Ctrl–h t**. Przemieszczaj się po tekście komendami **Ctrl–f**, **Ctrl–b**, **Ctrl–n**, **Ctrl–p**, **Ctrl–v**, **Alt–v**, **Ctrl–a**, **Ctrl–e**, **Alt–f**, **Alt–b**, **Alt–a**, **Alt–e**, **Alt–>**, **Alt–<**. Usuwanie tekstu: **Ctrl–d**, **Ctrl–k**, **Alt–k**.
2. Kopiowanie tekstu: **Ctrl–space**, przesuń kursor, **Ctrl–w** i wstaw w innym miejscu: **Ctrl–y**. Wycofanie operacji: **Ctrl–x u** (lub **Ctrl–_**).
3. Wczytywanie pliku **Ctrl–x Ctrl–f**. Zapis pliku **Ctrl–x Ctrl–s**.
4. Wyszukiwanie tekstów: **Ctrl–s**, **Ctrl–r**.
5. Wyjście z edytora: komenda **Ctrl–x Ctrl–c**.

4
Procesy

4.1 Lista procesów

1. Wyświetl listę własnych procesów komendą `ps`. Porównaj wyniki z wynikami poleceń: `ps -x` i `ps -ax`. Zbadaj działanie przełączników `-l` i `-u`. Zaloguj się do systemu kilkukrotnie poprzez wirtualne konsole lub otwierając nowe okno w środowisku graficznym. Sprawdź poleceniem `tty` nazwę terminala, na którym pracujesz.
2. Znajdź proces macierzysty dla procesu `ps`. Odszukaj przodka wszystkich procesów i sprawdź jego identyfikator. Wyświetl hierarchię procesów poleceniem `pstree`.
3. Obejrzyj listę procesów poleceniem `top`. Przećwicz następujące komendy:

P	sortowanie wg zajętości procesora
M	sortowanie wg zajętości pamięci
u	procesy danego użytkownika
1	informacje szczegółowe dot. każdego procesora/rdzenia
q	wyjście

   Zapoznaj się również z alternatywnymi wersjami programu: `htop` oraz `atop`.
4. Sprawdź obciążenie systemu komenda `uptime`. Uruchom program `xload` do monitorowania obciążenia.
5. Za pomocą polecenia `pgrep` wyświetl identyfikatory wszystkich swoich inter-

preterów poleceń oraz wszystkich procesów użytkownika root. Sprawdź również działanie komendy `pidof`.

4.2 Sygnały

1. Zapoznaj się z listą sygnałów na stronie pomocy systemowej signal. Uruchom program `sleep`:

   ```
   # sleep 100
   ```

   i wysyłaj do niego komendą `kill` kolejne sygnały: HUP(1), INT(2), TERM(15), QUIT(3), KILL(9), np.:

   ```
   # kill -2 12345
   # kill -INT 12345
   ```

   Uruchom sesję edytora `vi` i sprawdź reakcje tego programu na wymienione sygnały.
2. Zbadaj działanie poleceń `killall` i `pkill` .

4.3 Priorytety procesów

1. Obniż wybranemu procesowi priorytet poprzez zmianę wartości *nice*:

   ```
   # renice +5 12345
   ```

2. Uruchom w interpreterze poleceń pętlę nieskończoną i sprawdź obciążenie procesora komendą `top`:

   ```
   # while true; do : ; done
   ```

   Uruchom $n + 1$ powyższych pętli, gdzie n jest liczbą dostępnych rdzeni procesora. Następnie zmień wartość *nice* dla procesu interpretera, który konkuruje z innym w dostępie do procesora i obserwuj przydział czasu procesora dla tych procesów. Zwróć uwagę na obciążenie systemu (*load*).
3. Zmień priorytet interpretera poleceń i sprawdź priorytety procesów potomnych tego interpretera.
4. Uruchom nowy proces ze zmienionym priorytetem:

   ```
   # nice -10 sleep 100
   ```

4.4 Obsługa wielu procesów w interpreterze poleceń

1. Wstrzymaj sesję edytora `vi` kombinacją **Ctrl–z** i wznów ją komendą `fg`. Uruchom nową sesję i wstrzymaj ją również. Wyświetl aktywne sesje komendą `jobs`. Wznów pracę pierwszego procesu `vi` komendą `fg`:

   ```
   # fg %1
   ```

2. Uruchom proces `sleep`, wstrzymaj jego pracę i wznów jego wykonywanie w tle (komenda `bg`). Wystartuj nowy proces `sleep` w tle:

   ```
   # sleep 100 &
   ```

   Następnie przełącz go do pracy w trybie pierwszoplanowym i później znów do pracy w tle.

3. Wstrzymaj proces `sleep` poprzez wysłanie mu sygnału STOP, a następnie wznów jego pracę poprzez wysłanie sygnału CONT.

4. Uruchom program `screen` zarządzający sesjami terminalowymi:

   ```
   # screen -d -R
   ```

   Uruchom edytor `vi`, a następnie stwórz nowy terminal kombinacją **Ctrl–a c**. Uruchom program `mc` w nowym terminalu. Przełączaj się między terminalami kombinacją **Ctrl–a n** lub **Ctrl–a Spacja**. Wyświetl listę dostępnych terminali kombinacją **Ctrl–a "** lub **Ctrl–a w**. Przećwicz szybkie przełączanie się między oknami o konkretnych numerach: **Ctrl–a 0** … **Ctrl–a 9**. Wyjdź z programu `screen` kombinacją **Ctrl–a DD** i następnie odtwórz sesję uruchamiając program tak samo jak na początku.

5 Potoki

5.1 Filtr `cat`

1. Uruchom program `cat` bez argumentów i wpisz kilka linii tekstu:

```
# cat
Ala ma
Ala ma
kota
kota
^D
```

2. Uruchom program `cat` kierując standardowe wyjście do pliku:

```
# cat > a.txt
Ala ma
kota
^D
```

3. Wyświetl zawartość pliku:

```
# cat a.txt
# cat < a.txt
# cat - < a.txt
```

4. Co powoduje poniższe wywołanie?

```
# cat < a.txt > b.txt
```

5. Sprawdź efekty poniższego zlecenia:

```
# cat a.txt b.txt > c.txt
```

6. Przetestuj dopisywanie danych do pliku:

```
# cat a.txt >> b.txt
```

7. Przetestuj definiowanie danych wejściowych w linii poleceń:

```
# cat << EOF
> Ala ma
> kota
> EOF
Ala ma
kota
```

8. Wyświetl ponumerowaną zawartość pliku:

```
# cat -n a.txt
```

9. Zapoznaj się z komendami `dog` i `tac`.

5.2 Filtry `head`, `tail`

1. Wyświetl początkowe linie z pliku `a.txt`:

```
# head a.txt
# head < a.txt
# cat a.txt | head
```

2. Wyświetl informacje o najstarszych (i następnie najnowszych) 10 procesach.
3. Wyświetl linie od 3 do 5 z pliku `/etc/passwd`. Wyświetl przedostatnią linię z tego pliku.
4. Zapisz w pliku `wynik` pierwsze linie ze wszystkich plików `txt` z bieżącego katalogu.
5. Wyświetl wszystkie linie z pliku `a.txt` oprócz pierwszej i ostatniej.
6. Przetestuj monitorowanie zawartości pliku `a.txt` uruchamiając w jednym oknie:

```
# tail -f a.txt
```

a w drugim dodając nowe linie do tego pliku:

```
# echo "Ala ma kota" >> a.txt
# ps x >> a.txt
```

5.3 Filtr grep

1. Wyświetl listę podkatalogów katalogu bieżącego. Wyświetl informacje o dowiązaniach symbolicznych z katalogu /usr/bin.
2. Wyświetl informacje o takich plikach zwykłych z katalogu bieżącego, które mają ustawione prawo zapisu dla grupy.
3. Wyświetl z pliku a.txt linie nie zawierające żadnych liczb.
4. Wyświetl z pliku a.txt linie nie będące komentarzem, a więc linie nie zaczynające się od znaku „#”.
5. Wyświetl nazwy plików txt, w których (nie) są zawarte dowolne polskie litery „ą”, „ć” lub „ę”.
6. Pokaż 3-linijkowy kontekst wystąpienia słowa „abc” w pliku a.txt.
7. Przeszukaj swój cały katalog domowy w poszukiwaniu plików zawierających słowo „*abc*”.
8. Przeszukaj swój cały katalog domowy w poszukiwaniu plików txt zawierających słowo „*abc*”.
9. Skopiuj ze swojego konta wszystkie pliki txt, które zawierają słowo „*abc*” i były modyfikowane w ostatnim tygodniu, do katalogu /tmp.
10. Zapoznaj się w dokumentacji z różnicami pomiędzy programami grep, egrep i fgrep.

5.4 Filtr wc

1. Policz wszystkie procesy działające w systemie.
2. Policz dowiązania symboliczne w katalogu /usr/bin.
3. Zlicz wszystkie pliki zwykłe znajdujące się w katalogu /etc i jego podkatalogach (2 sposoby).
4. Oblicz długość trzeciej linii pliku /etc/passwd.
5. Oblicz liczbę pustych linii w pliku /etc/modprobe.conf.
6. Oblicz sumaryczną liczbę linii we wszystkich plikach txt w katalogu. W drugim kroku uwzględnij również pliki txt znajdujące się w podkatalogach.

5.5 Filtr tr

1. Zamień wszystkie małe litery na wielkie w wynikach komendy ls:

```
# ls -l | tr a-z A-Z
```

2. Zaszyfruj plik stosując algorytm ROT13:

   ```
   # cat a.txt | tr a-z n-za-m
   ```

   Przepuść dwukrotnie dane wejściowe przez powyższy filtr kodujący.
3. Wyświetl zawartość pliku `a.txt` usuwając polskie litery.
4. Wyświetl zawartość pliku `a.txt` w jednej linii:

   ```
   # cat a.txt | tr  '\n'  ' '
   ```
5. Rozbij tekst z pliku `a.txt` tak, aby każde słowo było w oddzielnej linii. Dodatkowo usuń wszelkie znaki interpunkcyjne:

   ```
   # cat a.txt | tr -d '.,:;!?'
   ```
6. Usuń powtarzające się spacje z danych wejściowych:

   ```
   # ls -l | tr -s ' '
   ```

   Jednocześnie włącz zamianę spacji na znaki tabulacji:

   ```
   # ls -l | tr -s ' ' '\t'
   ```
7. Dokonaj konwersji pliku tekstowego zapisanego w konwencji DOS/Windows (CR LF) do konwencji Unix (LF) poprzez usunięcie znaków CR. Plik w konwencji DOS/Windows można utworzyć programem `vi` ustawiając opcję:

   ```
   :set fileformat=dos
   ```

5.6 Filtr `cut`

1. Wyświetl listę praw dostępu do plików w aktualnym katalogu:

   ```
   # ls -l | cut -d ' ' -f 1
   ```
2. Wyświetl nazwy i opisy użytkowników z pliku `/etc/passwd` (pola 1 i 5).
3. Wyświetl rozmiary plików z bieżącego katalogu.
4. Wyświetl rozmiary plików z bieżącego katalogu zachowując wyrównanie liczb do prawej (spacje po lewej).

5.7 Filtr `sort`

1. Wyświetl posortowaną zawartość pliku `a.txt`:

   ```
   # sort a.txt
   ```
2. Posortuj trwale zawartość pliku `a.txt`.
3. Przećwicz sortowanie pliku zawierającego liczby, np.:

```
50
9
2000
100
```

4. Wyświetl listę plików w aktualnym katalogu, posortowaną według rozmiaru pliku[1]. Wyświetl same nazwy plików w tej samej kolejności.
5. Wyświetl listę nazw użytkowników z pliku `/etc/passwd` posortowaną wg numerów UID w kolejności od największego do najmniejszego.

5.8 Filtr `uniq`

1. Przećwicz działanie filtru `uniq` dla następujących danych wejściowych:

```
b
c
c
a
b
```

2. Wyświetl listę użytkowników posiadających dowolny proces w systemie.
3. Wyświetl statystykę liczb procesów uruchomionych przez poszczególnych użytkowników aktywnych w systemie.
4. Wyświetl nazwy użytkowników posiadających co najmniej dwa procesy w systemie.
5. Wyświetl nazwy (tylko nazwy!) maksymalnie 3 użytkowników posiadających najwięcej procesów w systemie.
6. Wyświetl liczby plików utworzonych przez poszczególnych użytkowników w podkatalogach katalogu `/tmp`.
7. Wypisz 3 najczęściej powtarzające się słowa z pliku `a.txt`.
8. Podaj w kolejności alfabetycznej nazwy trzech najmniejszych plików z bieżącego katalogu.
9. Wyświetl nazwy zalogowanych w systemie użytkowników, którzy maja uruchomiony program `vi`.
10. Wyświetl nazwy użytkowników będących właścicielami 10 procesów, które zajmują najwięcej pamięci w systemie.
11. Korzystając z komendy `history`, wyświetl statystykę ostatnio używanych komend (bez argumentów).

[1] Rozmiary plików mogą być prezentowane w formie bardziej czytelnej dla człowieka: 10K, 5M, 3G, co umożliwia przełącznik `-h` komendy `ls`. Tego typu wartości mogą następnie być adekwatnie interpretowane jako liczby przez program `sort` po użyciu analogicznego przełącznika `-h`.

6
Interpreter poleceń

6.1 Grupowanie poleceń

1. Zbadaj status zakończenia różnych poleceń:

```
# true
# echo $?
0
# false
# echo $?
1
# touch a.txt
# rm a.txt
# echo $?
0
# rm a.txt
# echo $?
1
```

   Sprawdź wartości zwracane przez polecenia `grep` i `find`.

2. Sprawdź reakcję na przerwanie programu:

```
# sleep 30
^C
# echo $?
130
```

Co reprezentuje zwracana wartość? Sprawdź status zakończenia w przypadku przerywania programu innymi sygnałami.

3. Przećwicz wykonywanie sekwencji poleceń:

```
# sleep 5; echo Koniec
Koniec
```

4. Przećwicz warunkowe wykonywanie poleceń, np.:

```
# grep -q ab a.txt && echo Jest
# grep -q ab a.txt || echo Brak
# grep -q ab a.txt && echo Jest || echo Brak
```

Napisz zlecenie, które dopisze do pliku `a.txt` słowo „koniec” jeżeli go tam nie ma lub wypisze napis „jest” w przeciwnym wypadku.

5. Przećwicz grupowanie poleceń:

```
# grep -q ab a.txt || (echo "ab" >> a.txt && echo "Plik zaktualizowany")
# grep -q ab a.txt || { echo "ab" >> a.txt && echo "Plik zaktualizowany"; }
```

Różnice pomiędzy nawiasami () i {}:

```
# (cd /usr; pwd); pwd
# { cd /usr; pwd; }; pwd
```

Spowoduj uruchomienie w tle z 5-sekundowym opóźnieniem programu `xload` bez blokowania interpretera poleceń.

6. Napisz zlecenie, które zaśnie na 2 sekundy, następnie usunie plik `log` i jeżeli usunięcie zakończy się poprawnie – wystartuje program `konsole` w tle.

6.2 Przekierowania strumieni standardowych

1. Przećwicz przekierowania strumieni standardowych w interpreterze Bash:

```
# find /etc -name mtab > out.txt
# find /etc -name mtab 2> out.txt
# find /etc -name mtab 1> out.txt
# find /etc -name mtab > out1.txt 2> out2.txt
# find /etc -name mtab > out.txt 2> out.txt
# find /etc -name mtab 2> /dev/null
# find /etc -name mtab > out.txt 2>&1
# (find /etc -name mtab 2>&1) > out.txt
```

2. Przećwicz przekierowania strumieni standardowych do potoków:

```
# find /etc -name mtab | head -n 3
# find /etc -name mtab | head -n 3 > out
# find /etc -name mtab 2> /dev/null | head -n 3
# find /etc -name mtab 2>&1 | head -n 3
```

```
# find /etc -name mtab |& head -n 3
# find /etc -name mtab 2>&1 > /dev/null | head -n 3
# (find /etc -name mtab > /dev/null) |& head -n 3
```

a) Zapisz pierwsze 3 linie ze standardowego wyjścia programu `find` do pliku `out.txt`.

b) Wypisz pierwsze 3 linie produkowane przez program `find` bez względu na to z jakiego strumienia danych pochodzą.

c) Zapisz pierwsze 3 komunikaty o błędach generowanych przez program `find` do pliku `out.txt`.

6.3 Zmienne środowiskowe

1. Ustaw zmienną środowiskową i wyświetl jej wartość:

```
# x=abc
# echo $x
abc
# x=out.txt
# cat $x
```

2. Sprawdź wartość zmiennej `y` po wykonaniu poniższych podstawień:

```
# x=5
# y=Ala ma $x kotów
# y="Ala ma $x kotów"
# y='Ala ma $x kotów'
# y=Ala\ ma\ $x\ kotów
```

Podstaw do zmiennej inne znaki specjalne interpretera: `*`, `?`, `[`, `]`, `(`, `)`, `{`, `}`, `<`, `>`.

3. Usuń zmienną:

```
# unset x
```

4. Sprawdź dostępność zmiennych w potomnych procesach:

```
# x=10
# echo $x
10
# konsole
# echo $x
# exit
# export x
# konsole
# echo $x
10
# exit
```

Wyświetl listę wszystkich eksportowanych zmiennych środowiskowych komendą `env`.

5. Wykonaj komendę zawartą w zmiennej środowiskowej:

```
# x=ls
$x
# x="ls -l"
$x
# x="sleep 2; ls"
$x
```

Zastosuj funkcję `eval` do wykonywania fragmentów kodu interpretera:

```
# eval $x
```

6. Zmień wartość zmiennej HOME i sprawdź efekty:

```
# HOME=/tmp
# cd
# pwd
```

7. Zmień wartość zmiennej środowiskowej PS1 i zaobserwuj efekty. Zmień również wartość zmiennej PS2 i przetestuj jej zastosowanie wykonując:

```
# ls -l \
> /usr
```

8. Wyświetl a następnie zmień wartość zmiennej TERM kolejno na wartości: `vt100`, `ansi` i `dumb`. Po każdej zmianie sprawdź poprawność pracy programów pełnoekranowych np. `vi`, `mc`.

9. Wyświetl zmienną środowiskową PATH i następnie zmień jej wartość:

```
# PATH=/tmp
# ls
bash: ls: No such file or directory
# PATH=/bin:/usr/bin
```

Skopiuj plik `/bin/ls` do katalogu bieżącego pod nazwę `ps`, a następnie próbuj wykonywać komendę `ps` przy różnych ustawieniach zmiennej PATH:

```
# ps
# ./ps
# PATH=/bin:/usr/bin
# ps
# PATH=/bin:/usr/bin:.
# ps
# PATH=.:/bin:/usr/bin
# ps
```

10. Sprawdź jaki edytor jest uruchamiany po wciśnięciu klawisza **F4** w programie Midnight Commander (`mc`). Następnie wyłącz opcję wykorzystywania we-

wnętrznego edytora `mc`: *Options* ▷ *Configuration* ▷ *use internal edit* i ponownie sprawdź rodzaj uruchamianego edytora. Wyjdź z programu `mc` (**F10**), ustaw wartość zmiennej EDITOR na `pico` i ponownie uruchom `mc` sprawdzając domyślny edytor.

11. Dopisz do plików konfiguracyjnych `.bashrc` i `.bash_profile` zlecenia następującej postaci:

```
echo "To jest plik .bashrc"
```

Następnie zaloguj się do systemu w trybie tekstowym i sprawdź, które komendy i w jakiej kolejności są wykonywane. Dopisz do odpowiedniego pliku definicje wybranych zmiennych. Zweryfikuj poprawność ustawień poprzez uruchomienie w *bieżącym* interpreterze komend zawartych w pliku konfiguracyjnym:

```
# . .bashrc
# source .bashrc
```

12. Przechwytywanie tekstu z wyjścia standardowego procesu:

```
  sh# x=`hostname -f`
bash# x=$(hostname -f)
bash# x=$(grep -l abc $(find . -type f -name "*.txt"))
```

6.4 Obliczenia w powłoce

1. Przećwicz działanie programu `expr`:

```
# expr 2 + 2
4
# expr 3 \* 5
15
# expr \( 2 + 3 \) \* 5
25
```

2. Zmodyfikuj wartość zmiennej środowiskowej:

```
# x=`expr $x + 1`
# x=$(expr $x \* 3)
```

3. Zastosuj mechanizm interpretera Bash do wykonywania obliczeń:

```
# echo $((2+2))
# echo $((3*5))
# x=$(( ($x+1)*3 ))
```

4. Zapoznaj się z interpreterem obliczeniowym `bc`:

```
# bc
2+5
7
```

```
2.3/(0.779+0.123)
2
scale=3
2.3/(0.779+0.123)
2.549
x=24
y=36
sqrt(x^2+y^2)
43.26
quit
# man bc
```

6.5 Aliasy

1. Zdefiniuj alias:

   ```
   # alias l="ls -l"
   ```

   Przetestuj przekazywanie argumentów do zlecenia, które jest aliasem.
2. Sprawdź typy różnych komend dostępnych z poziomu interpretera, np.: `ls`, `ps`, `echo`, `kill`.

6.6 Interpreter csh

1. Uruchom interpreter `csh` i sprawdź jego działania wykonując kilka prostych komend.
2. Zdefiniuj zmienną lokalną i eksportowaną w interpreterze `csh`:

   ```
   csh# set x=abc
   csh# echo $x
   csh# setenv y abc
   csh# sh
    sh# echo $y
   ```

7 Wprowadzenie do programowania w języku interpretera Bourne'a

7.1 Podstawy

1. Zapisz do pliku `test.sh` treść pierwszego skryptu:

   ```
   echo "Hello world"
   ```

   Wykonaj skrypt jawnie uruchamiając interpreter poleceń:

   ```
   # sh test.sh
   Hello world
   ```

   Wykonaj skrypt tak, jak inne programy:

   ```
   # ./test.sh
   ```

   Nadaj sobie prawo do wykonywania skryptu i ponów próbę.

2. Dodaj do skryptu w pierwszej linii wskazanie na interpreter poleceń:

   ```
   #!/bin/sh
   ```

   Sprawdź czy niepoprawna wartość tego specjalnego komentarza umożliwia wy-

konanie skryptu.

3. Dodaj do skryptu fragment kodu wyświetlający pierwsze 3 argumenty. Pierwszy argument można wyświetlić poniższym zleceniem:

```
echo "Arg 1 = $1"
```

Sprawdź wartości zmiennych pozycyjnych dla następujących argumentów skryptu:

```
a b c
"a b" c
a\ b c
*
"*"
\*
```

Sprawdź zawartość zmiennej `$0` oraz `$#`.

4. Odwołaj się do argumentów powyżej `$9` stosując polecenie `shift`:

```
echo "Arg 1 = $1"
shift
echo "Arg 1 = $1"
```

Zachowaj wartość argumentu `$1` w dodatkowej lokalnej zmiennej:

```
x=$1
shift
echo "x=$x  arg1=$1"
```

5. Sprawdź odwołanie do argumentów pozycyjnych w interpreterze Bash:

```
echo "Arg 10 = ${10}"
```

Zastosowanie interpretera Bash wymaga wskazania go w pierwszej linii skryptu:

```
#! /bin/bash
```

7.2 Instrukcja warunkowa

1. Przećwicz działanie programu testującego `test`:

```
# test ab = ab
# echo $?
0
# test ab = cd
# echo $?
1
```

Przećwicz działanie operatorów numerycznych: `-eq` (=), `-ne` (≠), `-gt` (>), `-ge` (≥), `-lt` (<), `-le` (≤), np.:

```
# test 5 -gt 3
```

Przećwicz operatory odwołujące się do systemu plików: `-f` (plik zwykły), `-d` (katalog), `-L` (dowiązanie symboliczne), `-r` (prawo do odczytu), `-w` (prawo do zapisu), `-x` (prawo do wykonywania), np.:

```
# test -d dir1
```

Przećwicz testowanie negacji warunków, np.:

```
# test ! -f a.txt
```

Przećwicz definiowanie warunków złożonych:

```
# test -f nowy -a -w nowy
```

2. Przećwicz działanie poniższego przykładu użycia instrukcji warunkowej sprawdzającej dostępność pierwszego argumentu przekazanego do skryptu.

```
if test -n $1
then
  echo "Arg 1 = $1"
else
  echo "Brak argumentu"
fi
```

Gdzie jest błąd w przedstawionej implementacji?

3. Zastąp wywołanie programu `test` programem `[` (program jest dostępny w pliku `/usr/bin/[`):

```
if [ -n "$1" ]
...
```

4. Przećwicz instrukcję warunkową dostępną w interpreterze Bash:

```
if [[ -n "$1" && -n "$2" ]]
...
```

5. Napisz skrypt dopisujący napis przekazany pierwszym argumentem do pliku wskazanego drugim argumentem. Skrypt powinien obsługiwać różne rodzaje błędów, które mogą się pojawić podczas jego wykonania, a więc:
   a) złą liczbę argumentów wywołania,
   b) brak pliku wskazywanego drugim argumentem,
   c) brak prawa zapisu do pliku wskazanego drugim argumentem.

   Wystąpienie błędu powinno być sygnalizowane odpowiednim komunikatem i zakończeniem skryptu ze statusem 1.

6. Korzystając z linii poleceń, przećwicz zapis instrukcji warunkowej w jednej linii.

7.3 Pętla for

1. Przećwicz działanie przykładowej pętli for:

```
for x in a b c
do
  echo "$x"
done
```

   W przykładzie wytłuszczono słowa kluczowe.

2. Przećwicz iterację po nazwach plików pasujących do określonego wzorca, np.:

```
for f in *.txt
```

3. Stwórz pętlę iterującą po wartościach numerycznych:

```
for x in `seq 1 10`
```

4. Napisz skrypt, który wyświetli ze wskazanego argumentem katalogu wszystkie podkatalogi iterując po kolejnych plikach i katalogach.

5. Rozbuduj skrypt z zadania 5 z punktu 7.2 tak, aby drugi argument mógł być nazwą rozszerzenia dla plików, które będą uzupełniane zawartością pliku wskazanego pierwszym argumentem. Przykładowe wywołanie skryptu może wyglądać następująco:

```
# dopisz.sh "Ala ma kota" txt
```

7.4 Pętle while i until

1. Ogólna postać pętli while jest następująca:

```
while warunek
do
  polecenia
done
```

   gdzie warunek jest wyrażany tak samo jak w przypadku instrukcji warunkowej. Pętla until jest niemal identyczna – różni ją jedynie odwrotna interpretacja warunku. Pętla while wykonuje się dopóki warunek jest spełniony, a pętla until dopóki warunek jest niespełniony:

```
until warunek
do
  polecenia
done
```

2. Napisz skrypt wyświetlający w kolejnych liniach argumenty przekazane do skryptu.

3. Rozbuduj skrypt z zadania 5 z punktu 7.3 tak, aby możliwe było wywołanie skryptu z dowolną liczbą rozszerzeń plików przekazywanych w kolejnych argumentach, np. w ten sposób:

```
# dopisz.sh "Ala ma kota" txt bak dat
```

4. Napisz skrypt sortujący zawartości wszystkich przekazanych argumentami plików.
5. Przećwicz przerywanie wykonywania pętli instrukcją `break`:

```
while true
do
  if [ $x -eq 0 ]
  then
    break
  fi
done
```

Sprawdź również działanie instrukcji sterującej `continue`.

7.5 Wczytywanie wartości

1. Przećwicz działanie komendy `read`:

```
# read x
Ala ma kota
# echo $x
# read x y
Ala ma kota
# echo $x
# echo $y
```

Zmień ciąg znaków separujący słowa i przećwicz czytanie pól z pliku `/etc/passwd`:

```
IFS=":"
```

2. Wyświetl informacje o procesach należących do użytkowników o nazwach wymienionych w kolejnych liniach wskazanego argumentem pliku. Informacje wyświetl w dwóch wyrównanych kolumnach: nazwa użytkownika i PID procesu.
3. Przećwicz 2 schematy przetwarzania zawartości plików:

```
cat a.txt |
while read LINIA
do
  echo "$LINIA"
done
```

Drugi sposób:

```
while read LINIA
do
  echo "$LINIA"
done < a.txt
```

Przećwicz obliczanie liczby linii i znaków w pliku stosując oba podejścia.

4. Przetestuj działanie programu `dialog` zapoznając się z pomocą dostępną po wyspecyfikowania przełącznika `--help`:

```
# dialog --help
```

Istnieją też odpowiedniki programu `dialog` dla trybu graficznego, np. `zenity`.

7.6 Funkcje

1. Zapisz w skrypcie funkcję oraz jej wywołanie:

```
function foo() {
  echo "Funkcja"
  echo "Argument: $1"
}

foo a
foo "Hello world"
```

2. Zweryfikuj zakres widzialności zmiennych w funkcjach:

```
function foo() {
  local y
  echo "x=$x  y=$y"
  x=5
  y=6
  echo "x=$x  y=$y"
}

x=1
y=2
foo
echo "x=$x  y=$y"
```

3. Przećwicz wcześniejsze kończenie funkcji:

```
function foo() {
  if [ ! -n "$1" ]
  then
    return 1
  fi
  ...
}
```

7.7 Zadania zaawansowane

1. Napisz skrypt zmieniający rozszerzenia wszystkich plików w katalogu bieżących z wartości wskazanej pierwszym argumentem na wartość wskazaną drugim argumentem. Skorzystaj z polecenia `basename`. Przykładowe wywołanie:

   ```
   # zmien.sh txt doc
   a.txt  =>  a.doc
   dane.txt  =>  dane.doc
   ```

2. Napisz skrypt wyświetlający informacje o plikach zwykłych w następującym formacie:

   właściciel atrybuty rozmiar nazwa

   przy czym pole *właściciel* może przyjmować następujące wartości: *moj*, jeśli plik należy do użytkownika, *administrator* jeśli plik należy do użytkownika o nazwie *root* i *inni* w pozostałych przypadkach.

3. Napisz skrypt wyświetlający w odwrotnej kolejności argumenty jego wywołania, np.

   ```
   # skrypt a b c d
   d c b a
   ```

4. Napisz skrypt sortujący zawartości wszystkich plików o rozszerzeniach wskazanych pierwszym argumentem i znajdujących się w podkatalogach katalogów wskazanych pozostałymi argumentami.

5. Napisz skrypt zapisujący do pliku o nazwie `plik.usr` pełną informację o użytkownikach którzy mają odblokowane przyjmowanie komunikatów na terminal.

6. Napisz skrypt sprawdzający czy użytkownicy o identyfikatorach podanych jako parametry wejściowe są zalogowani w systemie więcej niż raz. Dla każdego takiego użytkownika należy wyświetlić jego identyfikator i listę terminali na których pracuje. Informacje o kolejnych użytkownikach powinny być oddzielone ciągiem gwiazdek.

8 Zaawansowane programowanie w Bash

8.1 Instrukcja warunkowa

1. Przećwicz działanie instrukcji warunkowej w interpreterze Bash:

```
if [[ -f "$1" && -w "$1" ]]
then
  ...
fi
```

W nawiasach kwadratowych mogą być umieszczone warunki takie, jak w interpreterze Bourne'a oraz kilka dodatków:

- `&&` operator AND,
- `||` operator OR,
- `==` równość łańcuchów tekstowych (to samo co `=`),
- `!=` nierówność łańcuchów tekstowych,
- `()` grupowanie warunków – zapis nie wymaga znaków `\`.

Argumentem operatorów `==` i `!=` mogą być nazwy uogólnione (wzorce), np.:

```
if [[ "$PLIK" == *.txt ]]
```

2. W instrukcji warunkowej może być zastosowany operator `=~` dopasowujący

tekst do wyrażenia regularnego, np.:

```
if [[ "$KOD" =~ ^[0-9]{2}-[0-9]{3}$ ]]
```

Obsługiwane są wyrażenia regularne zgodnie z opisem na stronie pomocy systemowej regex(7).

8.2 Pętle

1. Pętla `for` iterująca po wartościach numerycznych:

```
for x in {1..10}
for x in {10..100..5}
for x in {010..100..5}
for x in {A..Z}
```

2. Pętla `for` może przybrać formę znaną z języka C:

```
for ((x=1; x<=10; x++))
do
  echo $x
done
```

8.3 Odwołania do zmiennych

1. Przećwicz złożone odwołania do zmiennych:

`${X:-wyrażenie}`	wartość domyślna
`${#X}`	długość napisu
`${!X}`	wartość zmiennej o nazwie zapisanej w zmiennej `X` (ang. *variable indirection*)
`${X:i:l}`	fragment tekstu od pozycji `i` o długości `l`
`${X%wzorzec}`	usuwanie najkrótszej końcówki opisanej wzorcem
`${X%%wzorzec}`	usuwanie najdłuższej końcówki
`${X#wzorzec}`	usuwanie najkrótszego prefiksu
`${X##wzorzec}`	usuwanie najdłuższego prefiksu
`${X/wzorzec/tekst}`	zamiana tekstu opisanego *wzorcem* na inny *tekst*
`${X//wzorzec/tekst}`	zamiana wszystkich wystąpień *wzorca* na *tekst*
`${X^^wzorzec}`	zamiana na wielkie litery
`${X,,wzorzec}`	zamiana na małe litery

8.4 Przetwarzanie danych

1. Przetwarzanie wyników poleceń:

```
N=0
while read LINIA
do
  echo "$LINIA"
  ((N=N+1))
done <<< "$(ls -l | tr a-z A-Z)"
echo $N
```

2. Dane wejściowe wewnątrz skryptu:

```
cat <<KONIEC | tr a-z A-Z
Ala
ma
kota
KONIEC
```

8.5 Prezentacja

1. Kolorowe napisy:

```
# printf "Jaki to \033[31mkolor\033[m?\n"
# echo -e "Jaki to \033[31mkolor\033[m?"
```

   Listę dostępnych kodów sterujących ECMA-48 można znaleźć na stronie pomocy systemowej console_codes(4).

2. Napis skrypt, który będzie raportował postęp swojej pracy wyświetlając cały czas w tym samym miejscu (w jeden linii) wartość procentową od 0 do 100.

8.6 Obsługa sygnałów

1. Zablokuj możliwość przerywania skryptu kombinacją **Ctrl-C**:

```
function obsluga() {
  echo "Tak się nie da"
}

trap obsluga 2
...
```

8.7 Testowanie skryptów

1. Wykonaj skrypt w trybie diagnostycznym:

   ```
   # bash -x skrypt.sh
   ```

2. Stałe wykonywanie w trybie diagnostycznym:

   ```
   #!/bin/bash -x
   ```

8.8 Tablice

8.8.1 Tablice zwykłe

1. Tworzenie tablic:

   ```
   TAB=()
   TAB=(a b c)
   TAB=( $(ls) )
   ```

2. Definiowanie elementów:

   ```
   TAB[0]=ab
   TAB[2]=cd
   TAB+=(ef)       // dodawanie elementów do tablicy
   unset TAB[2]    // usuwanie elementu z tablicy
   ```

3. Pobieranie danych z tablicy:

   ```
   ${TAB[2]}       // element na pozycji 2
   ${TAB[@]}       // wszystkie elementy tablicy
   ${#TAB[@]}      // liczba elementów w tablicy
   ${!TAB[@]}      // lista indeksów z tablicy
   ${TAB[@]:s:n}   // n elementów od pozycji s
   ```

4. Iterowanie po elementach tablicy:

   ```
   for x in "${TAB[@]}"
   do
     echo $x
   done
   ```

   Z wykorzystaniem numerów indeksów:

   ```
   for x in ${!TAB[@]}
   do
     echo ${TAB[x]}
   done
   ```

5. Wczytywanie elementów tablicy z pliku:

   ```
   readarray -t TAB < plik
   ```

6. Sortowanie tablicy:

```
TAB=(5 2 10 7 11 9)
IFS=$'\n'
echo "${TAB[*]}"
sort -n <<< "${TAB[*]}"
TAB=($(sort -n <<< "${TAB[*]}"))
```

8.8.2 Tablice asocjacyjne

1. Tworzenie tablic asocjacyjnych:

```
declare -A TAB
TAB=([key1]=10 [key2]=20)
TAB[key3]=30
unset TAB[key2]
```

2. Iteracja po tablicy:

```
for k in "${!TAB[@]}"
do
  echo "$k  -->  ${TAB[$k]}"
done
```

8.9 Inne mechanizmy

8.9.1 Rekurencja

1. Wywoływanie skryptu przez siebie samego z wykorzystaniem zmiennej `$0`.
2. Wywoływanie skryptu z wszystkimi argumentami:

```
$0 "$@"
```

8.9.2 Liczby losowe

1. Dostęp do zmiennej `RANDOM`:

```
# echo $(( RANDOM % 100 ))
```

8.9.3 Rozszerzone wzorce plików

1. Konfiguracja:

```
# shopt -s extglob
```

2. Odwołania:

```
# ls !(*.txt|*.doc)
```

9 Komunikacja, środowisko graficzne, system plików

9.1 Komunikacja pomiędzy użytkownikami

1. Zaloguj się do serwera unixlab:

```
# ssh unixlab
The authenticity of host 'unixlab (150.254.30.38)' can't be established.
RSA key fingerprint is 58:ff:f1:fa:1d:4a:fe:87:3c:11:e8:c1:eb:5b:69:73.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'unixlab,150.254.30.38' (RSA) to the list of known hosts.
Password: ********
Last login: Tue Feb 17 14:55:37 2009 from labx.cs.put.poznan.pl
Have a lot of fun...
inf12345@unixlab:~> ls
```

2. Skopiuj pliki z lokalnego komputera do systemu zdalnego:

```
# scp a.txt voytek@unixlab:/tmp
```

3. Sprawdź listę użytkowników zalogowanych w systemie:

```
# who
# w
# finger
```

4. Wyślij wiadomość do innego użytkownika:

```
# write inf12345
Cześć!
^D
```

5. Wyślij wiadomość do wszystkich:

```
# wall
Proszę do mnie nie pisać!
^D
```

6. Blokowanie wiadomości:

```
# mesg
is y
# mesg n
```

7. Przeprowadź rozmowę z innym użytkownikiem:

```
# talk inf12345@laboratorium1
```

Druga strona po odebraniu zaproszenia musi wykonać analogiczne zlecenie w odniesieniu do osoby zapraszającej. Rozmowę kończymy kombinacją **Ctrl-c**.

8. Wyślij list do koleżanki/kolegi:

```
# mail inf12345
Subject: To tylko test
Jak się masz?
.
```

a następnie odczytaj swoje listy:

```
# mail
Heirloom mailx version 12.2 01/07/07.  Type ? for help.
"/var/spool/mail/inf12345": 1 message 1 new
>N  1 inf54321@laboratorium  Fri Jun  5 12:49   18/662   To tylko test
? 1
Message  1:
From inf54321@laboratorium  Fri Jun  5 12:49:36 2009
Date: Fri, 05 Jun 2009 12:49:35 +0200
To: inf12345@laboratorium
Subject: test1
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

asdf
```

```
? q
```

Powrót do wcześniej odczytanej poczty:

```
# mail -f mbox
Heirloom mailx version 12.2 01/07/07.  Type ? for help.
"/home/inf12345/mbox": 1 message
>O  1 inf12345@laboratorium  Fri Jun  5 12:49   19/673   To tylko test
? d
? q
```

9. Wyślij list wsadowo list:

```
# ls -l | mail -s "Zawartość katalogu" voytek@example.com
```

oraz list z załącznikiem:

```
# mail -a rys.jpg voytek@example.com
```

9.2 Środowisko graficzne

1. Zaloguj się do systemu korzystając z różnych nakładek na środowisko X Window: Xfce, GNOME, KDE, IceWM i inne.
2. Przećwicz mechanizm kopiowania tekstów w środowisku graficznym: zaznacz tekst w jednym oknie i wklej go w innym używając środkowego przycisku myszy.

   Przećwicz również inne klawisze skrótów służące do kopiowania: **Ctrl-Shift-c** i **Ctrl-Shift-v** oraz **Shift-Ins**.
3. Zawartość schowka można definiować komendą `xclip`, np.:

```
# echo "Ala ma kota" | xclip
# xclip -o
Ala ma kota
# echo "Ola ma psa" | xclip -selection clipboard
# xclip -o -selection clipboard
Ola ma psa
```

4. Uruchom aplikację terminala wyświetlając okno na innym komputerze. W tym celu najpierw zezwól innym na wyświetlanie programów graficznych na swoim komputerze:

```
# xhost +
```

Wyświetlenie okna programu na innym komputerze wymaga wskazania serwera X Window:

```
# xterm -display lab1:0.0
```

Można również skorzystać ze zmiennej środowiskowej DISPLAY:

```
# export DISPLAY=lab1:0
# xterm
```

5. Przećwicz połączenie zdalne z przekierowywaniem połączeń środowiska graficznego:

```
# ssh -X unixlab
unixlab# xload
```

W przypadku aplikacji PuTTY należy przed połączeniem włączyć opcję *Connection ▷ SSH ▷ X11*: *Enable X11 forwarding*.

W systemach Windows funkcję serwera środowiska graficznego X Window dostarcza np. projekt VcXsrv.

9.3 System plików

1. Sprawdź ilość zajętej przestrzeni dyskowej przez pliki na swoim koncie:

```
# du
# du -k
# du -sk
# du -sh
# du -sk * | sort -n
# du -sh * | sort -h
```

2. Przetestuj pełnoekranową, interaktywną wersję programu `du`:

```
# ncdu DIR
```

3. Sprawdź wykorzystanie swojego limitu dyskowego:

```
# quota -s
Disk quotas for user inf12345 (uid 12345):
  Filesystem   space   quota   limit   grace   files   quota   limit   grace
       /home    313M   1075M   1000M            7544   36666   34133
```

Sprawdź reakcję na przekroczenie limitu tworząc bardzo duży plik:

```
# dd if=/dev/zero of=out
# quota
...
# rm out
```

4. Sprawdź ilość wolnej przestrzeni dyskowej:

```
# df
# df -h
```

9.3.1 Archiwizacja danych

1. Stwórz archiwum tar:

   ```
   # tar cvf dane.tar a.txt dir1/
   ```

   Wyświetl zawartość archiwum:

   ```
   # tar tvf dane.tar
   ```

   Rozpakuj archiwum w podkatalogu:

   ```
   # tar xvf ../dane.tar
   ```

   Rozpakuj częściowo archiwum:

   ```
   # tar xvf ../dane.tar dir1
   ```

   Rozpakuj do innego katalogu:

   ```
   # tar xvf dane.tar -C dir2/
   ```

2. Skompresuj archiwum:

   ```
   # ls -l dane.tar
   # gzip dane.tar
   # ls -l dane.tar.gz
   ```

   Rozkompresuj archiwum:

   ```
   # gunzip dane.tar.gz
   ```

3. Przygotuj archiwum skompresowane:

   ```
   # tar cvfz dane.tar.gz dir1/
   ```

   Rozpakuj archiwum skompresowane:

   ```
   # tar xvfz dane.tar.gz
   ```

4. Porównaj wydajność kompresji programów `gzip`, `bzip2` i `xz`:

   ```
   # tar cvfj dane.tar.bz2 dir1
   # tar cvfJ dane.tar.xz dir1
   ```

5. Przygotuj archiwum ZIP:

   ```
   # zip -r dane.zip dir1/
   ```

   Wyświetl zawartość:

   ```
   # unzip -l dane.zip
   ```

   Rozpakuj archiwum:

   ```
   # unzip dane.zip
   ```

6. Dokonaj podziału dużego pliku na mniejsze:

   ```
   # split -b 1M dane.tar.gz dane.tar.gz.
   ```

Połącz części archiwum.

10 Przetwarzanie tekstów

10.1 Edytor strumieniowy sed

1. Przećwicz mechanizmy selekcji linii:

```
# sed -n "1p" a.txt
# sed -n "3,5p" a.txt
# sed -n "3~2p" a.txt
# sed -n '3,$p' a.txt
# sed "1,10d" a.txt
# sed "1,2!d" a.txt
```

   Przetwarzanie linii wskazywanych wyrażeniami regularnymi:

```
# sed "/^a/d" a.txt
# sed "/^a/,/^b/d" a.txt
```

2. Dopisywanie linii tekstu:

```
# sed '$aHello' a.txt
# sed "1iHello' a.txt
# sed '$rb.txt' a.txt
```

3. Zamiana tekstu:

```
# sed "1,3cHello" a.txt
# sed "s/Ala/Ola/" a.txt          // dodatkowe flagi: 1,2,...,g,p
# sed "y/ąęćłńóśźż/aeclnoszz/" a.txt
```

4. Wykonywanie wielu komend jednocześnie:

```
# sed -n "1,3p;5p" a.txt
# sed -n -e 1,3p -e 5p a.txt
```

5. Przeprowadź bezpośrednią modyfikację pliku:

```
# sed -i '$d' a.txt
```

6. Usuń wszystkie komentarze ze skryptu:
   a) oprócz pierwszej linii,
   b) jeżeli po usunięciu komentarza linia zawiera tylko białe spacje, to usuń ją całkowicie,
7. Zamień liczby heksadecymalne zapisane w postaci `$XXXX` (dowolna liczba cyfr), na postać `0xXXXX`. Wypisz z dokumentu tylko te liczby.
8. Wypisz z dokumentu HTML, w oddzielnych liniach, adresy stron, do których są odwołania. Zmodyfikuj skrypt, aby wypisywał również tytuły tych stron w postaci:

```
Tytuł [http://www.example.com]
```

10.2 xargs

1. Porównaj efektywność wykonania następujących zleceń:

```
# find /tmp -type f -name "*.tmp" -exec rm {} \;
# find /tmp -type f -name "*.tmp" | xargs rm
```

2. Tryb diagnostyczny:

```
# find /tmp -type f -name "*.tmp" | xargs -t
```

3. Obsługa plików ze spacjami w nazwach:

```
# find /tmp -type f -name "*.tmp" -print0 | xargs -0 ls -ltr
```

4. Wykonywanie kilku poleceń jednocześnie:

```
# find . -name "*.txt" | xargs -I % sh -c 'cp % %.bak; echo X >> %'
```

   Uzupełni powyższe zlecenie tak, aby mogło poprawnie obsługiwać pliki ze spacjami w nazwach.
5. Stwórz archiwum wszystkich plików TXT w katalogu domowym i podkatalogach.
6. Oblicz sumę linii we wszystkich plikach TXT w katalogu domowym i podkatalogach.

10.3 TEX / LATEX / reStructuredText

Zobacz zadania na stronie http://www.cs.put.poznan.pl/csobaniec/edu/sop/latex.html.

Indeks