

## Cwiczenie 2

Cel cwiczenia: Poznanie mechanizmów wejścia/wyjścia, zapoznanie się ze sposobami wyświetlania plików tekstowych i wyszukiwania informacji, podstawowe operacje na plikach tekstowych, zmienne środowiskowe.

### Wykorzystane polecenia:

**more plik** - wyświetla tekst zajmujący więcej niż jeden ekran  
**less plik** - podobnie jak **more**, lecz z możliwością przewijania tekstu  
**tail plik** - wyświetla ostatni fragment pliku tekstowego (standardowo 10 linii)  
**grep tekst plik** - wypisuje wiersz pliku zawierający podany fragment tekstu  
**head plik** - podobnie jak **tail**, lecz wyświetla początek  
**chmod [opcje] plik** - zmienia prawa dostępu do pliku  
**chown użytkownik.grupa plik** - zmienia właściciela pliku  
**grep** - szuka w pliku podany fragment tekstu i wyświetla linie zawierające ten fragment  
**echo komunikat** - wyświetla komunikat na ekranie  
**set** - wyświetla zmienne środowiskowe oraz ich wartości  
**ps** - wyświetla procesy obecne w systemie  
**top** - wyświetla procesy CPU  
**cal** - wyświetla kalendarz  
**find** - szuka pliki według podanego wzorca  
**kill** - wysyła odpowiedni sygnał do procesu  
**&** - dodane na końcu polecenia uruchamia proces w tle  
**fg** - przenosi proces na pierwszy plan  
**unset** - czyszczy wartość zmiennej, nie usuwając jej  
**readonly** - ustawia wartość zmiennej tylko do odczytu  
**\$?** - zwraca kod zakończenia polecenia pierwszoplanowego

### Procesy.

Proces to inaczej mówiąc program działający w systemie. W systemie Linux możliwe jest uruchomienie wielu procesów, procesy mogą być uruchomione w tle, mogą też posiadać priorytety dzięki którym zyskują odpowiednio dużo czasu procesora na swoje działanie.

Do wyświetlenia procesów działających w systemie służy polecenie **ps**.

**ps** - wyświetla procesy użytkownika

**ps -a** - wyświetla procesy uruchomione na wszystkich konsolach

**ps -ax** - wyświetla procesy na wszystkich konsolach łącznie z procesami ukrytymi.

W pierwszej kolumnie wyświetlany jest identyfikator procesu (pid)

Aby zakończyć proces można użyć polecenia **kill**, które wysyła odpowiedni sygnał do procesu. Polecenia używa się następująco:

**kill pid**

**kill sygnał pid**

jako sygnał można podać numer sygnału lub jego nazwę. Aby uzyskać pełną nazwę sygnałów wraz z nazwami należy użyć polecenia

```
kill -1
```

Polecenie kill z sygnałem numer 9 nie może zostać zignorowane przez zadany proces. Domyślnie jest wysyłany sygnał TERM (15) nakazujący procesowi zapisanie danych oraz zakończenie pracy.

Uruchomienie procesu (programu) w tle możliwe jest przez dodanie na końcu linii znaku &. Po uruchomieniu polecenia w tle powłoka macierzysta nie oczekuje na zakończenie polecenia, a zwracany kod wykonanego polecenia wynosi 0.<sup>1</sup> Po uruchomieniu programu w tle otrzymujemy informacje o numerze zadania (w nawiasach kwadratowych) oraz identyfikator procesu. Aby przenieść zadanie na pierwszy plan należy użyć polecenia

```
fg numer_zadania
```

### **Kod zakończenia \$?**

Polecenie, które zakończyło się powodzeniem zwraca zerowy kod zakończenia. Niezerowy kod oznacza niepowodzenie. Kod zakończenia jest większy od zera, jeśli polecenie nie powiodło się z powodu błędu podczas interpretacji lub przekierowania. Polecenia wbudowane zwracają kod 2 dla wskazania niepoprawnego użycia. Jeśli polecenie nie zostało znalezione, proces potomny powołany do jego wykonania zwraca kod 127. Jeśli zaś polecenie zostało znalezione, ale nie jest wykonywalne, kod zakończenia jest równy 126. Jeśli polecenie skutkuje błędem krytycznym poziomu N, powłoka przyznaje wartość 128+N jako kod zakończenia.

### **Zmienne.**

W systemie Linux można tworzyć zmienne, które później mogą być wykorzystane w poleceniach lub skryptach. Ustanowienie wartości zmiennej uzyskuje się przez proste przypisanie:

```
ZMIENNA="Na przykład tekst"
```

aby odwołać się do zmiennej należy jej nazwę poprzedzić znakiem \$. Na przykład

```
echo $ZMIENNA
```

Przyjęło się, że nazwy zmiennych pisane są dużymi literami. Zmiennych można używać w linii poleceń jako parametry, opcje. Wtedy zamiast zmiennej zostanie podstawiona jej wartość.

Wydając polecenie:

```
unset ZMIENNA
```

wyzerujemy jej wartość. Natomiast

```
readonly ZMIENNA
```

spowoduje, że wartość zmiennej będzie tylko do odczytu i nie będzie można jej zmienić.

---

<sup>1</sup> Pamiętajmy, że polecenia oddzielane znakiem średnika ; wykonywane są sekwencyjnie, tj. powłoka oczekuje na zakończenie pracy każdego z nich po kolei. Zwracany kod jest kodem zakończenia ostatniego wykonanego polecenia.

## **Uruchamianie programów.**

Do uruchomienia pliku znajdującego się w dowolnej lokalizacji określonej w zmiennej środowiskowej PATH wystarczy podanie jego nazwy.

Jeśli chcemy uruchomić jakiś program lub polecenie, a nie ma do niego ustalonej ścieżki w zmiennej środowiskowej PATH to należy podać całą ścieżkę dostępu do pliku. Nawet jeśli znajdujemy się w katalogu, w którym znajduje się program, który chcielibyśmy uruchomić należy przed nazwą podać symbol ścieżki bieżącej, czyli:

```
./program
```

W jednej linii komend można wydać kilka poleceń od razu. Należy je wtedy rozdzielić od siebie średnikiem. Na przykład:

```
ls *s2* ; mkdir dokum ; cp /home/kat1/* ./ ; rm -f /home/kat/*
```

## **Standardowe wyjście i wejście.**

Z każdym poleceniem systemu Linux jest związane standardowe wyjście (stdout), standardowe wejście (stdin) oraz standardowe wyjście błędów (stderr). Standardowe wejście to źródło pobierania danych, wyjście to miejsce wyświetlania wyników, wyjście błędów - miejsce wyświetlania komunikatów o wykonaniu polecenia.

Standardowe wyjście/wejście może być zmienione za pomocą pewnych symboli:

Oto skierowanie standardowego wyjścia do pliku w dwóch równoważnych postaciach:

```
>plik
```

```
1>plik - alternatywna postać z jawnym podaniem deskryptora
```

Skierowanie standardowego wyjścia na koniec pliku (dopisanie):

```
>>plik
```

Skierowanie komunikatów o błędach do pliku:

```
2>plik
```

Skierowanie komunikatów o błędach na standardowe wyjście:

```
2>&1
```

```
2>/dev/stdout
```

Powiązanie standardowego wejścia z plikiem (pobranie danych z pliku):

```
<plik
```

```
0<plik - alternatywna postać z jawnym podaniem deskryptora,
```

```
/dev/stdin<plik - alternatywna postać z wyspecyfikowaniem pliku
```

Zauważ, że kolejność przekierowań jest znacząca. Na przykład

```
ls > plik 2>&1
```

skieruje zarówno standardowe wyjście jak i wyjście błędów do pliku plik, zaś

```
ls 2>&1 > plik
```

skieruje tylko standardowe wyjście do pliku plik, ponieważ wyjście błędu jest zduplikowane jako standardowe wyjście przed przekierowaniem do pliku.

Bash w sposób specjalny traktuje niektóre pliki:

```
/dev/stdin - duplikuje deskryptor pliku 0
/dev/stdout - duplikuje deskryptor pliku 1
/dev/stderr - duplikuje jest deskryptor pliku 2
/dev/fd/fd - oznacza deskryptor pliku fd, gdzie fd jest poprawna liczba całkowita
/dev/tcp/host/port - bash otwiera połączenie TCP do odpowiedniego gniazda, gdzie host
jest poprawna nazwa hosta lub adresem internetowym, a port jest liczba całkowita
okreslająca numer portu
/dev/udp/host/port - jak wyżej, tylko że otwiera połączenie UDP.
```

Dzięki możliwości przenoszenia standardowego wyjścia i wejścia możliwe jest przetwarzanie potokowe. Wyniki działania jednego polecenia mogą być źródłem danych dla kolejnego. Na przykład:

```
ls | less
```

spowoduje wyświetlenie listy plików w programie less. Przetwarzanie potokowe można wykorzystać w programach, które nie potrzebują pliku jako swojego parametru

### **Prawa dostępu do pliku.**

Do zmiany praw dostępu do pliku służy polecenie `chmod`. Jego format to:  
**chmod** [ugo][+-][xwr] *plik*

Jako parametry używane są litery:

- u - właściciel pliku
- g - grupa
- o - inni
- x - prawo do wykonywania
- w - prawo do zapisu
- r - prawo do odczytu
- + - nadaje uprawnienia
- - zdejmuję uprawnienia

Na przykład:

```
chmod ug+xr-w plik2
```

spowoduje nadanie właścicielowi pliku oraz grupie prawo do wykonania oraz odczytu pliku. Zdejmie natomiast dla nich prawo do zapisu.

Uprawnienia można też nadawać podając odpowiednią liczbę ósemkowo. Na przykład

```
chmod 777 plik2
```

Spowoduje nadanie pełnych uprawnień dla wszystkich. każda cyfra określa jedną grupę uprawnień *rwx* kolejno dla właściciela, grupy i innych. Każda z liter ma następującą wagę:

r - 4

w - 2

x - 1

$$rwx = 4+2+1=7$$

Uprawnienia *rwxr-x---* będą przedstawione w postaci liczbowej jako 750

#### Cwiczenia.

1. Przeczytaj pomoc man dla wszystkich poleceń wykorzystanych w ćwiczeniu.
2. Wyświetl plik */etc/inetd.conf*
3. Wyświetl wszystkie procesy obecne w systemie (polecenie *ps*). Jakie parametry są potrzebne do takiego wyświetlenia.
4. Utwórz w katalogu domowym pliki o nazwie *informacja*, wprowadz do niego swoją nazwę użytkownika. Ustaw prawa do odczytu i zapisu tylko dla właściciela pliku. Spróbuj odczytać pliki *informacja* z katalogów domowych innych użytkowników. Jaki skutek?
5. Ustaw prawo do odczytu i zapisu pliku *informacja* dla wszystkich za pomocą zapisu ósemkowego.
6. Utwórz pliki o nazwach *studia*, *informatyka*, *linux*. Dla pierwszego pliku ustaw prawa na *rw-r--r--* dla drugiego na *rw-rw-rw* dla trzeciego *r--r-----*. Wykonaj to za pomocą polecenia *chmod* i parametrów tekstowych.
7. Utwórz pliki o nazwach *politechnika*, *czestochowa*. Dla pierwszego pliku ustaw prawa na *rw-rw-r--* dla drugiego na *rw-r--r--* dla trzeciego *r--r--r--*. Wykonaj to za pomocą polecenia *chmod* i parametru w postaci liczby ósemkowej.
8. Utwórz katalog o nazwie *cwiczenie2*, przenieś do niego pliki z powyższych dwóch punktów. Zmień prawa dostępu dla katalogu na *rw-----*. Czy jesteś w stanie przejrzeć zawartość katalogu?
9. Dodaj uprawnienia do wykonywania dla katalogu *cwiczenie2*. Jednak przy poleceniu *chmod* ominij opcje dotyczące użytkowników. Jaki efekt?
10. Postaraj się zmienić uprawnienia dla wszystkich plików w katalogu *cwiczenie2* jednocześnie. Czy udało się to zrobić?
11. Wykonaj następujące polecenie *>stud*. Jaki dało efekt?
12. Spróbuj zmienić właściciela katalogu */sbin* na *student*. Jaki efekt? Jeśli był jakiś komunikat o błędzie wyprowadź go do pliku *err*
13. Utwórz plik o nazwie *wlasciciel* w swoim katalogu. Zmień właściciela na *root*. Czy operacje zmiany właściciela może wykonywać każdy użytkownik?
14. Wyświetl pierwsze 7 linii pliku */etc/services*
15. Wyświetl ostatnie 15 linii pliku */etc/services*
16. Pokaż wszystkie linie zawierające tekst „http” w pliku */etc/services*
17. Sprawdź przy pomocy *ps* i *grep* (przetwarzanie potokowe) czy w uruchomionych procesach w systemie są procesy o nazwie *bash*.
18. Użyj polecenia *ps -ax* i postaraj się zakończyć proces o numerze 100.
19. Uruchom w tle program *top*. Zakończ go używając polecenia *kill*.

20. Uruchom w tle program *top*. Przenies go na pierwszy plan za pomoca *fg*. Wyjście z programu *top* po wcisnięciu klawisza *q*.
21. Wyświetl plik */etc/termcap* używając poleceń *cat*, *more*, *less*
22. Zapisz do pliku zawartość katalogu */usr/doc*. Do tego samego pliku dołącz zawartość katalogu */usr/local*. Zobacz plik z wynikami.
23. Sprawdź zawartość zmiennej *TERM* za pomocą *echo* oraz *set* i *grep*. Jakie są różnice w wynikach?
24. Wyświetl kalendarz na rok 2002 i pokaż wszystkie linie zawierające wyrazy *Su*, *December*, *January*.
25. Stwórz zmienną o dowolnej nazwie oraz wartości. Wyświetl jej wartość i wyprowadź ją do pliku.
26. Czy wśród zmiennych środowiskowych istnieją zmienne *USER*, *UID*, *DIR*, *HOSTNAME*, *PWD*. Jeśli tak wyprowadź ich wartość oraz nazwy do pliku (polecenia *set* i *grep*).
27. Wyświetl komunikat „*Test działania polecenia echo*”
28. Sprawdź działanie polecenia *top*. Jaki proces zużywa najwięcej czasu procesora?
29. Za pomocą *man* znajdź opis polecenia *find* i wyszukaj wszystkie pliki w katalogu domowym zawierające literę *s*. Wyniki przeslij do pliku o nazwie *lista*
30. Utwórz plik o dowolnej nazwie i nadaj mu uprawnienia do wykonywania. Uruchom go. Jaki efekt?
31. Usun wszystkie pliki z katalogu domowego (oprócz ukrytych). Czy udało się usunąć wszystkie pliki? Jeśli nie to zmień odpowiednie uprawnienia i usun.

#### Pytania:

1. Jaka komenda wyświetlić wszystkie procesy działające w systemie.
2. Jak zapisać błędy powstałe w wyniku działania polecenia?
3. Czy w linii komend może znajdować się tylko jedno polecenie?
4. Jaka komenda można sprawdzić który proces najbardziej obciąża procesor.
5. Czy polecenie *cat plik1 > plik2* da taki sam efekt jak *cp plik1 plik2*?
6. Czy polecenie *cat \$Z* wyświetli wartość zmiennej *Z*? Dlaczego?
7. Jak wyświetlić plik zawierający ilość tekstu nie mieszczącą się na jednym ekranie.

#### Do sprawozdania:

Opisać przebieg ćwiczeń wraz z odpowiedziami na pytania. Wypisać 3 wybrane polecenia i dla każdego opisać 3 parametry.