Podstawy Linux - Wildcards, Vi, Wstęp do skryptów Bash

2 grudnia 2018

1 Wildcards

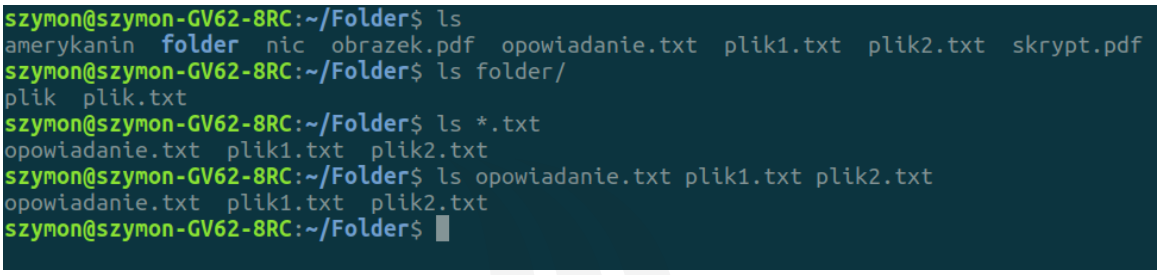
Narzędzia przydatne podczas przekazywania plików do komend operujących na plikach. Zamiast podawać jako argument wszystkie interesujące nas pliki możemy wykorzystać zbiór trzech narzedzi, które pozwolą na wyciągnięcie grupy dowolnej ilości (np. 1000) plików pasujących do zadanego schematu za pomocą jednej krótkiej komendy.

Dotychczas używaliśmy komendy **rm** podając nazwę pliku lub folderu który chcemy usunąć. W przypadku gdyby było to wiele plików o konkretnym schemacie (np. pliki z rozszerzeniem .txt, zawierające cyfrę itd.) takie podejście przestaje mieć sens. W tym celu powstały narzędzia Wildcards. Pozwalają na budowę wyrażeń - szablonów, które prównywane są ze wszystkimi (poza ukrytymi) plikami oraz folderami z poziomu aktualnego folderu.

1.1 Operator *

Pierwszy z operatorów oznacza dowolny ciąg znaków o dowolnej długości. Przykłady:

- **ls a*** - wypisanie wszystkich plików i zawartości folderów zaczynających się literą a (ls zostanie wywołane z argumentami np. ls a.txt asa.dat ala)
- **ls a*b** - wypisanie wszystkich plików i zawartości folderów zaczynających się na literę a oraz kończących na literę b
- **rm -r *** - usunięcie wszystkich plików i folderów z aktualnego folderu
- ***.*** - wszystkie pliki posiadające rozszerzenie
- ***.pdf** - wszystkie pliki o rozszerzeniu pdf
- **ls ∼/*/plik.txt** - przeszukanie podfolderów katalogu domowego w celu znalezienia pliku **plik.txt**



Rysunek 1: Komendy **ls *.txt** oraz **ls opowiadanie.txt plik1.txt plik2.txt** są identyczne - stosowanie wildcards wykonuje za nas pracę.

1.2 Operator ?

Oznacza jeden dowolny znak. Przykłady:

- ***.???** - wszystkie pliki z 3-literowym rozszerzeniem
- **?a*** - drugi znak jest równy a

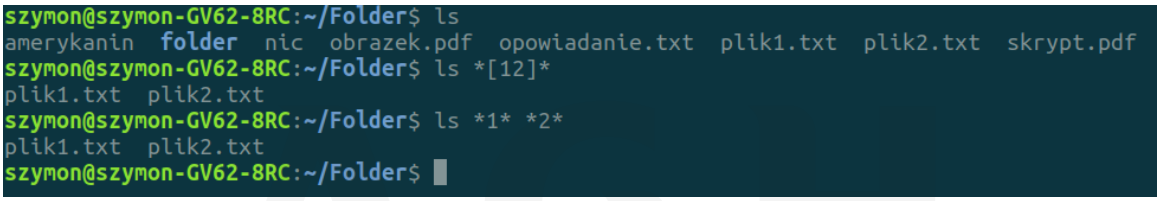
1.3 Operator [ ]

Operator [ ] służy do grupowania symboli - do szablonu pasują pliki które posiadają w miejscu [ ] którykowliek z podanych wewnątrz symboli. Sposoby użycia:

- symbole wewnątrz [ ], np. [abc123]
- zakres symboli, np. [a-c] lub [0-7]
- negacja ^, wszystkie symbole różne (nie należące) od grupy np. [^1-3], [^abc]

Przykłady:

- **[0-9]*** - wszystkie pliki zaczynające się od cyfry
- **?[abc]?** - wszystkie 3 znakowe pliki których 2 litera to a lub b lub c



Rysunek 2: Komendy **ls *[12]*** i **ls *1* *2*** są identyczne

1.4 Generator {}

Służy do generowania różnych wersji szablonów w jednym. Wstawiamy kilka alternatywnych wyrażeń. Przykład:

- *.{pdf,txt} - zostanie rozwinięte w *.pdf *.txt np. ls *.{pdf,txt} jest równoważne ls *.pdf *.txt

```
szymon@szymon-GV62-8RC:~/Folder$ ls
amerykanin  folder  nic  obrazek.pdf  opowiadanie.txt  plik1.txt  plik2.txt  skrypt.pdf
szymon@szymon-GV62-8RC:~/Folder$ ls *.{txt,pdf}
obrazek.pdf  opowiadanie.txt  plik1.txt  plik2.txt  skrypt.pdf
szymon@szymon-GV62-8RC:~/Folder$ ls *.txt *.pdf
obrazek.pdf  opowiadanie.txt  plik1.txt  plik2.txt  skrypt.pdf
szymon@szymon-GV62-8RC:~/Folder$
```

Rysunek 3: Komendy **ls *.{txt,pdf}** i **ls *.txt *.pdf** są identyczne

1.5 Zadania

Zadanie 1 Z poziomu katalogu domowego wypisać wszystkie pliki (za pomocą ls -d - jak działa ta opcja i dlaczego?) a) posiadające trzyliterowe rozszerzenie oraz b) wszystkie pliki posiadające rozszerzenie znajdujące się w katalogu /etc (dotyczy wszystkich następnych zadań z tego tematu).

Zadanie 2 Wypisać wszystkie a) nie ukryte pliki i foldery b) ukryte pliki i foldery c) wszystkie pliki i foldery z katalogu etc (wskazówka: wykorzystać {})

Zadanie 3 Wypisac wszystkie pliki które posiadają jedncześnie litery a oraz z, w dodowlnej kolejności (wskazówka, potrzebne 2 szablony)

Zadanie 4 Wypisac wszystkie obiekty, które znajdują się w podfolderach kończących się cyfrą. Nazwa obiektu musi zaczynać się od litery.

1.6 Rozwiązania

Zadanie 1 a) **ls -d /etc/*.???** b) **ls -d /etc/*.***

Zadanie 2 a) **ls -d /etc/*** b) **ls -d /etc/.*** c) **ls -d /etc/{.*,*}**

Zadanie 3 **ls -d /etc/*{a*z,z*a}***

Zadanie 4 **ls -d /etc/*[0-9]/[a-Z]***

2 Edytor Vi/Vim

Podstawowy edytor tekstowy systemu Linux. Mimo swojej przestarzałej formy, w dalszym ciągu jest on w użytku i mimo iż nie sprawia takiego wrażenia, jest bardzo rozbudowanym edytorem. Czasami, np. podczas pracy zdalnej jak na zajęciach, może być przydatna podstawowa znajomość tego edytora (jest w pełni kompatybilny z powłoką, więc nie wymaga GUI). Edytor uruchamiamy wpisując vi (lepiej vim), opcjonalnie podajac nazwę pliku który chcemu edytować/utworzyć.

2.1 Tryby Vi

Edytor posiada dwa tryby: **edycji (edit mode) - e** oraz **wpisywania (insert mode) - i**. Początkowo vi uruchamiany jest w trybie **e**. Aby przejść do trybu **i** należy wcisnąć klawisz 'i' - zaczyna dokładnie w miejscu kursora lub 'a' - zaczyna jeden znak za miejscem kursora. W trybie **i** po prostu wpisujemy linia po linii z klawiatury. Aby wyjść z powrotem do trybu edycji wciskamy klawisz 'esc'.

2.2 Tryb edycji

Będąc w tryie edycji możemy wykonać szereg funkcjonalności. Komendy zaczynające się od : wymagają potwierdzenia enterem, pozostałe wyknują się automatycznie po wciśnięciu ostatiego klawisza. Zostaną tu pokazane tylko te podstawowe.

1. Wyjście z edytora:
   (a) **:q** - wyjście z edytora, możliwe tylko gdy plik jest zapisany
   (b) **:q!** - wyjście z odrzuceniem zmian od odstatniego zapisu
   (c) **:w** - zapisanie pliku
   (d) **:wq** lub **ZZ** - zapisanie pliku oraz wyjście
2. Poruszanie się po pliku w trybie edycji:
   (a) strzałki lub 'j', 'k', 'h', 'l' - poruszanie się po pliku
   (b) ^ - przejście na koniec linii - przydatne przy długich liniach
   (c) $ - przejście na początek linii
   (d) ***n*G** - przejście do ***n***-tej linii
   (e) **{** oraz **}** - przechodzenie do poprzedniego oraz nastepnego paragrafu
   (f) **:set nu** - komenda dadająca numery linii w edytorze, ułatwia poruszanie się
   (g) **:set tabstop=*n*** - zmiana szerokości tab
3. Kopiowanie, usuwanie, cofanie zmian

(a) **u** - cofa zmiany o jedną akcje (sprawdzić co się stanie, gdy poprzednią akcją było wpisywanie tekstu w trybie **i**

(b) **x** - usunięcie znaku

(c) ***n*x** - usunięcie ***n*** znaków

(d) **dd** - usunięcie linii

(e) **d*c*** - usunięcie do miejsca do którego normalnie prowadzi komenda c, np d^ usunie wszystko od kursora do początku linii

(f) **v lub V** - wejście w tryb zaznaczenia tekstu, v - zaznaczenie z dokładnością do znaku, V - zaznaczanie linii

(g) **d** - podczas zaznaczania - "wytnij"

(h) **y** - podczas zaznaczania - "kopiuj"

(i) **P** oraz **p** - wklej przed (**P**) lub za **p** kursorem

(j) **yy** lub **y*n*y** - szybkie kopiowanie całej linii lub ***n*** linii

(k) **dd** lub **d*n*d** - szybkie wycięcie całej linii lub ***n*** linii

(l) **ctrl+shift+V** - wklejanie ze schowka systemowego

3 Skrypty bash

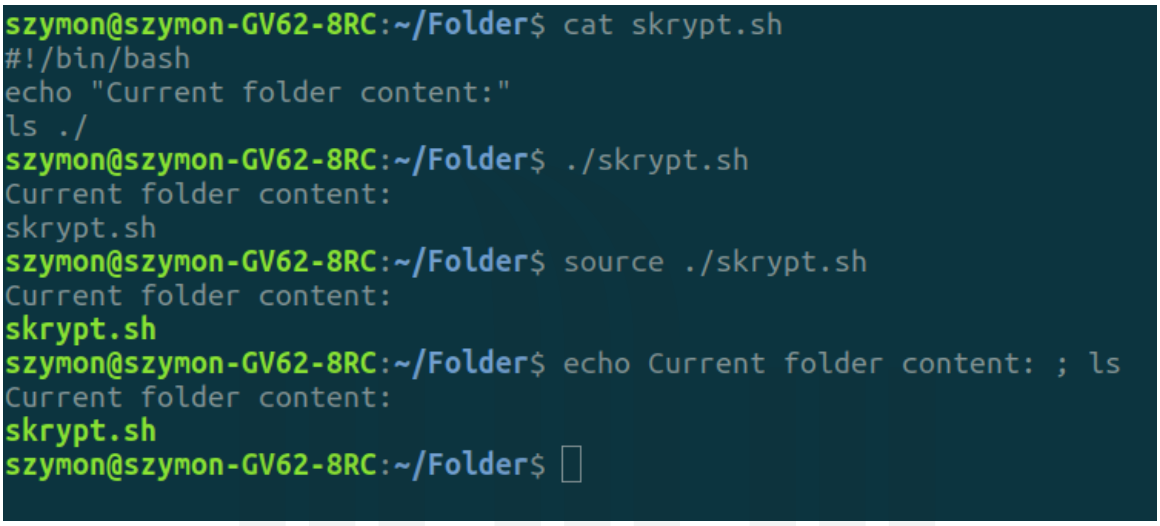
Skrypty służą do automatyzacji pracy, organizacji oraz zapisania bardziej skomplikowanych funkcjonalności w jednym pliku, który nastepnie może być traktowany tak samo jak komendy powłoki. Wszystko to, czego można użyć wewnątrz powłoki (dotychczasowe zajęcia) można uzywać wewnątrz skryptu. Mozna więc stwierdzić, że skrypty bash są po prostu uporządkowanymi zbiorami komend.

3.1 Budowa oraz wywołanie skryptu

Każdy skrypt musi zaczynać się znakami ***#!/bin/bash***, gdzie /bin/bash jest ścieżką do interpretera skryptu. Skryty posiadają rozszerzenie .sh. Muszą też mieć ustawioną opcję wykonywania (chmod u+x skrypt.sh). Dwa sposoby wywołania skryptu:

- **./skrypt.sh** - wywołanie skryptu w odrębnej powłoce
- **source skrypt.sh lub . skrypt.sh** - wywołanie skryptu wewnątrz aktualnej powłoki

Dodatkowo skrypty mogą pobierać argumenty. Przekazujemy je dokładnie tak samo jak w przypadku komend (np. ./skrypt.sh arg1 arg2)



Rysunek 4: **cat skrypt.sh** - wypisanie zawartości pliku, następnie różnica między wywołaniami **./skrypt.sh** i **source skrypt.sh** - w drugim przypadku komenda ls wywołana w aktualnej powłoce wykorzystuje formatowanie, w pierwszym zostaje tylko przechwycone wyjście. Wywołanie komend „ręcznie" daje ten sam efekt. ; oddziela komendy tak aby wykonały się po kolei, zapisane w jednej linii

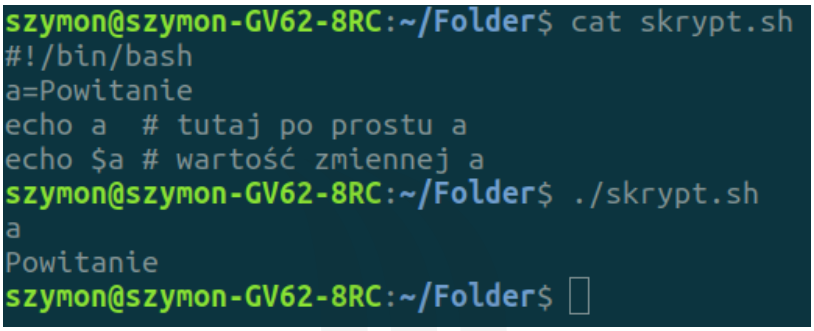
3.2 Zmienne

Zmienne w bash przechowują wartości. Niestety w bashu zmienne nie posiadają jawnych typów - wszystkie są zbiorami znaków, nie ma rozróżnienia między znakami a liczbami. Niektóre instrukcje pozwalają działać na zmiennych jak na liczbach, jednak to użytkownik musi dbać o poprawność wartości zmiennej. Aby utworzyć zmienną wystarczy użyć zapisu: *zmienna=wartość*. Należy pamiętać o braku spacji, inaczej zmienna zostanie potraktowana jako wywołanie komendy (która prawdopodonie nie istnieje) z argumentami = i *wartość*. Aby zapisać zmienną ze spacjami należy umieścić ją wewnątrz ' ' lub " ". Różnica między tymi dwoma zapisami polega na tym, że wewnątrz ' ' nie możemy odwoływac się do zmiennych. Przykład zmiennej: *a='Taka zmienna'*. Aby uzyskać wartość zmiennej wywołujemy **$nazwazmiennej**. Np. echo $a.

3.2.1 Specjalne zmienne

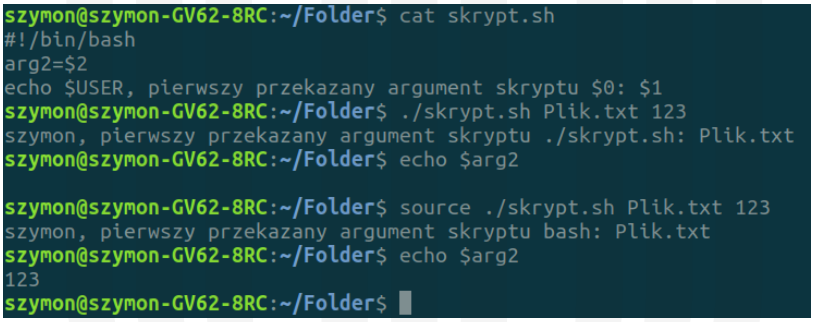
Istnieje wiele specjalnych zmiennych, do którym możemy odwołać sie wewnątrz skryptu. Najważniejsze z nich:

- $0 - nazwa skryptu
- $1 - $9 - pierwsze dziewięć argumentów przekazanych do skryptu
- $# - ilość argumentów przekazanych do skryptu



Rysunek 5: Przypisanie zmiennej oraz dwie próby wypisania jej wartości

- $@ - wszystkie argumenty przekazane do skryptu
- $USER - nazwa użytkownika wywołującego skrypt
- $SECONDS - liczba sekund odkąd skrypt został wywołany
- $RANDOM - zwraca liczbę losową



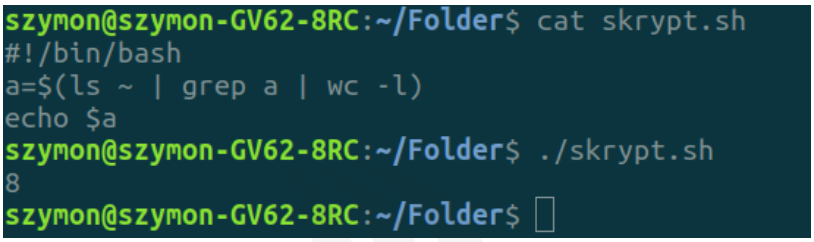
Rysunek 6: Specjalne zmienne oraz kolejna różnica między wywołaniami **./skrypt.sh** i **source ./skrypt.sh** - w drugim przypadku tymczasowe zmiany nie znikają po zakończeniu skryptu, mamy dostep do zmiennej *arg2*

3.2.2 Zmienne przechowujące wynik wywołania komendy

Do zmiennej możemy przypisać wynik wywołania komendy, należy umieścić komendę wewnątrz $( ), np. wstawić listę plików katalogu domowego do zmiennej: **pliki=$(ls ~)**.

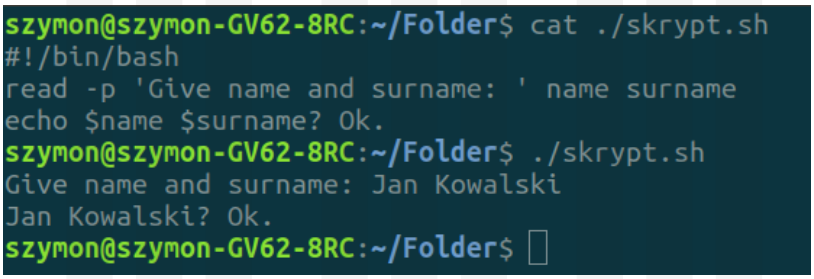
3.3 Wczytywanie z klawiatury

Aby poprosić użytkownika o wpisanie wartości, która następnie zostanie przypisana do zmiennej używamy komendy **read**. np. *read a*. Przydatny argument -p pozwala na dodanie



Rysunek 7: Przypisanie wyniku wywołania skomplikowanej komendy - najpierw **ls** $\sim$ - wypisanie zawartości katalogu domowego i przekazanie do komendy **grep a**, która znajduje linie w których występuje litera a, następnie komenda **wc -l** liczy ilość linii znalezionych przez **grep**

prompta: *read -p 'Podaj imie:' name*. Można wczytywać wiele zmiennych jednoczesnie, np. *read -p 'Podaj dwie liczby' a b*, następnie przy podawaniu argumentów oddzielamy je spacją.



Rysunek 8: Przykład wczytania dwóch zmiennych

3.4 Arytmetyka

Narzędzia podstawowe do wykonywania prostych działań aytmetycznych na liczach i zmiennych.

3.4.1 Polecenie let

Polecenie let przyjmuje wyrażenie, w którym po lewej stronie stoi zmienna, do której przypisany zostanie wynik. Dostępne operatory: +, -, /, * (gdy brak " " musimy użyć *, aby usunąć specjalne znaczenie symbolu * - patrz przykład z 2*2), %, val++, val--. Działa tylko na liczbach całkowitych Przykłady

- **let "a = 1 + 2 + 3"**
- **b=4; let "a = 2*b"** lub **let a=2*2**
- **let "a = $1 + 1"** - dodanie jedynki do pierwszego argumentu skryptu

- **let a++** - zwiększenie wartości *a* o jeden. Tzw. operator post-inkrementacji - do wyrażenia w którym występuje brana jest wartość *a* przed zwiększeniem. Analogicznie istnieje operator post-dekrementacji **a--**

```
szymon@szymon-GV62-8RC:~$ a=5
szymon@szymon-GV62-8RC:~$ let b=a+2 ; echo $b
7
szymon@szymon-GV62-8RC:~$ let b=$a+2 ; echo $b
7
szymon@szymon-GV62-8RC:~$ let a++ ; echo $a
6
```

Rysunek 9: Znak ; oddziela od siebie instrukcje w jednej linii, będą one wykonane po kolei

```
szymon@szymon-GV62-8RC:~$ a=1; b=1;
szymon@szymon-GV62-8RC:~$ let "c = a++ + b++"
szymon@szymon-GV62-8RC:~$ echo a=$a b=$b c=$c
a=2 b=2 c=2
```

Rysunek 10: Przykład działania post-inkrementacji

3.4.2 Wersja skrócona

Do prostych operacji arytmetycznych możemy wykorzystać prostszą składnię **((expr))**, np. **a=$((1 + 2))**. Znak $ używamy aby dostać wartość wyrażenia.

UWAGA: Zarówno w przypadku polecenia **let** jak i **$((expr))** możemy przekazać do wyrażenia wartość zmiennej ($a) jak i samą zmienną (a). W przypadku operatorów jednoargumentowych (jak np. var++) musimy przekazać zmienną, np. **let a++** lub ((a++)).

```
szymon@szymon-GV62-8RC:~$ a=1
szymon@szymon-GV62-8RC:~$ ((a++)) ; echo $a
2
szymon@szymon-GV62-8RC:~$ b=((a++))
bash: syntax error near unexpected token `('
szymon@szymon-GV62-8RC:~$ b=$((a++)) ; echo $b
2
szymon@szymon-GV62-8RC:~$ echo $a
3
```

Rysunek 11: Po przypisaniu **b=$((a++))** *b* ma wartość 2 a *a* jest równe 3

3.5 Długość zmiennej

Możemy uzyskać ilość znaków z których składa się zmienna za pomocą komendy **${#a}**. Np. **a=abc; echo ${#a}**.

3.6 Zadania

Zadanie 1 Napisać skrypt który wypisze na ekran "Hello *nazwa_użytkownika*". Następnie przypisać do zmiennej *a* nazwę skryptu, wypisać na ekran. Sprawdzić różnicę w wartości zmiennej *b* w przypadku przypisania b='$a' oraz b="$a".

Zadanie 2 Wewnątrz skryptu przypisać do zmiennej liczbę plików z katalogu domowego oraz wypisać na ekran.

Zadanie 3 Poprosić użytkownika o podanie loginu oraz hasła, znaleźć sposób, aby hasło nie byo widoczne podczas wpisywania.

Zadanie 4 Zmodyfikowac skrypt który pobiera login i hasło w taki sposób, aby wypisać długość podanego hasła

Zadanie 5 Napisać skrypt który jako argument przyjmie liczbę, nastepnie poprosi o podanie drugiej liczby oraz zapisze do pliku wynik sumy, różnicy, dzielenia oraz mnożenia tych liczb.

Zadanie 6 Napisać skrypt, który wywoła sekwencję komend, np. utworzenie struktury katalogów, utworzenie pliku, zapisanie do niego lowowej liczby oraz skopiowanie do kilku innych katalogów oraz drugi skrypt, który usuwa wszystkie utworzone przez poprzedni skrypt pliki oraz katalogi

3.7 Rozwiązania

Zadanie 1
```
#!/bin/bash
echo "Hello $USER" # USER - variable that stores user name
a=$0 #new variable a initialised with script name
echo $a
b='$a'
echo $b
b="$a"
echo $b
```

Zadanie 2
```
#!/bin/bash
cnt=$( ls ~ | wc -l )
echo $cnt
```

Zadanie 3
```
#!/bin/bash
read -p 'Login: ' log
read -sp 'Password: ' pass
echo
echo $log $pass
```

Zadanie 4
```
#!/bin/bash
read -p 'Login: ' log
read -sp 'Password: ' pass
echo ${#pass}
```

Zadanie 5
```
#!/bin/bash
a=$1
read -p 'Second number: ' b
echo $[$a+$b] $(($a-$b)) $(($a*$b)) $[$a/$b] > plik.txt
cat plik.txt
```

Pierwszy

Zadanie 6
```
#!/bin/bash
mkdir -p ./Kat1/Kat2/Kat4 ./Kat1/Kat3
plik="./Kat1/Kat2/Kat4/test.txt"
touch $plik
echo $RANDOM > $plik
cat $plik
cp $plik ./Kat1/plikcp.txt
```

Drugi

```
#!/bin/bash
rm -R ./Kat1/
```