Systemy operacyjne i sieci – I

Laboratorium – materiały i konspekty zajęć, wersja 1.0

***Opracowanie**: mgr inż. Krzysztof Dorosz przy wykorzystaniu opracowania zagadnień do przedmiotu SOS1 studentów Grzegorza Kozieł, Grzegorza Prucnal oraz Alicji Cyganiewicz kierunku EPI w roku akademickim 2010/2011.*

Laboratorium 1. Wprowadzenie do systemu Unix/Linux 2

Laboratorium 2. Podstawowe polecenia Linux (obsługa plików i katalogów) 6

Laboratorium 3. Użytkownicy, grupy i prawa dostępu 11

Laboratorium 4. Procesy, deamony i sygnały 18

Laboratorium 5. Wejścia i wyjścia procesu 26

Laboratorium 6. Zmienne środowiskowe 31

Laboratorium 7: Filtrowanie strumienia tekstu i wyrażenia regularne 37

Laboratorium 8: Skrypty powłoki bash 42

Laboratorium 9. Kompresja i narzędzia sieciowe 49

Laboratorium 10. Cykliczne uruchamianie programów oraz kontrola wersji dokumentów tekstowych 59

Chciałbym bardzo podziękować moim studentom, którzy opracowując zagadnienia do przedmiotu bardzo pomogli w powstaniu tego skryptu laboratoriów.

DK

Wszelkie uwagi i erraty mile widziane na adres krzysztof.dorosz@uj.edu.pl.

Laboratorium 1. Wprowadzenie do systemu Unix/Linux

Zagadnienia do laboratorium

Linux – rodzina uniksopodobnych systemów operacyjnych opartych o jądro Linux. Pełny system operacyjny oprócz jądra potrzebuje jeszcze powłoki systemowej, kompilatora, bibliotek, itp.

Jądro Linux

Jądro systemu - inaczej kernel; jest to ta część oprogramowania, która odpowiada za:

- komunikację systemu z urządzeniami (sterowniki, które w Linuksie nazywane są modułami),
- uruchamianie i komunikację pomiędzy programami,
- obsługę systemów plików - ma bezpośredni dostęp do dysków i plików,
- stworzenie środowiska uruchomieniowego dla programów (np. gdy program rezerwuje potrzebną ilość pamięci, to właśnie kernel przyznaje mu taki rejon, który nie jest przez żadne inne programy używany).

Powłoka

Nazywana inaczej shellem, odpowiada za wykonywanie poleceń wpisywanych przez użytkownika w trybie tekstowym. Samo jądro systemu nie "rozumie" wpisywanych przez użytkownika poleceń, to właśnie powłoka przyjmuje polecenia tekstowe, i odpowiednio je tłumaczy na funkcje kernela, jakie ma wykonać - porozumiewa się z kernelem w formie binarnej, bezpośrednio, bo właśnie taką formę poleceń kernel ma wbudowaną. Ponadto powłoka decyduje, czy przekazać polecenie do kernela, czy je wykonać za pomocą innego programu.

Najpopularniejszą powłoką w GNU/Linuksie jest bash (Bourne Again Shell).

Rodzaje powłok:

- Powłoka Bourne'a lub sh – powłoka dla systemów Unix. Stworzona została w laboratoriach AT&T przez Stephena Bourne'a przez rozbudowanie prostego interpretera poleceń o nazwie shell,
- zsh (Z shell) – uniksowa powłoka (ang. shell) nadająca się zarówno do interaktywnej pracy z systemem jak i do wykonywania skryptów. Spośród standardowych powłok zsh najbardziej przypomina Korn shell, ale zawiera wiele ulepszeń. Zsh posiada edycję wiersza poleceń, wbudowaną korekcję pisowni, programowalne dopełnianie poleceń, funkcje (z automatycznym ładowaniem), historię poleceń i mnóstwo innych cech,
- csh (od ang. C shell) jest jedną z systemowych powłok uniksowych. Została stworzona przez Billa Joya dla systemu BSD. Nazwa C shell jest grą słów: dosłownie oznacza powłokę [o składni] języka C, fonetycznie da się to jednak odczytać sea shell, czyli muszelka,
- Powłoka ta pochodzi od `/bin/sh`. Składnia języka zastosowanego w powłoce jest bazowana na języku C. Powłoka C shell wniosła wiele ulepszeń w stosunku do sh, takich jak m.in. aliasy i historia komend. Obecnie csh nie jest zbyt często wykorzystywana; zamiast niej w użyciu są takie powłoki jak tcsh, Korn shell (ksh) oraz GNU bash. Jej następcą jest z kolei tcsh,
- Tchs,

- Ksh,
- bash – powłoka systemowa UNIX napisana dla projektu GNU. Program jest rozprowadzany na licencji GPL. Bash to jedna z najpopularniejszych powłok systemów uniksowych. Jest domyślną powłoką w większości dystrybucji systemu GNU/Linux oraz w systemie Mac OS X od wersji 10.3, istnieją także wersje dla większości systemów uniksowych. Bash jest także domyślną powłoką w środowisku Cygwin dla systemów Win32. Bash umożliwia pracę interaktywną i wsadową. Język basha umożliwia definiowanie aliasów, funkcji, zawiera konstrukcje sterujące przepływem (if, while, for, ...). Powłoka bash zachowuje historię wykonywanych poleceń. Jest ona domyślnie zapisywana w pliku .bash_history w katalogu domowym użytkownika.

Są jeszcze inne powłoki, a także środowiska graficzne np. GNOME, KDE.

Pliki i systemy plików

W systemie Linux, podobnie jak i w innych systemach z rodziny UNIX, plik jest jednym z podstawowych pojęć. Definiuje się go jako "wirtualną jednostkę magazynowania informacji". Co ciekawe, plik w systemie Linux (i innych Uniksach) może reprezentować np. dysk twardy czy pamięć fizyczną komputera, dowolne urządzenie fizyczne, lub też reprezentować dane pobierane ze stanu kernela systemu.

Dla potrzeb systemu GNU/Linux został opracowany specjalny system plików - Extended Filesystem (extfs). Istnieje kilka systemów plików używanych w GNU/Linuksie jako główny system:

- ext2 (wersja 2 extfs) – drugi rozszerzony system plików dla systemu Linux. Ext2 zastąpił rozszerzony system plików ext. Rozpoznanie uszkodzenia systemu plików (np. po załamaniu się systemu) następuje przy starcie systemu, co pozwala na automatyczne naprawianie szkód za pomocą oddzielnego programu (e2fsck), uszkodzone pliki zapisywane są w katalogu `lost+found`. System plików ext2 zawiera mechanizm zapobiegający znacznej fragmentacji danych, co zdarzało się podczas używania poprzedniej jego wersji. ext2 przy domyślnym rozmiarze bloku (4 KB) obsługuje partycje o wielkości do 16 TB i pojedyncze pliki o wielkości do 2 TB. Nazwy plików mogą mieć do 255 znaków długości. Ważnym elementem systemu ext2 są wolne pola w strukturach danych – to dzięki nim między innymi, możliwa jest konwersja „w locie" do systemu ext3 – wykorzystuje on po prostu część z nich do przechowywania swoich danych.
- ext3 (ext2 tylko z dodanym tzw. plikiem dziennika - narzędziem umożliwiającym łatwe odzyskanie danych w przypadku awarii systemu plików)
- ext4
- ReiserFS – umożliwia utworzenie w jednym katalogu bardzo dużej liczby plików; bardzo wydajny i szybki; idealny jako filesystem dla katalogu `/tmp`.
- XFS
- JFS
- FAT 32 – odmiana systemu plików FAT, po raz pierwszy zastosowany w systemie operacyjnym Windows 95 OSR2, następca FAT16. FAT32 (który pomimo nazwy sugerującej 32 bity, używa tylko 28 bitów) pozwalając teoretycznie na opisanie 268.435.438 klastrów, co umożliwiałoby użycie go nawet na wielo-terabajtowych dyskach twardych. W rzeczywistości, z powodu ograniczeń wbudowanych w program

użytkowy "fdisk" firmy Microsoft, który obsługuje maksymalnie 4.177.920 klastrów (~222), wielkość dysku obsługiwanego w tym systemie plików nie może przekroczyć 2 terabajtów. Więcej informacji na ten temat można znaleźć na stronie Microsoftu [1]. Maksymalny rozmiar pliku to 4 GB - 1 B (232 B - 1 B). Limit 2TB jest podyktowany limitem wielkości partycji w tablicy partycji, zaś limit wielkości pliku wynika z kompatybilności ze starszymi systemami (4G-1 to limit wielkości pliku od czasów DOSa w wersji 3.x - pomimo ówczesnego limitu 32MB na partycję dyskową). W momencie powstania był wyraźnie lepszy od swojego poprzednika - FAT16, jednak od samego początku nie był planowany jako docelowy system plików, gdyż już wtedy istniał NTFS.

Katalog root i układ katalogów FHS

W systemie GNU/Linux podstawowa struktura katalogów jest dość ściśle określona wg FHS (Filesystem Hierarchy System). Przede wszystkim w katalogu głównym: /, (tzw. katalog "root"), jest tylko kilka katalogów, nie powinno umieszczać się w nim żadnych dodatkowych plików czy katalogów. Nie spowoduje to bynajmniej nieprawidłowego działania systemu, ale jest to ogólnie przyjęty i dość restrykcyjnie przestrzegany standard, dzięki czemu system katalogów jest przejrzysty.

W katalogu głównym poszczególne katalogi mają ściśle określone przeznaczenie:

Katalog	Rola
/bin	tutaj znajdują się binarne (wykonywalne) pliki najbardziej podstawowych narzędzi systemowych
/boot	tutaj znajdują się pliki niezbędne do uruchomienia systemu (kernel, initrd, pliki bootloadera - w przypadku GRUB)
/dev	znajdujące się tutaj pliki nie są faktycznie plikami na dysku, lecz odnoszą się do urządzeń - za ich pośrednictwem system komunikuje się z urządzeniami (komunikacja niskopoziomowa)
/etc	pliki konfiguracyjne, ustawienia systemowe
/home	w tym katalogu znajdują się katalogi domowe użytkowników
/lib	tutaj znajdują się systemowe biblioteki dzielone (shared libraries), zawierające funkcje, które są wykonywane przez wiele różnych programów
/media	zapewnia dostęp do nośników wyjmowanych (miejsce montowania nośników wymiennych) (np. pendrive, CD-ROM)
/mnt	tutaj natomiast są "montowane" dyski (w dystrybucjach takich jak Ubuntu, dyski są montowane w /media)
/proc	wirtualny katalog, zawierający wirtualne pliki umożliwiające dostęp do danych o kernelu
/root	tutaj znajduje się katalog domowy użytkownika root - głównego administratora każdego systemu uniksowego, który ma uprawnienia zupełne
/sbin	zawiera pliki wykonywalne poleceń, które mogą być wykonywane tylko przez administratora
/sys	pliki systemu
/tmp	systemowy folder przeznaczony na pliki tymczasowe
/usr	w tym katalogu są instalowane dodatkowe programy, które umożliwiają pracę zwykłemu użytkownikowi systemu
/var	katalog przeznaczony na pliki systemowe, ale których zawartość często się zmienia, jak np. logi programów/systemu

Polecenia do wykonania

1. Co to jest jądro systemu (kernel) i za co odpowiada?
2. Co to jest powłoka systemu (shell)?
3. Zapoznaj się (poszukaj informacji w Internecie) z następującymi powłokami systemu:
   a. `bash`
   b. `sh`
   c. `csh`
   d. `zsh`
4. Co to jest system plików (filesystem)?
5. Jakie systemy plików znasz, jakie systemy plików dostępne są w Linux? Czym różnią się systemy plików między sobą?
6. Zapoznaj się z drzewem plików i katalogów w systemie Linux. Do czego służą katalogi:
   a. `/boot`
   b. `/bin`
   c. `/etc`
   d. `/usr`
   e. `/sbin`
   f. `/lib`
   g. `/home`
   h. `/var`
   i. `/tmp`
   j. `/dev`
   k. `/mnt`
   l. `/proc`
7. Zapoznaj się z pojęciem Filesystem Hierarchy Standard
8. Co to jest katalog root `/`.

Tematy dodatkowe

1. Przeczytaj najnowszy FHS. Pozyskasz wiedze o dodatkowych elementach systemu Linux ponieważ wiele z nich ma swoje odwzorowanie właśnie w strukturze katalogów filesystemu.

Laboratorium 2. Podstawowe polecenia Linux (obsługa plików i katalogów)

Zagadnienia do laboratorium

W ramach zajęć zapoznamy się z następującymi poleceniami systemu Linux:

- `man` – program wyświetlający systemowy podręcznik dla innych poleceń; używa się go wpisując `man <nazwa programu>` (np. `man man`, wyświetli manual dla polecenia `man`)
- `ls` – wyświetla listę plików (zwróć uwagę na opcję `-a` oraz `-l`)
- `cat` – wyświetla zawartość pliku
- `rm` – usuwa plik (zwróć uwagę na opcję `-r` oraz `-f`)
- `mv` – przenosi oraz zmienia nazwę pliku/katalogu
- `mkdir` – tworzy katalog (zwróć uwagę na opcję `-p`)
- `ln` – tworzy link (skrót) do innego pliku (zwróć uwagę na opcję `-s`)
- `find` – wyszukuje pliki na dysku
- `basename` – wyciąga nazwę pliku ze ścieżki (przydatne w skryptach)
- `dirname` - wyciąga ścieżkę katalogów ze ścieżki do pliku (przydatne w skryptach)
- `echo` – wyświetla argumenty
- `less`, `more` – programy do paginacji treści w konsoli

Przed przystąpieniem do ćwiczeń opanuj posługiwanie się historią poleceń w bashu (strzałka w górę i w dół) oraz dopełnianiem nazw plików i katalogów za pomocą przycisku `[tab]`.

Polecenie Echo

Echo jest to prosty program, który umożliwia wyświetlenie argumentów polecenia na STDIN (polecenie echo odsyła ciąg znaków na standardowe wyjście).

Dostępne opcje

-n	nie odsyła końcowego znaku nowej linii
-e	włącza interpretacje znaków specjalnych w ciągu NAPIS
-E	wyłącza interpretację znaków specjalnych w ciągu NAPIS (domyślne)
--help	wypisuje komunikat pomocy i kończy pracę polecenia echo
--version	wypisuje informacje na temat wersji polecenia echo

Znaki specjalne z opcją -e

\0NNN	Dowolny znak o kodzie ASCII odpowiadającym liczbie ósemkowej NNN
****	pojedynczy znak \ (ukośnik wsteczny)
\a	dzwonek, alarm
\b	backspace
\c	pomiń znak nowej linii
\f	znak wysunięcia strony
\n	znak nowej linii
\r	znak powrotu karetki
\t	tabulacja pozioma
\v	tabulacja pionowa

Przykład użycia

```
$ echo "Linia przykladowego tekstu "; echo "Druga linia"
Linia przykladowego tekstu
Druga linia
$ echo -n "Linia przykladowego tekstu "; echo "Druga linia"
Linia przykladowego tekstu Druga linia
```

Polecenie ls

Wyświetla zawartość katalogu.

Polecenie cat

Wyświetla zawartość pliku.

`cat jeden.doc > wszystkie.doc` - nadpisanie zawartości w pliku wszystkie.doc zawartościami innych plików

`cat jeden.doc dwa.doc >> wszystkie.doc` - dopisanie do pliku wszystkie.doc zawartości innych plików

`cat nazwa_pliku` - czytanie zawartości pliku

`cat *.doc` - wyświetli np.: dwa.doc trzy.doc

`cat jeden.doc dwa.doc > wszystkie.doc` - wrzuca zawartość plików do jednego

`cat < jeden.doc > dwa.doc` - wypisze zawartość jeden.doc na ekran (standardowe wyjście) i nadpisze zawartość pliku jeden.doc do pliku dwa.doc

`cat parowka.doc | more` - wyjście programu `cat` (parowka.doc) jest przekazywane na wyjście programu more

Polecenie rm

`rm nazwa_pliku` - usuwa plik

`rm *` - usuwa wszystkie pliki z danego katalogu

`rm * -i` - usuwa wszystkie pliki z danego katalogu z potwierdzeniem

`rm * -f` - usuwa wszystkie pliki z danego katalogu i wyłącza potwierdzenia (nawet zabezpieczone przed usunięciem)

`rm -f` - usunięcie plików zabezpieczonych przed kopiowaniem

`rm -r` - usunięcie plików również w podkatalogach

`rm -rf` - usuwa wszystkie pliki bez pytania

Polecenie rmdir

`rmdir nazwa_katalogu` - usuwa katalog

Polecenie mv

`mv stara_nazwa nowa_nazwa` - zmiana nazwy pliku

`mv nazwa_pliku nazwa_katalogu` - przenoszenie pliku do katalogu

Polecenie mkdir

`mkdir nazwa_katalogu` - tworzy katalog

Polecenie ln

`ln źródło cel`– tworzenie linku (dowiązania). Polecenie to ma następujące opcje (-n):

`-d` - pozwól użytkownikowi uprzywilejowanemu (root) tworzenie dowiązań stałych (hardlinks) do katalogów.

`-f` - usuń istniejące pliki docelowe.

`-i` - pytaj czy nadpisywać istniejące pliki docelowe.

`-s` - twórz dowiązania symboliczne zamiast dowiązań twardych. Ta opcja wypisuje błędy na systemach nie wspierających dowiązań symbolicznych.

`-v` - wydrukuj nazwę każdego pliku przed utworzeniem dowiązania.

`ln -s /jakas/nazwa` - tworzy dowiązanie `./nazwa` wskazujące na `/jakas/nazwa`

`ln -s /jakas/nazwa mojanazwa` – tworzy dowiązanie `./mojanazwa` wskazujące na `/jakas/nazwa`

`ln -s a b` – tworzy dowiązania ../a i ../b wskazujące na ./a i ./b

Polecenie basename

Polecenie basename usuwa wiodące nazwy katalogów z nazwy, pozostawiając w ten sposób nazwę pliku bez ścieżki. Jeśli podano przyrostek i jest identyczny z końcem nazwy, to jest on także usuwany. basename wypisuje rezultat swojego działania na standardowe wyjście.

`basename sciezka_do_pliku [przyrostek]` – zwraca nazwę pliku do którego podano ścieżkę pomijając przyrostek

```
$ basename /home/ubuntu/katalog/plik.txt .txt
```

```
plik
```

Polecenie dirname

dirname wypisuje wszystkie, oprócz ostatniej, rozdzielone ukośnikami składowe nazwypliku. Otrzymujemy w ten sposób ścieżkę pliku. Jeśli nazwapliku jest pojedynczą składową, dirname wypisuje `.` (oznaczające bieżący katalog).

`dirname sciezka_do_pliku` – zwraca sciezke katalogu zawierającego dany plik

Polecenie pwd

Bezwzględna ścieżka do pliku.

Wildcards

Wieloznacznik (wildcard), symbol maski, znak globalny, metaznak, symbol wieloznaczny – nazwa symbolu stosowanego w informatyce w procedurach wyszukiwania ciągów znaków w dokumentach tekstowych i w zbiorach informacji o charakterze tekstowym.

`*` - wieloznacznik ogólny – zastępujący dowolną liczbę dowolnych znaków

`?` - wieloznacznik lokalny – zastępujący pojedyncze wystąpienie dowolnego znaku

Polecenia do wykonania i zagadnienia

1. Zapoznaj się z manualem każdego polecenia w tym laboratorium. W manualu można wyszukiwać wpisując znak `/` oraz fragment poszukiwanego tekstu, a następnie przycisk `[enter]`. Manual podświetli znalezione fragmenty, można przechodzić pomiędzy poprzednim i następnym fragmentem za pomocą przycisków `n` i `p` (next, previous).
2. W katalogu domowym `/home/student` utwórz następującą strukturę katalogów:

```
~/dokumenty
    praca/
       raporty/
    dom/
       zakupy/
    studia/
       podania/
```

3. Spróbuj wykonać powyższe zadanie używając tylko jednego polecenia zamiast sekwencji `mkdir`, `ls`, `cd`.
4. W katalogu `zakupy/` utwórz plik o nazwie `lista`. Zrób to za pomocą komendy:
   `echo mleko > lista`
5. Wyświetl zawartość pliku lista.
6. Zaobserwuj co się stanie jeśli wykonasz następnie polecenie `echo chleb > lista`
7. Usuń plik `lista` i przetestuj czym różni się poprzednia komenda to od użycia komend:
   `echo mleko >> lista`
   `echo chleb >> lista`
8. Usuń katalog `dom/` wraz z podkatalogami i plikami. Spróbuj wykonać to także tylko jednym poleceniem.
9. Do katalogu `raporty/` skopiuj plik `/etc/passwd` (który zawiera listę użytkowników w systemie). Plik w ścieżce docelowej powinien nazywać się `uzytkownicy`. Spróbuj skopiować plik jednocześnie nadając mu nowa nazwę (jedna komenda).
10. Zmniejsz okno terminala i spróbuj wyświetlić zawartość pliku `uzytkownicy`. Zawartość nie zmieściła się w jednym oknie. Spróbuj wykonać polecenia:

```
cat uzytkownicy | more

cat uzytkownicy | less
```

11. Spróbuj wyszukać za pomocą polecenia `find` wszystkie pliki w Twoim katalogu domowym zaczynające się na literę `d`. W tym celu użyj wildcard `*`.
12. W katalogu `studia/` utwórz katalog `programy/` w którym utwórz katalogi `program-1.0`, `program-1.1`, `program-1.3`. Zrób link symboliczny o nazwie `program` do katalogu z

najwyższą wersją (1.3). Spróbuj dodać plik do katalogu `program-1.3` i przekonaj się, że plik ten znajduje się także w katalogu `program`.

13. Sprawdź poleceniem `ls -al`, że plik `program` jest rzeczywiście linkiem symbolicznym i wskazuje na katalog `program-1.3`.
14. Na przykładzie pliku `użytkownicy` zaprezentuj działanie programów `pwd`, `dirname`, `basename`. Odczytaj pełną ścieżkę pliku za pomocą `pwd`, a następnie podaj ją jako argument do dwóch pozostałych programów.
15. Na końcu usuń cały katalog `dokumenty/`.

Tematy dodatkowe

1. Zainteresuj się komendą `find`. Oferuje wiele możliwości wyszukania, np. po dacie ostatniej modyfikacji, albo grupie pliku, a także uruchomienie dla każdego znalezionego pliku dowolnego polecenia.
2. Zainteresuj się czym różni się link twardy (hard link) od symbolicznego.

Laboratorium 3. Użytkownicy, grupy i prawa dostępu

Zagadnienia do laboratorium

System Linux wywodzi się z rodziny systemów unixowych, których przeznaczeniem było głównie działanie jako systemy serwerowe obsługujące bardzo wielu użytkowników. Z tego powodu w systemie Linux istnieje stosunkowo rozbudowany system zarządzania użytkownikami. Każdy użytkownik rozpoznawany jest po nazwie użytkownika oraz jednocześnie w systemie po specjalnym numerze `UID`, który jest unikalny dla niego. Dodatkowo użytkownicy mogą przypisani być do grup. Grupy także rozpoznawane są za pomocą nazwy oraz specjalnego unikalnego numeru `GID`. Grupa powinny reprezentować pewną wspólna cechę wielu użytkowników. Głównym użytkownikiem systemu jest `root` który ma zawsze `UID` równy `0`. Jest on także członkiem grupy `root`. Nazwy grup i użytkowników mogą się powtarzać, co jest w niektórych dystrybucjach domyślną polityką tworzenia grup dla użytkowników (nazwa grupy taka sama jak login).

Plik /etc/passwd

`passwd` jest plikiem tekstowym z jednym rekordem na linię, z których każda opisuje jedno konto użytkownika. Każdy rekord (linia) składa się z siedmiu pól oddzielonych dwukropkami. Kolejność rekordów w pliku jest zazwyczaj nieistotna.

Przykład: `jsmith:x:1001:1000:Joe Smith, pokój 1007, (234) 555-8910, (234) 555-0044, e-mail:/home/jsmith:/bin/sh`

Kolejne pola w rekordzie oznaczają:

1. Nazwa użytkownika
2. Drugie pole przechowuje informacje używana do sprawdzania użytkownika hasła, jednak w większości nowoczesnych zastosowań tej dziedzinie jest zwykle ustawiony na "x" (lub inny wskaźnik) z aktualnymi informacjami hasła przechowywane w oddzielnych haseł plików. Ustawienie tego pola na gwiazdkę "*" jest typowy sposób, aby wyłączyć konto, aby zapobiec jego użyciu.
3. Identyfikator użytkownika `UID`, numer, który system operacyjny używa do celów wewnętrznych.
4. Identyfikator grupy `GID`. Liczba ta określa podstawową grupę użytkownika, wszystkie pliki, które są tworzone przez użytkownika są początkowo dostępne dla tej grupy.
5. Piąte pole, zwane pola GECOS , jest komentarzem, który opisuje osoby lub konta. Zazwyczaj jest to zbiór wartości oddzielone przecinkami w tym pełnej nazwy użytkownika i dane kontaktowe.
6. Ścieżka do katalogu domowego użytkownika.
7. Domyślny shell. Program, który jest uruchamiany przy każdym zalogowaniu do systemu. Dla użytkownika interaktywnego, zazwyczaj jest to jeden z systemu tłumaczy linii komend np. bash (powłoki).

Plik /etc/group

`group` jest to plik, w którym przechowywane są informacje o grupach. Tak jak w przypadku pliku `passwd` jeden rekord stanowi jedna linia rozdzielana znakiem dwukropka.

Przykład: `cdrom:x:24:joe,admins,kate`

Poszczególne pola w rekordzie oznaczają:

1. Nazwa grupy
2. Pole hasła, przeważnie nie używane. Umożliwia tworzenie specjalnych uprzywilejowanych grup.
3. Identyfikator grupy `GID`.
4. Lista użytkowników grupy rozdzielona przecinkami. Wszyscy użytkownicy wymienieni w tym polu należą do danej grupy i zyskują jej uprawnienia.

Polecenie id

`id` - wyświetla nazwy bieżących ID użytkownika i grupy

`-a` - wyświetla zestaw grup w systemach, które obsługują równoczesne członkostwo w wielu grupach.

Zarządzanie użytkownikami

`useradd` – tworzenie nowego konta użytkownika

Argument	Znaczenie
-c komentarz	dodanie komentarza do pola komentarza w pliku haseł.
-d ktalog domowy	Ustawienie katalogu domowego dla nowego użytkownika, domyślnie odbywa się to poprzez dostawienie do domyślnego katalogu nazwy użytkownika.
-e data_wygaśnięcia	data, od której konto użytkownika zostanie zablokowane(wyłączone), datę należy podać w formacie YYYY-MM-DD, gdzie YYYY to rok, MM miesiąc w postaci dwucyfrowej tzn. np. maj to 05, dzień tak jak miesiąc również w postaci dwucyfrowej.
-f czas_nieaktywności	ustawienie liczby dni, po której konto ma być definitywnie wyłączone, podanie wartości 0 wyłączy konto zaraz po wygaśnięciu hasła, a wartość -1 wyłącza tą funkcję, wartość -1 jest wartością domyślną.
-g początkowa_grupa	numer lub nazwa początkowej grupy logowania użytkownika, grupa musi istnieć, domyślnym numerem grupy jest 1.
-G grupa[,...]	grupa lub lista grup których również ma należeć tworzony użytkownik, każda następna grupa powinna być oddzielona od poprzedniej przecinkiem, bez spacji pomiędzy. Użytkownik domyślnie należy tylko do grupy początkowej.
-m, -k katalog_z_profilem	ustawienie tej opcji spowoduje, że jeżeli katalog domowy użytkownika nie istnieje, to zostanie on utworzony. Jeśli ustawiona jest opcja -k, to z katalogu wpisanego jako wartość zostaną przekopiowane wzorcowe pliki startowe, w przeciwnym wypadku jako wzorzec posłuży katalog `/etc/skel`.
-s powłoka	ustawienie powłoki systemowej użytkownika, domyślnie wybierana jest domyślna powłoka systemowa
-u id_użytkownika	podanie numerycznej wartości identyfikatora użytkownika (uid). Numer ten musi być dodatni, unikatowy(chyba, że zostanie użyta opcja `-o`. Domyślnie ustawiana jest wartość najmniejsza począwszy od wartości 100, wartości od 0 do 99 przeznaczone są dla kont systemowych.
-p zakodowane_hasło	tutaj należy podać hasło w formie zakodowanej, np. takie jak utworzone poleceniem crypt
nazwa_użytkownika	tutaj należy podać nazwę nowego konta dla użytkownika.

Opis opcji polecenia `useradd` z opcją -D

Argument	Znaczenie
-D grupa_domyślna	numer lub nazwa początkowej grupy użytkownika, grupa musi istnieć.
-b katalog domyślny	ustawienie ścieżki do katalogu, w którym przechowywane są domyślne katalogi użytkowników.
-f domyślny_czas_nieaktywności	tutaj można zmienić domyślny czas nieaktywności dla wygaśnięcia hasła dla konta użytkowników.
-e domyślna_data_wygaśnięcia	ustawienie domyślnej daty wygaśnięcia, od której konto użytkownika jest wyłączane.
-s domyślna_powłoka	ścieżka z nazwą domyślnej powłoki systemowej dla użytkowników.

`usermod` - polecenie służące do modyfikowania kont użytkowników:

Argument	Znaczenie
-A metoda_dostępu DEFAULT	metoda autentykacji użytkownika, nie wybranie żadnej metody skutkuje wybraniem domyślnej.
-c komentarz	nowa wartość dla pola komentarza w pliku haseł.
-d katalog_domowy	ustawienie nowego katalogu domowego dla użytkownika, jeżeli zostanie użyta opcja `-m` to zawartość aktualnego katalogu domowego zostanie przeniesiona do nowego.
-m	ustawienie tej opcji spowoduje, że katalog domowy użytkownika zostanie przeniesiony do nowo utworzonego.
-e data_wygaśnięcia	data, od której konto użytkownika zostanie zablokowane(wyłączone), datę należy podać w formacie YYYY-MM-DD, gdzie YYYY to rok, MM miesiąc w postaci dwucyfrowej tzn. np. maj to 05, dzień tak jak miesiąc również w postaci dwucyfrowej.
-f czas_nieaktywności	ustawienie liczby dni, po której konto ma być definitywnie wyłączone, podanie wartości 0 wyłączy konto zaraz po wygaśnięciu hasła, a wartość -1 wyłącza tą funkcję, wartość -1 jest wartością domyślną.
-g początkowa_grupa	numer lub nazwa początkowej grupy logowania użytkownika, grupa musi istnieć, domyślnym numerem grupy jest 1.
-G grupa[,...]	grupa lub lista grup, do których ma dodatkowo należeć użytkownik, każda następna grupa powinna być oddzielona od poprzedniej przecinkiem, bez spacji pomiędzy. Użytkownik domyślnie należy tylko do grupy początkowej.
-s powłoka	ustawienie powłoki systemowej użytkownika, domyślnie wybierana jest domyślna powłoka systemowa.
-u id_użytkownika	podanie numerycznej wartości identyfikatora użytkownika (uid). Numer ten musi być dodatni, unikatowy(chyba, że zostanie użyta opcja `-o`. Domyślnie ustawiana jest wartość najmniejsza począwszy od wartości 100, wartości od 0 do 99 przeznaczone są dla kont systemowych.
-p zakodowane_hasło	tutaj należy podać hasło w formie zakodowanej.

-L	użycie tej opcji zablokuje hasło konta użytkownika.
-U	odblokowanie hasła konta użytkownika.
-l nowa_nazwa_użytkownika	stara nazwa użytkownika zostanie zamieniona na nową, tutaj należy również pomyśleć o zmianie nazwy katalogu domowego
nazwa_użytkownika	tutaj należy podać nazwę nowego konta dla użytkownika.

`userdell` – polecenie usuwa istniejące konto użytkownika z systemu

Argument	Znaczenie
-r	ustawienie tej opcji spowoduje usunięcie katalogu domowego użytkownika.
-f	opcja wymuszenia usunięcia konta użytkownika zalogowanego.
-p zachowaj_wartość	zachowanie użytkownika w pliku `/etc/passwd`, usunięcie odbywa się poprzez ustawienie hasła "niemożliwego", jest to robione, aby można było skorzystać z konta użtkownika.
-u	wyświetlenie informacji o tym jak użyć narzędzia.
-v (OpenSUSE)	użycie opcji wyświetli wersję polecenia.
-v (UNIX)	wyświetlanie informacji zwrotnej polecenia na konsolę.
-P ścieżka	Ustawienie powłoki systemowej żytkownika, domyślnie wybierana jest domyślna powłoka systemowa.
-S	pozwolenie na usunięcie również konta użytkownika z Samba.
-D binddn	Użycie "Distinguished Name binddn" do podłączenia do katalogu LDAP.
nazwa_użytkownika	tutaj należy podać nazwę konta użytkownika do usunięcia.

`passwd` – polecenie ustawia hasło dla użytkownika.

Prawa dostępu plików

Każdy plik i katalog w systemie Linux posiada prawa dostępu. Jest to przypisana do niego informacja kto i do jakich czynności posiada uprawnienia. Wyróżnia się trzy poziomy dostępu:

1. user- jako użytkownik,
2. group - jako członek grupy, do której należy użytkownik,
3. other - pozostali.

Prawa rwx

Dla każdej z tych kategorii możliwe są trzy rodzaje praw dostępu, opisane trzema bitami:

1. odczytywać zawartość pliku/katalogu - read (odczyt 100 (2) = 4 (8))
2. zapisywać do pliku/katalogu - write (zapis 010 (2) = 2 (8))
3. wykonywać/otwierać plik katalog - execute (wykonywanie 001 (2) = 1(8))

Standardowym symbolicznym sposobem zapisu uprawnień jest:

```
-rwxr-xr—1 fizyk fizyk 0 kwi 16 13:09 plik
tuuugggooo
```

- `t` - oznacza typ pliku (`-` zwykły, `d` katalog, `l` dowiązanie symboliczne, `s` gniazdo, `f` FIFO, `c` urządzenie znakowe, `b` urządzenie blokowe)
- `u` - uprawnienia właściciela
- `g` - uprawnienia grupy
- `-` uprawnienia pozostałych

Sticky bit - w przypadku pliku wykonywalnego powoduje wymuszenie przechowywania jego kodu w pamięci (ustawianie Sticky na plikach wykonywalnych jest obecnie rzadko stosowane); w przypadku Sticky ustawionego na katalogu umożliwia kasowanie/zmianę nazwy elementów w nim zawartych tylko przez właściciela "Sticky-katalogu", właściciela pliku/katalogu w nim zawartego lub roota (w normalnym przypadku wewnątrz katalogu każdy może zmieniać nazwę i usuwać elementy o ile podstawowe uprawnienia to dopuszczają)

Sposoby reprezentacji praw dostępu

Istnieje kilka sposobów zapisu praw do danego pliku. Najpopularniejszymi są: system numeryczny, oraz literowy. Numerycznie `chmod` przyjmuje odpowiednią wartość potęgi dwójki dla każdego typu akcji (zapisu, odczytu, uruchomienia).

Typ zapisu	Prawo odczytu	Prawo zapisu	Prawo uruchomienia	Specjalne prawo uruchomienia	UID/GID	sticky bit
BINARNY	2^2	2^1	2^0			
LICZBOWY	4	2	1			
SYMBOLICZNY	r (*read*)	w (*write*)	x (*execute*)	X	s	t

Polecenie chmod

Aby zapisać uprawnienia w systemie numerycznym należy dodać liczby odpowiadające uprawnieniom, które chcemy przyznać. Należy tak postąpić osobno dla właściciela, grupy oraz innych, np. właściciel - wszystkie prawa, grupa - odczyt, inni - brak należy zapisać jako:

```
chmod 740 nazwa_pliku
```

Tabela z interpretacją kodów ósemkowych

Cyfra	Prawa	Litera
0	Brak praw	
1	Wykonywanie	x
2	Pisanie	w
3	Wykonywanie i pisanie	wx
4	Czytanie	r
5	Czytanie i wykonywanie	rx
6	Czytanie i pisanie	rw
7	Czytanie, pisanie i wykonywanie	rwx

Aby zapisać uprawnienia w systemie znakowym należy wpisać znak `u`, `g`, `o` lub `a` a następnie znak:

- `+` jeżeli chcemy dodać uprawnienia,
- `-` jeżeli chcemy odebrać uprawnienia,
- `=` jeżeli chcemy zmienić uprawnienia (tzn. przypisać takie, jakie podamy),

a następnie odpowiednie oznaczenia z tabeli. Po przecinku można dopisać kolejne uprawnienia np.

```
chmod u=rwx,g+rw,o-r nazwap_pliku
```

wprowadza następujące zmiany:

- dla właściciela - ustawiono odczyt, zapis oraz wykonanie, ale to ostatnie tylko wtedy jeżeli ono już jest ustawione, w innych plikach pozostanie wyłączone,
- dla grupy - dodano odczyt i zapis,
- dla innych - odebrano odczyt.)

Polecenie chown

`chown` to polecenie systemu Unix i pochodnych używane do zmiany właściciela pliku. W większości implementacji może być wykonywane tylko przez administratora systemu.

Polecenie `chown` wywoływane jest w następujący sposób:

```
chown [użytkownik][:grupa] plik-1 [plik-2 ...]
```

gdzie:

- parametr użytkownik określa nowego właściciela pliku,
- parametr grupa (koniecznie poprzedzony dwukropkiem) określa grupę, do której plik ma zostać przypisany,
- parametry plik-n określają jeden lub więcej plików, których dotyczy zmiana.

Uwagi:

- Co najmniej jeden z parametrów użytkownik lub grupa musi zostać określony.
- Zarówno użytkownik jak i grupa mogą zostać określone poprzez nazwę symboliczną lub identyfikator liczbowy.

Polecenie chgrp

to komenda systemu Unix i pochodnych, umożliwiająca zwykłym użytkownikom zmianę przypisania pliku do grupy. W przeciwieństwie do polecenia `chown`, `chgrp` pozwala na przypisanie pliku tylko do takiej grupy, do której użytkownik sam należy.

`chgrp` wywoływane jest w następujący sposób:

```
chgrp [opcje ...] grupa plik1 [plik2 ...]
```

gdzie:

- parametr opcje określa opcje używane przy przypisywaniu,
- parametr grupa określa nazwę nowej grupy, do której plik ma być przypisany,
- parametry `plik1 [plik2 ...]` określają listę plików, których przypisanie ma być zmienione.

Parametr grupa może być podany jako nazwa symboliczna (jak w poniższym przykładzie), lub identyfikator liczbowy.

Różnica prawa x w odniesieniu do katalogu i pliku

Znaczenie praw dostępu dla zwykłych plików jest intuicyjne, natomiast dla katalogów znaczenie jest następujące:

- jeżeli użytkownik ma prawo x do katalogu, to może do niego "wejść",
- jeżeli użytkownik ma prawo r do katalogu, to może wyświetlić jego zawartość ,
- jeżeli użytkownik ma prawo w do katalogu, to może w nim tworzyć i kasować pliki/katalogi.

Polecenia do wykonania i zagadnienia

1. Zapoznaj się z ideą wirtualizacji oraz narzędziem VirtualBox dostępnym na komputerach w laboratorium.
2. Pobierz, rozpakuj i uruchom w VirtualBox obraz systemu Ubuntu z adresu: http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/ubuntu/ubuntu-9.10.vdi.7z
3. Uzyskaj prawa użytkownika root wykonując np. polecenie `sudo bash`. Hasło użytkownika znajdziesz w pliku http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/ubuntu/README, sprawdź czy poleceniem `whoami` czy posiadzasz dostęp jako użytkownik root.
4. Utwórz w systemie użytkowników : `marek`, `ania`, `jurek`.
5. Sprawdź jak zmienił się plik `/etc/passwd`.
6. Utwórz w systemie grupy: `marketing`, `zarzad`.
7. Sprawdź jak zmienił się plik `/etc/groups`.
8. Dodaj do grupy `marketing` dodaj użytkowników `marek` i `ania` , a do grupy `zarzad` tylko użytkownika `ania`.
9. Jako użytkownik `marek` utwórz katalog `~/projekt` a w nim plik tekstowy z dowolną treścią o nazwie `~/projekt/zalozenia`.
10. Ustaw takie prawa dostępu do tego pliku aby mogły go odczytać wszystkie osoby ale edytować tylko z grupy `marketing`. (to znaczy sprawdź, że logując się jako `marek` lub `ania` możesz wyświetlić plik i go zmodyfikować, a jako `jurek` możesz tylko wyświetlić).
11. Ustaw takie prawa dla katalogu projekt, aby wyświetlić jego zawartość mogła tylko osoba z grupy `zarzad` (czyli tylko `ania`). Sprawdź czy jako `marek` znając pełną ścieżkę pliku nadal możesz edytować plik `~/projekt/zalozenia`.
12. Ustaw takie prawa do katalogu projekt, aby wejść do katalogu mogła tylko osoba z grupy `zarzad`. Sprawdź czy jako `marek` nadal możesz edytować plik?
13. Dodaj kolejne pliki i katalogi do katalogu projekt (przynajmniej 2 poziomy). Spróbuj ustawić dowolne prawa dostępu rekurencyjnie wszystkim plikom i podkatalogom dla `~/projekt`. Np. prawa 666.
14. Usuń grupy i użytkowników w systemie.

Laboratorium 4. Procesy, deamony i sygnały

Podczas zajęć omówione zostaną komendy związane przeglądaniem stanu systemu.

Proces w systemie Linux jest to uruchomiona instancja programu binarnego. Taka uruchomiona instancja zajmuje pewne zasoby w pamięci , system operacyjny zarządza wskaźnikiem instrukcji i wykonaniem kolejnych kroków instrukcji procesu. Procesy posiadają specjalny unikatowy numer w systemie w skrócie PID. Numer ten przyznawany jest z reużytkowalnej puli, ale w danym momencie nie ma dwóch procesów o tym samym PID. Każdy proces uruchomiony jest z poziomu konkretnego użytkownika i dysponuje prawami takimi samymi jak ten użytkownik. Najważniejszym procesem w systemie jest `init` który ma zawsze PID `0` a jego właścicielem jest użytkownik root.

Procesy a deamony

Demony, inaczej usługi, są pewnym rodzajem programów, które są uruchamiane zazwyczaj przy starcie systemu działając najczęściej cały czas w tle. Procesy tych usług posiadają w Linux specjalne określenie: deamon. W praktyce wiele programów umozliwia uruchomienie ich w trybie deamona, a typowymi usługami działającymi w ten sposób są: serwery WWW (np. apache, nginx), DNS (np. bind), mail (pocztowe), X server, baza danych (np. mysql, postgresql), deamon usług sieciowych i DHCP, itp… Procesy deamonów będące usługami każdej chwili mogą przyjąć od innego programu polecenie (żądanie) w celu ich obsłużenia. Demonem jest np. serwer http, który cały czas jest uruchomiony w tle, a w momencie gdy przychodzi żądanie od przeglądarki wysyła odpowiedni plik, lub wykonuje jakąś akcję. Demon może również obsługiwać programy działające na tym samym komputerze.

Kolejną cechą charakterystyczną demonów, jest to że nowoczesne dystrybucje Linux najczęściej posiadają wbudowane podsystemy do zarządzania uruchamianiem tych procesów. Np. dbają one o to aby przy każdym starcie systemu procesy deamonów były automatycznie uruchomione. Często również do obsługi procesów deamonów używa się różnego rodzaju aplikacji monitorujących stan ich procesów po to aby np. gdy w deamon zakończy się z błędem, był on znowu uruchomiony i dzięki temu było dalej możliwe łączenie się z daną usługą. Do zatrzymania działania demona służą specjalne skrypty, które w popularnych dystrybucjach (Debian, Ubuntu) znajdują się zazwyczaj w katalogu `/etc/init.d/`

Skrypty te mogą być uruchamiane z następującymi opcjami:

- start - uruchamia demona
- stop - zatrzymuje demona
- restart - zatrzymuje, i ponownie uruchamia demona; przydatne np. w przypadku gdy zmieniona została konfiguracja danego demona

Niektóre mogą przyjmować również inne opcje, jednak te trzy są standardowe.

Polecenie ps

Wypisuje listę procesów w systemie

ps -l - dokładniejsze wypisanie stanów procesów
ps aux – dokładne listowanie wszystkich procesów w systemie wraz z argumentem wywołania

Polecenie top

`top [options]` - wyświetl najważniejsze procesy

`-d` - Określa opóźnienie między odświeżeniami ekranu. Można to zmieniać komendą interakcyjną `s`.

`-p` -Monitoruje jedynie procesy o danym id procesu.

`-q` - Powoduje to, że `top` odświeża się bez opóźnienia. Jeśli wywołujący jest superużytkownikiem, top działa z najwyższym możliwym priorytetem.

`-S` - Określa tryb kumulacyjny, gdzie każdy proces jest wypisywany z czasem CPU, który spożytkowanym przez niego *oraz jego martwe procesy potomne*.

`-s` -Nakazuje programowi `top` pracę w trybie bezpiecznym. Wyłącza potencjalnie niebezpieczne komendy interakcyjne (patrz niżej).

`-i` - Uruchamia **top**a, ignorując wszelkie procesy duchy i procesy próżnujące. Zobacz też komendę interakcyjną **i** poniżej.

`-C` - wyświetla oprócz indywidulnych stanów CPU również łączne stany CPU. Opcja ta ma znaczenie jedynie w systemach SMP.

`-c` - wyświetla linię poleceń zamiast samej nazwy polecenia. Domyślne zachowanie zostało zmienione, gdyż wydaje się to bardziej przydatne.

`-n` - Liczba iteracji. Odświeża wyświetlacz tyle razy i zakończ działanie.

`-b` - Tryb wsadowy. Przydatne do wysyłania wyjścia z top do innych programów lub do pliku. W trybie tym `top` nie przyjmuje wejścia z linii poleceń. Działa dopóki nie wykona określonej za pomocą opcji `n` liczby iteracji lub dopóki nie zostanie zabity.

Plik `proc/cpuinfo`

`cat /proc/cpuinfo` – wyświetla informacje na temat procesora

Plik `/proc/meminfo`

`cat /proc/meminfo` – wyświetla informacje na temat pamięci w systemie

Sygnały

(na podstawie/źródło: http://students.mimuw.edu.pl/SO/LabLinux/PROCESY/PODTEMAT_3/sygnaly.html)

Procesy komunikują się z jadrem systemu, a także między sobą aby koordynować swoją działalność. Linux wspiera kilka mechanizmów komunikacji zwanych IPC (Inter-Process Communication mechanisms). Jednym z nich są sygnały, zwane inaczej przerwaniami programowymi.

Sygnały mogą być generowane bezpośrednio przez użytkownika (funkcja `kill()`), może wysyłać je jądro oraz procesy między sobą (funkcja systemowa `kill()`) . Dodatkowo pewne znaki z terminala powodują wygenerowanie sygnałów. Na przykład na każdym terminalu

istnieje tak zwany znak przerwania (ang. interrupt character) i znak zakończenia (ang. quit character). Znak przerwania (zazwyczaj `Ctrl-C` lub `Delete`) służy do zakończenia bieżącego procesu (wygenerowani `SIGINT`). Wygenerowanie znaku zakończenia (zazwyczaj `Ctrl-\`) powoduje wysłanie sygnału `SIGQUIT` powodującego zakończenie wykonywania bieżącego procesu z zapisaniem obrazu pamięci.

Istnieją oczywiście pewne ograniczenia - proces może wysyłać je tylko do procesów mających tego samego właściciela oraz z tej samej grupy (te same UID i GID). Bez ograniczeń może to czynić jedynie jądro i administrator. Jedynym procesem, który nie odbiera sygnałów jest `init` (PID równy 1).

Sygnały są mechanizmem asynchronicznym - proces nie wie z góry kiedy sygnał może nadejść i głównym ich zadaniem jest informowanie procesu o zaistnieniu w systemie wyjątkowej sytuacji (np.: spadek napięcia w sieci). Ponadto są wykorzystywane przez shelle do kontroli pracy swoich procesów potomnych.

Każdy z sygnałów posiada swoje znaczenie (określające w jakiej sytuacji powinien być wysłany) jak również związana jest z nim pewna domyślna akcja jaką system wykonuje przy jego obsłudze. Oto opis znaczenia poszczególnych sygnałów według numeracji w systemie Linux.

(1) SIGHUP - Zerwanie łączności - sygnał zawieszenia (ang. hangup)

Jest wysyłany przy zerwaniu łączności z terminalem (np.: terminal jest zamykany) do wszystkich procesów, dla których jest on terminalem sterującym. Generowany również do wszystkich procesów grupie, gdy zakończy działanie jej przywódca (symulowane zerwania łączności z terminalami, których nie można odłączyć fizycznie, np.: komputerów osobistych). Domyślnie powoduje zakończenie procesu.

(2) SIGINT - Znak przerwania (ang. interrupt)

Jest zazwyczaj wysyłany do wszystkich procesów związanych z danym terminalem, gdy użytkownik naciśnie klawisz przerwania. Można znieść takie działanie klawisza przerwania, a także można zmienić sam klawisz, wywołując funkcję systemową `ioctl()`. Domyślnie powoduje zakończenie procesu.

(3) SIGQUIT - Znak zakończenia

Generowany zazwyczaj wtedy, gdy użytkownik naciśnie na terminalu klawisz zakończenia pracy. Jest bardzo podobny do sygnału SIGQUIT, ale dodatkowo powoduje wygenerowanie obrazu pamięci. Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(4) SIGILL - Niedozwolony rozkaz

Sygnał ten jest wysyłany po wystąpieniu wykrywanej sprzętowo sytuacji wyjątkowej, spowodowanej przez niewłaściwą implementację systemu. Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(5) SIGTRAP - Pułapka śledzenia

Sygnału tego używa się w funkcji systemowej `ptrace()`, zezwalającej na wykonywanie procesu z równoczesnym tworzeniem jego śladu. Z własności tej korzysta się w programach diagnostycznych, wspomagających testowanie programów. Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(6) SIGIOT,SIGABRT - Błąd sprzętowy (rozkaz IOT)

Sygnał ten jest generowany po wystąpieniu błędu sprzętowego spowodowanego niewłaściwą implementacją systemu. Ponadto wersja funkcji **abort()** w Systemie V oraz systemie 4.3BSD wysyła ten sygnał do bieżącego procesu (jest to przypadek wysyłania sygnału przez proces do samego siebie). Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(7) SIGBUS - Błąd szyny

Sygnał ten jest generowany po wystąpieniu błędu sprzętowego spowodowanego przez niewłaściwą implementację systemu. Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(8) SIGFPE - Błąd sprzętowy (rozkaz FBE)

Sygnał wysyłany po wystąpieniu wykrywanej sprzętowo sytuacji wyjątkowej, spowodowanej przez niewłaściwą implementację systemu. Na przykład system 4.3BSD dla maszyny VAX używa tego sygnału do wskazywania sytuacji wyjątkowych podczas wykonywania operacji zmiennopozycyjnych (np. niedomiar zmiennopozycyjny) i sytuacji wyjątkowych podczas wykonywania operacji na liczbach całkowitych (np. dzielenie przez zero). Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(9) SIGKILL - Zakończenie procesu

Jest to jedyny całkowicie pewny sposób zakończenia wykonywania procesu, ponieważ proces odbierający ten sygnał nie może go ani zignorować, ani przechwycić (dostarczając własną funkcję obsługi sygnału). Sygnału tego powinno sie używać tylko w sytuacjach wyjątkowych, w pozostałych natomiast zalecany jest sygnał SIGTERM.

(10) SIGUSR1 - Sygnał definiowany przez użytkownika

Jeden z dwóch sygnałów definiowanych przez użytkownika (drugim jest SIGUSR2). Można próbować użycia tego sygnału w programach użytkowych w celu komunikacji między procesami nie jest to jednak zbyt wygodny mechanizm (patrz w dalszej części ogólnego opisu). Domyślnie powoduje zakończenie procesu.

(11) SIGSEGV - Naruszenie segmentacji

Sygnał jest generowany po wystąpieniu błędu sprzętowego spowodowanego niewłaściwą implementacją systemu. Pojawia się na ogół wtedy, kiedy proces odwoła się do takiego adresu w pamięci, do którego nie ma dostępu. Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

(12) SIGUSR2 - Definiowany przez użytkownika

Drugi z sygnałów pozostawiony do zdefiniowania przez użytkownika - analogiczny do SIGUSR1. Domyślnie powoduje zakończenie procesu.

(13) SIGPIPE - Dane nie są odbierane z łącza komunikacyjnego

Kiedy proces wysyła dane do łącza komunikacyjnego lub kolejki FIFO, a nie istnieje proces odczytujący te dane, wtedy do procesu piszącego wysyłany jest właśnie ten sygnał. Ponadto jest sygnał jest generowany w systemie 4.3BSD, gdy proces wysyła dane do odłączonego gniazda. Domyślnie powoduje zakończenie procesu.

(14) SIGALRM - Budzik

Proces może nastawić budzik używając funkcji systemowej `alarm()` określającej kiedy proces ma być obudzony (wtedy jadro wyśle mu ten właśnie sygnał). Z sygnałem tym związana jest także funkcja `alarm()`, która sama nastawia budzik i oczekuje na nadejście tego sygnału by go następnie przechwycić. Domyślnie powoduje zakończenie procesu.

(15) SIGTERM - Programowe zakończenie procesu

Jest to standardowy sygnał zakończenia procesu, generowany przez oprogramowanie, wysyłany domyślnie do procesu wtedy, kiedy użyje się polecenia kill. Sygnału tego używa się także podczas wyłączania całego systemu w celu zakończenia wszystkich aktywnych procesów. Programy powinny przyjmować domyślną obsługę tego sygnału, bądź też błyskawicznie robić po sobie porządek (np. usunąć pliki tymczasowe) i wywoływać funkcję `exit()`. Domyślnie powoduje zakończenie procesu.

(16) SIGSTKFLT

W Linuxie nie zaimplementowany.

(17) SIGCHLD - Zakończenie procesu potomnego

Sygnał ten wysyłany jest do procesu macierzystego wtedy, gdy zakończy się działanie jego procesu potomnego. W systemach 4.3BSD i Linux sygnał ten wskazuje również, iż zmienił się stan procesu potomnego. Jest to ogólniejsze niż tylko wskazywanie śmierci potomka. Zmiana stanu może oznaczać jego śmierć, ale może także wynikać z zatrzymania tego procesu przez sygnały SIGSTOP, SIGTTIN, SIGTTOU lub też SIGTSTP. Domyślnie ignorowany.

(18) SIGCONT - Wznowienie wstrzymanego procesu

Sygnał ten wysyłany jest, kiedy w systemie 4.3BSD podejmuje się dalszy ciąg działania procesu, który został wstrzymany (patrz opis sygnałów SIGSTOP i SIGTSTP). Jeśli np. zatrzymano pracę edytora pełnoekranowego, to przed jej wznowieniem może on odtworzyć wyświetlany uprzednio obraz. W Linuxie domyślnie ignorowany.

(19) SIGSTOP - Zatrzymanie procesu

Sygnał ten zatrzymuje proces. Podobnie jak sygnału SIGKILL, nie można go zignorować, ani przechwycić (obsłużyć za pomocą własnej funkcji). Umożliwia on administratorowi systemu zatrzymanie procesu. Domyślnie powoduje zatrzymanie procesu.

(20) SIGTSTP - Wprowadzenie znaku zatrzymania

W systemie 4.3BSD sygnał ten jest wysyłany do procesu po naciśnięciu klawisza zawieszenia (na ogół Ctrl-Z) lub klawisza zawieszenia z opóźnieniem (na ogół Ctrl-Y). Domyślnie powoduje zatrzymanie procesu.

(21) SIGTTIN - Czytanie w tle z terminala sterującego

W systemie 4.3BSD sygnał ten jest wysyłany wtedy, gdy proces drugoplanowy przystępuje do czytania ze swego terminala sterującego. System generuje ten sygnał, aby uniknąć zamieszania spowodowanego tym, że więcej niż jeden proces czyta z tego samego urządzenia. Domyślnie powoduje zatrzymanie procesu.

(22) SIGTTOU - Pisanie w tle na terminalu sterującym

Sygnał zbliżony do sygnału SIGTTIN, wysyłany wtedy, gdy proces drugoplanowy przystępuje do pisania na swoim terminalu sterującym. W systemie 4.3BSD jest to działanie dozwolone, więc jest tam domyślnie ignorowany. W Linuxie domyślnie powoduje zatrzymanie procesu.

(23) SIGURG - Dane wysokopriorytetowe w gnieździe

W systemie 4.3BSD sygnał ten wysyłany jest po powstaniu sytuacji wyjątkowej związanej z nadejściem danych wysokopriorytetowych do gniazda albo z obecnością informacji o stanie sterowania, która trzeba odebrać od części nadrzędnej pseudo terminala pracującego w trybie pakietowym. Sygnał jest wysyłany do grupy procesów dla tego gniazda. W Linuxie domyślnie ignorowany.

(24) SIGXCPU - Przekroczenie czasu procesora

W systemie 4.3BSD oraz w Linuxie można w procesie ustalić ograniczenia zasobów, z których będzie korzystał dany proces i każdy utworzony przez niego proces. Sygnał SIGXCPU wskazuje, że proces przekroczył przydzielony mu czas procesora. Podobna jest idea sygnału SIGXFSZ. Domyślnie powoduje zakończenie procesu.

(25) SIGXFSZ - Przekroczenie rozmiaru pliku

Oznacza, ze proces przekroczył ograniczenie rozmiaru swoich plików (patrz opis sygnału SIGXCPU). Domyślnie powoduje zakończenie procesu.

(26) SIGVTALRM - Budzik zegara wirtualnego

W systemie 4.3BSD i w Linuxie istnieją trzy rodzaje mechanizmów zegarowych służących do sterowania budzikami: zegar związany z sygnałem SIGALRM odmierza czas rzeczywisty dla procesu; zegar wirtualny związany z sygnałem SIGVTALRM odmierza czas wirtualny procesu (czas wykonywania procesu); wreszcie zegar profilometra związany z sygnałem SIGPROF odmierza zarówno czas wirtualny procesu, jak i czas, w którym jądro pracuje na rzecz procesu. Domyślnie powoduje zakończenie procesu.

(27) SIGPROF - Budzik profilometra

Patrz opis SIGVTALRM. Sygnału SIGPROF używa się w interpretatorach do profilowania wykonania interpretowanego programu. Domyślnie powoduje zakończenie procesu.

(28) SIGWINCH - Zmiana rozmiaru okna

Sygnał oznacza, że zmienił się rozmiar okna na terminalu. Domyślnie ignorowany.

(29) SIGIO - Wejście-wyjście dozwolone dla deskryptora pliku

W systemie 4.3BSD sygnał ten wskazuje, że można wykonywać operacje wejścia-wyjścia w odniesieniu do deskryptora pliku. Sygnału tego używa się do stworzenia pewnej postaci asynchronicznego wejścia-wyjścia dla procesu.

SIGPOLL - Zdarzenie dotyczące urządzenia strumieniowego

Sygnał ten w systemie V umożliwia procesowi wykonywanie asynchronicznych operacji wejścia-wyjścia na jednym lub większej liczbie urządzeń strumieniowych. W Linuxie o oba te sygnały są utożsamiane i domyślnie ignorowane.

(30) SIGPWR - Niedobór mocy

Charakterystyczny dla Systemu V. Dokładne znaczenie tego sygnału zależy od konkretnej implementacji. Jedną z możliwości polega na wysyłaniu go po wykryciu znacznego spadku napięcia w sieci elektrycznej (np. napięcie spadło o 100V i nadal spada). Proces ma wówczas bardzo mało czasu na kontynuacje pracy - powinien więc zrobić porządek i zakończyć się. Domyślnie powoduje zakończenie procesu.

(31) SIGUNUSED

W Linuxie nie zaimplementowany.

Polecenie kill

Polecenie służy do wysyłania sygnałów do procesów. Przykłady użycia:

`kill PID`

`kill -9 PID` wysyła sygnał SIGKILL do procesu

Polecenie killall

Polecenie służy do wysyłania sygnałów do wielu procesów o tej samej nazwie.

Polecenia do wykonania i zagadnienia

1. Sprawdź jaki procesor znajduje się w komputerze w laboratorium oraz na serwerze wierzba. Zauważ, że dodatkowe rdzenie procesorów są reprezentowane jako kolejne procesory na liście.
2. Sprawdź ile pamięci RAM dysponuje komputer w laboratorium i serwer wierzba. Wartości podaj w MB oraz w GB dokonując odpowiednich przeliczeń.
3. Sprawdź jakie procesy zostały uruchomione w bieżącej sesji shella.
4. Sprawdź jakie procesy zostały uruchomione w całym systemie.
5. Sprawdź jakie procesy zostały uruchomione przez twojego użytkownika.
6. Uruchom polecenie top i posortuj listę procesów po ilości czasu procesora (TIME), po obciążeniu procesora (%CPU), po zajętej pamięci (VSIZE). Znajdź proces, który najbardziej obciąża procesor oraz proces który najbardziej obciąża pamięć. W tym celu możesz uruchomić dowolne programy na komputerze w laboratorium.
7. Utwórz prosty skrypt którym będziesz testować działanie programów zarządzających procesami. W tym celu utwórz ulubiony edytor tekstu i przepisz poniższy kod:

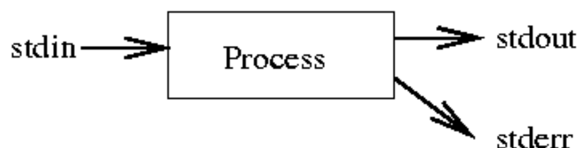
   ```
   #!/bin/bash
   echo „Zasypiam na $1 sekund”
   sleep $1
   echo „Pobudka. Koniec”
   ```

   Skrypt zapisz pod nazwą `spioch.sh`. Nadaj uprawnienia temu plikowi w taki sposób abyś mógł/mogła uruchomić go jako zalogowany użytkownik w systemie (dodaj uprawnienia executable).
8. Będąc w tym samym katalogu co plik `spioch.sh` spróbuj uruchomić skrypt w następujący sposób:
   `./spioch.sh 3`
   Skrypt powinien zasnąć na 3 sekundy, a następnie powinien pojawić się napis `Pobudka. Koniec`

9. Uruchom skrypt na wiele sekund (np. 300). Postaraj się zakończyć go przed czasem wciskając kombinację `Ctrl+C`. Jaki sygnał został wysłany do procesu? Czy napis `Pobudka. Koniec` wyświetli się na ekranie?
10. Uruchom ponownie skrypt na wiele sekund. Otwórz nową kartę terminala. Postaraj się odnaleźć go za pomocą komendy ps oraz top. Odnajdź PID tego procesu.
11. Znając numer PID procesu z uruchomionym skryptem śpiocha, spróbuj w osobnej karcie terminala wysłać do niego sygnały: SIGINT, SIGTERM, SIGQUIT, SIGKILL. Za każdym razem sprawdź co się stało z procesem śpiocha. Jeśli trzeba uruchom go ponownie.
12. Przy użyciu maszyny wirtualnej (VirtualBox) spróbuj uruchomić proces jako jeden użytkownik, a następnie zabić go jako inny użytkownik. Czy to możliwe? Próbę ponów jako użytkownik root.

Laboratorium 5. Wejścia i wyjścia procesu

W systemie Linux każdy proces komunikuje się z otoczeniem typowo za pomocą trzech specjalnych strumieni (kanałów), przez które może otrzymywać lub nadawać dane.



stdin (0)

Standardowy strumień wejścia to dane (zwykle tekst) przekazywane do programu. Nie wszystkie programy wymagają danych wejściowych. Przykładowo, `ls` wykonuje swoją funkcję nie pobierając żadnych danych z stdin. O ile strumień nie jest przekierowany, dane są pobierane z terminala, z którego został uruchomiony program.

stdout (1)

Standardowy strumień wyjścia to strumień, do którego program zapisuje dane wynikowe. Niektóre programy nie zwracają danych wynikowych – na przykład `mv` nic nie wypisuje jeżeli przeniesienie się powiodło. Jeżeli strumień nie jest przekierowany dane są wysyłane do terminala z którego uruchomiono program

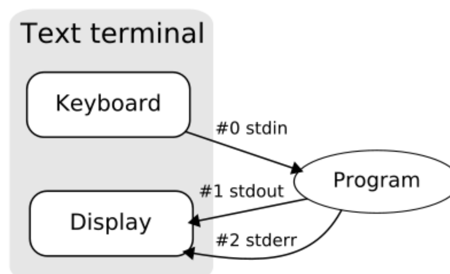
stderr (2)

Standardowy strumień błędów jest zwykle wykorzystywany do wyświetlania komunikatów o błędach i informacjach przydatnych do debugowania. Jest on niezależny od strumienia wyjścia. Zwykle celem strumienia jest, podobnie jak przy stdout terminal z którego uruchomiono program aby umożliwić zobaczenie błędu nawet wtedy, gdy strumień wyjścia jest przekierowany. Jeżeli używamy potoku aby użyć danych wynikowych jakiegoś programu jako danych wejściowych dla innego to błędy i tak zostaną wypisane na terminalu.

Gdy strumienie wyjścia i błędów mają ten sam cel (np. terminal) to są wyświetlane w takiej kolejności, w jakiej wypisuje je program, o ile nie korzysta się z buforowanego wyjścia. W takim wypadku dane z stderr wyświetlają się wcześniej, gdyż są zwykle niebuforowane, w przeciwieństwie do stdout, które są zwykle zapisywane w buforze przed wyświetleniem.

Typowe przekierowanie strumieni

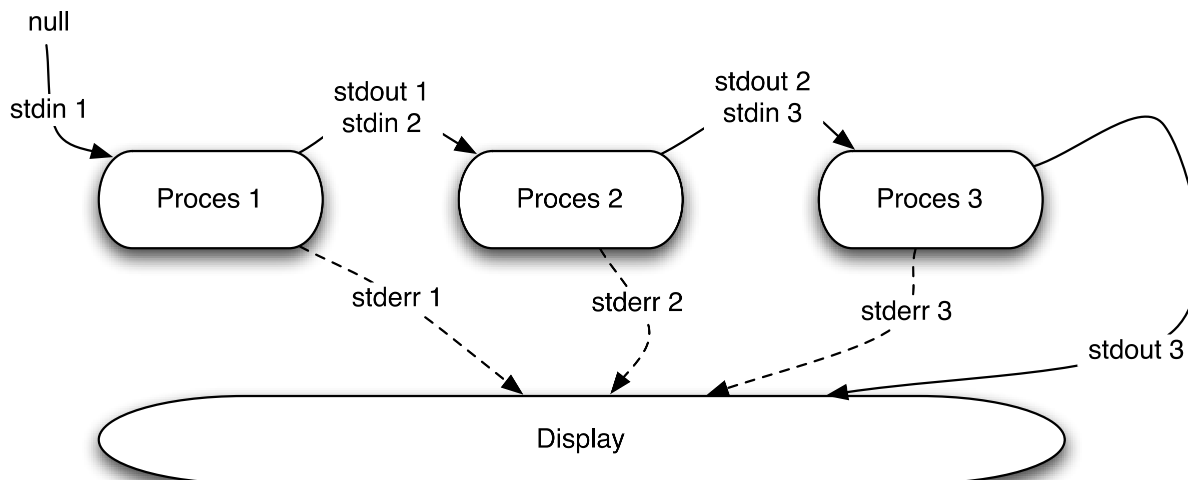
Gdy uruchamiamy proces w terminalu, typowo wszystkie strumienie przypięte są w następujący sposób do urządzeń komputera:



- Jeśli program wczytuje dane ze stdin, możliwe jest po jego uruchomieniu wpisanie standardowego wejścia bezpośrednio z klawiatury. Aby zakończyć wprowadzanie w nowej linii należy wcisnąć kombinacje klawiszy Ctrl+D
- Stdout oraz stderr są przypięte do konsoli, która wyświetli na ekranie wszystko co program zwróci (w trybie normalnym lub jako błąd).

Przetwarzanie potokowe (pipes)

Linux dostarcza niezwykle prostego a zarazem bardzo funkcjonalnego i wszechstronnego mechanizmu przetwarzania potokowego. Polega ono na tym, że można łączyć wywołania konkretnych procesów w ten sposób, aby strumień wyjściowy (najczęściej stdout) jednego procesu stanowił strumień wejściowy drugiego procesu (i tak wielokrotnie dalej).



Na rysunku powyżej widoczne są trzy procesy. Pierwszy proces uruchamia się i nie wymaga podania żadnych danych na wejściu. Tak np. zachowuje się program `ls` , który po uruchomieniu zwraca po prostu listę plików i katalogów na stdout. Standardowe wyjście pierwszego procesu staje się wejściem drugiego procesu. Drugi proces także dokonuje jakiś czynności, np. może filtrować linie z wejścia (zob. polecenie `grep`). To co zwróci drugi proces stanowi znowu wejście procesu trzeciego. On np. może policzyć ile linii zwrócił poprzedni proces po przefiltrowaniu (zob. polecenie `wc -l`). Wynik zostanie zwrócony na ekran. Dzieje się tak ponieważ stdout trzeciego procesu jest właśnie tam przekierowane (co jest domyślnym ustawieniem jak nie wskażemy żadnej innej akcji). Zwróć uwagę, że jedynym widocznych wyjściem z całej sekwencji uruchomionych trzech procesów jest tylko i wyłącznie to co zwróci trzeci proces. Dodatkowo na ekranie mogą pojawić się komunikaty błędów z każdego z trzech procesów, ponieważ ich standardowe wyjścia błędów są także przypięte do ekranu.

W systemie Linux uruchomienie takiej sekwencji trzech procesów połączonych wspólnym potokiem dokonuje się poprzez oddzielenie komend za pomocą pionowej kreski `|` . Znak ten na klawiaturze najczęściej uzyskuje się razem z Shiftem.

Przykładowe uruchomienie programu za pomocą potoku wygląda w tak:

```
ls /etc | grep ^a | wc -l
```

Najpierw uruchomi się `ls` zwracając listę plików i katalogów w `/etc`, informacja ta nie zostanie nigdzie wyświetlona, zamiast tego trafi jako wejście do programu `grep`. Program `grep` użyje wzorca (który w tym przypadku oznacza, że szuka wszystkich nazw zaczynających się na literę a) i zostawi tylko te linie pochodzące z komendy `ls`, które zaczynały się na literę a. Swój wynik przekaże do polecenia `wc` które z parametrem `-l` potrafi zliczyć ile linii zostało wprowadzonych na jego stdin. W ten sposób otrzymamy na ekranie jako wynik liczbę oznaczającą liczbę plików i katalogów które rozpoczynają się na literę a.

Przekierowania

Przekierowania służą do tego, aby zmienić domyślny sposób w jaki system Linux traktuje kierowanie strumieni procesu. W poprzednim rozdziale używając operatora pipe `|` skorzystaliśmy właśnie ze standardowego, domyślnego przypisania strumieni. Standardowo zawsze stdout trafia jako stdin kolejnego procesu, natomiast stderr trafia na ekran. Aby zmienić

Przekierowanie stdout do pliku z nadpisaniem `>`

Wyjście każdego procesu można przekierować do pliku. Używa się następującej składni:

```
polecenie [-parametry] > nazwa_pliku
```

lub

```
polecenie [-parametry] 1> nazwa_pliku
```

Po każdym uruchomieniu polecenia w ten sposób cała zawartość stdout trafi do pliku i nadpisze to co w nim się znajduje. Zauważ że numer 1 oznacza kod stdout. Można go pominąć ponieważ do domyślna wartość dla przekierowania.

Przekierowanie stdout do pliku z dopisaniem `>>`

Wyjście każdego procesu można przekierować do pliku. Używa się następującej składni:

```
polecenie [-parametry] >> nazwa_pliku
```

lub

```
polecenie [-parametry] 1>> nazwa_pliku
```

Po każdym uruchomieniu polecenia w ten sposób cała zawartość stdout trafi do pliku i dopisze się na koniec tego pliku. Zauważ że numer 1 oznacza kod stdout. Można go pominąć ponieważ do domyślna wartość dla przekierowania.

Przekierowanie stdout do pliku z nadpisaniem `2>`

Wyjście każdego procesu można przekierować do pliku. Używa się następującej składni:

```
polecenie [-parametry] 2> nazwa_pliku
```

Po każdym uruchomieniu polecenia w ten sposób cała zawartość stderr trafi do pliku i nadpisze to co w nim się znajduje. Zauważ, że numer 2 oznacza kod stderr. Nie można go pominąć, jeśli go pominiesz do pliku trafi zawartość stdout.

Przekierowanie stdout do pliku z dopisaniem `2>>`

Wyjście każdego procesu można przekierować do pliku. Używa się następującej składni:

```
polecenie [-parametry] 2>> nazwa_pliku
```

Po każdym uruchomieniu polecenia w ten sposób cała zawartość stderr trafi do pliku i dopisze się na koniec tego pliku. Zauważ, że numer 2 oznacza kod stderr. Nie można go pominąć, jeśli go pominiesz do pliku trafi zawartość stdout.

Przekierowanie stderr na stdout i na odwrót

Możliwe jest także przekierowanie strumieni jeden na drugi.

`1>&2` taki zapis spowoduje że standardowe wyjście (1) zostanie przekierowane do standardowego wyjście błędów (2)

`2>&1` taki zapis przekieruje stderr (2) na stdout (1)

Przekierowanie do urządzenia `/dev/null`

W systemie Linux istnieje specjalne urządzenie o nazwie `/dev/null`, które służy do „zapominania" informacji. Wszystko co na niego trafi zostanie po prostu usunięte z systemu (nie zostanie nigdzie zapisane, nawet na ekranie). Innymi słowy, jeśli zależy nam aby wyjście z programu było usunięte (czyli nie zapisane do pliku, ani nie zapisane na ekranie) należy przekierować dane wyjście na to urządzenie, np.:

```
polecenie [-parametry] > /dev/null
```

Takie polecenie pominie tylko stdout. Błędy tego polecenia nadal będą wyświetlane na ekranie. Często chcemy tego uniknąć (np. gdy uruchamiamy procesy w za pomocą crontab). Można wobec tego przekierować oba strumienie na `/dev/null`

```
polecenie [-parametry] > /dev/null 2>/dev/null
```

lub jeszcze prościej

```
polecenie [-parametry] 1> /dev/null 2>&1
```

powyższe polecenie przekieruje stdout na `/dev/null` a stderr na stdout (czyli w praktyce też na `/dev/null`).

Polecenie more i less

Polecenia `more` i less służą do tego, aby paginować (stronicować) treść ze stdin na ekranie terminala, który bardzo często jest zbyt mały aby wyświetlić całą zawartość pliku. `less` umożliwia bardziej zaawansowany interfejs paginacji (np. umożliwia wyszukiwanie w treści). W tym przypadku okazuje się że „less is more" ☺

Polecenie head i tail

Służy do obcinania treści ze stdin. Polecenie head potrafi obciąć treść z początku pliku (np. zwrócić dalej tylko 10 pierwszych linii), a polecenie tail od końca(np. ostatnich 10 linii). Szczególnie ciekawa opcja to `tail -f` pozwala na żywo wyświetlać treść plików który może się dynamicznie zmieniać (np. logi).

Polecenie wc

Wc jest skrótem od word count. Program umożliwia zliczenie różnych statystyk na temat danych wejściowych na stdin. Na przykład liczbę bajtów, wyrazów, linii.

Polecenie cut

cut - jest komendą uniksowego wiersza poleceń. Wykorzystuje się go do wyciągania części z każdej linii wejścia. Zazwyczaj polecenie to jest używane do wyciągania części z plików. Służą do tego sprecyzowane parametry:

- (-b) – dla określonych bajtów (ang. bytes),
- (-c) – dla określonych znaków (ang. characters),

- (-f) – dla określonych pól (ang. fields),
- (-d) – separator (ang. delimiter). Znak po -d jest separatorem. Standardowym separatorem jest: tab ale mogą być też inne symbole np. dwukropek.

Zasięg musi zawierać w każdym przypadku jeden z warunków N, N-M, N- (od N do końca linii), lub -M (od rozpoczęcia linii do M).

Polecenie sort

Umożliwia posortowanie danych wejściowych do stdin .

Polecenie uniq

Umożliwia ograniczenie liczby linii do unikalnych ze stdin .

Zadania do wykonania

1. Spróbuj policzyć liczbę plików i katalogów w twoim katalogu domowym za pomocą polecenia `ls` i `wc`.
2. Policz ile użytkowników znajduje się w systemie Linux na komputerze w laboratorium.
3. Za pomocą komendy `cut` spróbuj obciąć ze strumienia listy użytkowników same loginy użytkowników.
4. Wykonaj to samo co w powyższym punkcie, ale wytnij ścieżkę standardowego shella użytkowników. (wskazówka, polecenie `cut` posiada dwie przydatne opcje `-f -d`)
5. Policz ile różnych shelli używają użytkownicy w systemie Linux na komputerze w laboratorium oraz na wierzbie?
6. Pobierz plik z expose premiera (np. poleceniem wget)

```
wget http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/lab-05/expose_tuska.txt
```

   a. Wyświetl zawartość pliku.
   b. Użyj stronicowania more i less, aby przeglądnąć cały plik.
   c. Za pomocą polecenia less odnajdź na ekranie wszystkie wystąpienia wyrazu „Polska”.
   d. Wyświetl pierwsze 25 linii pliku.
   e. Wyświetl ostatnie 13 linii pliku.
   f. Wyświetl linie 10 do 15 pliku.
   g. Policz liczbę wyrazów w pliku.
   h. Dla każdej linii wyświetl 4 wyraz w tej linii.
7. Zapisz listę plików i katalogów w katalogu `/etc` do pliku `lista_etc`
8. Użyj tego samego polecenia jak w punkcie powyżej ale zamiast `/etc` użyj nieistniejącego katalogu. Czy komunikat błędu zapisał się do pliku czy wyświetlił na ekranie?
9. Zmodyfikuj polecenie z powyższego punktu tak aby komunikat błędu także zapisał się do pliku.
10. Napisz polecenie, które po każdym uruchomieniu dopisze obecną datę i godziną do pliku o nazwie `uruchomienia`.

Laboratorium 6. Zmienne środowiskowe

Zmienna środowiskowa to nazwana wartość, zazwyczaj zawierająca ciąg znaków, przechowywana i zarządzana przez powłokę (shell). Zmienna środowiskowa może wpływać na działanie procesów uruchamianych w systemie operacyjnym i wtedy staje się pewnym mechanizmem komunikacji lub też przechowywać wartość w celu jej późniejszego wykorzystania.

Zmienna środowiskowa stanowi kolejny sposób poza standardowym wejściem programu (stdio) oraz parametrami wywołania programu (opcjami) sposób na przekazanie wartości do uruchamianego programu. Bardzo często programy po uruchomieniu odczytują sobie stan zmiennych środowiskowych ponieważ i używają ich wartości jako pewnych dodatkowych aspektów konfiguracji (np. korzystają z informacji na temat jaką nazwę ma użytkownik, który uruchomił program).

Każda powłoka (np. csh, bsh, zsh, itp…) może posiadać własny odmienny sposób realizacji zmiennych środowiskowych. Na laboratorium poznajemy powłokę bash i w związku z tym wszystkie przykłady odnoszą się właśnie do tej powłoki.

Tworzenie zmiennej środowiskowej

Zmienne środowiskowe w bash nie posiadają typów. Oznacza to, że każda wartość, którą w niej przechowujemy zachowuje się jak tekst. Mimo to w zmiennych możemy przechowywać liczby, powłoka potrafi automatycznie rzutować typy i w niektórych kontekstach traktować ich zawartość jak liczbę.

Często spotykaną konwencją jest pisanie zmiennych środowiskowych KAPITALKAMI. Nie jest to jednak wymagane, co więcej wielkość liter ma znaczenie przy tworzeniu i posługiwaniu się zmiennymi środowiskowymi.

Nazwy zmiennych mogą składać się z dużych i małych liter, podkreślenia i cyfr. Cyfra nie może znajdować się na początku nazwy zmiennej.

`ZMIENNA=”Dowolny napis”` - tworzenie zmiennej jako zwykły łańcuch tekstowy (zwróć uwagę na brak spacji pomiędzy nazwą zmiennej i znakiem równości i wartością w cudzysłowach)

`ZMIENNA=napis` – tworzenie zmiennej jako łańcuch tekstowy (bez spacji nie trzeba cudzysłowów)

`ZMIENNA=123` – podanie liczby także utworzy zmienną (będzie ona interpretowana zarówno jako tekst składający się z trzech znaków, ale także jako liczba jeśli będzie trzeba)

`ZMIENNA=$INNA_ZMIENNA` – przypisanie wartości jednej zmiennej do drugiej

Operator wyłuskania wartości zmiennej

Dostęp do wartości zmiennej realizowany jest poprzez znak `$`. Przykładowo wyświetlenie wartości zmiennej można zrobić następująco:

`ZMIENNA=123`

`echo $ZMIENNA` – wyświetli 123

Uwaga, jeśli chcesz wypisać po prostu napis „$ZMIENNA" użyj znaku escape:

`echo \$ZMIENNA` – wyświetli napis $ZMIENNA

Zmienne specjalne bash

W powłoce bash zdefiniowanych jest kilka bardzo pomocnych zmiennych specjalnych. Poniżej zestawienie najważniejszych zmiennych specjalnych:

Nazwa zmiennej	Opis
$?	Zwraca „exit status" czyli kod zakończenia programu. Każdy program, który się kończy ustawia w systemie specjalną wartość zakończenia. Domyślnie program zwraca 0 jeśli wszystko przebiegło w nim bez zakłóceń. Wartość różna od 0 oznacza jakiś konkretny błąd właściwy dla tego programu.
$#	Zwraca liczbę parametrów z którymi uruchomiono program
$@	Zwraca listę parametrów z którymi uruchomiono program
$$	Zwraca PID powłoki
$!	Zwraca PID ostatnio uruchomionego procesu w tle (bg - background)
$0	Zwraca nazwę programu lub shella
$1, $2 … $n	Zwraca n-ty parametr wywołania programu

Standardowe zmienne

Istnieje kilka standardowo ustawianych zmiennych, których znajomość ułatwia np. pisanie skryptów.

Nazwa zmiennej	Opis
$PS1 .. $PS4	Zmienna $PS1 umożliwia zmianę domyślnego znaku zachęty (prompt)
$USER	Zwraca nazwę użytkownika aktualnie zalogowanego
$UID	Zwraca numer UID aktualnego użytkownika
$RANDOM	Zwraca losową liczbę przy każdym odczytaniu wartości z przedziału 0 do 32767
$PATH	Zwraca listę ścieżek na których wyszukiwane są programy do uruchomienia, ścieżki oddzielone są za pomocą znaku dwukropka
$HOME	Zwraca ścieżkę katalogu domowego użytkownika.
$SHELL	Zwraca ścieżkę interpretera uruchomionego w tej sesji

Przydatnym program do wylistowania wszystkich zmiennych w systemie jest polecenie `set`. Polecenie uruchomione bez parametrów wyświetli wszystkie zmienne i ich wartości.

Zasięg zmiennych

Zmienne mogą mieć dwa rodzaje zasięgów: lokalny i globalny. Lokalny zasięg jest to typowy domyślny zasięg zmiennej, gdy po prostu definiujemy zmienną w bashu. Zasięg lokalny oznacza, że zmienna ta nie zostanie przekazana do innego shella jeśli uruchomimy instancję kolejnego shella w bieżącej sesji shella.

Zasięg globalny zmiennej oznacza, że zmienna będzie przekazywana do kolejnych powłok uruchomionych z danej powłoki.

`export ZMIENNA="wartość"` – ustawienie właściwości eksportu dla ZMIENNA,

`export -n ZMIENNA` – wyłączenie właściwości eksportu dla zmiennej,

`export` lub `export -p` – wyświetlenie listy eksportowanych zmiennych w danej powłoce.

Zmiana wartości zmiennej dla konkretnego programu

Podczas gdy uruchamiamy dowolny program (np. skrypt powłoki bash jest tez programem) dysponuje on dostępem do wszystkich zmiennych środowiskowych posiadających opcję exportu.

Często chcemy jednak na konkretne uruchomienie programu przypisać inną chwilową wartość zmiennej tylko dla tego programu. Nie chcemy zmieniać wartości zmiennej w całej powłoce. Możemy w tedy uruchomić program poprzedzając go deklaracją zmiennej której wartość chcemy chwilowo podmienić.

```
export ZMIENNA="domyślna wartość"
```

```
ZMIENNA="chwilowa wartość" ./program
```

W powyższym przykładzie mimo, że wartość zmiennej ustawiona została na „domyślna wartość", program uruchomi się z chwilową wartością. Po jego zakończeniu shell nadal będzie w tej zmiennej miał „domyślną wartość".

Renderowanie zmiennej

Bash umożliwia nam renderowanie (czyli interpolacje, wkładanie wartości) zmiennej do dowolnego napisu. Przykładowo:

```
Z=kota
```

`echo „Ala ma $Z"` – wyświetli Ala ma kota

można także użyć wartości zmiennej jako argumentu

`ls $Z.zip` – wylistuje wszystkie pliki kota.zip

Może się tak zdarzyć, że przy renderowaniu chcemy połączyć nazwę zmiennej z tekstem bez spacji. Musimy wtedy ująć nazwę zmiennej w nawiasowanie `${ZMIENNA}`

```
Z=kot
```

`echo „Ala ma ${Z}a, $Z ma Alę"` - wyświetli `„Ala ma kota, kot ma Alę"`

Powyższy przykład pokazuje, że gdyby nie było nawiasowania, bash szukał by zmiennej `$Za` zamiast `$Z`.

Przypisanie stdout programu do zmiennej

Możliwe jest przekierowanie standardowego wyjścia programu w taki sposób, aby wartość została zapisana jako zmienna.

Możliwe są dwie notacje:

`ZMIENNA=`ls -al.``

lub

`ZMIENNA=$(ls -al)`

Obie powyższe linie są równoważne. Zwróć uwagę, że w pierwszej linii użyto znaku „odwrócony pazurek” czyli znaku znajdującego się razem z tyldą na jednym klawiszu standardowej klawiatury PC.

W powyższym przykładzie lista plików, którą zwróci polecenie `ls -al` zostanie przypisana jako wartości zmiennej.

Wczytanie wartości zmiennej interaktywnie od użytkownika

Bash umożliwia wygodny sposób na wczytanie wartości zmiennej od użytkownika. Szczególnie przydatne podczas pisania skryptów.

`read ZMIENNA` - polecenie to wczyta w terminalu wartość od użytkownika i ustawi jako wartość zmiennej podanej jako parametr

Wykonywanie operacji matematycznych na zmiennych

Niektóre interpretery jak bash posiadają wbudowany operator umożliwiających liczenie całkowito liczbowe. Dostępne są dwie notacje:

`echo $((2+3))` – wyświetli 5 ☺

`echo $[2+3]` – także wyświetli 5 ☺

Można oczywiście liczyć na zmiennych:

`A=2`

`B=3`

`echo $((A+B))` – wyświetli 5

`echo $[A+B]` – wyświetli 5

Zwróć uwagę na brak znaku dolara w takiej notacji. Pamiętaj, że dzielenie w ten sposób jest całkowito liczbowe:

`echo $[2/3]` – zwróci 0

Jeśli potrzebujesz dzielenia na liczbach rzeczywistych użyj wbudowanego programu kalkulatora bc z opcją –l uruchamiającego liczenie na liczbach rzeczywistych:

`echo "2/3" | bc -l` – zwróci .66666666666666666666

Wykonanie operacji na znakach w tekście zmiennej

Oprócz operacji na liczbach, możliwe jest też operowanie na zmiennej jak na tekście. Wobec tego możliwe są różne operatory związane z obcinaniem tekstu w zmiennej. Poniżej przedstawiono przykłady użycia konkretnych operatorów. Typowy schemat użycia jest taki:

`$ { NAZWA_ZMIENNEJ OPERATOR WZORZEC }`

dostępne operatory to:

- `%` - obcina jeden ostatni znaleziony wzorzec z wartości zmiennej
- `%%` - obcina najdłuższy od tyłu znaleziony wzorzec z wartości zmiennej
- `#` - obcina pierwszy znaleziony wzorzec z wartości zmiennej
- `##` - obcina najdłuższy z przodu znaleziony wzorzec z wartości zmiennej

`MYVAR=foodforthought.jpg`

`echo ${MYVAR##*fo}` – w tym przykładzie wzorzec to `*fo` a operator to `##`. Obcięte zostanie wobec tego najdłuższe od początku wystąpienie wzorca (`foodfo`) a zwrócone zostanie `rthought.jpg`

echo ${MYVAR#*fo} – w tym przykładzie wzorzec to `*fo` a operator to `#`. Obcięte zostanie wobec tego najkrótsze od początku wystąpienie wzorca (`fo`) a zwrócone zostanie `odforthought.jpg`

`MYFOO="chickensoup.tar.gz"`

`echo ${MYFOO%%.*}` – w tym przykładzie wzorzec to `.*` a operator `%%`. Obcięte zostanie najdłuższy od końca pasujący wzorzec (`.tar.gz`) a zwrócone zostanie `chickensoup`

`echo ${MYFOO%.*}` – w tym przykładzie wzorzec to `.*` a operator `%`. Obcięte zostanie najkrótszy od końca pasujący wzorzec (`.gz`) a zwrócone zostanie `chickensoup.tar`

W ten sam sposób można także obcinać ścieżki, ale o wiele wygodniej użyć do tego poleceń `basename` i `dirname`.

Polecenia do wykonania i zagadnienia

1. Wyświetl i zawartość zmiennych systemowych na komputerach w laboratorium.
2. Wyświetl numer użytkownika UID na komputerze w laboratorium i na wierzbie. Czy są takie same?
3. Ustaw zmienną środowiskową o nazwie `OPCJA` na wartość `-alh`
4. Użyj zmiennej środowiskowej `OPCJA` razem z poleceniem `ls` aby wyświetlić pliki w katalogu `$HOME` z tymi opcjami znajdującymi się w zmiennej `$OPCJA`.
5. Sprawdź czy zmienna opcja dostępna jest w kolejnej instancji shella którą uruchomisz z bieżącej instancji (uruchom polecenie `bash` , aby wyjść z kolejnej instancji shella wpisz `exit`)

6. Sprawdź jaki kod wyjścia (exit status) zwraca polecenie `ls` gdy poprawnie wylistuje pliki oraz wtedy gdy podany jako parametr plik lub katalog nie istnieje.
7. Spróbuj uruchomić kolejną instancję powłoki i poleceniem `exit` zwrócić kod wyjścia 13. Sprawdź w powłoce wyżej czy otrzymano kod właśnie taki kod wyjścia.
8. Ustaw wartość zmiennej `JABLKA` na 2. Skonstruuj polecenie uzupełniając puste pola aby uzyskać wymagany tekst na wyjściu:

   ```
   echo „Ja mam ……….. jabłek. Ty masz ……… jabłek. On ma ……… jabłek”
   ```

   ma wyświetlić

   ```
   Ja mam 2 jabłek. Ty masz 4 jabłek. On ma 8 jabłek
   ```

   Musisz użyć zmiennej `JABLKA` oraz operacji matematycznych.

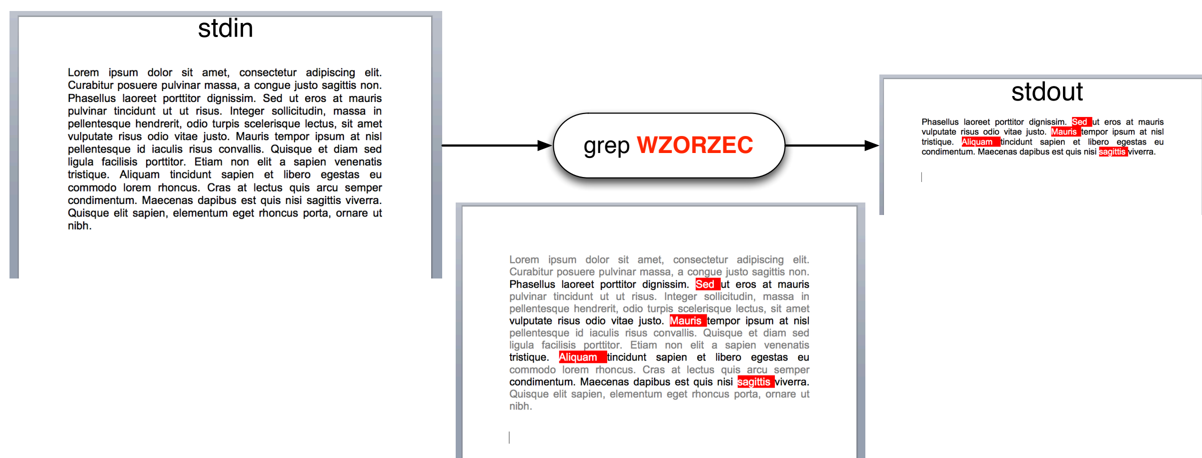
9. Dodaj do zmiennej środowiskowej `PATH` ścieżkę `$HOME/bin`.
10. Za pomocą operatorów obcinania napisów wyświetl tak obciętą ścieżkę `PATH` aby zawierała jedynie ścieżkę, która należało dodać w poprzednim zadaniu.
11. Ustaw zmienną `TEKST` na wartość `„raz-dwa;trzy-cztery;piec-szesc;siedem-osiem”`. Obetnij zmienną TEKST tak aby uzyskać następujące napisy:
    a. raz-dwa
    b. siedem-osiem
    c. raz
    d. osiem
    e. szesc;siedem
    f. dwa;trzy
    g. szesc
    h. piec
    i. cztery;piec

    Uwaga: możesz używać zmiennych pośrednich jeśli potrzebujesz.

12. Spróbuj wykonać powyższe ćwiczenie używając polecenia `cut` zamiast operatorów obcinania stringu.

Laboratorium 7: Filtrowanie strumienia tekstu i wyrażenia regularne

System Linux dostarcza narzędzia do filtrowania strumieni tekstu o bardzo dużych możliwościach. Program, który realizuje filtrowanie nazywa się `grep`, co jest skrótem od *global regular expression print*. Filtrowanie to polega na zdefiniowaniu wzorca, który jest wyszukiwany w treści. Typowe działanie programu grep zakłada, odfiltrowanie linii tekstu znajdujących się na standardowym wejściu (lub w pliku) i zwrócenie na standardowe wyjście tylko tych linii, które zawierały podany wzorzec.



Porgram grep można użyć zarówno w kontekście filtrowania tekstu w strumieniu danych:

```
cat file.txt | grep WZORZEC
```

jak i również w kontekście zawartości pliku na dysku podając ścieżkę tego pliku jako parametr:

```
grep WZORZEC file.txt
```

Wybrane opcje polecenia grep

Polecenie grep posiada wiele przydatnych opcji. Np.

Przełącznik	Opis
-v	negacja wzorca (zostają zwrócone tylko linie NIE zawierające wzorca)
-c	zamiast wyświetlać linie ze znalezionym fragmentem wyświetla liczbę znalezionych linii
-L	nie wyświetla znalezionego fragmentu tylko pokazuje nazwy plików, w których nie było tego wzorca
-l	odwrotnie do poprzedniego polecenia, wyświetla pliki w których znaleziono dany wzorzec
-n	wyświetlany jest numer linii w pliku w których znaleziono dany wzorzec
-w	wyszukuje tylko całe słowa
-h	o podaniu kilku plików do przeszukania po znalezieniu danego ciągu znaków przy danej linii nie będzie podany plik, w którym znaleziono

	wzorzec
-r	przeszukiwanie rekursywne podkatalogów
-i	ignorowanie rozróżniania dużych i małych liter
-o	wyświetl tylko dopasowany ciąg
-x	wyszukuje tylko całe linie

Proste wzorce

Najprostszym rodzajem wzorców są napisy, które należy bezpośrednio odnaleźć.

```
>cat file
big
bad
bug
bag
bigger
boogy

>grep big file
big
bigger
```

W prostych wzorcach można używać także wildcardów: znaku `.` oraz znaku `*`. Znak kropki oznacza jeden dowolny znak.

```
> grep "b.g" file
big
bad bug
bag
bigger
```

Złożone wzorce (grep –E)

Grep umożliwia użycia rozszerzonej (extended) składni, która udostępnia mechanizm wyrażeń regularnych. Wyrażenie regularne służą do opisu formalnego łańcuchów symboli. Jeśli symbole wykazują jakieś regularności (np. w wyrazach aba, abba, abbba, abbbbba, … regularność polega na tym, że na początku i końcu znajduje się litera a, a w środku dowolna niezerowa liczba liter b) to można utworzyć zapis, który opisze tą regularność (np. ab+a).

Znak gwiazdki `*` oznacza operator powtórzenia zero lub więcej razy. Oznacza on że znak znajdujący się bezpośrednio przed nim może wystąpić zero lub więcej razy.

```
> grep -E "b.g*" file --color
big
bad bug
bag
bigger
boogy
```

Znak plus + oznacza operator powtórzenia jeden lub więcej razy:

```
> grep -E "b.g+" file
```

```
big
bad bug
bag
bigger
```

Operaotry klamrowe `{}` pozwalają precyzyjnie sterować liczbą powtórzeń. Np.:

- `{m,n}` – może wystąpić dokładnie od n do m razy,
- `{,n}` – może wystąpić najwięcej n razy,
- `{m,}` – może wystąpić najmniej m razy,
- `{m}` – może wystąpić dokładnie m razy

Przykładowo poniże wyrażenie znajdzie ciąg znaków zaczynający się na literę b, które ma później dowolny znak, i dokładnie dwie litery g.

```
> grep -E "b.g{2}" file
bigger
```

Istnieje jeszcze specjalny operator `?` , który oznacza że dany ciag może wystąpić zero lub raz, czyli jest on równoważny użyciu `{0,1}`. W kolejnym przykładzie pierwsze g może się alternatywnie pojawić w wyszukanym wzorcu.

```
> grep -E "big?g" file
big
bigger
```

Operator klasy (nawiasy kwadratowe) definiują zbiór znaków jaki może pojawić się w jej miejscu. Np.

```
> grep -E "b[ai]g" file
big
bag
bigger
```

Operator klasy można łączyć z operatorem gwiazdki:

```
> grep -E "b[aio]*g" file
big
bag
bigger
boogy
```

Można również definiować przedziały, np. `[0-4]` to oznacza dowolny znak ze zbioru `[01234]`. Tak samo zakresem może być `[a-d]` co oznacza `[abcd]`. Zakresy można łączyć np. `[a-cg-j]` oznacza `[abcghj]`.

Jak i negować klasę (tzn. utworzyć klasę zawierającą wszystkie możliwe znaki poza tymi zadeklarowanymi w klasie). Negowanie robi się wstawiając znak `^` na początku definicji klasy:

```
> grep -E "b[^aio]*g" file
bad bug
```

Ważnymi operatorami jest `^` - początek linii oraz `$` koniec linii. Użycie ich we wzorcu wymusza wystąpienie początku lub końca linii w określonym miejscu:

```
> grep -E "^b[oaui]g" file
big
bag
bigger
> grep -E "b[oaui]g$" file
big
bag
```

Nawiasy okrągłe służą do grupowania wielu znaków, dzięki czemu możemy użyć operatorów na całych grupach wzorca. Np.

```
> grep -E "big(ger)?" file
big
bigger
```

Operator alternatywy `|` umożliwia dopasowanie jednego lub drugiego wzorca.

```
> grep -E "b(igger|ig)" file
big
bigger
```

Podsumowanie operatorów wyrażeń regularnych

Operator	Opis
.	Dowolny znak
*****	Wzorzec wystąpi zero lub więcej razy
+	Wzorzec wystąpi raz lub więcej razy
{m,n}	Dokładnie od m do n razy
{,m}	Wystąpi maksymalnie m razy
{n,}	Wystąpi najmniej n razy
{m}	Wystąpi dokładnie m razy
?	Wzorzec wystąpi zero lub raz
^	Reprezentuje początek linii
$	Reprezentuje koniec linii
[ ]	Klasa znaków
( )	Nawiasy grupujące wyrażenia
|	Alternatywa wyrażeń

Zadania do wykonania na zajęciach

1. Pobierz i rozpakuj plik `http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/lab-07/slowa.txt.zip`
2. Przy użyciu `grep` i `ls` odnajdź wszystkie pliki lub katalogi w `/etc` które zawierają słowo „conf”.
3. Przeglądnij zawartość pliku za pomocą poleceń `head` i `tail`. Sprawdź czy twoja konsola poprawnie wyświetla polskie znaki diakrytyczne. Kodowanie plików to UTF-8.

4. Wyświetl oraz policz wszystkie wyrazy, które:
    a. posiadają ciąg „maku”,
    b. zaczynają się od „maku”,
    c. kończą się na „maku”.
5. Napisz klasę reprezentującą wszystkie samogłoski w języku polskim. Sprawdź jaki wyraz (wyrazy) posiadają zbitkę o największej długości samogłosek w swojej treści. Np. wyraz „boeing” posiada zbitkę o długości 3 „oei”.
6. Wykonaj analogiczne zadanie, ale znajdź wyrazy o najdłuższej zbitce spółgłosek. Ile ich jest?
7. Znajdź wyrazy, które zaczynają na „he” a kończy na „by”.
8. Znajdź wszystkie wyrazy, które zawierają w sobie polskie znaki diakrytyczne.
9. Znajdź wszystkie wyrazy, które zawierają w sobie przynajmniej 5 lub 6 polskie znaki diakrytyczne.
10. Znajdź za pomocą jednego wyrażenia regularnego wyrazy zaczynające się albo do „pod” albo od „nie” i kończące się na „śmy”, zawierające w środku pomiędzy tymi członami zbitkę dwóch samogłosek.
11. Pobierz plik `http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/lab-07/plan_roku.txt`
12. Napisz wyrażenie regularne, które wyszuka wszystkie daty w tekście (wyświetli na wyjściu same daty). Skorzystaj w tym celu z opcji `-o` polecenia `grep`.
13. Napisz wyrażenie regularne, które dopasuje się do wzorca numeru uchwały (powinno być uniwersalne dla różnych numerów). Numer uchwały w tym tekście to 42/VI/2011.
14. Pobierz plik `http://wierzba.wzks.uj.edu.pl/~dorosz/SOS1/lab-07/email.txt` i napisz wzorzec który zwróci wszystkie adresy e-mail. Przetestuj go na innych znanych ci adresach e-mail dopisując je do pliku.

Laboratorium 8: Skrypty powłoki bash

Skrypty są to zestawy instrukcji zapisane w pliku, które mogą zostać zinterpretowane linia po linii. Najprostszy skrypt jest to po prostu zestaw zwykłych instrukcji. Umożliwia on np. zautomatyzowanie jakiegoś procesu (wykonanie po sobie kilku czynności na raz).

Np. wykonanie kolejnych instrukcji

```
#> echo "Wyswietlam liczbe plikow w katalogu $HOME"
Wyswietlam liczbe plikow w katalogu /Users/cypreess
#> cd $HOME
#> ls | wc -l
      27
```

Może zostać zapisane jako skrypt:

```
#!/bin/bash
echo "Wyswietlam liczbe plikow w katalogu $HOME"
cd $HOME
ls | wc -l
```

i uruchomione w dowolnym momencie. Warto przy okazji zwrócić uwagę, że takie instrukcje można też wykonać za pomocą jednej linii polecania bash używając operatora średnika `;`, który rozdziela instrukcje.

```
echo "Wyswietlam liczbe plikow w katalogu $HOME"; cd $HOME; ls | wc -l
```

Sha-bang #!

Pierwsza linia skryptu powinna zawierać tzw. sha-bang `#!`, który jest specjalnym dwubajtowym znacznikiem pliku w Linux. Podana po nim ścieżka oznacza program jaki należy uruchomić aby zinterpretować skrypt. W przypadku skryptów bash jest to np. `#!/bin/bash` (zakładając że bash można odnaleźć pod ścieżką `/bin/bash`). Nazwa sha-bang podchodzi od skrótu nazwa sharp (`#`) oraz bang (`!`).

Aby skrypt mógł zostać uruchomiony musi mieć prawa do wykonania (`+x`). Typowym błędem jaki można zaobserwować próbując uruchomić skrypt bez nadania mu prawd do wykonania jest:

```
-bash: ./test.sh: Permission denied
```

Skrypty mogą mieć dowolne rozszerzenia (lub w ogóle brak rozszerzenia), natomiast wg konwencji skrypty powłoki zwykło się nazywać rozszerzeniem `.sh` (od shell).

W skryptach bash można wykorzystywać możliwości wszystkich programów, zmiennych środowiskowych oraz korzystać z możliwości wielu konstrukcji pętli, warunków logicznych i konstrukcji warunkowych – otrzymując bardzo rozbudowane możliwości do pisania skomplikowanych programów.

Komentarze

Skrypty umożliwiają wprowadzanie komentarzy do kodu za pomocą znaku # który oznacza że wszystko od tego znaku do końca linii nie będzie interpretowane. Np.

```
#!/bin/bash

# Ten program wchodzi do katalogu $HOME

cd $HOME # tutaj następuje wejście do katalogu
```

Sha-bang jest znakiem specjalnym i zostanie zinterpretowany przez skrypt.

Przekazywanie parametrów do skryptu przez argumenty

Istnieją dwa typowe sposoby na sparametryzowanie działania skryptu. Pierwszym sposobem jest przekazanie argumentów wywołania skryptu. Uruchamiając skrypt w poniższy sposób:

```
#> ./skrypt.sh pierwszy drugi trzeci
```

w środku skryptu można korzystać ze specjalnych zmiennych środowiskowych:

Nazwa zmiennej	Opis działania
$0	Nazwa skryptu (tak jak został wywołany)
$1, $2 … $n	n-ty parametr skryptu
$*	Wszystkie argumenty jako jeden wyraz
$@	Lista wszystkich argumentów – jako wiele wyrazów
$#	Liczba argumentów

Do wygodnego wyciągania kolejnych argumentów z `$@` służy polecenie `shift`. Wykonanie tego polecenia powoduje, że usuwany jest pierwszy argument na liście, a zmienna `$@` zawiera pozostałe argumenty.

Przekazywanie parametrów do skryptu przez zmienne środowiskowe

Drugim wygodnym sposobem parametryzowania pracy skryptu jest odczytanie przez skrypt wartości zmiennej środowiskowej (czyli kontekstu środowiska w jakim został uruchomiony).

Typowymi przykładami jest np. użycie w skrypcie wartości zmiennej $HOME albo $USER, w ten sposób skrypt personalizuje swoje użycie do konkretnego katalogu domowego użytkownika który go uruchomił, albo do jego nazwy użytkownika.

Można oczywiście przekazać dowolne inne zmienne na dwa sposoby:

```
#> export DOWOLNA_OPCJA=123

#> ./skrypt.sh
```

lub

```
#> DOWOLNA_OPCJA=123 ./skrypt.sh
```

Więcej informacji znajdziesz w laboratorium poświęconym zmiennym środowiskowym.

Konstrukcje warunkowe

Bash dostarcza dwóch typowych konstrukcji warunkowych `if else` oraz `case`.

If … elif … else

W przypadku prostym kiedy chcemy wykonać tylko czynność jeśli warunek jest spełniony używamy konstrukcji:

```
if [ WARUNEK ]; then
        AKCJA;
fi
```

W przypadku gdy chcemy w przeciwnym razie wykonać inną akcję:

```
if [ WARUNEK ]; then
        AKCJA;
else
        AKCJA;
fi
```

W przypadku gdy jest więcej niż jeden warunek dla których chcemy wykonać jakieś akcje:

```
if [ WARUNEK ]; then
elif [ WARUNEK ]; then
        AKCJA;
else
        AKCJA;
fi
```

case

Bardzo wygodną konstrukcją jest `case` umożliwiający wykonanie różnych akcji w zależności od wartości jednej zmiennej

```
case $ZMIENNA in
        „wartość 1”) AKCJA 1;;
        „wartość 2”) AKCJA 2;;
        „wartość 3”) AKCJA 3;;
        *) AKCJA DOMYŚLNA ;;
esac
```

`AKCJA DOMYŚLNA` jest wykonana tylko w tedy jeśli wartość `$ZMIENNA` jest inna niż te zdefiniowane w sekcji `case`.

Budowanie warunków

Używając konstrukcji `if` oraz `elif` należy podać warunek. Istnieje kilka rodzajów operatorów warunków

Operatory numeryczne

Operator	Opis
n1-eq n2	Sprawdza czy n1=n2
n1-ne n2	Sprawdza czy n1!=n2
n1-gt n2	Sprawdza czy n1>n2
n1-ge n2	Sprawdza czy n1>=n2
n1-lt n2	Sprawdza czy n1<n2
n1-le n2	Sprawdza czy n1<=n2

Operatory do porównywania napisów

Operator	Opis
s1	Sprawdza, czy ciąg jest niepusty
s1 = s2	Sprawdza, czy ciągi są identyczne
s1 != s2	Sprawdza, czy ciągi są różne
s1 > s2	Sprawdza, czy ciąg s1 jest większy w porządku alfabetycznym od s2
s1 < s2	Sprawdza, czy ciąg s1 jest mniejszy w porządku alfabetycznych od s2
-z s1	Sprawdza czy ciąg jest pusty
-n s1	Sprawdza czy ciąg jest niepusty

Operatory plikowe

Operator	Opis
-d file	Directory
-e file	Exists (also -a)
-f file	Regular file
-h file	Symbolic link (also -L)
-p file	Named pipe
-r file	Readable by you
-s file	Not empty
-S file	Socket
-w file	Writable by you
-N file	Has been modified since last being read
file1 -nt file2	Test if file1 is newer than file 2. The modification date is used for this and the next comparison.
file1 –ot file2	Test if file1 is older than file 2.
file1 –ef file2	Test if file1 is a hard link to file2.

Opearatory logiczne

Operator	Opis
WARUNEK && WARUNEK	Logiczny AND
WARUNEK || WARUNEK	Logiczny OR
! WARUNEK	Logiczny NOT

W praktyce używanie warunków w konstrukcji `if else` wygląda np. następująco:

```
if [ $USER = „student” ]; then …
```

Wszystkie wyżej wymienione operatory służą argumenty do wbudowanego w bash polecenia `test`. Można więc testować także warunki osobno w linii poleceń używając polecenia `test` i podając jako argumenty warunek który chce się przetestować. **Uwaga, polecenie `test` zwraca exit code jako `0` w przypadku powodzenia warunku!** (czyli odwrotnie niż we wszystkich językach programowania).

Pętla for .. in

Pętli tej używamy jeśli chcemy przeiterować pętle po liście elementów, przypisując dany kolejny element do zmiennej w każdym kroku. Konstrukcja pętli for .. in jest następująca:

```
for VARIABLE in 1 2 3 4 5 .. N

do

        command1

        command2

        commandN

done
```

Pętla for (; ;)

Jest to pętla podobna do tej występującej w języku C:

```
for (( EXP1; EXP2; EXP3 ))

do

        command1

        command2

        command3

done
```

np.

```
#!/bin/bash

for (( c=1; c<=5; c++ ))

do

        echo "Welcome $c times..."
```

```
done
```

Pętla while

Najpierw sprawdza warunek czy jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub lista poleceń zawartych wewnątrz pętli, gdy warunek stanie się fałszywy pętla zostanie zakończona.

```
 while [ WARUNEK ]
do
        polecenie
done
```

Pętla until

Sprawdza czy warunek jest prawdziwy, gdy jest fałszywy wykonywane jest polecenie lub lista poleceń zawartych wewnątrz pętli, miedzy słowami kluczowymi `do` a `done`. Pętla `until` kończy swoje działanie w momencie gdy warunek stanie się prawdziwy.

```
until [ WARUNEK ]
do
polecenie
done
```

Polecenia i zadania do wykonania

1. Zapoznaj się ze skryptem `arguments.sh` i zobacz różnicę pomiędzy `$*` i `$@`.
2. Zapoznaj się z przykładem działania polecenia `shift` z przykładu `shift.sh`.
3. Napisz skrypt `projekt.sh`, który przygotowuje strukturę katalogów dla nowego projektu:
   a. Uruchamia się go z dwoma argumentami (skrypt powinien sprawdzić liczbę argumentów).
   b. Pierwszym argumentem jest ścieżka do dowolnego katalogu (sprawdzić czy podano prawidłową ścieżkę do istniejącego katalogu).
   c. Drugim argumentem jest nazwa projektu.
   d. W podany katalogu powinien utworzyć katalog taki jak nazwa projektu a w nim strukturę katalogów zdefiniowaną następująco:

```
materialy/
materialy /oficjalne/
materialy /robocze/
src/
dokumentacja/
dokumentacja/uzytkownika/
dokumentacja/administratora/
```

4. Napisz skrypt `archiwum.sh`, który działa w następujący sposób:
    a. Skrypt potrafi kopiować całą zawartość podanego katalogu, do katalogu zdefiniowanego jako archiwum, tworząc snapshot (zrzut) plików w danym momencie.
    b. Skrypt powinien korzystać ze zmiennej `$ARCHIVE_PATH`, która określa ścieżkę gdzie znajduje się archiwum, jeśli jest nie zdefiniowane, należy użyć `$HOME/archive/`
    c. Jedynym argumentem wywołania skryptu powinna być dowolna ścieżka (relatywna lub absolutna), należy sprawdzić czy ścieżka istnieje.
    d. Działanie programu powinno polegać na tym aby skopiować całą zawartość tego katalogu podanego jako argument, do katalogu `$ARCHIVE_PATH/NOWA_NAZWA`
    e. `NOWA_NAZWA` powinna być utworzona wg schematu `<nazwa katalogu kopiowanego do archiwum>-YYYY-MM-DD-HH-SS` (do wygenerowania daty użyj polecenia `date`)

    Przykład:

    Na dysku jest katalog: `/home/student/Dokumenty/Praca_Magisterska`

    Uruchamiam skrypt: `./archive.sh /home/student/Dokumenty/Praca_Magisterska`

    Po wykonaniu skryptu w katalogu `$HOME/archive` pojawia się katalog `Praca_Magisterska-2011-12-11-10-20/` w którym znajduje się kopia wszystkich plików i katalogów z katalogu źródłowego.

5. Spróbuj przerobić program `projekt.sh` tak, aby skrypt nie miał zdefiniowane na sztywno struktury katalogów, ale zamiast tego wczytywał z pliku (linia po linii) nazwy katalogów jakie trzeba utworzyć. Tzn. użyj pliku `szablon_projektu`, który posiada w każdej linii ścieżkę katalogów tak jak w punkcie 3. d. Ścieżka do pliku szablonu powinna znajdować się w zmiennej `$PROJECT_TEMPLATE`. Jeśli zmienna jest nieustawiona należy szukać pliku ukrytego o nazwie `$HOME/.project_template`

Laboratorium 9. Kompresja i narzędzia sieciowe

Archiwizacja i kompresja danych

W systemie Linux istnieje wiele poleceń związanych z archiwizacją oraz kompresją danych. Programy archiwizujące służą do przygotowania struktury plików wraz z katalogami oraz plikami, w celu umieszczenia ich na zewnętrznym nośniku danych. Programem służącym do tego celu jest program **tar**, który jest jednym z najczęściej wykorzystywanych programów tego typu w świecie systemów Linux i Unix. Aktualne wersje tego programu potrafią wykonywać także kompresję danych podczas archiwizacji plików.

Programy służące do kompresji najczęściej spotykane w środowisku Linux bazują na algorytmach opartych o popularny algorytm ZIP a dokładnie na algorytmach:

- LZ77
- kodowaniu Huffman'a
- algorytmie DEFLATE oraz jego kombinacjach.

Oprócz wspomnianych algorytmów, na platformę systemu Linux, przeniesione zostały także inne aplikacje służące do kompresji takie jak: LHA, RAR czy ACE.

Program	Algorytm
gzip	algorytm Deflate, stanowiący kombinację algorytmów LZ77 oraz algorytmu kodowania Huffman'a
bzip2	transformacja Burrowsa-Wheelera, algorytm Move To front oraz algorytm kodowania Huffman'a
zcat, compress	algorytm Lempel-Ziv-Welch (LZW) będący rozszerzeniem algorytmu LZ78
zip	algorytm Deflate
zoo	algorytm Lempel-Ziv-Welch (LZW)
lha	Deflate algorytm oraz w zależności od wersji różne formaty algorytmów Lempel-Ziv
arj	algorytm Lempel-Ziv-Storer-Szymanski (LZSS) będący rozszerzeniem algorytmu LZ77
7z	algorytm Lempel-Ziv-Markov chain-Algorithm (LZMA)
rar	algorytm Lempel-Ziv-Storer-Szymanski (LZSS)

Polecenie `tar`

Podstawowym narzędziem do archiwizacji danych w systemach Unix oraz Linux jest program tar (*ang. Tape Archiviser*). Obecnie znacznie częściej wykorzystywany jest w celu wykonywania archiwów o rozszerzeniach *.tar, pierwotnie został jednak zaprojektowany dla potrzeb obsługi archiwów na taśmach magnetycznych. Jego działanie polega na sklejeniu do jednego pliku dowolnej struktury katalogów i plików. W ten sposób łatwiej zarchiwizować dane, a także skompresować je jako jeden plik.

Najważniejsze opcje polecenia `tar` to:

- `c`, `--create` – utwórz archiwum;
- `x`, `--extract`, `--get` – rozpakuj zawartość archiwum do bieżącego katalogu,
- `t`, `--list` – wypisz zawartość archiwum;
- `f`, `--file target` – nazwa urządzenia lub pliku do którego zostanie zapisane tworzone archiwum. Domyślnie tar próbuje utworzyć archiwum na urządzeniu taśmowym np. /dev/rmt/0m (HP-UX); tar w systemie Linux, kieruje strumień danych na standardowe wyjście.
- `v`, `--verbose` – wyświetla na ekranie informacje na temat procesu tworzenia lub rozpakowywania archiwum.
- `C`, `--directory dir1 dir2` – przejście do katalogu `dir1` przed spakowaniem zawartości katalogu `dir2` podanego jako następny argument opcji. Oczywiste jest, że `dir2` jest podkatalogiem `dir1`.
- `z`, `--gzip`, `--ungzip`; `Z`, `--compress`, `--uncompress` -- dostępne tylko w niektórych implementacjach opcje pozwalające natychmiast po utworzeniu archiwum od razu skompresować go dostępnymi w systemie programami `gzip` i `compress` (lub zdekompresować przed rozpakowaniem archiwum). Jeśli archiwum zapisywane jest do pliku to zgodnie z przyjętą konwencją, jego rozszerzenie powinno mieć postać `*.tar.gz`/`*.tgz` w przypadku kompresji programem `gzip` i `*.tar.Z`, gdy użyjemy programu `compress`.

Przykładowe polecenie tworzące archiwum w pliku tar zawierające kilka podkatalogów z katalogu bieżącego ma następującą postać:

```
$ tar -cvf archiwum.tar kataog1 katalog2 katalog3
```

Podczas tworzenia archiwum wyświetlana będzie lista pakowanych plików z ścieżkami względem katalogu bieżącego.

Dzięki opcji `-C` możliwe jest przemieszczanie się po drzewie katalogów i włączanie do archiwum zawartości podkatalogów leżących w różnych miejscach drzewa katalogów np.

```
$ tar -cvf archiwum.tar -C /home/user1 katalog1 -C /home/user2 katalog2
```

Powyższe polecenie umieści w jednym archiwum o nazwie `archiwum.tar`, zawartość podkatalogów `katalog1` i `katalog2`, znajdujących się w różnych poddrzewach katalogu /.

Możliwe jest używanie również ścieżek względnych, należy jednak pamiętać, że kolejne przejścia odbywają się względem katalogu w którym poprzednio odbywało się przetwarzanie polecenia tar.

W liście argumentów polecenia można używać metaznaków *ang. wildcards* `*` i `?` itd.

Archiwum `*.tar` rozpakowujemy do bieżącego katalogu poleceniem:

```
$ tar -xvf archiwum.tar
```

Dodając opcję `z` (lub `Z` do każdego z powyższych poleceń możemy tworzyć, przeglądać i rozpakowywać archiwa skompresowane.

```
$ tar -xzvf archiwum.tar.gz
```

Polecenie `gzip`

Program `gzip` służy wyłącznie do kompresji plików, a nie do ich archiwizacji. Program `gzip` nie potrafi kompresować całych struktur katalogowych, czy też umieszczać w jednym zakodowanym pliku wiele plików. Poleceniem realizującym te zadania jest program tar, który bardzo często wykorzystywany jest wraz z programem `gzip`.

Składnia polecenia `gzip`:

gzip [-acdfhlLnNrtvV19] [-S rozszerzenie] [nazwa ...]

opcje polecenia:

- `-c` – zapisuje wynik na standardowym wyjściu,
- `-d` – rozpakowuje archiwum,
- `-f` – wymuszenie zapisu pliku, nawet jeżeli istnieje oraz kompresja linków,
- `-l` – wyświetla zawartość archiwum,
- `-n` – nie zapisuje nazwy pliku oraz praw pliku,
- `-N` – zapisuje nazwę pliku oraz uprawnienia,
- `-q` – cichy tryb, nie wyświetla żadnych informacji,
- `-r` – wszystkie operacje wykonywane są rekurencyjnie na podkatalogach,
- `-S` – dodaje przyrostek do pliku,
- `-t` – testuje poprawność archiwum,
- `-1` – szybki tryb kompresji,
- `-9` – najlepszy tryb kompresji.

W celu skompresowania plików należy wydać polecenie `gzip` wraz z nazwą pliku. Program skompresuje dane zawarte w tym pliku oraz doda dodatkowe rozszerzenie (suffix) `.gz` do nazwy pliku.

```
$ gzip plik.txt
```

Polecenie to utworzy plik `plik.txt.gz`, który zastąpi plik `plik.txt` skompresowaną wersją tego pliku.

Każdy plik może być skompresowany z użyciem dwóch trybów pracy algorytmu realizującego szybką kompresję danych (przełącznik `-1`) oraz realizującego najlepszy stopień kompresji danych (przełącznik `-9`).

```
$ gzip -1 plik.txt
```

```
$ gzip -9 plik.txt
```

Jeżeli użytkownik chce aby dane zawarte pliku nie były umieszczane po kompresji w pliku z dodanym sufiksem `.gz` może skorzystać z przełącznika `-c`, powodującego, że wynik po kompresji jest wyświetlany na standardowym wyjściu. Metoda ta wykorzystywana jest do zapisu.

```
$ gzip -c plik.txt > plik.txt.gz
```

W celu dekompresji pliku za pomocą programu `gzip` można użyć np. przełącznika `-d`. Wówczas program `gzip` dekompresuje podany jako argument plik (z rozszerzeniem `.gz`) oraz automatycznie usunie sufiks `.gz` z nazwy pliku.

```
$ gzip -d test.gz
```

Programy sieciowe

Linux umożliwia prace zdalną na innych maszynach opartych o system Linux na za pomocą zdalnego shell'a (remote shell).

Telnet

Telnet to standard protokołu komunikacyjnego używanego w sieciach komputerowych do obsługi zdalnego terminala w architekturze klient-serwer.

Protokół obsługuje tylko terminale alfanumeryczne, co oznacza, że nie obsługuje myszy ani innych urządzeń wskazujących. Nie obsługuje także graficznych interfejsów użytkownika. Wszystkie polecenia muszą być wprowadzane w trybie znakowym w wierszu poleceń. Polecenia wydawane za pomocą naszego komputera przysłane są poprzez sieć do serwera, na którym zainstalowane jest oprogramowanie serwera telnetu. W odpowiedzi serwer odsyła nam komunikaty, które następnie wyświetlane są na naszym ekranie.

Do korzystania z tej usługi niezbędne jest posiadanie na serwerze konta typu shell. Na niektórych serwerach administratorzy zakładają konta gościnne, które nie wymagają podania hasła, lub hasła te są publicznie podawane. Jednakże ze względów nadmiernego wykorzystywania takich kont, administratorzy wprowadzili możliwość używania tylko potrzebnych aplikacji, które mogą być wykorzystane na danym serwerze. Po połączeniu się z serwerem, na którym posiadamy konto shell program zapyta nas o identyfikator użytkownika (login) i hasło dostępu (password). Usługa Telnet umożliwia zatem pracę na zdalnym komputerze bez konieczności siedzenia bezpośrednio przed nim. Uruchomienie tej usługi wykonuje się poprzez wpisanie polecenia:

```
telnet host [port]
```

gdzie `host` jest adresem IP komputera, z którym chcemy się połączyć, bądź jego nazwą domenową, gdyż telnet dopuszcza obie te formy podawania adresu. Po nawiązaniu połączenia telnet wyświetli nam informację o wersji systemu operacyjnego serwera, jego nazwie oraz numerze wirtualnego terminala (np. ttyp0, ttyp1, ttyp2, itd...) Następnym krokiem jaki musimy wykonać w celu zalogowania się do serwera to podanie użytkownika oraz hasła (login i password).

Połączenia telnetem są nieszyfrowane (co oznacza, że każdy może przechwycić nasz login i hasło do serwera oraz przebieg całej sesji) dlatego obecnie praktycznie zupełnie nie jest wykorzystywane (na rzecz ssh). Natomiast samo narzędzie telnet może być przydatne do otwierania połączenia z dowolnymi innymi serwerami posługującymi się protokołem tekstowym (jak np. HTTP).

```
$ telnet onet.pl 80

Trying 213.180.146.27...

Connected to onet.pl.

Escape character is '^]'.
```

SSH

SSH to następca protokołu Telnet, służącego do terminalowego łączenia się ze zdalnymi komputerami. SSH różni się od Telnetu tym, że transfer wszelkich danych jest zaszyfrowany oraz możliwe jest rozpoznawanie użytkownika na wiele różnych sposobów. W szerszym znaczeniu SSH to wspólna nazwa dla całej rodziny protokołów, nie tylko terminalowych, lecz także służących do przesyłania plików (SCP, SFTP), zdalnej kontroli zasobów, tunelowania i wielu innych zastosowań. Wspólną cechą wszystkich tych protokołów jest identyczna z SSH technika szyfrowania danych i rozpoznawania użytkownika. Scp nie rozpoznaje.

Uwierzytelnienie użytkownika może się opierać na haśle, kluczu (RSA, DSA) lub protokole Kerberos.

```
ssh [user@]host[:port]
```

Oprócz samego zalogowania się na zdalny serwer, polecenie ssh pozwala na wykonanie polecenia bezpośrednio na zdalnym serwerze (bez pozostawania zalogowanym na tym serwerze).

```
$ ssh user@zdalny mkdir test
```

utworzy na serwerze zdalnym katalog `test`.

Autentykacja kluczem prywatnym w SSH

SSH udostępnia niezwykle wygodny mechanizm autentykacji na podstawie klucza publicznego/prywatnego. Pozwala to na logowanie się za pomocą protokołu SSH na zdalny host bez podania hasła, gdy na komputerze lokalnym posiadamy specjalny klucz prywatny, a host zdalny wyposażony jest w komplementarny klucz publiczny.

Pierwszym krokiem do utworzenia autentykacji za pomocą kluczy jest wygenerowanie kluczy poleceniem:

```
$ ssh-keygen -t rsa
```

Na zadawane przez program pytania należy odpowiadać naciskając po prostu *Enter*, czyli wybierając wartości domyślne.

Plik `.ssh/id_rsa` jest naszym kluczem prywatnym - należy go chronić i nikomu nie udostępniać. Natomiast plik `.ssh/id_rsa.pub` jest kluczem publicznym, kopiujemy go na konto użytkownika zdalnego na dodajemy do pliku `.ssh/authorized_keys` - jeżeli tego pliku nie ma, musimy go stworzyć. Sekwencja komend do wykonania mogłaby wyglądać tak:

```
$ scp .ssh/id_rsa.pub user@zdalny:
$ ssh user@zdalny
$ mkdir -p .ssh
$ cat id_rsa.pub >> .ssh/authorized_keys
$ chmod og-w .ssh/authorized_keys
```

Uwaga: ostatnia linijka jest po to aby upewnić się że nikt inny nie będzie mógł pisać do naszego pliku `authorized_keys`. Serwer SSH uruchomiony w trybie STRICT wymaga tego i inaczej nie pozwoli nawiązać połączenia.

Po wykonaniu prawidłowo tych czynności każdorazowe logowanie się na zdalny serwer na użyty wcześniej login użytkownika powinien skutkować brakiem zapytania o hasło do konta.

Dzięki temu w sposób wygodny można:

- używać polecenia `scp` do kopiowania bezpiecznego plików pomiędzy maszyną zdalną i lokalną,
- używać polecenia `ssh` do bezpośredniego wykonywania poleceń na maszynie zdalnej.

Polecenie `scp`

Polecenie scp (secure copy) służy do bezpiecznego transferowania plików pomiędzy lokalnym a zdalnym lub między zdalnymi komputerami, używając protokołu Secure Shell (SSH). Skrót SCP odnosi się do dwóch powiązanych ze sobą rzeczy: protokół SCP oraz polecenie `scp`.

Przykłady użycia

Kopiowanie pliku ze zdalnej lokalizacji na lokalny dysk

```
$ scp uzytkownik@serwer.pl:/scieżka/plik_serwer plik_lokalny
```

Kopiowanie pliku z dysku lokalnego do zdalnej lokalizacji

```
$ scp plik_lokalny uzytkownik@serwer.pl:/sciezka/plik_serwer
```

Polecenie `ping`

Najczęściej używanym narzędziem diagnostycznym jest program `ping` - pozwala on zbadać istnienie połączenia między dwoma komputerami, drogę pomiędzy nimi, czas potrzebny na przejście pakietu oraz sprawdza czy drugi komputer pracuje w danym momencie w sieci. Przy

okazji `ping` dokonuje tłumaczenia adresu domenowego na numer IP. Program ten jest przydatny do określania stanu sieci i określonych hostów, śledzenia i usuwania problemów sprzętowych, testowania, mierzenia i zarządzania siecią, oraz do badania sieci. Polecenie

```
$ ping host
```

wysyła specjalne pakiety ICMP do wskazanego komputera i czeka na odpowiedź. Możemy podawać jako cel adres domenowy lub numer IP. Przy okazji program ten dokonuje tłumaczenia adresu domenowego na numer IP.

```
$ ping pld-linux.org

PING pld-linux.org (81.0.225.27) 56(84) bytes of data.

64 bytes from 81.0.225.27: icmp_seq=1 ttl=44 time=135 ms

64 bytes from 81.0.225.27: icmp_seq=2 ttl=44 time=99.8 ms

64 bytes from 81.0.225.27: icmp_seq=3 ttl=44 time=149 ms

--- pld-linux.org ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2001ms rtt min/avg/max/mdev = 99.840/128.084/149.144/20.761 ms
```

Polecenie `traceroute`

Nieco bardziej zaawansowanym programem jest `traceroute`. Pokazuje on trasę jaką przechodzą pakiety między naszym komputerem, a sprawdzanym przez nas hostem. Wskazuje on czasy przesłania pakietów pomiędzy sąsiadującymi ze sobą routerami (tzw. czasy przeskoków), znajdującymi się na trasie połączenia dwóch maszyn. Pozwala śledzić trasę pakietów oraz wykrywać różnego rodzaje problemy w sieciach np.: błądzenie pakietów w sieci, "wąskie gardła" sieci, oraz awarie połączeń.

```
$ traceroute pld-linux.org

traceroute to pld-linux.org (81.0.225.27), 30 hops max, 38 byte packets

1  192.168.1.1 (192.168.1.1)  0.295 ms  0.608 ms  0.484 ms

2  217.153.188.173 (217.153.188.173)  1.012 ms  0.648 ms  0.495 ms

3  217.8.190.153 (217.8.190.153)  30.894 ms  28.983 ms  29.719 ms
```

Polecenie `wget`

Polecenie `wget` jest najprostszym klientem HTTP (pozwala ściągać pliki z WWW na dysk) w systemie Linux. Mimo swojej prostoty, program oferuje niezwykle bogatą listę często bardzo zaawansowane funkcji (jak np. Symulowanie przechodzenia po linkach na stronach jak robot internetowy, czy wysyłanie danych GET/POST i COOKIES podczas zapytań HTTP.

Typowe wywołanie programu to:

```
$ wget http://serwer.pl/plik_do_pobrania.html
```

Wywołanie powyższego polecenia spowoduje ściągnięcie na dysk pliku `plik_do_pobrania.html`

```
$ wget -c http://ścieżka/do/pliku
```

pobiera plik z sieci, jeśli przerwiemy pobieranie a następnie wykonamy to samo polecenie, plik zostanie dociągnięty z miejsca, w którym skończyliśmy,

```
$ wget --limit-rate=10k http://ścieżka/do/pliku
```

ściągamy plik z maksymalną szybkością 10 kb na sekundę,

```
$ wget -r -l 1 http://ścieżka/do/strony
```

ściągamy rekursywnie całą stronę WWW, razem z obrazkami.

Polecenia i zadania do wykonania

1. Korzystając z polecenia `gzip` utwórz plik `Laboratorium.pdf.gz`, który zawierać będzie skompresowaną wersję tego dokumentu. Użyj najszybszej i najmocniejszej wersji kompresji. Policz stopień kompresji (ang. CR – compression rate) w obu przypadkach. Miarę CR liczymy ze wzoru:
   CR = [Rozmiar po skompresowaniu] / [Rozmiar przed skompresowaniem]
2. Wykorzystaj polecenie `tar`, aby utworzyć archiwum katalogu Desktop na komputerze w laboratorium. Archiwum skompresuj. Plik wynikowy powinien mieć nazwę Desktop.tar.gz
3. Skonfiguruj autentykację SSH za pomocą klucza publicznego do swojego konta na wierzbie (pamiętaj o ustawieniu odpowiednich uprawnień dla katalogu `.ssh` oraz `authorized_keys`).
4. Przegraj za pomocą `scp` plik Desktop.tar.gz na wierzbie i tam go rozpakuj.
5. Przegraj z wierzby dowolny plik na komputer lokalny z użyciem `scp`.
6. Napisz prosty skrypt o nazwie backuper.sh, który będzie robić i odtwarzać backupy. Skrypt powinien działać w następujący sposób.
   a. Plik backuper.sh umieść w katalogu `~/bin` na komputerze w laboratorium, ustaw odpowiednio ścieżkę `$PATH`, żeby dało się uruchamiać skrypt z dowolnego miejsca w systemie przez podanie tylko jego nazwy.
   b. Skrypt backuper.sh można uruchomić na trzy sposoby w zależności od wartości pierwszego parametru:
      i. `backuper.sh` **`create`** `[ścieżka]`
      ii. `backuper.sh` **`list`**
      iii. `backuper.sh` **`restore`** `[nazwa]`
   c. Oprócz tego podczas uruchamiania skryptu powinna być zdefiniowana zmienna środowiskowa postaci $BACKUP_PATH będącą pełną ścieżką na zdalnym serwerze (np. „`dorosz@wierzba.wzks.uj.edu.pl:~/backupy/`”) . Do tej ścieżki przesyłane zdalnie będą backupy.
   d. Uruchomienie skryptu z argumentem 1 **create** oraz argumentem 2 [ścieżka] powinno:
      i. po uruchomieniu sprawdzić czy podana ścieżka jest prawidłowa (plik lub katalog istnieje)
      ii. utworzyć archiwum tar.gz z podanej ścieżki. Archiwum powinno być utworzone w katalogu `/tmp`
      iii. nazwa archiwum powinna składać się z nazwy pliku lub katalogu podanego w ścieżce jako argument, a następnie z obecnej daty

zawierającej rok, miesiąc i dzień oraz rozszerzenie tar.gz. Np. `Dokumenty-2011-12-15.tar.gz`.

iv. Przesłać plik z archiwum za pomocą polecenia `scp` na zdalny serwer wskazany poprzez `$BACKUP_PATH`. (wskazówka, zmienna `$BACKUP_PATH` powinna zawierać całą zdalną ścieżkę, login, serwer i katalog, tak żeby można było użyć tej zmiennej bezpośrednio jako argument dla polecenia `scp`)

v. Usunąć plik tymczasowy z `/tmp`

e. Uruchomienie skryptu z argumentem 1 **list** powinno:

i. Wyświetlić listę plików (archiwów) znajdującą się na zdalnym serwerze wskazanym przez `$BACKUP_PATH` (użyj do tego polecenia ssh i wykonania komendy `ls` na serwerze zdalnym)

f. Uruchomienie skryptu z argumentem 1 **restore** i argumentem 2 nazwa archiwum powinno:

i. Pobrać archiwum ze zdalnego katalogu do katalogu lokalnego (w którym wywołano program backuper.sh)

ii. Rozpakować archiwum i usunąć go.

iii. W rezultacie wykonania polecania w katalogu bieżącym powinien znajdować się katalog (lub plik) taki jak pierwotnie zarchiwizowano odtworzony z pełną zawartością.

Przykładowa symulacja użycia skryptu:

```
/home/student/$ backuper.sh create ~/Desktop

Utworzono archiwum o nazwie Desktop-2011-12-15.tar.gz i wysłano na serwer zdalny dorosz@wierzba.wzks.uj.edu.pl:~/moje_backupy/

/home/student/$ backuper.sh list

Na serwerze zdalnym dorosz@wierzba.wzks.uj.edu.pl:~/moje_backupy/ znajdują się następujące backupy:

Desktop-2011-08-01.tar.gz

Desktop-2011-09-13.tar.gz

Desktop-2011-12-15.tar.gz

/home/student/$ mkdir test; cd test

/home/student/test$ backuper.sh restore Desktop-2011-12-15.tar.gz

Odtwarzam backup Desktop-2011-12-15.tar.gz
```

(W katalogu test znajduje się teraz katalog Desktop, który ma odpowiednią zawartość)

7. UWAGA: po zakończeniu zajęć usuń autoryzację klucza SSH z serwera wierzba (edytuj lub usuń plik `~/.ssh/authorized_keys`), ponieważ w innym przypadku umożliwisz innym osobom siedzącym na tym komputerze w laboratorium dostęp do swojego konta na wierzbie.

Dodatkowe tematy

Zainteresuj się poleceniem `rsync` – umożliwia synchronizację dokumentów (różnicową) pomiędzy hostami.

Laboratorium 10. Cykliczne uruchamianie programów oraz kontrola wersji dokumentów tekstowych

Podczas tego laboratorium opanujesz zasadę działania polecenia at oraz cron oraz jego pliku konfiguracyjnego crontab, który umożliwia cykliczne uruchamianie programów o zaplanowanej porze. Następnie zapoznasz się z ideą systemów kontroli wersji (VCS – Version Control System) na przykładzie popularnego systemu SVN.

Polecenie at

Polecenie `at` jest prostym programem pozwalającym na zaplanowaniu pojedynczego zadania do uruchomienia w odroczonym czasie. Polecenie `at` uruchamia się standardowo w następujący sposób:

```
at -f  plik_z_zadaniem czas_uruchomienia
```

lub

```
at czas_uruchomienia
```

W drugim przypadku należy polecenie do wykonania przekazać jako standardowe wejście. W pierwszym przypadku powinno ono znajdować się jako zawartość pliku.

Czas uruchomienia może być jednym ze słów specjalnych: `midnight` (północ), `noon` (południe), `today` (dziś), `tomorrow` (jutro) oraz `now` (teraz) lub może mieć np. taki format:

```
at 14:12 January 9
```

```
at 2:12am Jan 9
```

```
at now + 5 minutes
```

Polecenie crontab

`Crontab` jest to tabela konfiguracyjna dla programu `cron`. Program `cron` jest demonem systemowym (nie uruchamiasz go samodzielnie, działa w tle w systemie), który umożliwia uruchamianie zadań (wywołań shella) o zadanych porach.

`Crona` konfiguruje się przy pomocy polecenia `crontab`, które otwiera edytor z plikiem konfiguracyjnym `crontab`. Typowe Parametry uruchomienia polecenia to:

```
crontab -l | -r | -e
```

Opcja	Opis
-l	Listuje zadania
-r	Usuwa zadania z wykonania
-e	Otwiera do edycji plik konfiguracyjny crontab

Składnia pliku konfiguracyjnego cronab jest następująca:

```
min godz dzień_miesiąca miesiąc dzień_tygodnia zadanie
```

Każda linia pliku zawiera jedno zadanie, które składa się z 6 segmentów oddzielonych dowolnym białym znakiem typu spacja lub tabulacja. Każdy segment definiuje inną właściwość zadania. Odpowiednio są to:

- Minuta w której należy uruchomić zadanie,
- Godzina, w której należy uruchomić zadanie,
- Dzień miesiąca, w którym należy uruchomić zadanie,
- Miesiąc, w którym należy uruchomić zadanie,
- Filtr dnia tygodnia, w którym należy uruchomić zadanie,
- Wywołanie polecenia, które ma się wykonać w ramach zadania.

W każdym z 5 pierwszych pół wprowadzić można następujące wyrażenia:

- Znak * , oznacza wszystkie możliwe wartości
- Liczba, oznacza konkretną wartość (dzień, miesiąc, itp.)
- Wylistowanie za pomocą przecinka, oznacza serie wartości (np. 1,2,3)
- Zakres za pomocą myślnika, (np. 1-3)
- Operator „co n jednostek", */n oznacza że zadanie wykona się co n-tą wartość (np. */2, oznacza wykonaj co drugi raz, */3 wykonaj co trzeci raz, itp.)

Przykładowo, tak zdefiniowane zadanie wykona się co minutę:

```
* * * * /ścieżka/do/polecenia argumenty
```

Taki zapis oznacza, że polecenie ma się wykonywać w każdej możliwej minucie (pierwsza *), w każdej możliwej godzinie (druga *), w każdy możliwy dzień (trzecia *), itd..

Wykonanie zadania co 5 minut można zapisać tak:

```
*/10 * * * /ścieżka/do/polecenia argumenty
```

lub tak

```
0,10,20,30,40,50 * * * /ścieżka/do/polecenia argumenty
```

zauważ że w pierwszej opcji użyto operatora `/` a w drugiej wylistowano po prostu 6 równych momentów w czasie kiedy należy uruchomić polecenie.

Jeśli chcesz wykonać polecenie co 5 minut ale tylko raz w miesiącu należy napisać:

```
*/10 * 1 * /ścieżka/do/polecenia argumenty
```

W ten sposób zawsze w styczniu (1. miesiąc) polecenie będzie wykonywane przez cały ten miesiąc co 10 min non stop (w każdej możliwej godzinie!).

Można także określić szczegółowo dni tygodnia. Np. jeśli chcemy robić backup na serwerze w każdą sobotę o godzinie 3:00 w nocy:

```
0 3 * * 6 /ścieżka/do/polecenia argumenty
```

Dni tygodnia zdefiniowane są licząc od 0 do 6, z tym że liczenie rozpoczyna się od niedzieli. Dlatego sobota ma numer 6. Zwróć uwagę że polecenie bez zdefiniowania dnia tygodnia

uruchamiało by się po prostu codziennie o 3:00, natomiast dołożenie warunku o dniu tygodnia równym 6 powoduje, że dzień tygodnia musi być sobotą aby polecenie było uruchomione.

Polecenie svn

Systemy kontroli wersji służą do zarządzania zmianami w czasie zawartości plików. Są bardzo wygodne w pracy deweloperów kodu (ale i nie tylko) ponieważ umożliwiają nie tylko powrót do historycznych wersji dokumentów (na wypadek gdyby ktoś wprowadził nieodpowiednie zmiany i trzeba było przywrócić stare wersje) ale także umożliwiają na synchronizację pracy wielu osób nad jednym zestawem plików.

System SVN jest tylko jednym z wielu systemów kontroli wersji. Starszym systemem tego typu obecnie praktycznie już nie używany jest CSV. Nowszymi systemami są mercurial (hg) oraz git. Te nowsze systemy są obecnie bardzo popularne, ale są to system DVCS (Distributed Version Control System) i wprowadzają dodatkowy poziom abstrakcji związany z rozproszeniem repozytorium głównego. Dlatego prościej najpierw jest zrozumieć idee działania prostego systemu jakim jest SVN z centralnym repozytorium, a potem ewentualnie zacząć korzystać z hg lub git.

Aby skorzystać z możliwości jakie daje SVN najpierw należy utworzyć gdzieś centralne repozytorium kodu do którego będą synchronizowane wszystkie zmiany. Służy do tego polecenie `svnadmin`:

```
svnadmin create
```

Do obsługi reszty czynności służy polecenie `svn`. Polecenie `svn` posiada wygodny system pomocy uruchamiany jako

```
svn help <nazwa komendy>
```

Takie repozytorium może utworzyć sobie w dowolnym katalogu każdy użytkownik. Jeśli utworzysz go na serwerze zdalnym w swoim katalogu domowym, będziesz mógł uzyskać dostęp do niego przy użyciu protokołu ssh.

Aby dodać dowolną strukturę drzewa katalogów i pliku do tego repozytorium należy użyć polecenia svn z opcją import:

```
svn import
```

Aby móc korzystać z wersjonowanej wersji katalogu na zdalnym serwerze należy wykonać checkout katalogu z repozytorium do katalogu na lokalnej maszynie:

```
svn checkout
```

Jeśli chcesz dodać nowy plik do repozytorium należy wykonać

```
svn add
```

Usuwanie plików wykonujemy za pomocą delete

```
svn delete
```

uwaga – nie usuwaj plików ręcznie za pomocą polecenia `rm` w systemie ponieważ SVN nie będzie umiał wykryć takiej sytuacji i zgłosi trudny do usunięcia błąd struktury repozytorium.

Aby zgłosić zmiany wykonane na plikach i katalogach uzyj opcji commit z obowiązkowym argumentem –m „opis” podający opis wykonanych zmian

```
svn commit –m „opis zmian”
```

W każdej chwili można podejrzeć informacje o repozytorium

```
svn info
```

Oraz informacje o różnicach w plikach

```
svn diff
```

Oraz status plików

```
svn status
```

Polecenia do wykonania na zajęciach

1. Utwórz skrypt tester.sh, który posłuży ci do testowania cyklicznego wywołania. Jego działanie powinno polegać na wyświetlaniu na stdout napisu „[data] ping” (gdzie data powinna być aktualną datą w momencie wyświetlania napisu).
2. Przetestuj działanie polecenia at planując uruchomić skrypt tester.sh za 2 min od chwili obecnej zapisując wynik działania do pliku /tmp/test1. Sprawdz po 2 min. czy w pliku /tmp/test1 znajduje się ping.
3. Spróbuj skonfigurować crontab w taki sposób, aby program tester.sh uruchomiony co 5 min i zapisywał wyjście swoich wywołań do pliku /tmp/test2. Pod koniec zajęć sprawdź czy w pliku znajdują się pingi wykonywane co 5 min.
4. Skonfiguruj na czas tego laboratorium autoryzację po kluczu publicznym z serwerem wierzba.
5. Załóż na wierzbie nowe repozytorium SVN w katalogu ~/svn
6. Na komputerze w laboratorium załóż katalog „~/projekt” i wykonaj import do twojego repozytorium SVN na wierzbie.
7. Następnie wykonaj checkout projektu do katalogu „~/projekt-kopia1”.
8. Do katalogu projekt-kopi1 dodaj plik test.txt. Dodaj go do repozytorium svn i wykonaj commit zmian.
9. Zasymuluj działanie drugiego użytkownika w systemie. Spróbuj teraz wykonać checkout projektu z SVN do drugiego innego katalogu „~/projekt-kopia2”. Sprawdź, że znajduje się tam plik test.txt utworzony przez „pierwszego użytkownika”.
10. Będac w katalogu „~/projekt-kopia2” wykonaj teraz modyfikację pliku test.txt dodając do niego 10 dowolnych linijek tekstu.
11. Wykonaj commit do SVN z katalogu ~/projekt-kopia2
12. Wróć do katalogu ~/projekt-kopia2. Sprawdź zawartość pliku test.txt. Plik jest nadal pusty. Aby ściągnąć zmiany jakie zostały zgłoszone przez innych użytkowników musisz wykonać update SVN.
13. Po wykonaniu polecenia update sprawdź zawartość pliku test.txt. Powinno być tam 10 linijek wpisanych przez „użytkownika 2”.
14. Możesz teraz sprawdzić za pomocą log historie zmian rewizji dokumentu.
15. Wykonasz teraz symulację sytuacji, w której dwie osoby jednocześnie edytują i zgłaszają ten sam plik.

16. W katalogu ~/projekt-kopia1 dopisz coś na koniec **pierwszej** linii. Użyj polecenia diff aby przekonać się jaki fragment pliku został zmieniony. Użyj polecenia status aby przekonać się, że plik jest oznaczony jako zmieniony („M”).
17. W katalogu ~/projekt-kopia2 dopisz coś na koniec **ostatniej** linii. Użyj polecenia diff aby przekonać się jaki fragment pliku został zmieniony. Użyj polecenia status aby przekonać się ,że plik jest oznaczony jako zmieniony („M”).
18. Wykonaj teraz commit z katalogu ~/projekt-kopia1. To polecenie powinno się udać.
19. Wykonaj teraz commit z katalogu ~/projekt-kopia2. SVN nie pozwoli wprowadzić zmian, ponieważ ktoś już edytował ten plik.
20. Wykonaj update SVN w katalogu ~/projekt-kopia2 i przekonaj się, że ponieważ SVN domyślił się że edytowane przez kogoś sekcje pliku (pierwsza linia) nie pokrywają się z Twoimi zmianami, połączył obie zmiany w jednym pliku automatycznie. Twój plik zawiera teraz zarówno zmiany „użytkownika 1” jak i „użytkownika 2”.
21. Możesz teraz wykonać commit zmian z katalogu ~/projekt-kopia2 aby zgłosić je także do repozytorium.
22. Wykonaj update w katalogu ~/projekt-kopia1 aby przekonać się, że zmiany użytkownika 2 są także widoczne dla użytkownika 1.
23. Teraz zasymulujesz sytuację konfliktu. Obaj użytkownicy zmodyfikują ten sam plik w tym samym miejscu.
24. W katalogu ~/projekt-kopia1 dopisz coś na koniec **piątej** linii. Użyj polecenia diff aby przekonać się jaki fragment pliku został zmieniony.
25. W katalogu ~/projekt-kopia2 dopisz coś na koniec **piątej** linii. Użyj polecenia diff aby przekonać się jaki fragment pliku został zmieniony.
26. Wykonaj teraz commit z katalogu ~/projekt-kopia1. To polecenie powinno się udać.
27. Wykonaj update SVN w katalogu ~/projekt-kopia2 i przekonaj się, że SVN wykrył konflikt.
28. Rozwiąż konflikt porównując obie zmiany w pliku tekstowym i ostatecznie zostawiając zmiany użytkownika 2.