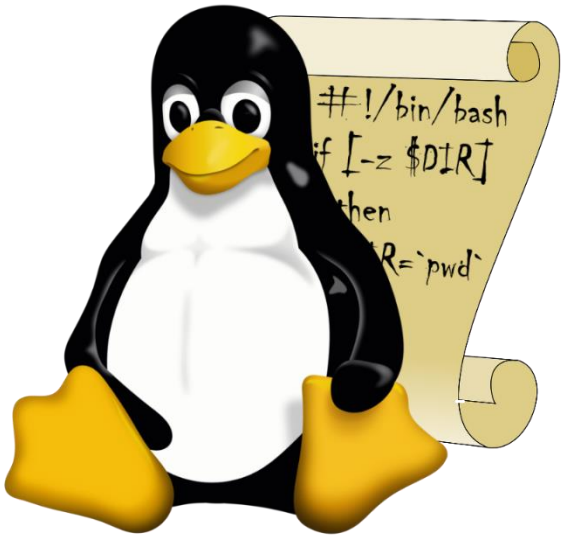


Administrator Linux Skryptowanie w BASH

Podręcznik kursowy

W tym miejscu zwykle pojawia się informacja o tym kto i jak może posługiwać się tym podręcznikiem. Bez regułek prawnych odwołam się po prostu do kilku prostych zasad, które oddają sens kto, kiedy i jak może wg zamysłu autora korzystać z tego materiału:

- Ten podręcznik jest integralnym elementem kursu online.
- Możesz z niego korzystać będąc uczestnikiem tego kursu. Podręcznik jest dla Ciebie i korzystaj z niego do woli – drukuj, wypełniaj, uzupełniaj, przeglądaj, póki Twoim celem jest samodzielne opanowanie tematu.
- Proszę nie umieszczaj go w publicznie dostępnych miejscach, jak blogi, repozytoria git hub, chomik itp.
- Nie wykorzystuj go w innych celach, np. organizacji własnych szkoleń, gdzie występujesz np. jako instruktor.
- Jeśli nie posłuchasz moich próśb, to jako autor zapiszę się do ZAIKS-u i następnym razem kupując smartfona zapłacisz za niego kilkaset złotych więcej 😉, więc może lepiej po prostu przestrzegaj praw autorskich 😊

Z góry dziękuję!

Zapraszam do odwiedzenia strony:

http://www.kursyonline24.eu/

Review 2022-02-07

Spis treści

Pusta strona wstawiona celowo

Wstęp – o kursie

U podstaw Linuxa leżą polecenia uruchamiane w linii komend. Konieczność nauczenia się tych poleceń, to zmora dla tych wszystkich, którzy dopiero rozpoczynają pracę z tym system operacyjnym.

Ale linia komend, ma też swoje ogromne zalety. Jedną z nich jest to, że wystarczy umiejętnie połączyć ze sobą proste komendy, a można stworzyć zupełnie nowe narzędzia, skrypty i funkcje, które mogą zastąpić dziesiątki poleceń. Od tej pory administrator systemu może po prostu uruchomić swój skrypt – i już – gotowe.

Co mogą takie skrypty robić? Generalnie, jeśli jesteśmy w stanie zrobić coś na systemie za pomocą poleceń, to jesteśmy w stanie te polecenia umieścić w skrypcie. Ponieważ na Linuxie da się z linii komend zrobić wszystko, to właściwie wszystko można oskryptować. Powtarzające się kopiowanie plików, instalacja programu, rekonfiguracja sieci, tworzenie użytkowników, analiza logów systemowych, tworzenie raportów i sprawdzanie zabezpieczeń – to tylko kilka przykładów… a można by je mnożyć.

Do efektywnego skryptowania nie wystarczy jednak tylko znajomość samych komend. Oprócz tego trzeba wiedzieć, jak używać zmiennych, jak pracować z instrukcjami warunkowymi, pętlami, jak porównywać liczby i teksty, jak pobierać wyniki innych uruchamianych instrukcji itp. I o tym właśnie nauczysz się na tym kursie.

Znajdziesz tu często nie tylko odpowiedź na pytanie „jak coś zrobić”, ale również „dlaczego tak a nie inaczej”. Zobaczysz różne metody wykonania podobnej czynności i poznasz zalety i wady tych różnych podejść. Nauczysz się tricków stosowanych przez rasowych skrypciarzy. Poznasz elementy architektury, a może raczej „filozofii” budowania aplikacji na Linuxie, zbudujesz funkcje i w oparciu o nie stworzysz własną bibliotekę gotową do wykorzystania w innych skryptach.

Kurs składa się z niezbyt długich lekcji, każda dedykowana innemu tematowi, a do każdej lekcji w kursowym podręczniku dostępnym w postaci PDF znajdziesz krótką notatkę oraz zadania do samodzielnego rozwiązania. W razie czego, są też propozycje rozwiązań. Kurs jest e-learningowym odpowiednikiem tradycyjnego szkolenia w klasie, no dobrze… nie ma poczęstunku – to chyba główna różnica.

Czy warto się uczyć programowania BASH-a? Z moich doświadczeń i przewidywań wynika, że tak. Sam uczyłem się go z 20 lat temu i do dzisiaj go używam – zmian w tym czasie nie było wiele, czego nie da się powiedzieć np. o językach programowania, które zmieniają się z roku na rok – prawie jak moda.

Przerabiając solidnie ten kurs, osiągniesz gotowość do automatyzowania pracy systemu operacyjnego Linux/Unix, a nie ulega wątpliwości, że automatyczne zarządzanie systemem jest i będzie niezwykle pożądane w świecie IT. Rozwój chmury powoduje, że jeden admin nie zarządza kilkoma maszynami, ale dziesiątkami, setkami, tysiącami… - nie wcale nie przesadzam.

Pozostaje mi więc zaprosić do wspólnej nauki. Zapisz się na kurs i pamiętaj, że w razie czego możesz z niego zrezygnować (sprawdź warunki w regulamienie). ----Skryptuj, a reszta nich dzieje się sama!

Powodzenia!

Rafał

Jak się uczyć?

Skoro tutaj zaglądasz, to znaczy, że planujesz samodzielnie skryptować w shellu. To świetnie!

Naukę oczywiście zorganizujesz sobie po swojemu, ale pozwól, że zaproponuję kilka sposobów nauki, a Ty sam/a wybierzesz, co z tego Ci się podoba, a co wolisz zrobić po swojemu

1. **Co za dużo to niezdrowo** – nie rób na raz za dużo materiału. Nie od razu Kraków zbudowano. Jedna lub dwie lekcje na dzień powinny wystarczyć.
2. **Liczy się regularność** – niekoniecznie uczyć trzeba się codziennie, ale jeśli postanowisz przerabiać lekcje we wtorki, czwartki i soboty to już coś!
3. **Wykonuj zadania praktyczne**. Od samego oglądania filmów się nie nauczysz. Trzeba samodzielnie rozwiązywać problemy, które na pewno się pojawią!
4. **Zmieniaj treść poleceń na własną rękę**. Wykonaj podobny przykład na innych danych. Im więcej kreatywności podczas nauki, tym więcej się zapamiętuje
5. Uczę się uczę, a do głowy nie wchodzi – wszyscy tak mamy i to pewnie dlatego nauka w szkole trwa aż tyle lat! **Od czasu do czasu zrób sobie powtórkę**. Przecież nikt Cię nie goni i nie rozlicza z postępów.
6. Sugeruję regularnie wracać do zadań i rozwiązywać je wielokrotnie. Jeśli potrafisz je rozwiązać – świetnie przerabiaj następną lekcję
7. Jeśli zadania sprawiają problem, wróć do notatki lub lekcji – zobaczysz, że słuchając drugi raz tego samego, materiał nie będzie już taki trudny
8. Notatki w podręczniku są dla Twojej wygody. Niestety wygoda leży blisko lenistwa. Nie bądź leniem. Przygotuj sobie zeszyt lub kilka luźnych kartek i **zapisuj to czego się uczysz**. To co wejdzie oczami lub uszami, będzie wychodzić rękami i… nie ma wyjścia – po drodze zahaczy o mózg 😊
9. Jeśli możesz – wydrukuj sobie podręcznik, dopisuj do niego własne notatki, uwagi itp.
10. Kiedy osiągniesz jakiś „kamień milowy”, ukończysz sekcję kursu, a może nawet cały kurs – **daj sobie nagrodę** – to niesamowicie zwiększa motywację!
11. **Nie bój się korzystać z innych materiałów**: książek, blogów, forów itp. Część z nich na początku może być nieco za trudna, ale nic nie stoi na przeszkodzie, żeby na początku nic nie mówić, tylko słuchać 😊.
12. Kiedy już ukończysz kurs – **zaktualizuj CV na LinkedIn**, pochwal się swoim certyfikatem, daj się odnaleźć rekrutrom, zaproś mnie do znajomych (link w profilu). Chętnie potwierdzę Twoją nową umiejętność!

    Powodzenia!

    Rafał

Skrypty? A po co to komu?

Notatka

- Skrypty są integralną częścią systemów Unixowych. Ich przykładowe zastosowanie to:
  - Konfiguracja środowiska
  - Instalacja oprogramowania
  - Automatyzacja pracy
  - Budowanie prostych programów wykorzystywanych w konsoli
- Często wykorzystywane polecenia to: if, while, for, case, funkcje.
- Komentarze poprzedza się znakiem #
- Nazwy zmiennych zwyczajowo zapisuje się wielkimi literami
- Komenda file pozwala ustalić czym jest plik: plikiem binarnym, tekstowym, skryptem
- W oparciu o tzw. shebang (pierwsza linijka testu w pliku) file może rozpoznać plik jako skrypt określonego języka
- SHEBANG dla skryptów bash wygląda następująco:

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

Laboratorium

1. Poczytaj o shebang: https://en.wikipedia.org/wiki/Shebang_(Unix) Zwróć uwagę na historię w ostatnich akapitach
2. Ustal czym jest plik (o ile te pliki występują w Twojej dystrybucji Linuxa):
   a. /sbin/ifcfg
   b. /usr/sbin/dhclient-script
   c. /usr/sbin/ifstat
3. Przejrzyj plik /sbin/ifcfg i zwróć uwagę na (mam nadzieję) znane Ci już z filmu elementy języka. Nie spędzaj nad tym zadaniem zbyt wiele czasu, nie wczytuj się w plik. Na to jeszcze przyjdzie czas.

Propozycja rozwiązania (tylko polecenia)

```
file /sbin/ifcfg
file /usr/sbin/dhclient-script
file /usr/sbin/ifstat
```

Pierwszy skrypt – czasami lepiej zacząć od początku

Notatka

- Pliki skryptów:
    - Mogą mieć rozszerzenie sh, ale nie jest to obowiązkowe
    - Bardzo często w pierwszej linijce ma tzw. shebang określający, jaki interpreter ma wykonywać skrypt. Postać shebang dla skryptu BASH to:

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

- Skrypty są wrażliwe na wielkość liter oraz np. na dodatkowe spacje
- Dobrym zwyczajem jest, aby skrypt kończył się instrukcją exit określającą kod wyjścia, np.:

```
exit 0
```

    - Kod wyjścia zero oznacza brak błędu
    - Kod inny niż zero oznacza błąd. Stosowanie różnych niezerowych kodów wyjścia pozwala sygnalizować różne napotykane w skrypcie błędy
- Kod wyjścia skryptu lub programu można sprawdzić komendą:

```
echo $?
```

- Aby uruchomić skrypt:
    - Należy odwołać się do niego podając ścieżkę względną lub bezwzględną lub umieścić ten skrypt w jednym z katalogów wskazywanych przez zmienną środowiskową PATH
    - Należy nadać uprawnienie execute to pliku skryptu, np.:

```
chmod u+x script.sh
```

Laboratorium

1. Napisz skrypt, który:
    a. Przypisze zmiennej SENTENCE wartość ”If cats could talk, they wouldn’t.”
    b. Wyświetli tą zmienną

    Staraj się przy tym zachować wszystkie dobre praktyki przedstawione w lekcji.
2. Uruchom ten skrypt

Propozycja rozwiązania (tylko kod skryptu)

```
#!/bin/bash


SENTENCE="If cats could talk, they wouldn’t."
echo $SENTENCE

exit 0
```

Skrypty zmienne są! Korzystanie ze zmiennych

Notatka

- Zmienne pozwalają na przechowywanie informacji w pamięci. Raz zdefiniowana zmienna może być wielokrotnie wykorzystywana
- Do wyświetlenia wszystkich zmiennych można użyć polecenia set
- Oto instrukcje do utworzenia zmiennej i do wyświetlenia jej wartości:

```
IMPORT_DIR=/tmp/import
echo $IMPORT_DIR
```

- Konstruując napisy, w których występują zmienne pamiętaj o kilku zasadach:
    - Nazwy zmiennych są zastępowane ich wartością, tylko jeśli konstruowany napis jest zamknięty w cudzysłowy
    - Nazwy zmiennych nie są zastępowane ich wartością, gdy ten napis jest zamknięty w apostrofach
    - Jeśli część zmiennych ma być zamieniona na wartość, a część nie, to napis umieść w cudzysłowie, ale przed zmiennymi, których nie chcesz interpretować poprzedź znak $znakiem cytowania \

```
echo ”this is $IMPORT_DIR”
echo ‘this is $IMPORT_DIR’
echo “the value of \$IMPORT_DIR is $IMPORT_DIR”
```

- Do zapisania informacji w logu systemowym (/var/log/messages lub /var/log/syslog) można korzystać z polecenia logger

```
logger ”Starting installation”
```

Laboratorium

1. Twoim zadaniem jest napisanie skryptu, który przeniesie pliki generowane przez system monitoringu z katalogu, w którym pliki te są umieszczane przez kamery do archiwum
2. Oto nasze założenia:
    a. Katalog źródłowy to /tmp/ftp_data (w prawdziwym przypadku byłby to pewnie inny katalog – ale.. to tylko ćwiczenie)
    b. Katalog docelowy to /tmp/monitor_arch
    c. Przed rozpoczęciem przenoszenia plików, należy wysłać informację do loga systemowego, podobnie należy zrobić na koniec działania skryptu
    d. Skrypt powinien być „gadatliwy” i informować użytkownika o tym co się dzieje
    e. Zakładamy, że odpowiednie katalogi już istnieją
    f. Staraj się maksymalnie korzystać ze zmiennych
3. Przetestuj działanie skryptu: utwórz katalogi, wgraj kilka plików do ftp_data i uruchom skrypt

Propozycja rozwiązania

```
# przygotowanie katalogów i plików:

mkdir /tmp/ftp_data
mkdir /tmp/monitor_arch

touch /tmp/ftp_data/1.txt
touch /tmp/ftp_data/2.txt




# zawartość skryptu:

cat > archive.sh
#!/bin/bash

INPUT_DIR=/tmp/ftp_data
OUTPUT_DIR=/tmp/monitor_arch

logger "Starting moving files from $INPUT_DIR to $OUTPUT_DIR"
echo "Starting moving files from $INPUT_DIR to $OUTPUT_DIR"

mv $INPUT_DIR/* $OUTPUT_DIR

echo "done"
logger "Moving files from $INPUT_DIR to $OUTPUT_DIR finished!"

exit 0




# przygotowanie i uruchomienie skryptu
chmod u+x archive.sh
./archive.sh



# sprawdzenie wyników
sudo tail -f /var/log/syslog
ls -R /tmp
```

Sztuczki ukryte w .bash_profile i .bashrc

Notatka

- Środowisko użytkownika jest budowane przez kilka skryptów
    - /etc/profile – uruchamiany dla każdego użytkownika po zalogowaniu, plik systemowy
    - .bash_profile – uruchamiany dla użytkownika po zalogowaniu, może być konfigurowany przez użytkownika
    - .bashrc – uruchamiany dla użytkownika każdorazowo po uruchomieniu bash-a, może być konfigurowany przez użytkownika
- Domyślnie obecne w shellu zmienne nie są przekazywane do procesów potomnych. Żeby proces potomny „zobaczył” zmienną zdefiniowaną w procesie rodzicielskim należy ją wyeksportować:

```
export INSTALL_DIR=/usr/finbook
```

- Skrypt można podzielić na dwie części: statyczne definiowanie zmiennych w jednym pliku, dynamiczny kod skryptu w drugim pliku. Następnie zawartość pliku statycznego można wczytać w skrypcie korzystając z polecenia source lub „.” (kropka)

```
source config.sh
```

Laboratorium

1. Choć jesteś zaprawionym w bojach fanatykiem linii komend, to pozazdrościłeś kolegom z działu Windows możliwości podejrzenia kalendarza. Dodaj do swojego pliku profilu polecenie wyświetlające kalendarz na bieżący miesiąc (cal)
2. Tworzysz aplikację, która co kilka godzin ma pobrać zawartość strony: http://feeds.bbci.co.uk/news/world/rss.xml
   i zapisać ją w pliku news.xml. Przekonaj się, że to zadanie jest poprawnie realizowane przez komendę:

```
wget --quiet --output-document=news.xml  http://feeds.bbci.co.uk/news/world/rss.xml
```

3. Napisz skrypt, który pobierze zawartość strony i zapisze ją w pliku, ale:
    a. Adres strony do pobrania powinien być zapisany jako parametr WWW_ADDRESS w osobnym pliku
    b. Nazwa pliku powinna również być zapisana w tym pliku jako zmienna FILE_NAME
4. Przetestuj działanie skryptu

Propozycja rozwiązania

```
cat > config.sh
WWW_ADDRESS=http://feeds.bbci.co.uk/news/world/rss.xml
FILE_NAME=news.xml
<CTRL+D>



cat > download_news.sh
#!/bin/bash

source ./config.sh

echo "Downloading $WWW_ADDRESS to $FILE_NAME..."
wget --quiet --output-document=$FILE_NAME $WWW_ADDRESS
echo "Done!"

exit 0
<CTRL+D>



chmod u+x download_news.sh



./download_news.sh
```

Jak liczyć w BASH?

Notatka

- Najpopularniejsza i najwygodniejsza metoda wykonywania obliczeń to stosowanie składni z $ i podwójnymi nawiasami:

```
x=8
y=4
echo $((x+y))
```

- Ta składnia pozwala na wykonanie większej liczby operacji za jednym zamachem. Dodatkowo, można w niej wykonywać przypisanie wyników do zmiennych. Wartość zwracana przez takie wyrażenie to wynik ostatniego działania:

```
echo $((z=x+y, u=x-y, v=u/z))
```

- Można nawet wykonywać proste porównania. (Uwaga 1 – oznacza True, 0 oznacza False)

```
echo $((x==y))
echo $((x>y))
```

- Starsza metoda na obliczenia to polecenie expr. Tu nie można opuszczać znaku $ przed zmiennymi ani usuwać spacji między składowymi wyrażenia. Działanie mnożenia (*) należy poprzedzić znakiem backslash (\)

```
expr $x + $y
expr $x \* $y
```

- Aby wynik działania zapisać w zmiennej należy skorzystać z $() lub ``

```
z=$(expr $x + $y)
u=`expr $x \* $y`
```

- Do dyspozycji jest też polecenie bc. Jest to interaktywny kalkulator, ale da się go wykorzystać do bardziej zaawansowanych obliczeń w shellu. W tym celu należy skonstruować napis, który zostanie przekazany do bc na wejściu:

```
echo "$x + $y" | bc
```

Laboratorium

1. Na dysku masz plik o wielkości 356 GB. Zamierzasz skompresować ten plik i skompresowaną kopię umieścić na file systemie, na którym pozostało 220 GB wolnego miejsca. Spodziewany wskaźnik kompresji to 60% (wynikowy plik będzie miał wielkość 60% oryginalnego rozmiaru). Czy plik zmieści się na docelowym systemie plików?
2. W pkt. (1) założyliśmy, że wskaźnik kompresji to 60%. Ponieważ jest to wartość orientacyjna, to chcielibyśmy wykonać obliczenia z pewnym marginesem. Zakładamy więc, że na docelowym systemie plików powinno być miejsce na 60% oryginalnego pliku plus 10GB. Czy teraz kopia się zmieści?
   W obliczeniach staraj się używać zmiennych i metod obliczeń BASH

Propozycja rozwiązania

```
FILESIZE=356
FREESPACE=220
COMPRESSPERCENT=60

NEWSIZE=$((FILESIZE*COMPRESSPERCENT/100))

echo $NEWSIZE
213

echo $((FREESPACE>NEWSIZE))
1
```

```
FILESIZE=356
FREESPACE=220
COMPRESSPERCENT=60
MARGIN=10

NEWSIZE=$(((FILESIZE*COMPRESSPERCENT/100)+MARGIN))

echo $NEWSIZE
223

echo $((FREESPACE>NEWSIZE))
0
```

Operacje na napisach

Notatka

- „Cięcie napisów” można wykonywać w oparciu o maskę określającą, co ma być usunięte z napisu:
    - ${VARNAME%/*} - usuń najkrótszy napis pasujący do maski /* idąc od końca napisu
    - ${VARNAME%%/*} - usuń najdłuższy napis pasujący do maski /* idąc od końca napisu
    - ${VARNAME#*/} - usuń najkrótszy napis pasujący do maski*/ idąc od początku napisu
    - ${VARNAME##*/} - usuń najdłuższy napis pasujący do maski*/ idąc od początku napisu
- Można też wycinać podnapis z napisu wskazując, ile znaków w oryginalnym napisie należy usunąć, a następnie, ile skopiować:
    - ${VARNAME:x:y} - opuść x liter, a następnie pobierz y liter (x i y to liczby, które w szczególności mogą być zerem

Laboratorium

1. Komedą touch utwórz nowy plik:

```
touch  /tmp/data.csv
```

2. Zapisz nazwę pliku w zmiennej FILEPATH.
3. Wytnij z FILEPATH tylko samą nazwę katalogu do zmiennej DIRONLY
   Uwaga: Istnieje specjalna komenda dirname, która może to zrobić, ale… powiedzmy, że o tym nie wiesz 😉
4. Wytnij z FILEPATH tylko samą nazwę pliku i umieść ją w zmiennej FILEONLY
   Uwaga: Istnieje specjalna komenda basename, która może to zrobić, ale… powiedzmy, że i tej komendy nie pamiętasz 😉
5. Wytnij z FILEONLY tylko samą nazwę pliku bez rozszerzenia i umieść ją w zmiennej FILENOEXT
6. Utwórz nową zmienną o nazwie NEWPATH, która składać się będzie z:
    a. Nazwy katalogu zapisanej w DIRONLY
    b. Znaku /
    c. Nazwy pliku zapisanej w FILENOEXT
    d. I rozszerzenia .old
7. Korzystając ze zmiennych skopiuj plik wskazywany przez FILEPATH na NEWPATH

Propozycja rozwiązania

```
touch /tmp/data.csv


FILEPATH=/tmp/data.csv


DIRONLY=${FILEPATH%/*}
echo $DIRONLY
/tmp


FILEONLY=${FILEPATH##*/}
echo $FILEONLY
data.csv


FILENOEXT=${FILEONLY%.*}
echo $FILENOEXT
data


NEWPATH=$DIRONLY/$FILENOEXT.old
echo $NEWPATH
/tmp/data.old


cp $FILEPATH $NEWPATH
```

Wejście, wyjście i wyjście błędów

Notatka

- Każdy program w Unixie ma domyślnie dostępne 3 urządzenia/pliki:
    - 0 – standardowe wejście – domyślnie klawiatura
    - 1 – standardowe wyjście -domyślnie ekran
    - 2 – standardowe wyjście błędów – domyślnie ekran
- Te 3 urządzenia można przekierować

```
# przekierowanie wyjścia do pliku
./script  >  ./output.txt
./script 1>  ./output.txt

# przekierowanie wyjścia błędów do pliku
./script 2>errors.txt

# przekieowanie wyjścia błędów do pliku w trybie dopisywania
./script 2>>errors.txt


# przekierowanie wyjścia do pliku i standardowego wyjścia błędów
# do tego samego urządzenia co wyjście skryptu
./script > /output.txt 2>&1

# przekierowanie wyjścia błędów do /dev/null
./script 2>/dev/null

# przekierowanie wejścia do skryptu
./script << EOF
1
2
3
EOF
```

Laboratorium

1. Chcesz zapisać w pliku informacje o tym kto jest zalogowany. Napisz skrypt, który:
    a. Wyświetli aktualną datę i godzinę
    b. Wyświetli nazwę komputera korzystając z polecenia hostname
    c. Wykona polecenie who
2. Po przetestowaniu skryptu skieruj jego wyjście do pliku report.txt
3. Po chwili wykonaj skrypt jeszcze raz dopisując jego wyjście do pliku report.txt
4. Korzystając z polecenia

```
du -sh  /var/*
```

   sprawdź rozmiar podkatalogów katalogu /var. Ponieważ uruchamiasz polecenie bez sudo, można spodziewać się licznych komunikatów o błędach. Prześlij je do /dev/null
5. Czy jednak takim wynikom można ufać? Wykonaj to polecenie jeszcze raz, tym razem przez sudo i porównaj wyniki

Propozycja rozwiązania

```
cat > report.sh
#!/bin/bash

date
hostname
who

exit 0


chmod u+x report.sh

./report.sh > report.txt
./report.sh >> report.txt



du -sh /var/*
4.8M    /var/backups
du: cannot read directory '/var/cache/apt/archives/partial': Permission denied
du: cannot read directory '/var/cache/lightdm': Permission denied
du: cannot read directory '/var/cache/private': Permission denied
du: cannot read directory '/var/cache/cups': Permission denied
…
du: cannot read directory '/var/tmp/systemd-private-971ebb5aeaef4f37906320dff3bdd1bf-
rtkit-daemon.service-sIR8dY': Permission denied
du: cannot read directory '/var/tmp/systemd-private-971ebb5aeaef4f37906320dff3bdd1bf-
systemd-timesyncd.service-y6HEWb': Permission denied
12K     /var/tmp

du -sh /var/* 2>/dev/null
4.8M    /var/backups
737M    /var/cache
146M    /var/lib
4.0K    /var/local
0       /var/lock
798M    /var/log
4.0K    /var/mail
4.0K    /var/opt
0       /var/run
36K     /var/spool
100M    /var/swap
12K     /var/tmp

sudo du -sh /var/* 2>/dev/null
4.8M    /var/backups
737M    /var/cache
146M    /var/lib
4.0K    /var/local
0       /var/lock
798M    /var/log
4.0K    /var/mail
4.0K    /var/opt
0       /var/run
48K     /var/spool
100M    /var/swap
20K     /var/tmp
```

Pobieranie wartości zmiennych

Notatka

- Zmienne w programach i skryptach mają za zadanie przechowywać podręczną informację w tym programie lub skrypcie
- Dużą zaletą korzystania ze zmiennych jest to, że nadają one wartościom liczbowym napisowym nazwę. Ta nazwa ułatwia zadanie programisty i skryptera, któremu łatwiej jest posługiwać się wartością nazwaną MAX_SIZE aniżeli liczbą 1000.
- Zmienne mogą również przechowywać wartości zwrócone przez komendy systemu operacyjnego. Służy do tego składnia $(…)

```
DISK_FREE=$(df -h /home | tr -s ' ' | grep -v 'Filesystem' | cut -d ' ' -f 4)
echo $DISK_FREE
16G
```

- Starsze skrypty posługują się w tym miejscu również „odwróconym apostrofie”:

```
DISK_FREE=`df -h /home | tr -s ' ' | grep -v 'Filesystem' | cut -d ' ' -f 4`
echo $DISK_FREE
16G
```

Laboratorium

1. Wskutek pewnych zależności w skrypcie chcesz wykorzystać zmienną do sprawdzenia wielkości podkatalogów katalogu /var
    a. Przejdź do katalogu /var
    b. Do zmiennej DIRS zapisz wynik polecenia ls
    c. Korzystając z $DIRS oraz polecenia du -sh wyświetl informacje o wielkości wszystkich podkatalogów w /var
    d. Uruchamiając polecenie du skorzystaj z sudo lub przekieruj standardowe wyjście błędów na /dev/null
2. Chcemy policzyć, ile sesji ma nawiązanych do serwera aktualnie zalogowany użytkownik, dlatego:
    a. Do zmiennej ME zapisz wynik polecenia whoami
    b. Do zmiennej SESSION_COUNT zapisz informację o liczbie otwartych sesji dla użytkownika z nazwą zapisaną w $ME:
        i. Uruchom polecenie who, którego wynik prześlij potokiem do…
        ii. Polecenia grep, które wyfiltruje tylko wiersze dotyczące użytkownika $ME i prześle je potokiem do…
        iii. Polecenia wc -l, które policzy, ile takich wierszy jest
        iv. Wyświetl komunikat mówiący o tym, ile sesji ma otwartych bieżący użytkownik

Propozycja rozwiązania

```
cd /var

DIRS=$(ls /var)

sudo du -sh $DIRS
```

```
ME=$(whoami)

echo $ME
pi

who | grep $ME
pi       tty1         2021-07-28 22:32
pi       pts/0        2021-07-31 14:56 (192.168.2.114)

who | grep $ME | wc -l
2

SESSION_COUNT=$(who | grep $ME | wc -l)
echo "$ME has $SESSION_COUNT session(s)"
pi has 2 session(s)
```

Instrukcja test w kilku wydaniach z kilkoma pułapkami

Notatka

- Testowanie wartości logicznych można wykonać na trzy sposoby (Uwaga na składnię! Każda spacja się liczy!):
    - test jest najstarszą formą i ze względu na to należy go raczej unikać (ale działa – stary ale jary)
    - [] – następca polecenia test, zaimplementowany we wszystkich obecnych shellach
    - [[ ]] – wprowadza pewne usprawnienia (np. bardziej intuicyjne operacje na napisach lub operacje logiczne && ||). Jeśli nie masz problemów z kompatybilnością używaj tej instrukcji

```
test -f /etc/passwd
[ -f /etc/passwd ]
[[ -f /etc/passwd ]]
```

- Polecenie test może wykonywać różnego rodzaju testu (pełna lista w man test):
    - -f plik istnieje i jest zwyczajnym plikiem
    - -d katalog istnieje
    - $x -gt $y x jest większe niż y
    - $x -le $y x jest mniejsze lub równe y
    - -z ”$x” napis x jest pusty
    - -n ”$x” napis x jest niepusty
    - $x = $y napis x jest równy napisowi y (w składni [[ ]] należy używać podwójnego znaku równości ==)
    - -a AND – koniunkcja – w przypadku [[ ]] należy używać &&
    - -o OR – alternatywa – w przypadku [[ ]] należy używać ||
- Wynik testu znajduje się w zmiennej $? Wartość 0 oznacza prawdę , a 1 fałsz. Odczyt wyniku należy wykonać jako kolejną instrukcję w kodzie.

Laboratorium

1. Zdefiniuj zmienne LIMIT=16, AVERAGE=10, PEAK=20 i USED=15 i napisz polecenia testujące:
    a. Czy USED jest mniejsze niż LIMIT
    b. Czy USED jest większe lub równe od AVERAGE
    c. Czy LIMIT jest większe niż PEAK
    d. Czy USED jest mniejsze niż LIMIT i większe niż AVERAGE
2. Zdefiniuj zmienną MEDIA=/media/cdrom i TMPLOC=/tmp i sprawdź:
    a. Czy istnieje katalog MEDIA
    b. Czy istnieje plik TMPLOC
    c. Czy istnieje katalog TMPLOC
3. Nie definiując wcześniej zmiennej NAME, sprawdź, czy napis zapisany w takiej zmiennej jest pusty, czy niepusty.
4. Wpisz do zmiennej NAME imię Max i powtórz poprzednie testy

Propozycja rozwiązania

```
LIMIT=16
AVERAGE=10
PEAK=20
USED=15

test $USED -lt $LIMIT
echo $?
0

[ $USED -ge $AVERAGE ]
echo $?
0

[[ $LIMIT -gt $PEAK ]]
echo $?
1

[[ $USED -lt $LIMIT && $USED -gt $AVERAGE ]]
echo $?
0



MEDIA=/media/cdrom
TMPLOC=/tmp

test -d $MEDIA
echo $?
1

test -f $TMPLOC
echo $?
1

test -d $TMPLOC
echo $?
0




[[ -z $NAME ]]
echo $?
0

[[ -n $NAME ]]
echo $?
1

NAME=Max

[[ -z $NAME ]]
echo $?
1

[[ -n $NAME ]]
echo $?
0
```

Decyzja w skrypcie – instrukcja if

Notatka

- If pozwala na warunkowe wykonanie części skryptu
    - Polecenie rozpoczyna się od if, po którym znajduje się warunek do sprawdzenia
    - Po słowie then występują instrukcje do wykonania, jeśli warunek był spełniony
    - Opcjonalnie może się dalej wielokrotnie pojawić instrukcja elif, która podobnie jak if sprawdza warunek i wykonuje wskazane komendy, jeśli ten warunek jest spełniony
    - Opcjonalnie na końcu może się pojawić instrukcja else. Polecenia zostaną wykonane, jeśli żaden z wcześniejszych testów if/elif nie został spełniony
    - Składnia wymaga zakończenie wyrażenia słowem fi

```
if [[ -d $DIR ]]
then
   echo "Directory $DIR already exists. Nothing to do."
elif [[ -e $DIR ]]
then
   echo "Path $DIR already exists, but it is not a directory!"
else
   echo "Creating directory $DIR"
   mkdir $DIR
fi
```

Laboratorium

1. Przygotuj środowisko do LAB uruchamiając komendy:

```
touch /tmp/output.txt
sudo groupadd DEV
sudo useradd mat -g DEV
```

2. Napisz skrypt permissions.sh, a w nim:
    a. Utwórz zmienne:

```
USER=mat
GROUP=DEV
FILE=/tmp/output.txt
```

    b. Sprawdź, czy USER lub GROUP są pustymi napisami. Jeśli tak, wyświetl komunikat i wyjdź ze skryptu z kodem wyjścia 11
    c. Sprawdź, czy FILE jest istniejącym plikiem
        i. jeśli tak, wyświetl komunikat o zmianie właściciela dla **pliku** i zmień właściciela pliku FILE na użytkownika USER i grupę GROUP
        ii. w przeciwnym razie, jeśli jest katalogiem, to wyświetl komunikat o zmianie właściciela **katalogu** i zmień właściciela rekurencyjnie dla tego katalogu na użytkownika USER i grupę GROUP
        iii. w przeciwnym razie wyświetl komunikat, mówiący, że FILE jest niepoprawny i zakończ skrypt z kodem 12
3. Przetestuj działanie skryptu. Pamiętaj o korzystaniu z sudo. Przetestuj działanie różnych kombinacji wartości zmiennych prowadzących do różnych błędów

Propozycja rozwiązania

```
#!/bin/bash

USER=mat
GROUP=DEV
FILE=/tmp/output.txt


if [[ -z $USER || -z $GROUP ]]; then
   echo "USER and GROUP cannot be empty"
   exit 11
fi


if [[ -f $FILE ]]; then
   echo "Changing permissions to file"
   chown $USER:$GROUP $FILE
elif [[ -d $FILE ]]; then
   echo "Changing permissions to directory"
   chown -R $USER:$GROUP $FILE
else
   echo "The FILE is incorrect"
   exit 12
fi

exit 0
```

Jak pracować z kodem wyjścia zewnętrznego programu

Notatka

- Skrypt może definiować listę zmiennych liczbowych o przypisanych wartościach odpowiadającym wszystkim możliwym kodom wyjścia. Zgłaszając kod wyjścia, należy się wtedy posługiwać wartościami zmiennych, których nazwy są bardziej opisowe niż sama liczba.
- Każde z poleceń wykonywanych przez skrypt również zwraca kod wyjścia. Zgodnie z unixowym standardem jest to
    - Zero w przypadku powodzenia
    - Wartość niezerowa w przypadku porażki
- Jeśli chcesz przechwycić błąd zwrócony przez inne polecenie, to po wykonaniu tej komendy sprawdź wartość zmiennej $?

```
mkdir $CAT
RESULT=$?
if [[ $RESULT -ne 0 ]]; then
   exit 1
fi
```

- Ten zapis można skrócić:

```
mkdir $CAT
if [[ $? -ne 0 ]]; then
   exit 1
fi
```

- A nawet wywoływać zewnętrzne polecenie od razu w warunku instrukcji if:

```
if mkdir $CAT; then
   exit 1
fi
```

Laboratorium

1. Do skryptu z poprzedniego LAB dodaj obsługę błędów związaną ze:
    a. Zmianą właściciela pliku
    b. Zmianą właściciela katalogu
2. Przetestuj skrypt w różnych scenariuszach

Propozycja rozwiązania

```
#!/bin/bash

USER=mat
GROUP=DEV
FILE=/tmp/output.txt


if [[ -z $USER || -z $GROUP ]]; then
   echo "USER and GROUP cannot be empty"
   exit 11
fi


if [[ -f $FILE ]]; then
   echo "Changing permissions to file"
   chown $USER:$GROUP $FILE
   if [[ $? -ne 0 ]]; then
      exit 13
   fi
elif [[ -d $FILE ]]; then
   echo "Changing permissions to directory"
   chown -R $USER:$GROUP $FILE
   if [[ $? -ne 0 ]]; then
      exit 13
   fi
else
   echo "The FILE is incorrect"
   exit 12
fi

exit 0
```

Skrócona forma wyrażenia warunkowego – to logiczne!

Notatka

- BASH interpretując wartości logiczne
    - Analizuje wyrażenie od lewej do prawej
    - Jeśli w oparciu o do tej pory uzyskane wyniki wartość wynikowa wyrażenia jest już znana, to BASH rezygnuje w szacowania pozostałych składników wyrażenia
- Każde polecenie zwraca wynik, który można interpretować, jako wartość logiczną
- Dzięki temu można skrócić zapis wyrażenia warunkowego if do jednej linijki:
    - Jeśli użytkownik yoda istnieje, to echo się nie wykona, ale jeśli użytkownik NIE istnieje, to echo się uruchomi

```
grep yoda /etc/passwd || echo "No such user"
```

    - Jeśli użytkownik yoda istnieje, to echo się wykona, ale jeśli użytkownik NIE istnieje, to echo się nie uruchomi:

```
grep yoda /etc/passwd && echo "No such user"
```

- W ten sposób można ze sobą połączyć nawet więcej niż tylko dwa warunki
- Ta składnia pozwala w zwięzły sposób zapisać proste wyrażenia if

Laboratorium

1. Napisz polecenie, które w przypadku braku katalogu /tmp/workdir, utworzy go. Zrób to na 2 sposoby (w obu wykorzystując zaprezentowaną w tej lekcji technikę)
    a. Raz zastosuj operator &&
    b. A raz zastosuj operator ||
2. Napisz polecenie, które w przypadku istnienia pliku /tmp/stop.txt wyświetli komunikat mówiący o tym, że nie można kontynuować pracy skryptu i zakończy go zwracając kod wyjścia 1

Propozycja rozwiązania

```
[ ! -d /tmp/workdir ] && mkdir /tmp/workdir
[ -d /tmp/workdir ] ||  mkdir /tmp/workdir



[ -f /tmp/stop.txt ] && echo "Cannot continue" && exit 1
```

1, 2, 3 – pętla for - wprowadzenie

Notatka

- for pozwala wykonać czynność określoną liczbę razy:

```
for i in {1..5}; do
   echo $i
done
```

- Pętle for można zagnieżdżać:

```
for i in {1..5}; do
   for j in {1..5}; do
      echo "$i * $j = $((i * j))"
   done
done
```

- For może uruchamiać polecenie dla każdej linijki w pliku tekstowym:

```
for server in $(cat ./servers.txt); do
   echo $server
done
```

- Polecenie for może przetwarzać pliki znajdujące się w określonym katalogu

```
for file in files/*.txt; do
    echo $file
done
```

- For może również iterować po zmiennych

```
DAYS="Mon Tue Wed Thu Fri Sat Sun"

for day in $DAYS; do
  echo $day
done
```

Laboratorium

1. Napisz pętlę wyświetlającą nazwy plików z katalogu /var/log. (Wiem, że można użyć ls, ale zróbmy to pętlą, please)
2. Napisz pętlę przetwarzającą każdą linijkę z pliku /etc/fstab (każda linijka odpowiada za jeden montowany system plików). Linijki tego pliku zawierają spacje, niestety for widząc spacje traktuje słowa jako osobne elementy. Aby rozwiązać ten problem dodaj następujące polecenie w skrypcie, gdzieś przed rozpoczęciem przetwarzania danych:

```
IFS=''
```

   Powoduje ona, że spacja traci swoje specjalne właściwości i jest traktowana, jako zwykły znak
3. Napisz pętlę, która dla liczb od 1 do 20 wyświetli tą liczbę oraz jej kwadrat (mało administracyjny przykład, ale i takie się zdarzają….)

Propozycja rozwiązania

```
#!/bin/bash

DIR=/var/log

for m in $DIR/*; do
   echo  $m
done

exit 0
```

```
#!/bin/bash

FILE=/etc/fstab
IFS=''

for line in $(cat $FILE); do
   echo "$line"
done

exit 0
```

```
#!/bin/bash

for i in {1..10}; do
   echo "$i   $(( $i * $i ))"
done

exit 0
```

Pętla for - zastosowania

Notatka

- for dobrze nadaje się do zastosowania w scenariuszach, gdzie dana czynność musi być wykonana określoną liczbę razy:
    - Dla każdego pliku w katalogu
    - Dla każdego wiersza w pliku
    - Dla każdego słowa w napisie
- Chociaż podobne czynności można też oprogramować stosując inne rodzaje pętli, to for wydaje się być najnaturalniejszym wyborem, bo od samego początku jest znana liczba powtórzeń wykonania tej pętli

Laboratorium

1. Utwórz plik folders.txt, w którym w każdej linijce znajduje się nazwa określająca nazwę jednego folderu. W kolejnym kroku będziemy tworzyć podkatalogi w /tmp o wymienionych tu nazwach, użyj więc takich nazw, aby operacja tworzenia podkatalogów się udała.
2. Napisz pętlę for, która przetworzy wszystkie linijki tego pliku i dla każdej z nich utworzy podkatalog w katalogu /tmp
3. Utwórz plik folders-du.txt i kolejnych linijkach podaj nazwy istniejących folderów, np.:
    a. /etc
    b. /usr/lib
    c. /var/mail
    d. /tmp
    e. …
4. Napisz pętlę for, która dla każdej linijki tego pliku (czyli dla każdego wymienionego tam katalogu) uruchomi polecenie du . Zadbaj o to, aby ewentualne błędy dotyczące braku uprawnień do plików znajdujących się w tych katalogach zostały ukryte.

```
du -sh
```

Propozycja rozwiązania

```
#!/bin/bash

FILE=folders.txt
DIR=/tmp

for folder in $(cat $FILE); do

    CATALOG=$DIR/$folder
    echo "Creating directory $CATALOG"
    mkdir $CATALOG

done

exit 0


#!/bin/bash

FILEDU=./folders-du.txt

for folder in $(cat $FILEDU); do

   du -sh $folder 2> /dev/null

done
```

While - wprowadzenie

Notatka

- while to pętla, która wykonuje się tak długo, jak warunek pętli jest prawdziwy
- Pętla while jest zazwyczaj wybierana, kiedy nie wiadomo, ile razy pętla powinna się wykonać, chociaż po odpowiednim skonstruowaniu warunków, może pracować jak pętla for
- while może się wykonywać określoną liczbę razy:

```
i=0
while [[ $i -lt 10]]; do
      echo ”$((++i))”
done
```

- while może wykonywać się dla każdej linii pliku wejściowego:

```
while read line; do
      echo line
done < my_file.txt
```

Laboratorium

1. Chcesz zatrzymać skrypt na ok. 10 sekund. Chcesz jednak, aby w czasie tego „zatrzymania”, mniej więcej co 1 sekundę na ekranie była wyświetlana kropka, wskazująca, że skrypt się nie zawiesił, tylko czeka na coś. Wiedząc, że
    a. sleep 1 zatrzymuje skrypt na ok. 1 sekundę
    b. echo -n ‘.’ Wyświetla znak kropki bez przejścia do nowej linijki

    Napisz skrypt, który zatrzyma się na wskazany czas, wyświetlając co sekundę kropkę

Propozycja rozwiązania

```
#!/bin/bash

MAXTIME=10
TIME=1

while [[ $TIME -le $MAXTIME ]]; do

  echo -n '.'
  sleep 1
  TIME=$(( $TIME + 1 ))

done

echo ''

exit 0
```

Pętla while - zastosowanie

Notatka

- while może w wygodny sposób obsłużyć sytuację oczekiwania skryptu na pewne zdarzenie, do którego powinno dojść na systemie:
    - Oczekiwanie na pojawienie się pliku
    - Oczekiwanie na to aż serwer będzie odpowiadał
    - Oczekiwanie na wylogowanie się użytkownika
- Do prawidłowego zbudowania określonego warunku należy w dobry sposób zdefiniować warunek sterujący wykonaniem pętli while
- Zarówno while, jak i for mogą być zagnieżdżane

Laboratorium

1. Jeśli nie masz na systemie żadnych dodatkowych użytkowników, to dodaj jednego komendą:

```
sudo useradd helen
sudo passwd helen
```

2. Napisz skrypt, który będzie wyświetlał kropki tak długo, aż na systemie zaloguje się użytkownik helen. Do sprawdzania, czy użytkownik jest zalogowany, możesz wykorzystać wynik polecenia:

```
who | grep helen
```

   Warunek pętli może sprawdzać, czy wynik tego polecenia jest napisem pustym. Gdzie można, staraj się używać zmiennych
3. Na zakończenie (czyli po zalogowaniu użytkownika helen) wyświetl komunikat i wygeneruj w terminalu dźwięk. Powinno się to udać zrobić poleceniem

```
echo -e "\\07"
```

   (Jeśli wydawanie dźwięku nie działa, to może to zależeć od wykorzystywanego terminala – w każdym razie nie spędzaj na tym zbyt wiele czasu…)
4. Przetestuj działanie skryptu:
    a. Uruchom skrypt – powinien on co sekundę wyświetlać kropkę
    b. W innym terminalu zaloguj się jako helen – skrypt powinien się zakończyć i powinien zostać wygenerowany dźwięk

Propozycja rozwiązania

```
#!/bin/bash

USER='helen'

while [[ -z $(who | grep $USER) ]]; do

   echo -n '.'
   sleep 1

done


echo -e "\\07"
echo "$USER is on the system"
echo -e "\\07"
```

Tablice - arrays

Notatka

- Tablice (arrays) mogą służyć do chwilowego przechowania danych skryptu (np. zajętość poszczególnych systemów plików, sumaryczna przestrzeń dyskowa zajmowana przez użytkowników itp.)
- Składnia wykorzystywana w pracy z tablicami jest bardzo specyficzna:

```
# Definiowanie grupy:
GROUPS=(office managers support)

# Wyswietlenie pozycji 0 – UWAGA! – Numeracja od zera!
echo ${GROUPS[0]}

# Wyswietlenie ostatniego elementu:
echo ${GROUPS[-1]}

# Wyswietlenie wszystkich elementow listy:
echo ${GROUPS[@]}

# Wyswietlenie indeksów elementow listy:
echo ${!GROUPS[@]}

# Iteracja for po tablicy:
for group in ${GROUPS[@]}: do
  echo $group
done

# Wyświetlenie informacji o liczbie elementow tablicy:
echo ${#GROUPS[@]}

# Zmiana wartości na pozycji 1
GROUPS[1]=’admins’

# Dodanie nowej pozycji
GROUPS+=(‘devops’)

# Pobranie 4 elementow listy od 2-giego
echo ${GROUPS[@]:2:4}
```

Laboratorium

1. Przygotowujesz raport, który ma wyświetlać pewne informacje na każdy dzień tygodnia. Raport będzie czytany przez… Włochów. Zainicjuj tablice weekdays nazwami dni roboczych:
   Lunedi Martedi Mercoledi Giovedi Venerdi
2. Ops.. weekend też należy uwzględnić. Dodaj jeszcze: Sabato Domenica
3. Numer dnia tygodnia dla dnia dzisiejszego może być pobrany poleceniem date. Zapisz wynik tego polecenia do zmiennej today:

```
Date  +%w
```

4. Wyświetl włoską nazwę dnia tygodnia korzystając z $today i $weekdays
5. Wyświetl nazwy roboczych dni tygodnia.

Propozycja rozwiązania

```
weekdays=(Lunedi Martedi Mercoledi Giovani Venerdi)

echo ${weekdays[@]}
Lunedi Martedi Mercoledi Giovani Venerdi

weekdays+=(Sabato Domenica)

echo ${weekdays[@]}
Lunedi Martedi Mercoledi Giovani Venerdi Sabato Domenica

date +%w
5

today=$(date +%w)

echo $today
5

echo ${weekdays[$today]}
Sabato

echo ${weekdays[@]:0:5}
Lunedi Martedi Mercoledi Giovani Venerdi
```

Array - zastosowanie

Notatka

- Tablice przydają się do tymczasowego przechowywania większej ilości danych w skrypcie
- Dane te mogą pochodzić z interakcji z użytkownikiem, ale równie dobrze mogą być statycznymi wartościami w skrypcie lub wynikiem uruchomienia innych poleceń.
- Dane mogą być zapisywane do tablic w jednym miejscu skryptu, a następnie przetwarzane w innym miejscu, zazwyczaj przy użyciu dedykowanej do tego pętli for lub while

Laboratorium

1. Ten przykład nie pracuje na obiektach array, ale prezentuje podobne kroki, które należałoby wykonać, gdy pracuje się z array.
2. Napisz skrypt, który dla każdego zalogowanego użytkownika policzy, ile procesów ten użytkownik uruchomił.
    a. Polecenia, które się do tego mogą przydać to:
        i. Pobranie listy aktualnie zalogowanych użytkowników:

```
who | cut -d ' ' -f 1 | uniq
```

        ii. Wyznaczenie liczby procesów dla użytkownika helen:

```
ps --no-headers -u helen | wc -l
```

    b. Najpierw zapisz listę aktualnie zalogowanych użytkowników do zmiennej USERS. Możesz skorzystać z polecenia (i)
    c. Następnie pętlą for przejdź przez listę USERS i dla każdego użytkownika policz liczbę uruchomionych procesów korzystając z polecenia (ii)
    d. Wynik wyświetl na ekranie

Propozycja rozwiązania

```
#!/bin/bash

USERS=$(who | cut -d ' ' -f 1 | uniq)

for u in $USERS; do

   count=$(ps --no-headers -u $u | wc -l)
   echo "$u - $count"

done

exit 0
```

Argumenty skryptu

Notatka

- Przekazując argumenty do skryptu podaje się ich wartość po nazwie skryptu
- Wewnątrz skryptu, można uzyskiwać informacje o wartości parametrów poprzez zmienne:
    - $0 – nazwa skryptu
    - $1. $2. $3,… ${10}, ${11}… - wartości argumentów
    - $# - liczba argumentów
    - $@ - lista argumentów (array)
    - $* - lista argumentów (napis porozdzielany spacjami)
- Kiedy chcesz przetworzyć wszystkie argumenty po kolei, to można stworzyć pętlę, która:
    - Będzie wykonywać się tak długo, jak $# jest większe od 0
    - Będzie pobierać wartość argumentu zawsze ze zmiennej $1
    - Wykona przesunięcie argumentów poleceniem shift (które to polecenie zmieni wartość $1 i $#)

```
#!/bin/bash

while [[ $# -gt 0 ]]; do

   echo $1
   shift

done
```

Laboratorium

1. W lekcji „Pętla for – zastosowania” pisaliśmy skrypt, który obliczał rozmiary katalogów zapisanych w pliku tekstowym. Skrypt wyglądał tak:

```
#!/bin/bash

FILE=folders.txt
DIR=/tmp

for folder in $(cat $FILE); do

    CATALOG=$DIR/$folder
    echo "Creating directory $CATALOG"
    mkdir $CATALOG

done

exit 0
```

2. Zmień go tak, aby katalogi do sprawdzenia można było przekazać poprzez argumenty. Skrypt ma być w stanie przetworzyć dowolną liczbę katalogów (czyli musisz zastosować jakiś rodzaj pętli)
3. Przetestuj działanie skryptu. Podnosząc poprzeczkę, postaraj się, aby skrypt radził sobie również z folderami, które w nazwie zawierają spacje.
   Wskazówka – w czasie testów zrezygnuj z przekierowania standardowego wyjścia błędów na /dev/null co pozwoli analizować wyświetlane błędy, które inaczej są ukryte…

Propozycja rozwiązania

```
#!/bin/bash
for folder in "$@"; do
   du -sh "$folder"        # 2> /dev/null
done



sudo mkdir /home/folder\ with\ space
./du.sh /tmp /root ‘/home/folder with space’
```

Przyjmowanie argumentów – instrukcja case

Notatka

- Skrypt przyjmujący parametry może mieć następujące „sekcje”
  - Deklaracja zmiennych (czasami początkowe wartości przypisane do zmiennych mogą być niepoprawne – dalsza część skryptu powinna je weryfikować)
    - Pętla while z warunkiem. W tej pętli dochodzi do przesunięcia argumentów (instrukcja shift), dlatego $1 wskazuje na „w danej chwili przetwarzany argument”

```
while [[ $# -gt 0 ]]; do
```

    - W pętli, wyrażenie case analizuje kolejne argumenty zapisane w zmiennej $1. Analizując opcje należy zmieniać wartości zdefiniowanych wcześniej zmiennych
    - Opcje analizowane przez case mogą wskazywać na krótką lub długą ich nazwę
    - Jeśli za jednym zamachem chcesz przeczytać opcję i jej wartość, to opcja znajduje się w $1, a wartość w $2. W takim przypadku należy wykonać dwukrotnie shift
    - Inne/niepoprawne opcje są obsługiwane przez symbol *

```
case $1 in
   -r|--report)
      echo ”using -r”
      MAKE_REPORT=’Y’
      shift
      ;;
   # put here other options
   *)
      echo “incorrect option $1”
      exit 1
   ;;
esac
```

    - Sprawdzenie poprawności opcji
    - Pozostała, właściwa część skryptu korzystająca ze zmiennych

Laboratorium

1. Napisz skrypt, który wyświetli informacje o użytkowniku, akceptujący następujące parametry:
   a. -n <nazwa_uzytkownika> - w ten sposób powiemy, jaki użytkownik ma być analizowany
   b. -i spowoduje wykonanie polecenia id <nazwa_uzytkownika>
   c. -l spowoduje wykonanie polecenia last <nazwa użytkownika>
   d. -d spowoduje wykonanie polecenia du -sh /home/<nazwa_uzytkownika>
   e. -h spowoduje wyświetlenie helpa
   f. Przekazanie niepoprawnych opcji ma powodować błąd
   g. Przed uruchomieniem poleceń, sprawdź, czy nazwa użytkownika jest niepusta
   h. Uwaga: część w/w poleceń musi być uruchamiania przez sudo, dlatego wywołując ten skrypt poprzedź go przez sudo
2. Zachowaj rozwiązanie do kolejnego laboratorium

Propozycja rozwiązania

```
#!/bin/bash


USERNAME=''
RUN_ID='N'
RUN_LAST='N'
RUN_DU='N'


while [[ $# -gt 0 ]]; do

  case $1 in

    -i|--id)
      echo "id will be started"
      RUN_ID='Y'
      shift
      ;;
    -l|--last)
      echo "last will be started"
      RUN_LAST='Y'
      shift
      ;;
    -d|--du)
      echo "du will be started"
      RUN_DU='Y'
      shift
      ;;
    -h|--help)
      echo "USAGE:  $0 [-i|--id] [-l|--last] [-d|--du] [-n|--name <user_name>]"
      exit 0
      ;;
    -n|--name)
      echo "Information about user $2 will be shown"
      USERNAME=$2
      shift
      shift
      ;;
    *)
      echo "Unknown option! Use $0 -h to display help"
      exit 1
      ;;

  esac

done


[[ -z $USERNAME ]] && echo "User parameter is mandatory - use option -n or display
help using -h" && exit 1

if [[ $RUN_ID == 'Y' ]]; then
  echo "id $USERNAME"
  id $USERNAME
fi

if [[ $RUN_LAST == 'Y' ]]; then
  echo "last $USERNAME"
  last $USERNAME
fi

if [[ $RUN_DU == 'Y' ]]; then
  echo "du -sh /home/$USERNAME"
  du -sh /home/$USERNAME
fi

exit 0
```

Odczyt argumentów za pomocą getopts

Notatka

- getopts to dedykowana instrukcja do przyjmowania parametrów w skryptach
- Instrukcja przyjmuje argument wskazujący na to, jakie opcje należy przetworzyć
    - Jeśli po literze opcji znajduje się dwukropek, to oznacza to, że dana opcja wymaga również wartości argumentu dla tej opcji
    - Dwukropek na początku oznacza, że getopts ma samodzielnie obsługiwać błędy
- Podczas analizy argumentów, litera odpowiedzialna za opcję jest dostępna w zmiennej opt
- Ewentualna wartość argumentu przypisana do danej opcji znajduje się w zmiennej OPTARG

```
while getopts ":rdhf:" opt; do
   case $opt in
      r)
         MAKE_REPORT='Y'
         ;;
      f)
         FILE=$OPTARG
         ;;
    esac
done
```

- Gdy korzystasz z getopts, masz do dyspozycji zmienną OPTIND wskazującą na numer kolejnego argumentu. Dlatego, jeśli chcesz przesunąć listę argumentów tak, aby pominąć argumenty już przeanalizowane przez getopts, to można dokonać przesunięcia o OPTIND-1:

```
shift $((OPTIND - 1))
```

- Jeśli komunikat generowany przez skrypt powinien zostać wysłany na standardowe wyjście błędów, to po echo należy dodać >&2

```
echo 'missing parameter' >&2
```

Laboratorium

1. Zmień kod z poprzedniego laboratorium tak, aby wykorzystywał getopts
    a. Dostosuj się do wszelkich ograniczeń, jakie wprowadza funkcja getopts w porównaniu do poprzedniego rozwiązania

Propozycja rozwiązania

```
#!/bin/bash


USERNAME=''
RUN_ID='N'
RUN_LAST='N'
RUN_DU='N'

while getopts ":ildhn:" OPT; do

  case $OPT in

    i)
      echo "id will be started"
      RUN_ID='Y'
      ;;
    l)
      echo "last will be started"
      RUN_LAST='Y'
      ;;
    d)
      echo "du will be started"
      RUN_DU='Y'
      ;;
    h)
      echo "USAGE:  $0 [-i] [-l] [-d] [-n <user_name>]"
      exit 0
      ;;
    n)
      echo "Information about user $OPTARG will be shown"
      USERNAME=$OPTARG
      ;;
    *)
      echo "Unknown option! Use $0 -h to display help"
      exit 1
      ;;

  esac

done


shift $((OPTIND - 1))


[[ -z $USERNAME ]] && echo "User parameter is mandatory - use option -n or display
help using -h" && exit 1

if [[ $RUN_ID == 'Y' ]]; then
  echo "id $USERNAME"
  id $USERNAME
fi

if [[ $RUN_LAST == 'Y' ]]; then
  echo "last $USERNAME"
  last $USERNAME
fi

if [[ $RUN_DU == 'Y' ]]; then
  echo "du -sh /home/$USERNAME"
  du -sh /home/$USERNAME
fi

exit 0
```

Wprowadzenie do funkcji

Notatka

- Funkcje pozwalają na wielokrotne wykorzystywanie tego samego kodu i dlatego są obecne chyba w każdym języku programowania. Funkcja to nazwany zestaw operacji do wykonania
- Funkcje mogą zwracać kod wyjścia przez słowo kluczowe return
- Funkcje mogą też zwracać tekst przechwytywany przez kod wywołujący funkcję.
- Składnia:

```
function  is_weekend()
{
    WEEKDAY=$(date +%w)
    if [[  $WEEKDAY -eq 0  ||  $WEEKDAY -eq 6 ]]; then
       echo “it is a weekend”
       return 0
    else
        echo “it is a working day”
        return 1
    fi
}
```

- Wywołanie funkcji:
    - W celu podjęcia decyzji w skrypcie zewnętrznym:

```
if is_weekend; then
     echo ”Hey it it a weekend”
fi
```

    - W celu pobrania zwracanej wartości

```
MESSAGE=$(is_weekend)
echo $MESSAGE
```

Laboratorium

1. Przygotowujesz bibliotekę skryptów, która ma być wykorzystywana do automatyzacji systemu. Musisz napisać funkcję, która zwróci 0, gdy katalog /tmp/logs zajmuje mniej niż 1 MB lub 1 gdy ten katalog jest większy:
    a. Utwórz katalog /tmp/logs i wgraj do niego jakiekolwiek pliki
    b. Napisz i przetestuj funkcję is_small, która zwróci 0 lub 1 w zależności od rozmiaru katalogu
2. Zmień funkcję tak, aby dodatkowo wyświetlała informację o wielkości katalogu. Zmień wywołanie funkcji tak, aby generowany przez funkcję tekst został zapisany w zmiennej tekstowej. Wyświetl zawartość tej zmiennej.
3. Zachowaj funkcję – przyda się w kolejnym laboratorium

Propozycja rozwiązania

```
#!/bin/bash

function is_small()
{
  DIR=/tmp/logs
  SIZE_KB=$(du -s $DIR | cut -f 1)

  if [[ $SIZE_KB -lt 1024 ]]; then
    echo "Directory $DIR is still small $SIZE_KB KB"
    return 0
  else
    echo "Directory $DIR is big ($SIZE_KB KB)"
    return 1
  fi
}


MESSAGE=$(is_small)
IS_SMALL=$?

echo $IS_SMALL

if [[ $IS_SMALL -eq 0 ]]; then
  echo "No action needed: $MESSAGE"
else
  echo "ACTION NEEDED: $MESSAGE"
fi

exit 0
```

Parametry funkcji

Notatka

- Parametry funkcji działają podobnie jak parametry skryptu:
    - Dostęp do parametrów uzyskuje się przez zmienne $1, $2, $3 itd.
    - Wywołując funkcję z parametrami wystarczy wymienić te parametry po nazwie funkcji
- Jeśli funkcja definiuje nowe zmienne, lub zmienia wartość istniejących zmiennych, to zmiany te są widoczne również na zewnątrz tej funkcji w skrypcie
- Jeśli zmienna ma być wewnętrzną zmienną funkcji (niewidoczną na zewnątrz tej funkcji), to należy ją zadeklarować poprzedzając jej nazwę słowem local:

```
function user_dir()
{
   local USER=$1
   local homedir=$(grep $USER /etc/passwd | cut -d : -f 6)
   echo $homedir
}

user_dir   lucy
```

Laboratorium

1. Zmień definicję funkcji z poprzedniego laboratorium tak, aby:
    a. Ścieżka sprawdzanego katalogu mogła być przekazywana przez parametr ($1)
    b. Graniczna wielkość katalogu mogła być również przekazywana jako parametr ($2)
    c. Zadbaj o to, aby wszystkie zmienne tworzone wewnątrz funkcji były widoczne tylko lokalnie (wewnątrz tej funkcji)

Propozycja rozwiązania

```
#!/bin/bash

function is_small()
{
  local DIR=$1
  local MAX_SIZE=$2
  SIZE_KB=$(du -s $DIR | cut -f 1)

  if [[ $SIZE_KB -lt $MAX_SIZE ]]; then
    echo "Directory $DIR is still small $SIZE_KB KB"
    return 0
  else
    echo "Directory $DIR is big ($SIZE_KB KB)"
    return 1
  fi
}


MESSAGE=$(is_small '/tmp/logs1' 2048)
IS_SMALL=$?

echo $IS_SMALL

if [[ $IS_SMALL -eq 0 ]]; then
  echo "No action needed: $MESSAGE"
else
  echo "ACTION NEEDED: $MESSAGE"
fi

exit 0
```

Korzystanie z funkcji (rekurencja)

Notatka

- Funkcje programowane w BASH, generalnie mogą działać tak, jak funkcje tworzone w innych językach programowania
    - Mogą wywoływać inne funkcje
    - Mogą wywoływać same siebie (rekurencja)
    - Mogą przyjmować parametry i zwracać wartości
    - Mogą odpowiadać z interaktywną komunikację z użytkownikiem, ale mogą również służyć tylko do wykonania czynności wsadowych (bez interakcji)
    - Mogą obudowywać złożony ciąg instrukcji, który w przypadku braku możliwości definiowania funkcji, trzeba by wpisywać wielokrotnie ręcznie
    - Mogą wykonywać obliczenia, przetwarzać napisy lub wykonywać operacje na systemie operacyjnym

Laboratorium

1. W tym laboratorium przygotujesz funkcję odpowiedzialną za interakcję z użytkownikiem. Zadanie funkcji read_yes_no() to:
    a. Wyświetlenie ma ekranie przesłanego jako parametr pytania
    b. Wyświetlenie zachęty do wprowadzenia odpowiedzi YES lub NO
    c. Pobranie interaktywnej odpowiedzi użytkownika
    d. Sprawdzenie czy odpowiedź to”
        i. YES lub yes lub Y lub y – jeśli tak, to funkcja ma zwrócić 0 i zakończyć się
        ii. NO lub no lub N lub n – jeśli tak, to funkcja ma zwrócić 1 i zakończyć się
        iii. W przeciwnym razie należy ponownie wyświetlić pytanie i wczytać odpowiedź – i tak do skutku, aż użytkownik wprowadzi jedną z akceptowanych odpowiedzi
2. Przetestuj funkcję wywołując ją w następujący sposób:

```
if (read_yes_no 'Should I continue with disk formatting?'); then
  echo "CONTINUING"
else
  echo "STOPPING"
fi
```

3. Wskazówki
    a. Gdy pętla ma się wykonywać w nieskończoność, to jej warunek może wyglądać tak:

```
while true; do
….
done
```

    b. Porównując napisy w składni z podwójnym nawiasem kwadratowym używaj podwójnego znaku równości. Gdy chcesz zbudować alternatywę używaj ||
    c. Do wyświetlenia pytania zapisanego w zmiennej QUESTION i wczytania odpowiedzi od użytkownika i zachowania jej w zmiennej **resoponse** użyj polecenia

```
read -p $QUESTION response
```

Propozycja rozwiązania

```
#!/bin/bash

function read_yes_no()
{
  local QUESTION=$1

  while true; do

    echo $QUESTION
    local response=''
    read -p 'Enter YES or NO: ' response

    if [[ $response == 'YES' || $response == 'Y' || $response == 'yes' || $response ==
'y' ]]; then
      return 0
    elif [[ $response == 'NO' || $response == 'N' || $response == 'no' || $response ==
'n' ]]; then
      return 1
    fi

  done
}

if (read_yes_no 'Should I continue with disk formatting?'); then
  echo "CONTINUING"
else
  echo "STOPPING"
fi

exit 0
```

Biblioteki własnych funkcji

Notatka

- Biblioteki funkcji pozwalają na wielokrotne wykorzystywanie tych samych funkcji w różnych skryptach
- Jest to plik, który zawiera w sobie definicje funkcji (bez ich wywołań i bez końcowego exit). Ponadto warto opisać plik, umieszczając w nagłówku kilka informacji:
    - Rodzaj umieszczonych tu funkcji
    - Informacje o autorze
    - Wersję, itp.
- Dodatkowo każda z funkcji również powinna być opisana. Dobrym zwyczajem jest np. umieszczenie informacji o przykładowym wywołaniu takiej funkcji
- Aby skorzystać z biblioteki funkcji, można
    - Wczytać plik do wykonywanego skryptu instrukcją source
    - Lub skorzystać z kropki

```
. ./fun_lib.sh
```

- W podobny sposób można wczytać bibliotekę do bieżącej sesji. W takim przypadku wszystkie zdefiniowane w bibliotece funkcje mogą być wykorzystywane jakby były komendami

Laboratorium

1. Utwórz plik biblioteki, opisz go jako „course library” w wersji „1.0” i podaj informacje o autorze, okolicznościach powstania biblioteki itp.
2. Dodaj do tego pliku utworzone we wcześniejszych laboratoriach funkcje. Każdą z tych funkcji opisz komentarzem. Dodaj informacje o przykładowym uruchomieniu danej funkcji.
3. Napisz skrypt, który:
    a. W pętli
        i. Poprosi o wprowadzenie nazwy użytkownika (polecenie read)
        ii. Pobierze ścieżkę do katalogu domowego tego użytkownika (funkcja user_dir)
        iii. Oceni czy ten katalog jest duży czy mały (funkcja is_small)
        iv. Zapyta, czy kontynuować. Jeśli tak, to pętla ma się powtórzyć kolejny raz, a jeśli nie, to pętla i skrypt mają się zakończyć (funkcja yes_no)

Propozycja rozwiązania

```
## FILE lib.sh




#!/bin/bash

# Title       : Course Library
# Version     : 1.0
# Author      : Mobilo24.eu
# Description : This is a final script from course "Scripting with BASH"


# Function finds home dir of a user sent as a parameter
# Returns: home directory path of a user
#
#  ----- Example 1  ------
#  user_dir lucy

function user_dir()
{
   local USER=$1
   local homedir=$(grep $USER /etc/passwd | cut -d : -f 6)
   echo $homedir
}


# Function checks if the directory is "small" or not
# Returns: 0 if the dir is small and 1 in other case
#
#  ----- Example 1  ------
#  MESSAGE=$(is_small '/tmp/logs1' 2048)
#  IS_SMALL=$?

function is_small()
{
  local DIR=$1
  local MAX_SIZE=$2
  SIZE_KB=$(du -s $DIR | cut -f 1)

  if [[ $SIZE_KB -lt $MAX_SIZE ]]; then
    echo "Directory $DIR is still small $SIZE_KB KB"
    return 0
  else
    echo "Directory $DIR is big ($SIZE_KB KB)"
    return 1
  fi
}


# Function reads answer yes/no or y/n or YES/NO or Y/N
# Returs: 0 for "yes" and 1 for "no"
#
#  ----- Example 1  ------
#  if (read_yes_no 'Should I continue with disk formatting?'); then
#     echo "CONTINUING"
#  else
#     echo "STOPPING"
#  fi

function read_yes_no()
{
  local QUESTION=$1

  while true; do
```

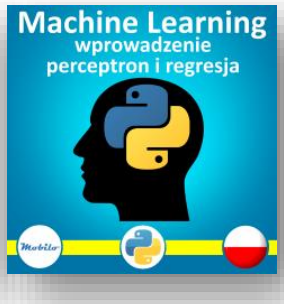
```
    echo $QUESTION
    local response=''
    read -p 'Enter YES or NO: ' response

    if [[ $response == 'YES' || $response == 'Y' || $response == 'yes' || $response ==
'y' ]]; then
      return 0
    elif [[ $response == 'NO' || $response == 'N' || $response == 'no' || $response ==
'n' ]]; then
      return 1
    fi

  done
}







## FILE my_script.sh




#!/bin/bash

. ./lib.sh


while $true; do

  NAME=''
  read -p "Enter name of the user that's home dir should be checked " NAME

  HOMEDIR=$(user_dir $NAME)
  echo "Home dir of user $NAME is $HOMEDIR"

  if (is_small "$HOMEDIR" 1024); then
    echo "This directory is small"
  else
    echo "This directory can be checked - it is big"
  fi

  if ! read_yes_no "Continue for a next user? " ; then
    break
  fi

done

exit 0
```

Spróbuj też!


Python
Dla początkujących


Analiza danych
w Python i PANDAS


Python dla średnio
zaawansowanych


Machine Learning
wprowadzenie
perceptron i regresja


Python Flask
Aplikacje webowe


Linux
dla początkujących


Administrator Linux
Instalacja i konfiguracja


Administrator Linux
Skryptowanie w BASH


PostgreSQL
dla początkujących


Microsoft Azure
AZ-900


Azure Administrator
Storage (AZ-104)


PowerShell
dla administratora Windows


SQL Server #1
Instalacja i narzędzia
Microsoft
SQL Server


SQL Server #2
Zapytania SQL
Microsoft
SQL Server


SQL Server #3
Programowanie
typy zaawansowane, XML
Microsoft
SQL Server


SQL Server #4
Modyfikacja danych
transakcje, obsługa błędów
Microsoft
SQL Server


SAP ASE 16 / Sybase
Installation and Configuration


MS Excel Tips & Tricks


Język R
dla początkujących


Reporting Services
Tworzenie raportów
Microsoft
SQL Server