Operativni Sistemi

II Semestar - 2022/23 - Vježbe

Sedmica 9

Handout za Vježbe

Agenda:

- Shell Scripting
- Pisanje skripti
- Razumijevanje programiranja na nivou shella-a (ljuske)

Kontakt:

Narcisa.hadzajlic@size.ba (A Grupa)

Adin.jahic2019@size.ba (B Grupa)

Shell Scripting

Shell je interpreter komandnog jezika. On prevodi komande u UNIX-u razumljive naredbe. Postoje razni shell-ovi, a tri najčešće korištene su (originalna, poznata i kao Bourne shell), csh (Command SHell) i tcsh (poboljšana verzija csh). Ostali uključuju bash (Bourne-Again SHell), ksh (Korn SHell) i zsh. Obično podrazumevana ljuska (shell) (ono što se koristi sve dok ne izabere nešto drugačije) je csh.

"UNIX shell" koristimo da pokrenemo i zaustavimo procese, kontrolišemo terminal i na druge načine sarađujemo sa sistemom. **Shell scripting** je proces pružanja instrukcija shell-u u jednostavnom, interpretiranom programskom jeziku. Da bismo vidjeli kojim shell-om radimo, prvo se trebamo povezati pomoću SSH (Secure SHell protocol) na mašinu koju koristmo i ukucamo:

echo $SHELL ---- da vidimo koji shell radi u SSH-u.

Skripte u ovom handout-u su za shell csh. Jezik shell skriptovanja ne odgovara klasičnoj definiciji korisničkog jezika. Nema mnogih funkcija u raznim bibliotekama kao što su mogućnosti rekurzija, hashing ili sortiranja. Nema posebnih struktura podataka kao što su nizovi i hash tablice. Nema mogućnosti za direktni pristup hardveru ili dobre sigurnosne funkcije.

Ali na mnoge druge načine, jezik shell-a je veoma moćan - ima funkcije, uslove, petlje. Ne podržava posebne tipove podataka i potpuno je „ne-tipiziran“ (**sve je string**). Ali, prava snaga programa shell ne dolazi od samog jezika, već od raznovrsnosti biblioteka koje može pozvati za bilo koji program. Shell programiranje ostaje popularno jer pruža brz i jednostavan način za integraciju alata komandne linije i rješavanje složenih problema.

Jednostavne skripte

Najjednostavnije skripte su samo liste naredbi. Razmotrimo skriptu u nastavku.

Možete je upisati u datoteku nazvanu: first.sh

#!/bin/sh --Ova linija uvijek treba biti prva linija u vašoj skripti

#Jednostavna skripta

who am i

date

pwd

Općenito, sve u liniji nakon # je komentar te ignorirano od strane shell-a. "**#!/bin/sh**" govori ljusci da pozove **/bin/sh** kako bi pokrenula skriptu. To je potrebno jer različiti korisnici mogu koristiti različite ljuske: sh, csh, bash, zcsh, tcsh, itd. I ove ljuske imaju malo različite jezike i ugrađene funkcije. Kako bismo osigurali dosljedno djelovanje, želimo osigurati da se ista ljuska koristi za pokretanje skripte svaki put. To se postiže pokretanjem određene ljuske i slanjem skripte u njegov standardni ulaz.Za izvođenje shell skripte u csh, jednostavno koristimo sljedeću naredbu:

sh first.sh

Sada bi se trebalo prikazati izvođenje niza naredbi napisanih gore. Većina ove jednostavne skripte je lista naredbi. Ove naredbe se izvršavaju jedna za drugom i rezultati se prikazuju. Naredbe se nalaze pretraživanjem standardne putanje pretraživanja PATH.

PATH je lista direktorija odvojenih znakom ":" koja se treba pretraživati za izvršive datoteke. Možete naći svoju putanju tipkanjem:

echo $PATH

Varijable

PATH koja je spomenuta ranije je jedan primjer varijable. To je ono što se naziva „environment“ varijablom. Ona odražava jedan aspekt shell okruženja gdje tražiti izvršne datoteke. Promjena nje mijenja okruženje u kojem shell izvršava programe. Environment varijable su posebne zato što su definirane prije početka shell-a te se kao i većina drugih varijabli, mogu redefinirati jednostavno dodjeljivanjem nove vrijednosti:

```
echo $PATH
PATH=$PATH:/usr/local/apache/bin:.
echo $PATH
```

Da biste stvorili nove varijable, jednostavno im dodijelite vrijednost:

```
vrijednost ="mapa"
echo $vrijednost
```

Sve varijable shell skripte su netipizirane te način na koji će biti interpretirane zavisi o programu koji ih koristi ili koji operator ih manipulira ili pregledava. Shell skripte koriste naredbe za upravljanje datotečnim sustavom, ulazno/izlaznim operacijama i preusmjeravanjima datoteka među mnogim drugima.

Command Line Argumenti (CLA)

CLA su važan dio pisanja skripti. Definiraju očekivani ulaz u ljusku skripte. Primjerice, možda želimo proslijediti naziv datoteke ili naziv mape ili neku drugu vrstu argumenta ljusci skripte.

Postoji nekoliko posebnih varijabli koje pomažu u upravljanju argumentima naredbenog retka za skriptu:

- *$# - predstavlja ukupni broj argumenata (slično kao argv) - osim naredbe*
- *$0 - predstavlja naziv skripte, kako je pozvana*
- *$1, $2, $3, .., $8, $9 - prvih 9 argumenata naredbenog retka*
- *$* - svi argumenti naredbenog retka ILI*
- *$@ - svi argumenti naredbenog retka*

Vježba: Napiši jednostavan shell skript "myscript.sh" koji kao argument prima putanju direktorija te ispisuje popis svih datoteka i mapa unutar navedenog direktorija.

Navodnici

U shell skriptingu postoje tri različita načina navođenja navodnika, a svaki ima drugačije značenje:

- *nenavođene riječi se normalno interpretiraju*
- *"navođene riječi su uglavnom doslovne - ali $varijable se evaluiraju"*
- *'navođene riječi su apsolutno doslovno interpretirane'*
- *`naredbe u navodnicima poput ovih se izvršavaju, njihov se izlaz zatim umetne kao da je dodijeljen varijabli, a zatim se ta varijabla evaluira`*

Expr

Shell skripte nisu namijenjene za složene matematičke izraze. Međutim, program "expr" se može koristiti za manipulaciju varijablama, koje su normalno interpretirane kao stringovi, kao integeri. Razmotrimo sljedeći "adder" skript:

```
sum=expr $1 + $2
echo $sum
```

Predikati

Konvencija među UNIX programerima je da programi trebaju vratiti vrijednost 0 u slučaju uspjeha. Obično nenulta vrijednost ukazuje na to da program nije mogao učiniti ono što je zatraženo. Neki programeri vraćaju negativan broj u slučaju pogreške, poput nepostojanja datoteke, a pozitivan broj u slučaju nekog drugog terminalnog stanja, npr. stanje korisnika koji odabire prekid zahtjeva.

Kao rezultat toga, shell pojamovi *true* i *false* se razlikuju od onoga što bismo očekivali. 0 se smatra true, a nenulta vrijednost se smatra false.

Možemo koristiti test komandu za procjenu izraza. Sljedeći primjer će ispisati 0 ako je korisnik neko, a 1 u ostalim slučajevima. Ilustrira ne samo test već i upotrebu statusne varijable.

```
test "$LOGNAME" = neko
echo $?
```

Jezici shell skriptiranja su bez tipova. Prema zadanim postavkama, sve se interpretira kao string. Stoga, pri upotrebi varijabli, moramo navesti kako želimo da se interpretiraju. Dakle, operatori koje koristimo variraju ovisno o tome kako želimo interpretirati podatke.

Operators for strings, ints, and files						
string	x = y, comparison: equal	x != y, comparison: not equal	x, not null/not 0 length	-n x, is null		
ints	x -eq y, equal	x -ge y, greater or equal	x -le y, lesser or equal	x -gt y, strictly greater	x -lt y, strictly lesser	x -ne y, not equal
file	-f x, is a regular file	-d x, is a directory	-r x, is readable by this script	-w x, is writeable by this script	-x x, is executible by this script	
logical	x -a y, logical and, like && in C (0 is true, though)			x -o y, logical or, like && in C (0 is true, though)		

Selekcione strukture (IF,THEN,ELSE)

IF naredba sa else u shell skriptanju koristi se za kontrolu toka programa, gdje se neki blok koda treba izvršiti samo ako je neki uvjet ispunjen, a drugi blok koda ako uvjet nije ispunjen. Ova naredba koristi se za izvršavanje alternativnog bloka koda kada se zadovolji određeni uvjet, a drugi blok koda kada taj uvjet nije ispunjen.

Sintaksa IF naredbe sa else u shell skriptanju je sljedeća:

```
if [ uvjet ]
then
  # blok koda koji se izvršava ako je uvjet ispunjen
else
  # blok koda koji se izvršava ako uvjet nije ispunjen
fi
```

Primjer:

```
#!/bin/bash
# Skripta koja provjerava da li je broj pozitivan, negativan ili jednak nuli

echo "Unesite neki broj: "
read broj

if [ $broj -gt 0 ]
then
   echo "Uneseni broj je pozitivan"
elif [ $broj -lt 0 ]
then
   echo "Uneseni broj je negativan"
else
   echo "Uneseni broj je jednak nuli"
fi
```

Sekvencijalne strukture (petlje)

Petlje su bitan dio u shell skriptanju i omogućavaju nam izvršavanje jedne ili više naredbi više puta. Shell podržava dvije vrste petlji: for i while.

for petlja iterira kroz elemente nekog skupa podataka, dok **while petlja** izvršava naredbe sve dok je uvjet ispunjen.

Sintaksa:

1) for

```
for variable_name in list
do
   # komande za izvršavanje unutar petlje
done
```

2) **while**

```
while [ uvjet ]
do
   # komande za izvršavanje unutar petlje
done
```

Primjer:

```
for i in $(seq 1 10)
do
  echo "Broj: $i"
done
```

Primjer:

```
i=0
while [ $i -lt 10 ]
do
  echo "Broj: $i"
  i=$(($i+1))
done
```

Šta gore navedeni programi rade?

Vježba: koncepte iz predmeta UTP implementirati u shell-u

ZADACI ZA VJEŽBU:

1. Napravite skriptu koja generiše random broj između 1 i 10, a zatim traži od korisnika da pogodi taj broj?
2. Napišite skriptu koja provjerava da li su dva broja jednakih vrijednosti?
3. Napravite skriptu koja provjerava postoji li datoteka "example.txt" u trenutnom direktoriju i ispisuje poruku o uspjehu ili neuspjehu?
4. Napravite skriptu koja provjerava da li je datoteka veća od 1 MB, a ako jeste, pita korisnika želi li je izbrisati?
5. Napišite skriptu koja traži od korisnika da unese broj, a zatim ispisuje sve brojeve od tog broja do 1?

Za sve eventualne primjedbe, komentare, sugestije obratiti se na mail: narcisa.hadzajlic@dl.unze.ba; adin.jahic2019@size.ba