Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i računarske komunikacije

Upoznavanje sa BASH skriptama

Zašto? (1/2)

- Radno znanje Shell Scripting-a je od suštinskog značaja za svakoga ko želi da postane razumno vešt u administraciji sistema
- Čak i ako nemate potrebe za stvarnim pisanjem skripti, treba ih razumeti
- Kad se Linuks mašina podigne, izvršava se shell skripta u `/etc/rc.d` da povrati konfiguraciju sistema i uspostavi usluge

Zašto? (2/2)

- Detaljno razumevanje ovih početnih skripti je važno za analizu ponašanja sistema, kao i za modifikaciju istog
- Korisno je znanje i za kreiranje sopstvenih rešenja za prevođenje programskog koda

Odlike

- „Zanat skriptinga“ nije teško savladati
- skripta može biti izgrađena u delovima veličine sekcija
- postoji samo (relativno) mali skup Shell-specifičnih operatora i opcija da se nauče
- Sintaksa je jednostavna

I opet, zašto koristiti bash?

- Shell skript je brz i prljavi (quick and dirty) metod kreiranja prototipa složene aplikacije
- Dobijanje čak i ograničenog podskupa funkcionalnosti za rad u scenariju je često korisna prva faza u razvoju projekta
- Na ovaj način, struktura aplikacije može biti testirana i menjana
- Glavne zamke mogu biti pronađene pre nastavka kodiranja

Kada NE koristiti bash (1/2)

- Resource-intenzivni zadaci, naročito kada je brzina bitan faktor (sortiranje, heširanje, rekurzije)
- Postupci koji se odnose na teške uslove rada matematičkih operacija, posebno aritmetika u pokretnom zarezu, proizvoljne precizne kalkulacije, ili kompleksni brojevi
- Kada je potrebna cross-platform portabilnost
- Kompleksne aplikacije, gde je strukturirano programiranje neophodno (provera tipa promenljivih, funkcija prototipa, itd)

Kada NE koristiti bash (2/2)

- Kritične aplikacije od kojih zavisi budućnost (npr. kompanije)
- Situacije u kojima je važna bezbednost, gde bi trebalo da se garantuje integritet vašeg sistema i štiti od upada, pucanja i vandalizama
- Ekstenzivan rad sa datotekama
- Potrebna ugrađena podrška za višedimenzionalne nizove
- Treba da se generiše / manipuliše grafikom ili sa GUI

Šta je bash?

- BASH je akronim za „Bourne-Again shell“
- Bash je postao defakto standard za shell skripting na većini Uniks sistema
- Predstavljen će biti, ukratko, uvod u bash skripting

Primer 1

```
# Cleanup
# Pokreni kao root.
cd /var/log
cat /dev/null > messages
cat /dev/null > wtmp
echo "Log files cleaned up."
```

- Ništa neobično, naređane komande
- Mogu se pozivati jedna po jedna u konzoli
- Jedna od prednosti je što ne moramo ponovo da kucamo sve ove komande
- Skripta postaje program ili alat koji možemo po želji da modifikujemo i koristimo za razne aplikacije

Primer 2

- Ovo što sad imamo počinje da liči na skriptu, ali može još više da se unapredi

```
#!/bin/bash
# Pravilan header za Bash skriptu.
# Cleanup, verzija 2
# Pokreni kao root, naravno.
# Ubaciti kod ovde da završi izvršavanje ako nije root.
LOG_DIR=/var/log
# Varijable su bolje nego zakucane vrednosti.
cd $LOG_DIR
cat /dev/null > messages
cat /dev/null > wtmp
echo "Logs cleaned up."
exit # Pravilan našin izlaženja iz skripte.
# Čist "exit" (bez parametara) vraća izlazni status
#+ prethodno pozvane komande.
```

Primer 3

```
#!/bin/bash
# Cleanup, verzija 3
LOG_DIR=/var/log
ROOT_UID=0 # Samo korisnici sa $UID 0 imaju root prava.
LINES=50 # Broj linija koji se čuva.
E_XCD=86 # Ne može da se promeni direktorijum?
E_NOTROOT=87 # Ako nije root, exit greška.
# Pokreni kao root, naravno.
if [ "$UID" -ne "$ROOT_UID" ]
then
echo "Must be root to run this script."
exit $E_NOTROOT
fi
if [ -n "$1" ]
# Testiraj da li imamo prosleđen argument.
then
lines=$1
else
lines=$LINES # Po default-u, ako nije specifirano argumentom
Fi
cd $LOG_DIR
if [ `pwd` != "$LOG_DIR" ] # or if [ "$PWD" != "$LOG_DIR" ]
# Nema u /var/log?
then
echo "Can't change to $LOG_DIR."
exit $E_XCD
fi # Proveri da li si u pravom direktorijumu pre nego što se dira log fajl.
tail -n $lines messages > mesg.temp
mv mesg.temp messages
cat /dev/null > wtmp
echo "Log files cleaned up."
exit 0
```

Sha-bang (1/4)

- SHA-Bang (#!) na čelu skripte govori vašem sistemu da je fajl skup komandi koje se daju navedenom komandnom prevodiocu
- #! je zapravo dvobajtni magični broj, poseban marker koji označava tip datoteke
  - U ovom slučaju izvršnu shell skriptu

Sha-bang (2/4)

- Odmah nakon `#!` je putanja
- To je putanja za program koji tumači komande u tekstu, bilo da je shell, programski jezik, ili uslužni program
- Komandni prevodilac zatim izvršava komande u tekstu, sa početkom u vrhu (linija nakon Sha-bang linije) i ignoriše komentare

Sha-bang (3/4)

- Svaki od header-a koristi različit komandni interpreter

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/tcl
#!/bin/sed -f
#!/bin/awk -f
```

- Korišćenjem `#!/bin/sh` podrazumevani Bourne shell čini da će skripta biti portabilna na većini varijanti UNIX-a
- Čak može biti i portabilna na različitim ne-Linuks sistemima, ali zarad toga bismo morali žrtvovati bash-specifične alate

Sha-bang (4/4)

- Putanja koja se prosledi nakon `#!` mora biti tačna, u suprotnom, jedino što se dobije je "Command not found." kada se pokrene skripta
- `#!` može biti izostavljeno ukoliko skripta koristi samo set generičkih sistemskih komandi, bez internih shell direktiva
- `#!/bin/sh` poziva podrazumevani shell interpreter, koji je najčešće `/bin/bash` na Linuks mašini

Pokretanje skripte

- Skripta može biti pokrenuta na više načina:
  - `sh ime_skripte`
  - `bash ime_skripte`
  - Učiniti da skripta bude izvršiva `chmod` komandom i pokrenuti sa `./ime_skripte`

Rezime i šta dalje

- Na primerima smo videli upotrebu:
  - varijabli
  - parametara
  - grananja
  - Petlji
- Dalje ćemo se detaljnije dotaći tih tema

Varijable

- Jako je bitno raspoznavati ime varijable od njene vrednosti
- `variable1` je ime i dodeljena joj je vrednost `23`
- Kad hoćemo da ispišemo tu vrednost, moramo koristiti `$variable1`
- A kad ispisujemo bez `$`, što je znak interpreteru da nailazi na varijablu, ispisace običan string
- Dodela vrednosti mora biti bez razmaka

```
bash$ variable1=23

bash$ echo variable1
variable1

bash$ echo $variable1
23
```

Korišćenje navodnika sa varijablama

- Ubacivanje varijable u dvostruke navodnike (`"..."`) se ne meša sa izmenom varijable
- Ovo se zove delimično citiranje, ponekad nazivano i “slabo citiranje”
- Koristeći jednostruke navodnike (`'...'`) varijabla će se koristiti bukvalno, a ne zamena za njenu vrednost
- Ovo je puno citiranje, ponekad nazivano i "jako citiranje"

Varijable - napomene

- Napomena:
  - **`$variable`** je zapravo uprošćena forma od **`${variable}`**
  - U kontekstu gde je **`$variable`** sintaksa uzrok greške, duža forma bi mogla da radi
  - Neinicijalizovana varijabla ima vrednost `NULL` – nema dodeljenu vrednost (*NIJE* nula!)
  - Ne zameniti `=` i `-eq`, koji testira, a ne dodeljuje vrednost!
  - `=` može biti i dodela vrednost, a i korišćena za proveru, zavisno od konteksta upotrebe

Vrste varijabli

- Varijable mogu biti lokalne i globalne
- Lokalne
  - Žive su samo tokom izvršavanja skripte
- Globalne
  - Žive su i pre i posle izvršavanja skripte, ali u skripti može biti promenjena njena vrednost i promena ostaje trajna

Parametri (1/2)

- Argumenti prosleđeni skripti iz komandne linije: `$0`, `$1`, `$2`, `$3` . . .
- `$0` je ime same skripte
- `$1` je prvi argument, `$2` je drugi, `$3` je treći, i tako dalje. Nakon `$9` argumenti moraju biti zatvoreni u vitičaste zagrade, na primer: `${42}`, `${43}`, `${159}`

Parametri (2/2)

- Postoji i indirektna referenca na argumente
- `$#` - `#` označava broj prosleđenih argumenata
- Na ovaj način možemo saznati broj argumenata, odnosno da li to zadovoljava uslove naše skripte
- Napomena:
  - Na nekim sistemima je potrebno prvo proveriti `$0` argument za ime skripte pa je tek onda moguće pozvati `$#` za broj argumenata

```
args=$# # Number of args passed.
lastarg=${!args}
# Note: This is an *indirect reference* to $args ...
# Or: lastarg=${!#}
# This is an *indirect reference* to the $# variable.
# Note that lastarg=${!$#} doesn't work.
```

Petlje (1/4)

- `For` petlja:
  -
```
for arg in [list]
do
 command(s)...
done
```

- Ako je `do` u istoj liniji sa `for`, moramo posle liste da stavimo `;`
  - `for arg in [list] ; do`

Petlje (2/4)

- Primer `For` petlje:

```
#!/bin/bash
# Listing the planets.
for planet in Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto
do
echo $planet # Each planet on a separate line.
done
echo; echo
for planet in "Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto"
# All planets on same line.
# Entire 'list' enclosed in quotes creates a single variable.
# Why? Whitespace incorporated into the variable.
do
echo $planet
done
echo; echo "Whoops! Pluto is no longer a planet!"
exit 0
```

Petlje (3/4)

- `While` petlja:
  -
```
while [condition]
do
 command(s)...
done
```

- Važi isto pravilo za `do` u istoj liniji sa petljom:
  - `while [condition]; do`

Petlje (4/4)

- Primer `While` petlje:

```
#!/bin/bash
var0=0
LIMIT=10
while [ "$var0" -lt "$LIMIT" ]
#      ^                    ^
# Spaces, because these are "test-brackets" . . .
do
echo -n "$var0 "       # -n suppresses newline.
#             ^          Space, to separate printed out numbers.
var0=`expr $var0 + 1` # var0=$(($var0+1)) also works.
                      # var0=$((var0 + 1)) also works.
                      # let "var0 += 1" also works.
done                  # Various other methods also work.
echo
exit 0
```

Kontrola petlji

- `Break` i `continue` mogu da se koriste u bash petljama
- Njihova upotreba je skoro identična upotrebi u C-u
- `Break` može imati i parametar uz sebe da se naglasi da iz ugnježdenih petlji želimo da izađemo i iz više nivoa petlji
- `Continue` takođe može imati parametar, ali `continue` na taj način preskače `N` sledećih iteracija petlje

If - then - else

```
if [ condition ]
then
    commands
else
    commands
fi
```

```
#!/bin/bash
if [ $# -ne 5 ]
then
    return 0 # true
else
    return 1 # false
fi
```

Case (1/2)

```
case "$variable" in
    "$condition1" )
    command...
    ;;
    "$condition2" )
    command...
    ;;
esac
```

```
#!/bin/bash
# Testing ranges of characters.
echo; echo "Hit a key, then hit return."
read Keypress
case "$Keypress" in
    [[:lower:]] ) echo "Lowercase letter";;
    [[:upper:]] ) echo "Uppercase letter";;
    [0-9] ) echo "Digit";;
    * ) echo "Punctuation, whitespace, or other";;
esac
```

Case (2/2)

- Citiranje varijabli nije obavezno, jer se ne događa razdvajanje reči
- Svaki test linija završava desno zagradom - `)`
- Svaki blok uslov završava se dvostrukim tačka-zarezom - `;;`
- Ako je uslov istinit, onda se pridružene komande izvršavaju i `case` blok prestaje
- Čitav `case` blok završava sa `esac` (`case` napisano unazad)