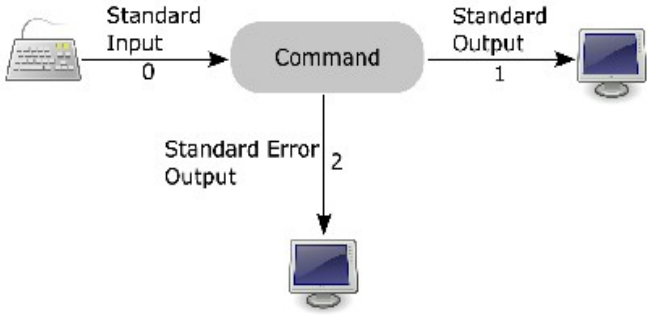
2.3 Zaawansowane wykonywanie poleceń

2.3.1 Stosowanie potokowania (*piping*) i przekierowań

Linux ma trzy standardowe kanały danych:



Opis standardowych kanałów:

Kanał	*Numer*	*Opis*
Standardowe wejście (stdin)	0	Uruchomiony program na bieżąco czyta dane z tego kanału (zwykle jest to klawiatura)
Standardowe wyjście (stdout)	1	Program wysyła dane do tego kanału (zwykle jest to monitor)
Standardowe wyjście błędów (stderr)	2	Błędy są przesyłane do tego kanału (zwykle na monitor)

Każdy kanał może być przez powłokę przekierowany. Na przykład, standardowe wejście może być przekierowane na plik, albo dane wyjściowe lub informacje o błędach mogą być kierowane do pliku.

Symbole przekierowania:

- < przekierowuje standardowe wejście,
- > przekierowuje standardowe wyjście (jest to skrót od „1>”),
- 2> przekierowuje standardowe wyjście dla błędów.

„>” nadpisze istniejący plik. Jeżeli dane mają być dopisane do zawartości pliku, należy użyć „>>” lub „2>>”.

Poniższe przykłady pokazują standardowe wejście, wyjście,wyjście błędów oraz

domyślne zachowanie powłoki (*shell*).

```
geeko@da51:~ > ls /opt /recipe
/bin/ls: /recipe: No such file or directory
/opt:
gnome kde3
```

Wyjaśnienie. Zawartość katalogu /opt wyświetlona jest w wierszu 3 i 4. Katalog /recipe nie istnieje. Komunikat błędu wyświetlony jest w wierszu 2.

Jeżeli standardowe wyjście błędów przekierujemy do **/dev/null**, tylko standardowe wyjście wyświetlone zostanie na ekranie:

```
geeko@da51:~ > ls /opt /recipe 2> /dev/null
/opt:
gnome kde3
```

By przekierować standardowe wyjście i standardowe wyjście błędów do pliku (takiego jak „output”) napiszemy:

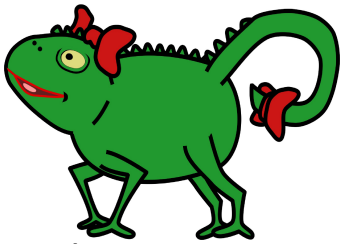
```
geeko@da51:~ > ls /opt /recipe > output 2>&1
geeko@da51:~ >
```

Wyjaśnienie. Standardowe wyjście jest przekierowane do pliku output, a następnie standardowe wyjście błędów jest przekierowane na standardowej wyjście (2>&1). Znak „&” wskazuje na poprzedzający opis pliku.

W innym przykładzie użyto pliku **list** zamiast **output**. By wyświetlić jego zawartość użyjemy polecenia **cat**:

```
geeko@da51:~> cat list
/bin/ls: /recipe: No such file or directory
/opt:
gnome
kde3
```

Te opcje komunikacji procesów są dostępne nie tylko w powłoce, ale mogą też być używane bezpośrednio w programach. Wszystkie pliki w systemie mogą być użyte jako wejście lub wyjście.



Przydatną czasem możliwością jest użycie pliku jako wejścia do programu oczekującego strumienia danych z klawiatury. Należy wtedy odpowiednio

przekierować standardowe wejście:

```
geeko@da51:~ # echo "Hello Tux,
>
> how are you?
> Is everything okay?" > greetings
geeko@da51:~ # mail tux < greetings
```

Wyjaśnienie. Po pierwsze, tekst jest przekierowywany do pliku greetings poleceniem > (wiersze 1-3). Program mail otrzymuje wejście z pliku greetings (nie z klawiatury), a następnie wysyła email do użytkownika tux (wiersz 4).

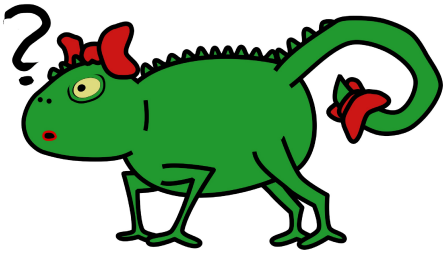
Ćwiczenie. Stosowanie potokowania i przekierowywania – część I

Utwórz plik o nazwie Redirection w swoim domowym katalogu, który zawiera:

... wyjście (wynik działania) polecenia ls /; uzupełniony następnie...

...zawartością zmiennej $PATH, oraz ...

...komunikatem o błędnym wykonaniu polecenia cd /root



Wyjście jednego polecenia może być użyte jako wejście do innego przez użycie potoku (*pipe*, symbol „|”).

składnia: ***command1 | command2***

W pojedynczym potoku (*pipe*) może istnieć maksymalnie 4 KB nieprzetworzonych danych. Jeżeli proces generujący wyjście próbuje zapisać dane do zapełnionego potoku, jest to zatrzymywane i wznawiane w chwili, gdy zapis jest już możliwy. Z drugiej strony – proces czytający jest zatrzymywany, gdy próbuje czytać z pustego potoku.

```
geeko@da51:~ > ls -l /etc | less
```

Czasami możemy chcieć wynik polecenia mieć zarówno wyświetlony na ekranie jak i zapisany do pliku. Użyjemy wtedy polecenia **tee**:

```
geeko@da51:~ > ls -l | tee output
```

W tym przypadku wyjście polecenia zostanie wyświetlone na ekranie i jednocześnie zapisane do pliku output.

Aby przekierować wyjście sekwencji kilku poleceń w wierszu poleceń, polecenia te muszą być oddzielone od siebie średnikami i cała sekwencja zamknięta nawiasami (command1; command2):

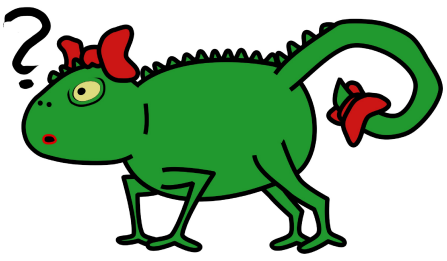
```
geeko@da51:~> (id ; ls ~) > output
geeko@da51:~> cat output
uid=1000(geeko) gid=100(users)
groups=14(uucp),16(dialout),33(video),100(users)
bin
Desktop
Documents
output
public_html
geeko@da51:~>
```

Ćwiczenie. Stosowanie potokowania i przekierowywania – część II

Polecenie **wc** (*word count*) zlicza słowa w pliku tekstowym. Polecenie **wc -l** zlicza wiersze w pliku.

Przekaż potokiem wyjście polecenia **ls -l** do polecenia **ws -l**. Co będzie rezultatem takiej konstrukcji?

_____



Powłoka uruchamia osobną podpowłokę (subshell) dla przetwarzania indywidualnych poleceń. By przekierować powiązane polecenia, powłoka musi być zmuszona do wykonywania łańcucha powiązanych poleceń w tej samej

podpowłoce. Jest to robione przez zamknięcie wyrażenia w nawiasach.

Po wykonaniu, każdy program zwraca wartość stanu wykonania (status). Jeżeli ta wartość jest równa zeru – program zakończył się sukcesem. W przypadku błędu – zwracana jest wartość różna od zera. Zależnie od programu – różne wartości wskazują na różne błędy (są kodami błędu).

Można użyć polecenia echo $? do wyświetlenia zwracanej wartości:

```
geeko@da51:~> ls
bin  Desktop  Documents  public_html
geeko@da51:~> echo $?
0
geeko@da51:~>
```

Zwracany kod może być użyty jako wyzwalacz (trigger) wykonania innego polecenia:

Wiązanie	***Wynik***
Polecenie1 && polecenie2	Polecenie 2 zostanie wykonane tylko wtedy, gdy polecenie1 zakończy się bezbłędnie.
Polecenie1 || polecenie2	Polecenie 2 zostanie wykonane tylko wtedy, gdy polecenie1 zakończy się z błędem.

Przykład:

```
1 geeko@da51:~> ls recipe || ls ~
2 /bin/ls: recipe: No such file or directory
3 bin Desktop Documents output public_html test
4 geeko@da51:~> ls recipe && ls ~
5 /bin/ls: recipe: No such file or directory
6 geeko@da51:~>
```

Wyjaśnienie. Plik recipe nie istnieje i polecenie ls recipe musi zakończyć się błędem (wiersze 1-2). Z tej przyczyny polecenie ls ~ z wiersza pierwszego jest wykonywane, a polecenie ls ~ w wierszu czwartym nie jest wykonywane.

Ćwiczenie. Stosowanie potokowania i przekierowywania – część III

Jaka wartość jest zwracana po wykonaniu poniższych poleceń?

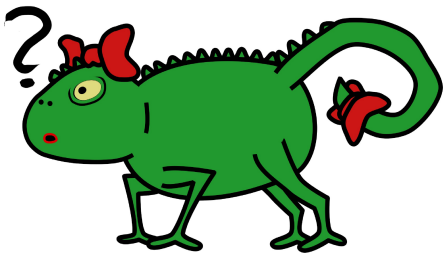
ls – l | wc – l _____

cd /root _____

less wall _____

echo $LOREM _____

echo $? _____



2.3.2 Wyrażenia regularne

Wyrażenia regularne to ciągi znaków i metaznaków, wzorce opisujące łańcuchy symboli. Metaznaki to takie znaki, które nie reprezentują siebie, ale mają specjalne znaczenie. Mogą być symbolami zastępczymi innych znaków lub wskazywać pozycję w łańcuchu.

Uwaga. Ważne jest, by pamiętać, że niektóre metaznaki stosowane przez powłokę mają różne zastosowania, różne też od tych przykładowych, omawianych w tym podręczniku.

Wiele poleceń opiera się na wyrażeniach regularnych budując wzorce do porównań.

Na przykład, polecenie **grep** i jego odmiana **egrep**.

Te polecenia są stosowane do przeszukiwania plików:

składnia: grep *szukany_wzorzec nazwa_pliku*

Polecenie to przeszukuje wskazany plik szukając tekstu pasującego do wzorca i wyświetla wiersze zawierające ten wzorzec.

Można podać więcej plików do przeszukania, wtedy będą wyświetlone zarówno pasujące wiersze tekstu jak i źródło (nazwa pliku), z którego pochodzą.

Dostępne jest kilka opcji, które pozwalają kontrolować wyjście i format wyświetlanej informacji (na przykład tylko numery pasujących wierszy, albo całe kontekstowe akapity).

Możesz podawać wzorce do wyszukania w formie wyrażeń regularnych, chociaż podstawowe polecenie grep jest dość ograniczone pod tym względem. By wyszukiwać w oparciu o bardziej złożone wzorce, należy użyć polecenia **egrep** (albo **grep -E**), które akceptuje rozszerzone wyrażenia regularne.

Zaleca się stosowanie polecenia egrep we wszystkich skryptach powłoki. Regularne wyrażenia stosowane w poleceniu egrep muszą być zgodne ze standardową składnią wyrażeń regularnych.

Więcej informacji o grep i wyrażeniach regularnych znajdziesz w dokumentacji, na stronie **man grep**.

By uniknąć niebezpieczeństwa interpretacji znaków specjalnych wyrażeń regularnych we wzorcach wyszukiwania przez powłokę (shell), zamknij wzorzec w nawiasach.

Przykład użycia poleceń **grep** oraz **egrep**:

```
geeko@da51:~> egrep (b|B)lurb file*
bash: syntax error near unexpected token `|'
geeko@da51:~> grep "(b|B)lurb" file*
geeko@da51:~> egrep "(b|B)lurb" file*
file1:blurb
filei2:Blurb
geeko@da51:~>
```

Teraz omówimy opcje, które można używać z poleceniem grep.

- **-i** polecenie nie będzie rozróżniać dużych i małych liter.

```
1  geeko@da51:~> grep "duis" *.txt
2  loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
3  loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
4  geeko@da51:~> grep -i "duis" *.txt
5  loremipsum.txt:Duis te feugifacilisi.
6  loremipsum.txt:Duis autem dolor in hendrerit
7  loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
8  loremiloremipsum.txt:Duis te feugifacilisi per suscipit
9  loremipsum.txt:Duis te feugifacilisi.
10 loremipsum.txt:Duis autem dolor in hendrerit in
11 loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
12 geeko@da51:~>
```

Wyjaśnienie. Pierwsze polecenie grep (wiersz 1) odnajduje dwa wiersze w pliku tekstowym (wiersze 2-3). Polecenie grep -i (wiersz 4) ignoruje rozróżnienie między dużymi i małymi literami i wyświetla dodatkowo wiersze z dużą literą „D” w słowie „Duis” (wiersze 5-11).

- **-l** pokazuje tylko nazwy plików zawierających poszukiwany ciąg znaków:

```
1 geeko@da51:~> grep "duis" *.txt
2 loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
3 loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
4 geeko@da51:~> grep -l "duis" *.txt
5 loremipsum.txt
6 geeko@da51:~>
```

Wyjaśnienie. Jeżeli polecenie grep znajdzie więcej niż jeden plik, wypisuje jego nazwę na początku każdego wiersza (wiersze 2-3). Grep z opcją – l (wiersz 4) wyświetla tylko nazwę pliku tekstowego(wiersz 5).

- -r przeszukuje całe drzewa katalogów. Opcja ta pracuje wyłącznie na katalogach (w poleceniu grep trzeba podać katalog a nie nazwy plików).

```
1 geeko@da51:~> grep "duis" *
2 loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
3 loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
4 geeko@da51:~> grep -r "duis" *
5 dirgrep/loremipsum2.txt:delenit au gue duis dolore te feugat nulla facilisi.
6 dirgrep/loremipsum2.txt:au gue duis dolore te feugat nulla facilisi.
7 loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
8 loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
9 geeko@da51:~>
```

Wyjaśnienie. Bieżący katalog jest przeszukiwany w wierszu 1. Wiersze 2-3 pokazują wynik teko przeszukania – plik loremipsum.txt. Grep -r (wiersz 4) przeszukiwane są też podkatalogi (podkatalog dirgrep) i wynik pokazują wiersze 5-8.

- **-v** pokazuje wszystkie wiersze pliku, które nie zawierają poszukiwanego ciągu znaków:

```
geeko@da51:~> grep "duis" *.txt
loremipsum.txt:delenit au gue duis dolore te feugat nulla facilisi.
loremipsum.txt:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~> grep -v "duis" *.txt
loremipsum.txt:LOREM IPSUM DOLOR SIT AMET
loremipsum.txt:
loremipsum.txt:Lorem ipsum dolor sit amet,
loremipsum.txt:consectetuer adipiscing elit,
...
```

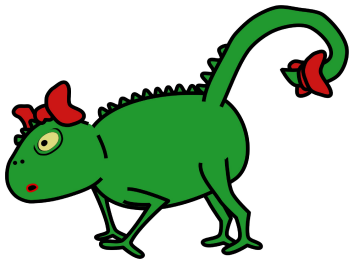
```
numbers.txt:0
numbers.txt:98798
numbers.txt:765765
geeko@da51:~>
```

- **-n** pokazuje numery wierszy w tekście:

```
geeko@da51:~> grep -n "duis" *.txt
loremipsum.txt:18:delenit au gue duis dolore te feugat nulla facilisi.
loremipsum.txt:39:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```

- **-h** nie pokazuje nazw plików:

```
geeko@da51:~> grep -h "duis" *.txt
delenit au gue duis dolore te feugat nulla facilisi.
au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```



Zostaną teraz pokazane i krótko omówione niektóre najważniejsze metaznaki.

- ^ oznacza początek wiersza

```
geeko@da51:~> grep -n "^et" *.txt
loremipsum.txt:16:et accumsan et iusto odio dignissim
loremipsum.txt:37:et iusto odio dignissim qui blandit
geeko@da51:~>
```

Wyjaśnienie. Wyrażenie „^et” mówi o tym, że chodzi nam o znaki „et” występujące wyłącznie na początku wiersza.

- **$** oznacza koniec wiersza

```
geeko@da51:~> grep -n "at$" *.txt
loremipsum.txt:14:vel illum dolore eu feugiat
geeko@da51:~>
```

Wyjaśnienie. Wyrażenie „at$” mówi o tym, że chodzi nam o znaki „at” występujące wyłącznie na końcu wiersza.

- **\<** oznacza początek słowa

```
geeko@da51:~> grep -n "\<se" *.txt
loremipsum.txt:5:sed diem nonummy nibh euismodtincidunt
loremipsum.txt:27:sed diem nonummy nibh euismodtincidunt
geeko@da51:~>
```

Wyjaśnienie. Wyrażenie „\<se” mówi o tym, że chodzi nam o znaki „se” występujące wyłącznie na początku słowa.

- **\>** oznacza koniec słowa

```
geeko@da51:~> grep -n "se\>" *.txt
loremipsum.txt:13:in vulputate velit esse molestie consequat,
loremipsum.txt:34:vulputate velit esse molestie consequat,
geeko@da51:~>
```

Wyjaśnienie. Wyrażenie „se\>” mówi o tym, że chodzi nam o znaki „se” występujące wyłącznie na końcu słowa.

- [znaki] występuje jeden znak z zestawu:

```
geeko@da51:~> grep -n "[ex]e" *.txt
loremipsum.txt:6:ut lacreet dolore magna aliguam erat volutpat.
loremipsum.txt:8:quis nostrud exerci tution ullamcorper
loremipsum.txt:20:quis nostrud exerci taion
loremipsum.txt:28:ut lacreet dolore magna aliguam erat volutpat.
loremipsum.txt:30:quis nostrud exerci tution ullamcorper
geeko@da51:~>
```

Wyjaśnienie. Kryterium opisane wyrażeniem „[ex]e” spełniają wszystkie wiersze zawierające „ee” lub „xe”.

Użycie myślnika pozwala na określenie zakresu znaków:

```
geeko@da51:~> grep -n "^[a-c]" *.txt
loremipsum.txt:4:consectetuer adipiscing elit,
loremipsum.txt:26:adipiscing elit,
loremipsum.txt:39:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```

Wyjaśnienie. Kryterium określone wyrażeniem „^[a-c]” spełniają wszystkie wiersze, które zaczynają się od litery a,b lub c.

[^znaki] żaden z wymienionych znaków

```
geeko@da51:~> grep -n "^[^a-c]" *.txt
loremipsum.txt:1:LOREM IPSUM DOLOR SIT AMET
loremipsum.txt:3:Lorem ipsum dolor sit amet,
loremipsum.txt:5:sed diem nonummy nibh euismodtincidunt
...
numbers.txt:11:0
numbers.txt:98798
numbers.txt:765765
geeko@da51:~>
```

Wyjaśnienie. Polecenie grep wyszuka wszystkie wiersze, które nie zaczynają się od liter a, b lub c.

. dowolny pojedynczy znak

```
geeko@da51:~> grep -n "^.$" *.txt
numbers.txt:11:0
geeko@da51:~>
```

Wyjaśnienie. Polecenie grep wyszuka wszystkie wiersze, które zawierają tylko jeden znak (między początkiem a końcem wiersza).

+ tylko z egrep! Jedno lub więcej z podanych wyrażeń.

```
geeko@da51:~> egrep -n "^0+$" *.txt
numbers.txt:1:0000
numbers.txt:11:0
geeko@da51:~>
```

Wyjaśnienie. Kryterium „^0+$” spełnią wszystkie wiersze z jednym lub więcej znakiem „0” (między początkiem a końcem wiersza).

* dowolna ilość (też nic) wymienionych znaków.

```
geeko@da51:~> grep -n "^0*$" *.txt
loremipsum.txt:2:
numbers.txt:1:0000
numbers.txt:11:0
geeko@da51:~>
```

Wyjaśnienie. Kryterium „^0*$” spełnią wszystkie wiersze z żadnym, jednym lub więcej znakiem „0” (między początkiem a końcem wiersza).

{min,max} Tylko z egrep!

Podane wyrażenie występuje minimum *min* razy, a maksymalnie *max* razy.

```
geeko@da51:~> egrep -n "[0-9]{4,5}" *.txt
numbers.txt:1:0000
numbers.txt:12:98798
numbers.txt:13:765765
geeko@da51:~>
```

Wyjaśnienie. Kryterium „[0-9]{4,5}” spełnią wszystkie wiersze z czterocyfrowymi (jako minimum) i pięciocyfrowymi (jako maksimum) liczbami.

Można przydzielać trzy rodzaje uprawnień do pliku lub katalogu:

Dlaczego w wierszu 4 przykładu jest liczba sześciocyfrowa???

Kryterium jest nieprecyzyjne, bo nie podaje co jest przed i co jest po liczbie cztero- i pięciocyfrowej.

Przeanalizuj przykład następny. Dodanie ^ z przodu oraz $ z tyłu wyrażenia pozwoliło na uzyskanie zamierzonego efektu.

```
geeko@da51:~> egrep -n "^[0-9]{4,5}$" *.txt
numbers.txt:1:0000
numbers.txt:12:98798
geeko@da51:~>
```

| Tylko z egrep! Wyrażenie przed lub po znaku potoku (*pipe*).

```
geeko@da51:~> egrep -n "ea|gue" *.txt
loremipsum.txt:10:ut aliquip ex ea commodo consequat.
loremipsum.txt:18:delenit au gue duis dolore te feugat nulla facilisi.
loremipsum.txt:31:suscipit lobortis nisl ut aliquip ex ea commodo consequat.
```

```
loremipsum.txt:39:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```

Wyjaśnienie. Kryterium „ea|gue” spełnią wszystkie wiersze z wyrażeniami „ea” lub „gue”.

(...) Tylko z egrep! Włącza alternatywy do grupowania z innymi.

```
geeko@da51:~> egrep -n "(d|D)uis" *.txt
loremipsum.txt:11:Duis te feugifacilisi.
loremipsum.txt:12:Duis autem dolor in hendrerit
loremipsum.txt:18:delenit au gue duis dolore te feugat nulla facilisi.
loremipsum.txt:23:Duis te feugifacilisi per suscipit
loremipsum.txt:32:Duis te feugifacilisi.
loremipsum.txt:33:Duis autem dolor in hendrerit in
loremipsum.txt:39:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```

Wyjaśnienie. Kryterium „(d|D)uis” spełnią wszystkie wiersze z wyra żenieniami „duis” lub „Duis”.

? Tylko z egrep! Zero lub jeden znak poprzedzający.

```
geeko@da51:~> egrep -n "it,$" *.txt
loremipsum.txt:4:consectetuer adipiscing elit,
loremipsum.txt:26:adipiscing elit,
geeko@da51:~> egrep -n "it,?$" *.txt
loremipsum.txt:4:consectetuer adipiscing elit,
loremipsum.txt:12:Duis autem dolor in hendrerit
loremipsum.txt:23:Duis te feugifacilisi per suscipit
loremipsum.txt:26:adipiscing elit,
loremipsum.txt:37:et iusto odio dignissim qui blandit
loremipsum.txt:38:praesent luptatum zzril delenit
```

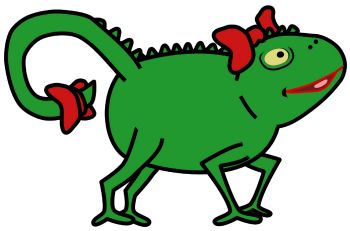
Wyjaśnienie. Kryterium „it,$” spełnią wszystkie wiersze kończące się ciągiem „it,”. Kryterium „it,?” spełnią wiersze kończące się na „it,” lub samo”it”.

**** Maskuje następujący po nim znak, by zneutralizować jego specjalne znaczenie (dla powłoki).

```
geeko@da51:~> grep -n "\.$" *.txt
loremipsum.txt:6:ut lacreet dolore magna aliguam erat volutpat.
loremipsum.txt:10:ut aliquip ex ea commodo consequat.
loremipsum.txt:11:Duis te feugifacilisi.
loremipsum.txt:18:delenit au gue duis dolore te feugat nulla facilisi.
loremipsum.txt:22:nisl ut aliquip ex en commodo consequat.
loremipsum.txt:24:lobortis nisl ut aliquip ex en commodo consequat.
loremipsum.txt:28:ut lacreet dolore magna aliguam erat volutpat.
loremipsum.txt:31:suscipit lobortis nisl ut aliquip ex ea commodo consequat.
loremipsum.txt:32:Duis te feugifacilisi.
loremipsum.txt:39:au gue duis dolore te feugat nulla facilisi.
geeko@da51:~>
```

Wyjaśnienie. Kryterium „\.$” spełnią wszystkie wiersze kończące się „.” Bez maskowania ukośnikiem kropka oznacza dowolny znak (→ zobacz kilka stron wcześniej).

By nauczyć się więcej o strukturach regularnych wyrażeń, przeczytaj podręcznik → **man 7 regex**.



Ćwiczenie. Rozumienie wyrażeń regularnych – część I

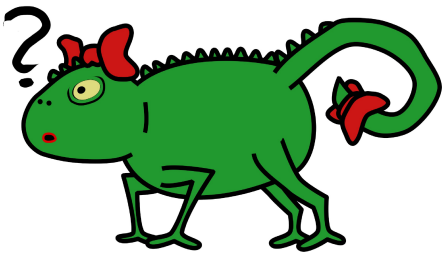
Co znaczą podane niżej wyrażenia regularne? Jak dużo wierszy je spełnia z pliku loremipsum.txt?

^.t _____

it\> _____

m{2} _____

[^.,]$ _____



Ćwiczenie. Rozumienie wyrażeń regularnych – część II

Utwórz wyrażenia regularne opisujące podane kryteria. Jak dużo wierszy je spełnia z przykładowego pliku loremipsum.txt?

Wszystkie wiersze z przynajmniej jedną dużą literą.

_____

Wszystkie wiersze zawierające słowo „in” lub „ad”.

_____

Wszystkie wiersze, w których występuje przynajmniej jedno słowo trzyliterowe.

_____

2.3.3 Zarządzanie procesami bash

Środowisko powłoki Linuksa pozwala na uruchamianie procesów zarówno na pierwszym planie (*foreground*) jak i w tle (*background*).

Procesy wykonywane na pierwszym planie uruchamiane są w oknie terminala i wykonywane są do końca; okno terminala nie powróci do linii poleceń (ze znakiem zachęty) póki proces się nie zakończy.

Procesy wykonywane w tle pozwalają na powrót do linii poleceń zanim się zakończą (można uruchamiać kolejne procesy).

Można wykonywane procesy przełączać między pierwszym planem a tłem, ale pod pewnymi warunkami:

- proces musi byś uruchomiony w oknie terminala lub konsoli powłoki,
- proces nie wymaga wprowadzania danych z okna terminala.

Jeżeli proces spełnia powyższe warunki, może być przesunięty w tło.

Uwaga. Procesy, które wymagają podawania danych (na wejściu) z terminala mogą być również przenoszone w tło, ale w momencie, w którym należy wprowadzić dane, proces się zawiesi i będzie czekał na przywrócenie do pierwszego planu i wprowadzenie wymaganych danych.

Polecenia powłoki mogą być uruchamiane zarówno na pierwszym planie, jak i w tle. Procesy na pierwszym planie mogą bezpośrednio otrzymywać przenoszone sygnały.

Na przykład, jeżeli wprowadzisz xeyes by uruchomić program, pracuje on na pierwszym planie. Naciśnięcie kombinacji klawiszy Ctrl oraz Z zatrzyma proces:

```
[1]+ Stopped xeyes
geeko@da51:~>
```

Możesz uruchomić zatrzymany proces w tle poleceniem **bg**:

```
geeko@da51:~> bg
[1]+ xeyes &
geeko@da51:~>
```

Znak & (ampersand) (wiersz 2) oznacza, że proces aktualnie jest uruchomiony w tle. Dołączenie znaku & do polecenia startującego dany program od razu uruchamia go w tle:

```
geeko@da51:~> xeyes &
[2] 4351
geeko@da51:~>
```

W ten sposób, powłoka, z której wystartowałeś program jest natychmiast dostępna dla użytkownika.

W przykładzie została wyświetlana zarówno informacja o identyfikatorze zadania (job ID) [2] jak i identyfikatorze procesu [4351].

Każdemu procesowi uruchamianemu z powłoki jest przydzielany identyfikator zadania przez kontrolę zadań powłoki. Polecenie **jobs** wyświetla wszystkie wykonywane aktualnie zadania:

```
geeko@da51:~> jobs
[1]+ Stopped xeyes
```

```
[2] Running xeyes &
[4]- Running sleep 99 &
geeko@da51:~>
```

W tym przykładzie, proces zadania ID 3 już się zakończył. Procesy 2 i 4 są uruchomione w tle (znak &), a proces 1 jest wstrzymany.

Znak „+” wskazuje proces, który zareaguje na polecenie **fg** bez opcji, a znak „-” wskazuje proces, który odziedziczy znak „+” , gdy aktualnie nim oznaczony proces się zakończy.

Następnemu procesowi uruchomionemu w tle zostanie przydzielony ID 5 (najwyższy aktualnie przydzielony numer + 1).

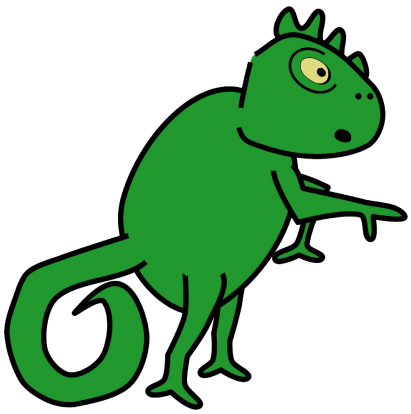
Można wznowić pracę wstrzymanego procesu w tle poleceniem **bg**, albo przełączyć go na pierwszy plan poleceniem **fg job_ID**:

```
geeko@da51:~> fg 1
xeyes
```

Powłoka informuje o zakończeniu procesu uruchomionego w tle:

```
[4]- Done sleep 99
```

Identyfikator zadania wyświetlony jest w nawiasach kwadratowych. *Done* znaczy, że proces zakończył się prawidłowo. *Terminated* oznacza, że zażądano zakończenia procesu. **Killed** wskazuje na wymuszone zakończenie procesu.



Ćwiczenie. Zarządzanie procesami powłoki

1.Uruchom program Acrobat Reader z wiersza poleceń:

```
acroread /usr/share/doc/manual/sled-gnome-user\_en/ sled-gnome-user\_en.pdf
```

2.Zatrzymaj proces. Co się dzieje, kiedy próbujesz używać programu Acrobat Reader?

3.Przenieś proces acroread w tło. Co się dzieje, kiedy próbujesz używać programu Acrobat Reader?