Pliki przechowujace konta

Ze względu na wieloużytkownikowość i wielodostęp, systemy UNIX posiadają bazę danych informacji o użytkownikach. W bazie tej zawarte są podstawowe dane o każdym z użytkowników, takie jak nazwa użytkownika, numer, grupa, do której należy, krótki opis, katalog domowy oraz powłoka, której używa. Baza użytkowników znajduje się w pliku***/etc/passwd***, natomiast zaszyfrowane hasła użytkowników i informacje uzupełniające w ***/etc/shadow***. Wpisy do pliku ***/etc/passwd*** mają następującą strukturę:

```
Username:Password:UID:GID:Gecos:HomeDirectory:Shell
```

przykład:

```
root:x:0:0:root:/root:/bin/bash
```

username
: nazwa użytkownika w systemie.

haslo
: hasło kodowane DES, w systemach z shadow zazwyczaj x.

uid
: numer użytkownika.

gid
: numer głównej grupy.

gecos
: pole komentarza, zwane też polem GECOS zawiera dodatkowe informacje o użytkowniku, składa się z czterech pól oddzielonych przecinkami: imię z nazwiskiem, adres/położenie biura, telefon w pracy, telefon domowy. Pole GECOS nie jest obowiązkowe.

home
: katalog domowy użytkownika.

shell
: domyślny interpreter poleceń.

Wpisy w pliku */etc/shadow* mają następującą strukturę:

```
Username:haslo:t1:t2:t3:t4:t5:t6:zarezerwowane
```

przykład:

```
smithj:Ep6mckrOLChF.:10063:0:99999:7:::
```

username

nazwa użytkownika.

haslo

hasło, zakodowane w DES lub MD5; patrz crypt(3).

t1

data ostatniej zmiany hasła, w dniach od 01.01.1970 GMT.

t2

dni od zmiany hasła, po których można zmienić je ponownie.

t3

dni od zmiany hasła, po których hasło musi być zmienione.

t4

dni przed wygaśnięciem hasła, kiedy użytkownik jest ostrzegany.

t5

dni po wygaśnięciu hasła, po których konto jest wyłączane.

t6

data włączenia konta, w dniach od 01.01.1970 GMT.

zarezerwowane

pozostawione dla przyszłego użytku.

Aby łatwiej zarządzać prawami dostępu do zasobów, użytkownicy należą do jednej lub więcej grup. W systemach UNIX przyjęto, że każdy użytkownik ma swój numer (UID), który jest jego identyfikatorem. Prawa dostępu do zasobów określane są względem tego numeru, a nie nazwy użytkownika. Nazwa jest jakby "aliasem" do numeru - można ją w każdej chwili zmienić bez żadnych konsekwencji - wszystkie prawa dostępu zostają zachowane, (bo przyznawane są według numerów, a nie nazw). Dzięki ***/etc/passwd*** możliwe jest "tłumaczenie" nazw na UIDy i odwrotnie.

Możliwe też jest stworzenie dwóch lub więcej kont z tymi samymi numerami UID pod różnymi nazwami. Wówczas użytkownicy, korzystający z tego samego UIDu mają identyczne prawa i ograniczenia w systemie, pomimo, że mogą posiadać różne hasła, katalogi domowe czy powłoki. Grupy użytkowników funkcjonują na podobnych zasadach. Każda grupa posiada swój numer (GID) i to właśnie ten numer wykorzystywany jest przy określaniu praw dostępu.

Każdy z użytkowników musi należeć przynajmniej do jednej grupy. Główna grupa, do której należy użytkownik zapisana jest w ***/etc/passwd***. Pozostałe w ***/etc/group***.

```
nazwa_grupy:hasło:GID:lista_użytkowników
```

przykład:

```
gdftp::502:dave,nick,pete,ben,rwm
```

Aby przekonać się, w jakich grupach znajduje się użytkownik i jaki jest jego UID należy

wydać komendę:

```
id [UŻYTKOWNIK]
```

lub:

```
groups [UŻYTKOWNIK]
```

Przykład:

```
user@ttyp0[linux]$ id
uid=1000(user) gid=1000(user)
grupy=1000(user),20(dialout),21(fax),22(voice),
24(cdrom),25(floppy),26(tape),27(sudo),29(audio),30(dip),44(video),60(ga
mes),100(users),106(usb)
```

Z powyższego zapisu wynika, że w systemie jest użytkownik **user** i ma nadany numer UID=1000, jego główną grupą jest **user** (GID=1000), a dodatkowo należy do grup "users" GID=100, "dialout" GID=20 i innych.

Inne polecenia pokrewne: users, who, whoami

Własność zasobów

Użytkownicy i grupy ułatwiają zarządzanie dostępem do zasobów.

Dla przykładu:

```
user@ttyp0[tmp]$ ls -l kopia
-rw-r--r-- 1 user user 0 2004-04-19 21:43 kopia
user@ttyp0[tmp]$
```

Właścicielem pliku kopia jest użytkownik **user**. Prawa dla tego pliku, określane są dla właściciela, grupy i wszystkich pozostałych użytkowników. Aby optymalnie zarządzać tymi prawami, właściciel pliku może zmienić grupę względem której określane są prawa. Użytkownik **user**, zmieniając grupę np. na **users** spowoduje, że tylko użytkownicy tej grupy będą mogli przeglądać zawartość tego pliku. W tym celu posłużyć się należy poleceniem **chgrp** (**ch**ange **gr**ou**p**)o następującej składni:

```
chgrp GRUPA PLIK(I)
```

gdzie:

GRUPA

grupa, na którą należy zmienić,

PLIK(I)

lista plików do zmiany.

Przykład:

```
user @ttyp0[tmp]$ ls -l kopia
-rw-r--r-- 1 user user 0 2004-04-19 21:43 kopia
user @ttyp0[tmp]$ chgrp users kopia
user @ttyp0[tmp]$ ls -l kopia
-rw-r--r-- 1 user users 0 2004-04-19 21:43 kopia
```

Grupę może zmienić tylko właściciel pliku i to tylko i wyłącznie na grupę w której należy. Ograniczenia te nie dotyczą oczywiście użytkownika **root**. Dodatkowo administrator może zmienić właściciela pliku poleceniem **chown** (**ch**ange **own**er) o następującej składni:

```
chown [WLASCICIEL][:GRUPA] PLIK(I)
```

gdzie:

WLASCICIEL

nowy właściciel pliku,

GRUPA

nowa grupa,

PLIK(I)

lista plików poddawanych operacji zamiany.

Inne polecenia związane z grupami: chmod, newgrp, groupadd, groupdel, groupmod

Zarządzanie kontami

By użytkownik mógł pracować w systemie, Unix powinien mieć w nim założone konto.

Konto w tym przypadku rozumiemy, jako wszystkie zasoby, pliki i informacje o tym użytkowniku. Każde konto charakteryzują dwie podstawowe cechy - **hasło** i **login**. Dzięki loginowi komputer wie, z którym użytkownikiem aktualnie pracuje, a hasło służy do zabezpieczenia zasobów tego użytkownika. Aby jednak było to możliwe, w systemie jest wbudowane konto **root** posiadające uprawnienia do wykonywania czynności administracyjnych. Uwaga: cechą wyróżniającą konto administracyjne jest identyfikator UID równy 0. Konto administracyjne może mieć zmienioną nazwę, a nawet może istnieć kilka kont tego typu.

Konta wykorzystywane są również przez programy, które na każdym z tych kont mogą posiadać własne pliki konfiguracyjne, dostosowujące program do potrzeb danego użytkownika.

Polecenie służące do zakładania konta ma następującą postać:

```
useradd [-c komentarz] [-d katalog_domowy] [-e data_ważności] [-f
```

```
dni_nieaktywności] [-g grupa_początkowa] [-G grupa [,...]] [-m [-k
katalog_wzorców]] [-o] [-p hasło]  [-s powłoka] [-u uid] login
```

Jak widać polecenie zawiera pokaźny zbiór opcji, w krańcowym przypadku wystarczy jednak ograniczyć się do tego co niezbędne: useradd *login.* Inne spotykane polecenia związane z grupami: userdel, usermod, chfn,

Często zachodzi potrzeba, aby wykonać pewne czynności administracyjne, do których zwykły użytkownik nie ma prawa. W tym celu istnieje możliwość zmiany tożsamości za pomocą polecenia **su** zarówno w odniesieniu do użytkownika **root** jak i pozostałych użytkowników.

Część z tych zasobów użytkownik może zmienić podczas pracy w systemie. Przede wszystkim użytkownik może zmienić powłokę, której używa poprzez polecenie **chsh** (**ch**ange**sh**ell). Przykładowa zmiana powłoki wygląda następująco:

```
user @ttyp0[linux]$ chsh
Password:
Zmieniam powłokę logowania dla user
Wpisz nową wartość lub wciśnij ENTER by przyjąć wartość domyślną
       Powłoka logowania [/bin/bash]: /bin/sash
```

Innym poleceniem umożliwiającym takie zmiany jest **chfn** (**ch**ange **f**inger i**n**formation), które pozwala na dokonanie zmiany pełnej nazwy użytkownika, ewentualnie na podanie miejsca pracy i telefonów.

Wielodostępna natura systemów Unix/Linux oznacza, że z usług systemu może korzystać większa liczba użytkowników jednocześnie. Aby dowiedzieć się, kto jest aktualnie zalogowany do systemu należy wydać komendę **users**. Oto przykładowy wynik działania tej komendy:

```
user @ttyp0[linux]$ users
user root root
user @ttyp0[Linux]$
```

Wynika z tego, że zalogowanych jest 2 użytkowników: **user** i **root**. Innym poleceniem sprawdzającym, kto jest aktualnie zalogowany jest: **w** oraz **who**. Działanie tej ostatniej wygląda następująco:

```
barakuda@angband ~ $ who
barakuda tty1         2006-10-31 23:34
test     tty3        2006-11-04 17:06
root     tty2        2006-11-03 15:46
barakuda@angband ~ $
```

W wyniku działania tej komendy stwierdzić można, z jakiej konsoli logowali się użytkownicy oraz od kiedy są zalogowani. Konsole oznaczone jako **ttyX** (X – numer

konsoli) są to wirtualne konsole, natomiast **pts** to pseudoterminale, czyli użytkownicy zalogowali się do serwera poprzez sieć i usługę telnet.

Większą porcję danych o użytkownikach zalogowanych w systemie dostarcza polecenie **w**. Przykładowe działanie wygląda następująco:

```
barakuda@angband ~ $ w
 17:07:58 up 3 days, 17:34,  3 users,  load average: 0.71, 0.59, 0.41
USER     TTY       LOGIN@   IDLE   JCPU    PCPU   WHAT
barakuda tty1      Tue23   25:19m  0.09s   0.00s /bin/sh /usr/bin/startx
test     tty3              17:06m  1:31s   0.00s  0.00s -bash
root     tty2      Fri15   21:46m  0.16s   0.16s -bash
barakuda@angband ~ $
```

Pierwsza linia wyświetlana przez to polecenie zawiera ogólne informacje o systemie - godzinę (17:07:58), jak długo serwer pracuje (17:34 minut), ilość zalogowanych użytkowników (3 users) oraz obciążenie serwera w ciągu ostatniej minuty, w ciągu ostatnich 5 minut i ostatnich 15 minut (load average: 0.71, 0.59, 0.41). Następne linie zawierają tabelaryczne zestawienie danych dotyczących zalogowanych użytkowników. W pierwszej kolumnie znajduje się identyfikator użytkownika. Druga kolumna zawiera nazwę terminala, z którego zalogowany jest dany użytkownik. Trzecią kolumnę stanowi nazwa hosta, z którego użytkownik się zalogował (przydatne, gdy użytkownik korzysta z pseudoterminala). Dalsze kolumny to kolejno: informacja, o której godzinie nastąpiło logowanie, jak długo użytkownik jest bezczynny, ile czasu procesora zajmuje obsługa danego terminala, ile czasu procesora zajmuje obsługa zadania wyświetlanego w ostatniej kolumnie oraz aktualnie wykonywane zadanie przez użytkownika.

Poleceniem, które dostarcza informacji o konkretnym użytkowniku jest **finger**, np. finger alvin

```
barakuda@angband ~ $ finger
Login    Name      Tty     Idle  Login Time   Office    Office Phone
barakuda          *tty1      1d  Oct 31 23:34
root     root     *tty2    21:46  Nov  3 15:46
test              *tty3       1  Nov  4 17:06
barakuda@angband ~ $
```

W odróżnieniu od polecenia **w**, wyświetla dane w nieco innym formacie i dołącza dodatkowo pełną nazwę użytkownika. Dodatkowo **finger** wyświetla informacje o tym, czy jesteśmy w stanie z danym użytkownikiem porozmawiać (dokładniej: użyć polecenia **write** bądź **talk**). Jeśli nie ma takiej możliwości, wówczas w kolumnie **Tty** pojawia się gwiazdka obok nazwy terminala (jak w przypadku użytkownika **root**).

Unix pozwala także sprawdzić nie tylko, kto aktualnie jest, zalogowany ale również, kto pracował w systemie w przeszłości (w określonym przedziale czasu). Do tego służy polecenie:

```
last [-NUMER] [UŻYTKOWNIK]
```

gdzie:

NUMER

ogranicza rozmiar wyniku do podanej ilości linii

UŻYTKOWNIK

podanie użytkownika oznacza wyświetlenie statystyk tylko dla podanego użytkownika. Zamiast użytkownika można wpisać nazwę terminala bądź słowo **reboot**, aby uzyskać informację o czasie pracy systemu.

Przykład:

aby zobaczyć 10 ostatnich logowań i restartów systemu:

```
barakuda@angband ~ $ last -n 10
root     tty2                      Sat Nov  4 17:11   still logged in
root     tty2                      Sat Nov  4 17:11 - 17:11  (00:00)
barakuda tty2                       Sat Nov  4 17:11 - 17:11  (00:00)
barakuda tty2                       Sat Nov  4 17:11 - 17:11  (00:00)
test     tty2                      Sat Nov  4 17:11 - 17:11  (00:00)
test     tty2                      Sat Nov  4 17:11 - 17:11  (00:00)
test     tty3                      Sat Nov  4 17:06   still logged in
test     tty3                      Sat Nov  4 17:06 - 17:06  (00:00)
barakuda pts/2        :0.0           Sat Nov  4 16:47 - 16:50  (00:03)
barakuda pts/2        :0.0           Sat Nov  4 16:34 - 16:37  (00:03)

wtmp begins Wed Nov  1 15:29:23 2006
barakuda@angband ~ $
```

Hasła

Dostęp do konta każdego użytkownika chroniony jest zwykle jedynie hasłem. Zwykły użytkownik może zmienić wyłącznie hasło własnego konta, użytkownik **root** może zmieniać hasła dowolnych kont, natomiast administrator grupy może zmienić hasło tej grupy. Należy pamiętać o podstawowych zasadach związanych z bezpieczeństwem haseł, tzn. hasło powinno mieć minimum kilkanaście znaków zawierających małe i duże litery, cyfry i znaki przestankowe. Nie powinno być „słownikowe” oraz powinno być łatwe do zapamiętania.

Zmiana konta dokonywana jest za pomocą polecenia: **passwd**. Po czym użytkownik zostaje poproszony o podanie starego hasła i dwukrotne wprowadzenie nowego.

Dobrym pomysłem przy tworzeniu haseł wydaje się tworzenie skrótów z dłuższych fraz. np. "k2Bwh,pBt" utworzone jest z łatwego do zapamiętania zdania "kupiłem 2 bułki w hipermarkecie, ponieważ były tanie".

Zdalny dostęp

Wszystkie systemy uniksowe pozwalają na obsługę komputera poprzez sieć. Najczęściej

spotyka się trzy metody dostępu do zdalnej konsoli:

- rsh
- telnet
- ssh

Jednakże ze względów bezpieczeństwa administratorzy preferują program ssh i często blokują dostęp pozostałymi metodami.

Aby zalogować się poprzez protokół ssh należy wydać polecenie:

```
ssh -l użytkownik nazwa_komputera
```

lub

```
ssh użytkownik@nazwa_komputera
```

gdzie:

użytkownik
to nazwa użytkownika na którego konto chcemy się zalogować

nazwa_komputera
to nazwa domenowa hosta lub jego adres IP

Po wydaniu polecenie zostaniemy zapytani o hasło (lub nie, jeśli tak został skonfigurowany serwer). W chwili połączenia uzyskujemy dostęp do konsoli zdalnego hosta z pełnymi uprawnieniami danego użytkownika. Aby zerwać połączenie należy wydać polecenie:

```
exit
```

Zadania

- Utwórz użytkowników test1, test2 oraz test3
- Ustal hasła dla użytkowników test1 i test2 (identyczne jak nazwy)
- Zaloguj się na użytkownika test1
- Będąc zalogowanym na użytkownika test1 ustaw hasło użytkownikowi test3. Jest to możliwe? Skorzystaj z narzędzia nadającego Ci prawa administratora.
- Utwórz grupy grupa1 i grupa2
- Przypisz użytkowników test1 i test2 do grupy grupa1.
- Przypisz użytkowników test2 i test3 do grupy grupa2. Czy użytkownik test2 może należeć do dwóch grup? Do jakiej grupy należą pliki zakładane przez użytkownika test2?
- Ustaw grupę domyślną użytkownika test2 na grupa2. Do jakiej grupy należą pliki tworzone przez tego użytkownika?

- Zmień powlokę użytkownika test2 na /bin/sh. Umiał byś to zrobić edytując pliki konfiguracyjne?
- Zmień powlokę użytkownika test3 na /bin/passwd. Jest to możliwe? Jeśli zajdzie potrzeba zmodyfikuj odpowiednie pliki. W jakich okolicznościach takie ustawienie może mieć sens?
- Przekształć konto test1 w „zapasowe" konto administracyjne.
- Zaloguj się zdalnie do komputera osoby siędzącej obok.

Prawa dostępu do plików i katalogów

Prawa dostępu do plików i katalogów są jednymi z najważniejszych mechanizmów bezpieczeństwa systemu. Uniemożliwiają one innym użytkownikom przeglądanie naszych zasobów. Prawa dostępu podzielone są na trzy sekcje:

- właściciel pliku lub katalogu
- grupa związana z plikiem lub katalogiem
- wszyscy inni użytkownicy systemu

Prawa dostępu można odczytać wydając polecenie *ls -l*:

```
angband ~ # ls -l
razem 5016
-rw-r--r--   1 root root   31624 kwi  7  2006 agpgart.ko
drwxr-xr-x   3 root root    4096 lut 11  2005 app-portage
drwxr-xr-x   2 root root    4096 gru 11  2005 arsene
-rw-r--r--   1 root root       0 maj 26 20:39 CHG
-rw-r--r--   1 root root       0 maj 26 20:39 CHGCAR
```

znaki minus (-) i litery występujące na początku każdej linii reprezentują typ pliku i prawa dostępu.

Prawa dostępu do pliku

Każdy plik ma ściśle określone prawa dostępu stwierdzające, czy określony użytkownik jest uprawniony do odczytania lub zapisania pliku bądź do jego wykonania. Każdy użytkownik może mieć dowolną kombinację tych praw. Są one całkowicie niezależne i posiadanie jakiegokolwiek z nich nie jest warunkiem posiadania innego. W przypadku pliku prawa są interpretowane w następujący sposób:

- r – prawo czytania umożliwia oglądanie zawartości pliku, oznacza jednocześnie prawo do kopiowania,
- w – prawo pisania oznacza zezwolenie na modyfikację zawartości pliku,
- x – prawo do uruchomienia pliku wykonywalnego.

Prawa dostępu do katalogu

Te same kategorie praw - czytania, pisania i wykonywania odnoszą się do katalogów:

- r – prawo czytania umożliwia przeszukiwanie zawartości katalogu, jest interpretowane jako prawo wypisywania zawartości (komenda *ls*),
- w – prawo pisania daje możliwość modyfikowania zawartości katalogów umożliwia dodawanie nowych oraz usuwanie dotychczasowych plików z katalogu,
- x – prawo wykonywania w stosunku do katalogu pozwala na dostęp do plików zapisanych w nim oraz na wejście do danego katalogu -uczynienie go katalogiem bieżącym (polecenie:*cd katalog*).

Wydając polecenie ls –l otrzymujemy na ekranie:

```
-rwxr--r-- 1 root root 497 May 23 11:57 index.html
drwxr-xr-x 5 root root 512 Sep 17 12:47 www
```

znaki minus (-) i litery występujące na początku każdej linii reprezentują typ pliku i prawa dostępu. Przyjmują one postać:

```
krwxrwxrwx
```

Wyjaśnienie tego zapisu jest następujące:

- *k* – jest to identyfikator typu, gdzie:
  - - zwykły plik
  - b specjalny plik blokowy
  - c specjalny plik znakowy
  - d katalog
  - l link symboliczny
  - p potok
  - s gniazdo
- prawa dostępu:
  - Pierwsza trójka **rwx** – oznacza uprawnienia dla właściciela (u – user)
  - Druga trójka **rwx** – oznacza uprawnienia dla grupy (g – group)
  - Trzecia trójka **rwx** – oznacza uprawnienia dla pozostałych użytkowników (o – others)

Prawo	**Plik**	**Katalog**
r	czytania zawartości	przeszukania zawartości
w	zmiany zawartości	zmiany zawartości

x	Wykonywanie	przejścia do tego katalogu

Przypisywanie uprawnień

Do zmiany uprawnień użytkowników w stosunku do katalogu lub pliku służy polecenie *chmod*. Wymaga ono określenia, czyje uprawnienia należy zmienić, na jakie oraz jakiego pliku lub katalogu ta zmiana będzie dotyczyć. Prawa dostępu mogą być podane na dwa sposoby:

- przy pomocy systemu kodów numerycznych (w formacie ósemkowym): 4 2 1
- przy pomocy systemu kodów znakowych: r w x

Oto porównanie tych systemów (w nawiasach przy zapisie numerycznym podano zapis binarny uprawnień):

Prawa dostępu	**Zapis numeryczny**	**Zapis znakowy**
Tylko do czytania	4 (100)	r--
Tylko do pisania	2 (010)	-w-
Tylko do wykonywania	1 (001)	--x
Do czytania i pisania	6 (110)	rw-
Do czytania i wykonywania	5 (101)	r-x
Czytania, pisania i wykonywania	7 (111)	rwx

Kody numeryczne praw

Jeśli prawa dostępu do danego katalogu lub pliku będziemy podawać numerycznie, wówczas składnia polecenia *chmod* ma postać:

```
chmod numeryczny_kod nazwa_zasobu
```

Przykład:

```
chmod 644 plik.txt
```

zapis ten oznacza:

- prawo dostępu właściciela **6 = 4 + 2 + 0 czyli rw**
- prawo dostępu grupy **4 = 4 + 0 + 0 czyli r**
- prawo dostępu pozostałych użytkowników **4 = 4 + 0 + 0 czyli r**

```
chmod 701 katalog
```

zapis ten oznacza:

- prawo dostępu właściciela **7 = 4 + 2 + 1 czyli rwx**
- prawo dostępu grupy **0 = 0 + 0 + 0 czyli brak uprawnień**
- prawo dostępu pozostałych użytkowników **1 = 0 + 0 + 1 czyli x**

```
chmod 755 plik2.txt
```

zapis ten oznacza:

- prawo dostępu właściciela **7 = 4 + 2 + 1 czyli rwx**
- prawo dostępu grupy **5 = 4 + 0 + 1 czyli r-x**
- prawo dostępu pozostałych użytkowników **5 = 4 + 0 + 1 czyli r-x**

kody symboliczne praw

Jeśli prawa dostępu do danego katalogu lub pliku będziemy podawać symbolicznie, wówczas składnia polecenia *chmod* ma postać:

```
chmod kto_oper_prawo nazwa_zasobu
```

gdzie:

kto

określa komu nadawane są prawa i może być jednym, bądź kilkoma, spośród symboli:

- a - wszyscy użytkownicy,
- u - właściciel pliku
- g - grupa pliku,
- o - inni użytkownicy.

Pominięcie symbolu kategorii nadaje wszystkim (właścicielowi, grupie, pozostałym) takie same prawa.

oper

jest jednym z następujących symboli:

- **–** oznacza odebranie prawa,
- **+** oznacza dodanie prawa,
- **=** ustala nowe uprawnienia niezależnie od stanu poprzedniego.

prawo

typ praw dostępu, który jest jednym, bądź kilkoma, spośród symboli: r, w, x,

Przykład:

```
chmod u-w plik1.txt
```

zapis ten zabiera właścicielowi prawo pisania do pliku plik1.txt,

```
chmod a=rw plik1.txt
```

zapis ten nadaje wszystkim prawo rw do pliku plik1.txt,

```
chmod u+w, og+r-x plik1.txt
```

zapis ten nadaje właścicielowi prawo pisania do pliku plik1.txt, a członkom grupy oraz pozostałym użytkownikom systemu nadaje prawo czytania i odbiera prawo wykonania pliku plik1.txt,

Porównanie obu zapisów:

Prawo do czytania dla właściciela pliku

```
chmod 400 nazwa_pliku
chmod u+r nazwa_pliku
```

Wszystkie prawa dla właściciela pliku i prawo do czytania dla grupy

```
chmod 740 nazwa_pliku
chmod u+rwx,g+r nazwa_pliku
```

Domyślne prawa dostępu przy tworzeniu plików i katalogów

Domyślne prawa dostępu dla plików i katalogów nadawane są podczas ich tworzenia. Zmianę tych praw uzyskujemy poleceniem *umask*. Jeśli chcielibyśmy, aby tworzone pliki miały domyślne prawa 644, które zezwalają właścicielowi na czytanie i pisanie, a reszcie tylko na czytanie to od wartości 777 należy odjąć 644, a wynik podać jako parametr polecenia *umask*

777-644 = 133

```
umask 133
```

suid i sgid - pożyteczne i niebezpieczne narzędzie

Zadaniem tego potężnego, a zarazem niebezpiecznego narzędzia jest uruchamianie programu (nie skryptu) z prawami właściciela lub grupy przypisanej temu programowi, a nie z prawami użytkownika, który ten program uruchamia.

Zagrożenie z używania tych flag może wynikać z możliwości przejęcia kontroli nad systemem. Jeśli zwykłemu użytkownikowi uda się tak zawiesić program (którego właścicielem jest użytkownik root i który ma ustawioną flagę suid lub sgid), aby dostać się do powłoki to otrzyma on prawa właściciela programu (czyli w tym przypadku użytkownika root) co stanowi ogromne zagrożenie dla systemu.

Dlatego należy z dużym rozsądkiem używać tych flag.

Nadawanie plikom **suid** lub **sgid** wygląda następująco:

suid

```
chmod u+s nazwa_plku
chmod 2*** nazwa_pliku
```

sgid

```
chmod g+s nazwa_pliku
```

```
chmod 4*** nazwa_pliku
```

suid i **sgid**

```
chmod 6*** nazwa_pliku
```

gdzie *** oznacza dowolne prawa dla właściciela, grupy i innych użytkowników.

Flaga **suid** w listingach plików reprezentowana jest przez literkę s w prawach dla właściciela pliku :

```
rws r-x r-x
```

Flaga **sgid** w listingach plików reprezentowana jest przez literkę s w prawach dla grupy:

```
rwx r-s r-x
```

sticky bit

Dla pliku ustawienie sticky bitu oznacza, że program, który on przechowuje będzie po jego zakończeniu nadal przechowywany w pamięci komputera. Dla katalogów sticky bit oznacza, że tylko właściciel może go usunąć mimo ustawienia praw na przykład na 777.

Ustawienie sticky bitu dla plików wygląda następująco:

```
-rwxr--r-- 1 root root 497 May 23 11:57 index.html
drwxr-xr-x 5 root root 512 Sep 17 12:47 www
krwxrwxrwx
chmod numeryczny_kod nazwa_zasobu
chmod 644 plik.txt
chmod 701 katalog
chmod 755 plik2.txt
chmod kto_oper_prawo nazwa_zasobu
chmod u-w plik1.txt
chmod a=rw plik1.txt
chmod u+w, og+r-x plik1.txt
chmod 400 nazwa_pliku
chmod u+r nazwa_pliku
chmod 740 nazwa_pliku
chmod u+rwx,g+r nazwa_pliku
umask 133
chmod u+s nazwa_plku
chmod 2*** nazwa_pliku
chmod g+s nazwa_pliku
chmod 4*** nazwa_pliku
chmod 6*** nazwa_pliku
rws r-x r-x
rwx r-s r-x
chmod 1*** nazwa_pliku_katalogu

chmod +t nazwa_pliku_katalogu
rwx r-x r-t
chown nowy_właściciel nazwa_pliku(ów)

chgrp nowa_grupa nazwa_pliku(ów)
chown nowy_właściciel:nowa_grupa nazwa_pliku(ów)
```

gdzie *** oznacza dowolne prawa dla właściciela, grupy i innych użytkowników.

Reprezentantem tego bitu w listingach katalogów jest literka **t** w sekcji dotyczącej reszty użytkowników :

```
rwx r-x r-t
```

Zmiana właściciela i grupy pliku

Zmiana właściciela i grupy pliku możliwa jest przy użyciu poleceń: *chown*, *chgrp*. Rzecz się ma następująco:

```
chown nowy_właściciel nazwa_pliku(ów)
```

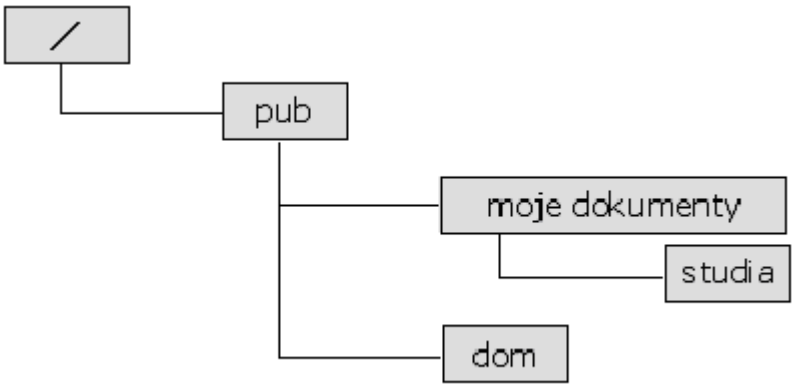
lub w połączeniu

```
chown nowy_właściciel:nowa_grupa nazwa_pliku(ów)
```

Używając opcji *-R* polecenia *chown*, możesz zmienić właściciela wszystkich plików w danym katalogu i wszystkich jego podkatalogach. Wówczas zamiast nazwy pliku podaje się nazwę żądanego katalogu. Opcja ta jest również ważna dla polecenia *chgrp*.

Zadania

- Utwórz następującą strukturę katalogów:



- Rys. 1. Struktura katalogów do "Sesji przy terminalu 2"

- Skopiuj wszystkie polecenia dwuliterowe zaczynające się od litery *d* z katalogu */bin* do katalogu *moje dokumenty*
- Utwórz w katalogu *dom* dwa pliki tekstowe *plik1.txt* oraz *plik2.txt*. Pliki te powinny zawierać przykładowy tekst.
- Jakie uprawienie domyślne mają *pliki1.txt* oraz *plik2.txt*.
- Ustaw dla pliku plik1.txt uprawnienia rw-r--rw-.
- Ustaw dla pliku plik2.txt następujące uprawnienia właściciel: odczyt, zapis, wykonanie, grupa: tylko zapis, pozostali: odczyt, zapis. Jak to zrobić korzystając z zapisu numerycznego?

- Wyświetl uprawnienia plików plik1.txt oraz plik2.txt.
- Wydaj polecenie chmod 574 dla plik1.txt? Jakie zostaną nadane mu prawa?
- Wydaj polecenie chmod u=rw, g=r, o-r dla pliku plik2.txt. Jakie zostaną nadane mu prawa?
- Wyświetl uprawnienia plików plik1.txt oraz plik2.txt. Które polecenie (pkt. 8 i 9) nadpisuje uprawnienia, a które je modyfikuje?
- Utwórz plik mojedane.txt w katalogu studia i umieść w nim swoje dane.
- Załóż plik o nazwie *email.txt*, w katalogu studia i umieść swój adres e-mail
- Ustaw dla katalogu moje dokumenty i wszystkich podkatalogów następujące uprawnienia właściciel: odczyt, zapis, wykonanie, grupa: brak uprawnień, pozostali: odczyt, zapis.
- Sprawdź, czy możesz nie nadawać plikowi żadnych praw?
- Pracując jako zwykły użytkownik utwórz plik moj.txt. Następnie zaloguj się na konto root i przejmij ten plik na własność.
- Kto jest właścicielem plików znajdujących się w katalogu dom?

Procesy

Procesy w systemiach uniksowych

Z systemami unixowymi związane jest pojęcie procesu. W takim ujęciu, proces, rozumiany jest jako wykonywany w systemie program. Każdy proces charakteryzuję się pewnymi atrybutami: przestrzeń adresowa, licznik programu, stanu rejestrów, deskryptory plików, dane procesu, zależności rodzinne, liczniki statystyczne. Wynikiem obecności w systemie procesów jest to, że jądro systemu może nim sterować tak i może go ustawiać w kilku stanach. Zależnie od źródeł literatury, może być 9 lub 5 stanów. Ostatni przypadek to stan:

Pracujący w trybie użytkownika
: proces znajduje się na procesorze i wykonuje swój kod.

Pracujący w trybie jądra
: jądro wykonuje wywołanie systemowe wykonane przez proces.

Uśpiony
: proces czeka na jakieś zdarzenie, na przykład na odczyt danych z dysku lub otrzymanie danych z sieci.

Gotowy do wykonania
: może być uruchomiony w każdej chwili, jednak nie ma jeszcze przydzielonego procesora.

Zombie
: proces zakończył działanie i czeka na odebranie kodu powrotu przez proces macierzysty.

Wszystkie procesy w Unixie powstają jako procesy potomne procesu głównego init o numerze 1, który tworzony przez jądro podczas uruchamiania systemu. Każdy proces może być zarówno procesem potomnym jak i procesem macierzystym innego procesu. |System wykonuje każdy proces przez określony czas następnie pobiera kolejny proces do wykonania. W tym czasie grupa procesów oczekuje na wykonanie. Sprawne działanie zapewnia szeregowanie z wywłaszczaniem oraz system priorytetów i co pozwalający tak ustawić intensywnie używające procesor procesy tła, aby nie blokowały pracy procesom interakcyjnym.

Polecenia związane z procesami

ps

Podstawowym poleceniem do zarządzania procesami przez użytkownika jest: ps. Polecenie **ps** występuje w systemadch uniksowych w kilku wersjach. Różnią się on między sobą sposobem podawania parametrów i nieznacznie zachowaniem. Wersja dostarczana z systemem Ubuntu Linux obsługuje większość opcji podawanych w jednej z trzech konwencji:

1. Opcje w stylu UNIX, które mogą być grupowane i muszą być poprzedzone myślnikiem.
2. Opcje BSD, które mogą być grupowane i nie mogą być użyte z myślnikiem.
3. Długie opcje GNU, które są poprzedzone dwoma myślnikami.

Polecenie:

```
ps [-] [lujsvmaxScewhrnu] [txx] [O[+|-]k1[[+|-]k2...]] [pids]
```

ps uruchomiony bez parametrów wyświetla wszystkie procesy danego użytkownika związane z bieżąca konsolą.

Wybrane opcje w formacie Uniksa:

-?

wyświetla najważniejsze opcje

-A lub -e

wyświetla wszystkie procesy

-a

wyświetla wszystkie procesy posiadające terminal

-l

długi format

-j

format prac: pgid, sid

-H

"forest" (las) - format drzewiasty

Wybrane opcje w formacie BSD:

a

wyświetla wszystkie procesy posiadające terminal

x

wyświetla wszystkie procesy posiadające i nie posiadające terminala, należące do bieżącego uzytkownka

l

długi format

j

format prac: pgid, sid i inne identyfikatory, dla danego użytkownika

f

"forest" (las) - format drzewiasty

e

pokaż środowisko (wszystkie zmienne systemowe) dla każdego polecenia

h

bez nagłówka

Wybrane opcje o identycznym działaniu:

-u lub u

podaje nazwy użytkowników i czas startu

-v

format v

```
knoppix@ttyp0[knoppix]$ ps
  PID TTY          TIME CMD
12369 pts/1    00:00:00 bash
17270 pts/1    00:00:00 ps
```

Wyświetlany jest numer PID, terminal sterujący procesu, całkowity czas, w którym proces zajmował procesor, oraz komenda, za pomocą, której proces został uruchomiony. Wyświetlono jedynie te procesy, które pracują na tym samym terminalu, co użytkownik.

przykład:

```
knoppix@ttyp0[knoppix]$ ps ax
PID TTY STAT    TIME    COMMAND
1   ?    S  0:04    init [2]
2   ?   SW  0:19     [keventd]
3   ?   SW  0:00     [kapmd]
4   ?   SWN 0:14     [ksoftirqd_CPU0]
5   ?   SW  2:16     [kswapd]
6   ?   SW  0:00     [bdflush]
7   ?   SW  0:04     [kupdated]
11  ?   SW  2:58    [kjournald]
(...)
```

Wyświetla wszystkie procesy pracujące w systemie. przykład:

```
knoppix@ttyp0[knoppix]$ ps f
PID    TTY      STAT   TIME COMMAND
```

```
19376 pts/17    S          0:00 bash
30005 pts/17    R          0:00  \_ ps f
20673 pts/13    S          0:00 bash
32152 pts/13    S          0:00  \_ mc
15158 pts/15    S          0:00     \_ bash -rcfile .bashrc
(...)
```

Wykorzystanie opcji f powoduje wyświetlenie drzewa procesów, uwzględniających zależność proces macierzysty - proces potomny Za pomocą polecenie pstree można wyświetlić drzewo procesów w systemie. przykład:

```
knoppix@ttyp0[knoppix]$pstree
init-+-MailScanner---5*[MailScanner]
     |-Server
     |-aacraid
     |-acpid
     |-arpwatch
     |-atd
     |-aveserver
     |-clamd
     |-crond
     |-dbus-daemon-1
     |-dccm
     |-dovecot-+-dovecot-auth
```

kill

W wielu przypadkach zachodzi potrzeba usunięcia przez użytkownika procesu z systemu Unix. Użytkownik ma takie prawo w stosunku do swoich procesów natomiast użytkownik root do wszystkich. Polecenia do tego służące ma następującą składnię:

```
kill [ -s sygnał | -p ]  [ -a ] pid ...
```

nazwa	**numer**	**dom. akcja**	**opis**
SIGHUP	1	zakończenie	Wyłączenie terminala sterującego bądź śmierć procesu kontrolującego
SIGINT	2	zakończenie	Przerwanie z klawiatury (CTRL+C)
SIGQUIT	3	zrzut core	Wyjście nakazane z klawiatury
SIGILL	4	zrzut core	Próba wykonania nieprawidłowej instrukcji
SIGABRT	6	zrzut core	Sygnał przerwania pracy procesu wywołany przez abort()
SIGKILL	9	zakończenie	Natychmiastowe usunięcie procesu; niemożliwy do złapania ani zignorowania.
SIGSEGV	11	zrzut core	Nieprawidłowe odwołanie do pamięci wirtualnej
SIGPIPE	13	zakończenie	Zerwany potok: pisanie do potoku, który nie posiada procesu po stronie czytania
SIGALRM	14	zakończenie	Sygnał alarmowy wywołany przez funkcję alarm()

SIGTERM	15	zakończenie	Sygnał zakończenia pracy procesu
SIGCHLD	17	ignorowanie	Zatrzymanie bądź wyłączenie procesu potomnego
SIGCONT	18	start	Kontynuacja zatrzymanego procesu
SIGSTOP	19	zatrzymanie	Zatrzymanie procesu; niemożliwy do złapania ani ignorowania

Po wydaniu polecenia kill z właściwym sygnałem, proces przerywa pracę i wykonuje kod obsługi sygnału. Część sygnałów służy do komunikowania procesu o kluczowych wydarzeniach przez jądro. W tabeli znajdują się najczęściej wykorzystywane sygnały,

Przykład:

```
knoppix@ttyp0[knoppix]$ cat /dev/zero > /dev/null &
[1] 8606
knoppix@ttyp0[knoppix]$kill -9 8606
knoppix@ttyp0[knoppix]$
[1]+  Unicestwiony          cat /dev/zero >/dev/null
knoppix@ttyp0[knoppix]$
```

killall

Stosowanie polecenia *kill* jest niezbyt wygodne, gdyż za każdym razem należy sprawdzić PID zatrzymywanego procesu. W systemach linuksowych dostępne jest polecenie *killall*, które odnajduje proces na podstawie nazwy. Najprostsza jego składnia to:

```
killall [-sygnał] nazwa
```

Uwaga: należy pamiętać, że polecenie wysyła sygnał do wszystkich procesów o podanej nazwie.

fuser

Polecenie wyświetla wszystkie procesy używające dany plik:

```
fuser [-sygnał|-k] plik
```

Opcje pozwalają na wysłanie sygnału (-sygnal) lub zabicie (-k) wszystkich znalezionych procesów. Szczególnie przydatne przed odmontowaniem używanego systemu plików.

top

Top jest programem działającym w czasie rzeczywistym, prezentującym najbardziej absorbujące procesor i pamięć procesy w systemie. Po uruchomieniu, ekran terminala wygląda następująco:

```
20:50:46 up 21 days,  5:22, 35 users,  load average: 0,66, 0,54, 0,43
242 processes: 239 sleeping, 2 running, 1 zombie, 0 stopped
CPU states:   9,3% user,  18,4% system,   0,1% nice,  72,2% idle
Mem:    386248K total,   369808K used,    16440K free,    34032K buffers
Swap:   457844K total,   170436K used,   287408K free,   112924K cached

 PID   USER     PRI    NI  SIZE    RSS   SHARE  STAT   %CPU     %MEM    TIME    COMMAND
```

```
9248   gin      19    0  1916   1916   1584      R   12,4      0.4     0:00   top
  1    root      8    0   808    772    752      S    0,0      0.1     0:04   init
  2    root      9    0     0      0      0     SW    0,0      0.0     0:19   keventd
  3    root      9    0     0      0      0     SW    0,0      0.0     0:00   kapmd
```

(...)

Standardowo, procesy sortowane są według zużycia procesora. Można jednak przełączyć sortowanie, naciskając jeden z klawiszy:

N

według numeru PID

A

według wieku

P

według użycia procesora

M

według użycia pamięci

T

- według czasu pracy

praca w tle, fg, bg, jobs

Polecenia do zarządzania procesami na bieżącej konsoli.

Domyślnie po uruchomieniu procesu jego wyjście kierowane jest na bieżącą konsolę. Jednakże po wciśnięciu klawiszy **Ctrl-Z** konsola zostaje zwolniona a program **zatrzymany i pozostawiony "w tle"** (zakończenie działania powoduje zazwyczaj klawisz *Ctrl-C*!). W tym momencie użytkownik może zdecydować co zrobić z tym procesem.

Innym sposobem uruchomienia programu w tle jest wydanie polecenie zakończonego znakiem *&*:

```
polecenie &
```

Przy pomocy komendy *fg* można ponownie przenieść proces na pierwszy plan. Jednakże wiele procesów może pracować poprawnie w tle. Aby kontynuować pracę procesu w tle należy wydać polecenie *bg*.

Programy *fg* i *bg* uruchomione bez parametrów obsługują ostatnio zatrzymany proces. Istnieje możliwość obsłużenia innego procesu z danej konsoli. W tym celu należy uruchomić powyższe polecenia z parametrem:

```
bg [identyfikator zadania]
fg [identyfikator zadania]
```

W celu pobrania listy działających zadań należy wydać polecenie

```
jobs
```

nohup

Wiele programów nie pozwala na utworzenie procesu nie związanego z konsolą. Zazwyczaj uniemożliwiają tego programy aktywnie komunikujące się z konsolą. Aby umożliwić tym programom pracę w tle służy polecenie:

```
nohup polecenie [argumenty]
```

Polecenie tworzy plik *nohup.out* do którego przekierowany jest wynik działania programu.

Pozostałe polecenia związane z procesami: nice, bg, fg, jobs, killall

Przykład:

```
nohup ping 127.0.0.1 &
```

nice, renice

Procesy w systemach uniksowych mają określone priorytety, które system dobiera automatycznie na podstawie sposobu działania procesu. Użytkownik ma jednak możliwość wpływania na sposób dobierania priorytetu poprzez określenie wartości *nice* wpływającej na to jaki maksymalny priorytet może proces otrzymać. W systemie Linux liczba nice posiada wartości **ujemne** dla preferowanych zadań i dodatnie dla zadań o niższych priorytetach. "Najwyższą" wartością *nice* jest -20. Wartości ujemne może przypisywać procesom jedynie użytkownik *root*.

Aby uruchomić proces z zadaną wartością *nice* należy wydać polecenie:

```
nice [priorytet] polecenie [argumenty]
```

Aby zmienić wartość *nice* bieżącego procesu należy wydać polecenie:

```
renice [priorytet] PID
```

Zadania

- Uruchom program *top* i sprawdź jakie informacje wyświetla oraz jak zmienić sposób sortowania danych. (pomoc - klawisz *?*)
- Spróbuj zmienić częstotliwość odświeżania (*d*). Czy da się ostawić zerowy interwał.
- Uruchom 3 procesy top (na osobnych konsolach lub oknach xterm). Zmień wybranemu wartość *nice*. Sprawdź zachowanie procesów *top* przy zerowych interwałach.
- Utwórz plik i wyświetl go na 3 konsolach poleceniem *less*. Sprawdź działanie programu *fuser*. Spróbuj przy jego pomocy zakończyć działanie przeglądarek *less*.
- W XWindow wprowadź polecenie gedit. Czy możesz na tej konsoli wykonywać inne polecenia? Czy znasz sposób pozwalający na wykonywanie inny poleceń na tej konsoli (poza zakończeniem pracy przez aplikację gedit)?
- Przejdź do konsoli zablokowanej przez gedit i naciśnij kombinację ctrl+z. Spowoduje to zatrzymanie tego procesu. Co się stało z aplikacją?

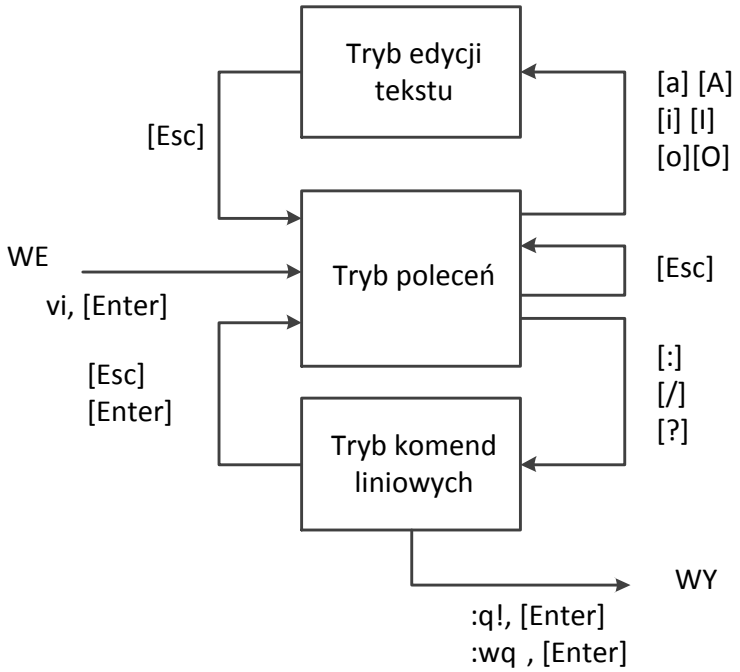
- Na konsoli z zatrzymanym gedit wprowadź polecenie bg 1. Co się stało z aplikacją? Jak wytłumaczysz uzyskany efekt?
- Zamknij konsolę na której pracowałeś. Co stało się z aplikacją gedit?

Edytor vi

Edytor *vi* jest edytorem ekranowym, w trybie tekstowym, pozwala poruszać za pomocą kursorów, dokonywać zmian w tekście i dopisywać nowy tekst. Edytor *vi* można uruchomić z argumentem będącym nazwą pliku. Jeżeli taki plik nie istnieje, to zostanie utworzony nowy np:

```
vi nowy_plik
vi istniejacy_plik
```

Podstawowym faktem, o którym zawsze należy pamiętać przy pracy z edytorem vi jest to, że pracuje on zawsze w jednym z trzech trybów (rysunek poniżej):



- tryb klawiszowy (tryb poleceń) - w tym trybie każdy klawisz na klawiaturze powoduje wykonanie jakiejś akcji na edytowanym pliku
- tryb edycji tekstu (dopisywania) - po wykonaniu (w trybie klawiszowym) komendy inicjującej wpisywanie tekstu - przechodzimy do trybu edycji tekstu. W tym trybie tekst wpisywany z klawiatury jest wstawiany do edytowanego pliku - aż do naciśnięcia klawisza Esc - co spowoduje powrót do trybu klawiszowego
- tryb komend liniowych (tryb poleceń) - jeśli w trybie klawiszowym naciśniemy **':'** (dwukropek) przechodzimy do trybu komend liniowych. W ostatniej linii na ekranie pojawia się : po którym możemy wpisać komendę 'liniową'. Komenda taka

działa zwykle na grupie linii. Po jej wykonaniu *vi* wraca automatycznie do trybu klawiszowego. Bezpośrednio po uruchomieniu *vi* znajduje się w trybie poleceń.

Przejście z trybu dopisywania do trybu poleceń odbywa się poprzez wciśnięcie klawisza Esc (escape). Przejście z trybu poleceń do trybu dopisywania odbywa się zwykle po wciśnięciu klawisza a (append - dołączanie tekstu za aktualnie wskazywanym znakiem) lub i (insert - wstawianie tekstu przed aktualnie wskazywanym znakiem). Oczywiście w trybie dopisywania klawisz a powoduje wstawienie do tekstu znaku a, a klawisz i - znaku i. Będąc w trybie poleceń możemy przejść do tzw. wiersza poleceń, który umożliwia wydawanie komend słownych, wykonujących pewne operacje na dokumencie. Wiersz poleceń uaktywnia się przez : (dwukropek), a powrót do trybu poleceń uzyskujemy, podobnie jak w trybie dopisywania, przez wciśnięcie Esc.

NAWIGACJA W TEKŚCIE:

- w większości nowoczesnych terminali działają klawisze kursora (strzałki) oraz PageUp, PageDown, Home i End (lub część z nich)
- h - lewo, j - dół, k - góra, l - prawo
- 0 - na początek wiersza, $ - na koniec wiersza
- Ctrl-f - jeden ekran w dół, Ctrl-b - jeden ekran w górę
- w - na początek następnego słowa, b - na początek poprzedniego słowa
- + - do pierwszego znaku w następnym wierszu, - - do pierwszego znaku w
- poprzednim wierszu
- nG - do wiersza nr n (również poprzez :n[enter])
- r*znak* - zastąpienie znaku pod kursorem znakiem *znak*
- gg – przejście do początku pliku
- G – przejście na koniec pliku

- KOŃCZENIE PRACY I ZAPISYWANIE TEKSTU:
- ZZ - wyjście z zapisaniem tekstu
- :q - wyjście bez zapisania tekstu o ile nie był on zmieniony
- :q! - wyjście bez zapisania tekstu nawet jeśli został zmieniony (zmiany przepadną)
- :w - zapis bieżącego dokumentu i kontynuacja pracy z vi

- PRZEJŚCIE DO TRYBU WPROWADZANIA:
- a - dopisywanie za znakiem wskazywanym przez kursor
- i - dopisywanie przed znakiem wskazywanym przez kursor

- A - dopisywanie na końcu wiersza (równoważne z $a)
- I - dopisywanie na początku wiersza (równoważne z 0i)
- R - dopisywanie w trybie nadpisywania (zastępowania)
- o - dopisywanie w nowym wierszu poniżej kursora
- O - dopisywanie w nowym wierszu powyżej kursora
- Esc - przejście do trybu poleceń
- :r plik - wstawia zawartość pliku plik poniżej wiersza z kursorem (pozostaje w
- trybie poleceń)
-
- MANIPULACJA DOKUMENTEM:
- J - połącz dwa wiersze
- dd - usuń bieżący wiersz
- D - usuń tekst od pozycji kursora do końca wiersza (jak d$)
- d0 - usuń tekst od początku wiersza do pozycji kursora
- dG - usuń tekst od pozycji kursora do końca dokumentu
- dw - usuń następne słowo
- db - usuń poprzednie słowo
- dd – usuń linię (np. 20dd – usuń 20 linii)
- yy - zaznacz bieżący wiersz do wstawienia (kopiowanie do bufora)
- p - wstaw bufor poniżej (lub na prawo, zależnie od typu bufora) kursora
- P - wstaw bufor powyżej (lub na lewo, zależnie od typu bufora) kursora
- cx - zamiana (usunięcie i przejście do trybu dopisywania - x jest odpowiednikiem drugiego znaku polecenia d)
- x - usunięcie znaku pod kursorem
- X - usunięcie znaku na lewo od kursora xp - zamiana miejscami dwóch kolejnych znaków (korekta czeskiego błędu) - kursor
- powinien być na pierwszym z nich (odpowiednik usunięcia x i wstawienia na prawo
- od kursora p)
- u - cofnij ostatnio dokonaną zmianę w tekście (undo)
- yx - zaznaczenie fragmentu tekstu (kopiowanie do bufora) - x jest odpowiednikiem drugiego znaku polecenia d (np. yw - kopiowanie do bufora następnego słowa, 3yl - kopiowanie trzech kolejnych znaków) (UWAGA: usuwanie tekstu przez d również kopiuje tekst do bufora)
- "buforpolecenie - wstawia tekst (polecenie to któryś z wariantów y lub d) do

- bufora o nazwie bufor (nazwa powinna być jednym znakiem)
- "buforp lub "buforP - wstawia zawartość bufora bufor za lub przed kursorem
- (ew. poniżej lub powyżej kursora)
- . - ponów ostatnią czynność zmieniającą tekst (działa w bieżącym położeniu
- kursora)
- x,yw plik - zapisz wiersze od x do y do pliku plik (x,y mogą być numerami
- wierszy, znakiem . oznaczającym bieżący wiersz lub kombinacją .+n oznaczającą
- bieżący i n kolejnych wierszy, np. .,.+2w plik zapisuje trzy kolejne wiersze,
- począwszy od bieżącego, do pliku plik)
- ~ - zmiana wielkości liter (z małych na wielkie i odwrotnie) - działa na znak pod
- kursorem
- f*znak* - przesuwa kursor w bieżącym wierszu do pierwszego wystąpienia znaku *znak*

UWAGA: większość poleceń można wykonywać wielokrotnie przez podanie liczby ponowień przed

poleceniem, np. 3dw usunie 3 kolejne słowa, 2dd usunie 2 kolejne wiersze itp.

WYSZUKIWANIE I ZAMIANA:

- /tekst - ustawia kursor na pierwszym wystąpieniu tekst (szuka w dół od kursora)
- ?tekst - jak /tekst, lecz szuka w górę od kursora
- n - ustawia kursor na kolejnym wystąpieniu tekstu szukanego przez /tekst lub ?tekst
- :s/txt1/txt2 - zamienia pierwsze wystąpienie txt1 w bieżącym wierszu tekstem txt2
- & - ponawia ostatnio wykonaną zamianę od bieżącej pozycji kursora
- :s/txt1/txt2/g - zamienia wszystkie wystąpienia txt1 w bieżącym wierszu
- tekstem txt2
- :%s/txt1/txt2/g - zamienia wszystkie wystąpienia txt1 w całym dokumencie tekstem txt2 (jeśli zamiast /g podamy /gc konieczne będzie potwierdzenie każdej zamiany)
- :g/txt/polecenie - polecenie działające globalnie na wierszach zawierających txt, np. :g/^$/d - usuwa wszystkie puste wiersze z dokumentu
- mznak - oznacza bieżący wiersz symbolem znak (pojedyncza litera) - można się do takiego oznaczenia odwołać w innych poleceniach przez 'znak, np. ma w wierszu,
- przejście 2 wiersze w dół, mb, po czym :'a,'bw plik - zapisze wiersze od oznaczonego

przez a do oznaczonego przez b włącznie do pliku plik

INNE:

- :map znak sekwencja - mapuje (przypisuje do znaku) sekwencję poleceń, co umożliwia posługiwanie się utworzonym skrótem
- :ab skrót tekst - wpisanie skrót (w trybie dopisywania) rozwijane jest do tekst po wciśnięciu Esc bezpośrednio za wpisanym skrótem :unab skrót - usuwa zapamiętany skrót (od tego momentu nie będzie on już rozwijany)
- nieużywane znaki (bez przypisanych im poleceń): g, K, q, V, v
- nieużywane symbole: _, *, \, =
- :e plik - otwarcie nowego dokumentu
- :e # - przejście do poprzedniego dokumentu
- :e nazwa - przejście do dokumentu o podanej nazwie
- :%nu - chwilowo pokazuje numery wierszy w dokumencie
- : set number - pokazuje numery wierszy

Efektywne przesuwanie bloków tekstu (vim)

Używaj trybu zaznaczania i jego różnych trybów

W odróżnieniu od oryginalnego vi, vim pozwala na podświetlanie zaznaczenia tekstu i

operacje na nim. Istnieją trzy główne tryby selekcji wizualnej (trybu zaznaczania tekstu). Są to:

- **v** Zaznaczanie na poziomie znaków. To tryb do których większośd osób jest przyzwyczajona, dlatego wypróbuj go przed przed innymi trybami selekcji.
- **V** Zaznaczanie na poziomie linii. Zawsze zaznaczane są całe linie.Ten tryb jest bardziej przydatny, gdy chcesz kopiować lub przenosić grupę linii.
- **<Ctrl+V>** Zaznaczanie blokowe. Interesujące narzędzie, dostępne w nielicznych edytorach. Można zaznaczyć prostokątny blok i każdy tekst w nim będzie podświetlony.

Wszystkie standardowe klawisze ruchu w trybach selekcji wizualnej działają normalnie – na przykład vwww przestawi edytor w tryb selekcji wizualnej i zaznaczy trzy następujące słowa. Vjj przestawi vim-a w tryb selekcji wizualnej linii i zaznaczy bieżącą i dwie następne linie.

Po zaznaczeniu tekstu użycie klawiszy **y** lub **d** w trybie poleceń pozwala odpowiednio na skopiowanie lub wycięcie zaznaczenia do bufora. Zastosowanie klawiszy **P** lub **p** umożliwia natomiast wstawienie zaznaczonego tekstu odpowiednio przed lub za kursorem

---

Zadania

- Napisać używając edytora vi moduł “main.c” prostego programu C wykonywanego na ostatnim laboratorium Podstaw iIformatyki . Ponumerować linie programu aby ułatwić wyszukiwanie będów. Skompilować i skonsolidować program przy pomocy polecenia w konsoli

  ```
  gcc  main.c -o program
  ```

Skrypty powłoki

W każdym systemie Unix/Linux dostępnych jest kilka powłok (shell). Ich zmianę można dokonać za pomocą polecenia chsh, natomiast lista wszystkich dostępnych powłok i ich ścieżek zapisana jest w pliku ***/etc/shells***. Za pomocą poleceń udostępnianych przez powłoki możemy w systemach unixowych tworzyć skrypty, które ułatwiają administrowanie systemem i wykonywanie powtarzających się pracochłonnych czynności.

Skrypt jest plikiem tekstowym, o pewnej strukturze, zawierającym polecenia charakterystyczne dla danej powłoki i nadanym atrybucie wykonywalności. Polecenia w nim zawarte będą wykonywane w takiej kolejności, w jakiej byłyby wpisywane z klawiatury.

Podstawowa struktura skryptu wygląda następująco:

```
#!/bin/bash
polecenie1
polecenie2
```

Istotną cechą skryptu jest, że zawsze zaczyna się znakami **#!** z dołączona ścieżką powłoki w jakiej ma być wykonywany skrypt. Należy przestrzegać tego zapisu i stosować polecenia danej powłoki, ponieważ w innym przypadku skrypt nie będzie wykonywany. W trakcie tworzenia skryptu możliwe jest wykonywanie (testowanie) skryptu bez nadawania atrybutu wykonywania, w tym celu należy uruchomić go w następujący sposób:

```
. skrypt1
```

Nazwę skryptu poprzedzamy kropką i spacją a następnie wpisujemy nazwę skryptu. W przypadku wywołania skryptu w powyższy sposób, wykonywany on jest przez bieżącą powłokę. Jeśli wykonamy skrypt poprzez nadanie mu praw wykonania – zostanie utworzony nowy proces powłoki wykonujący skrypt. Różnica ta może mieć pewne znaczenie w przypadku różnych powłoki i bardziej złożonych skryptów (operujących na zmiennych).

```
#!/bin/bash
# Skrypt powitalny
```

```
echo Witaj $USER. Zalogowałeś się w systemie
echo Miłego dnia
echo Aktualnie w systemie są zalogowani następujący użytkownicy:
w
sleep 5
clear
```

Powyższy skrypt uruchamiany jest w powłoce bash (*/bin/bash*). Poleceniem echo wyświetlamy następujące po nim ciągi znaków oraz zawartości zmiennych. W tym przypadku nazwę logowanego użytkownika. Kolejne plecenie (w) wyświetla, kto jest aktualnie zalogowany, następnie odczekuje 5 sekund i czyści ekran monitora (clear). Za pomocą znaku # umieszczonego w pierwszej kolumnie wiersza możemy w skrypcie umieszczać komentarze.

Przykładem bardziej zaawansowanego zastosowania skryptu jest przekopiowywanie plików z rozszerzeniem bak do nowo utworzonego katalogu:

```
#!/bin/bash
echo Podaj nazwę katalogu
read zmienna
mkdir $zmienna
for i in *
do
cp $i.bak $zmienna/$i.bak
done
```

Skrypt wyświetla komunikat i czeka na wprowadzenie nazwy katalogu (read) zapamiętanej w zmiennej *zmienna* i tworzy w bieżącej lokalizacji ten katalog. Następnie wykonuje instrukcje pętli (for) przeglądając wszystkie pliki i kopiuje pliki z rozszerzeniem bak do nowoutworzonego katalogu

Powłoki dysponują bardziej zaawansowanymi sposobami wpływania na prace systemu niż to przedstawiono, z powodu ograniczeń czasowych, w niniejszej instrukcji. By móc je wykorzystywać należy zapoznać się z ich opisem dostępnym np. w manualu bash.

Powłoka Bourne'a

Powłoka Bourne’a jest zarówno interpreterem wiersza poleceń (przetwarza wtedy wprowadzone przez użytkownika komendy), jak i zaawansowanym językiem programowania (przetwarza skrypty przechowywana w plikach). Powłoka Bourne'a jest jedną z trzech dostępnych powłok w systemach z rodziny Unix. Inne powłoki to CShell oraz KornShell. BASH (ang. Bourne Again SHell) to stworzony przez Briana Foxa i Cheta Rameya zgodny z powłoką Bourne'a (czyli sh) interpreter poleceń, łączący w sobie zalety powłoki ksh i csh. Powłoka ta jest to najbardziej popularna powłoka używana na systemach z rodziny Linux

Pierwszy skrypt

Spróbujmy stworzyć nasz pierwszy skrypt wyświetlający na ekranie napis. W tym celu należy stworzyć odpowiedni plik, w którym umieścimy kod. Następnie przy pomocy

dowolnego edytora dostępnego w systemie Linux wprowadzimy odpowiednie polecenia:

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

Znak # (hash) oznacza komentarz. A więc wszystko, co znajduje się za nim w tej samej linii, jest pomijane przez interpreter.

Polecenie echo "Hello world" wydrukuje na standardowym wyjściu (stdout) napis: Hello world.

Po nadaniu naszemu skryptowi uprawnienia execute skrypt ten można uruchomić. Jeśli katalog bieżący, w którym znajduje się skrypt nie jest dopisany do zmiennej PATH i skrypt został zapisany w pliku o nazwie przykładowy_skrypt to można go uruchomić w następujący sposób:

```
./przykładowy_skrypt
```

Komunikacja skryptu z użytkownikiem

Polecenie echo

Polecenie echo umożliwia wydrukowanie na standardowym wyjściu napisu podanego jako parametr.

Składnia:

```
echo parametry tekst_do_wyświetlenia
```

Parametry:

-n nie jest wysyłany znak nowej linii
-e włącza interpretacje znaków specjalnych

Przykłady:

```
#!/bin/bash
echo -n "Napis1"
echo "Napis2"
```

Polecenie read

Polecenie read umożliwia odczytanie ze standardowego wejścia pojedynczego wiersz.

Składnia:

```
read parametry nazwa_zmiennej
```

Spróbujmy użyć polecenia read w skrypcie. Utwórzmy skrypt o nazwie odczyt. Kod tego skryptu przedstawiony jest poniżej.

```
#!/bin/bash
echo -ne "Wprowadź tekst:\a"
```

```
read wpis
echo "$wpis"
```

Po uruchomieniu otrzymamy następujący rezultat:

```
$./odczyt
Wprowadź tekst:
Jakiś tekst
Jakiś tekst
$
```

Wprowadzony przez użytkownika tekst zostanie zapisany w zmiennej wpis (polecenie read wpis). Zmienna ta zostanie następnie wypisana na ekran przy użyciu polecenia echo "$wpis". Polecenie read pozwala również na przypisanie kilku wartości kilku zmiennym. Przedstawia to skrypt odczyt_wielu zaprezentowany poniżej.

```
#!/bin/bash
echo "Wpisz trzy wyrazy:"
read a b c
echo $a $b $c
echo "Wartość zmiennej a to: $a"
echo "Wartość zmiennej b to: $b"
echo "Wartość zmiennej c to: $c"
```

Wybrane parametry polecenia read:

-p Pokaże znak zachęty bez kończącego znaku nowej linii

-a Kolejne wartości przypisywane są do kolejnych indeksów zmiennej tablicowej

```
#!/bin/bash
echo "Podaj elementy zmiennej tablicowej:"
read tablica
echo "${tablica[*]}"
```

-e

Jeśli nie podano żadnej nazwy zmiennej, wiersz trafia do $REPLY

```
#!/bin/bash
echo "Wpisz coś:"
read -e
echo "$REPLY"
```

Zmienne w skryptach

W językach skryptowych wyróżniamy kilka rodzajów zmiennych. Są to

- zmienne programowe;
- zmienne środowiskowe;
- zmienne specjalne;
- zmienne tablicowe.

Zmienne programowe

Zmienne programowe są to zmienne, które zostały zdefiniowane samodzielnie przez

użytkownika. W skrypcie definiujemy je pisząc po prostu:

```
nazwa_zmiennej="wartość"
```

np.:

```
x="napis",
```

a odwołujemy się do niej poprzez polecenie:

```
$nazwa_zmiennej,
np.
read x
echo $x
```

Do wyzerowania zmiennej używamy polecenia:

```
nazwa_zmiennej=.
```

Należy pamiętać, że **nie może być spacji** po obu stronach znaku =, dlatego też linia x = "napis" jest błędna

Za zmienną można również podstawić polecenie umieszczone w tzw. odwrotnych apostrofach (`). W czasie wykonywania takiego skryptu, interpreter zastąpi komendę, która się tam znajduje, wynikiem jej działania. Przedstawia to poniższy przykład.

```
$cat listuj
#!/bin/bash
dir=`pwd`
echo 'Aktualnie korzystasz z katalogu ' $dir 'na Twoim dysku'
$
```

Efekt działania powyższego przykładu przedstawia poniższy wydruk:

```
$./listuj
Aktualnie korzystasz z katalogu /home/user/skrypt na Twoim dysku
$
```

Przykład ten pokazał dwa istotne elementy. Pierwszy z nich to mechanizm podstawiania poleceń. Drugi element, na który warto zwrócić uwagę, to sklejanie łańcuchów znaków, z którego skorzystaliśmy w tym przykładzie.

Istnieje jeszcze jeden sposób podstawiania poleceń. Zrealizowane to może zostać przy użyciu mechanizmu rozwijania zawartości nawiasów: $(polecenie). Działanie tego mechanizmu wygląda podobnie, jak poprzednio, co pokazuje poniższy przykład.

```
$cat listuj2
#!/bin/bash
dir=$(pwd)
echo "Aktualnie korzystasz z katalogu $dir na Twoim dysku"
$
```

Zmienne środowiskowe

Zmienne środowiskowe służą do zdefiniowania środowiska użytkownika dostępnego dla wszystkich procesów potomnych. Dzielimy je na:

- globalne
- lokalne.

Poniższy przykład obrazuje różnicę między nimi. W tym celu należy uruchomić polecenie xterm (terminal w środowisku X Window) i wpisać:

```
x="Przykładowy napis"
echo $x
xterm
```

Ostatnie polecenie spowoduje wywołanie kolejnego terminala. Jeśli teraz w nowym terminalu wydamy polecenie echo $x, to nie zobaczymy nic, gdyż zmienna x jest zmienną lokalną, niewidoczną w innych podpowłokach.

Aby zmienną x uczynić globalną, należy wydać polecenie export x="napis", dzięki czemu będzie ona widoczna w innych podpowłokach. Polecenie export nadaje zmiennym atrybut zmiennych globalnych, natomiast wywołane bez parametrów wyświetla listę aktualnie eksportowanych zmiennych.

W przypadku powłoki bash na liście tej pojawi się polecenie declare. Jest to wewnętrzne polecenie tej powłoki służące do definiowania zmiennych i nadawania im odpowiednich atrybutów (-x parametr odpowiadający poleceniu export). Należy pamiętać, że polecenie to dostępne jest jedynie dla powłoki bash i nie występuje w innych powłokach.

Przykładowe zmienne środowiskowe to m.in.:

```
$HOME #ścieżka do katalogu domowego użytkownika
$USER #login użytkownika
$HOSTNAME #nazwa hosta na który zalogowany jest użytkownik
$OSTYPE #rodzaj systemu operacyjnego
```

Wszystkie dostępne zmienne środowiskowe można wyświetlić za pomocą polecenia:

```
printenv | more
```

Zmienne specjalne

Zmienne specjalne są to najbardziej prywatne zmienne powłoki, które są udostępniane użytkownikowi tylko do odczytu (z pewnymi wyjątkami). Poniżej przedstwiono kilka przykładów:

$0

nazwa bieżącego skryptu lub powłoki. Przykład:

```
#!/bin/bash
echo "$0"
Skrypt ten wyświetli nazwę naszego skryptu.
```

```
```

$1..$9

Parametry przekazywane do skryptu (użytkownik może modyfikować ten rodzaj zmiennych specjalnych.

```
#!/bin/bash
echo "$1 $2 $3"
```

Wywołanie tego skrypt z parametrami spowoduje, że zostaną one przypisane zmiennym od $1 do $9.

$@

Pokaże wszystkie parametry, które zostały przekazywane do skryptu (możliwość modyfikacji), równoważne $1 $2 $3..., jeśli nie podane są żadne parametry $@ interpretowany jest jako pusty:

```
#!/bin/bash
echo "Skrypt uruchomiono z parametrami: $@"
```

$?

kod powrotu ostatnio wykonywanego polecenia

$$

PID procesu bieżącej powłoki

Zmienne tablicowe

BASH pozwala na stosowanie jednowymiarowych zmiennych tablicowych. W BASH'u nie ma górnego ograniczenia rozmiaru tablic. Kolejne wartości zmiennej tablicowej indeksowane są przy pomocy kolejnych liczb całkowitych, zaczynających się od 0.

Składnia: **zmienna=(wartos c1 wartosc2 wartosc3 ... wartoscn)**

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[0]}
echo ${tablica[1]}
echo ${tablica[2]}
```

Zadeklarowana zmienna tablicowa o nazwie: *tablica*, zawiera trzy elementy: *elementl element2 element3*. Polecenie: **echo ${tablica [0]}** wyświetli na ekranie pierwszy elementu tablicy. W powyższym przykładzie w ten sposób wypisana zostanie cała zawartość tablicy. Do elementów tablicy odwołujemy się za pomocą indeksów.

${nazwa_zmiennej[indeks]}

Indeksy to liczby od 0 do n-1 (gdzie n jest liczbą elementów tablicy) oraz **@** i *****. Gdy

odwołując się do zmiennej nie podamy indeksu: **${nazwa_zmiennej}** to nastąpi odwołanie do elementu tablicy o indeksie 0. Jeśli jako indeks podamy **@** lub ***** to zwrócone zostaną wszystkie elementy tablicy.

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[*]}
```

BASH umożliwia też uzyskanie informacji o długości (liczbie znaków) danego elementu tablicy:

${#nazwa_zmiennej[indeks]}.

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${#tablica[0]}
```

Polecenie echo **${#tablica[0]}** wydrukuje liczbę znaków, z jakich składa się pierwszy element tablicy (czyli 8). W podobny sposób można otrzymać liczbę wszystkich elementów tablicy - wystarczy jako indeks podać **@** lub *****. Przykład:

```
#!/bin/bash
tablica=(element1   element2   element3)
echo ${#tablica[@]}
```

Aby zmienić lub dodać element tablicy należy użyć następującej składni:

nazwa_zmiennej[indeks]=wartosc.

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
tablica[3]=element4
echo ${tablica[@]}
```

Dany element tablicy usuwa się za pomocą polecenia **unset**, o składni:

unset nazwa_zmiennej[indeks]

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[@]}
unset tablica[2]
echo ${tablica[*]}
```

Usunięty został ostatni element tablicy. Aby usunąć całą tablicę wystarczy podać jako indeks: **@** lub *****. Przykład:

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

```
tablica=(element1 element2 element3)
unset tablica[*]
echo ${tablica[@]}
```

Cytowanie

Znaki cytowania pozwalają na wyłączenie mechanizmu interpretacji znaków specjalnych przez powłokę. Wyróżniamy następujące znaki cytowania:

cudzysłów (ang. double quote) - " "

Między cudzysłowami umieszcza się dowolny tekst oraz zmienne poprzedzielane spacjami. Cudzysłowy zachowują znaczenie specjalne trzech znaków:

- $ - wskazuje na nazwę zmiennej, umożliwiając podstawienie jej wartości
- \ - znak maskujący
- `` - odwrotny apostrof, umożliwia zacytowanie polecenia

```
#!/bin/bash
x=2
echo "Wartość zmiennej x to $x"
echo -ne "Dzwonek\a"
echo "Polecenie date pokaże: `date`"
```

apostrof (ang. single quote) - ' '

Elementy ujęte w symbol apostrofu traktowane są jak łańcuch tekstowy. Apostrof wyłącza interpretowanie wszystkich znaków specjalnych.

```
#!/bin/bash
echo '$USER' #nie wypisze twojego loginu
```

odwrotny apostrof (ang. backquote) - ``

Pozwala na zacytowanie polecenia (przydatne gdy chcemy podstawić pod zmienną wynik działania jakiegoś polecenia):

```
#!/bin/bash
x=`ls -la $PWD`
echo $x
```

Powłoka *bash* udostępnia również alternatywny zapis podstawienia polecenia, w postaci $(...). Na przykład:

```
x=$(ls -la $PWD)
```

**Znak maskujący (ang. backslash) - **

Symbol ten jest przydatny, gdy chcemy na ekranie wyświetlić się napis jakiś symbol dla systemu zastrzeżony np. napis $HOME. Obrazuje to poniższy przykład.

```
echo "$HOME" #na ekranie /home/katalog_domowy_użytkownika
echo \$HOME #na ekranie pojawi się napis $HOME
```

Pierwsza linia wypisze na ekranie katalog domowy użytkownika, który uruchomił ten skrypt, w drugiej linii dzięki użyciu \ została wyłączona interpretacje przez powłokę zmiennej $HOME.

Operacje arytmetyczne

Podstawą operacji arytmetycznych w powłoce jest polecenie expr. expr wykonuje obliczenie i zapisuje rezultat na standardowe wyjście. Każdy żeton wyrażenia musi być oddzielnym argumentem. Operandy mogą być liczbami lub ciągami znaków. Łańcuchów znaków nie cytuje się dla expr, choć możesz być zmuszonym do tego, by ochronić je przed powłoką (znaki i łańcuchy o specjalnym znaczeniu dla powłoki, np. spacja). Dopupszczalne są następujące operacje:

- operacje arytmetyczne na liczbach całkowitych
  +, -, *, / ;
- operacje logiczne. Wartość prawdziwa oznaczane jest liczbą 1, zaś fałszywa 0 (!)
  <, <=, =, ==, !=, >=, >

Operatory o specjalnym znaczeniu dla powłoki (np. '<' lub '*') należy poprzedzić znakiem '\'.

Powłoka *bash* udostępnia alternatywny, znaczenie wygodniejszy zapis w postaci $((...)). Operacja wewnątrz nawiasów zgodna jest ze składnią języka C.

Na przykład:

```
$ a=2
$ a=`expr $a + 1`
$ echo $a
3
$ echo $((2+2))
4
$ echo $(((12+43)/3))
18
$ echo $((2==3))
0
$ echo $((4>5))
0
$ a=2
$ a=$((a+1))
$ echo $a
3
```

Można także użyć expr do wykonywania operacji na zmiennych:

```
$ expr 2 + 2
4
$ expr \( 12 + 43 \) / 3
18
$ expr 2 == 3
0
$ expr 4 \> 5
0
```

Zadania

- Co to jest skrypt?

- Jakie uprawnienia powinien posiadać plik zawierający skrypt, aby możliwe było jego uruchomienie?
- Co oznacza zmienna środowiskowa $USER ?
- Jakie informacje przechowuje zmienna specjalna $$?
- Co oznacza znak # w skrypcie?
- Co powoduje dodanie do polecenia echo opcji -e ?
- Gdzie zostanie umieszczony wprowadzony tekst po wydaniu polecenia read -e ?
- Napisz skrypt, który wypisze na ekran twoje imię i nazwisko oraz zawartość twojego katalogu domowego.
- Napisz skrypt, który będzie Cię pytał o datę urodzin, a następnie wyświetlał ją na ekranie w postaci komunikatu: Urodziłeś się ….
- Skrypt z poprzedniego zadania zmodyfikuj tak, aby data urodzin była wprowadzana w postaci parametru.
- Napisz skrypt, który poprosi Cię o wpisanie dowolnego zdania, następnie poprosi o numer wyrazu w zdaniu i wyświetli ten wyraz.