Technologie Informacyjne - Linux 3

Instytut Matematyki
Uniwersytet Gdański

Tryby plików i uprawnienia

Każdy z plików uniksowych posiada zbiór **uprawnień** określających, czy możemy dany plik **odczytać (r), zapisać (w)** albo **uruchomić (x)**.

Wywołanie polecenia **ls -l** powoduje wyświetlenie wszystkich uprawnień.

Pierwsza kolumna określa **typ pliku**, każda kolejna trójka kolumn oznacza odpowiednio **uprawnienia użytkownika (u), grupy (g) i pozostałych (o)**. Kreska (-) oznacza, że jest to zwykły plik.

Tryby plików i uprawnienia

Do zmiany uprawnień plików służy polecenie **chmod**. Najpiew wybieramy zbiór uprawnień, który będziemy zmieniać, a następnie podajemy bit uprawnień.

Przykładowo jeśli chcemy nadać grupie (g) i pozostałym użytkownikom (o - czyli other) uprawnienia do odczytu (r) pliku o nazwie plik, należy wydać polececia:
chmod g+r plik
chmod o+r plik
albo też używając jednego polecenia: **chmod go+r plik**.
Uprawnienia te możemy usunąć poleceniem: **chmod go-r plik**.

Zadanie 1 Utworzyć plik o nazwie uprawnienia. Sprawdzić jakie on ma uprawnienia. Ustawić, aby użytkownik i grupa miała wszystkie możliwe uprawnienia natomiast pozostali mają nie mieć żadnych uprawnień do tego pliku.

Podstawy skryptów powłoki

Skrypt powłoki jest sekwencją poleceń zapisanych w pliku, z którego są odczytywane przez powłokę tak, jak byśmy wpisywali je w terminalu.

Wszystkie skrypty powłoki Bourne'a powinny zaczynać się od wiersza:
#!/bin/sh

Uwaga: Na początku pliku nie mogą pojawić się żadne puste miejsca. Wiersz ten oznacza, że wszystkie polecenia z tego pliku powinny być wykonane przez program /bin/sh.

Podstawy skryptów powłoki

Przykładowy skrypt powłoki:

```
#!/bin/sh
# Najpierw wpisuje jakiś tekst, a potem uruchamia polecenie ls.
echo Mam zamiar uruchomić polecenie ls.
ls
```

Uwaga: Znak krzyżyka (#) na początku wiersza oznacza, że ten wiersz jest komentarzem.

Po przygotowaniu skryptu i nadaniu mu odpowiednich uprawnień możemy go uruchomić, umieszczając plikt skryptu w jednym z katalogów wymienionych w ścieżce wyszukiwania i wywołując nazwę skryptu w wierszu poleceń: ./skrypt. Można też użyć pełnej nazwy ścieżki.
W przeciwieństwie do innych programów w systemie Unix skryptom powłoki nie wystarczy ustawić bitu wykonania, ale w celu umożliwienia odczytywania pliku przez powłokę konieczne jest także ustawienie bitu odczytu. Najłatwiej można to zrobić wywołując polecenie: **chmod +x skrypt**.
Wywołanie w ten sposób polecenie chmod pozwoli innym użytkownikom odczytywać i uruchamiać skrypt.

Zadanie 2 Zapisz w pliku skrypt01 i uruchom powyższy skrypt.

Cudzysłowy i literały

Zadanie 03 Uruchom każde z następujących trzech poleceń: echo $100, echo "$100", echo ’$100’.

Literał to łańcuch, który ma zostać przekazany przez powłokę do wiersza poleceń w niezmienionej postaci.

Oprócz znaku $, inne podobne sytuacje jak w powyższym przykładzie mają miejsce, gdy chcemy aby zamiast rozwijana przez powłokę znak * został przekazany do polecenia (np.), a także wtedy, gdy konieczne jest użycie średnika (;) w poleceniu

Cudzysłowy i literały

Gdy używamy **pojedycze cudzysłowy** powłoka nie dokona żadnych operacji zastąpienia.

Gdy używamy **podwójne cudzysłowy** otrzymamy podobny efekt jak przy pojedynczych cudzysłowach, tylko zostaną rozwinięte zmienne.

Zadanie 4 Sprawdź działanie poleceń:
echo ”Nie ma * w mojej ścieżce: $PATH”
echo ’Nie ma * w mojej ścieżce: $PATH’

Zmienne

W każdym języku programowania potrzebne są **zmienne**, aby przy ich pomocy wykonać pewne operacje.
Definicja zmiennych w skryptach basha:

ZMIENNA=witam jeżeli w wartości zmiennej nie ma spacji nie trzeba używać znaku cudzysłowie

ZMIENNA=12 jeżeli przypisujemy liczby lub cyfry nie używamy znaku cudzysłowie

ZMIENNA="witam was" jeżeli w wartości zmiennej znajduje się znak spacji, to wartość tą trzeba umieścić w cudzysłowie.

Uwaga: Po obu stronach znaku = nie mogą znajdować się spacje. Zwyczajowo nazwy zmiennych w bashu pisze się dużymi literami (jeśli nazwy zmiennych napiszemy małymi literami to też program będzie działał).

Zmienne

Odwołanie się do zmiennej odbywa się poprzez poprzedzenie nazwy zmiennem znakiem dolara **$ZMIENNA**.

Uwaga: BASH rozróżnia duże i małe litery, zatem odwoływanie się do zmiennej odbywa się przez użycie dokładnie takiej samej nazwy zmiennej przy pomocy jakiej ta zmienna została zadeklarowana.

Zadanie 5 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
ZMIENNA=”Mój drugi skrypt”
echo ”$ZMIENNA”
echo ’$ZMIENNA’
```

Zmienne

Jeśli chcemy wypisać wartość zmiennej a bezpośrednio za nią (bez znaku spacji) wyświetlić jakiś tekst należy ująć nazwę zmiennej w nawiasy klamrowe {}.

Zadanie 6 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
ZMIENNAN=nie
ZMIENNAT=””
echo ”${ZMIENNAN}dotrzymać”
echo ”${ZMIENNAT}dotrzymać”
```

Zmienne

Przy pomocy zmiennych możemy również wywoływać polecenia linux'a. Dzieje się tak dlatego, że shell rozwija zmienne.

Zadanie 7 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
Z1="ls"
Z2="-l"
$Z1 $Z2
```

Zmienne

Aby przekać zmiennej wartości, która została wprowadzone przez użytkownika przy pomocy klawiatury należy użyć polecenia **read**.

Zadanie 8 a)Określić wynik działania poniższego skryptu, a następnie sprawdzić to.
#!/bin/bash
echo ”Podaj swoje imie:”
read IMIE
echo ”Na imie masz: $IMIE”
b) Uzupełnij powyższy skrypt, aby program również pytał się o nazwisko i wiek. A następnie wyświetlał informację o imieniu, nazwisku i wieku.

Zmienne

Kiedy zachodzi potrzeba przeprowadzenia jakichś obliczeń można skorzystać z mechanizmu interpretacji wyrażeń arytmetycznych wbudowanego w Basha, obliczenia dokonywane są na liczbach całkowitych.

Składnia:

$((wyrażenie))

$[wyrażenie]

let wynik=wyrażenie
echo $wynik

Zadanie 9 Oblicz w bashu: $2+3$, $2-7$, $3*4$, $9/3$ i $9/4$. Wyjaśnij ostatni wynik.

Zadanie 10 a)Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
echo ”Podaj pierwsza liczbe:”
read L1
echo ”Podaj druga liczbe:”
read L2
SUMA=$(($L1 + $L2))
echo ”Wynik dodawania wynosi: $SUMA”
```

b) Przekształć powyższy skrypt używając polecenia: i) [], ii) let.

Zadanie 11 Napisać skrypt, który będzie się pytał o długość boku kwadratu a następnie obliczał obwód i pole tego kwadratu.

Parametry

Do wywoływanego skryptu można przekazywać parametry jego wywołania. Parametrów można używać dokładnie tak samo jak zmiennych jednakże nie można zmienić ich wartości.
Odwołanie się w treści skryptu do parametrów odbywa się w następujący sposób:

S* - wszystkie parametry począwszy od parametru numer 1,
$0 – nazwa programu,
$1 – parametr nr 1,
. . .
$9 – parametr nr 9,
$# - liczba parametrów,
$$ - numer procesu.

Parametry

Zadanie 12 Zapisać poniższy skrypt do pliku o nazwie skrypt1 i wywołać go następująco: ./skrypt1 123 434 2312

```
#!/bin/bash
echo ”nazwa skryptu: $0”
echo ”liczba parametrow: $#”
echo ”wszystkie parametry wywolana: $*”
echo ”parametr pierwszy: $1”
echo ”parametr drugi: $2”
echo ”parametr trzeci: $3”
echo ”numer procesu: $$”
```

Instrukcja case

Instrukcja case pozwala na dokonanie wyboru spośród kilku wzorców. Najpierw sprawdzana jest wartość zmiennej po słowie kluczowym case i porównywana ze wszystkimi wariantami po kolei. Jesli dopasowanie zakończy się sukcesem wykonane zostanie polecenie lub polecenia przypisane do danego wzorca. W przeciwnym wypadku użyte zostanie polecenie domyślne oznaczone symbolem gwiazdki: *). Instrukcja case kończy się zapisaniem tego słowa od końca (esac).

```
case $ZMIENNA in
   wzorzec1) polecenie1 ;;
   wzorzec2) polecenie2 ;;
   wzorzec3) polecenie3 ;;
   ...
   *) polecenie_domyślne
esac
```

Instrukcja case

Zadanie 13 Określić wynik działania poniższego skryptu, a następnie sprawdzić to. Sprawdzenia należy dokonać wywołując kilkakrotnie skrypt z różnymi parametrami np. ./nazwa_skryptu sobota.

```
#!/bin/bash
case $1 in
    piatek) echo ”weekend jest juz tuz tuz”;;
    sobota) echo ”juz weekend”;;
    niedziela) echo ”weekend się juz powoli konczy”;;
    poniedzialek) echo ”znowu trzeba isc do pracy”;;
    *) echo ”srodek tygodnia”;;
esac
```

Instrukcja case

Zadanie 14 Dopisać do poniższego skryptu instrukcję case, która jako wzorzec będzie miała cyfrę dnia tygodnia, a jako polecenie będzie wypisywała nazwę dnia tygodnia. Polecenie_domyślne ma uwzględnać przypadek, gdy zostanie wybrana spoza zakresu 1-7.

```
#!/bin/bash
echo ”Podaj cyfrę dnia tygodnia”
read d
```

Instrukcja if

Instrukcja warunkowa if sprawdza czy warunek jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub polecenia znajdujące się po słowie kluczowym **then**. Instrukcja kończy się słowem **fi**.

```
if warunek
then
   polecenie lub polecenia
fi
```

lub

```
if warunek; then
   polecenie lub polecenia
fi
```

Instrukcja if

W przypadku gdy należy wykonać inny zestaw poleceń, jeżeli zadany warunek kończy się wynikiem negatywnym, składnia polecenia wygląda następująco:

```
if warunek
then
   polecenie1
else
   polecenie2
fi
```

lub

```
if warunek; then
   polecenie1
else
   polecenie2
fi
```

Instrukcja if

Do testowania warunków używa się operatorów, które w wyniku swojego działania zwracają prawdę lub fałsz w zależności od tego czy sprawdzany warunek jest prawdziwy czy nieprawdziwy. Zazwyczaj warunek jest zapisywany w następującej postaci:
[wyrażenie1 operator wyrażenie2]
lub
[operator wyrażenie]

Między nawiasami a treścią warunku muszą być **spacje**, tak jak powyżej.
Ponadto bardzo istotne jest także używanie **znaków cudzysłowu** we wszystkich warunkach w których operator przyjmuje postać **–n**.

Instrukcja if

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
   echo ”Masz plik .bashrc”
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
   echo ”Masz plik .bashrc”
else
   echo ”Nie masz pliku .bashrc”
fi
```

Instrukcja if

A to kilka przykładów:

-e plik istnieje
= sprawdza czy wyrażenia są równe
!= sprawdza czy wyrażenia są różne
-n wyrażenie ma długość większą niż 0
-d wyrażenie istnieje i jest katalogiem
-lt mniejsze niż
-gt większe niż
-ge większe lub równe
-le mniejsze lub równe

Instrukcja if

Zadanie 15 Napisać instrukcję warunkową, która będzie sprawdzała, czy w zmiennej $1 znajduje się słowo witaj.

Zadanie 16 Napisać skrypt, który będzie prosił o podanie dwóch liczb. A następnie sprawdzał, czy pierwsza z wprowadzonych liczb jest większa od drugiej liczby.

Pętla for

Pętle stosuje się w celu wykonania pewnych instrukcji dla kolejnych iteracji lub kilku parametrów.

Pętla for wykonuje polecenia zawarte wewnątrz pętli, na każdym składniku listy (iteracja).

```
for zmienna in lista
do
   polecenie
done
```

Zadanie 17 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
for x in jeden dwa trzy
do
   echo ”To jest $x”
done
```

Pętla for

Zmiennej x przypisana jest lista, która składa się z trzech elementów: jeden, dwa, trzy. Wartością zmiennej x staje się po kolei każdy element listy, na wszystkich wykonywane jest polecenie: echo ”To jest $x”.

Pętla for jest bardzo przydatna w sytuacjach, gdy chcemy wykonać jakąś operację na wszystkich plikach w danym katalogu. Na przykład chcemy uzyskać listę wszystkich plików o danym rozszerzeniu znajdujących się w jakimś katalogu.

Zadanie 18 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
for x in *html
do
   echo ”To jest plik $x”
done
```

Pętla for

Zadanie 19 Uzupełnij skrypt tak, aby zmieniał nazwy plików pisane wielkimi literami na nazwy pisane małymi literami.

```
#!/bin/bash
.....
do
   mv $nazwa $(echo $nazwa | tr '[A-Z]' '[a-z]')
.....
```

Pętla while

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

while warunek
do
polecenie
done

Pętla while

Zadanie 20 Określić wynik działania poniższego skryptu, a następnie sprawdzić to.

```
#!/bin/bash
x=1;
while [ $x -le 10 ]
do
echo ”Napis pojawił się po raz: $x”
x=$[x + 1]
done
```

Uwaga x=$[x + 1] - zwiększa wartość zmiennej x o 1.

Pętla while